Autonomous Services and Workflows for Production Automation: *Managed Software Evolution*

Prof. Dr. Winfried Lamersdorf

Distributed Systems (VSYS)

Hamburg University, MIN-Faculty, Informatics Dept.
Vogt-Kölln-Straße 30, D-22527 HAMBURG, Germany

Winfried.Lamersdorf@informatik.uni-hamburg.de

http://vsis-www.informatik.uni-hamburg.de/vsys

Universität Hamburg

Examples from ongoing research as part of German Research Fund, Priority Programme (*Schwerpunktprogramm*) 1593:
„Design for Future: Managed Software Evolution“

LINKED EVOLUTION COMMUNITY FYPA2C

(Linked) Forever Young Production Automation with Active Components

Winfried Lamersdorf, Christopher Haubeck, Alexander Pokahr et al.
Distributed Systems and Information Systems
University of Hamburg, Germany

Universität Hamburg

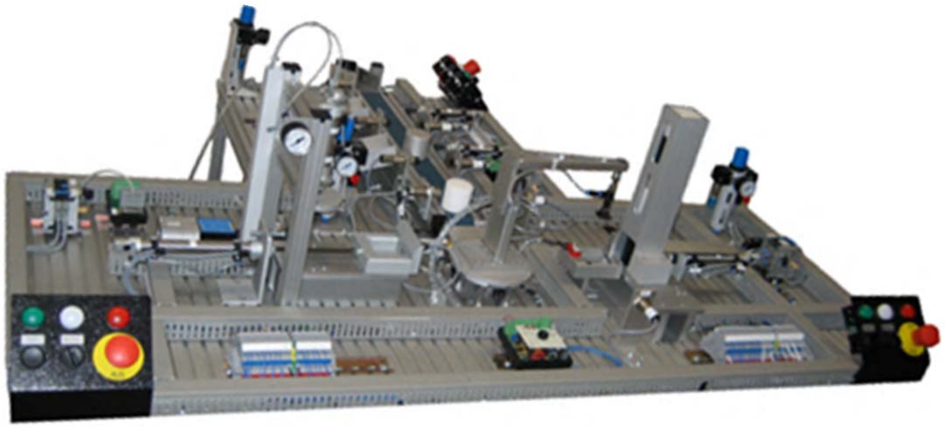
Alexander Fay, Jan Ladiges et al.
Automation Technology Institute
Helmut Schmidt University, Hamburg, Germany

HELMUT SCHMIDT UNIVERSITÄT
Universität der Bundeswehr Hamburg

Services & Workflows for *Production Automation*

Example Application Area: **“Production Automation”:**

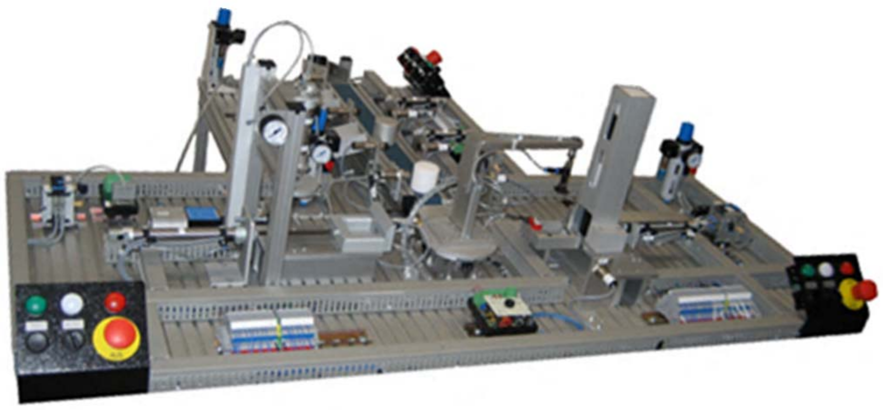
- Complex combination of hardware and software (“cyber-physical systems”)
- Hardware expensive and long-lasting
- Traditionally: little or only low-level software management
- Nowadays: increasingly software-driven (“Industry 4.0”)
- Software components represent (component) functionalities (“services”) & application (production) “workflows” (-> *“digital twin”*)



Example Application Scenario: *Production Automation*

Further characteristics of long-living applications such as “Production Automation”:

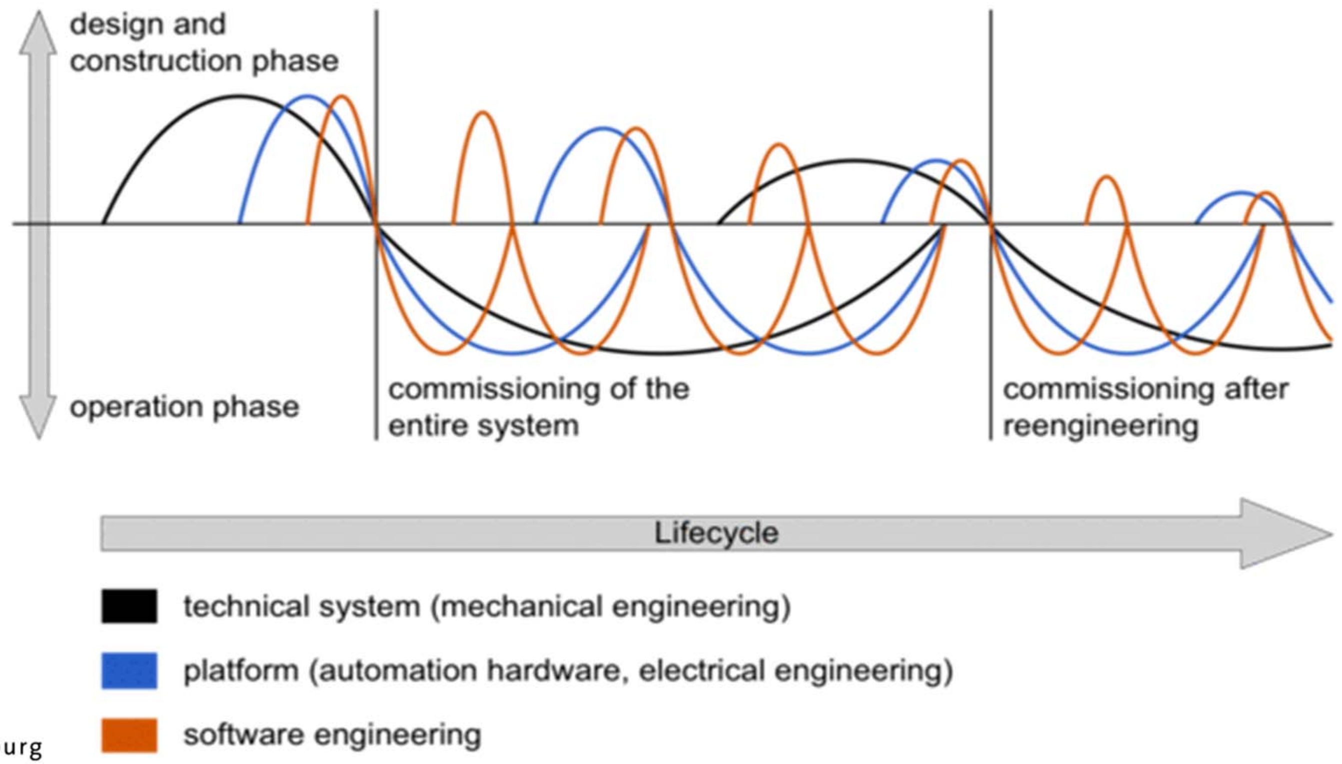
- Hardware changes occasionally (e.g. due to new/changed requirements)
- Software has to mirror that ASAP (often delayed/ done only partially/ forgotten…)
- **Question:** How to keep software – services as well as workflows – “in synch” with (changing) hardware components?
- **Goal:** coordinated development of hard- and software (system supported)
- Solution(?): “Autonomous” services & workflows which automatically “detect” such changes and then (also automatically?) “adapt” to them → **“Managed software evolution”**



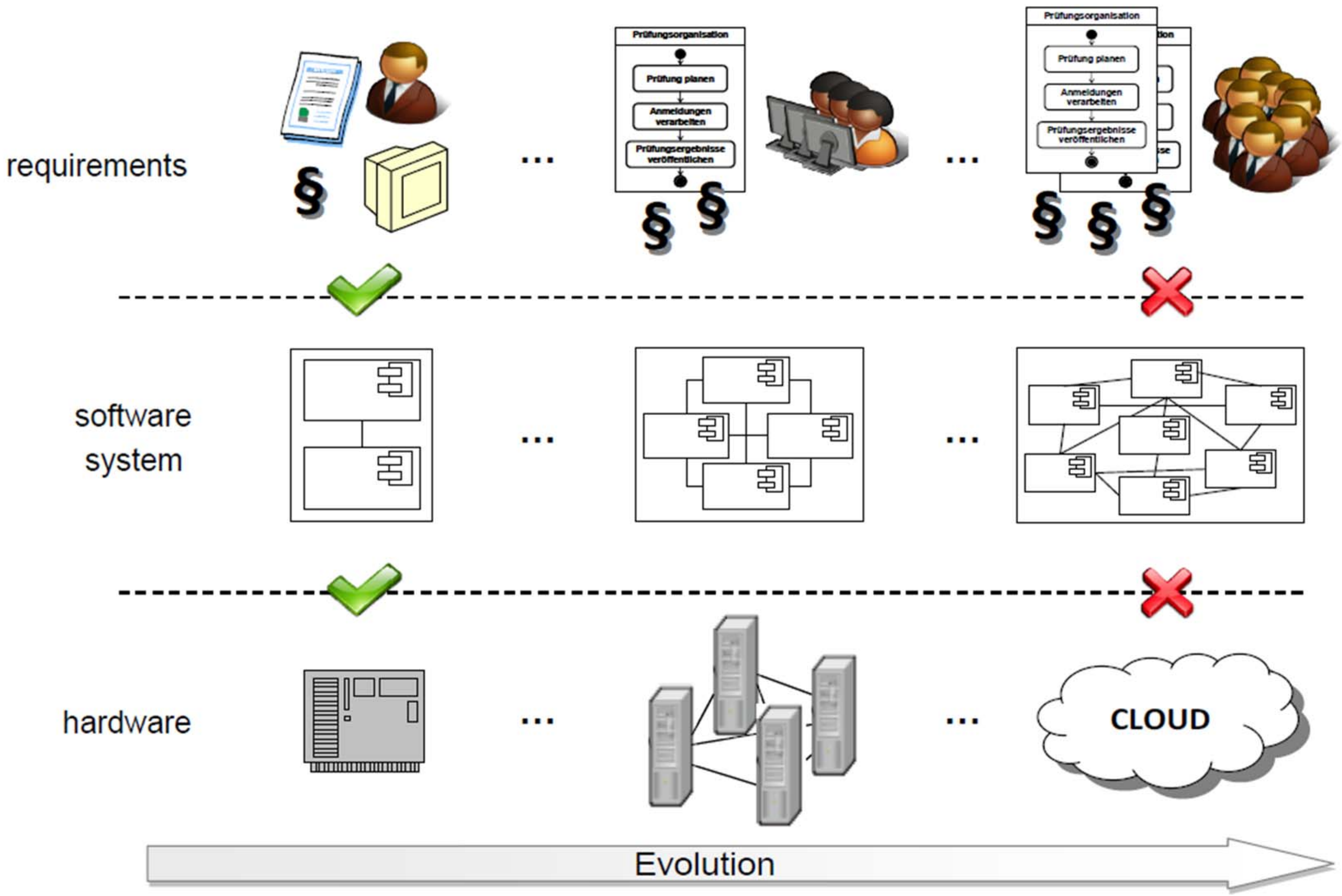
Motivation for “Managed Software Evolution”

