HPI Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Model-Driven Software Engineering of Self-Adaptive Systems

KTH, Stockholm, Sweden, 25.11.2010

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

Outline

I Motivation

– Why self-adaptiveness?

II Foundations

- What is self-adaptiveness?

III Construction

– How to build them?

IV Quality Assurance

– How to ensure their quality?

V Conclusion

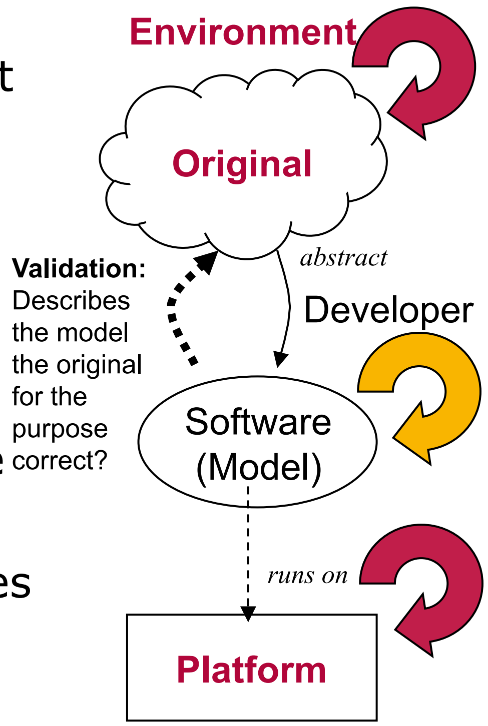
I Motivation: Software Evolution & Aging

Software Evolution [Lehman&Belady1985,Lehman1997]:

- Programs always include explicit and implicit assumptions about the real world domain
- The real world domain and the program (and its explicit and implicit assumptions) must be maintained compatible and valid with one another
- Developing software is a complex feedback system

Two types of **software aging** [Parnas1994]:

- **Lack of Movement:** Aging caused by the failure of the product's owners to modify it to meet changing needs.
- **Ignorant Surgery:** Aging caused as a result of changes that are made.
- This "one-two punch" can lead to rapid decline in the value of a software product.



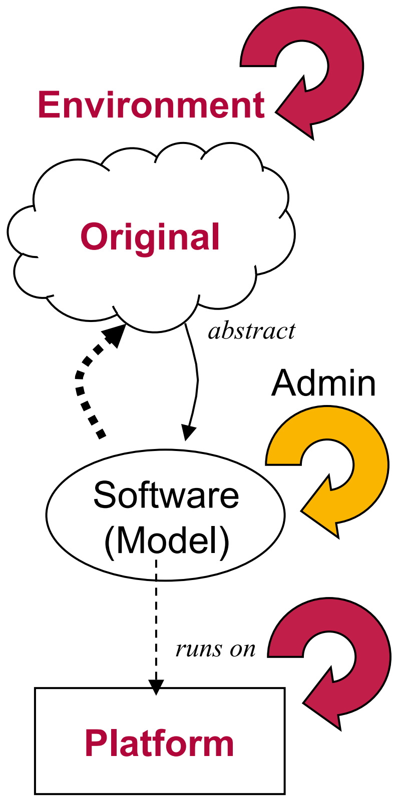
I Motivation: Software Complexity & Administration

Autonomic Computing [AC2001]:

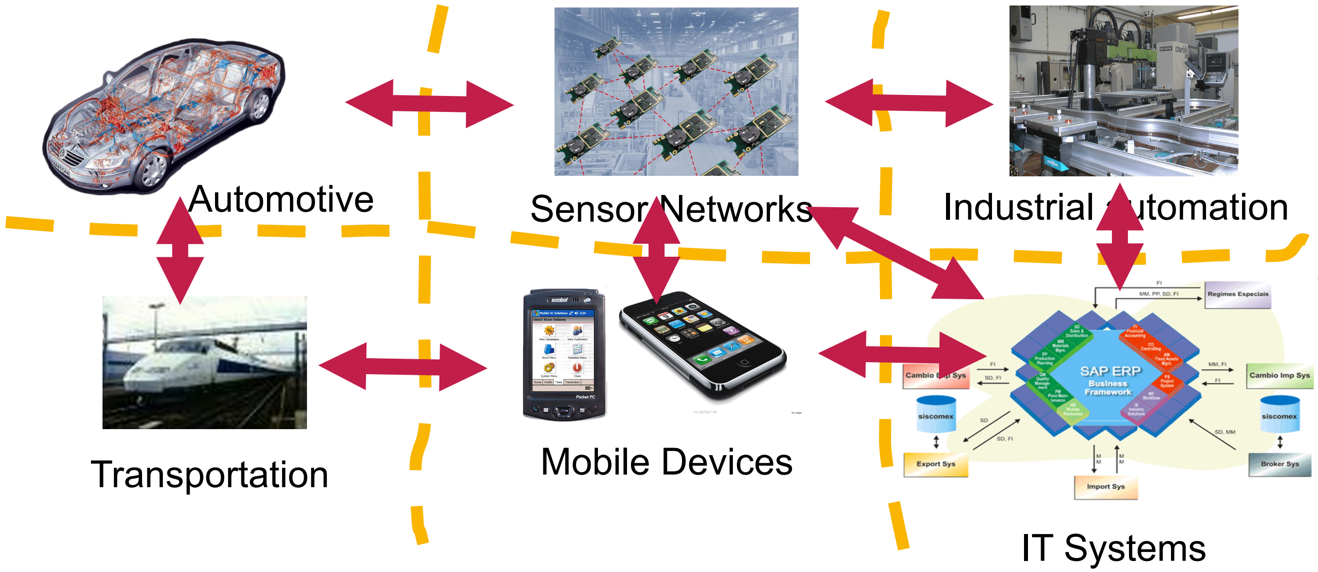
- Evolution via automation also produces **complexity** as an unavoidable byproduct (especially true for IT systems: incredible progress in speed, storage and communication; extreme growth software with >30 million loc and > 4,000 programmers)
- In fact, the growing complexity of the I/T infrastructure **threatens to undermine** the very benefits information technology aims to provide, because systems cannot be managed any more.

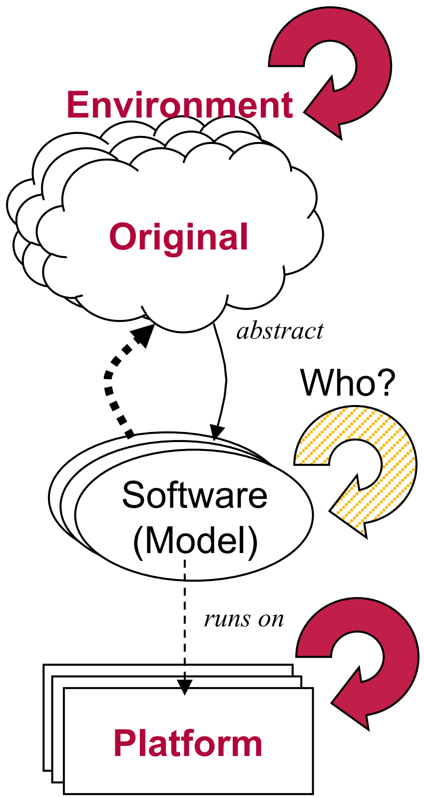
Proposed solution:

- make things simpler for administrators and users of I/T by **automating its management** (Paradoxically, it seems we need to create even more complex systems).
- Inspiration is the massively complex systems of the human body: *the* ***autonomic nervous system.***



I Motivation: Software Landscapes vs. Applications

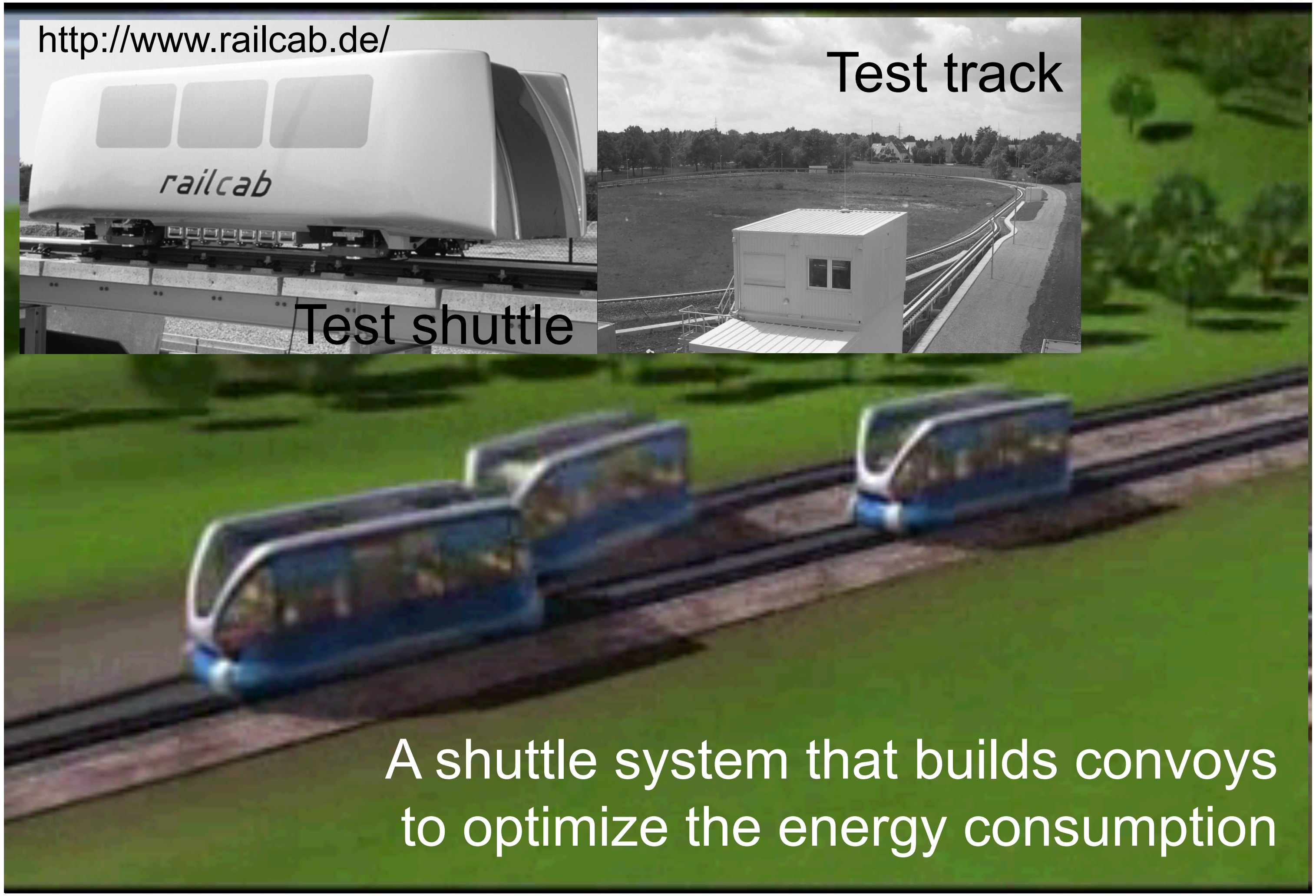




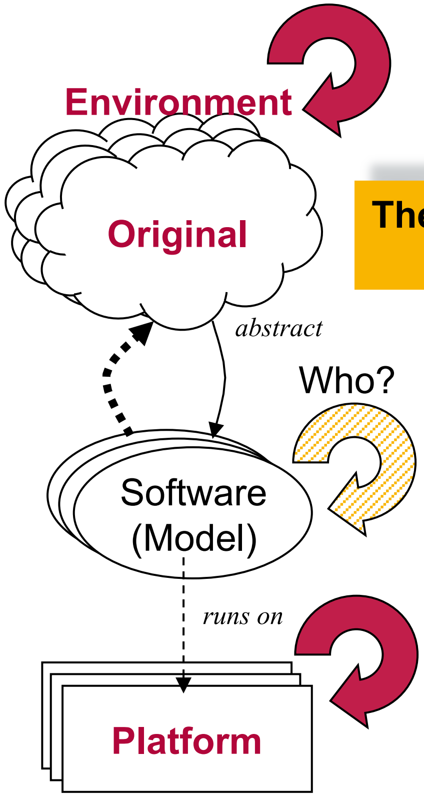
- **Characteristics:** large-scale, heterogeneous, distributed, ad hoc evolution, no central authority
- **May include:** Server backends, embedded subsystems, wireless ad hoc networks, mobile devices, …

The software **must** resolve adaption needs due to changes in the context and platform itself to be able to work at all

I Motivation: Future Software Landscape Example



II Foundations: Self-Adaptiveness



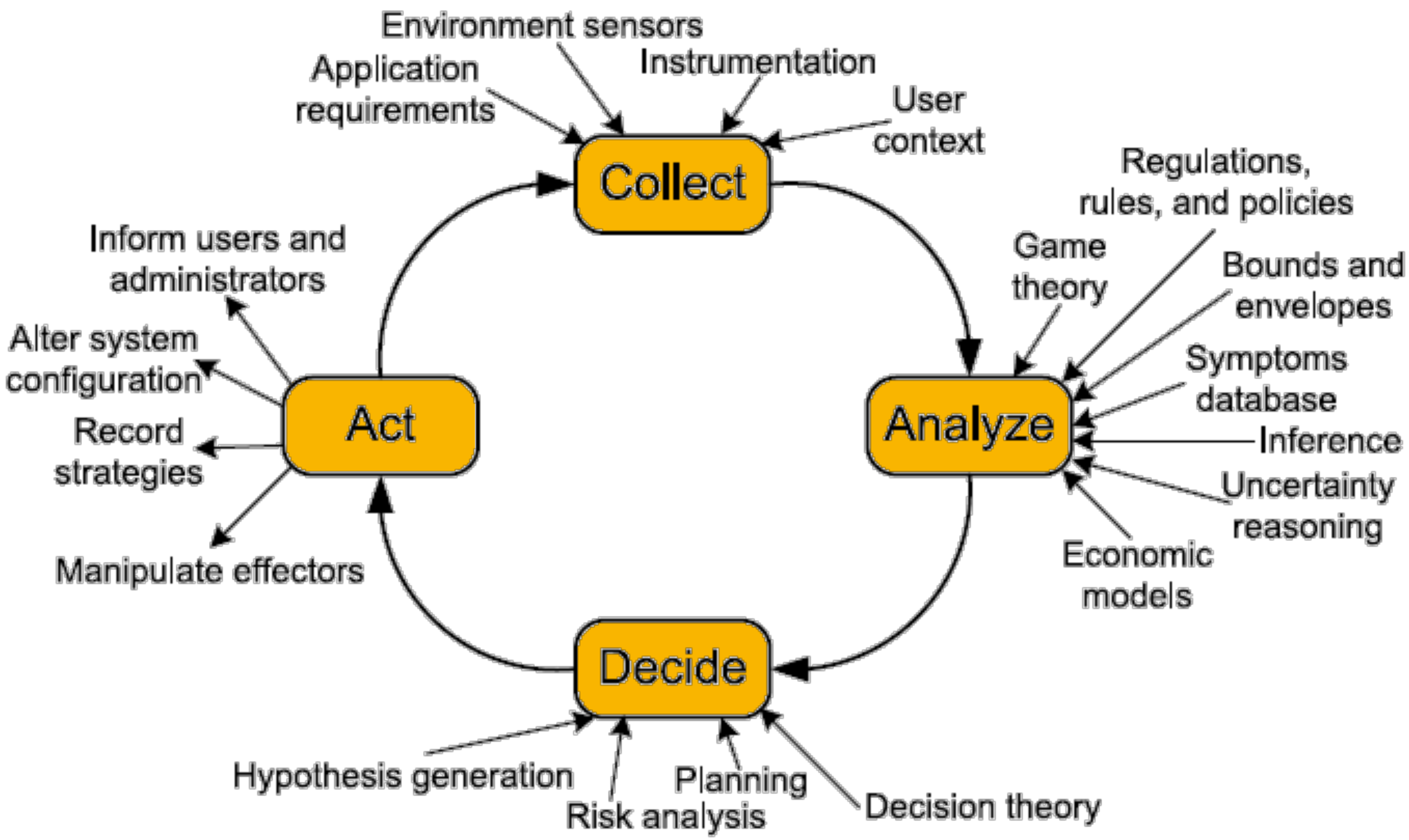
The software itself!

