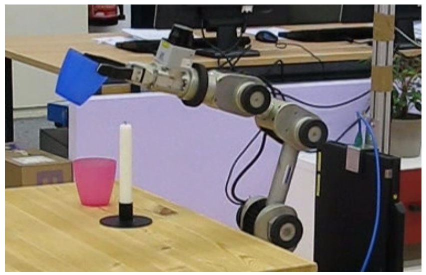
Model-Driven Engineering and Run-Time Model-Usage in Service Robotics

Andreas Steck (M.Sc.),
Alex Lotz (M.Sc.),
Prof. Dr. Christian Schlegel

Institut für Informatik
Hochschule Ulm

http://www.hs-ulm.de/schlegel
http://www.zafh-servicerobotik.de/ULM/index.php
http://smart-robotics.sf.net/
http://www.youtube.com/user/roboticsathsulm

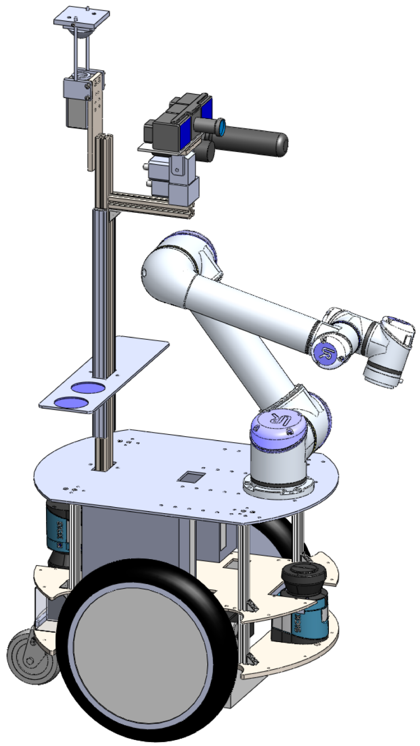


Hochschule Ulm

Servicerobotik

Autonomous mobile Service Robots

Part I



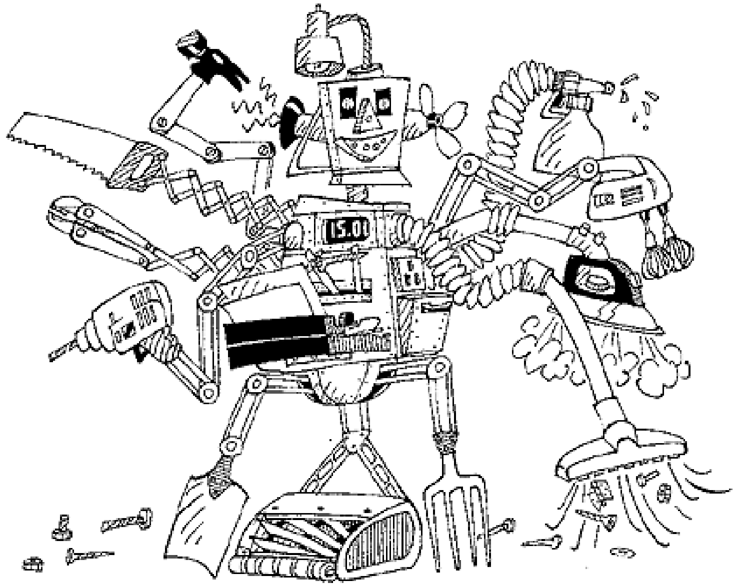
Hochschule Ulm

What is the Challenge in Robotics?

Part I

- The current situation in software for robotics can be compared with the early times of the *World Wide Web* where one had to be a computer engineer to setup web pages.

- The *World Wide Web* turned into a universal medium only since the availability of tools
  - which have made it accessible to everyone
  - which allow domain experts (like journalists) to provide content without bothering with technical details
  - which ensure sustainability / availability of contents independently of preferred operating systems, browsers etc.



=> separation of roles and separation of concerns
=> this is a universal approach towards successfully handling complexity:
applications, markets, sharing efforts / risks

What is the Challenge in Robotics?

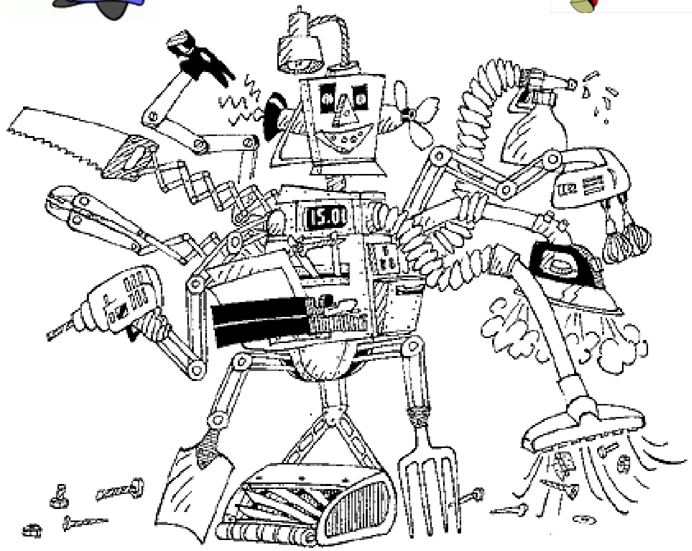
Part I

- Current situation:
  - *no "Separation of Roles"*
    - end users
    - *system integrators*
    - *component developers*
    - *framework developers*

  - *no "Separation of Concerns"*
    - computation
    - communication
    - configuration
      (parameters at component / system level)
    - coordination
      (orchestration, ressource management)








Robotics so far circumvented the problem of a missing abstraction by not separating between the roles of the component builder and the system integrator.

As long as both roles are carried out by the same persons, explicit descriptions which allow black-box reuse of existing solutions are not considered as essential.

Separation of Roles
Separation of Concerns

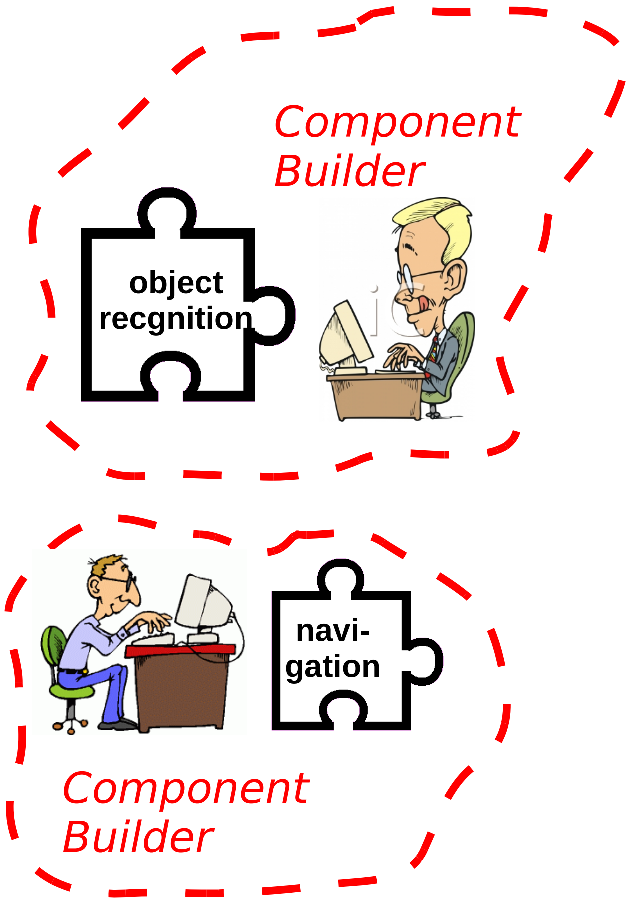
Part I

The Big-Bang Theory:
Howard unpacking food with robot

http://youtu.be/bKT13zcX_3U

Separation of Roles
Separation of Concerns

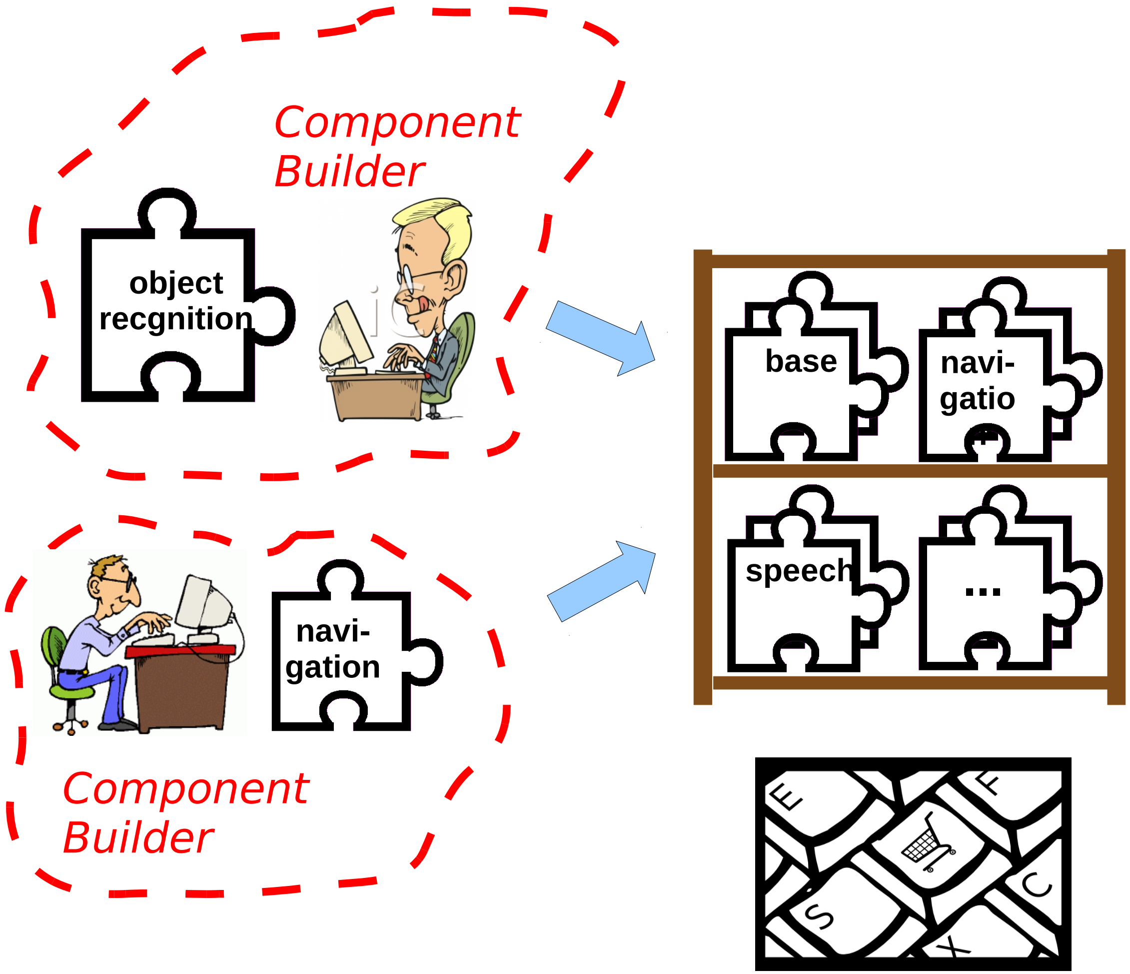
Part I



„freedom from choice“
in order to ensure
system-level conformity

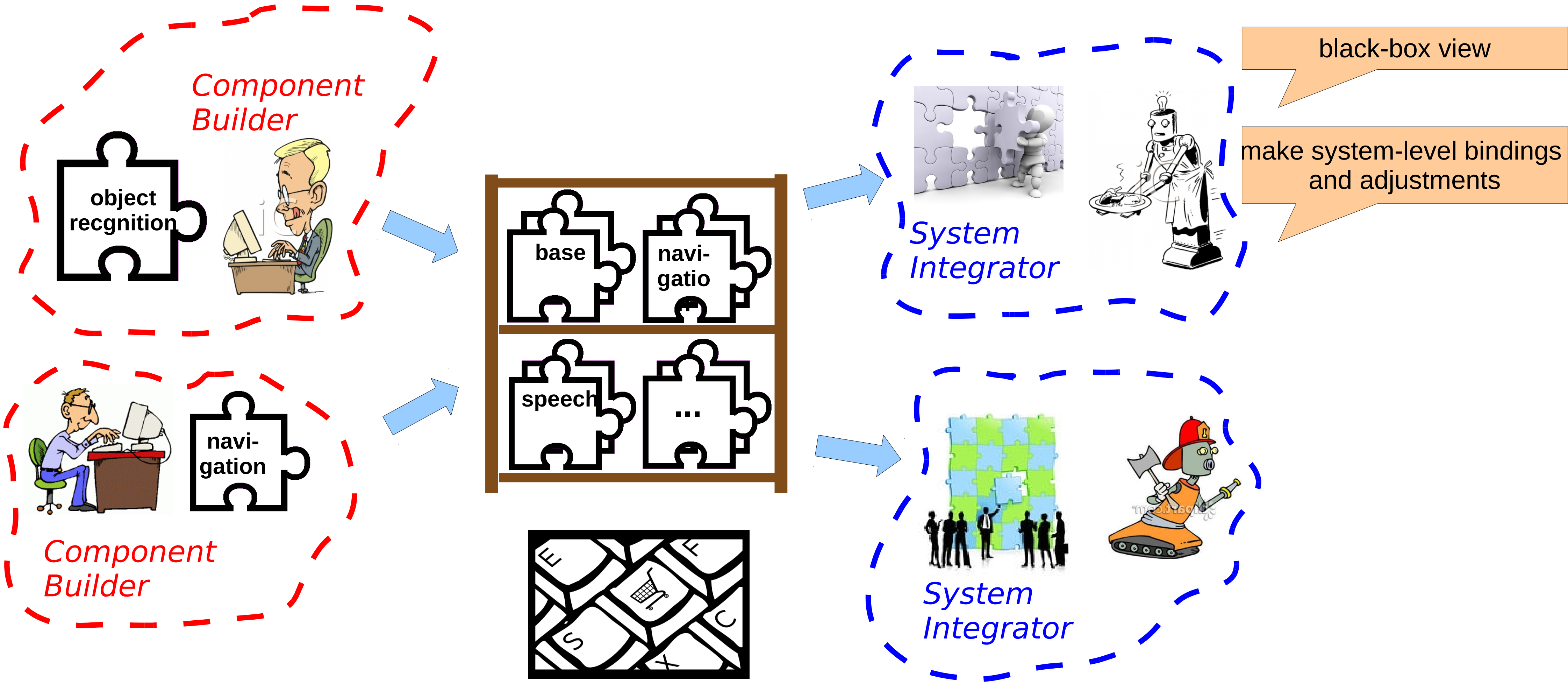
Separation of Roles
Separation of Concerns