Challenges during the whole lifecycle of software systems

- Legacy software
- New emerging technologies, and integration of new software, hardware and system components
- Adaption of software to new platforms and then continuous evolution of software systems with respect to continuously changing requirements
- Technical systems' evolution includes both *design and construction* phases as well as *operation* phases and involves different disciplines with different evolution process cycles

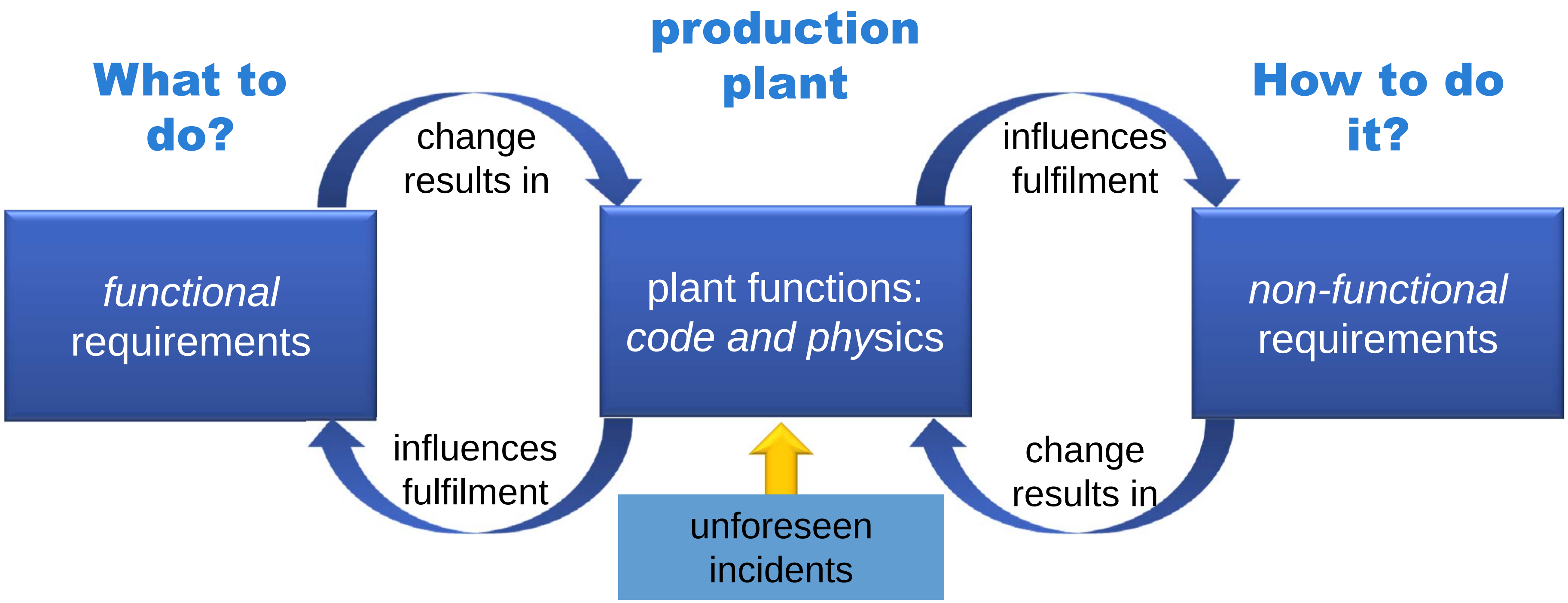


Different evolution of requirements, soft-, and hardware



[C. Momm, S. Sauer, GI AK L2S2 Presentation, 2009]

Interrelations of Changes and Requirements



Tasks:

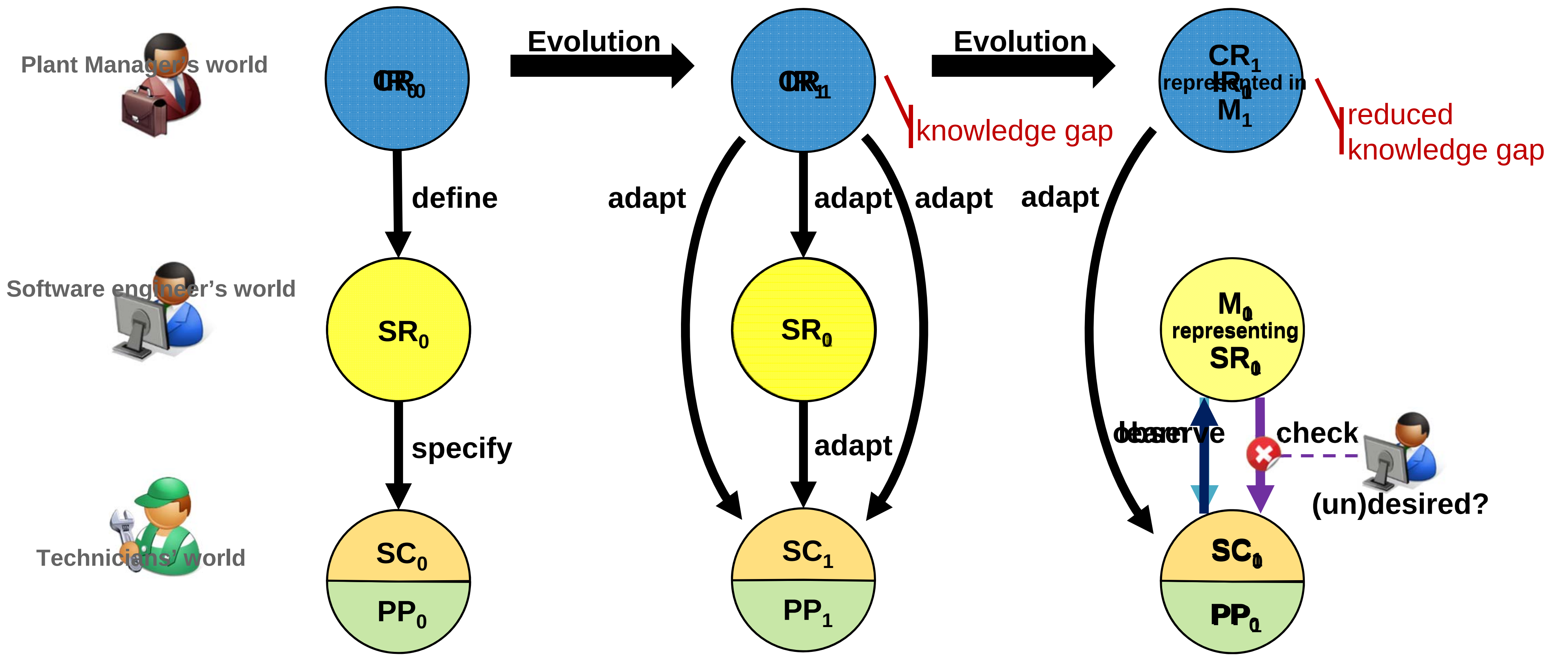
(1) detect changes within a plant and
(2) evaluate the (quality) influences of changes

„Forever Young Production Automation“

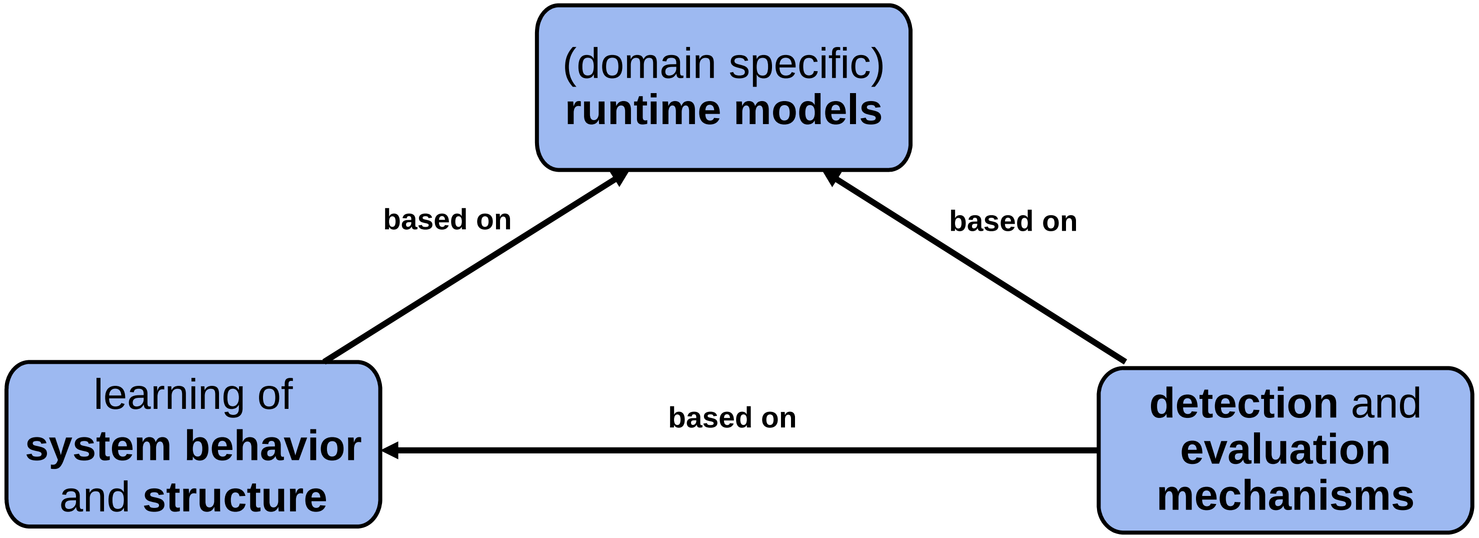
(*FYPA²C* – UHH 2013-2019)

IR_k – Informal Requirements
SR_k – Specified Requirements
CR_k – Covered Requirements
SC_k – Software Code
PP_k – Physical Plant
M_k – Model representation

at time k



FYPA²C: Keeping Pace with (Undocumented) Changes



Questions:

- How to **preserve available knowledge** during evolution?
- How can **evolutionary changes** be **detected and evaluated** at runtime?
- How to **construct** a **software that gathers knowledge** about a system **by observation**?