What do we need?

[Salehie&Tahvildari2009]

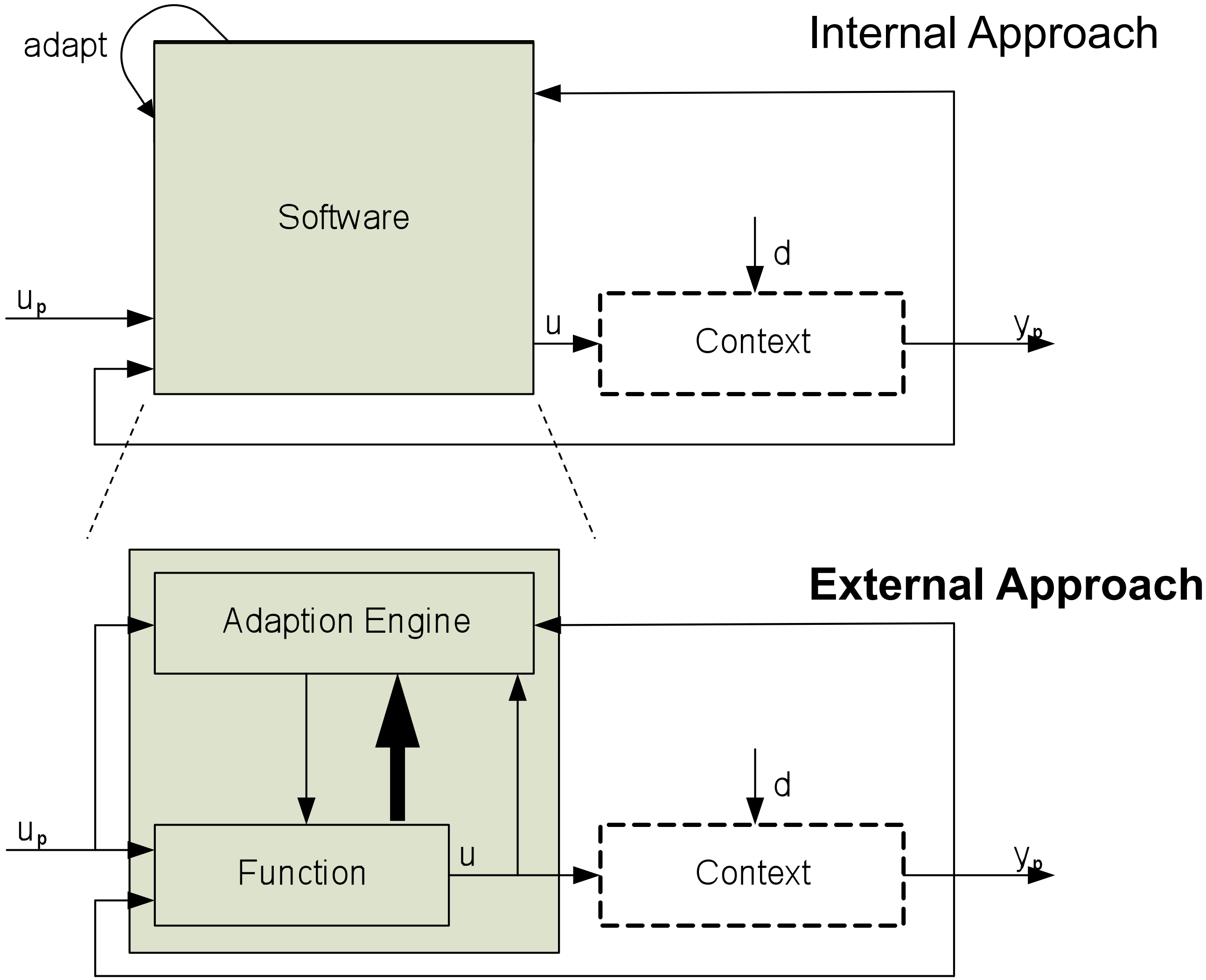
II Foundations: Adaptation Loop

Hasso Plattner Institut



[Brun+2009]

II Foundations: Architecture & Self

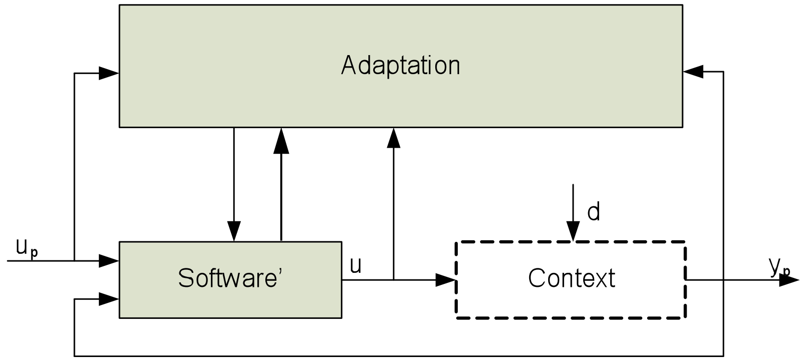


[Salehie&Tahvildari2009]

II Foundations: Models and Adaptation

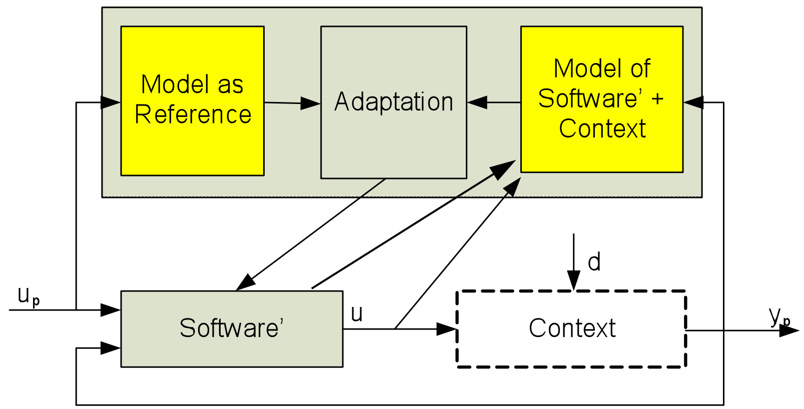
Adapt "without" models:

- Still explicit or implicit **design-time models** are used to guide adaptation processes
- **Limitation:** covers only changes covered by **one** model of the software' + context (potentially including some parameters or structural changes that can be observed)



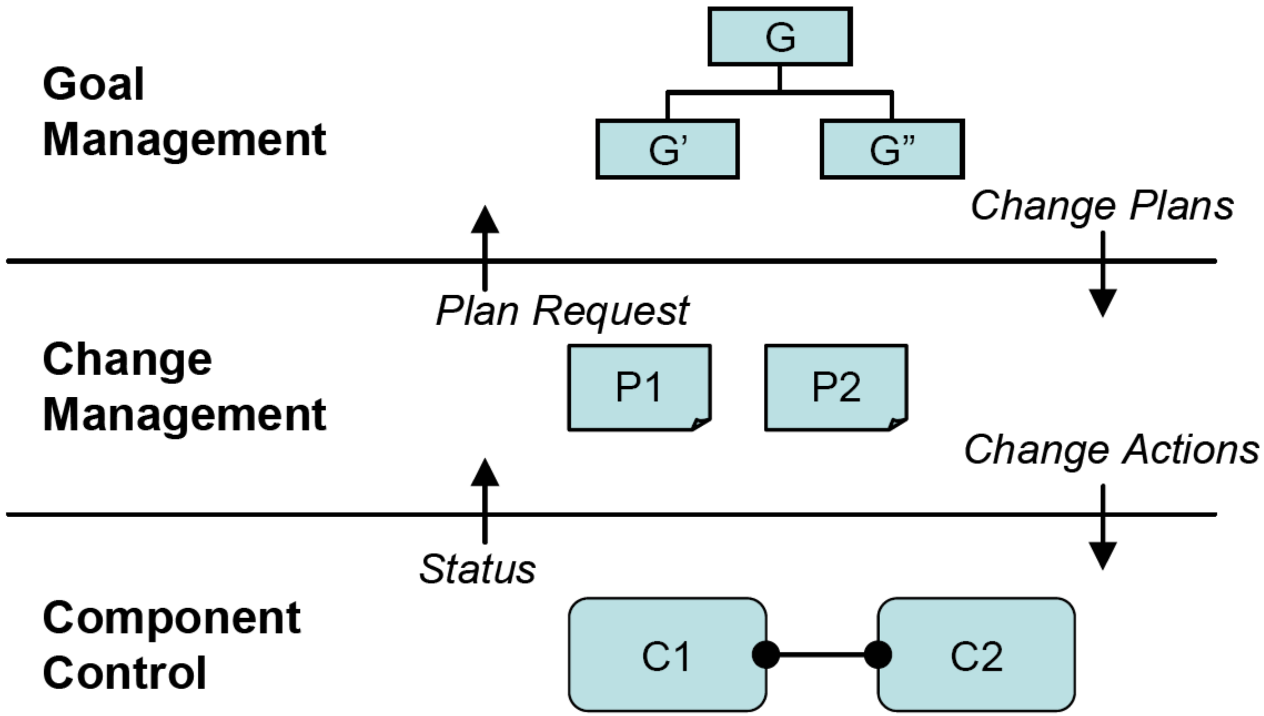
Adapt with runtime models:

- Explicit **runtime models** are used to guide adaptation processes
- **Limitation:** covers only changes captured by the runtime models (**multiple**!); requires correct adjustment of them from the observations



II Foundations: Top-Down Architecture

Reference Architecture for Self-Management:



- Layers for different purposes
- Decoupling of the layers in time

[Kramer&Magee2007]

II Foundations: Bottom-Up Architecture

Hasso Plattner Institut

[Dressler2007]

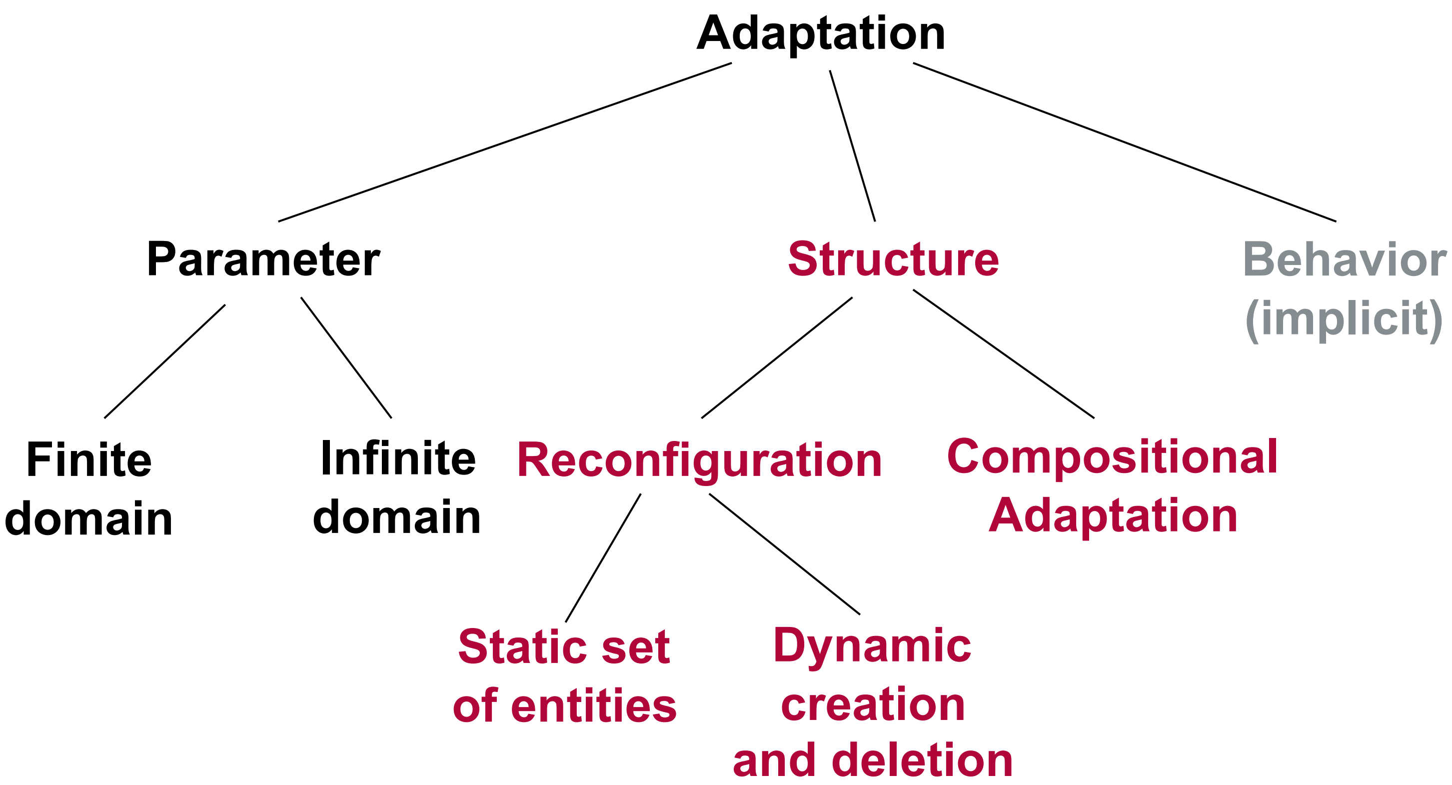
Self-organization is a process in which **structure and functionality** at the global level of a system **emerge** solely **from numerous interactions** among the lower-level components.

Characteristics:

- No central control
- Emerging structure
- Resulting complexity
- High scalability

Emergence is an **apparently meaningful collaboration** of components (individuals) resulting in capabilities of the overall system (far) beyond the capabilities of the single components.