Part I



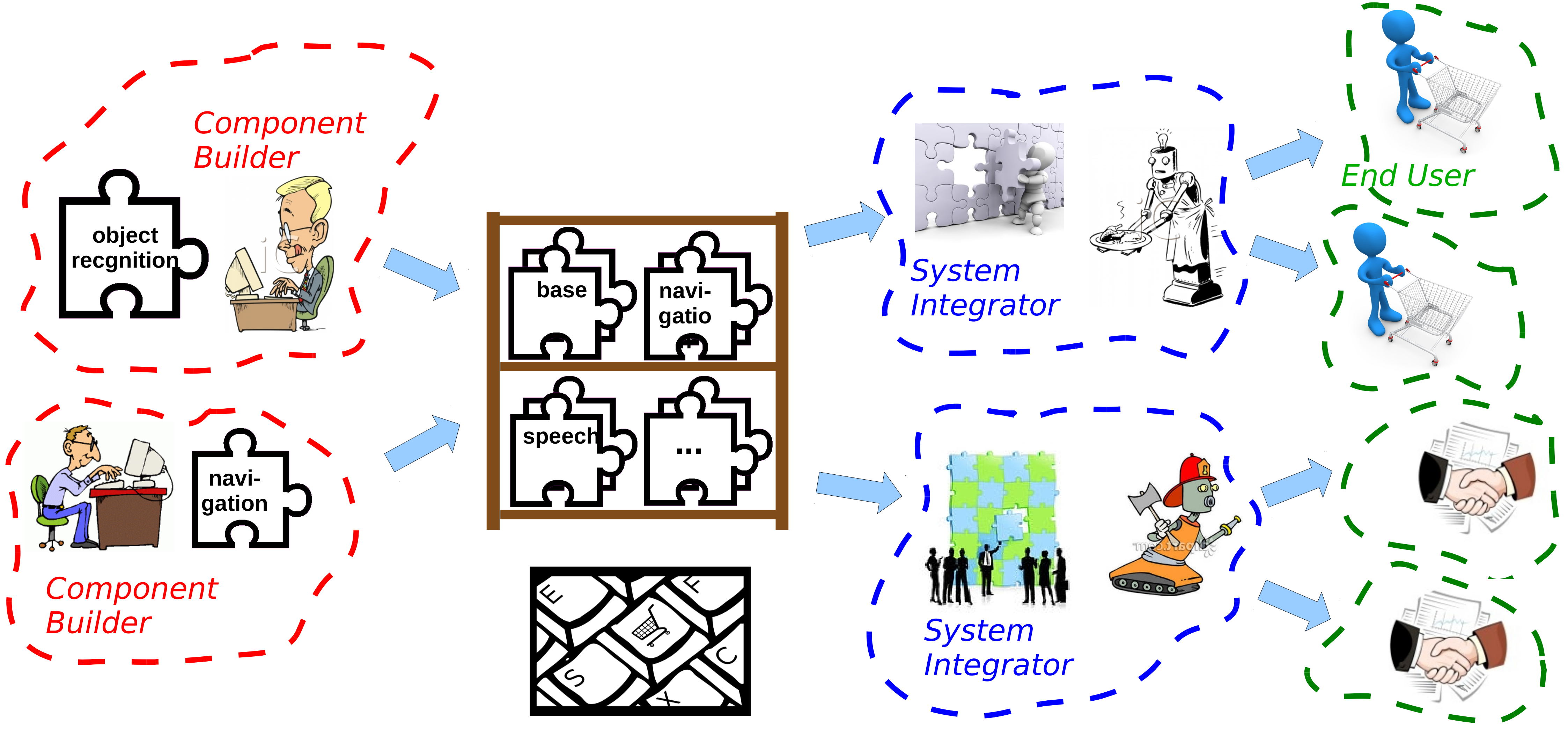
Separation of Roles
Separation of Concerns

Part I



Separation of Roles
Separation of Concerns

Part I

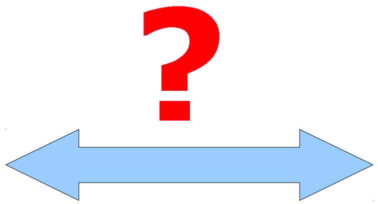


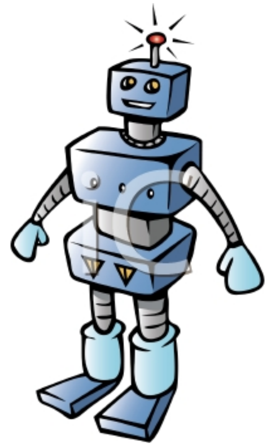
What is Different in Robotics?

Part I

- The *difference* of robotics compared to other disciplines (e.g. automotive, avionics) is
  neither the huge variety of different sensors, actuators, hardware platforms
  nor the number of different disciplines being involved.
- We are convinced that *differences* of robotics compared to other domains *originate from* the need of a robot to cope with *open-ended environments* *while having* only *limited resources* at its disposal.

=> The best matching between current situation, proper robot behavior and ressource assignment becomes overwhelming even for the most skilled robot engineer!





What is Different in Robotics?

Part I

- The *difference* of robotics compared to other disciplines (e.g. automotive, avionics) is
  neither the huge variety of different sensors, actuators, hardware platforms
  nor the number of different disciplines being involved.
- We are convinced that *differences* of robotics compared to other domains *originate from* the need of a robot to cope with *open-ended environments while having* only *limited resources* at its disposal.

- *Limited resources* require decisions: when to assign which resources to what activity taking into account perceived situation, current context and tasks to be fulfilled.

- Due to *open-ended real-world environments*, it is impossible to statically assign resources in advance in such a way that all potential situations arising at runtime are properly covered.

- Due to the *enormeous sizes of the problem space and the solution space* in robotics, there will *always be a deviation between design-time and run-time optimality*.

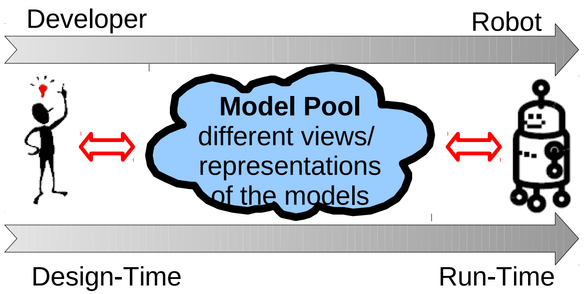
- Therefore, there is a need for dynamic resource assignments at runtime: managing variants / variability at runtime by late bindings of purposefully left-open variation points *(models@runtime, accessible via MDSD + DSLs)*

- ***future automotive systems face the very same challenges ...***

The Big Picture …
… Design-time / Run-time Model Usage

Part II

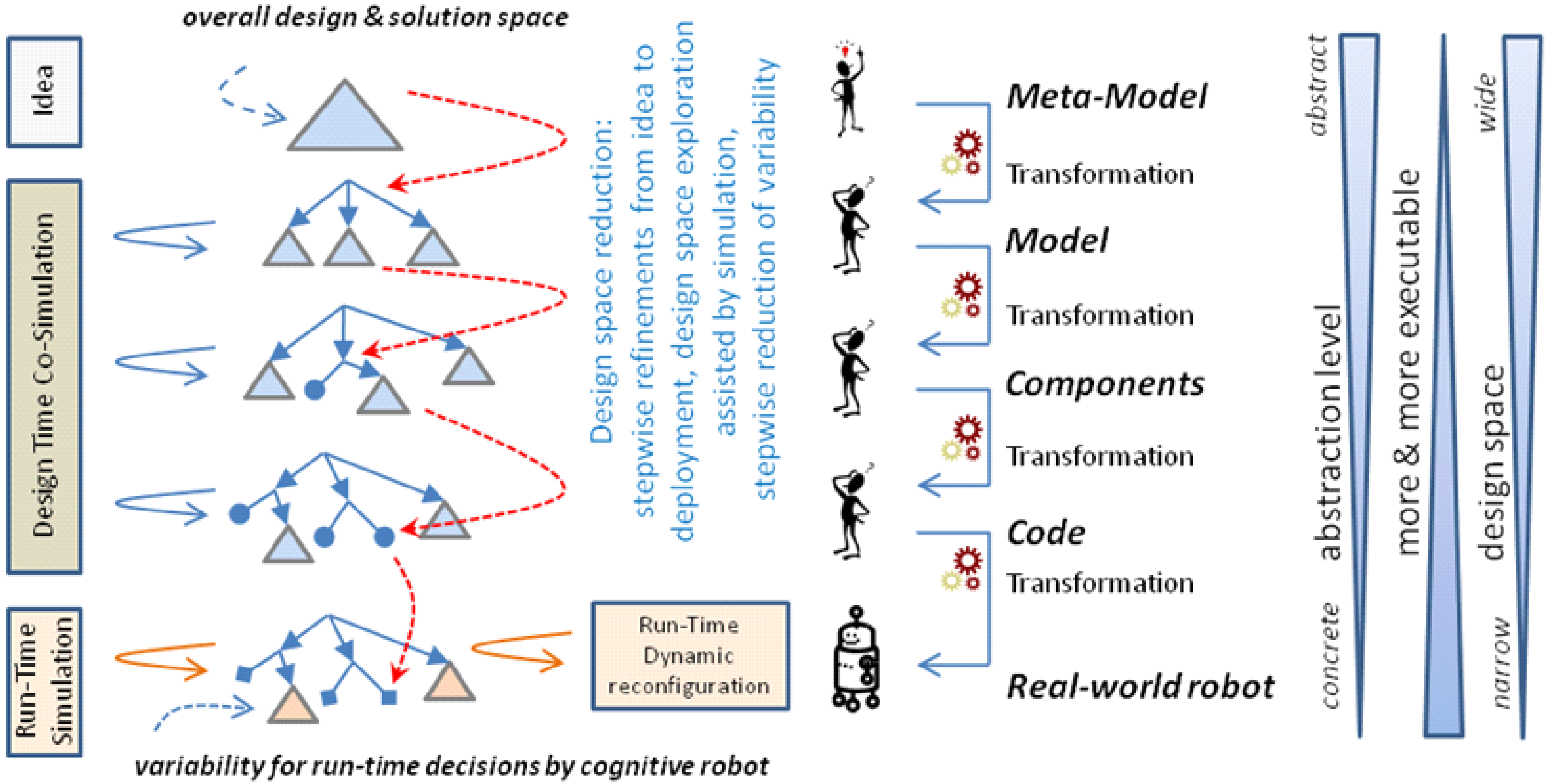
From code-driven to model-driven engineering in robotics in order to achieve:
- separation of roles
- separation of concerns
- managing run-time decisions



Contributions / Focus of work:
- make the step from code-driven to model-driven development of robotic systems by providing a robotics meta-model for robotic software components,
- providing levels of abstraction which allow to transform the models and generate code out of them,
- using the models of the robotics software components at design-time for simulation and analysis purposes, for example, real-time schedulability analysis of the real-time tasks,
- bridging between design-time models of robotics software components and their run-time representation,
- using models at run-time to support the decision making process of the robotic system by binding at run-time variation-points that have been left-open purposefully at design-time

The Big Picture …
… Design-time / Run-time Model Usage

Part II



- *use models for the entire life-cyle of the robot*
- *models are refined step-by-step until finally they become executable*
- *variation points: design-time (component builder, system integrator), runtime (robot)*

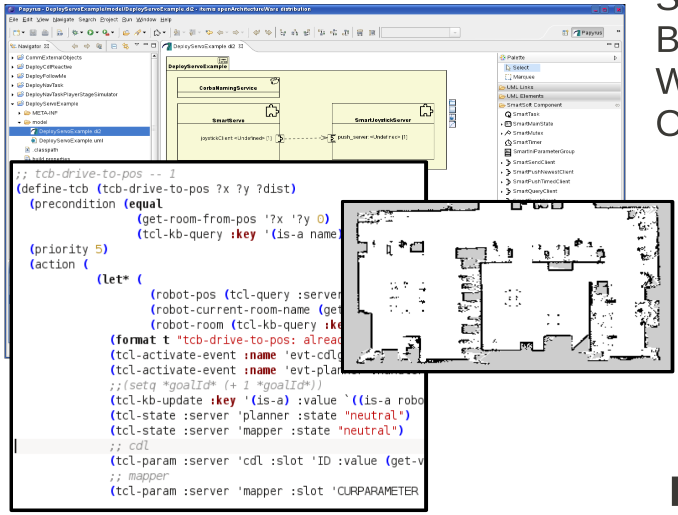
Hochschule Ulm

The Big Picture ...
... Model-Centric Robotic Systems

Part II

Developer → *Robot*

Create/ Modify Models



SmartMDSD Toolchain,
Blender, Solid Works,
World / Map Editor,
Ontosaurus, ...

Model Pool
different views/
representations
of the models

Modify Models

Manipulate models at run-time

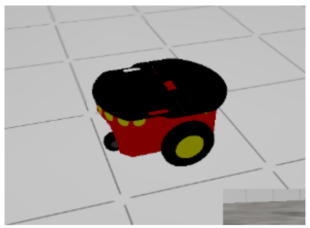
Reflect current state of the world and robot in the models

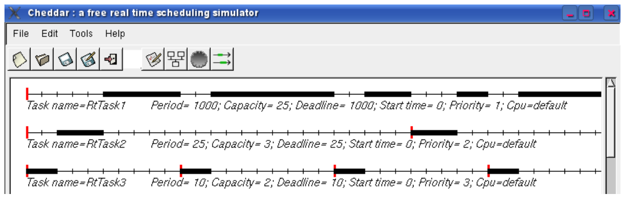
Make decisions at run-time depending on the models

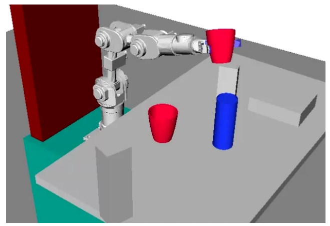
name	: *CDL*
state	: *"neutral"*
parameter	
strategy	: followPerson
freebehavior	: activate
lookuptable	: default
goalmode	: person
approachdist	: 500
id	: –
resources	
task–1	
period	: 100ms

Reason on the Models

Analysis, Simulation, Planning, ...