FYPA²C: Forever Young Production Automation with Active Components

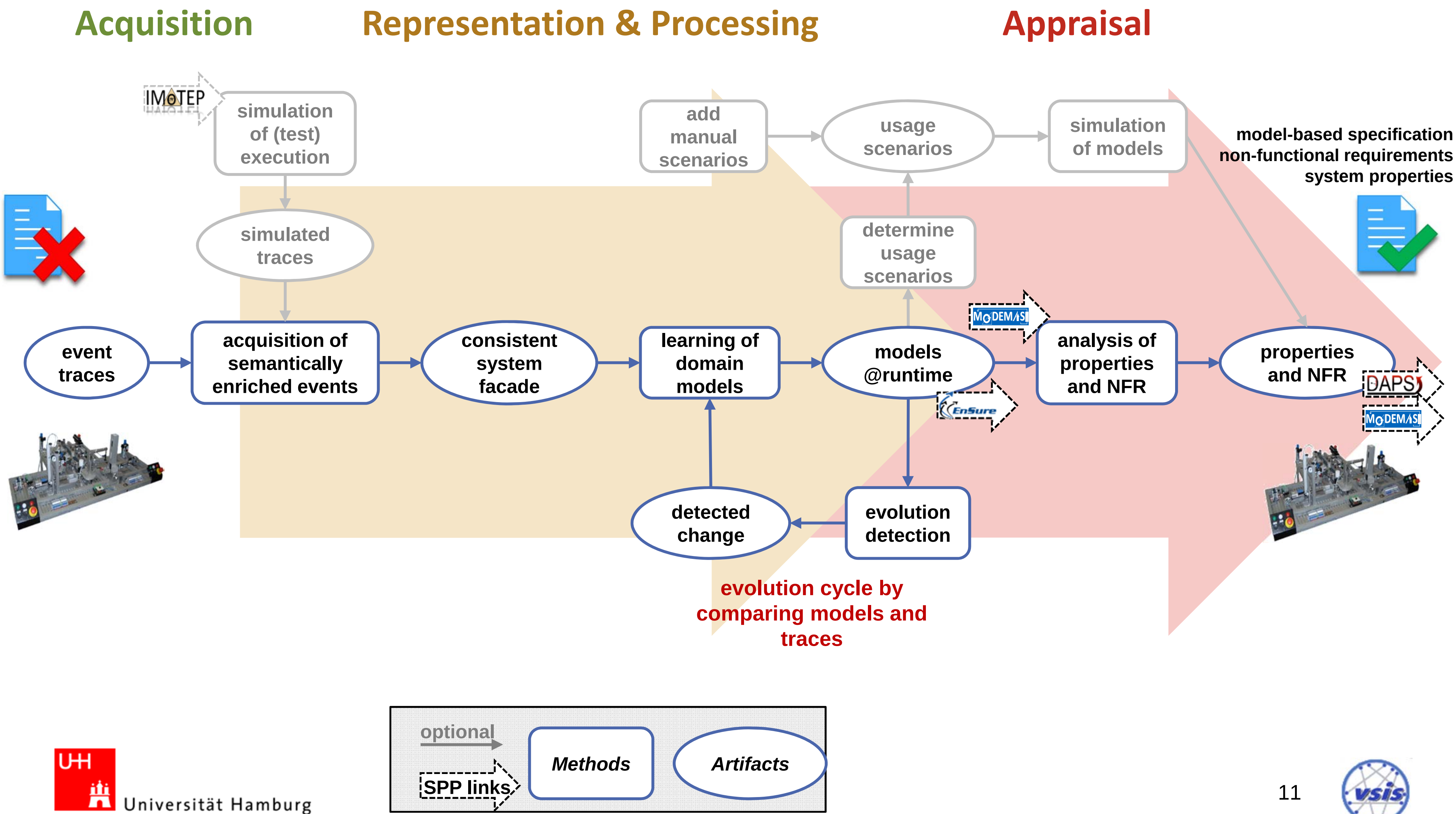
Goal: **Methods and processes for knowledge carrying software** (**KCS**) in order to counteract ***aging*** (i.e. undocumented changes) of evolving production systems

Questions:

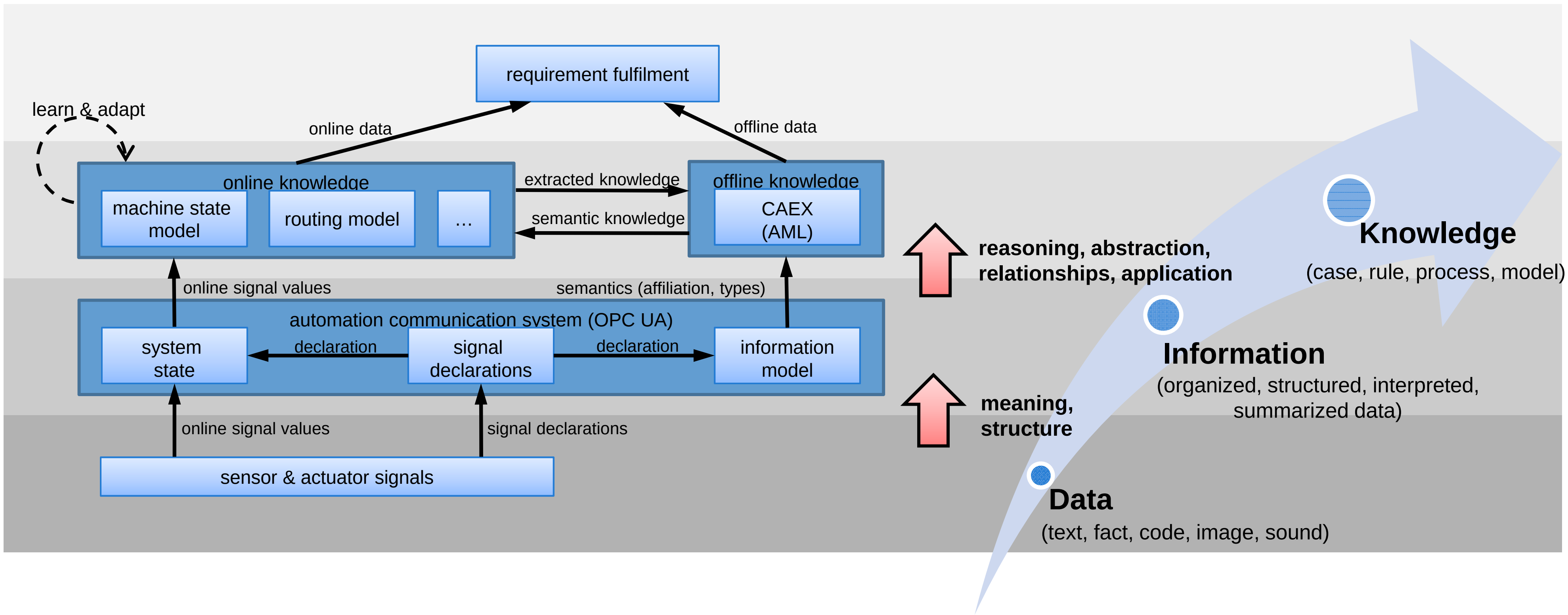
- How to **gain and preserve knowledge** about production processes by externally **available information** (i.e. I/O data)
  - without influencing execution
  - by establishing (formal) documentation that is constantly analyzable regarding typical non-functional requirements (**NFRs)**
- What is a **reasonable meta-model** and **software architecture** for **KCS** in an external monitoring context?
  - implementable mechanisms for the monitoring and analyzing processes
  - necessary components and services of an evolution support platform

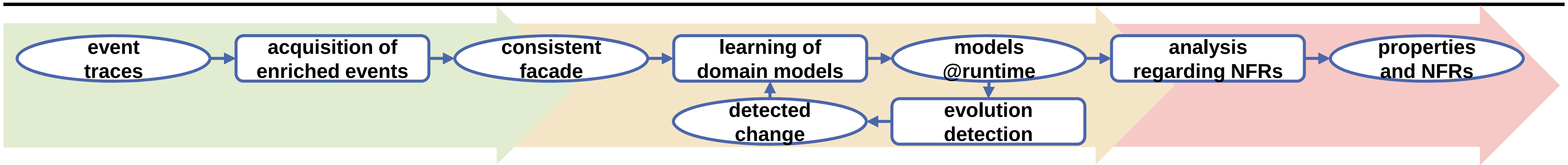
Approach: **Automatic generation** and scenario-based evaluation of **knowledge models** based on low-level signal (event) traces within an **Active Component architecture**.

Evolution Support Methods (overview)

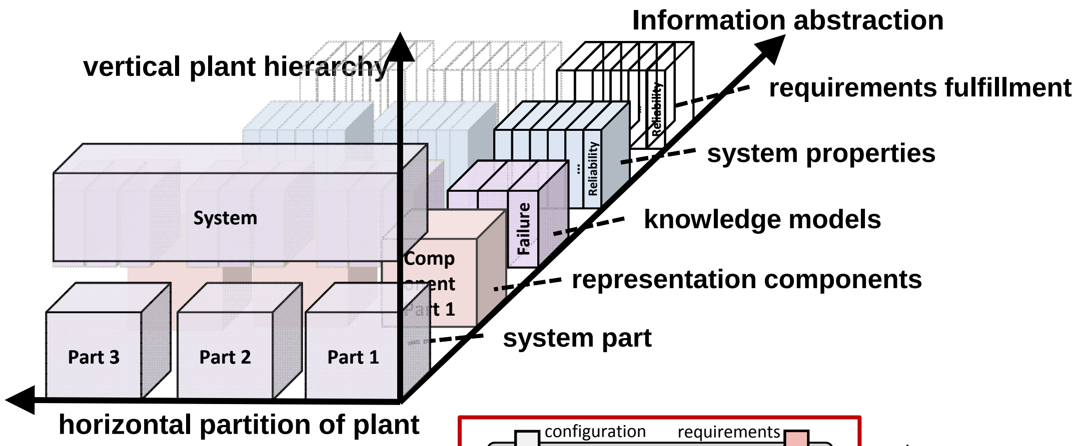


Overall Results – Knowledge Concept

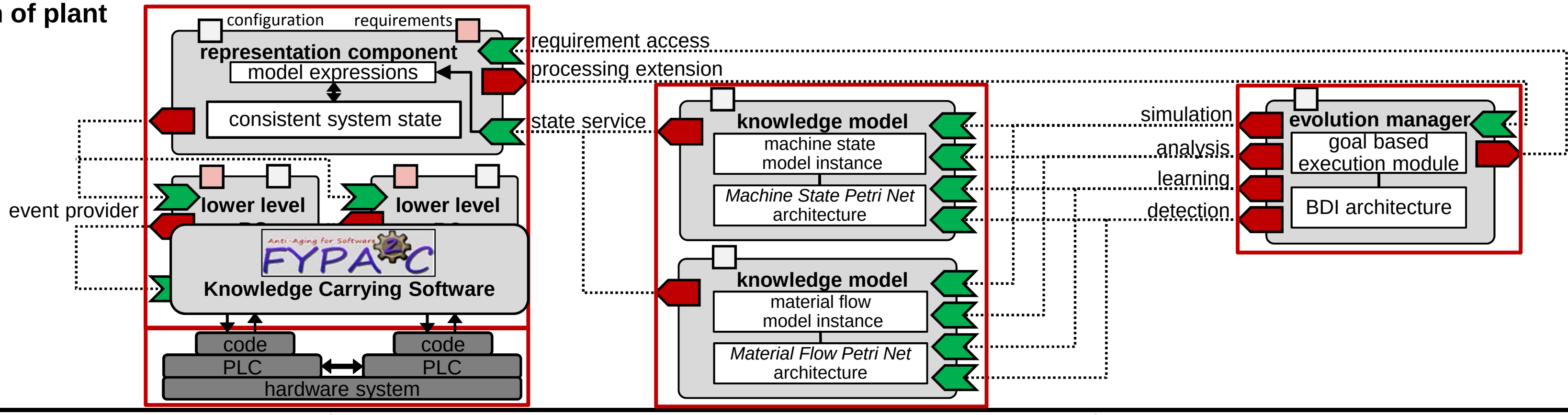


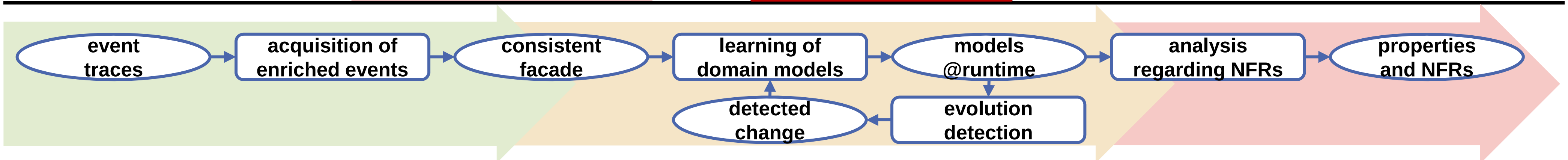


Overall Results – Knowledge Carrying Software (KCS)



- Knowledge information levels
- External KCS that uses monito-ring to support evolution
- Implemented in ***active compo-nent*** runtime artefacts





Question: How to represent theses components???

(In order to answer that…) Short excursus: Choices of *Software Development Paradigms*

Problem: No coherent overall view of application problems

- Many single separate problems
- Many single separate solutions

→Overall Approach needed!

