HPI Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Architecting Self-Adaptive Critical Systems: Contradiction or Panacea?

Invited Talk at WADS 2009, 29.06.2009

Holger Giese
System Analysis & Modeling Group, Hasso Plattner Institute for Software Systems Engineering at the University of Potsdam, Germany

holger.giese@hpi.uni-potsdam.de

What are Critical Software Systems?

Characteristics:

- large-scale, heterogeneous, distributed

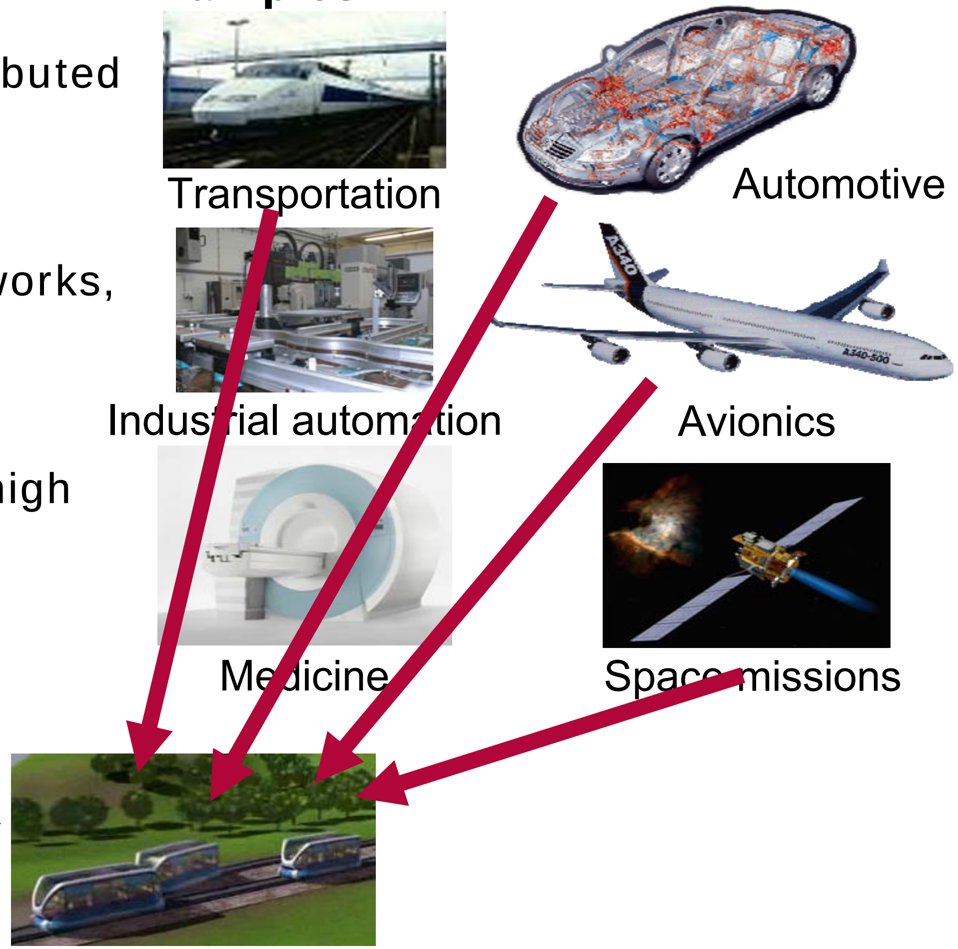
May include:

- Server backends, embedded subsystems, wireless ad hoc networks, mobile devices

Require:

- Safety, security, high reliability, high availability, ...