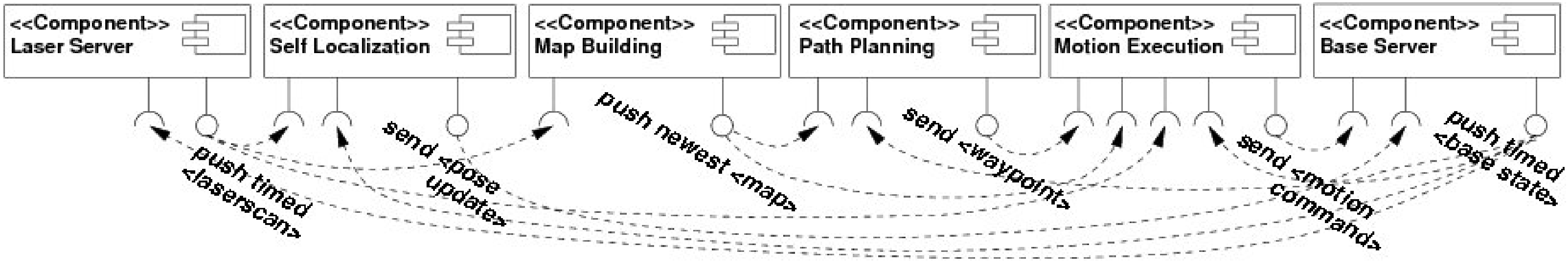
CHEDDAR, OpenRAVE, Gazebo, Metric-FF, LAMA, ...

Design-Time → *Run-Time*

Where to Start?

Part II

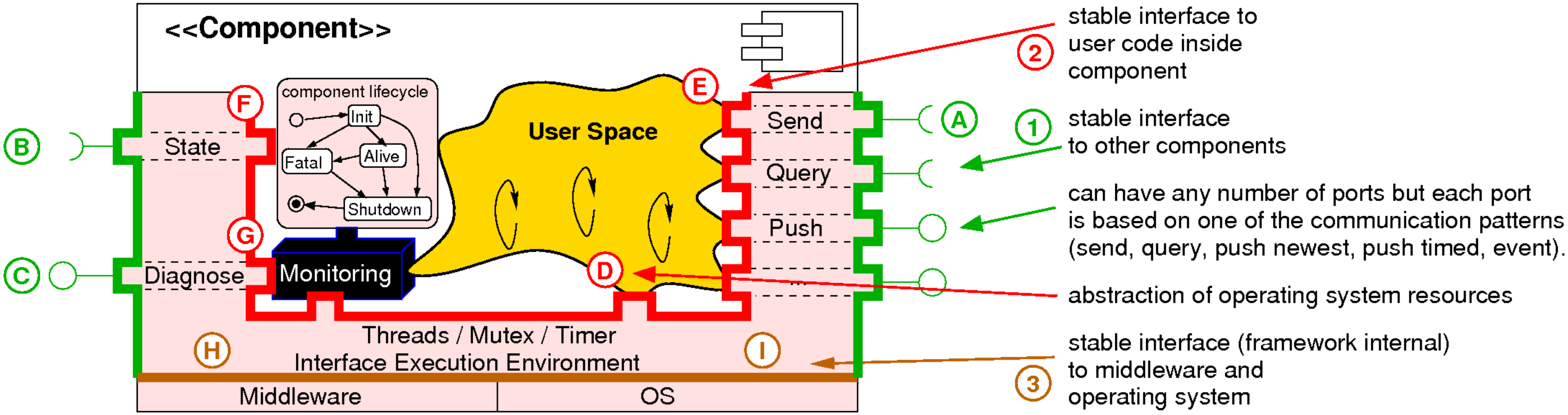
- **CBSE** (Component Based Software Development)
- **SOA** (Service-Oriented Architecture)
- **MDSD** (Model-Driven Software Development)



- Separating the roles of the component builder, system integrator and the robot requires to identify, specify and explicate stable structures as well as variation points each role can rely on.
- These stable structures and variation points build the ground for a model-based representation. Representing the structure of the component as meta-model enforces compliance of components with the meta-model via a MDSD-toolchain.
- We identified the component hull as the key structure to address the above challenges.

The SmartSoft Component Model
Stable Interfaces

Part II



- *Services* are defined by a *Communication Pattern* and *Communication Objects*
- *Communication Objects* are communicated between components: platform-independent, by-value
- *Services* are offered / used by components via *Ports*

The SmartSoft Communication Patterns

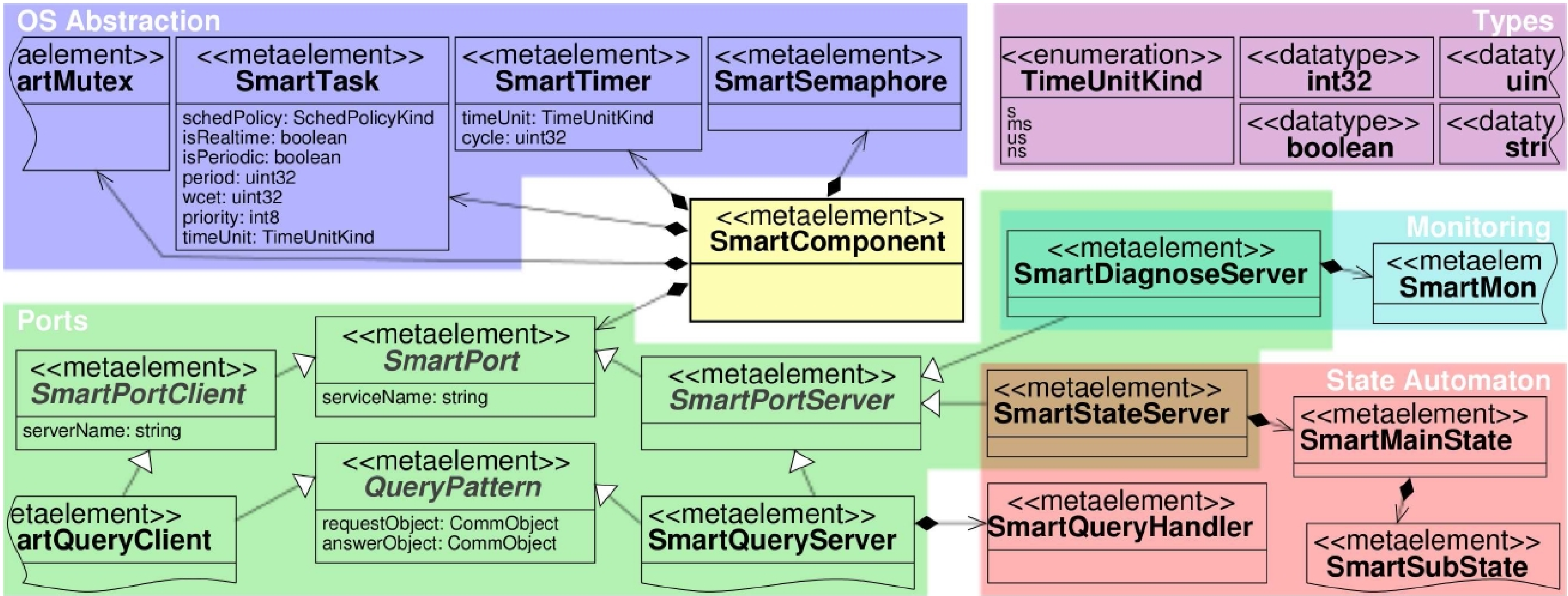
send	one-way communication
query	two-way request/response
push newest	1-to-n distribution
push timed	1-to-n distribution
event	asynchronous conditioned notification

The SmartSoft Services

param	component configuration
state	activate/deactivate component services
wiring	dynamic component wiring
diagnose	introspection of components
(internally based on communication patterns)	

The SmartSoft Component Model
Excerpt of the SmartMARS Meta-Model

Part II



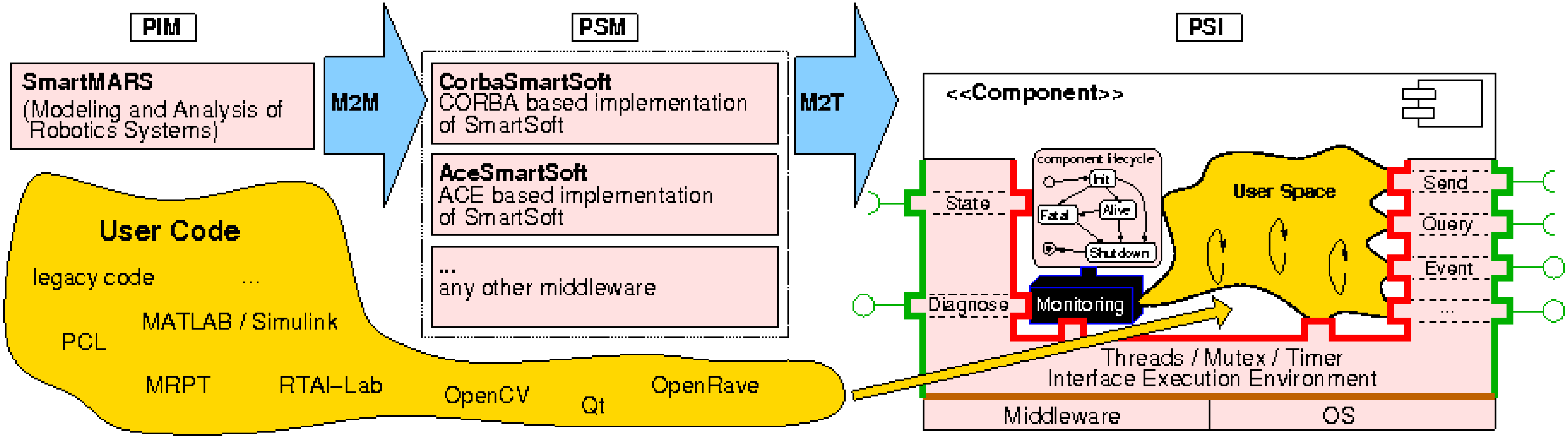
service robotik ZAFH

Model-Driven Software Development SmartMDSD

Part II

Illustration of the Development Process

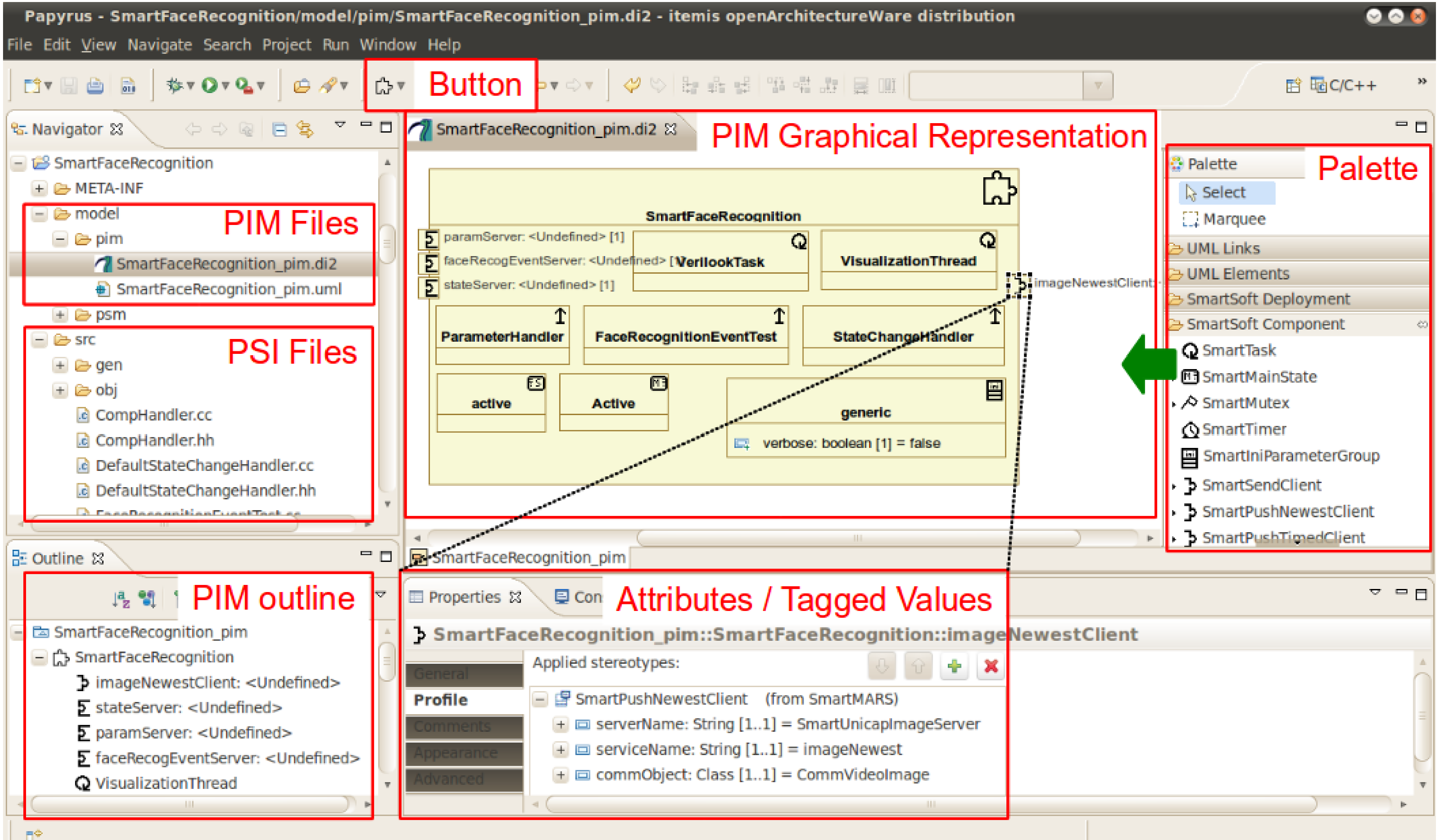
- Implemented as UML 2.0-Profile for Robotics Software Components
- supports Component Development, System Integration, Deployment
- based on standards: UML 2.0, Papyrus, Eclipe Modeling Project, etc.
- different Runtime-Platforms, Middleware-Systems etc.



