The Robotics Butler Scenario: Selected Details and Algorithms

Christian Schlegel
Prof. Dr.

M.Sc. Alex Lotz

M.Sc. Matthias Lutz

M.Sc. Dennis Stampfer

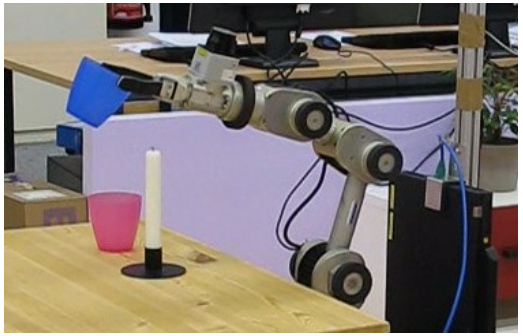
M.Sc. Andreas Steck

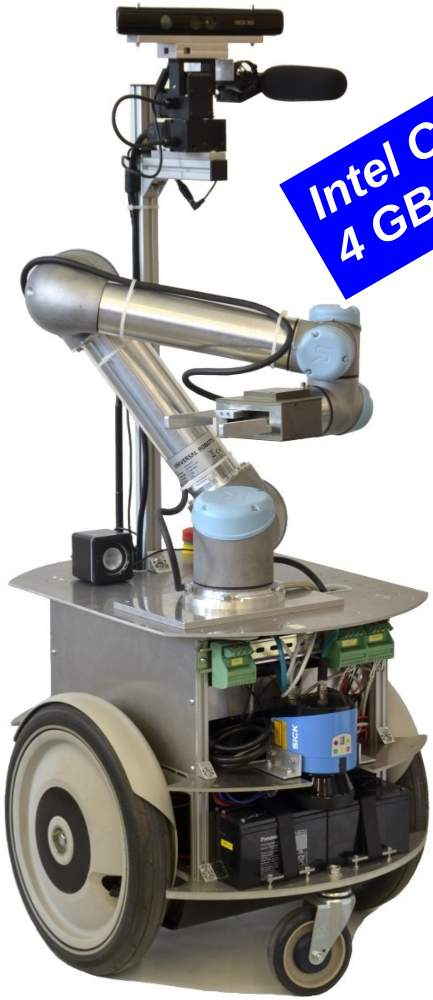
Dr. Siegfried Hochdorfer

servicerobotik ZAFH

Hochschule Ulm

Hochschule Ulm

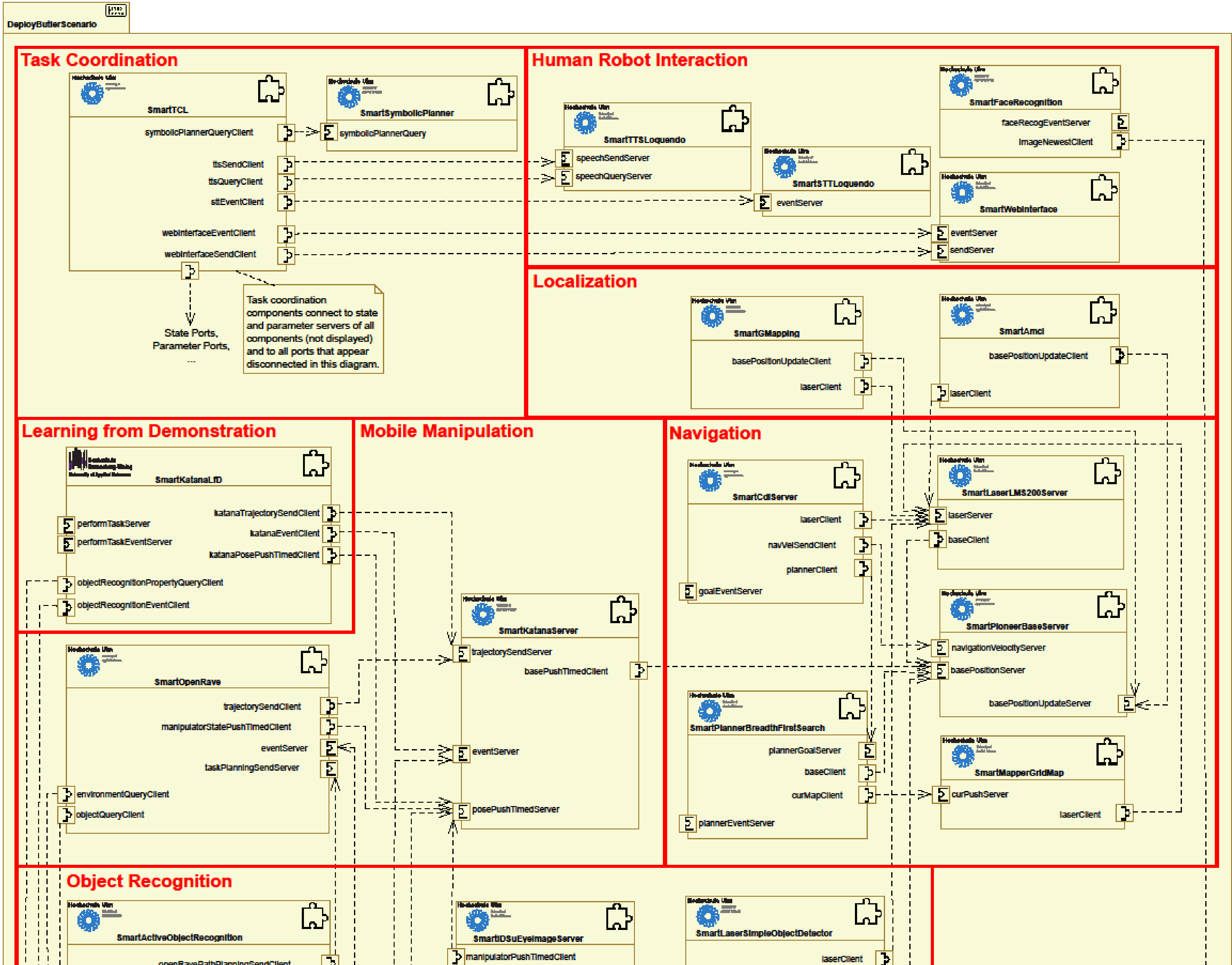




Intel Core2Duo P8800
4 GB RAM

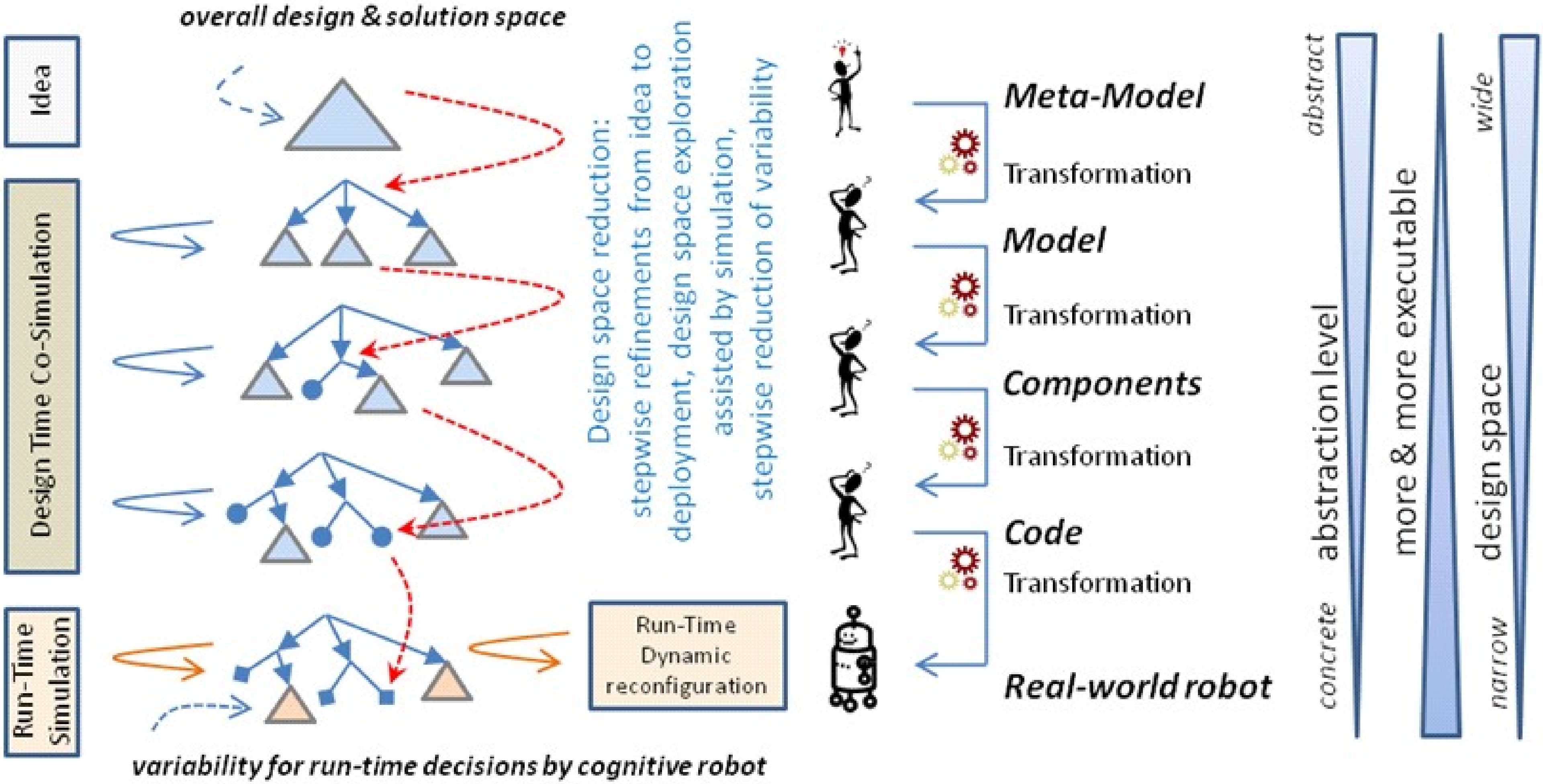
servicerobotik ZAFH

21.08.2012


DeployButlerScenario
Task Coordination
SmartTCL
symbolicPlannerQueryClient
ttsSendClient
ttsQueryClient
sttEventClient
webInterfaceEventClient
webInterfaceSendClient
SmartSymbolicPlanner
symbolicPlannerQuery
State Ports,
Parameter Ports,
...
Task coordination components connect to state and parameter servers of all components (not displayed) and to all ports that appear disconnected in this diagram.
Human Robot Interaction
SmartTTSLoquendo
speechSendServer
speechQueryServer
SmartSTTLoquendo
eventServer
SmartFaceRecognition
faceRecogEventServer
ImageNewestClient
SmartWebInterface
eventServer
sendServer
Localization