- Consistent and intuitive concepts
- Adequate Abstraction level for distributed systems
- Close to real-world concepts
- Success factor: small delta to established paradigms

→***"Software paradigm":***

Fundamental principle for describing and implementing software systems

The Concept of a *Software Paradigm*

Software paradigms…

- determine concepts for the description and realization of software systems
- define the level of abstraction for the description („World Model“)
- support/hinder specific architectures
- lead to increasingly abstract concepts

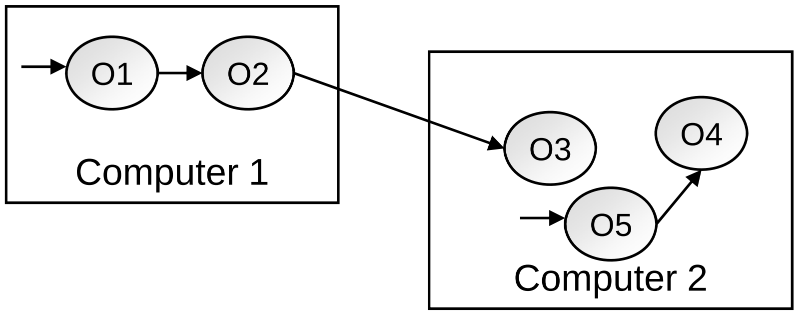
- *Historic examples* for the development of program paradigms – from imperative to object-oriented programming
  - imperative: program as linear sequence of commands
  - object-oriented: concealing data and methods to classes/objects

  - Conceptional background:
    - *imperative*: von-Neumann computer
    - *object-oriented*: real world of items and objects

Object-oriented Paradigm

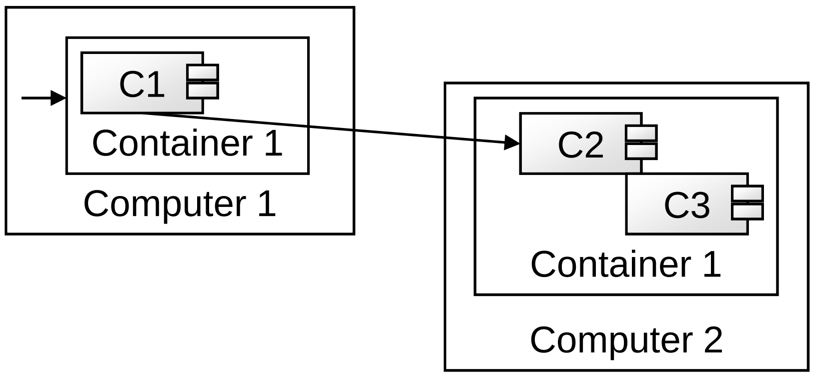
- Objects as units for data and behavior
- Based on method-oriented communication and client/server model
- Client/server are objects of any granularity
- Object identities allow for system-wide identification of clients/server
- Migration of objects allows for transparent runtime adaptation of application configuration

- Problems: Re-usability of objects low, based on no separation of complementary concepts (as, e.g., persistence- or security aspects)
- Examples: DCE, CORBA



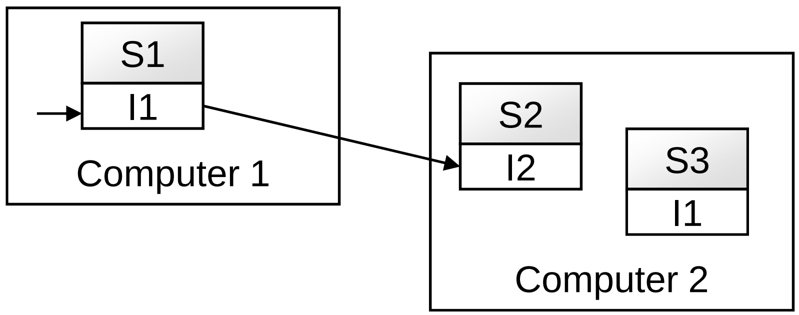
Component-oriented **Paradigm**

- Generalization of object-oriented paradigms
- Components are coarse-grained units on application level with clear interfaces
- Components are self-contained, resp. have well-defined dependencies
- Idea: Component repositories for clear composition of software from predefined components
- In general restricted to application logic, separate from application context, i.e. full configuration not before deployment (security, transactions, persistence, …)
- Unified deployment model
- Execution in specific “containers”
- Examples: Enterprise JavaBeans, .NET Components



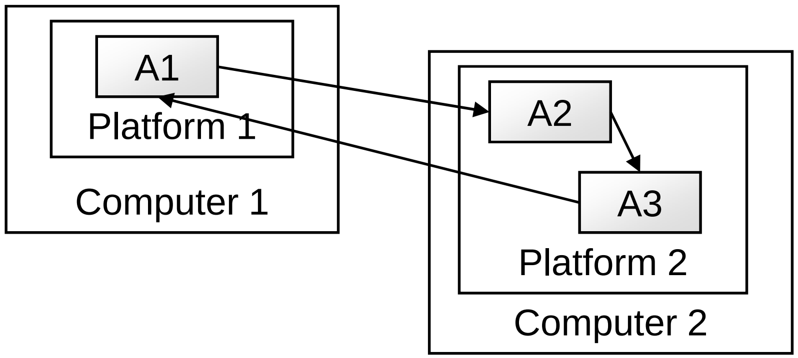
Service-oriented Paradigm

- SOA – Service Oriented Architecture
- Based on process-oriented view of application services
- Services are coarse-grained units of software systems, loosely coupled with business processes/workflows – can be integrated by means of
  - Orchestration
  - Choreography
- Have well-defined interfaces
- Could be used either synchronously or asynchronously
- Interoperability by use of standards (technology independent)
- Examples: Web Services (WSDL, SOAP, UDDI)

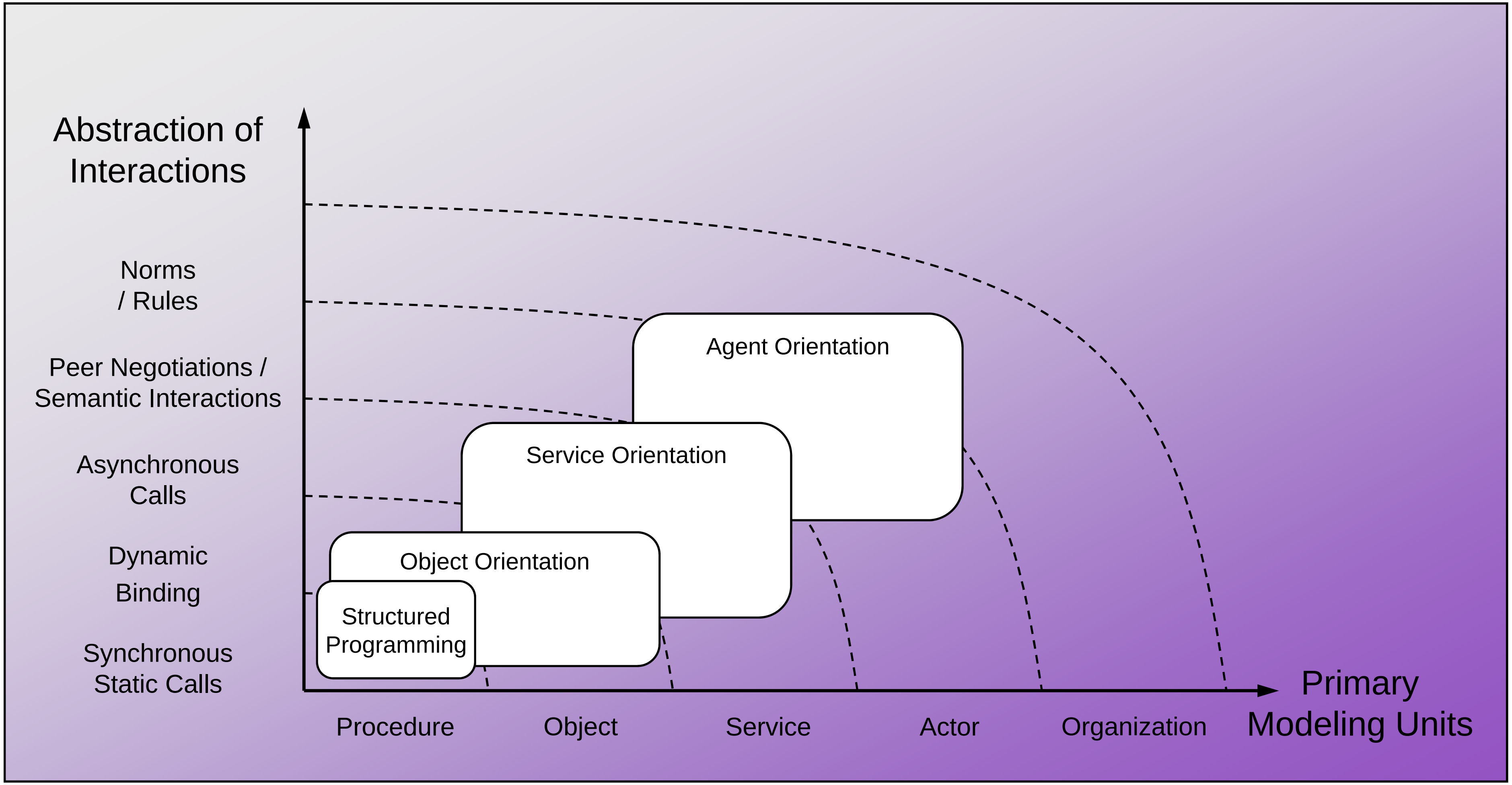


Agent-based Paradigm

- A System is viewed as a composition of independent actors (agents - i.e. multi-agent system)
- Communication is always asynchronous (message-based)
- Basic concept „agent“ as unit – in well-defined context – which uses sensors as well as effectors
- Agents make decisions autonomously, based on context as well as interpre-ting messages
- Behavioral specification of agents via internal architecture
- Behavioral specification of a multi-agent system via coordination of single agents („social architecture“)
- Examples: agent platforms as, e.g., JACK, JADE, Jadex



Categorization of Software Paradigms



MAS Application Areas

Sector / Class	*Industrial Applications* (A)	*Commercial Applications* (B)	*Entertainment Applications* (C)	*Medical Applications* (D)	*Military Applications* (E)	…
Multi-Agent Simulation (1)	Factory simulations	Market / trading simulations	Movie scene Productions / Games	Hospital simulations	Battlefield Simulations / Pilot training	…
Problem Solving (2)	Goods transport	E-Business	Strategy games	Hospital logistics	War logistics	…
Robot Control (3)	Production robots	Household robots	"Intelligent" toys	Medical device control	Unmanned aerial vehicles	…
Information Management (4)	Tracking and Tracing	Web search Email filtering	Artificial game reporters	Disaster management / Medical information management	Decision support / Smart dust	…
Human Computer Interface Mgmnt. (5)	Augmented reality tools	Shop bots / Help assistants	Avatars in games	Telemedicine / Home care management	Augmented reality tools for soldiers	…
…	…	…	…	…	…	…

Industry / Production

- Increase efficiency of production process
  - Flexibility when changing initial parameters (resources, characteristics of end product)
  - Application areas:
    - Whole production processes
    - Support for employees for single production steps
    - Production robots
    - Workflow simulation

- Examples:
  - CAARS – Project
    - Used for car production (BMW, GM)
    - HMDs (Head Mounted Displays) support employees
    - Mobile Augmented Reality System (MARS) → additional information to real world
    - Used for training
    - Agents support flow of information
    - Joint project of Juxtopia (HMDs) and Georgetown University, USA
  - Jadex application for Daimler production plant
    - agent-based simulation of (new) production processes
    - control of production steps and interrelationships
    - DFG transfer project with UHH

Industry / Transport Logistics

- Planning and executing logistic processes
- Optimal use of transport facilities
- Time management (e.g. for delivery)
- Many sub-systems are coordinated (e.g., storage, transporter, etc.)