2-step transformation workflow (framework builder view)

Model-Driven Software Development Component Builder View

Part II



service robotik ZAFH

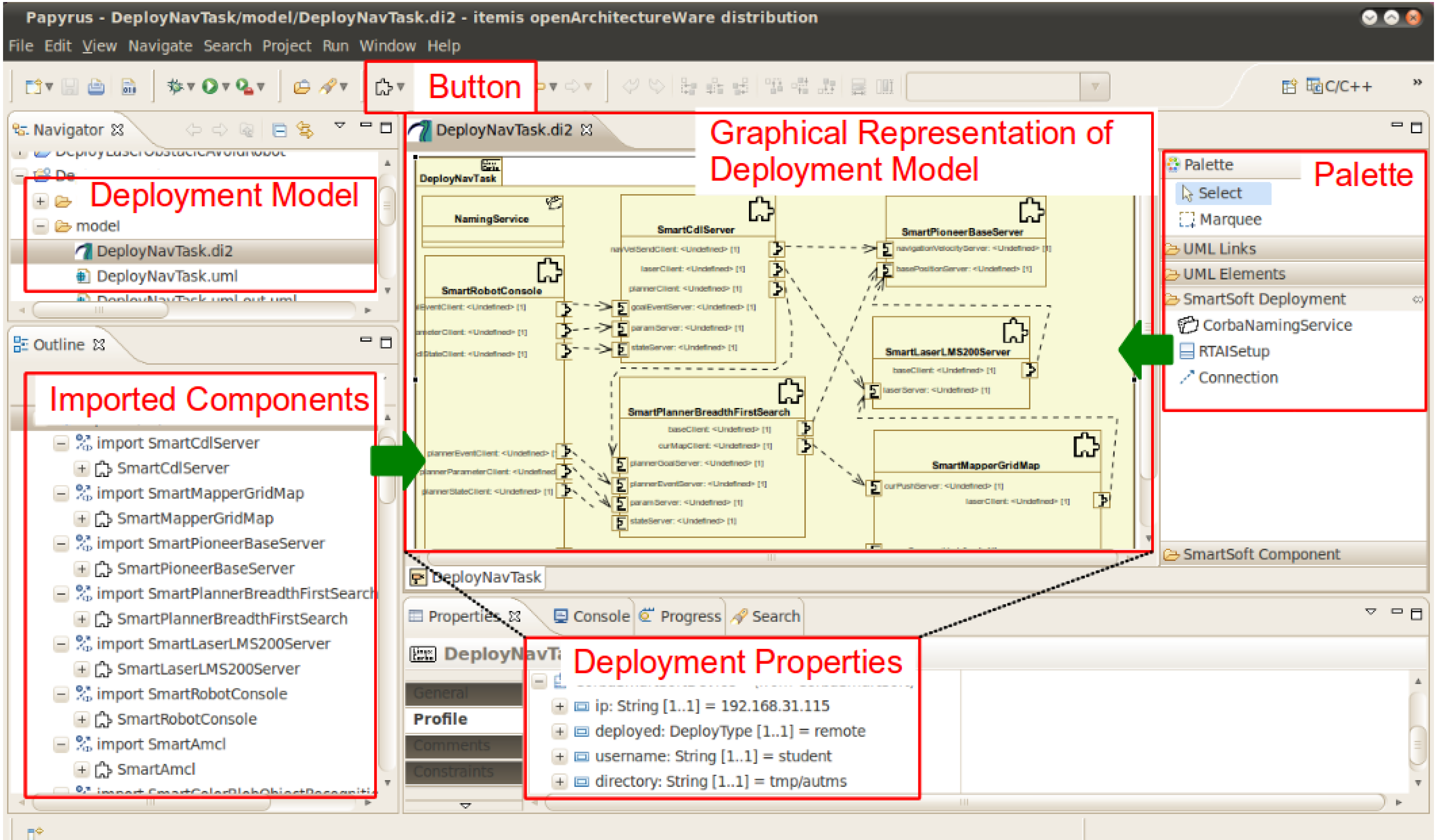
Model-Driven Software Development
Component Builder View

Part II

Screencast „Build a Component Hull“

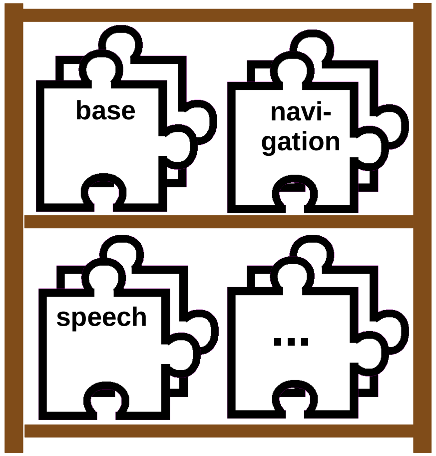
Model Driven Software Development
System Integrator View

Part II



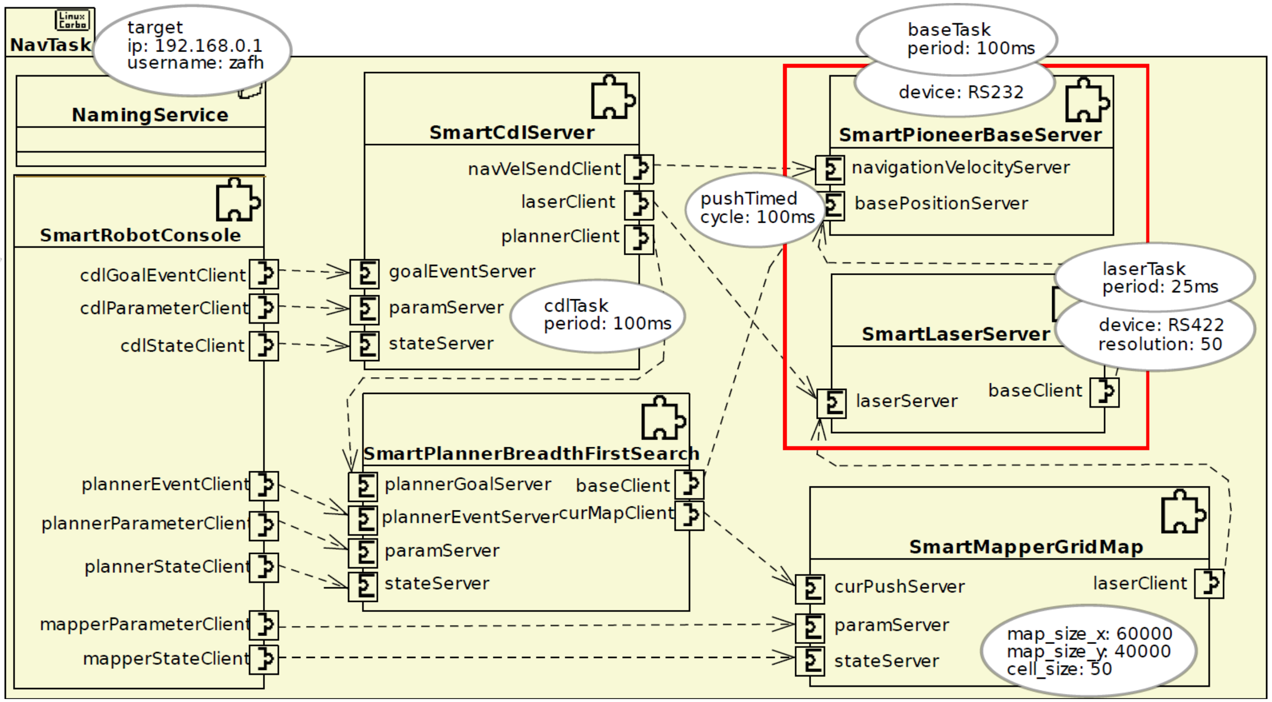
Model-Driven Software Development
System Integrator View

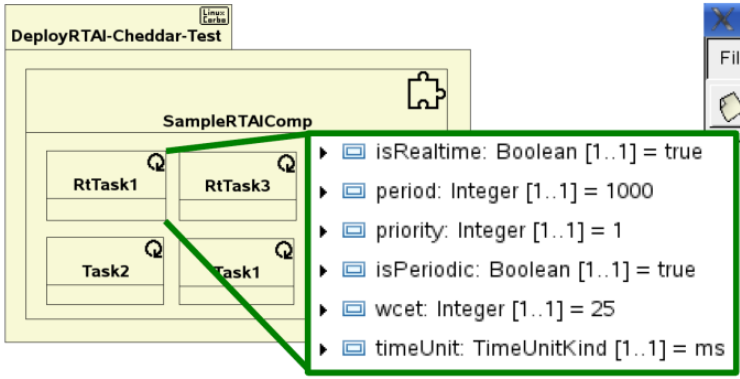
Part II

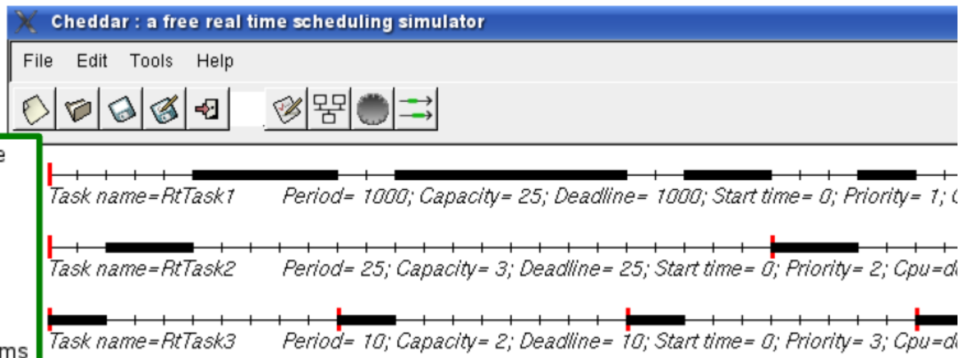


Component Shelf
Reusable Components

System Level Properties / Bindings / Conformance Checks



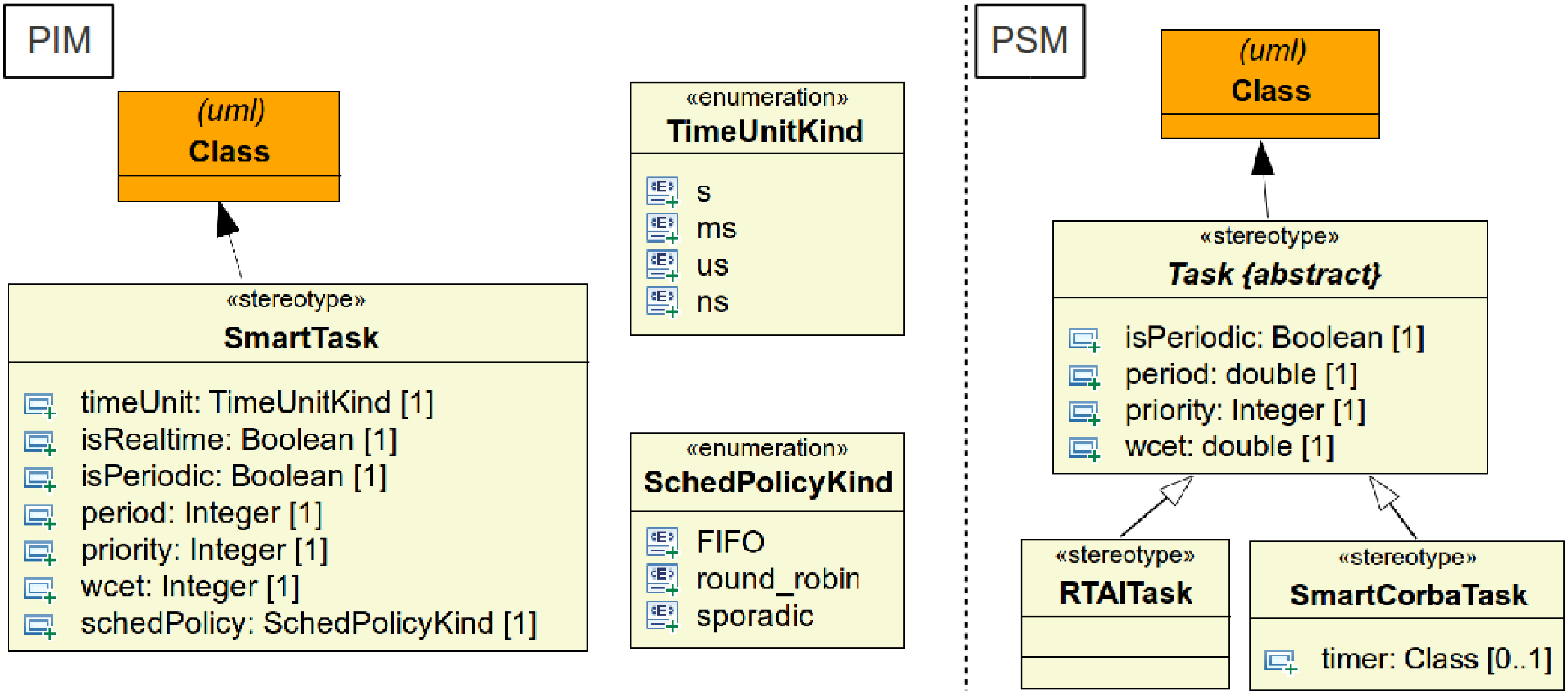




Hochschule Ulm

Model-Driven Software Development SmartMARS UML Profiles (PIM, PSM)