SmartGMapping
basePositionUpdateClient
laserClient
SmartAmcl
basePositionUpdateClient
laserClient
Learning from Demonstration
SmartKatanaLfD
performTaskServer
performTaskEventServer
katanaTrajectorySendClient
katanaEventClient
katanaPosePushTimedClient
objectRecognitionPropertyQueryClient
objectRecognitionEventClient
Mobile Manipulation
SmartKatanaServer
trajectorySendServer
basePushTimedClient
eventServer
posePushTimedServer
SmartOpenRave
trajectorySendClient
manipulatorStatePushTimedClient
eventServer
taskPlanningSendServer
environmentQueryClient
objectQueryClient
Navigation
SmartCdlServer
laserClient
navVelSendClient
plannerClient
goalEventServer
SmartLaserLMS200Server
laserServer
baseClient
SmartPioneerBaseServer
navigationVelocityServer
basePositionServer
basePositionUpdateServer
SmartPlannerBreadthFirstSearch
plannerGoalServer
baseClient
curMapClient
plannerEventServer
SmartMapperGridMap
curPushServer
laserClient
Object Recognition
SmartActiveObjectRecognition
openRavePathPlanningSendClient
SmartIDSuEyeImageServer
manipulatorPushTimedClient
SmartLaserSimpleObjectDetector
laserClient