II Foundations: Subject of Adaptation

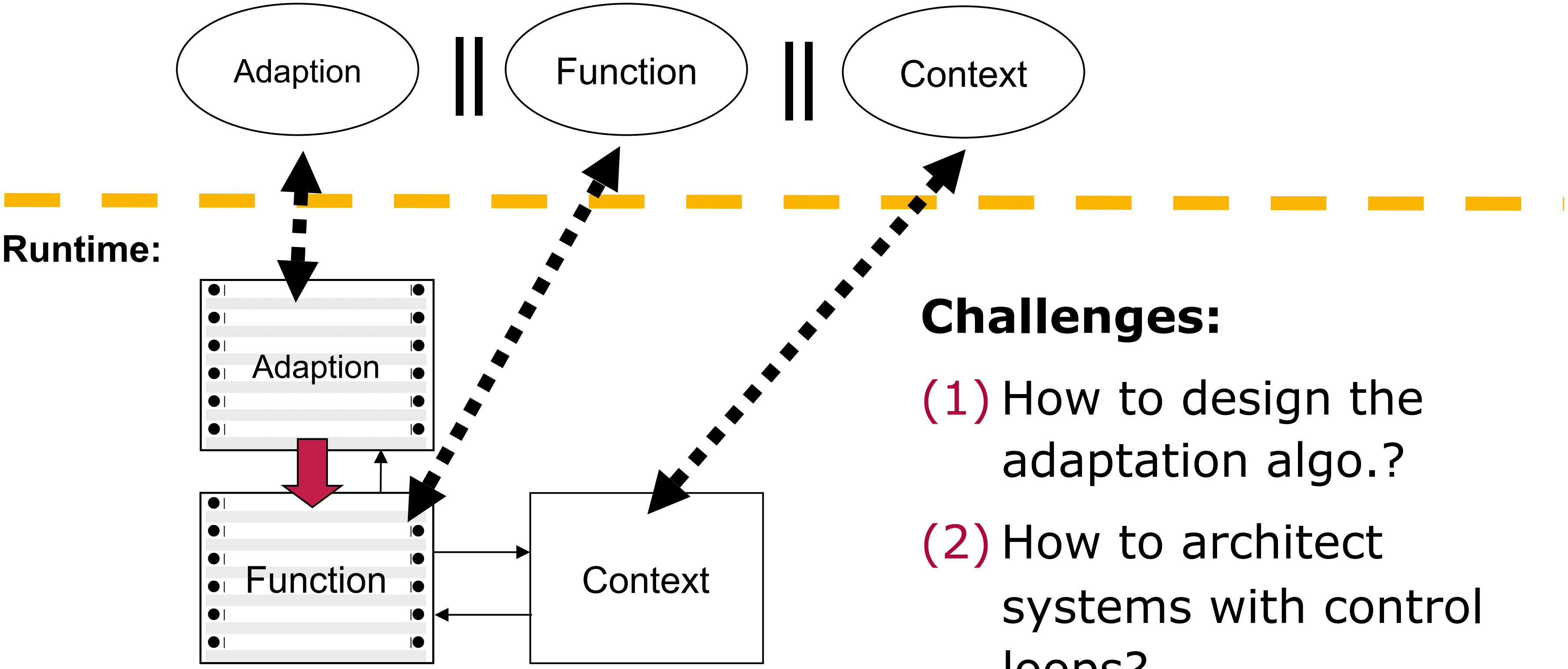


But how can we systematically build the software for such systems (construction)?

III Construction:
Control Loop

Development time:

Runtime:



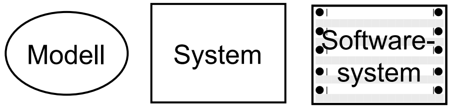
Challenges:

(1) How to design the adaptation algo.?

(2) How to architect systems with control loops?

(3) How to develop the necessary elements of the loop?

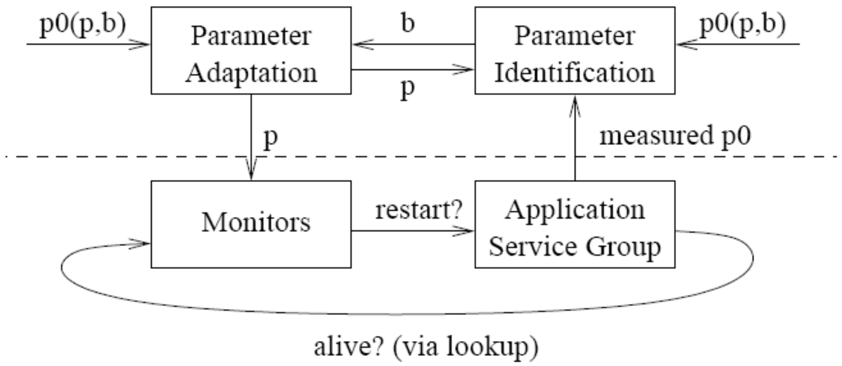
Legend:

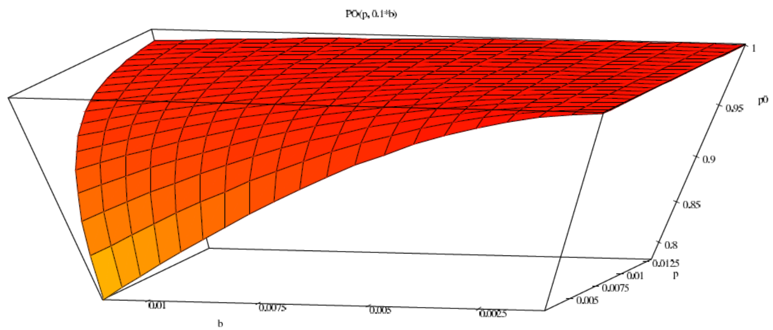


III Construction: Complex (1) design adaptation algo.

Complex development time models:

- Application: Monitoring and Restart of Services
- Instance of the MIAC scheme
- Identification of required reliability and availability parameters via monitoring
- An development time **availability model** in form of an Stochastic Petri Net is used to **precompute** values for the required parameter adaptation (using interpolation)



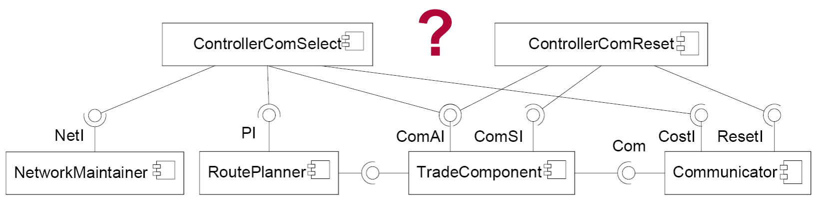


[ADS2004]

III Construction:
(2) Architecting Loops

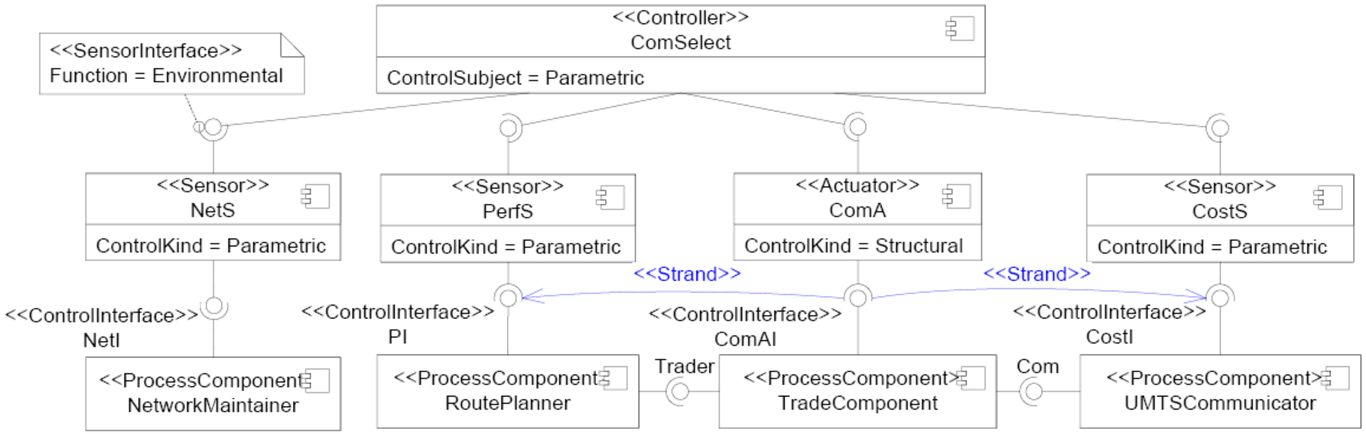
Problem [Brun+2009]:

- Control loops are not directly supported when architecting



Proposal: A UML Profile for feedback loops:

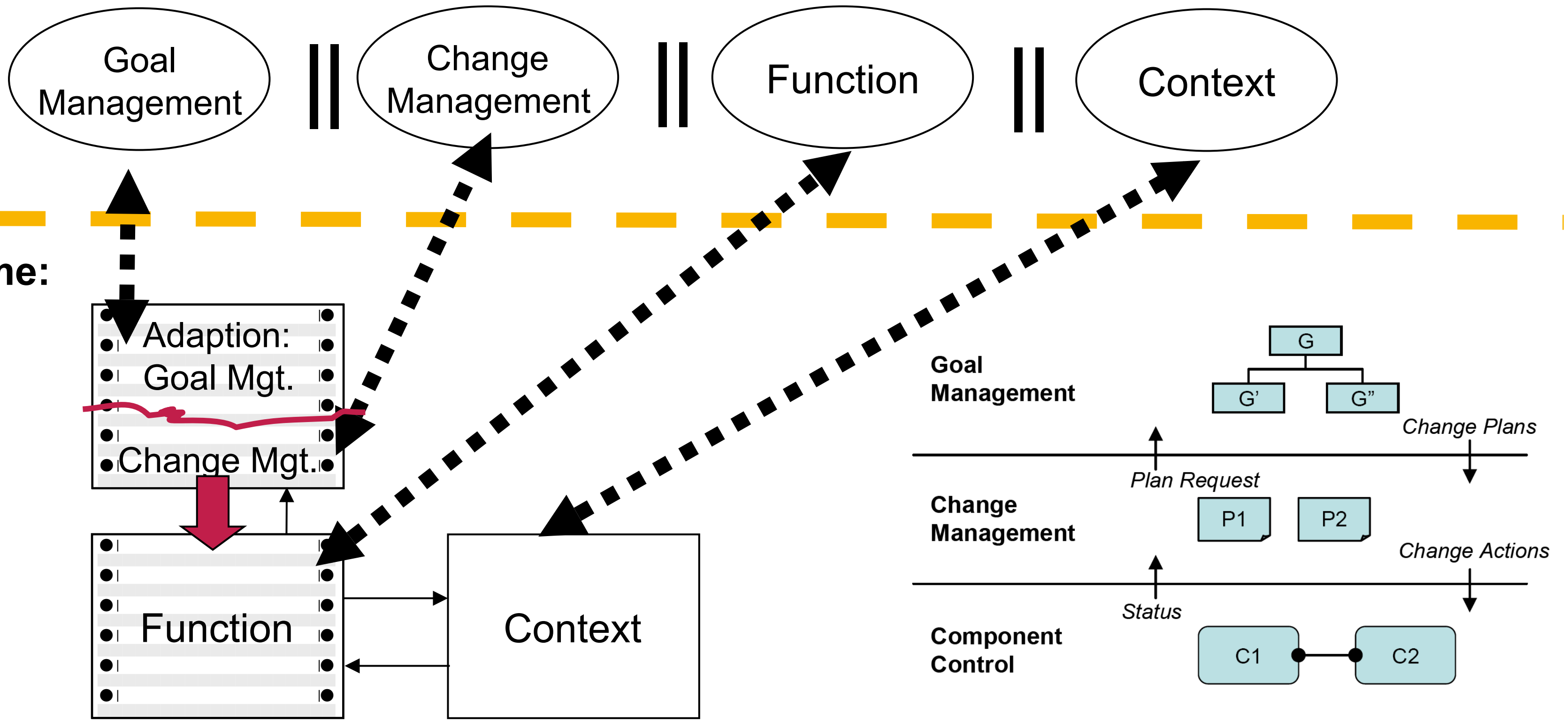
- Identify loop elements and mark them using stereotypes
- Identify whether loops overlap in undesired ways
- Identify control related effects



III Construction: Control Loop & Layers

Development time:

Runtime:



Challenges:

(1) Support layers

(2) Provide decoupling between layers

III Construction: Micro Architecture

Operator-Controller Module

- **Cognitive operator ("intelligence")**

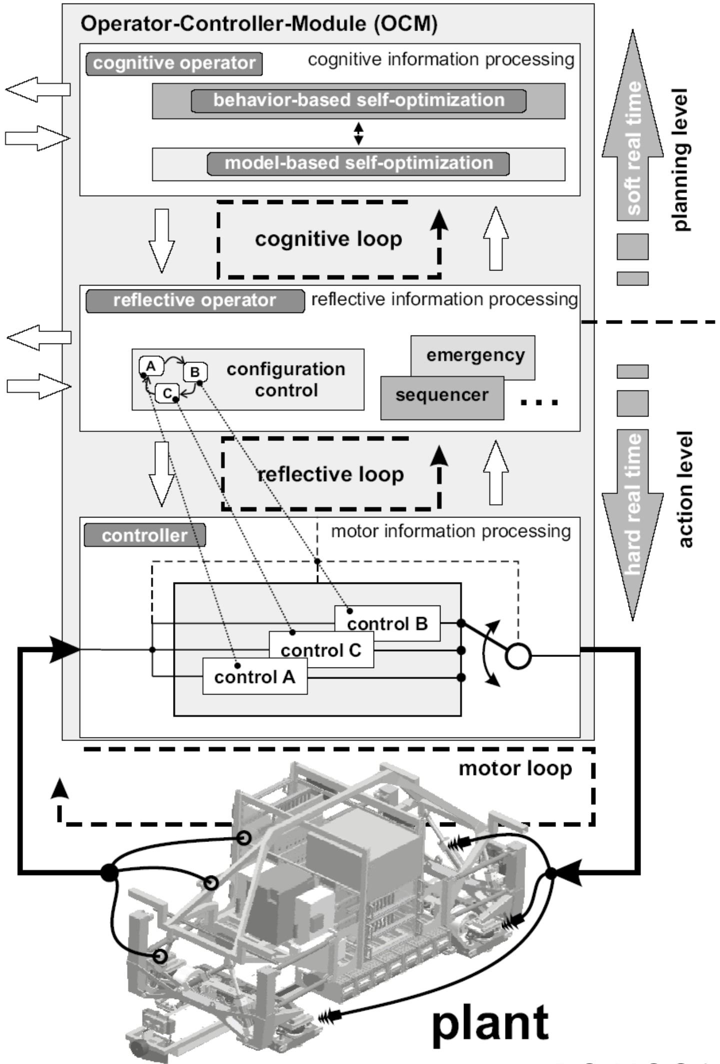
  decoupled from the hard real-time processing