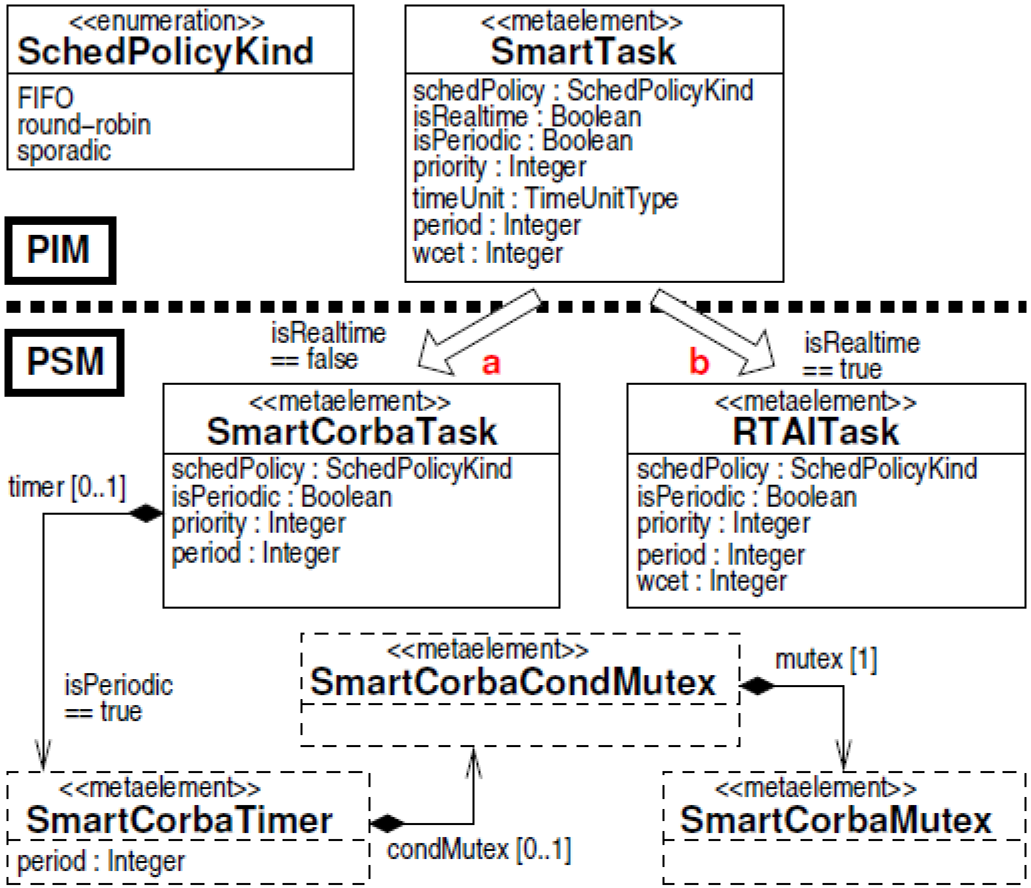
Part III

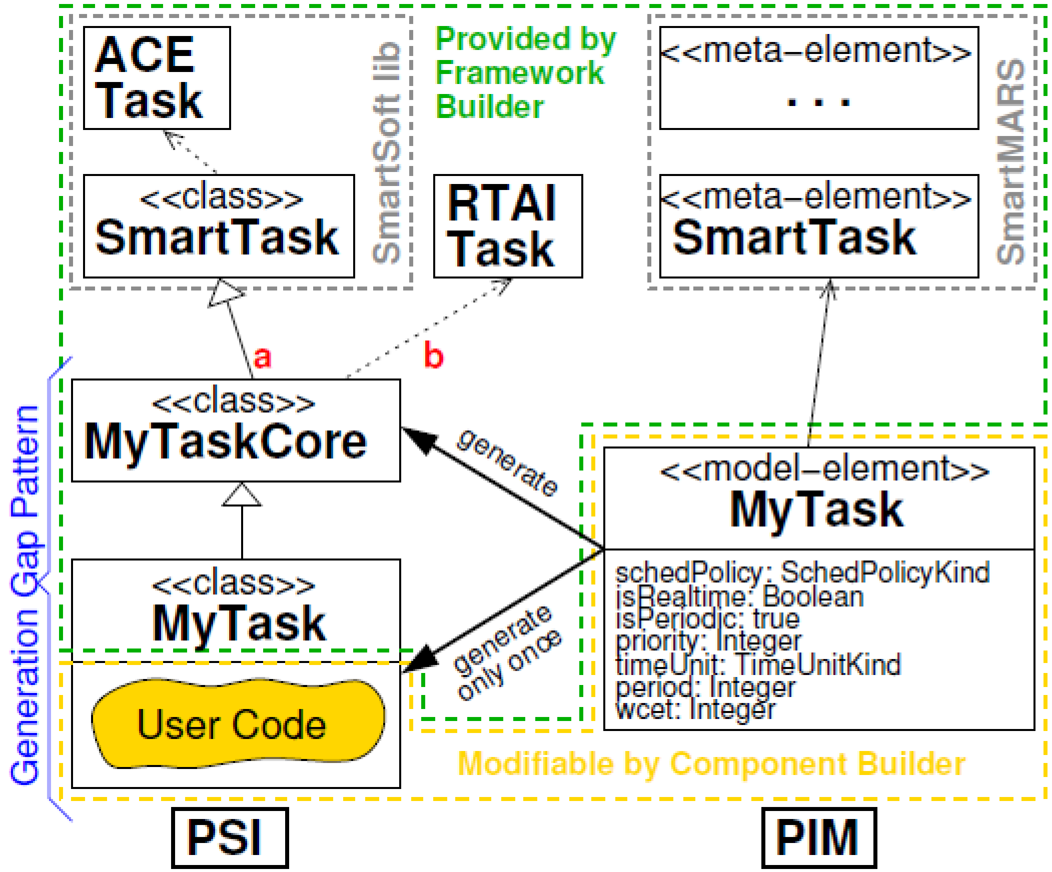


excerpts of UML Profile created with Papyrus UML (left PIM, right PSM)

Model-Driven Software Development Model Transformation + Code Generation

Part III





Transformation PIM into PSM

Generation Gap Pattern

Model-Driven Software Development PIM to PSM / SmartTask / isRealtime

Part III

task_mutex.ext

```
create uml::Class this addSmartTask(SmartMARS::SmartTask tsk, uml::Component cmp) :
    cmp.packagedElement.add(this) ->
    this.setName(tsk.name) ->
    if( tsk.isRealtime == true) then
    {
        this.applyStereotype("CorbaSmartSoft::RTAITask") ->
        setTaggedValue(this, "CorbaSmartSoft::RTAITask", "isPeriodic", tsk.isPeriodic) ->
        setTaggedValue(this, "CorbaSmartSoft::RTAITask", "wcet", tsk.wcet.toSecond(tsk.timeUnit.name)) ->
        setTaggedValue(this, "CorbaSmartSoft::RTAITask", "period", tsk.period.toSecond(tsk.timeUnit.name)) ->
        setTaggedValue(this, "CorbaSmartSoft::RTAITask", "priority", tsk.priority)
    }
    else
    {
        this.applyStereotype("CorbaSmartSoft::SmartCorbaTask") ->
        setTaggedValue(this, "CorbaSmartSoft::SmartCorbaTask", "isPeriodic", tsk.isPeriodic) ->
        setTaggedValue(this, "CorbaSmartSoft::SmartCorbaTask", "wcet", tsk.wcet.toSecond(tsk.timeUnit.name)) ->
        setTaggedValue(this, "CorbaSmartSoft::SmartCorbaTask", "period", tsk.period.toSecond(tsk.timeUnit.name)) ->
        setTaggedValue(this, "CorbaSmartSoft::SmartCorbaTask", "priority", tsk.priority) ->
        if( tsk.isPeriodic == true ) then
        {
            setTaggedValue(this, "CorbaSmartSoft::SmartCorbaTask", "timer", cmp.addTimer(tsk.name, tsk.period, tsk.timeUnit.name))
        }
    };
```

Xtend Transformation Rule (M2M):
PIM to PSM model transformation of the SmartTask depending on the attribute "isRealtime"

service robotik ZAFH

Model-Driven Software Development PSM to PSI

Part III

smartTask.xpt

PSM → PSI Template

```
«DEFINE TaskUserSourceFile FOR CorbaSmartSoft::Task-»
«FILE this.getUserSourceFilename() writeOnce-»
«getCopyrightWriteOnce()»
#include "«this.getUserHeaderFilename()»"
#include "gen/«((CorbaSmartSoft::SmartCorbaComponent)this.eContainer()).getCoreHeaderFilename()»"

#include <iostream>

«this.getName()»::«this.getName()»()
{
    std::cout << "constructor «this.getName()»\n";
}

int «this.getName()»::svc()
{
    // do something -- put your code here !!!
    while(1)
    {
        «IF this.isPeriodic == true-»
        std::cout << "Hello from «this.getName()»  - periodic\n";
        smart_task_wait_period();
        «ELSE-»
        std::cout << "Hello from «this.getName()»\n";
        sleep(1);
        «ENDIF-»
    }
    return 0;
}
«ENDFILE»
«ENDDEFINE»
```

ServoTask.cc

PSI (user code .cc file)

```
#include "ServoTask.hh"
#include "gen/SmartServo.hh"

#include <iostream>

ServoTask::ServoTask()
{
    std::cout << "constructor ServoTask\n";
}

int ServoTask::svc()
{
    // do something -- put your code here !!!
    while (1)
    {
        std::cout << "Hello from ServoTask  - periodic\n";
        smart_task_wait_period();
    }
    return 0;
}
```

Xpand / Xtend Transformation (M2T): PSM to PSI model transformation

Hochschule Ulm

What do we need in Robotics?

Part IV

- Support for instances of components in tools:
  - including dedicated parametrization per instance
  - not adequately supported by UML and its extension mechanism (UML Profiles)
  - use case:
    - laser ranger component is used for front / rear laser ranger but with different bindings

- Variation Points: Support for different roles in tools / models:
  - each role (component builder, system integrator, robot) should have different access policies
  - use cases:
    - component builder binds a value that must not be changed by others
    - component builder specifies a range / set of values to define the decision space for other roles and defines which role is allowed to change / must bind the variation point

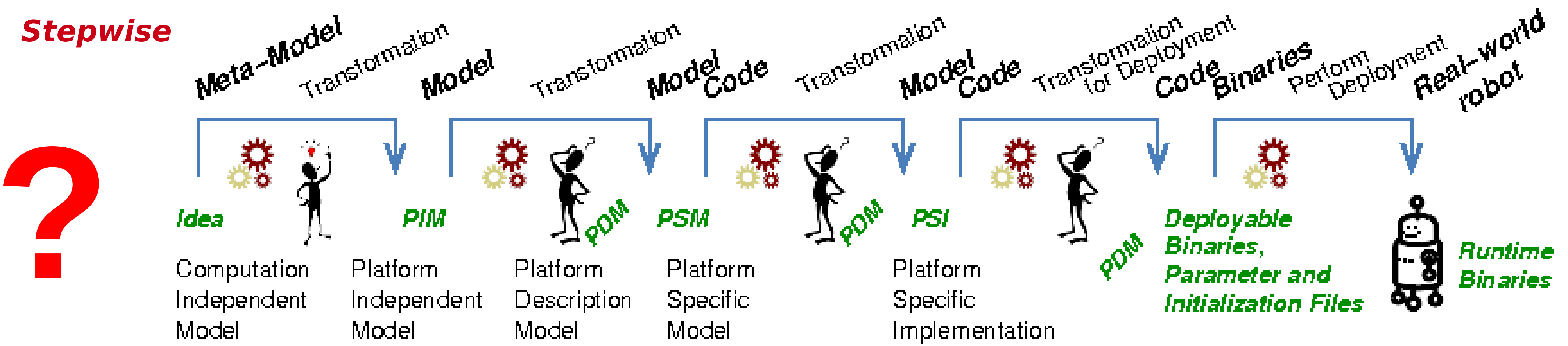
- Variation Points: Mechanisms to express relations between model elements and their parameters:
  - use cases:
    - modifying property „cycle time“ of navigation component directly changes property „maximum allowed velocity“ (is needed to allow for modifications of parameters without having to know about their internal functional relationship)

- Variation Points: Support for bindung / unbinding of model parameters:
  - modifying a specific parameter in the model may induce that depending parameters get unbound and have to be bound with respect to the new configuration
  - use case:
    - changing the processor type invalidates all hard real-time WCET

What do we need in Robotics?

- OMG MDA far too restrictive with respect to the workflow:
  - we want to make bindings at any place of the model at any time until finally there are enough bindings to become
    - (partially) executable by co-simulation
    - usable by the robot
  - we want to be assisted with respect to consistency etc. but we do not want to be restricted by a narrow and strictly ordered set of steps as within MDA

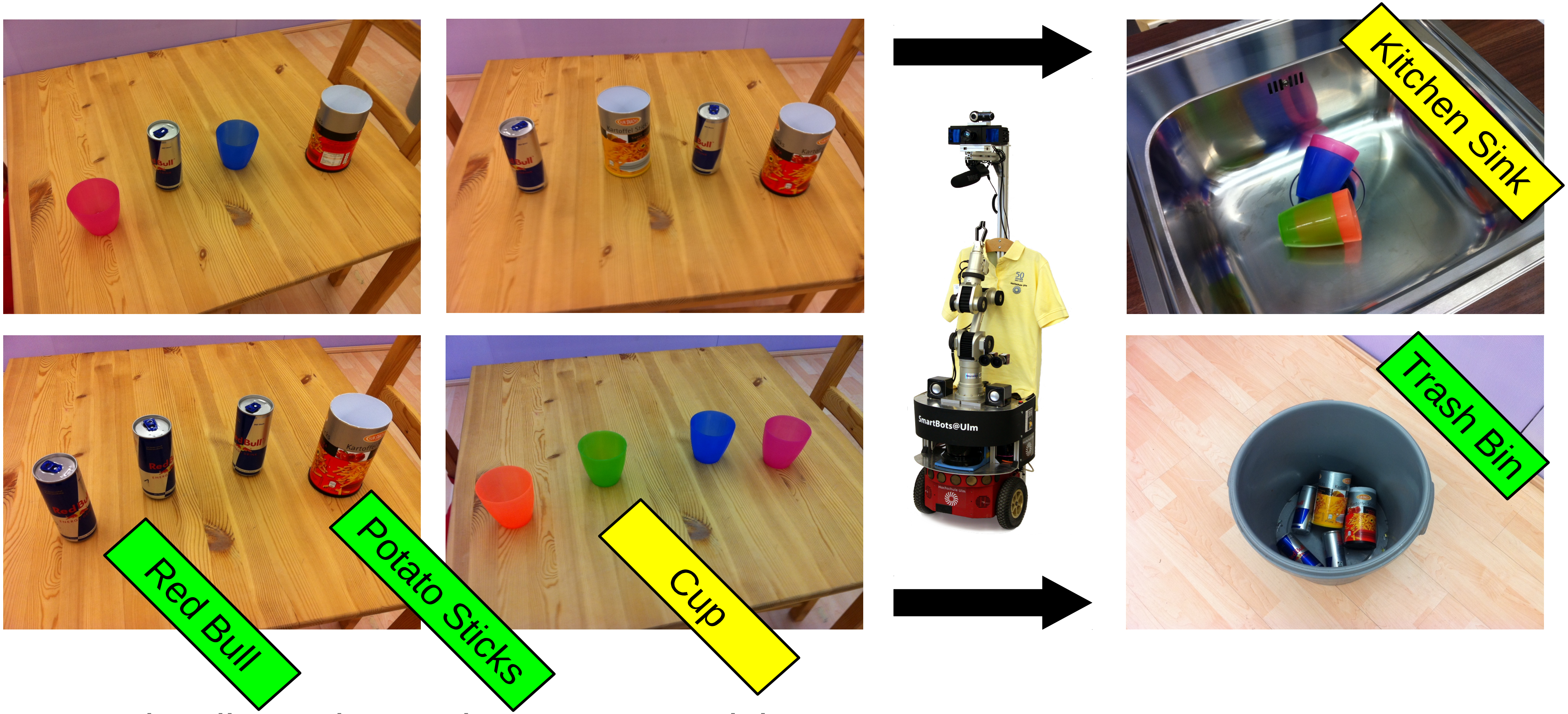
    (see e.g. platform specific information: parts need to be added early and other parts might be post-poned for late bindings)