Examples:

- Open ID Center
  - Fraunhofer Institut for Material and Logistics (IML)
  - System for delivery of products in real-time without manual intervention
  - Product identification via RFID (Radio Frequence Identification) labels
  - Shuttles controlled by agents fetch tasks for delivery (e.g. using an agent with minimal distance to store)
- Project AgentFly
  - Agent technology center (Czech University), http://exile.felk.cvut.cz/
  - Based on agent platform A-Globe
  - Simulation of autonomous control of airplanes
  - Real experiments with UAVs
- Jadex (agent)-based simulation and optimization of hospital logistics
  - Developed by UHH as part of DG SPP on agent use in logistics
  - Including evaluation against centralized approaches

Industry / Transport Logistics

Application Lab IML Dortmund

Commercial Applications / Electronic Assistents

- Support for humans for
  - Data management
  - Support for software problems
  - Mediation between information producers and consumers
  - Information search in the Internet
  - Home tasks

- Example: Kärcher RC 3000 Robo Cleaner
  - Robot
  - Autonomous cleaning
  - Uses sensors

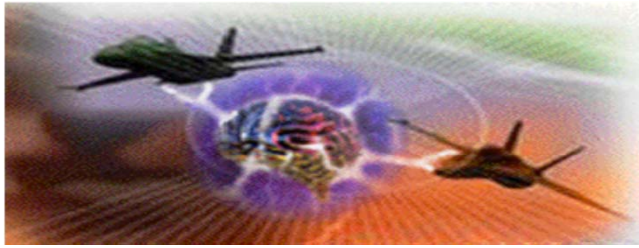
Computer Games

- Important for games: most realistic virtual world → fun!
- Reality view achieved by mirroring physical laws
- intelligent Non Player Characters (NPCs)
- Use of different AI approaches
  - Action: tactic capabilities of NPCs
  - Role play: Interaction with NPCs ...

- Examples:
  - *QUAKE*
    - Action play
    - Behavior of NPCs realized with BDI Agents
    - Quake Engine Basis for many Action Shooter

  - *AgentKeeper* (http://code.google.com/p/jadex-agentkeeper/)
    - Clone and extension of Dungeon Keeper
    - First development in teaching course at VSIS/UHH
    - Based on Jadex platform
    - Further developed as OpenSource project (P. Willuweit)

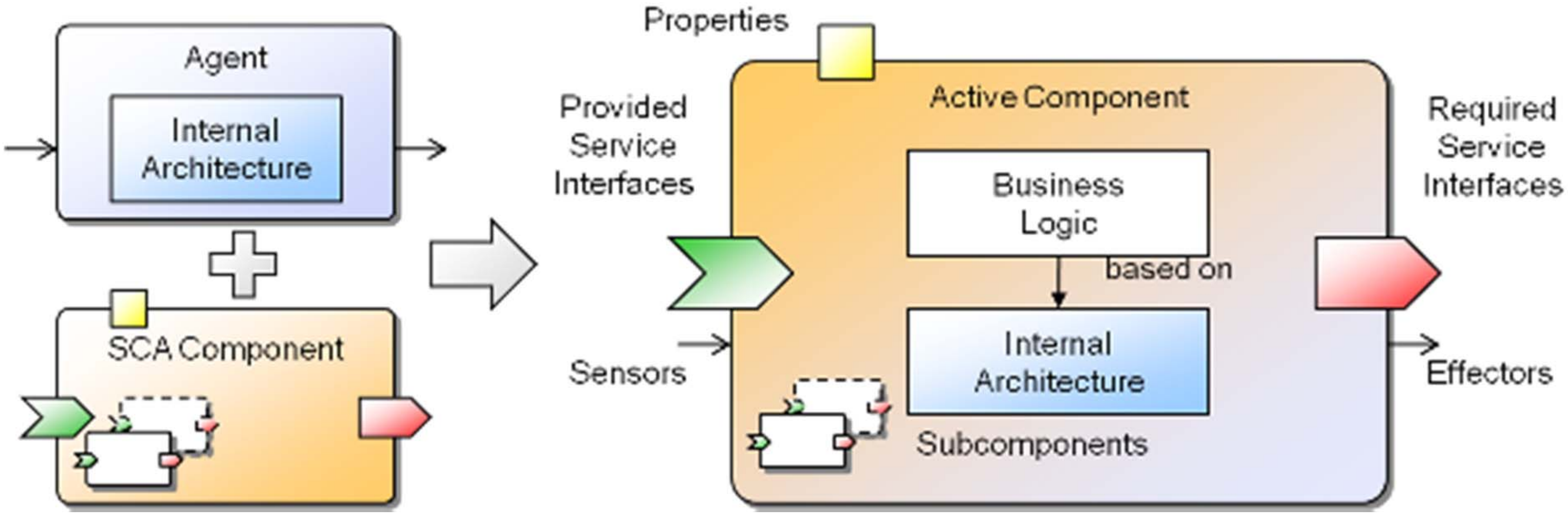
Military / Simulation



- Complex battle field simulations
  - Simulation of errors of dedicated (enemy) intervention
  - What determines a successful crossing of a (e.g. urban) area (e.g. number of forces, arms, etc.)
- Training simulation for soldiers
  - Intelligent agents play roles of other (ground or air) forces/units
- E.g. TacAirSoar (http://www.soartech.com/)
  - Training software for pilots of air forces
  - Computer-generated forces (CGFs)
  - Realized in Soar
    - Environment realized in if-the-else-rules
    - Airplanes realized with some 5000 rules
    - 15 alternative air plane types
  - First Simulation 1997 on 25 Pentium PCs
    - 722 Missions in 48 hours
    - Average time of missions: 3 hours
    - 30 – 80 flights per Mission
    - only 5% software problems

New Combined Approach to Software Development: "Active Components" (AC):

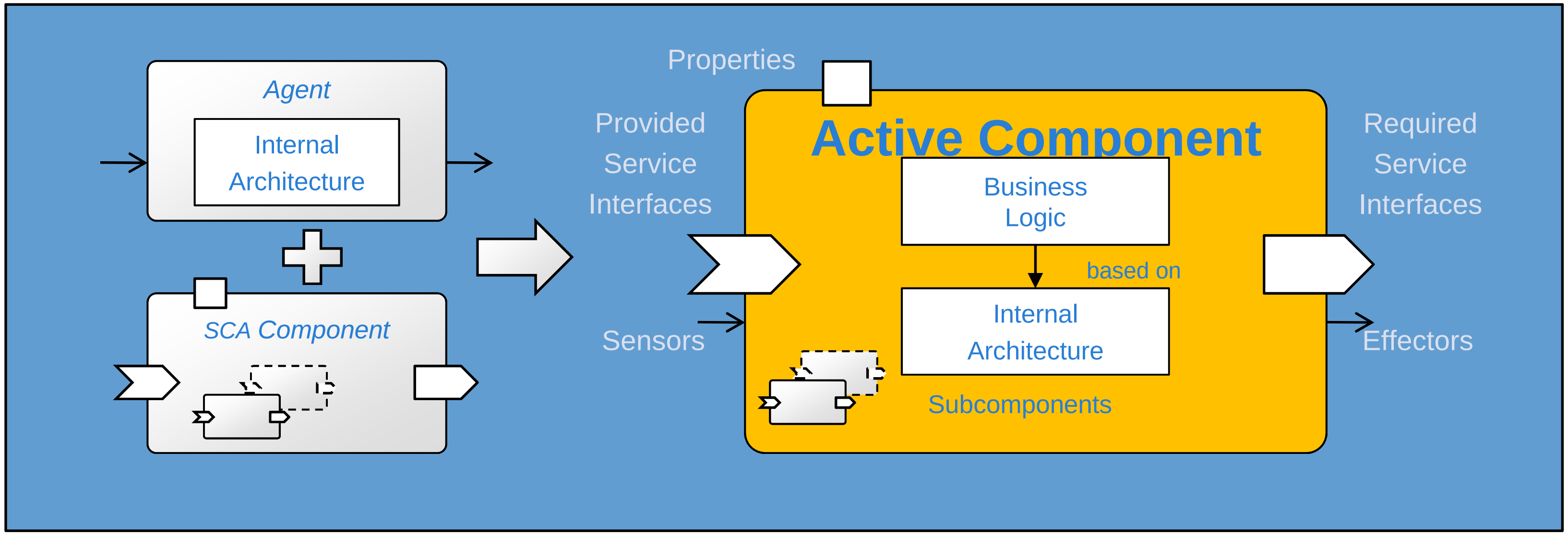
… based on *autonomous*, *adaptive*, *knowledge-carrying* software *components* using both software-engineering and agent technology



Active Components (AC):

Definition: *"An **active component** (AC) is an autonomous, managed and potentially hierarchical software entity that is capable of interacting with other active components in different modes including message passing and method calls."*

- management infrastructure and composability of components
- invocation styles like agents and services/active objects
- rich behavior styles like agent architectures or workflows

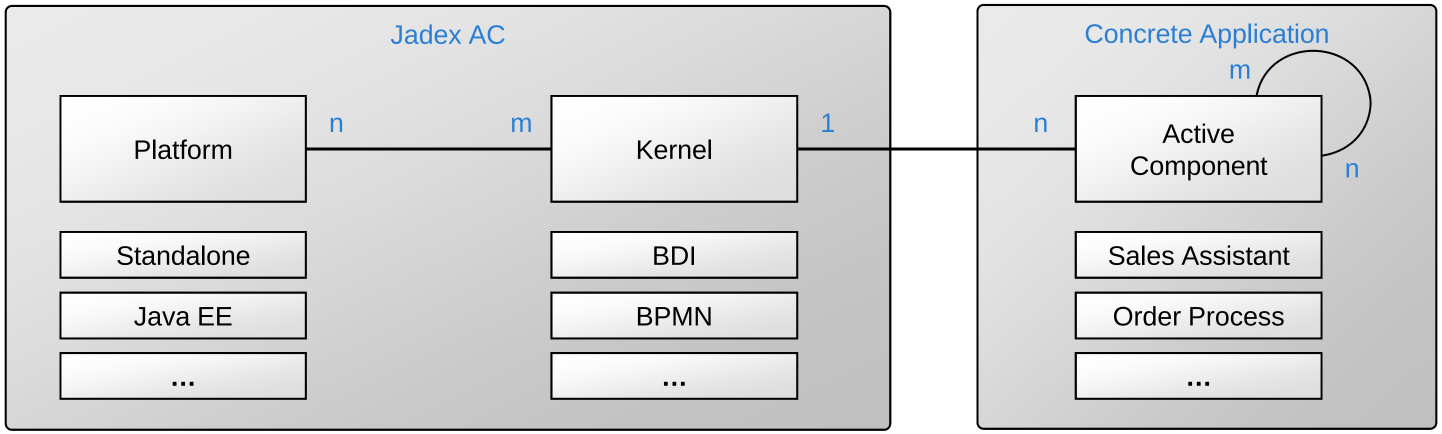


AC Implementation Platform: „Jadex v2"

*A **platform** is the* management infrastructure for components, *which is responsible for their execution as well as for providing administration capabilities like a messaging system or a component service registry.*

Design Goals:

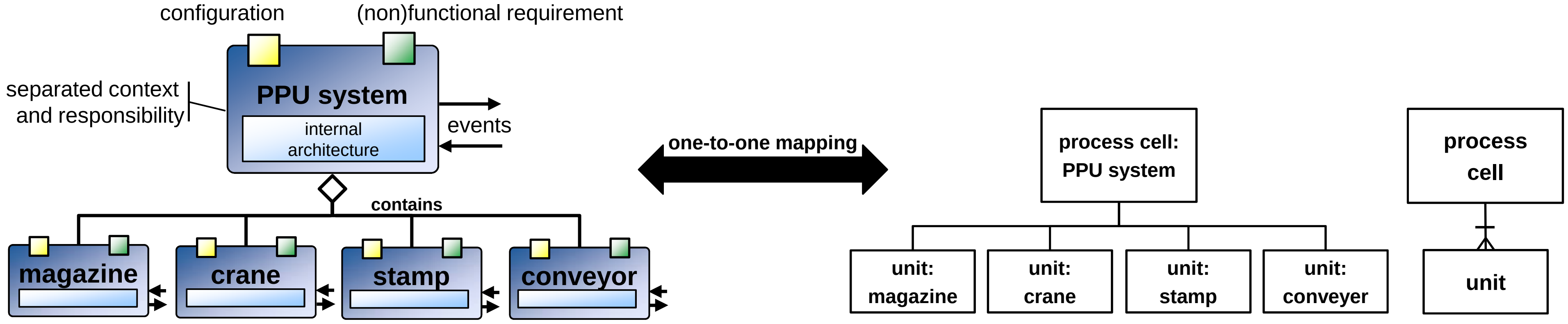
- Platform can execute different kinds of ("active") components
- Component kernels should be enabled to run on different platforms
- Applications should be platform independent
- Applications should be composable from arbitrary component types (heterogeneous applications)



Back to the application example: Knowledge Carrying Software for Production Automation Systems

Expected characteristics of a knowledge carrying software (KCS) for production systems

- Direct mapping between KCS and the physical plant hierarchy
- Reaction to plant events along its original responsibility chain
- Encapsulation of local knowledge in a separated processing context
- Enrichment of (non-) functional requirements at each granularity level
- Autonomous management of requirement verification

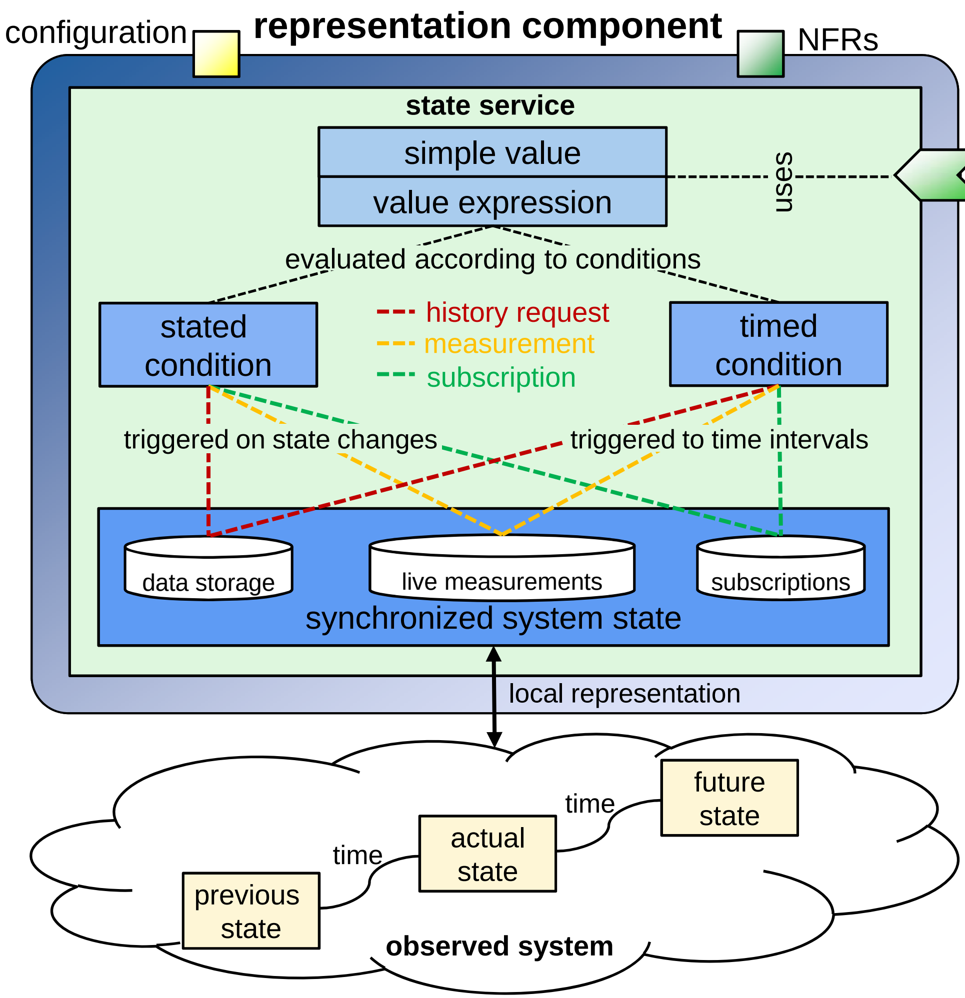


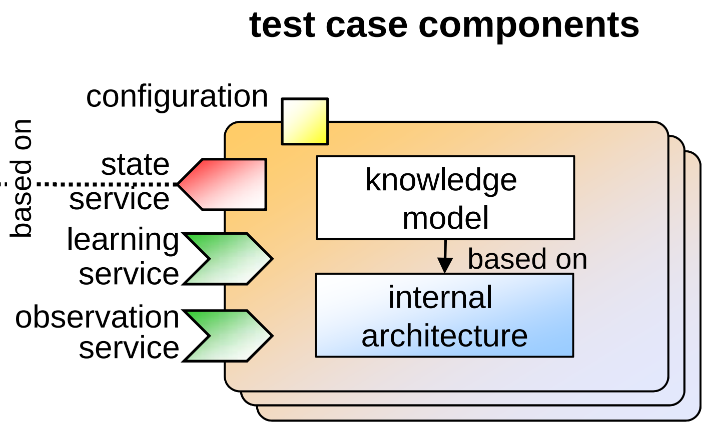
Representation components of KCS

Physical plant hierarchy according to IEC 61512

Universität Hamburg

Generic Management of Knowledge within a KCS

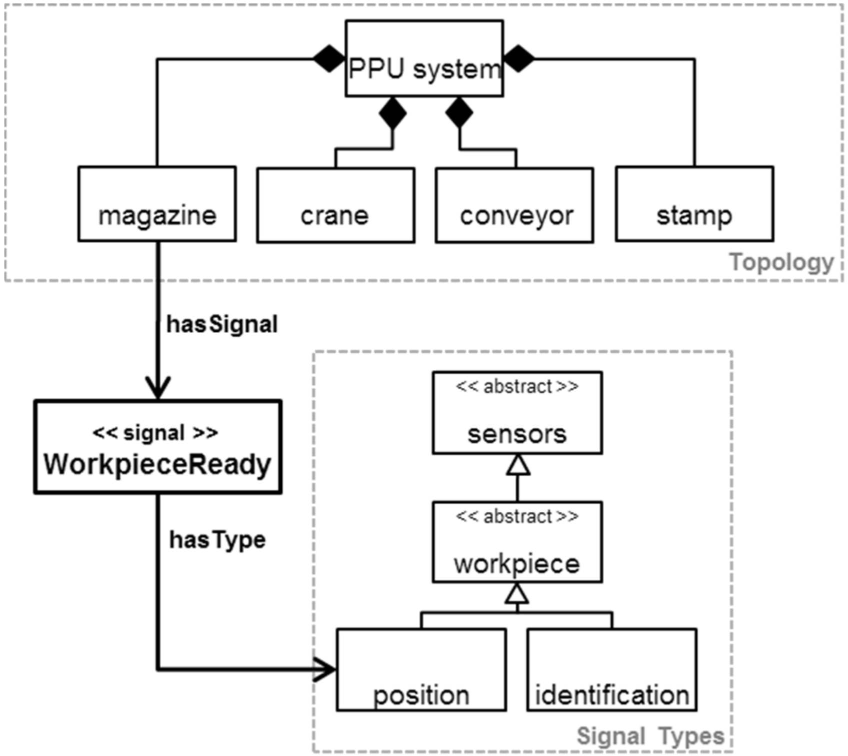




Execution of knowledge containing models

- Synchronization and consistency of the observed (partial) system via a component state
- Management of local and domain-specific knowledge in model-based test case components
- Knowledge modification methods incorporated in services allow calculations of NFRs
- Runtime dependence due to simple values or complex expressions according to stated or timed condition
- Executable on current values (by live measurement), history values (by data storage) and future values (by subscriptions)

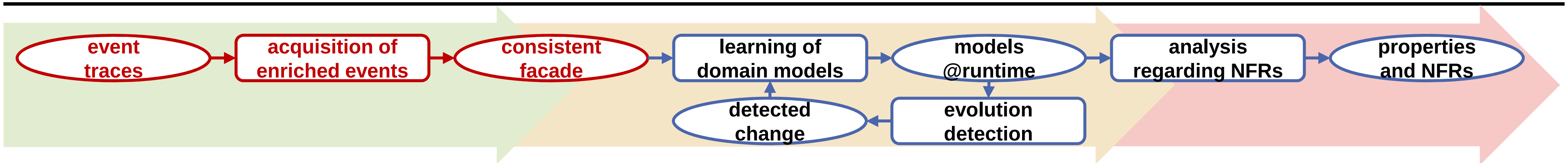
Knowledge Management for Production Automation: *Acquisition and Representation of Enriched Plant Data*



- Rule based selection of signals for model learning and analysis

Workpiece (WP) routing signals:
1: [affiliation = *ppu.crane*, type = *sensor.workpiece.position*]
2: [affiliation = *ppu.crane*, type = *sensor.machine.position*] AND
[affiliation = *ppu.crane*, type = *sensor.workpiece.WPInteract.hold*] CH

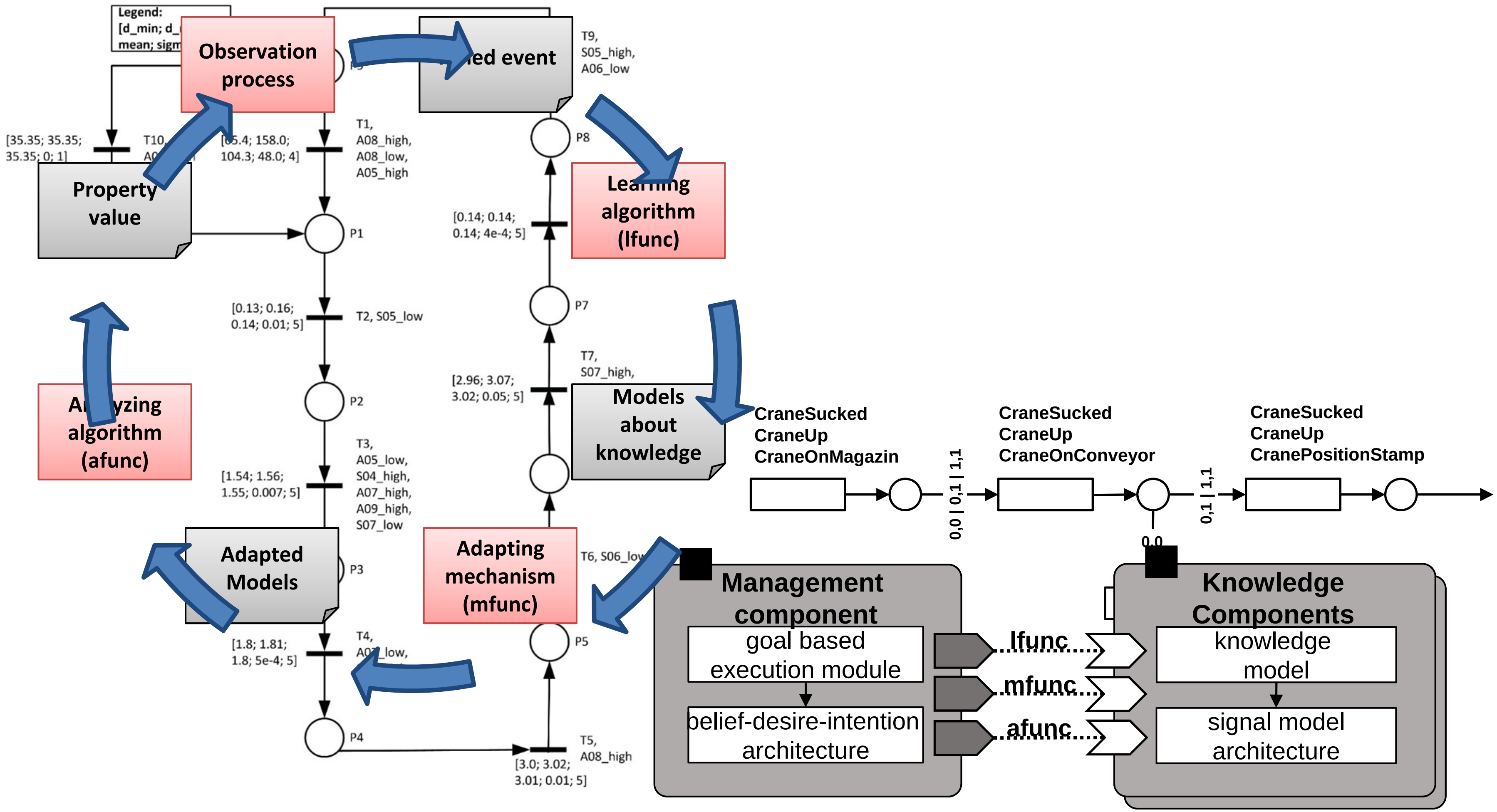
Workpiece (WP) identification signals:
3: [Affiliation = ppu.*crane AND* type = *workpiece.identification]*