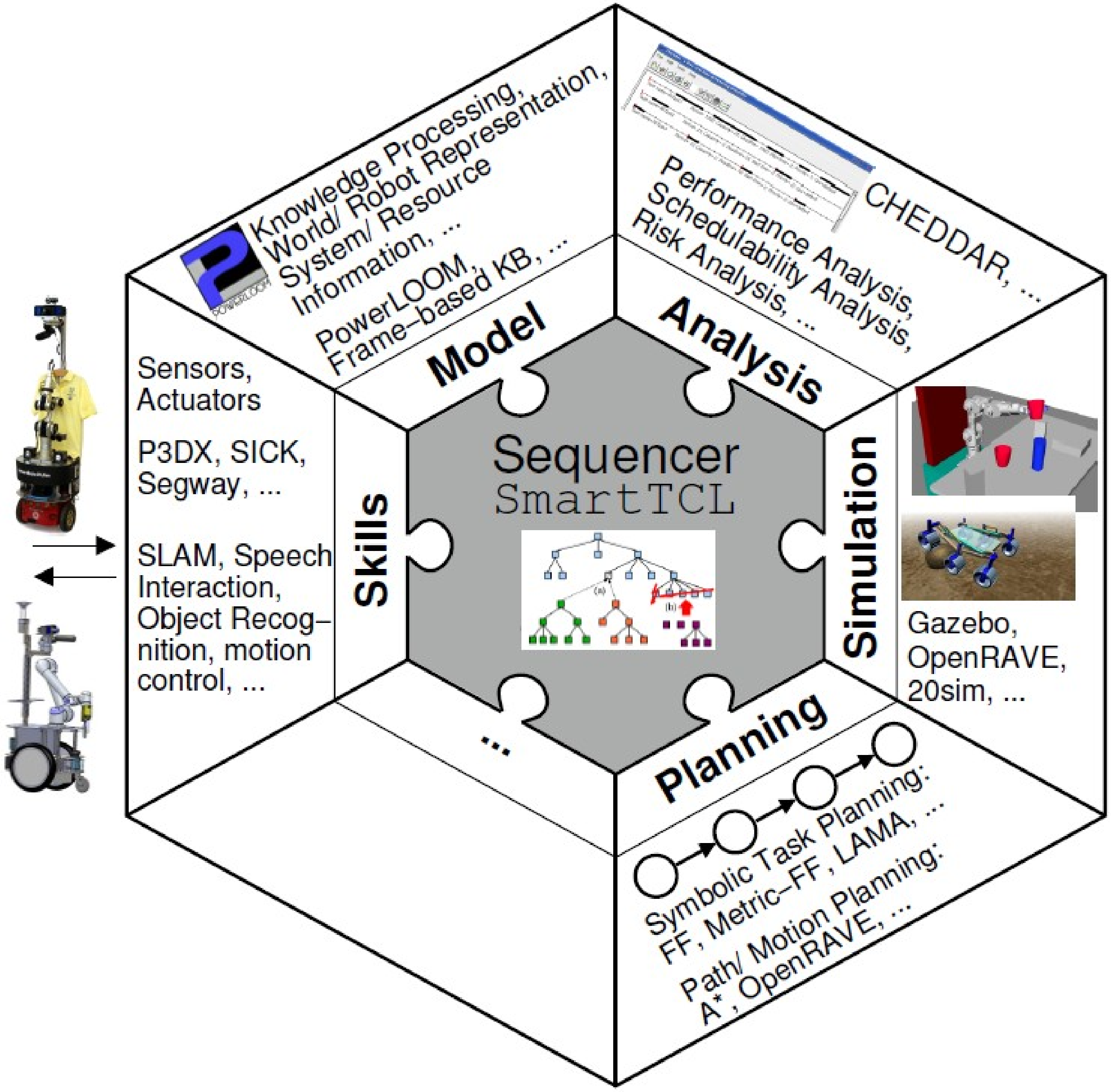
What is different in robotics and what we need...

- *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
- 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*



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

Managing Execution-Variants at Run-Time
Sequencer Orchestrates the System

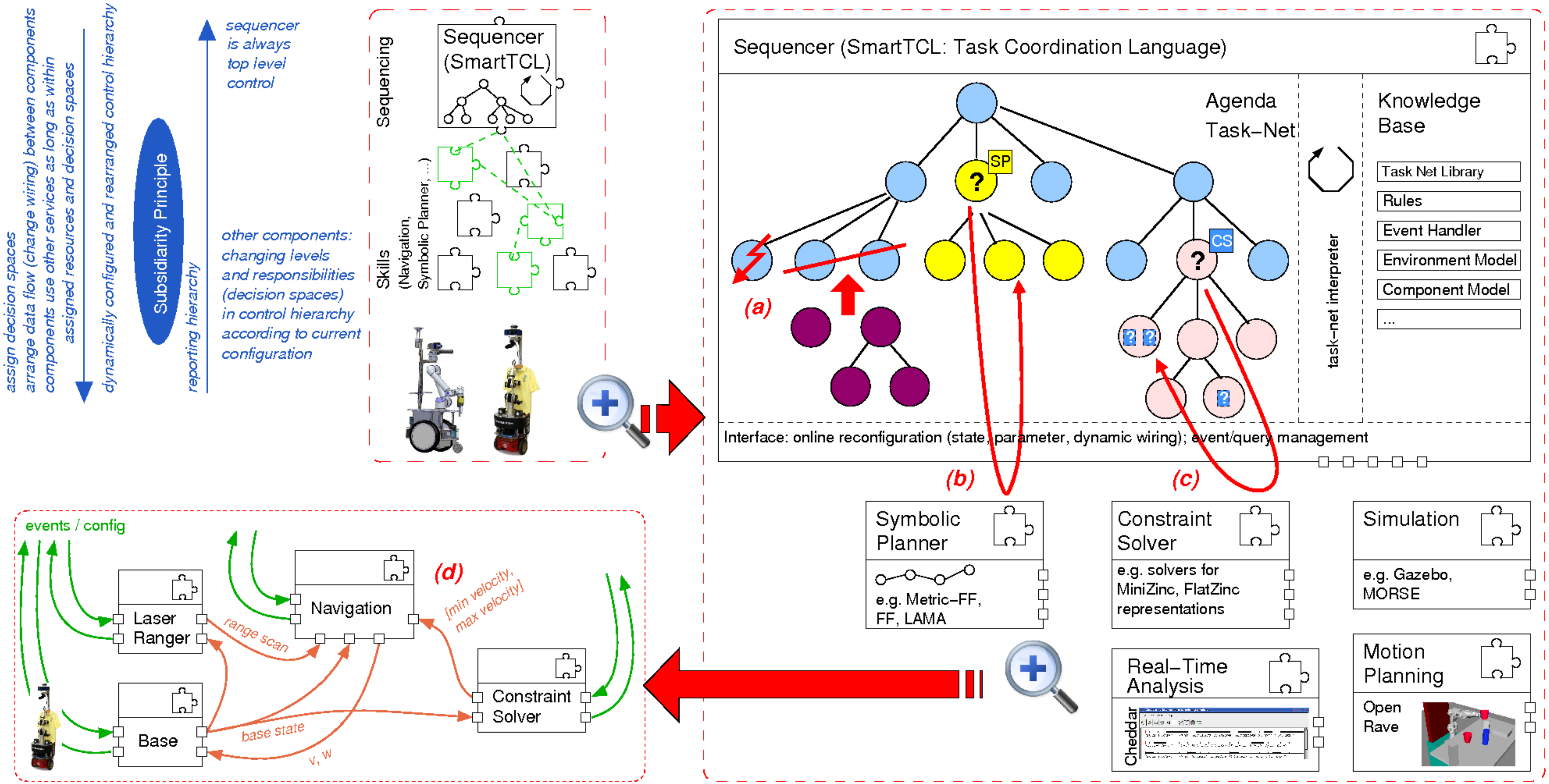


Sequencing Layer with SmartTCL:

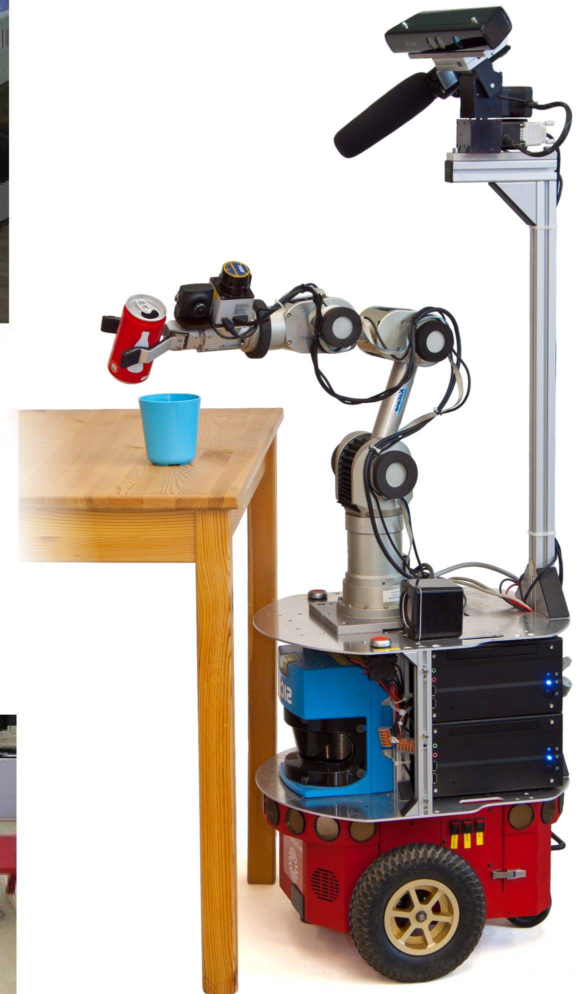
- bridges between continuous processing and event-driven task execution
- orchestrates the software components in the system
- assign decision spaces to components
- involve dedicated experts for run-time binding of designed variability
- coordinate analysis, simulation and planning capabilities

  - send parameters and configurations
  - switch components on/off to manage resources
  - change the wiring between the components
  - query information and wait for events
  - ...

Overall Architecture



The Robotics Butler Scenario



Details Task-Nets and Task Execution: SmartTCL

Use Cases?

Overall principles behind our work

Needs?

Benefits?

Goals? Achieve suitability for everyday life of service robots

Task-nets for conditional reative task execution

Model-driven software development for robotics

Learning from demonstration for manipulation

Technology?

Resource-aware SLAM

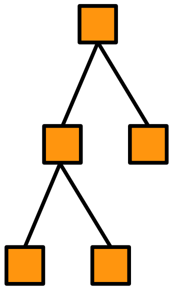
Active object recognition with information-driven sensor placement

Approach?

- focus on tools for systematic engineering of service robotic applications (e.g. MDSD)
  - separation of roles
  - separation of concerns
- focus on robust and efficient key functionalities and components
  - extending and merging so far separated techniques

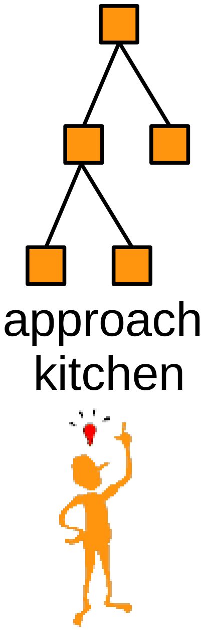
Details Task-Nets and Task Execution: SmartTCL

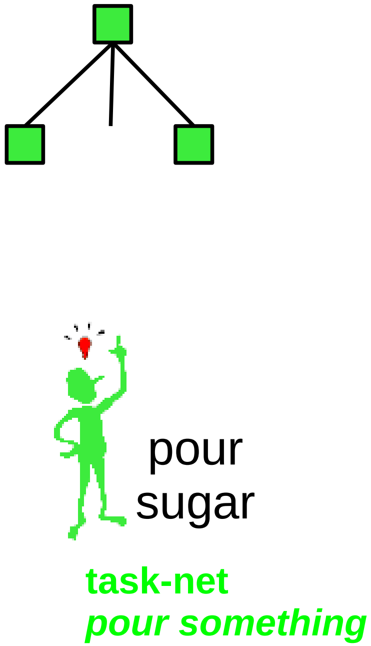
Composability, Reusability, Separation of roles (designer, robot)



Details Task-Nets and Task Execution: SmartTCL

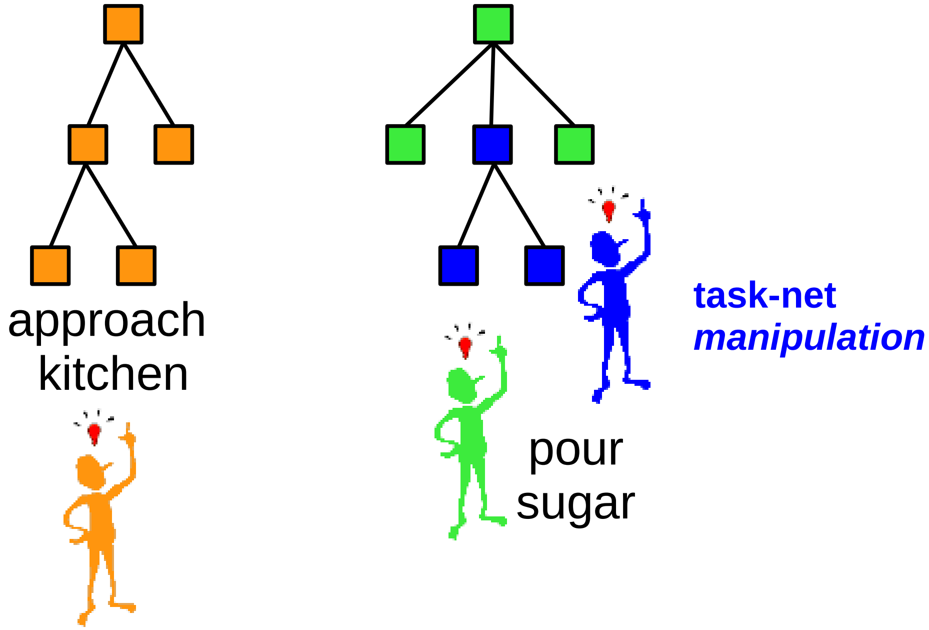
Composability, Reusability, Separation of roles (designer, robot)