- **Reflective operator**

  Real-time coordination and reconfiguration

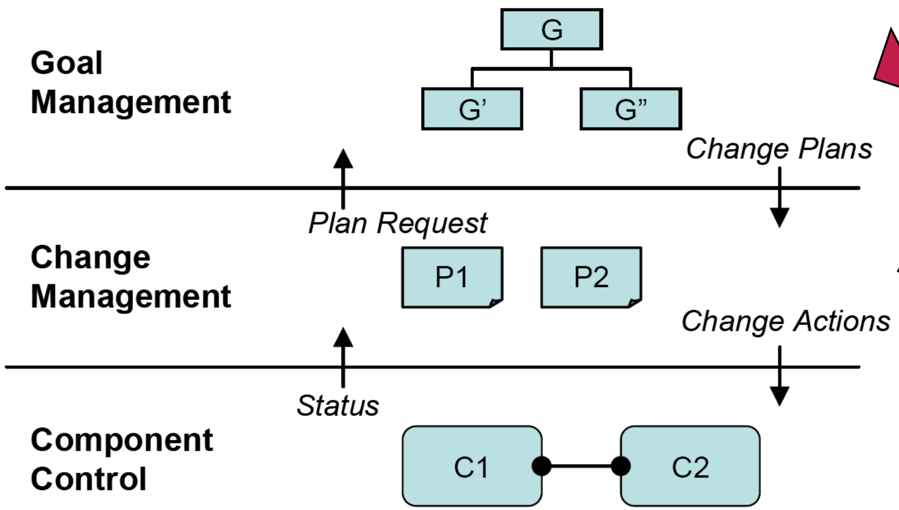
- **Controller**

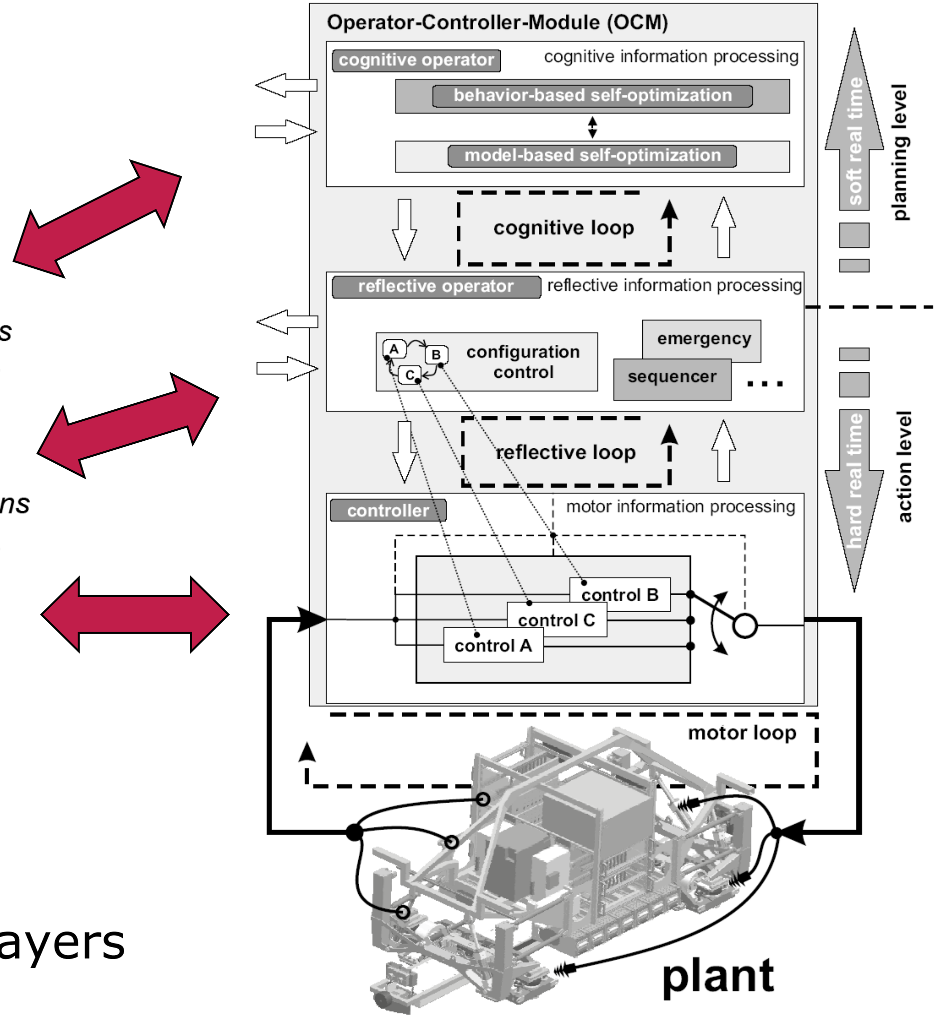
  Control via sensors and actuators in hard real-time



[ICINCO04]

III Construction: Relation to the Reference Architecture



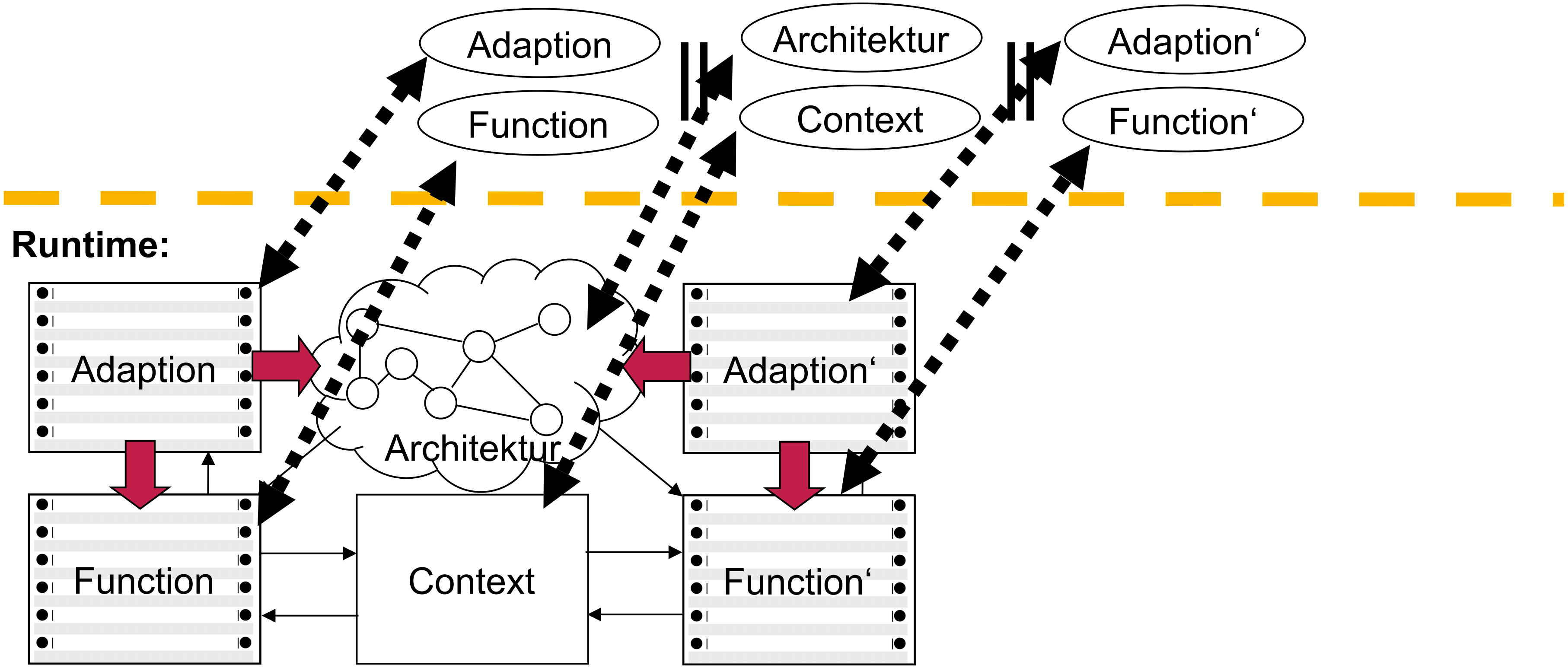


(1) Support layers

(2) Provides decoupling between layers

III Construction: Control Loop & Architecture

Development time:



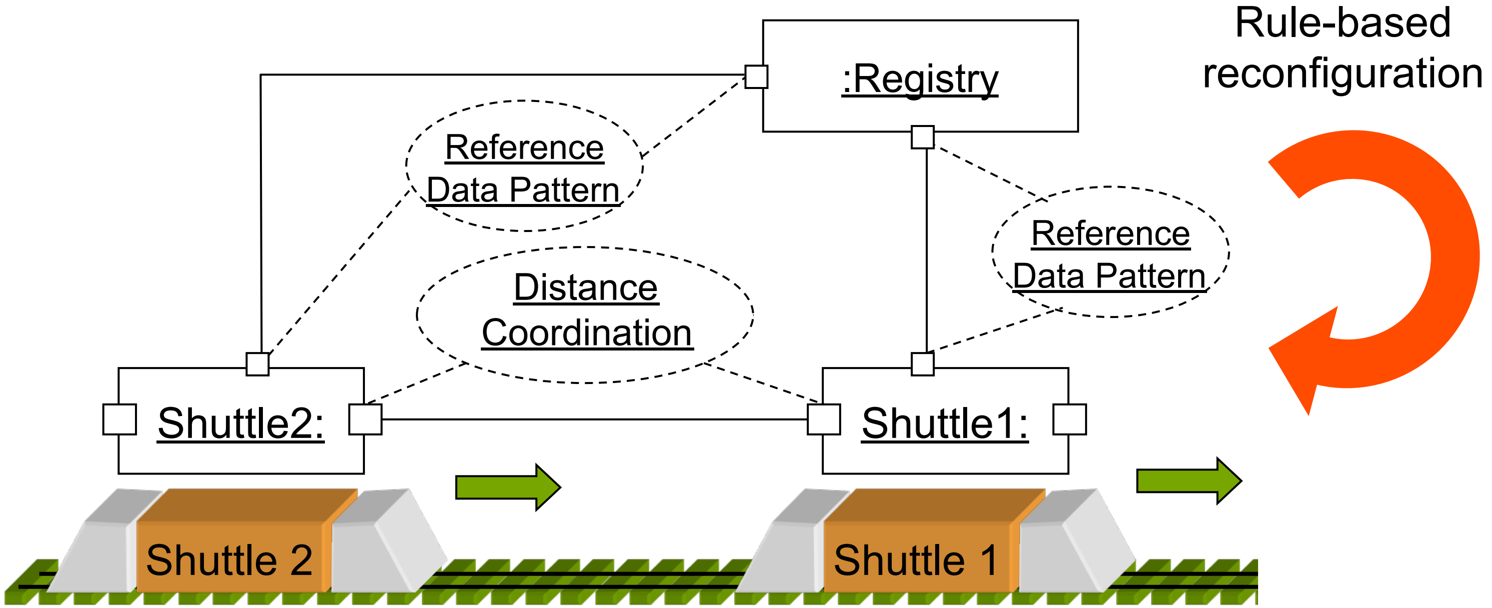
Relevant cases:

(1) Hierarchies

(2) Self-organizing

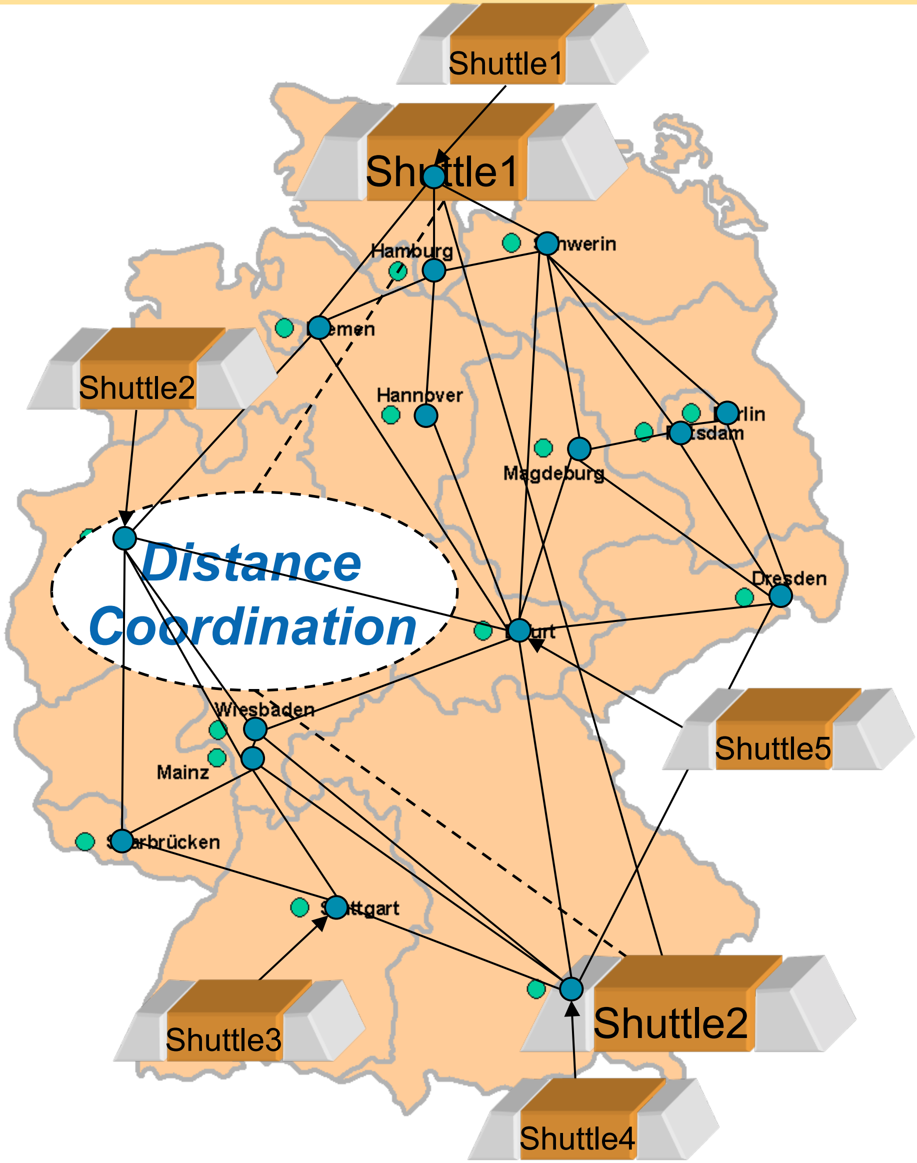
III Construction: Complex Coordination

- **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



III Construction: Rule-Based Reconfiguration

Hasso Plattner Institut



Problem:

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

Solution:

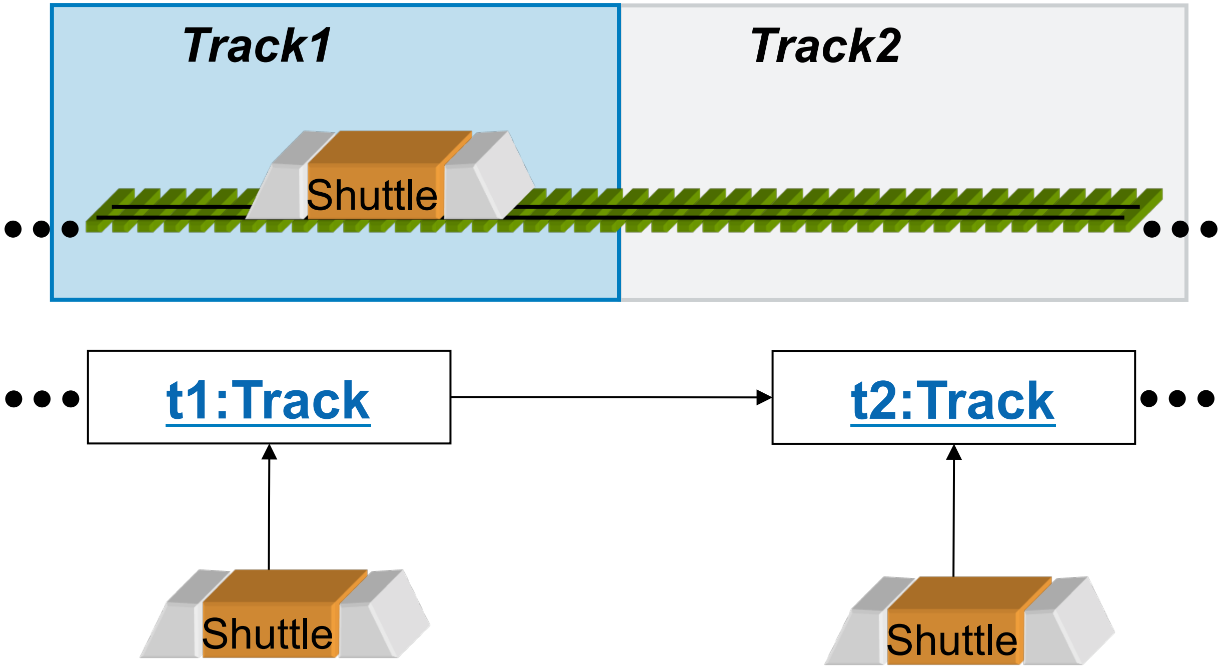
- State = Graph
- Reconfiguration rules = graph transformation rules
- Safety properties = forbidden graphs

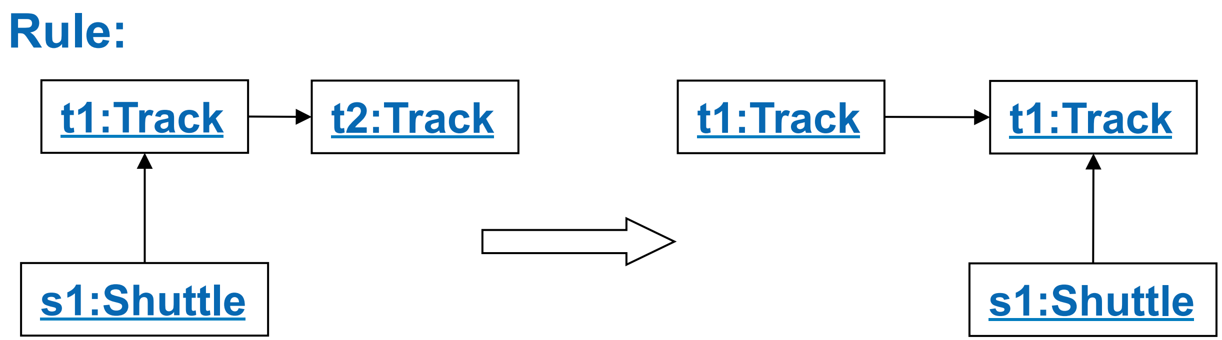
⇨ Formal Verification possible

III Construction: Rule-Based Reconfiguration

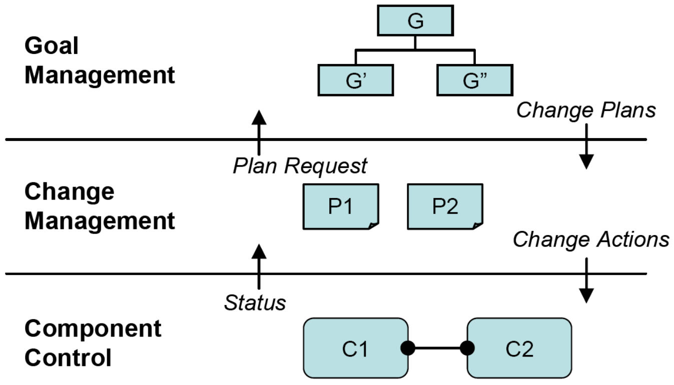
Apply Graph Transformation Systems

- Map the tracks
- Map the shuttles
- Map the shuttle movement to rules (move- ment equals reconfiguration)





III Construction: (2) Self-organizing



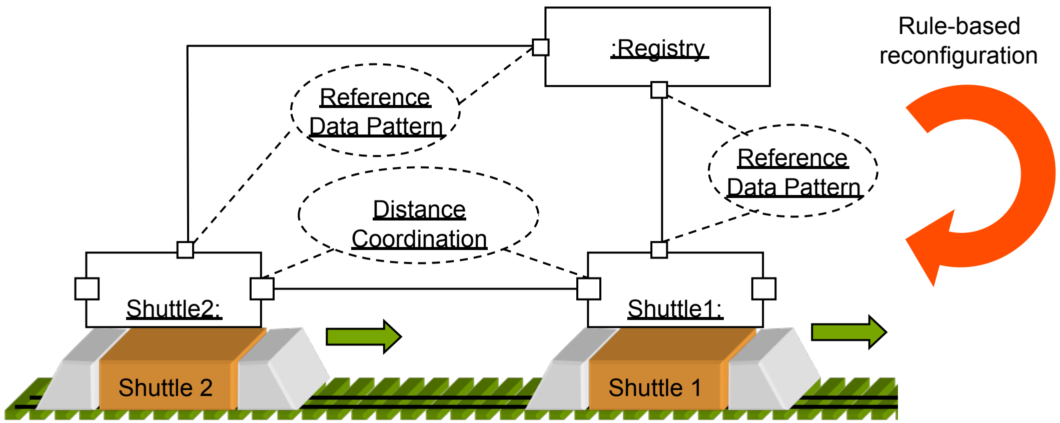
Self-organizing (degrees of freedom for the local rule-based configuration)

Rule-based configuration

distributed over the patterns and the components realizing the pattern roles

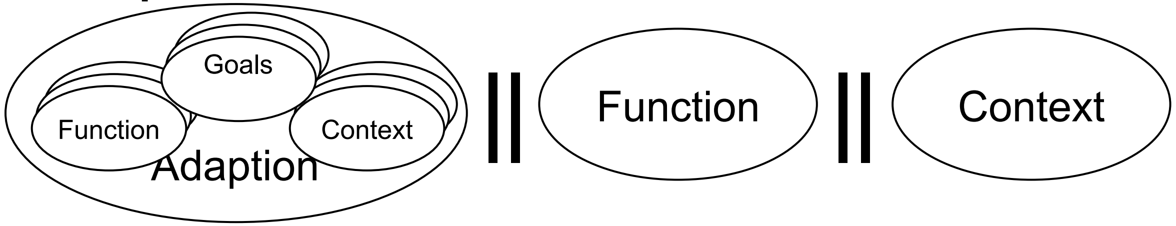
Difference:

- 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

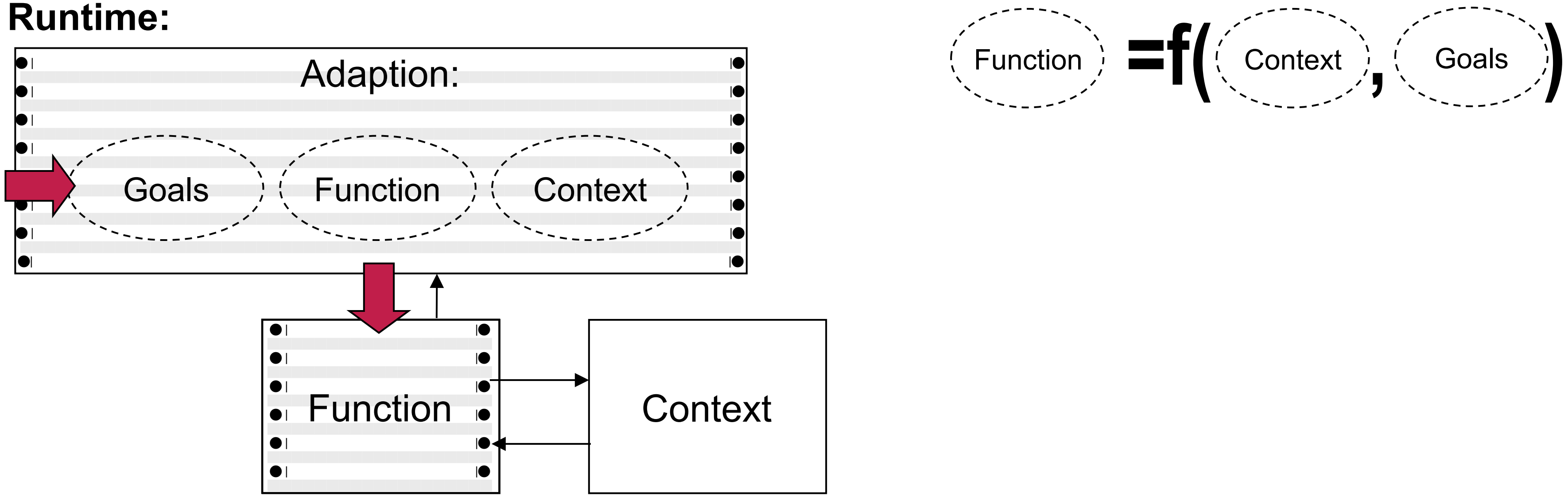


III Construction:
Runtime Models

Development time:



Runtime:



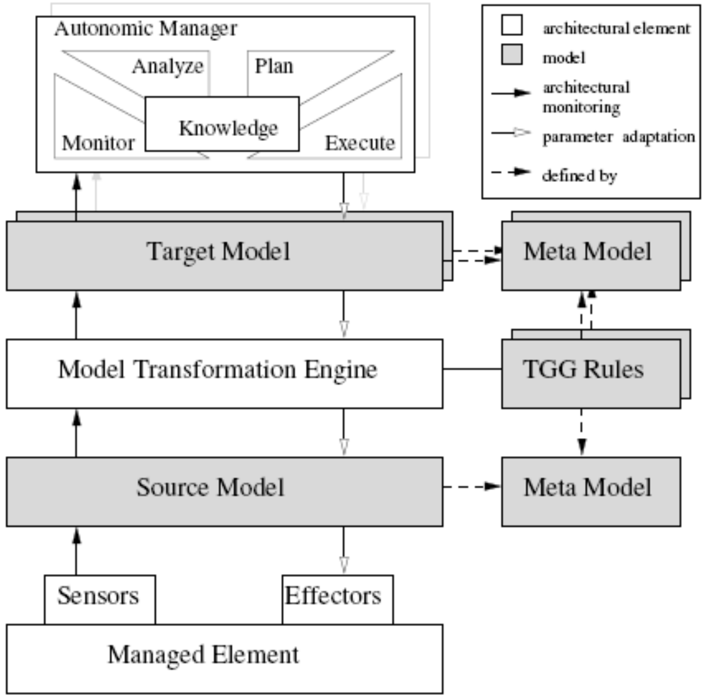
Challenge:

(1) Efficient and cost-effective realization of the runtime models

(2) Efficient and cost-effective realization of the function update (**f**)

III Construction:
(1) Runtime Models: MDE

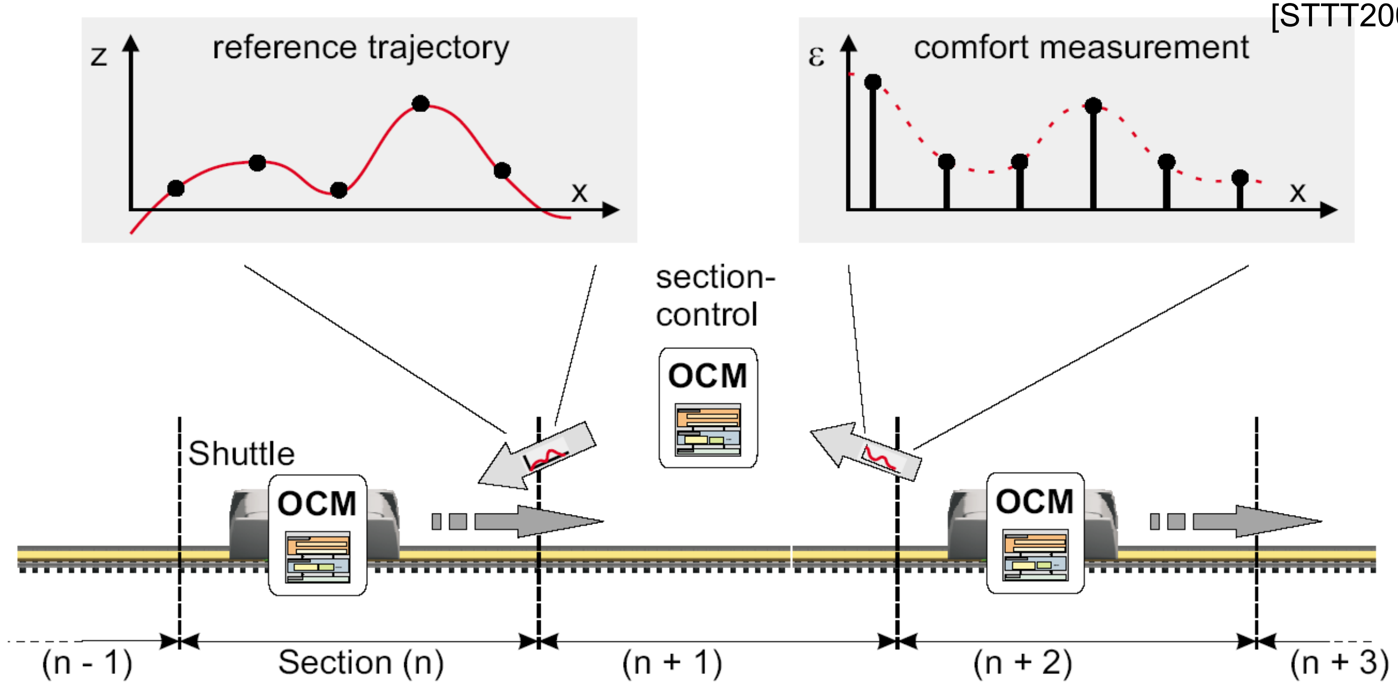
- Supports **adaptation loops** for models using “meta-models” (EMF) and bidirectional model transformation techniques (Tripple Graph Grammars) for an EJB application server
- Extract abstract runtime models for different autonomic managers for **monitoring** EJB applications (**unchanged**)
- **Adapting** managed subsystem via extracted runtime models (parameter and structural adaptation; not as easy as monitoring!)
- Synchronize runtime models **incrementally** (faster as non incremental manual implementations)



[ICAC2009]

III Construction:
(2) Function Update (Distributed)

[STTT2008]



- 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

But how can we guarantee that they have a sufficient quality (quality assurance)?