service robotik ZAFH

Scenario: Robot "Kate" cleans up a table
Model-based Runtime Decisions

Part V



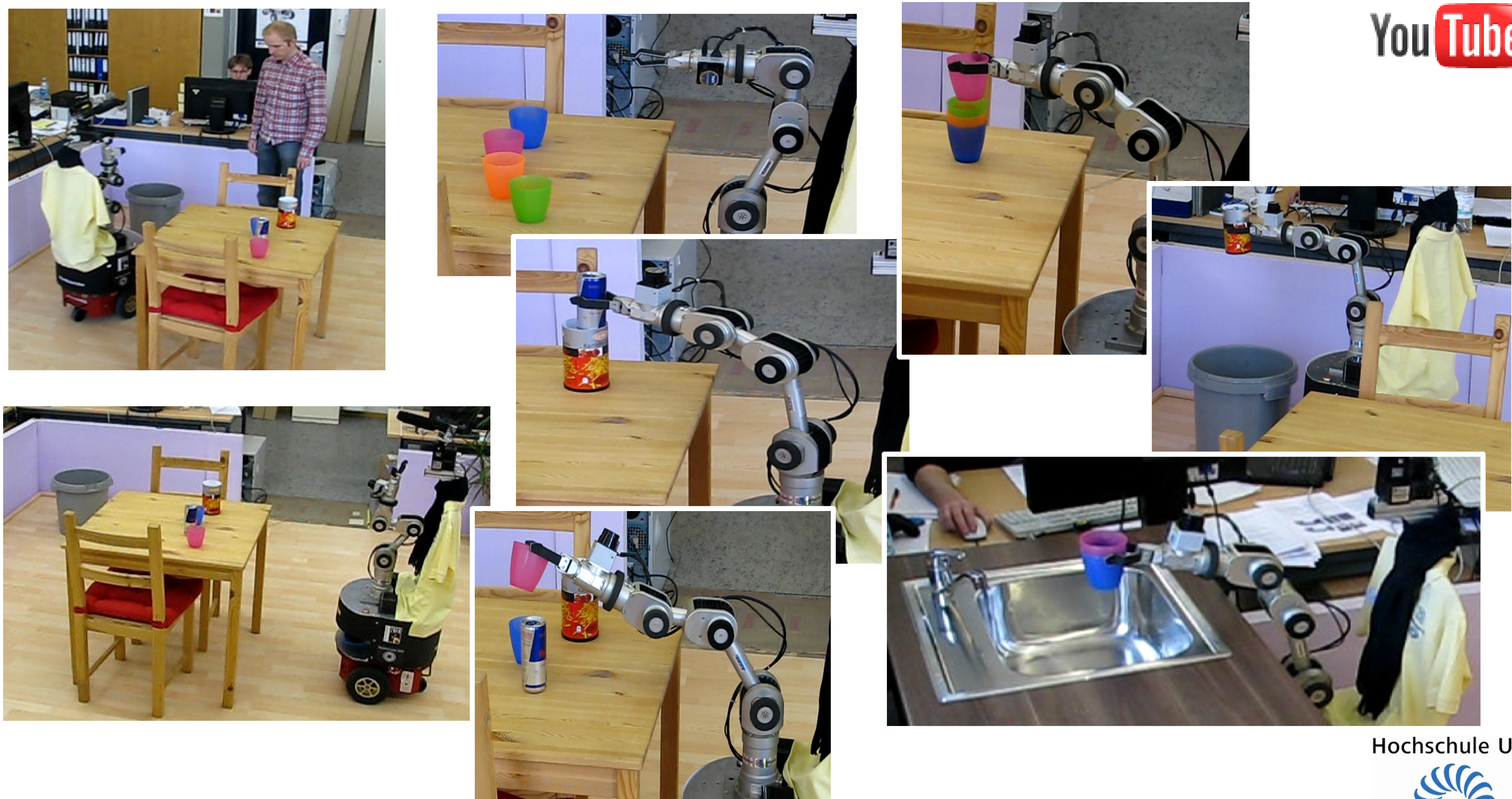
- a *„Red Bull"* can be put into *„Potato Sticks"*
- *cups* can be stacked into each other

Hochschule Ulm

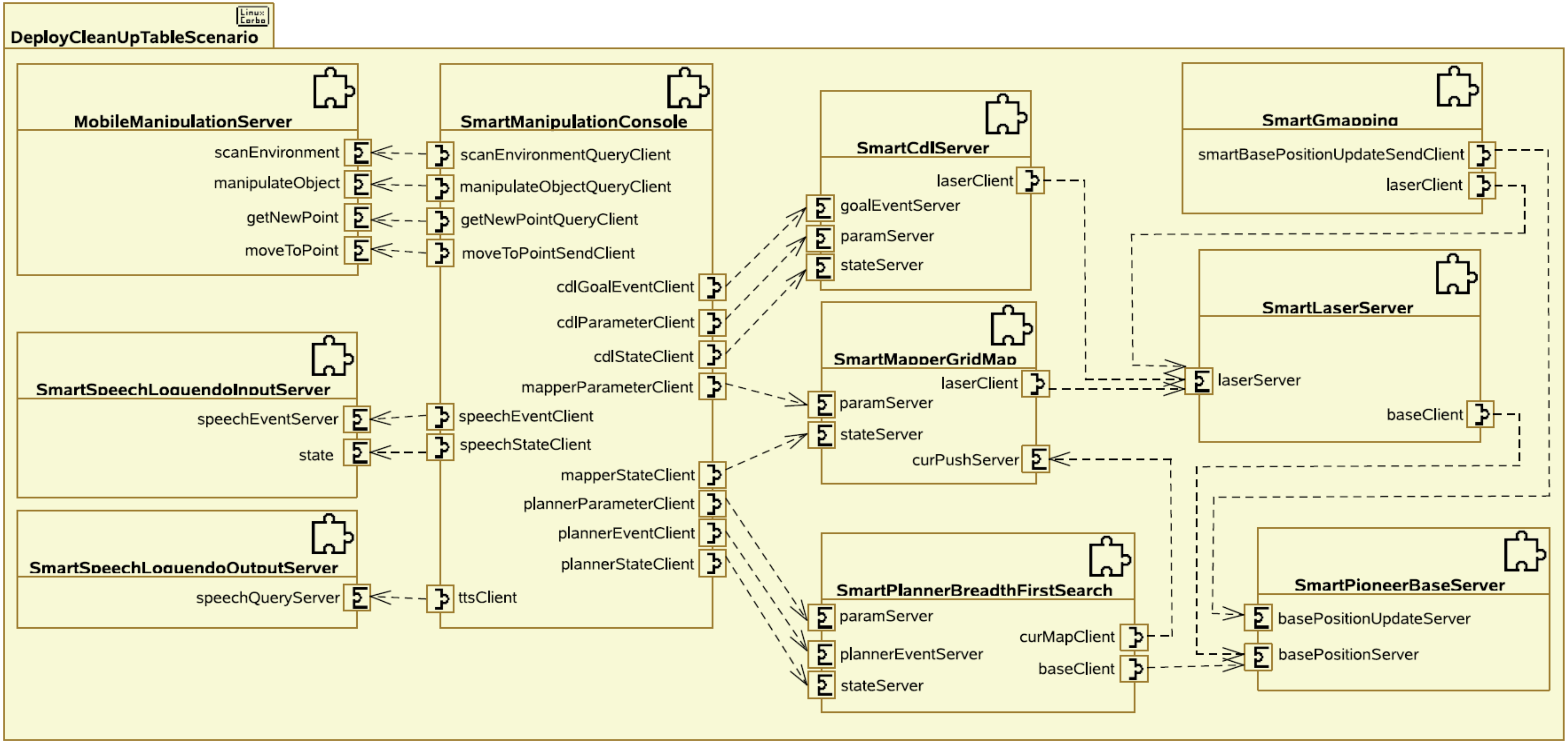
Scenario: Robot “Kate” cleans up a table
Model-based Runtime Decisions

Part V

You Tube

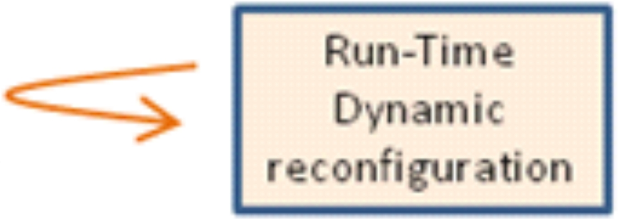


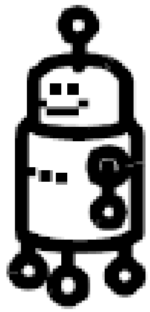
Scenario: Robot "Kate" cleans up a table
System Integration / Deployment
Part V

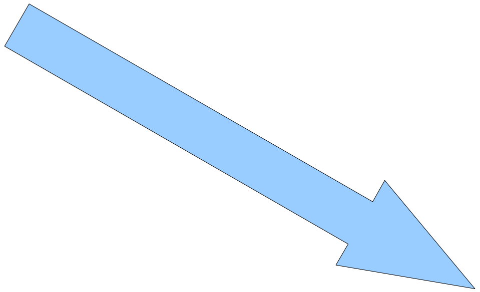


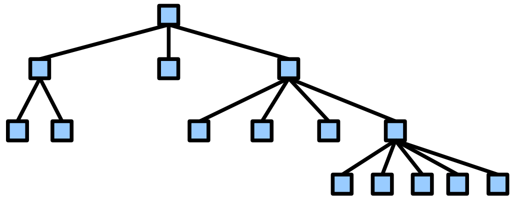
Model-based Runtime Decisions

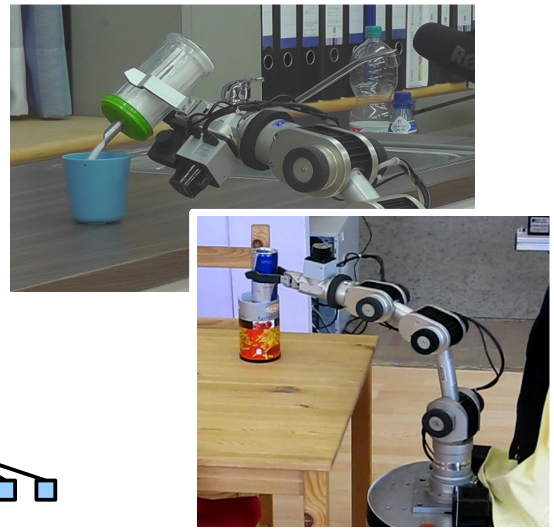
Part V







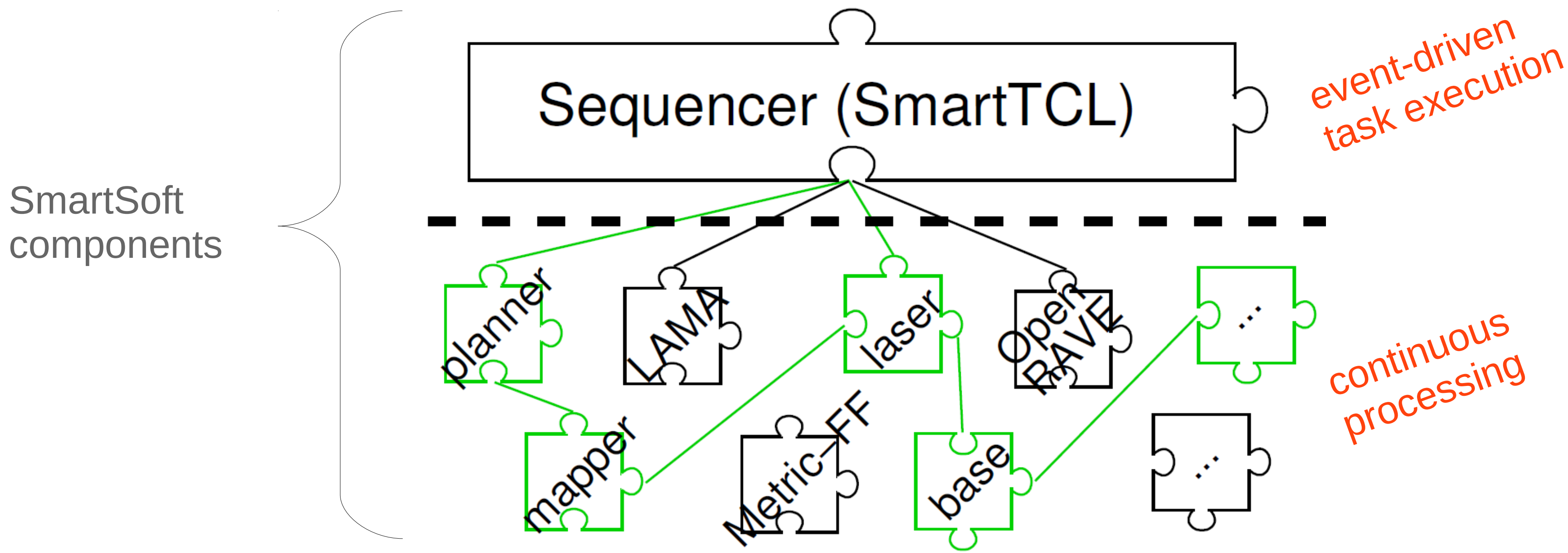




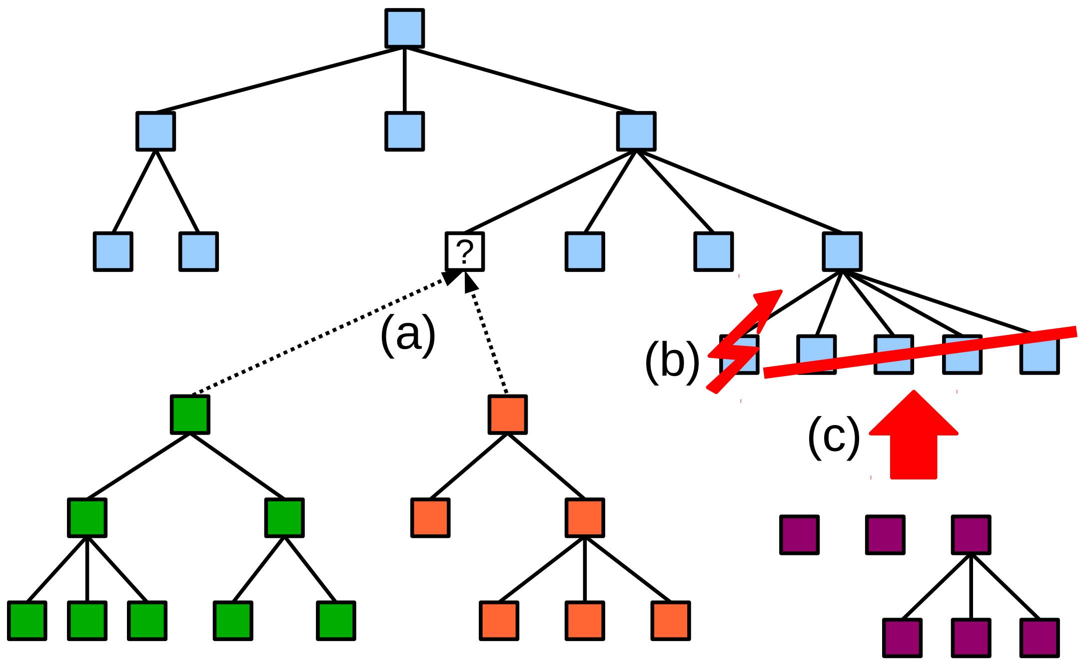
Hochschule Ulm

Model-based Runtime Decisions
Sequencer Orchestrates the Components

- bridges between continuous processing and event-driven task execution
- the sequencer orchestrates the software components in the system:
  - send parameters / configurations
  - switch components on/off to manage resources
  - change the wiring between the components
  - query information / wait for events