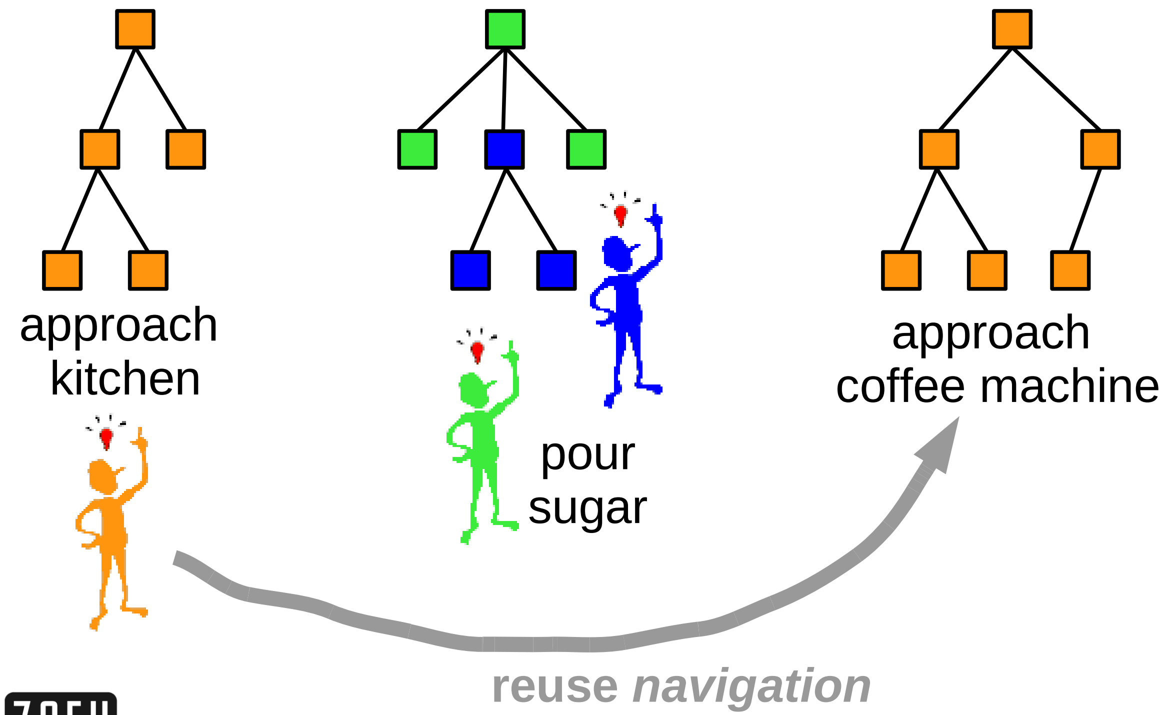
Details Task-Nets and Task Execution: SmartTCL

Composability, Reusability, Separation of roles (designer, robot)



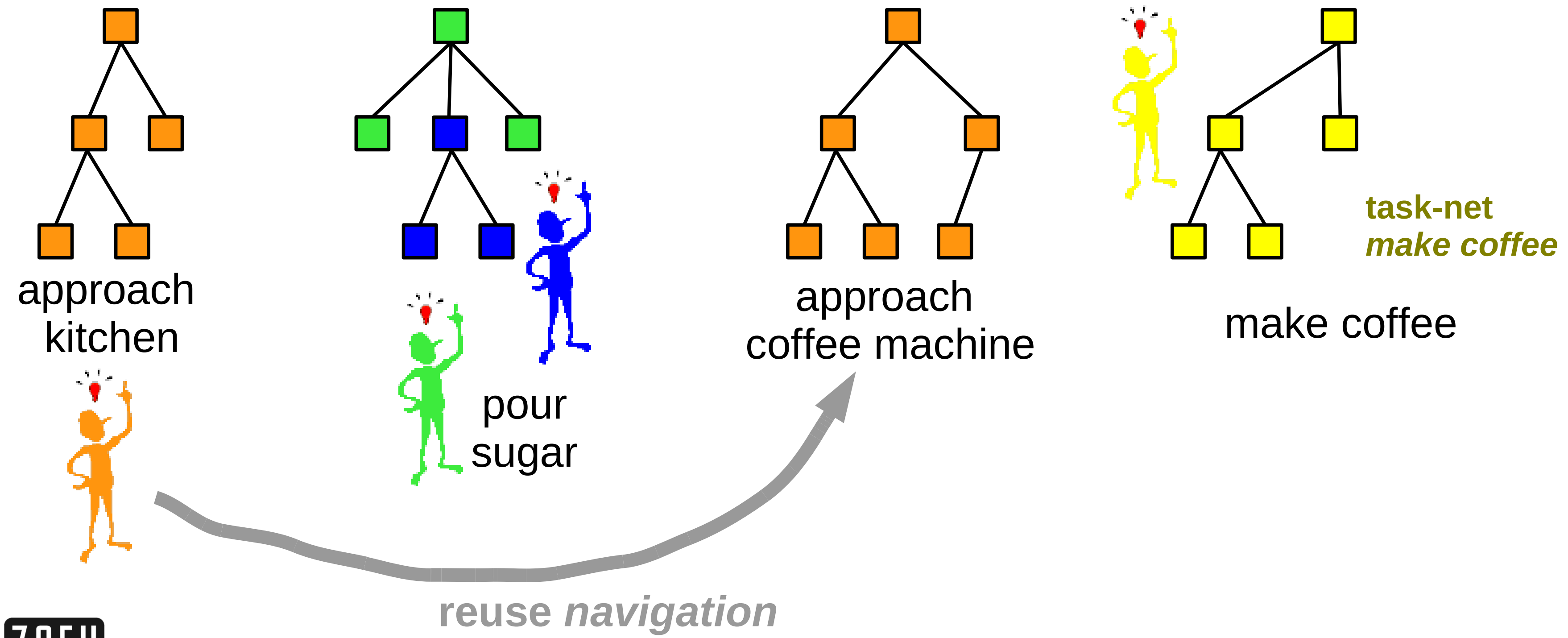
Details Task-Nets and Task Execution: SmartTCL

Composability, Reusability, Separation of roles (designer, robot)