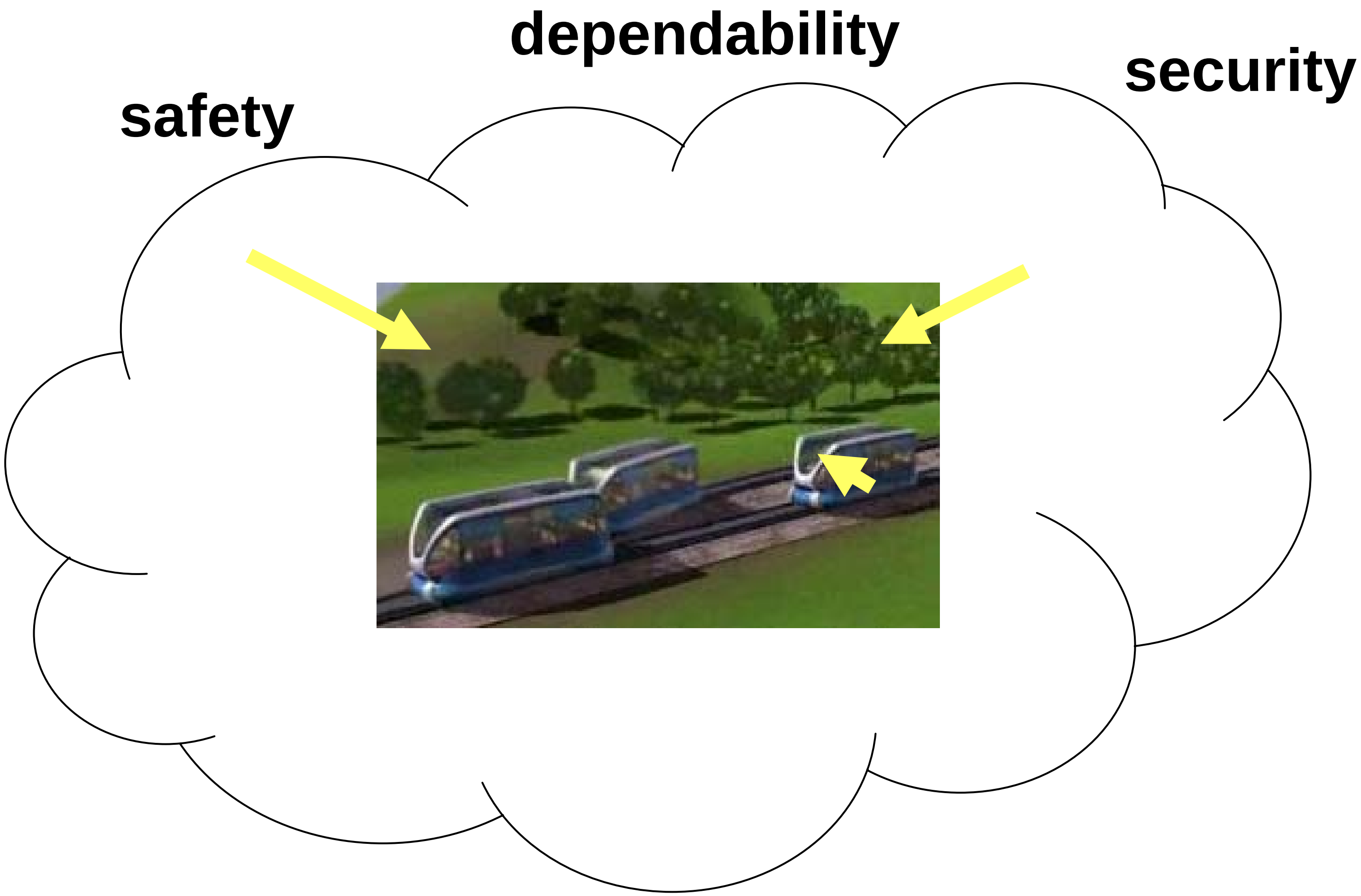
Examples:



Enterprise Critical Systems

Critical System of Systems: RailCab

Cirtical Software Systems - Threats



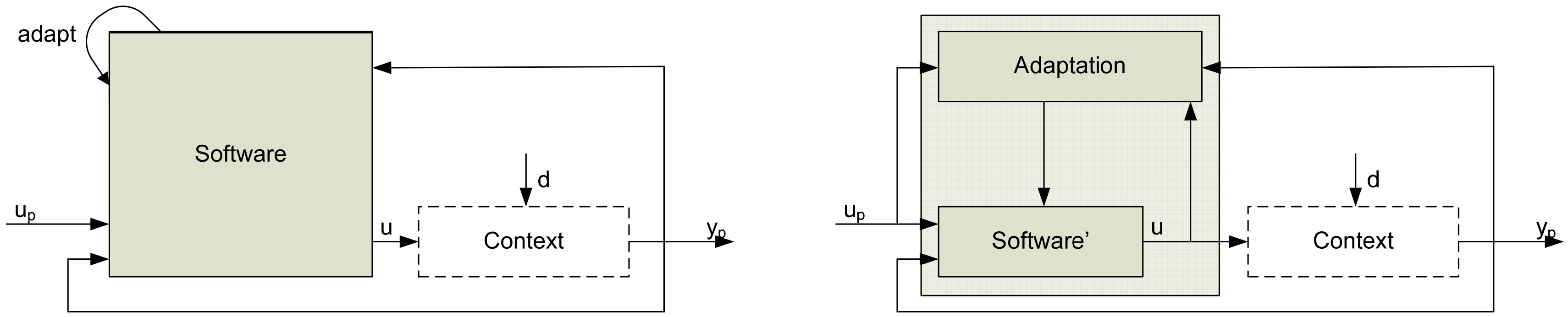
Typical threats: hardware failure, not fulfilled context assumption, misuse, attacks, …

Sources for threats: system hardware (incl. computer), environment, software, …

Self-Adaptive Critical Software Systems

4

Context = system hardware + environment



Adaptation to compensate threats (self-healing, self-configuring):

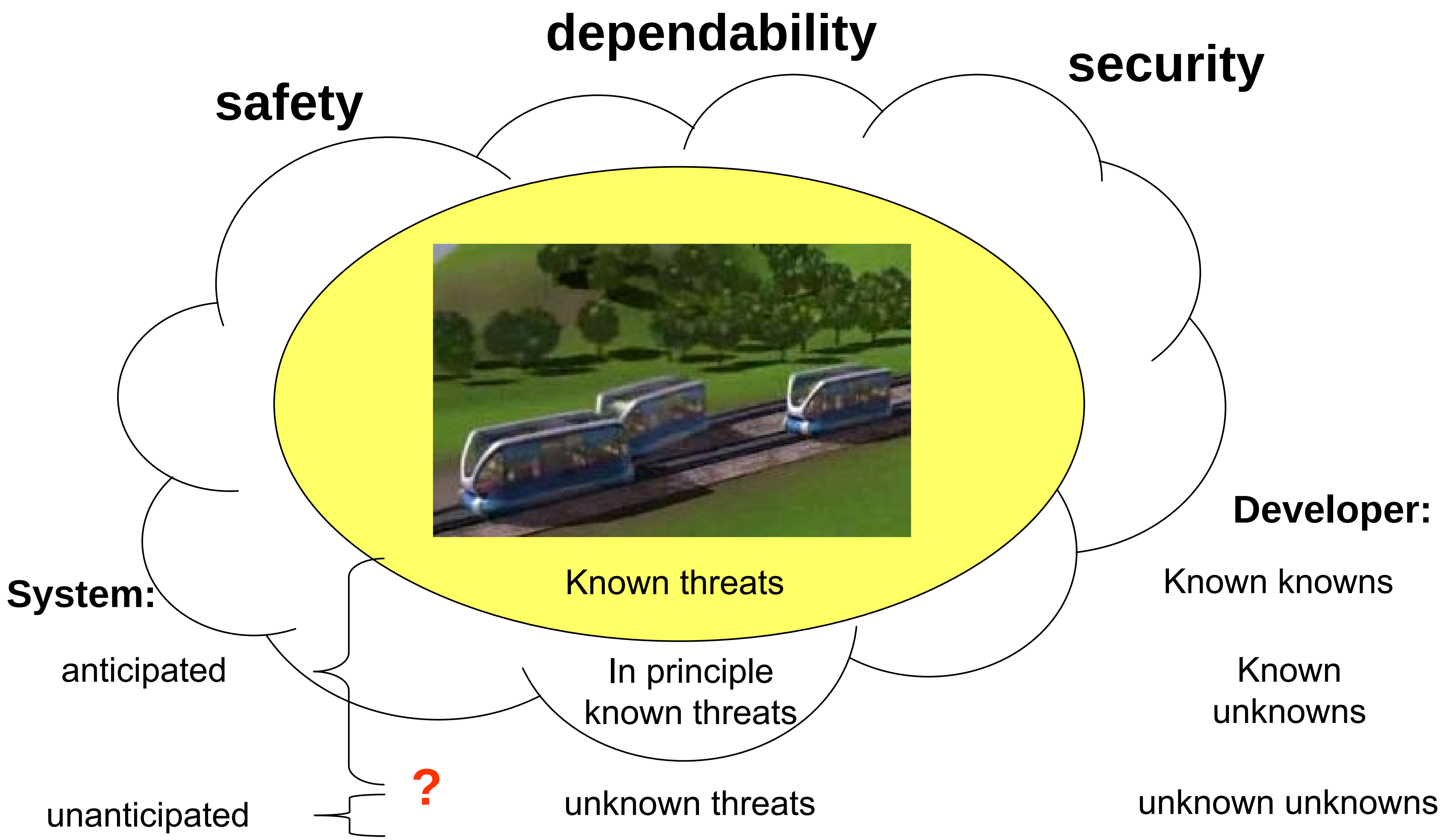
- **Absolute position:** adaptation must guarantee that **all** threats are properly handled (this CANNOT be achieved)
- **Relative position:** adaptation must guarantee that all relevant threats are properly handled (relevant = likelihood+ severity + ...; CAN ONLY be achieved in rare cases)