IV Quality Assurance:
Development time models

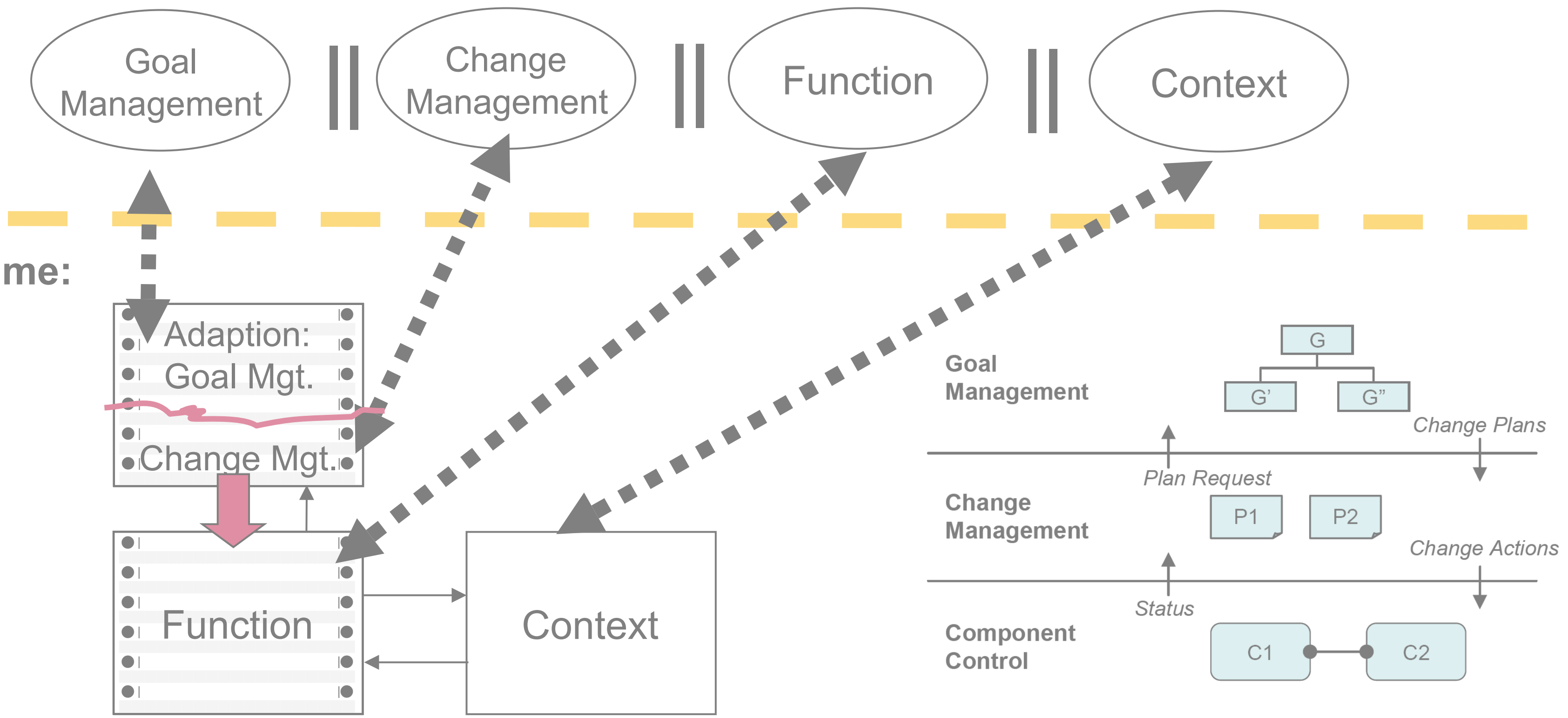
Bottom line: Self-adaptive systems must simply be "better" and not "worse"

(1) Correct working adaptation algorithm
(2) Correct adaptation implementation
   a. Correct monitoring: handle measurement failures; …
   b. Correct system analysis: consistent with real changes; …
   c. Correct adaptation decisions: fits to real changes; guarantees required properties; …
   d. Correct execution of the adaptation: consistent update; timing, …

IV Quality Assurance: Control Loop & Layers

Development time:

Runtime:



- Correct working adaptation algorithm (1) ⇨ if simple properties, abstract models can be formally verified
- Correct adaptation implementation (2) ⇨ Can be tackled to some extend if we abstract from adaptation details (consider only change management)

IV Quality Assurance: Correct (1) + (2)

Operator-Controller Module (OCM) for

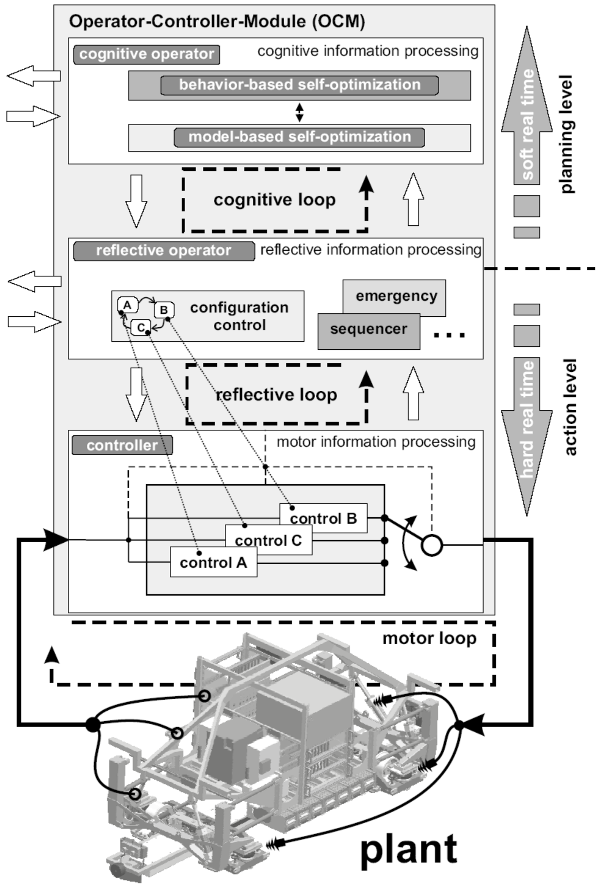
- **Cognitive operator (CO)**
- **Reflective operator (RO)**
- **Controller (C)**

Formal verification ("RO part" only):

- Formal model covers possible pre-planned configuration steps
- Only consistent and steps of the controller that the reflective operator can do within required time bounds occur (correct (1))

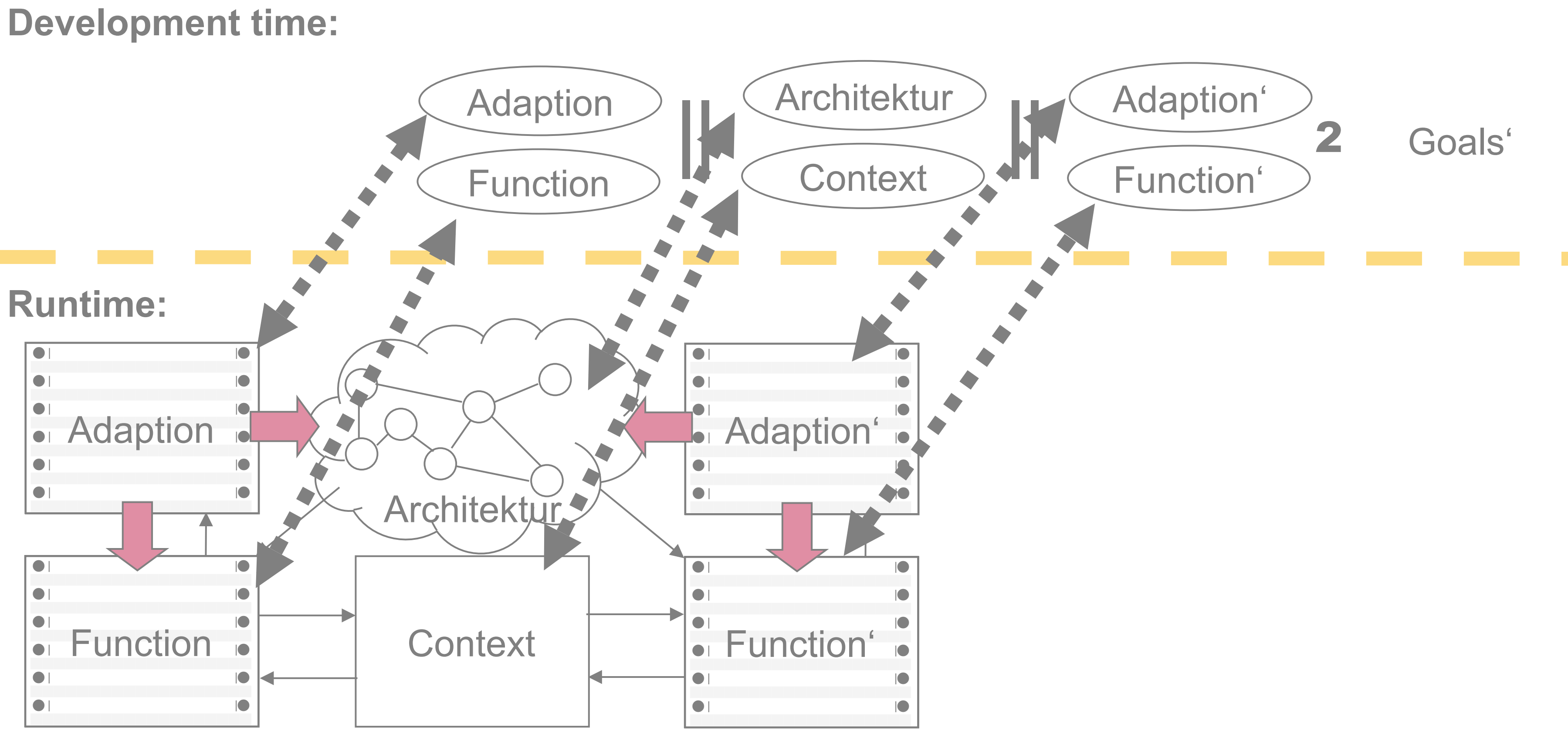
Code generation:

- guarantees functional and timing properties (correct (2))



[FSE2004]

IV Quality Assurance: Control Loop & Architecture



- Correct working adaptation algorithm (1) ⇨ simple properties for abstract models can be formally verified, if we abstract from adaptation details (consider only change management) and decomposing the problem (apply a modular or compositional reasoning schemes)

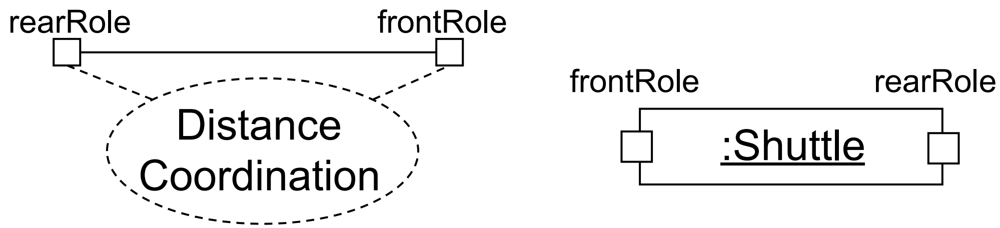
IV Quality Assurance:
(1) Correct adaptation algo. (1/3)

Hasso Plattner Institut

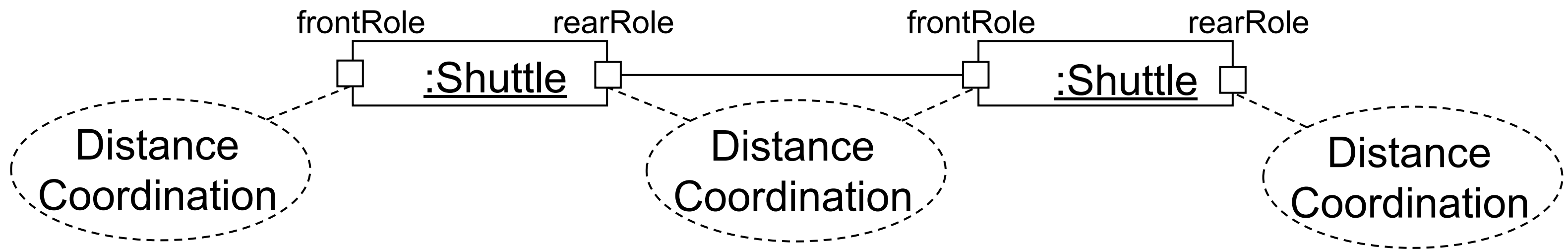
[ESE/FSE2003]

Decompose verification:

- Verification guarantees properties for the collaborations
- Verification guarantees conformance for components (ports refine roles)



Compositional result: Properties hold for all collaborations in correctly composed component deployments

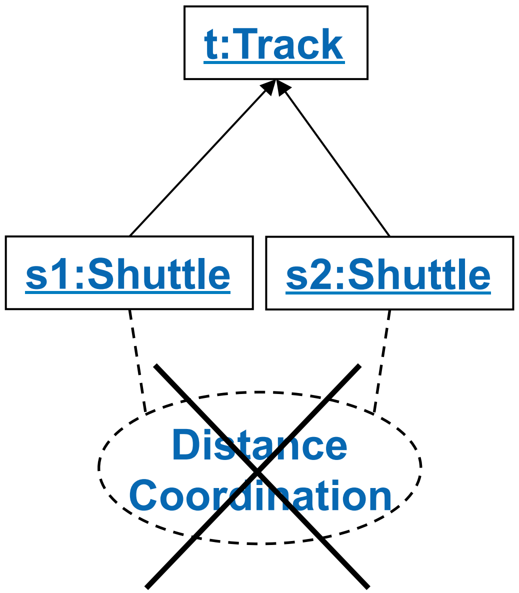


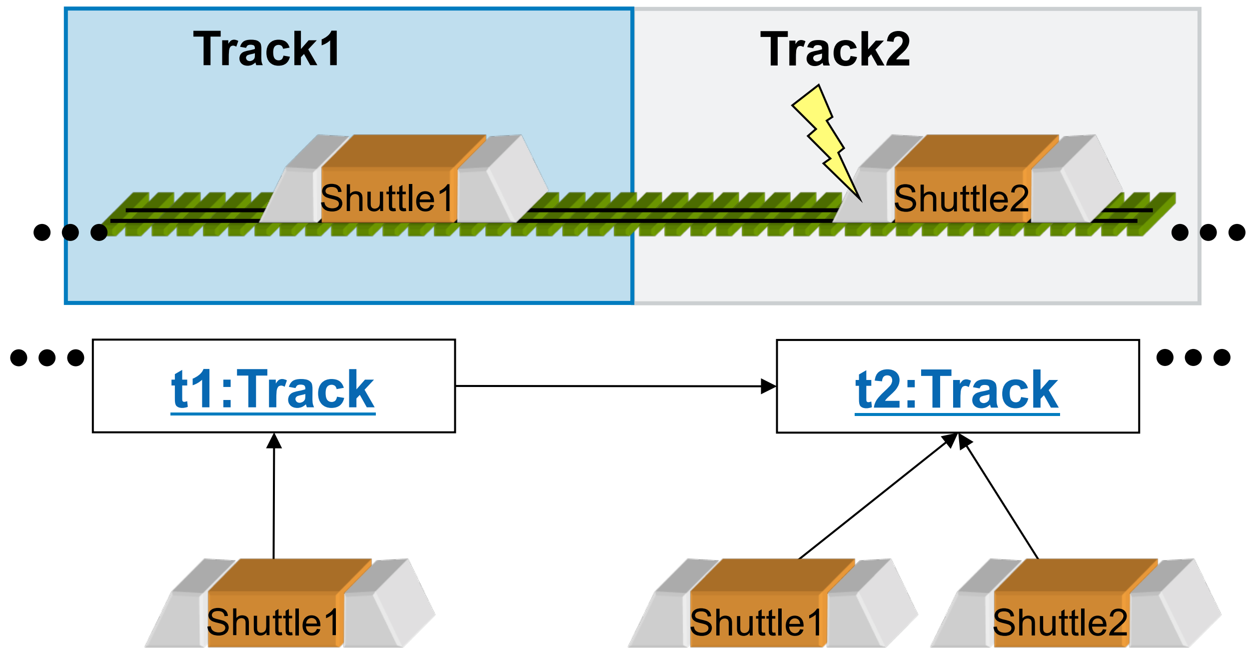
But, it is yet not guaranteed that shuttles nearby are connected via a collaboration!