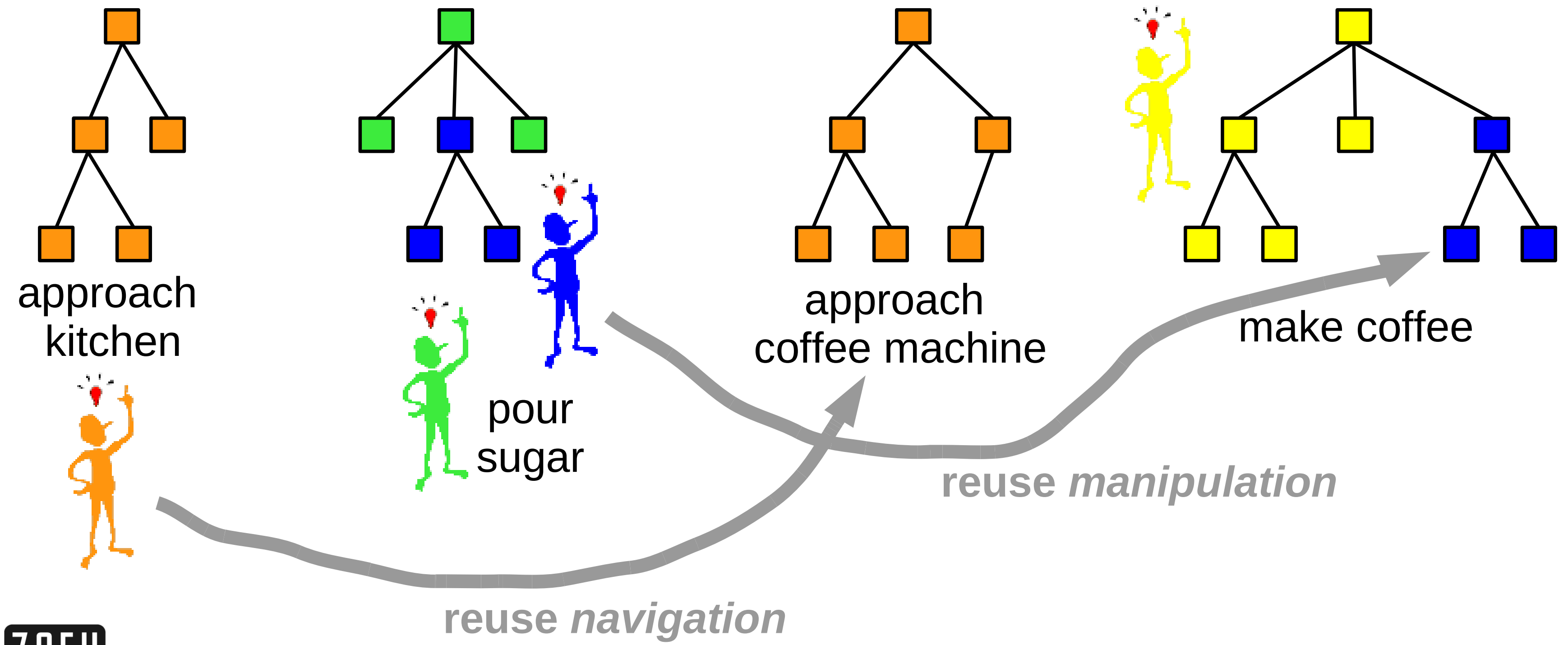
Details Task-Nets and Task Execution: SmartTCL

Composability, Reusability, Separation of roles (designer, robot)



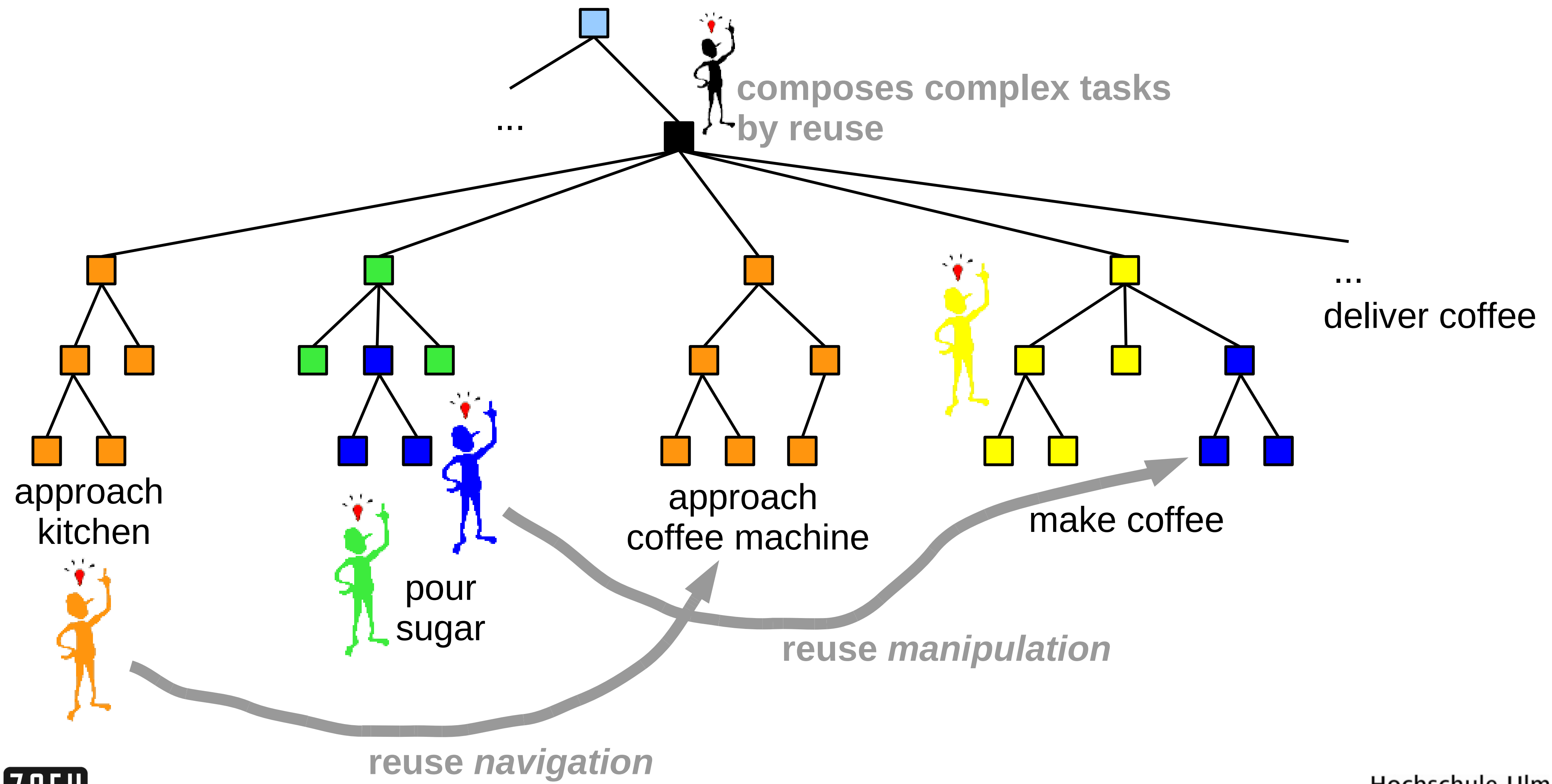
Details Task-Nets and Task Execution: SmartTCL