Problem: Usually not all threats are known!

- **Practice:** adaptation must guarantee that all known and relevant threats are properly handled (relevant = likelihood+ severity + ...)

Known and Unkown Threats

Hasso Plattner Institut



Self-Adaptive Critical Systems: Pros & Cons

Pros (cliché):

- Self-adaptive systems can handle unanticipated threats which classical system design do not cover

Cons (cliché):

- For self-adaptive systems no guarantees can be given as they can change their behavior

Resulting Open Questions (Contradiction or Panacea?):

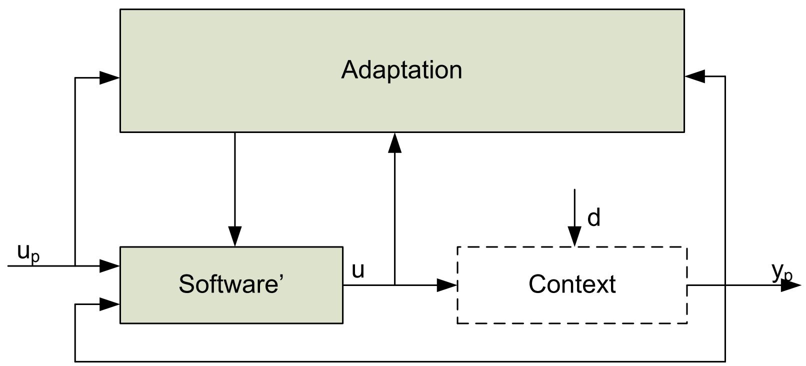
What kind of additional threats can self-adaptive systems cover?

Can we establish the required guarantees for self-adaptive systems?

Adaptation & Models

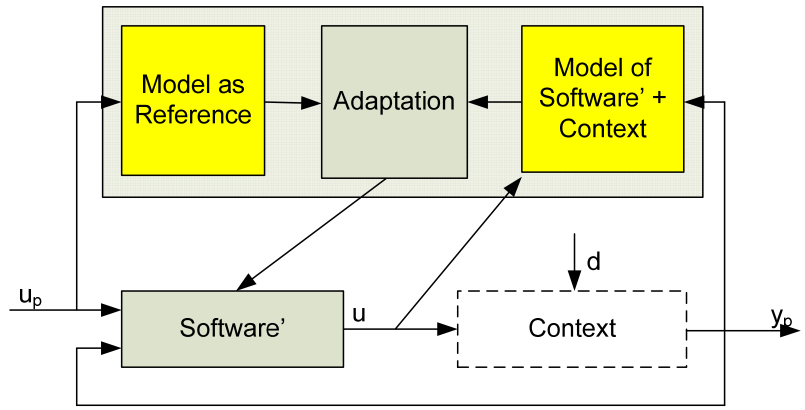
Adapt "without" models:

- Still implicit **design-time models** are used to establish guarantees offline
- **Limitation:** covers only threats included in **one** model of the software' + context (potentially including some parameters that can be observed)

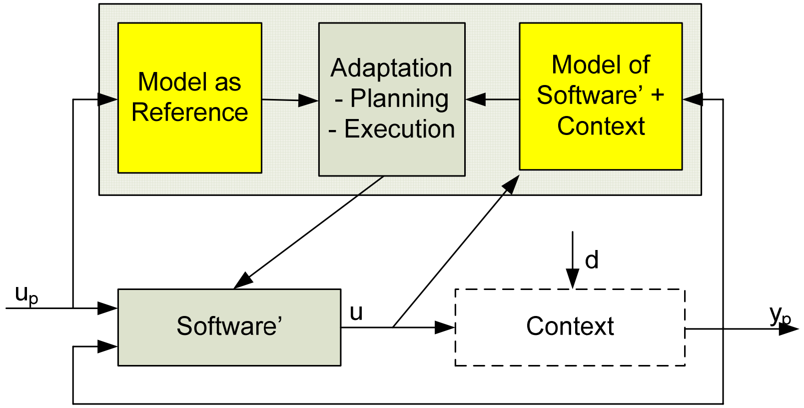


Adapt with runtime models:

- Explicit **runtime models** are used to establish guarantees online
- **Limitation:** covers only threats captured by the runtime models (**multiple**!); assume correct learning of them from the observations



Adaptation & Guarantees



Bottom line: Self-adaptive systems must simply be “better” and not “worse”

Cases that must be covered offline:

(1) Execution of the adaptation: consistent update; timing, …

Additional cases that must be covered offline for runtime models:

(2) Adapting the model of the software‘: consistent; fast enough; …

(3) Adapting the model of the context: consistent; fast enough; accurate enough, …

(4) Model as reference: correct reference, complete, …

Cases that must be covered offline and/or online:

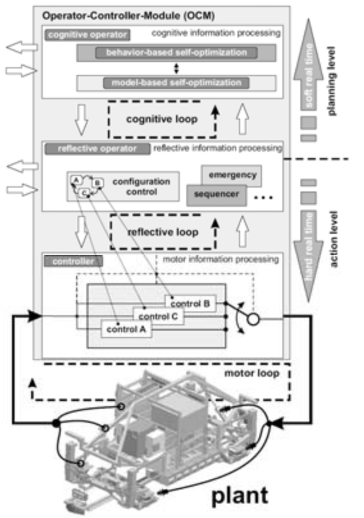
(5) Planning of the adaptation: does it really ensure the required guarantees?

Open Question: are the required guarantees possible/feasible? Some examples …

Example for (1) Execution of the adaptation

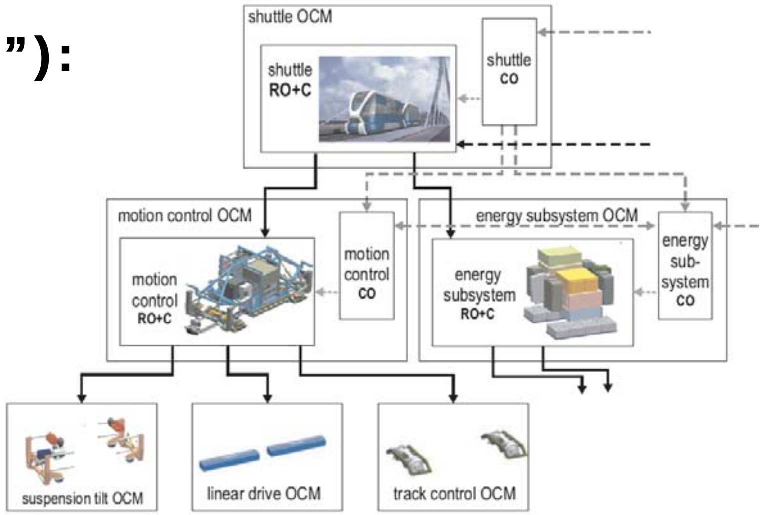
Operator-Controller Module (OCM) for self-optimizing mechatronic systems