Model-based Runtime Decisions
Sequencer: SmartTCL Task-Tree



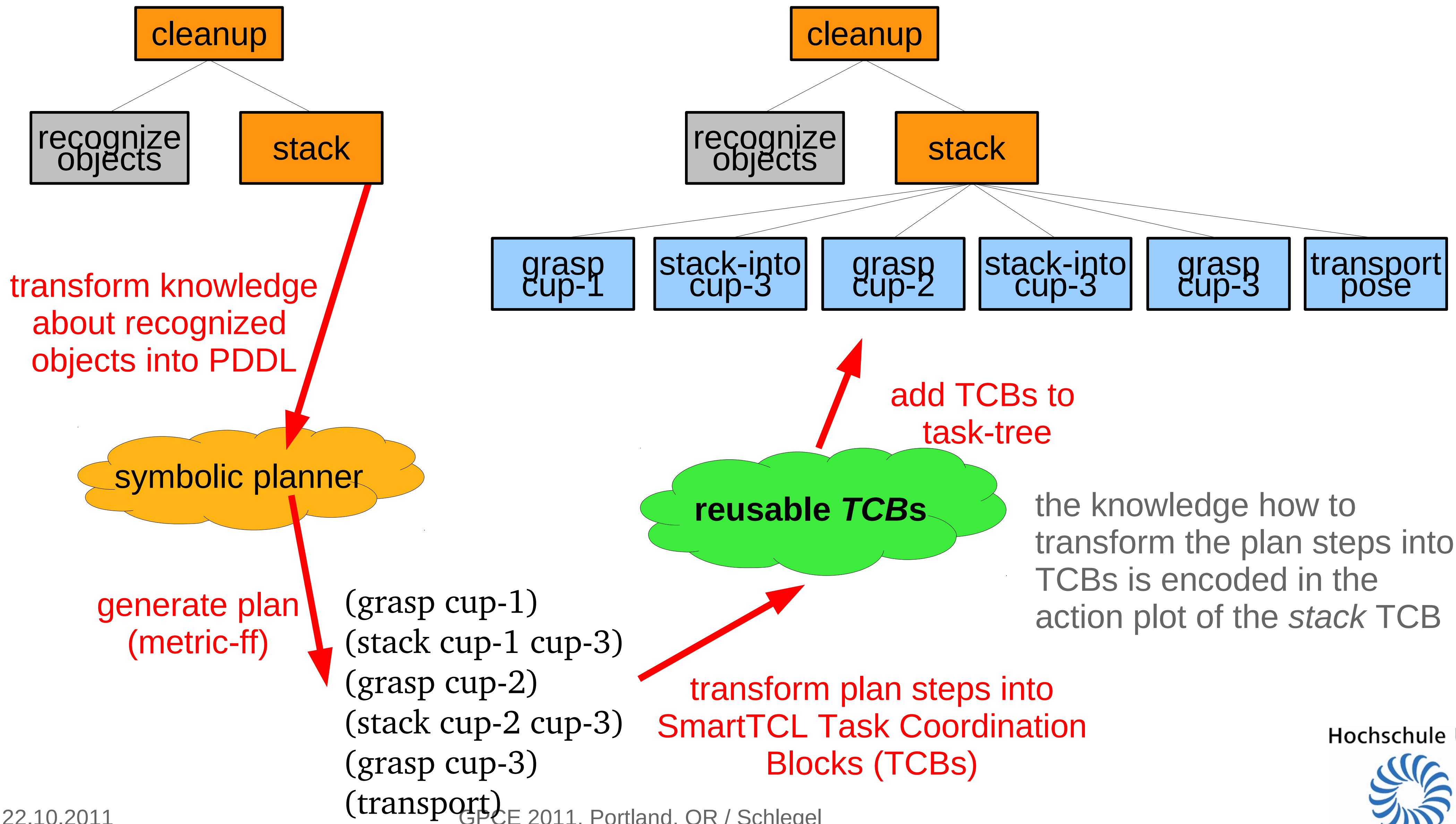
(a) select between alternatives at runtime

(b) handle contingencies

(c) delete, add or replace parts of the task-tree at runtime

- at runtime a task-tree is dynamically created, modified and executed
- composes reusable action-plots to complex behaviors
- manages execution variants and contingencies of real world environments
- provides context and situation-driven task execution
- mediates between symbolic and subsymbolic mechanisms of information processing

Model-based Runtime Decisions
Calling a Symbolic Planner



Scenario: Robot "Kate" cleans up a table
Model-based Runtime Decisions

Part V

Watch Video on YouTube
http://www.youtube.com/roboticsathsulm

You Tube™

http://www.youtube.com/user/roboticsathsulm

SmartSoft MDSD Toolchain Links

Addendum

Where to start?
CBSE – Component Based SWE

“A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be developed independently and is subject to composition by third parties.” (Szyperski, 2002).

- explicitly consider reusable pieces of software including notions of independence and late composition
- composition can take place during different stages of the lifecycle of components:
  - » design phase (design and implementation)
  - » deployment phase (system integration)
  - » runtime phase (dynamic wiring of data flow according to situation and context).
- CBSE is based on the explication of all relevant information of a component to make it usable by other software elements ***whose authors are not known***.

Encapsulation / Composability (Meyer 2000):

- may be used by other software elements (clients),
- may be used by clients without the intervention of the component’s developers,
- includes a specification of all dependencies (hardware and software platform, versions, other components),
- includes a precise specification of the functionalities it offers,
- is usable on the sole basis of that specification,
- is composable with other components,
- can be integrated into a system quickly and smoothly

Where to start?
SOA – Service-Oriented Architecture

SOA are “the policies, practices, frameworks that enable application functionality to be provided and consumed as sets of services published at a granularity relevant to the service consumer. Services can be invoked, published and discovered, and are abstracted away from the implementation using a single, standards-based form of interface” (Sprott& Wilkes, 2004).

A SOA has to ensure hat services don’t get reduced to the status of interfaces, rather they have an identity of their own.

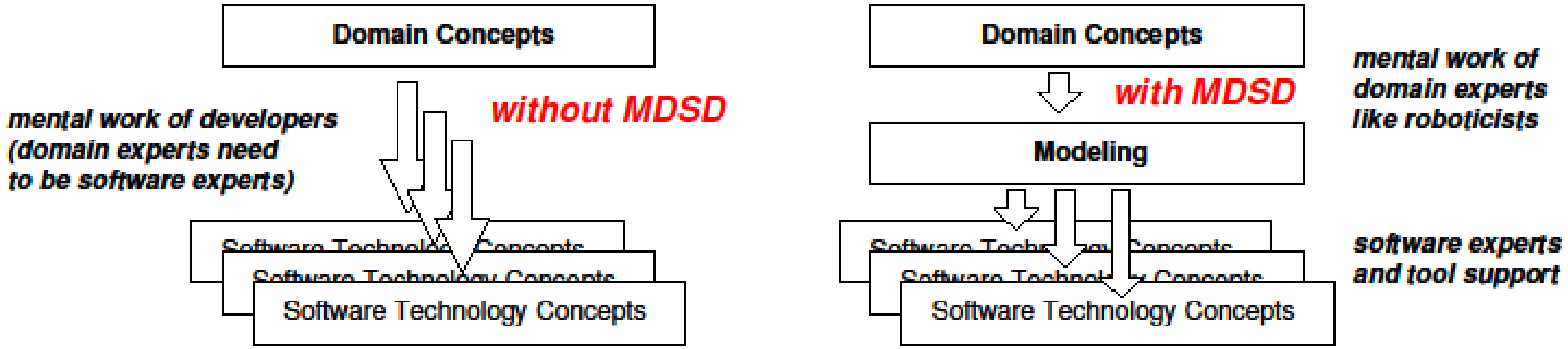
With SOA, it is critical to implement processes that ensure that there are at least two different and separate processes - for providers and consumers (Sprott & Wilkes, 2004).

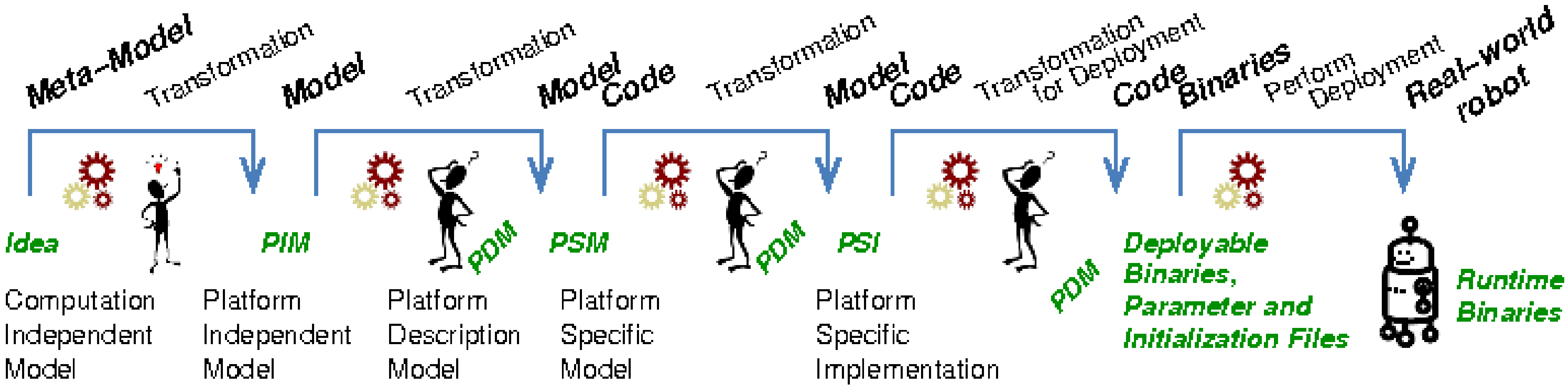
reusable	use of service, not reuse by copying of code/implementation
abstracted	service is abstracted from the implementation
published	precise, published specification functionality of service interface, not implementation
formal	formal contract between endpoints places obligations on provider and consumer
relevant	functionality is presented at a granularity recognized by the user as a meaningful service

Principles of good service design enabled by characteristics of SOA (Sprott & Wilkes, 2004) servicerobotik ZAFH

Where to start?
MDSD – Model-Driven SW Development

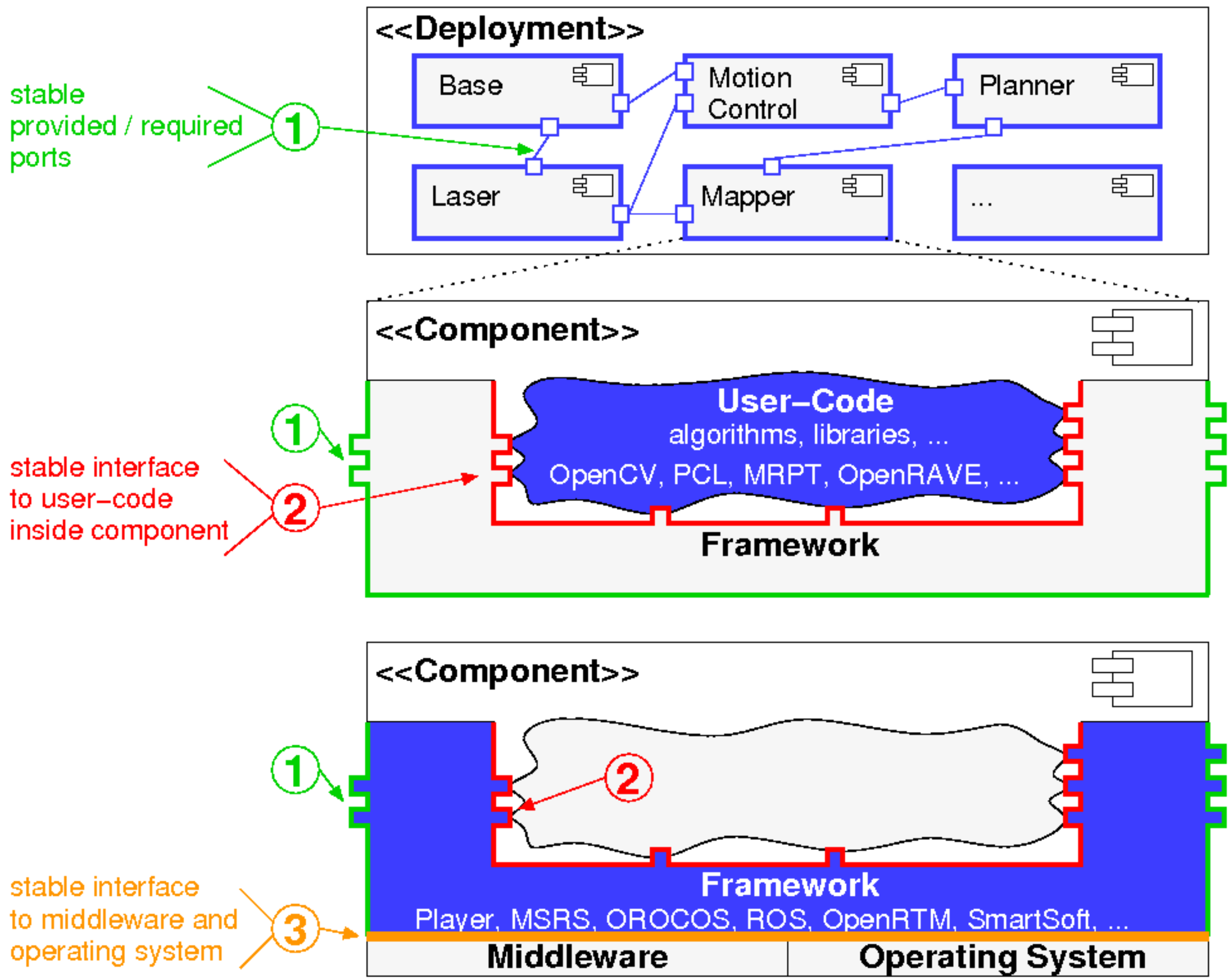
- make software development more domain related as opposed to computing related
- it is also about making software development in a certain domain more efficient and more robust due to design abstraction
- Analysis / requirements models are ***non-computational,*** **MDSD models are *computational***
- MDSD models are no „paperwork“, they ***are*** the solution which is translated into code automatically