Composability, Reusability, Separation of roles (designer, robot)



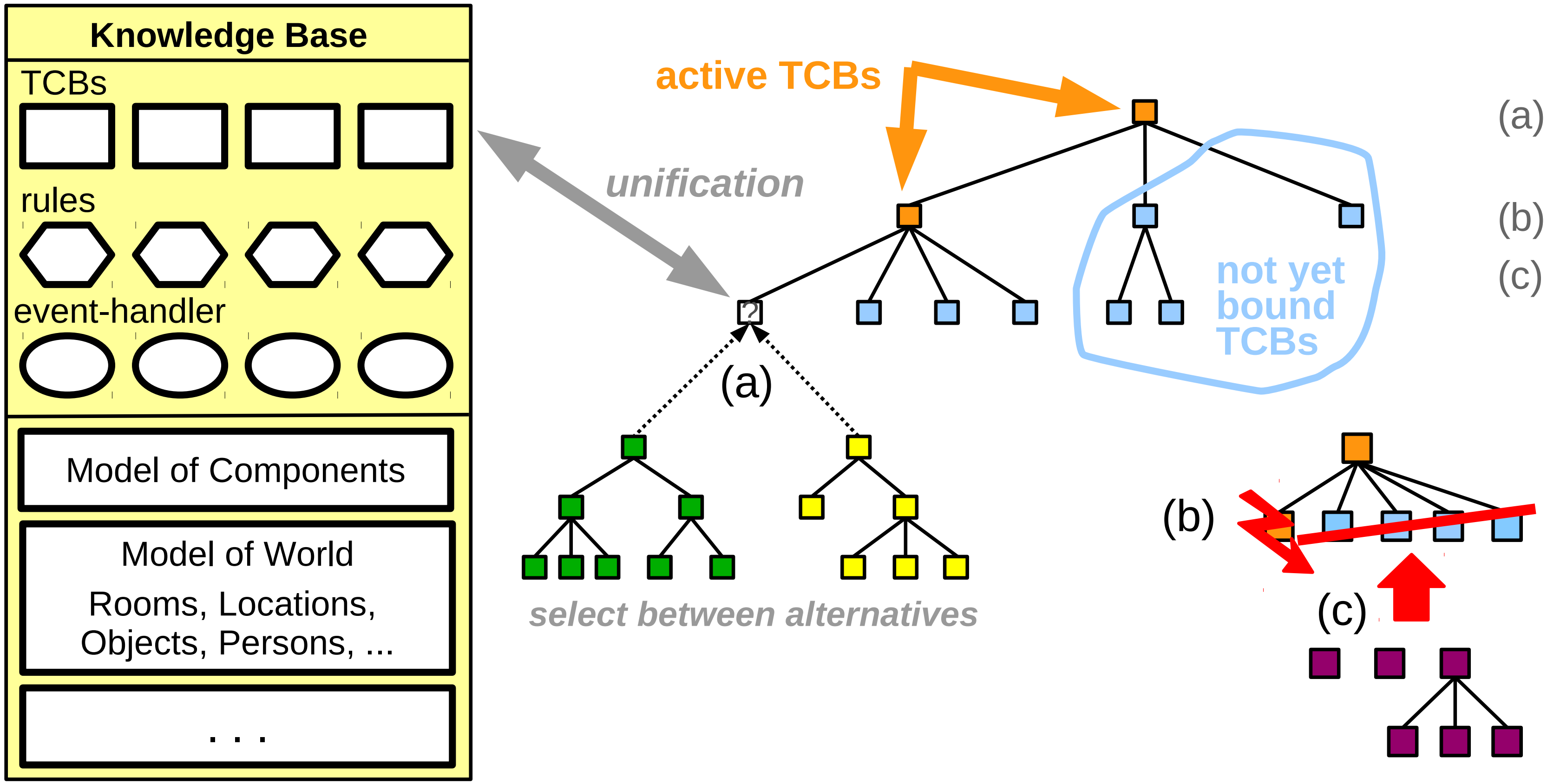
Details Task-Nets and Task Execution: SmartTCL

Composability, Reusability, Separation of roles (designer, robot)



Details Task-Nets and Task Execution: SmartTCL

Run-time: managing execution variants, TCB selection, late binding

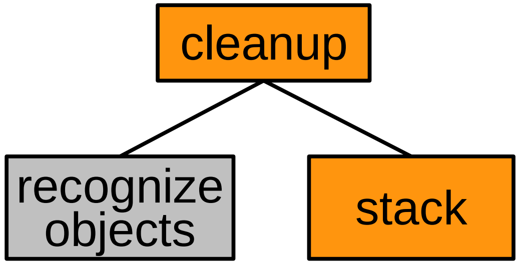


(a) select between alternatives at runtime
(b) handle contingencies
(c) delete, add or replace parts of the task-tree at runtime

- 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

Details Task-Nets and Task Execution: SmartTCL

Run-time: managing execution variants, TCB selection, late binding

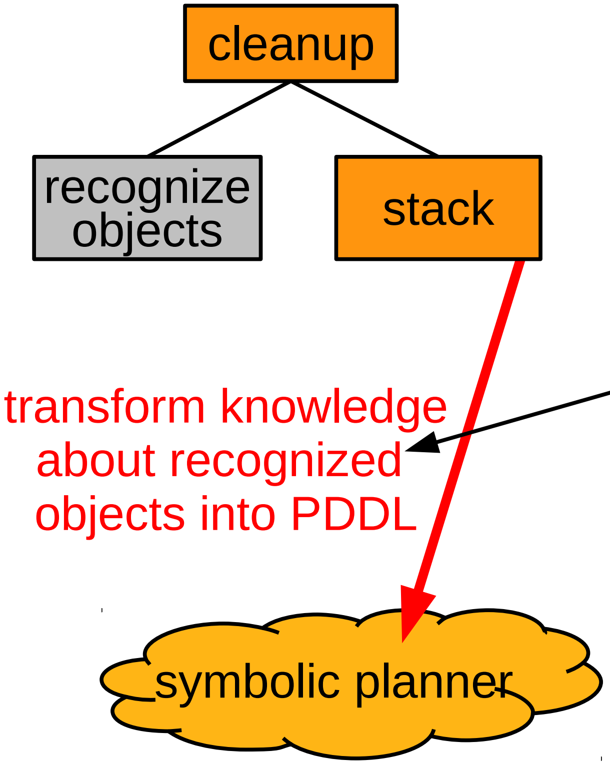


stores recognized objects in KB

the number of different variants how to stack the different objects is huge
→ calling a symbolic planner in that specific situation helps to manage the combinatorial explosion