- **Cognitive operator (CO):** decoupled from the hard real-time processing (**flexible**)
- **Reflective operator (RO):** Real-time coordination and reconfiguration (**pre-planned**)
- **Controller (C):** Control via sensors and actuators in hard real-time



Modular formal verification (“RO part”):

- Formal interface covers possible pre-planned configuration steps
- Consistent configuration across complex hierarchies: correct timing



Holger Giese, Sven Burmester, Wilhelm Schäfer, and Oliver Oberschelp, 'Modular Design and Verification of Component-Based Mechatronic Systems with Online-Reconfiguration', in *Proc. of 12th ACM SIGSOFT Foundations of Software Engineering 2004 (FSE 2004), Newport Beach, USA*, pp. 179--188, ACM Press, November 2004.

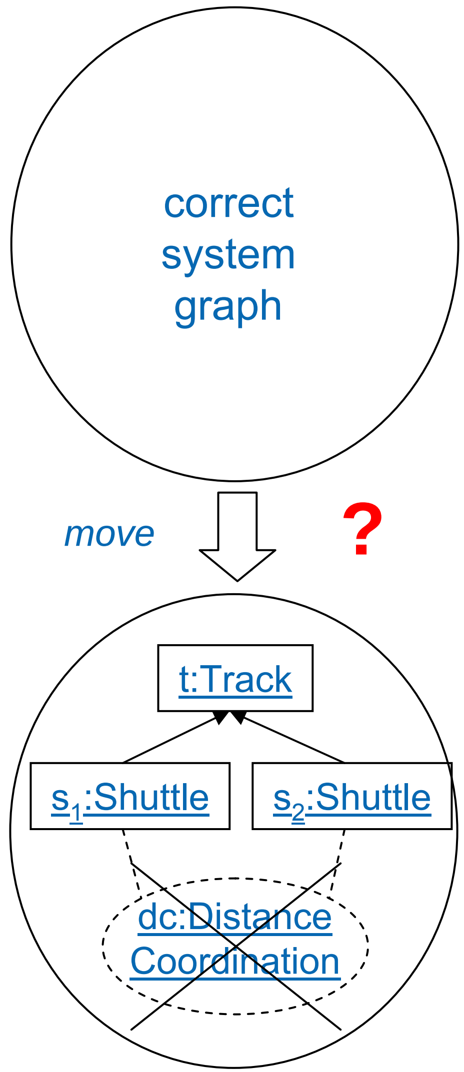
Example for (1) Execution of the adaptation (cont.)

Distributed Software Architecture + Context:

- Supports system with flexibly changing structure (real-time clocks, linear variables)
- Model all possible **structural changes** in the system and its environment in form of extended graphs and graph transformations