IV Quality Assurance:
(1) Correct adaptation algo. (2/3)

Hasso Plattner Institut

Forbidden Graph





- **Correctness:** all reachable system graphs do not match the forbidden graph pattern

Problems:

- there could be **infinite** many reachable system graphs
- fixed initial topology **not known** (may change)

Now, both results together would guarantee the absence of collisions!

[Monterey2007]

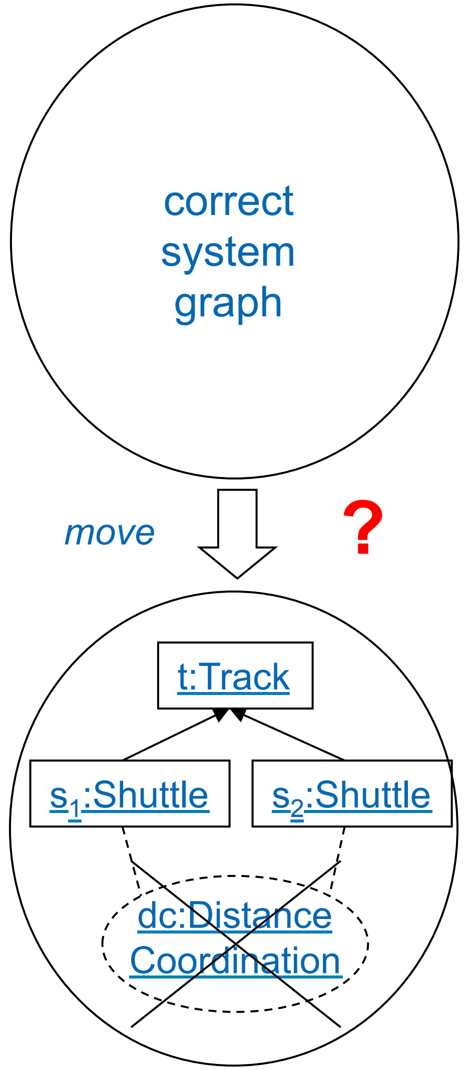
IV Quality Assurance:
(1) Correct adaptation algo. (3/3)

Verification:

- Analyze whether structural changes can lead from safe to unsafe situations (**inductive invariants**)

Checking Options:

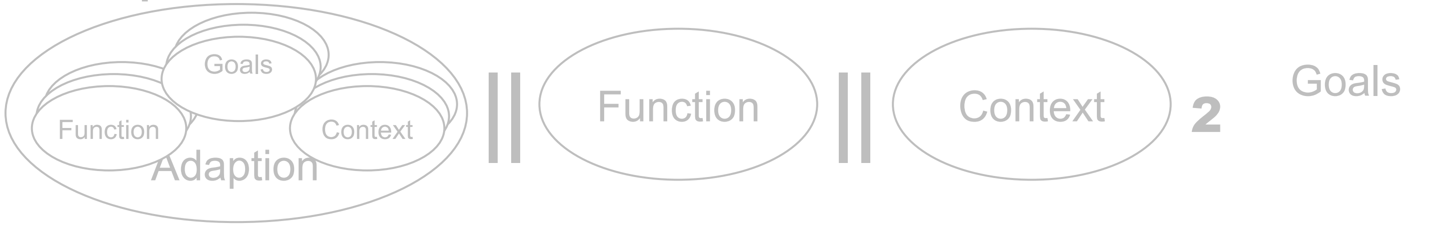
- Model Checking (mapping to GROOVE; only debugging)
  - Limited to small configurations and finite models
  - Extension for continuous time have been developed
- Invariant Checker (our own development)
  - Supports infinite many start configurations specified only by their structural properties
  - Supports infinite state models
  - Extension of time and discrete variables exist
  - Incremental check for changed rules
  - Extension of hybrid behavior (recently!)



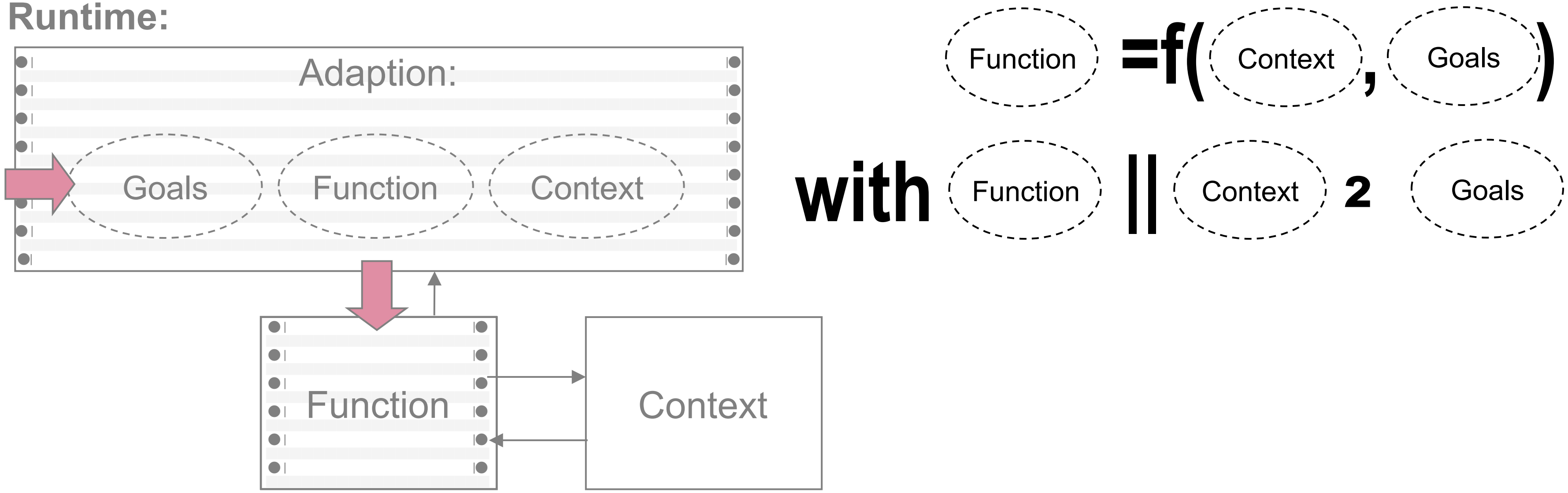
[ICSE2006, ISORC2008]

IV Quality Assurance: Runtime Models

Development time:



Runtime:

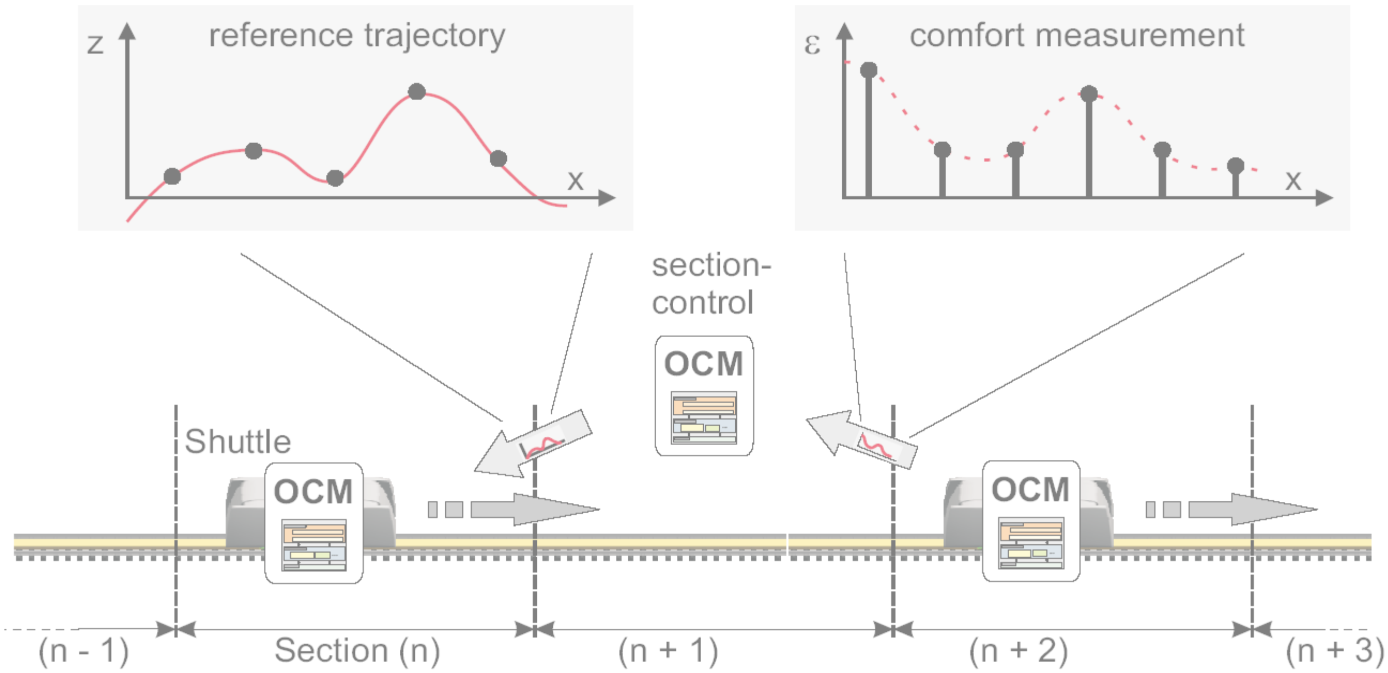


- Guarantee correct working adaptation algorithm (1)?

⇨ If solver for **f** exists, correctness can be derived

- Correct adaptation implementation (2) ⇨ Can be tackled by MDE

IV Quality Assurance:
(1) Correct adaptation algo.



- 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 for **f**
- Backup for the case of data errors!

[STTT2008]

III Construction:
(2) Correct implementation

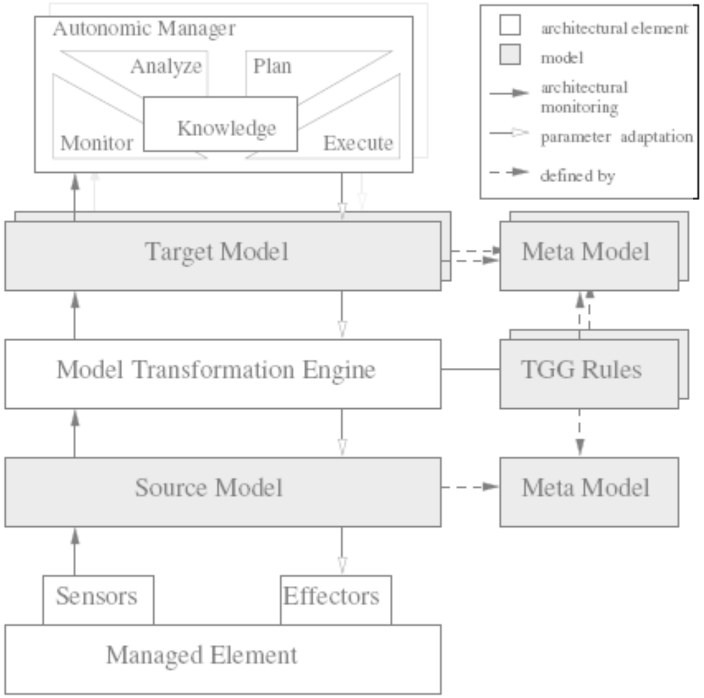
Correct adaptation implementation (2):

a. Correct system model updates:
   valid abstraction by construction

b. Correct system model analysis

c. Correct adaptation decisions

d. Correct execution of the adaptation (special case: propagate changes in updates system model):
   functional correctness by construction; timing?