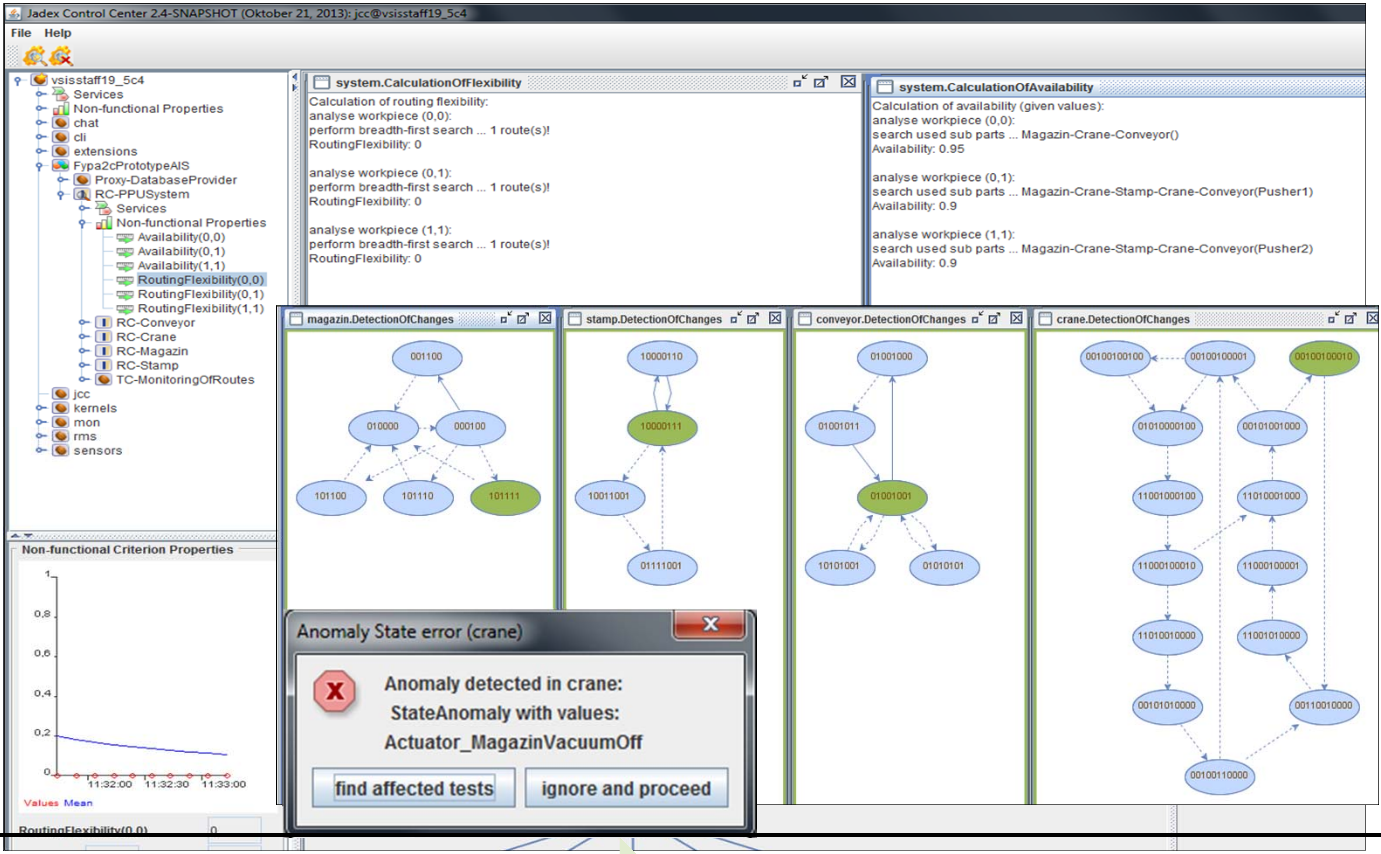
CH1 Hier noch gleiche Farbe wie oben bei AC

Christopher Haubeck; 23.02.2015

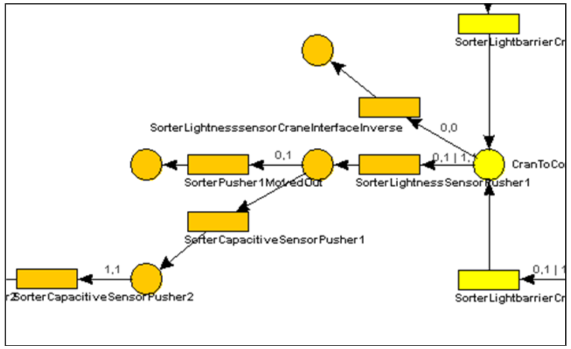
Evolution Support for Executable Specifications in Service Components

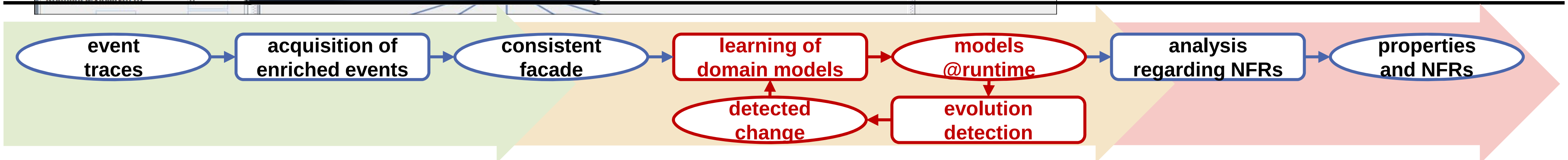


Model-based Specification at Runtime

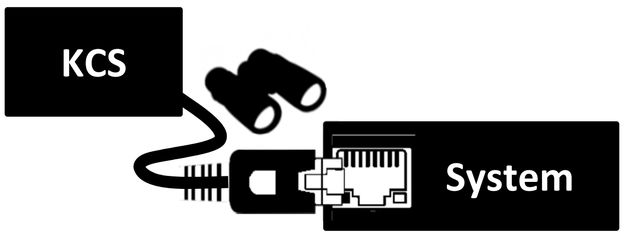


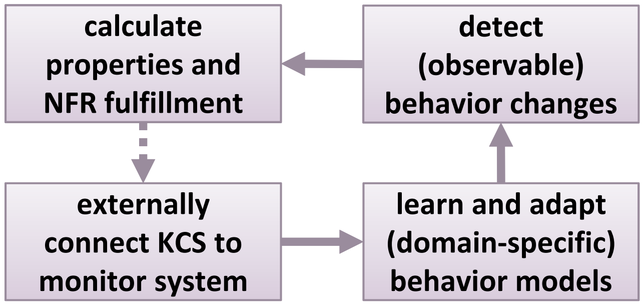
- Evolution detection
- First learning of knowledge models
- First calculation of
  - Routing Flexibility
  - Availability



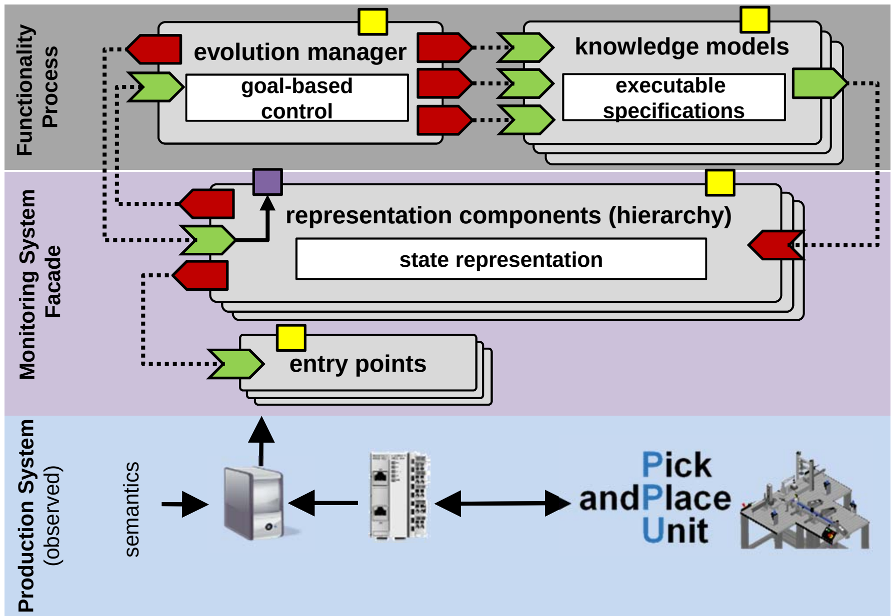


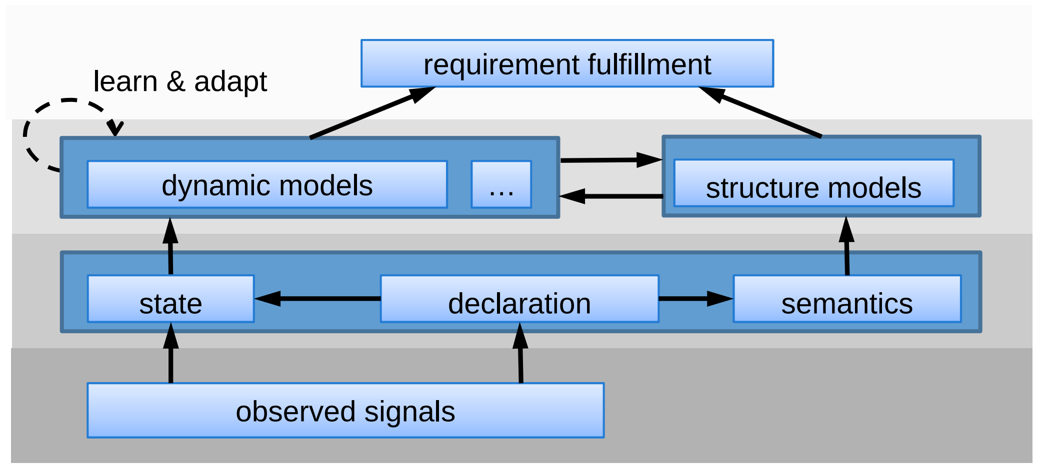
Summary: FYPA²C in a Nutshell – General Approach