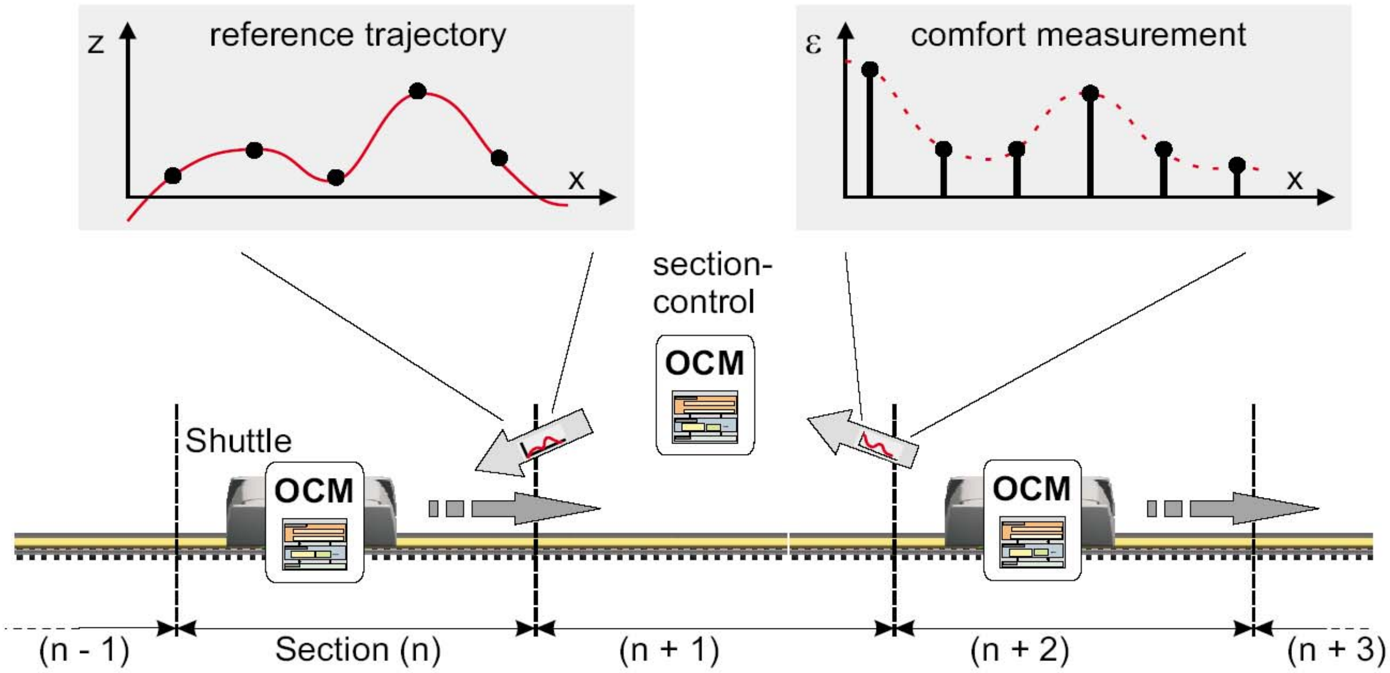
Verification:

- Analyze whether structural changes can lead from safe to unsafe situations (inductive **invariants**; incremental check for changed transformations)



Basil Becker and Dirk Beyer and Holger Giese and Florian Klein and Daniela Schilling, Symbolic Invariant Verification for Systems with Dynamic Structural Adaptation, Proc. of the 28th International Conference on Software Engineering (ICSE), Shanghai, China, vol. , 2006,

Example for (5) Planning of the adaptation

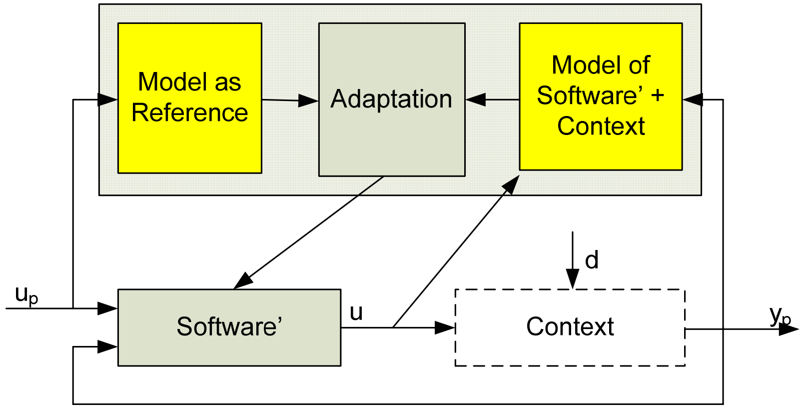


- Distributed learning of a model of the track (environment)
- Local learning of a model of the shuttle (system hardware)
- Planning an adaptation in form of an optimal trajectory
- Trajectory synthesis establishes required guarantees

Sven Burmester and Holger Giese and Eckehard Münch and Oliver Oberschelp and Florian Klein and Peter Scheideler,. *Tool Support for the Design of Self-Optimizing Mechatronic Multi-Agent Systems*, International Journal on Software Tools for Technology Transfer (STTT) **10** (3), 207-222, 2008.