[ICAC2009]

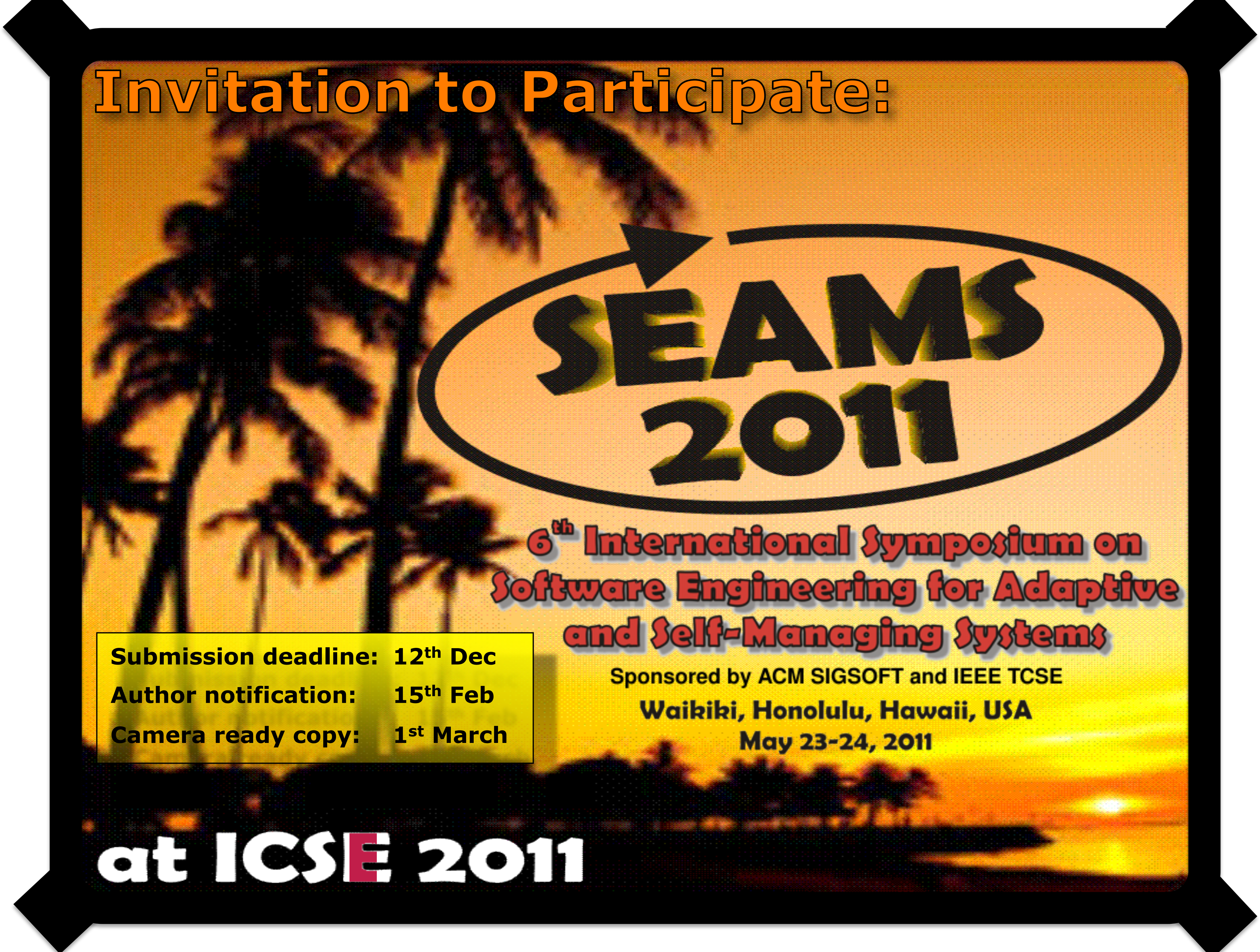
V Conclusion & Outlook

- **Self-adaptive systems** promises to automate the efforts required today to adapt the software (by the developer and admin) as well as enables software landscapes not feasible without. However, it also makes the software even more complex.
- Therefore, techniques for the systematic and cost-effective **software engineering of self-adaptive systems** are crucial for the while vision.
  - **Construction** (adaptation algo., loops, layers, hierarchies, self-organizing, runtime models, function updates, …)
  - **Quality assurance** (adaptation algo., loops, layers, hierarchies, self-organizing, runtime models, function updates, …)

Conclusion & Outlook

- **Models** and **model-driven engineering** can play a major role for the cost-effective construction and quality assurance of such systems.
  - **Development time models** permit to construct such systems and verify the correctness of the adaptation algo.
  - In case of **runtime models**, suitable function updates can be constructed and verified to show the correctness pf the adaptation alog.
  - Model-driven engineering can often assure via **code generation** that the verified properties also hold for the running system
  - In case of **runtime models**, model-driven engineering can in addition be employed to provide a basis for structural adaptation that guarantees correct implementation.

But much left to be done ...


Invitation to Participate:
SEAMS 2011
6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems
Sponsored by ACM SIGSOFT and IEEE TCSE
Waikiki, Honolulu, Hawaii, USA
May 23-24, 2011
Submission deadline: 12th Dec
Author notification: 15th Feb
Camera ready copy: 1st March
at ICSE 2011


SEAMS
2011
See you in
Hawaii

References (1/2)

[AC2001] *Autonomic Computing: IBM's Perspective on the State of Information Technology*. International Business Machines Corporation, 2001.

[Andersson+2009] Jesper Andersson, Rogério de Lemos, Sam Malek, Danny Weyns: *Modeling Dimensions of Self-Adaptive Software Systems*. Software Engineering for Self-Adaptive Systems 2009: 27-47

[Brun+2009] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger M. Kienle, Marin Litoiu, Hausi A. Müller, Mauro Pezzè, Mary Shaw: *Engineering Self-Adaptive Systems through Feedback Loops*. Software Engineering for Self-Adaptive Systems 2009: 48-70

[Cheng+2009] Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, Jon Whittle: *Software Engineering for Self-Adaptive Systems: A Research Roadmap*. Software Engineering for Self-Adaptive Systems 2009: 1-26

[Dressler2007] Falko Dressler, *Self-Organization in Sensor and Actor Networks*, Chichester, John Wiley & Sons, 2007.

[Kramer&Magee2007] Kramer, J. and Magee, J. 2007. Self-Managed Systems: an Architectural Challenge. In 2007 Future of Software Engineering (May 23 - 25, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 259-268.

References (2/2)

[Oreizy+1999] Peyman Oreizy, Michael M. Gorlick, Richard Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum and Alexander L. Wolf. *An Architecture-Based Approach to Self-Adaptive Software*. In IEEE Intelligent Systems, Vol. 14(3):54--62, June 1999.

[Kephart&Chess2004] Jeffrey O. Kephart and David Chess. *The Vision of Autonomic Computing*. In Computer, Vol. 36(1):41--50, IEEE Computer Society Press, Los Alamitos, CA, USA, January 2003

[Kokar+1999] Mieczyslaw M. Kokar, Kenneth Baclawski and Yonet A. Eracar. *Control Theory-Based Foundations of Self-Controlling Software*. In IEEE INTELLIGENT SYSTEMS, Vol. 14(3):37-45, Article, 1999.

[Serugendo+2004] Giovanna Di Marzo Serugendo, Noria Foukia, Salima Hassas, Anthony Karageorgos, Soraya Kouadri Mostéfaoui, Omer F. Rana, Mihaela Ulieru, Paul Valckenaers and Chris Van Aart. *Self-Organisation: Paradigms and Applications*. In Engineering Self-Organising Systems, Vol. 2977:1--19 of Lecture Notes in Computer Science, 2004.

[Salehie&Tahvildari2009] Mazeiar Salehie and Ladan Tahvildari. *Self-adaptive software: Landscape and research challenges*. In ACM Trans. Auton. Adapt. Syst., Vol. 4(2):1--42, ACM, New York, NY, USA , 2009.

Own References

[ESEC/FSE03] Giese, H., Tichy, M., Burmester, S., and Flake, S. 2003. Towards the compositional verification of real-time UML designs. In *Proceedings of the 9th European Software Engineering Conference Held Jointly with 11th ACM SIGSOFT international Symposium on Foundations of Software Engineering* (Helsinki, Finland, September 01 - 05, 2003). ESEC/FSE-11. ACM, New York, NY, 38-47.

[FSE04] 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.

[ICINCO04] Thorsten Hestermeyer, Oliver Oberschelp, and Holger Giese, 'Structured Information Processing For Self-optimizing Mechatronic Systems', in *Proc. of 1st International Conference on Informatics in Control, Automation and Robotics (ICINCO 2004), Setubal, Portugal* (Helder Araujo, Alves Vieira, Jose Braz, Bruno Encarnacao, and Marina Carvalho, eds.), pp. 230--237, INSTICC Press, August 2004.

[ADS2004] Matthias Tichy and Holger Giese, *A Self-Optimizing Runtime Architecture for Configurable Dependability of Services*. Architecting Dependable Systems II, (Rog\'erio de Lemos and Cristina Gacek and Alexander Romanovsky, ed.), vol. 3069, Lecture Notes in Computer Science (LNCS), Springer Verlag, 2004. p. 25–51,

[WADS2005] 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,

[ICSE06] Basil Becker, Dirk Beyer, Holger Giese, Florian Klein, and Daniela Schilling, 'Symbolic Invariant Verification for Systems with Dynamic Structural Adaptation', in *Proc. of the 28th International Conference on Software Engineering (ICSE), Shanghai, China*, pp. 72--81, ACM Press, 2006.

[Monterey2007] Holger Giese, Modeling and Verification of Cooperative Self-adaptive Mechatronic Systems, Reliable Systems on Unreliable Networked Platforms - 12th Monterey Workshop 2005 . Laguna Beach, CA, USA, September 22-24,2005 . Revised Selected Papers, (Fabrice Kordon and Janos Sztipanovits, ed.), vol. 4322, Lecture Notes in Computer Science, Springer Verlag, 2007, p. 258-280,

[ISORC2008] Basil Becker and Holger Giese, On Safe Service-Oriented Real-Time Coordination for Autonomous Vehicles, In Proc. of 11th International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC), vol. , IEEE Computer Society Press, 5 2008, p. 203–210,

[STTT2008] 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.

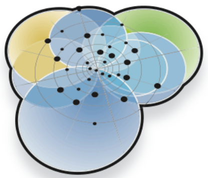
[ICAC2009] Vogel, T., Neumann, S., Hildebrandt, S., Giese, H., Becker, B.: Model-Driven Architectural Monitoring and Adaptation for Autonomic Systems. In: Proc. of the 6th International Conference on Autonomic Computing and Communications (ICAC'09), Barcelona, Spain, ACM (15-19 June 2009).

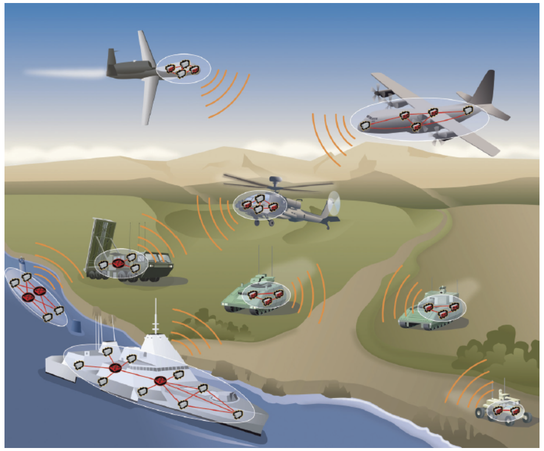
Additional Slides

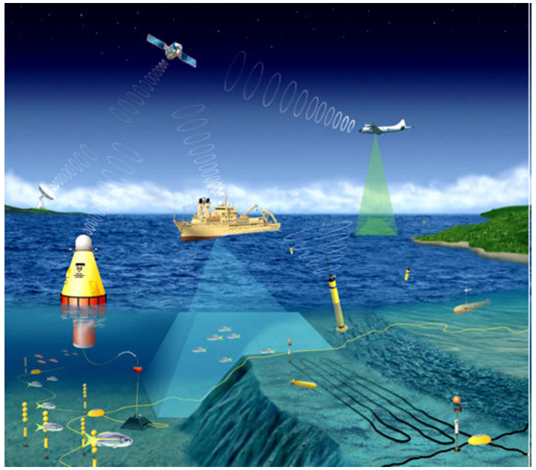
I Motivation: Future Software Landscapes

Prognoses:

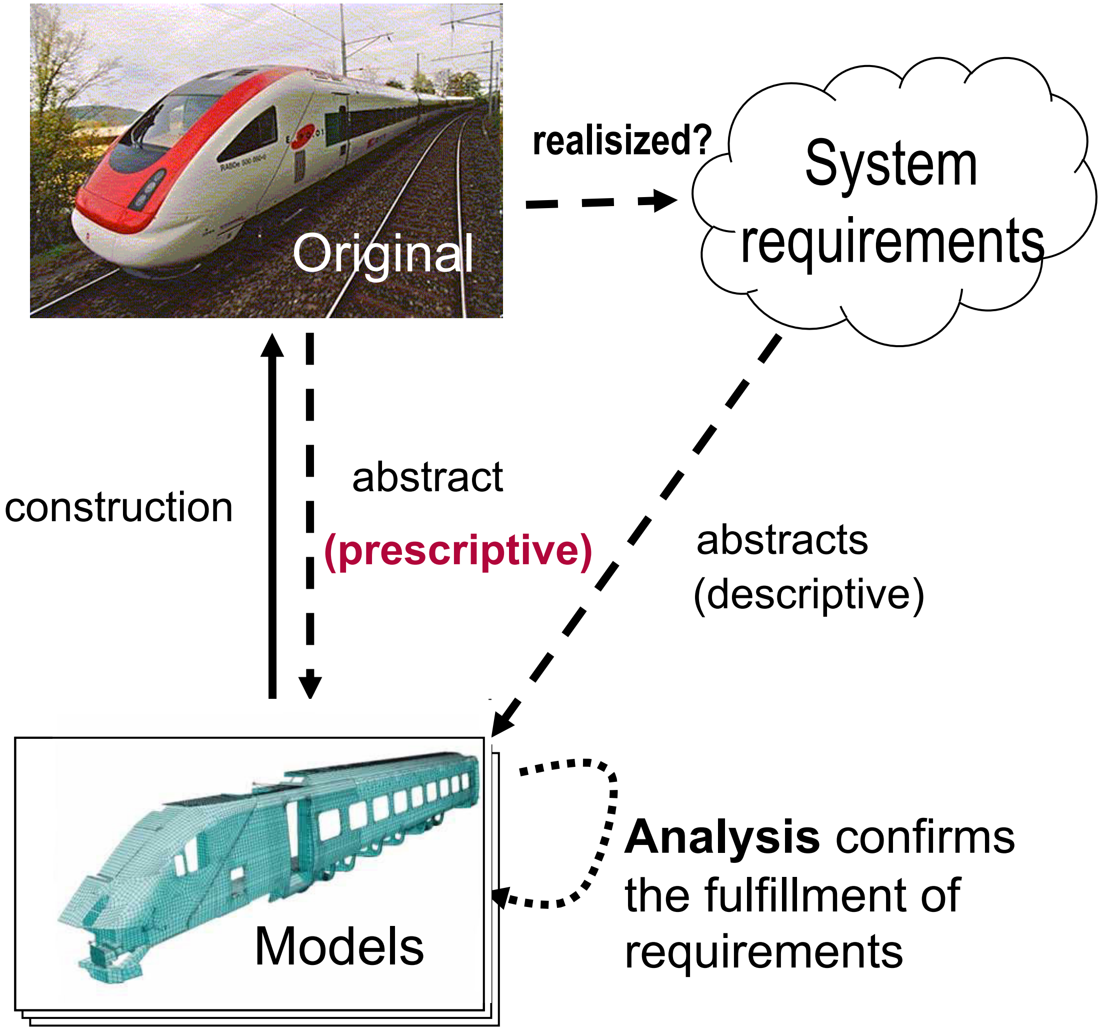
- "In the near future, software-intensive systems will exhibit **adaptive** and **anticipatory behavior**; they will process knowledge and not only data, and **change their structure dynamically**. Software-intensive systems will act as global computers **in highly dynamic environments** and will be based on and **integrated** with service-oriented and pervasive computing."
  M. Wirsing, ed., Report on the EU/NSF Strategic Workshop on Engineering Software-Intensive Systems *"Challenges, Visions and Research Issues for Software-Intensive Systems" at ICSE 2004.* Edinburgh, UK, May 2004.
- "The sheer scale of ULS systems will change everything. ULS systems will necessarily be **decentralized** in a variety of ways, developed and used by a wide variety of stakeholders with conflicting needs, **evolving continuously**, and constructed from heterogeneous parts.
- **Adaptation** is needed to compensate for changes in the mission requirements (…) and operating environments (..)
  Northrop, Linda, et al. Ultra-Large-Scale Systems: The Software Challenge of the Future. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2006.







The Basic Case: Engineering & Design Models



- **Question:** How do engineers develop complex systems?
- **Solution: design models**
  - used as representations for real or imaginary systems
  - Allow to try out alternatives
  - Allow reliable predictions

- **Characteristics of design models**
  - Complete coverage of the problem
  - Accurate representation
  - **Constant**