Hochschule Ulm

The SmartSoft Component Model
Stable Interfaces



compose components

Application Builder

System–Level (S)

- build system
- reuse components
- black–box view on components

integrate algorithmas and libraries

Component Builder

Component–Level (C)

- build component
- use framework
- black–box view on framework

map component–model onto specific target platform

Framework Builder

Framework–Level (F)

- build framework
- abstract middleware and OS details
- black–box view on middleware / OS

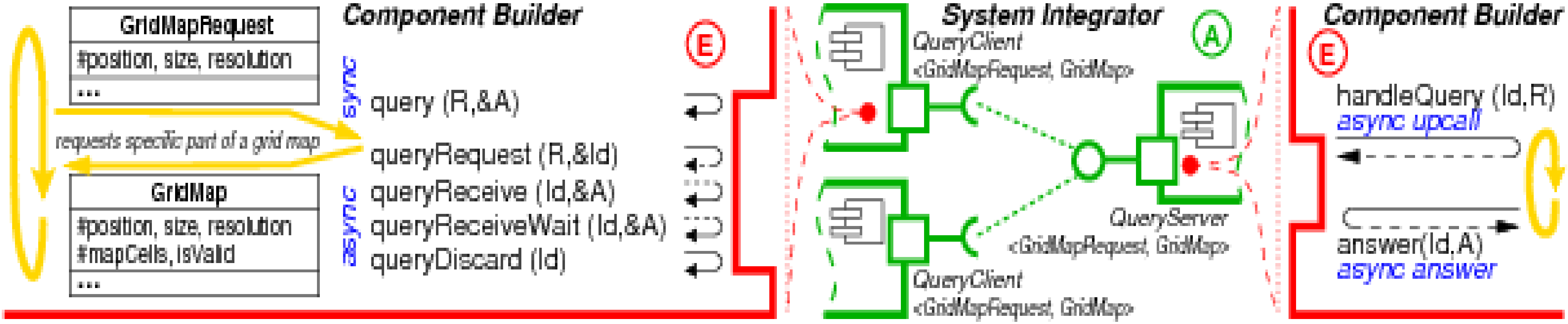
The SmartSoft Component Model
Stable Interfaces

R
A

Query Client

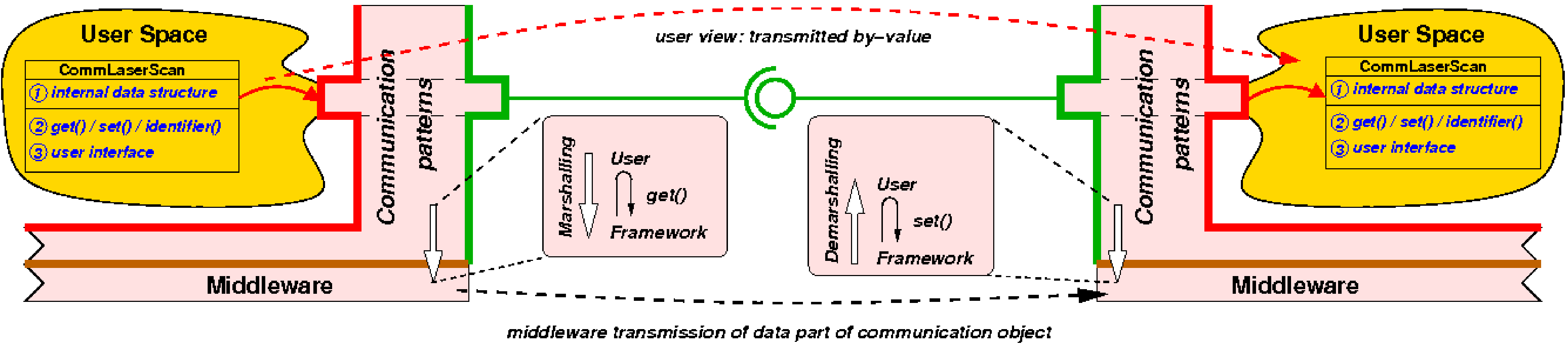
```
+ QueryClient(:SmartComponent*) throw(SmartError)
+ QueryClient(:SmartComponent*, server:const string&, service:const string&) throw(SmartError)
+ QueryClient(:SmartComponent*, port:const string&, slave:WiringSlave*) throw(SmartError)
+ ~QueryClient() throw() [virtual]

+ add(:WiringSlave*, port:const string&) : StatusCode throw()
+ remove() : StatusCode throw()

+ connect(server:const string&, service:const string&) : StatusCode throw()
+ disconnect() : StatusCode throw()

+ blocking(flag:const bool) : StatusCode throw()

+ query(request:const R&, answer:A&) : StatusCode throw()
+ queryRequest(request:const R&, id:QueryId&) : StatusCode throw()
+ queryReceive(id:const QueryId, answer:A&) : StatusCode throw()
+ queryReceiveWait(id:const QueryId, answer:A&) : StatusCode throw()
+ queryDiscard(id:const QueryId) : StatusCode throw()
```

SmartSoft Component Model
Stable Interfaces

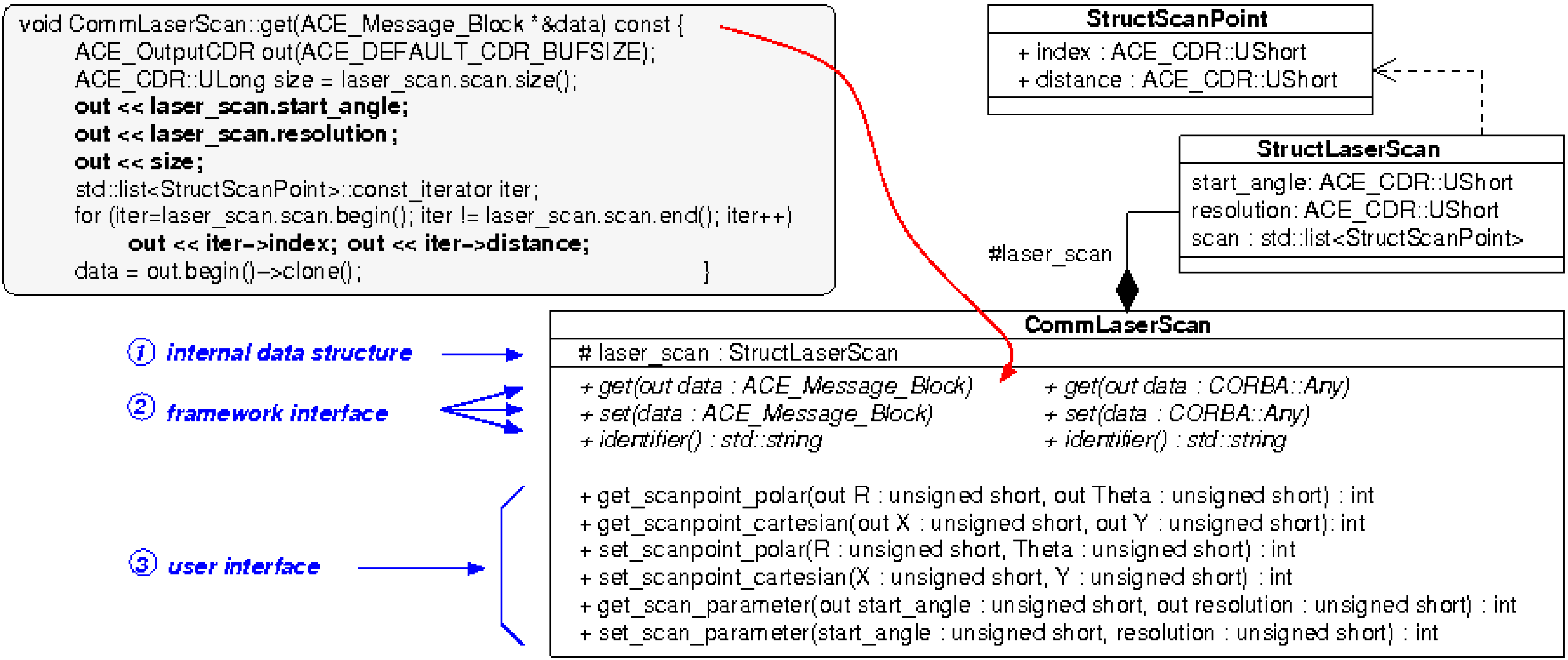


servicerobotik ZAFH

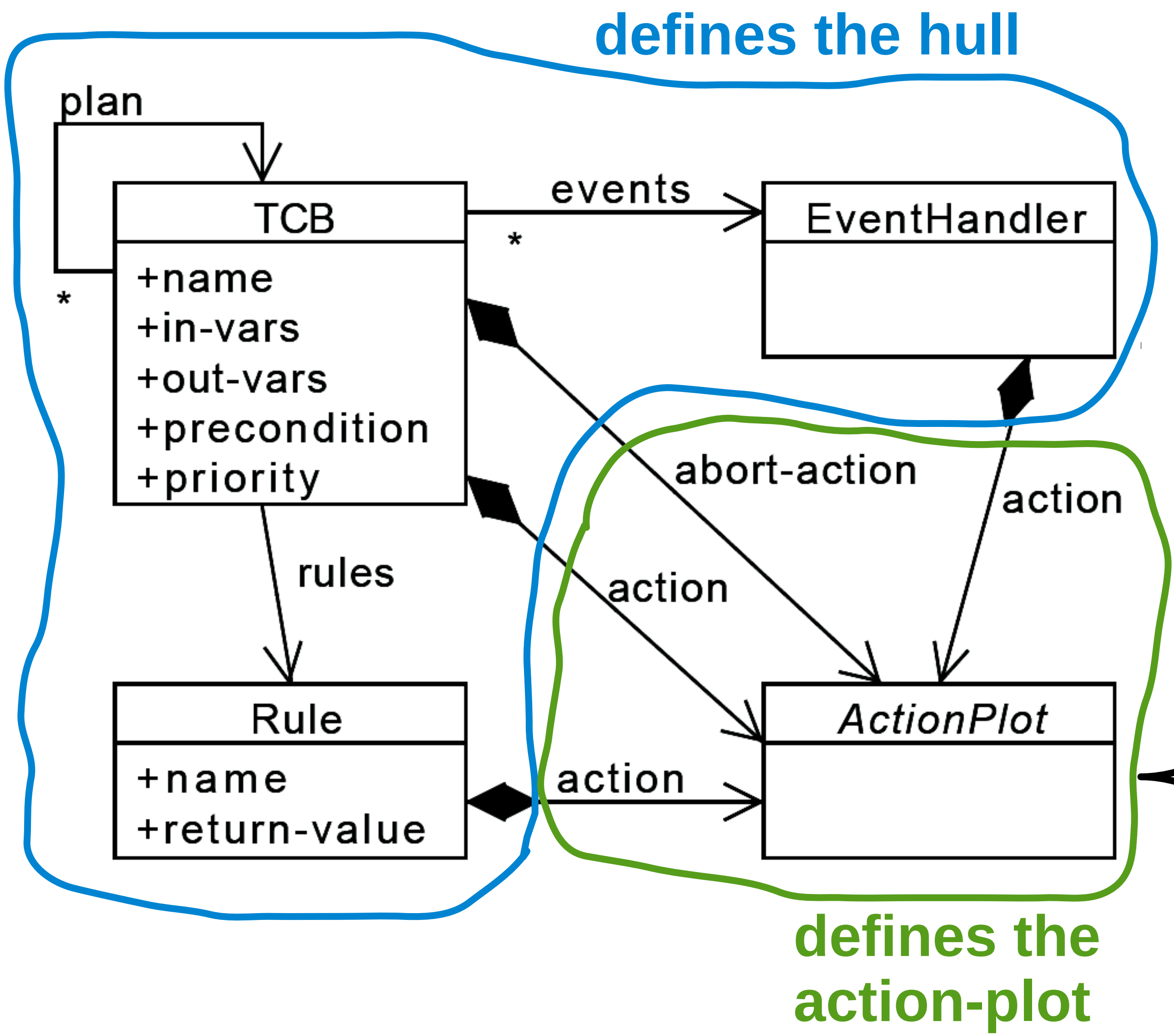
SmartSoft
Technical Details



SmartSoft
Technical Details



Run-Time: Managing Execution Variants
The SmartTCL Meta-Model



Actions are encapsulated by a hull:
- TCB
- EventHandler
- Rule

Lisp code (with restrictions):
- actions should not invoke blocking calls that take a long time relative to the reactivity which is expected from SmartTCL
- SmartTCL specific function:
  - tcl-param, tcl-state
  - tcl-wiring, tcl-query
  - tcl-activate-event
  - tcl-delete-event
  - ...

service robotik ZAFH

Run-Time: Managing Execution Variants

TCB Programming

Part VII

The *Hull* provides a stable structure that allows a black-box view on the action-plots and thus ensures reusability and composability → **Seperation of Roles**

defines the hull

```
(define-tcb (tcb-get-coffe-machine-cup-pose ?coffeMachineId => ?x ?y ?z)
  (rules nil)
  (precondition nil)
  (action (
            (format t "==========================>>> tcb-get-coffe-machine-cup-pose ~d ~%" '?coffeMachineId)
            ;; query coffe machine pose and cup-offset from KB
            (let* ((coffeeMachine (tcl-kb-query :key '(is-a id) :value '((is-a object)(id ?coffeMachineId))))
                   (coffeeMachinePose (get-value coffeeMachine 'pose))
                   (cup-offset (get-value coffeeMachine 'cup-offset))
                   (pose nil))
              ;; transform pose to point
              (setf pose (eval (append '(transformPoseToPoint) coffeeMachinePose cup-offset)))
              ;; bind output variables
              (tcl-bind-var :name '?x :value (first pose))
              (tcl-bind-var :name '?y :value (second pose))
              (tcl-bind-var :name '?z :value (third pose))
              '(SUCCESS ()))))))
```

defines the action-plot

To programm the Action-Plots the developers are free, for example, to do calculations, query for information from components or the KB.

Run-Time: Managing Execution Variants

TCB Selection at Run-Time