Example 2 for (5) Planning of the adaptation

- Application: Monitoring and repair of complex architectures with redundancy (self-repair)
- Uses a model as reference and to capture the state of software' + context
- The model as reference is used to compute the required repair (computed solution ensures online the required guarantees)
- Trade-off: speed of repair vs. quality of structural adaptation



	complete		expansion		reduction	
	time	damage	time	damage	time	damage
1	13630	773	40	7	99.7%	99.1%
2	14890	97	20	30	98.7%	59%
3	13790	4	10	5	99.9%	-25%
4	13660	34	40	34	99.7%	0%

Matthias Tichy and Holger Giese and Daniela Schilling and Wladimir Pauls, Computing Optimal Self-Repair Actions: Damage Minimization versus Repair Time, Proc. of the ICSE 2005 Workshop on Architecting Dependable Systems, St. Louis, Missouri, USA, (Rog\'erio de Lemos and Alexander Romanovsky, ed.), vol. , ACM Press, 2005, p. 1–6,

Covers: arbitrary changes within the model of software' + context

Conclusions

Open Questions (Contradiction or Panacea?):

What kind of additional threats can self-adaptive systems cover?

- Self-adaptive systems allows in principle to cover more threats
  - Without runtime models coverage is restricted to what is covered by the design-time model
  - With runtime models coverage is restricted to what can be covered by the different possible forms of the runtime model

Can we establish the required guarantees for self-adaptive systems?

- Some guarantees for self-adaptive solutions can be established offline
  - (1) Execution of the adaptation
  - (2) Adapting the model of the software
  - (3) Adapting the model of the context
  - (4) Model as reference
- Some guarantees for self-adaptive solutions can be established online/offline
  - (5) Planning of the adaptation: does it really ensure the required guarantees?

Conclusions (Cont.)

- Self-adaptive solutions only help when
  - Adaptation itself is guaranteed to work,
  - Guarantees for the adaptation can be established (offline or online) or
  - When cases are covered that are otherwise not covered.
- Coverage not having a runtime model itself counts!

Critical Self-adaptive software systems are thus

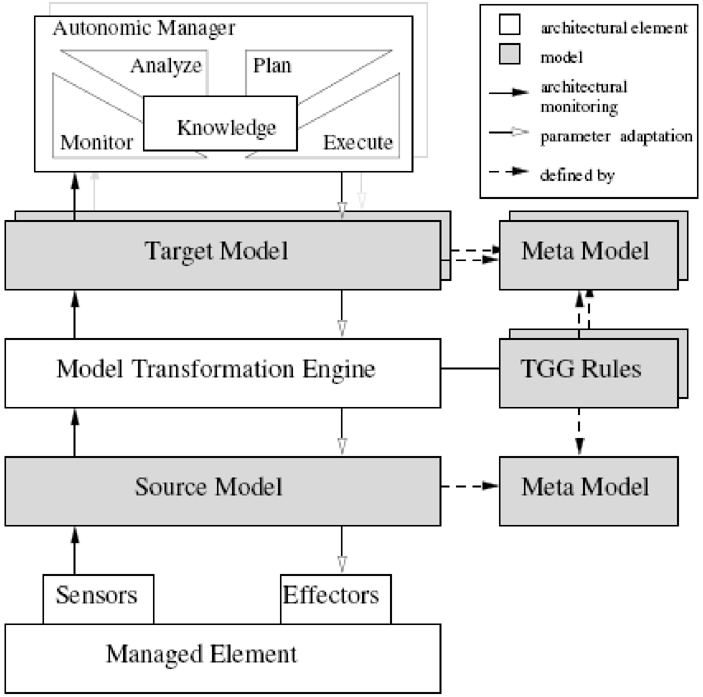
- No contradiction but also
- No panacea as building them requires a lot of effort

⇨ ease building self-adaptive systems is key

MDE for Runtime Models in Self-Adaptive Systems

Hasso Plattner Institut

- Supports feedback loop for models using "meta-models" and model transformation techniques for an EJB application server
- Extract abstract runtime models for different autonomic managers as requirec
- Synchronize runtime models **incrementally** between the autonomic manager and the managed element (faster as manual implementations)
- Adapt managed subsystem incrementally via model (just parameters yet)



Covers: arbitrary changes within the model of software' (not context)

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) accepted paper.