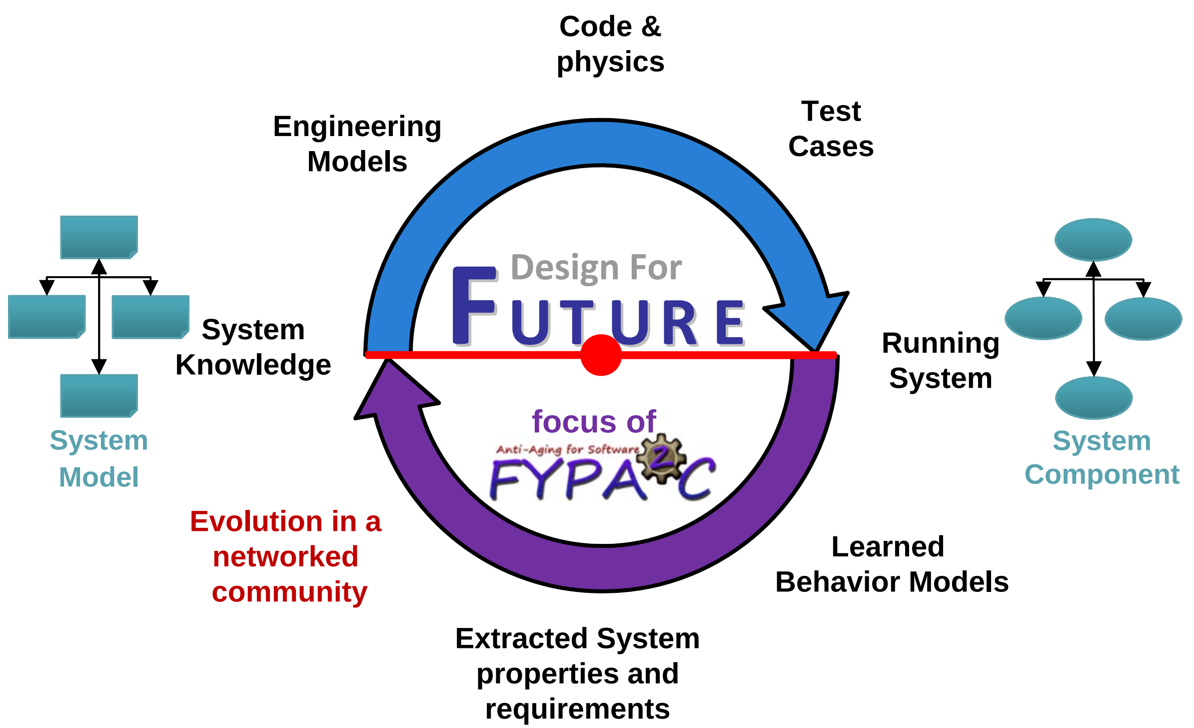
- Minimal setup costs for automated evolution support
- Generalizable concept of knowledge artefacts and evolution-handling KCS by exchanging model types





Future Work: Supporting Evolution in a Networked Community

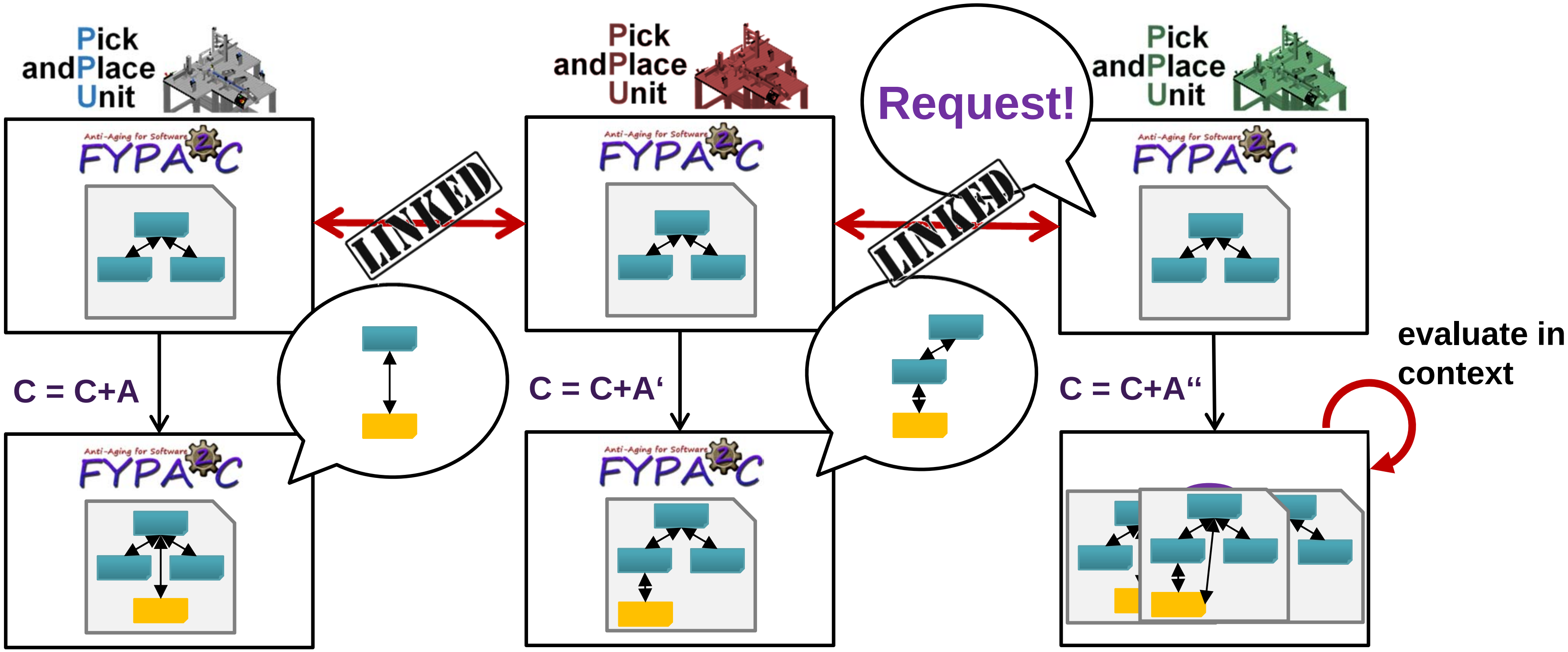
- Connecting systems in an evolution-aware network platform
- Co-evolution of system and model (also by integrating further SPP results)



Motivation and Idea – LinkedFYPA²C

FYPA²C allows collecting "experiences" with evolution steps

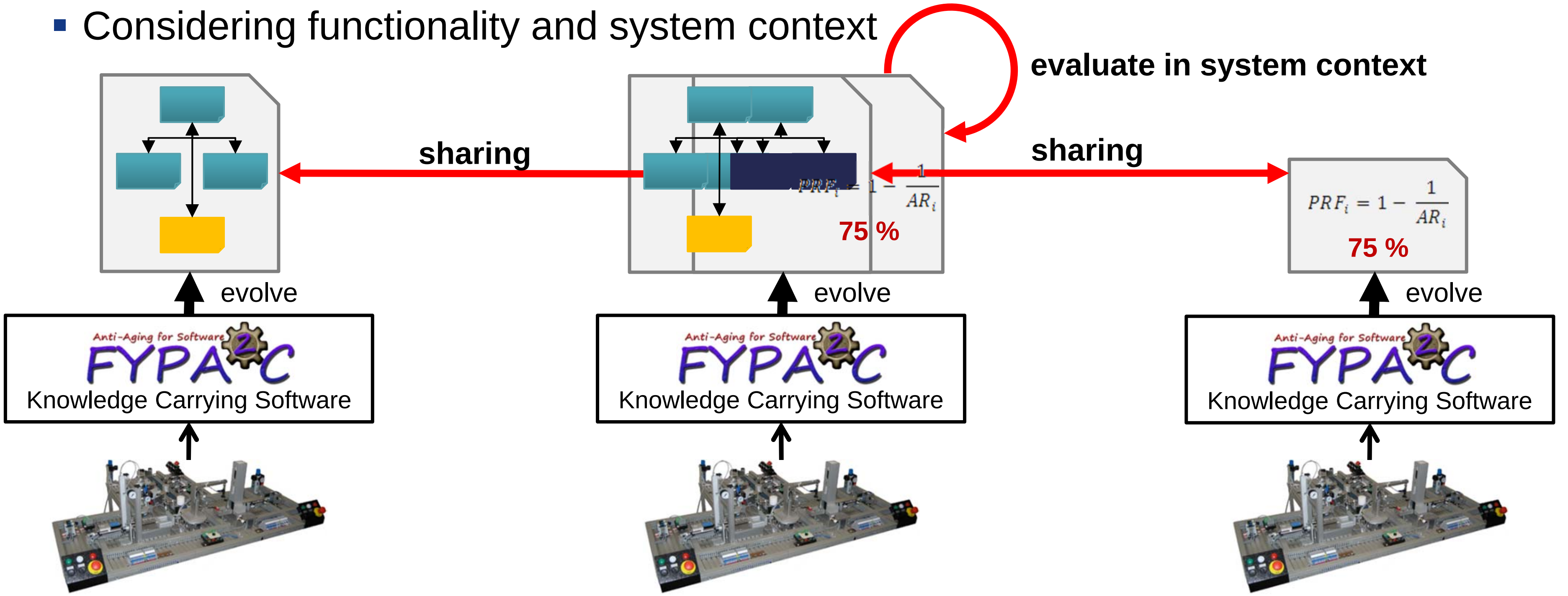
- Experiences could be used to actively support evolution of similar systems
- LinkedFYPA²C connects systems to a knowledge carrying evolution community



Future Work – Supporting Evolution in a Networked Community

Connecting systems in an evolution-aware network platform

- As part of co-evolution of systems and models (also by integrating further SPP results)
- Sharing evolution solutions by means of models and their differences
- Detecting suitability of evolution solution for a single connected system
  - Considering functionality and system context



plant picture courtesy of
© AIS TU München

Future Project Goals and Research Questions

- **Machine interpretable base of evolution-relevant knowledge**
  - How can evolution relevant “experiences” be captured in artefacts?
  - How to capture the system context?
  - How can evolution steps and their influence be related to the system context?
- **Evolution assessment methods**
  - Under which conditions is an evolution step applicable on another system?
  - How can similarity between systems be measured and expressed?
  - Can evolution steps of several systems be aggregated and tailored for a specific system?
- **Proactive evolution support method**
  - Can “evolution trends” be recognized in a cooperative evolution network?
  - Which methods are needed for a proactive evolution recommendation?
- **Networked evolution support platform for knowledge exchange**
  - What is a suitable middleware to realize such an evolution support network?

Conclusion and Summary

Services and Workflows are needed for a variety of applications – preferably in an appropriate *Software Engineering Context*

Example application area here: *Production Automation*

Application Characteristic:

- **Cyber-physical** systems (hard- and software combined)
- (Often undocumented) **changes** – independently w.r.t requirements, hard- and software
- Automation support needed for continuous **hard- & software evolution**

***Implementation: Agent-/component-oriented* software development platform (*“Jadex”* => *“Active Components”*)**

- Implementation platform for autonomous *services* (with both functional and non-functional characteristics) as well as (embedded or separate) *work-flows*
- Potential for building and managing *knowledge models* (for application system services as well as workflows)
- Appropriate adaptation platform for change management (software evo-lution): *Active Components (AC)*