Details Task-Nets and Task Execution: SmartTCL

Run-time: managing execution variants, symbolic planner



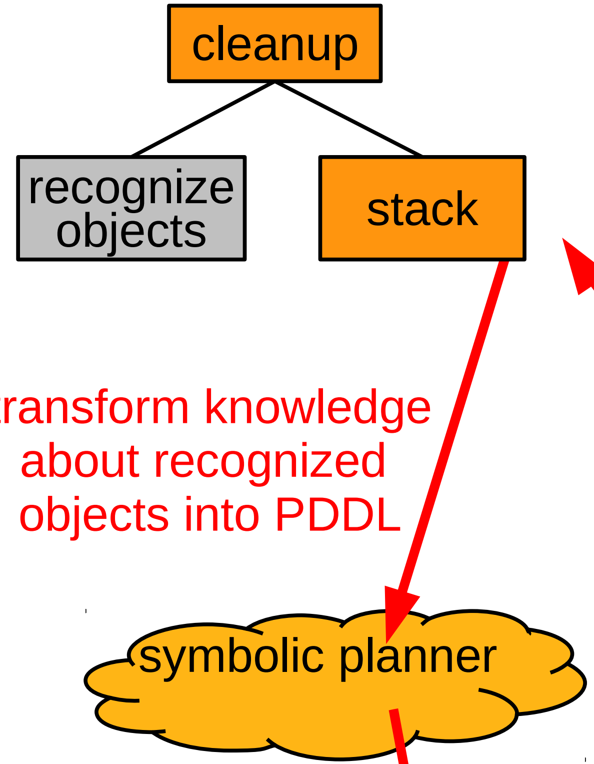
this is encoded within the action plot of the TCB *stack*:

- the recognized objects are queried from the KB
- the stable domain description (PDDL) as well as the situation specific fact description (PDDL) are created and forwarded to the symbolic planner

the *SmartSymbolicPlanner* component provides the service to call symbolic planners like ff, metric-ff, lama, ...

Details Task-Nets and Task Execution: SmartTCL

Run-time: managing execution variants, symbolic planner



transform knowledge about recognized objects into PDDL

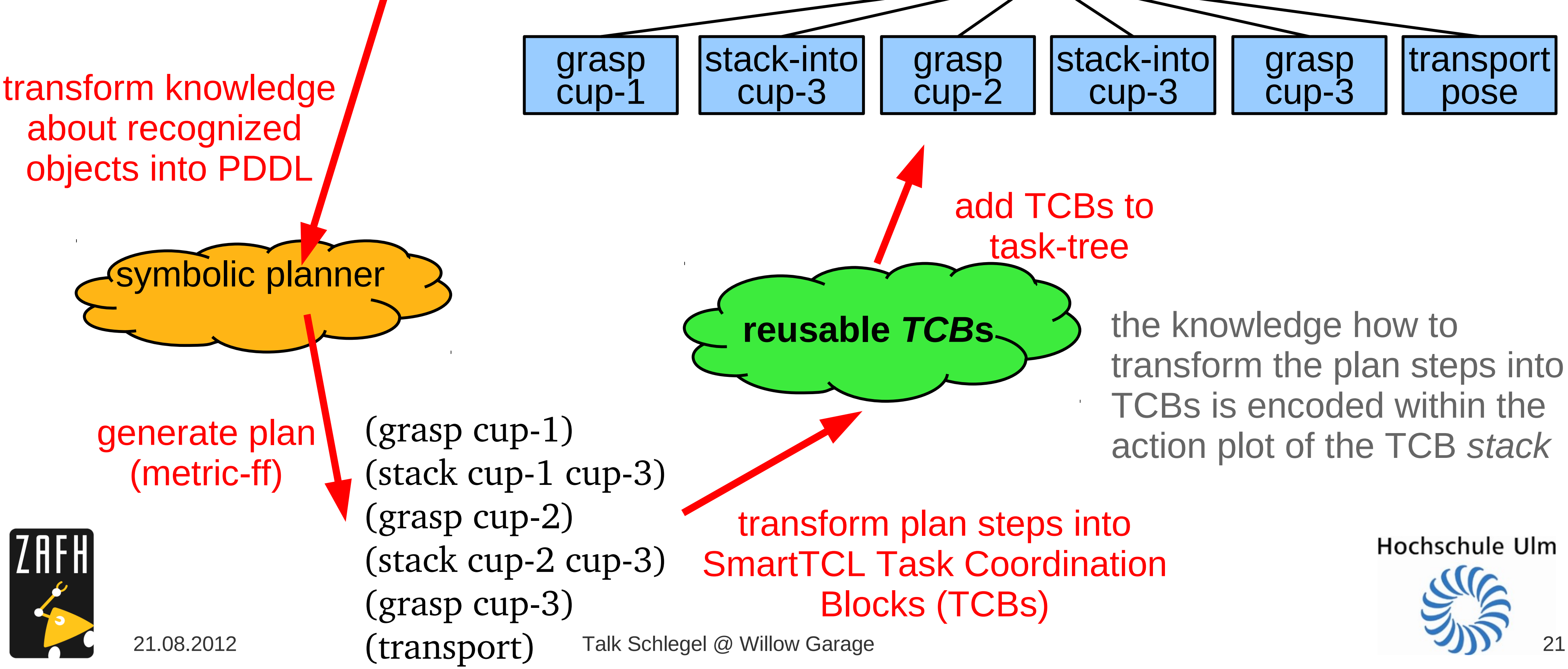
the generated plan is returned to the *stack* TCB

generate plan (metric-ff)

(grasp cup-1)
(stack cup-1 cup-3)
(grasp cup-2)
(stack cup-2 cup-3)
(grasp cup-3)
(transport)

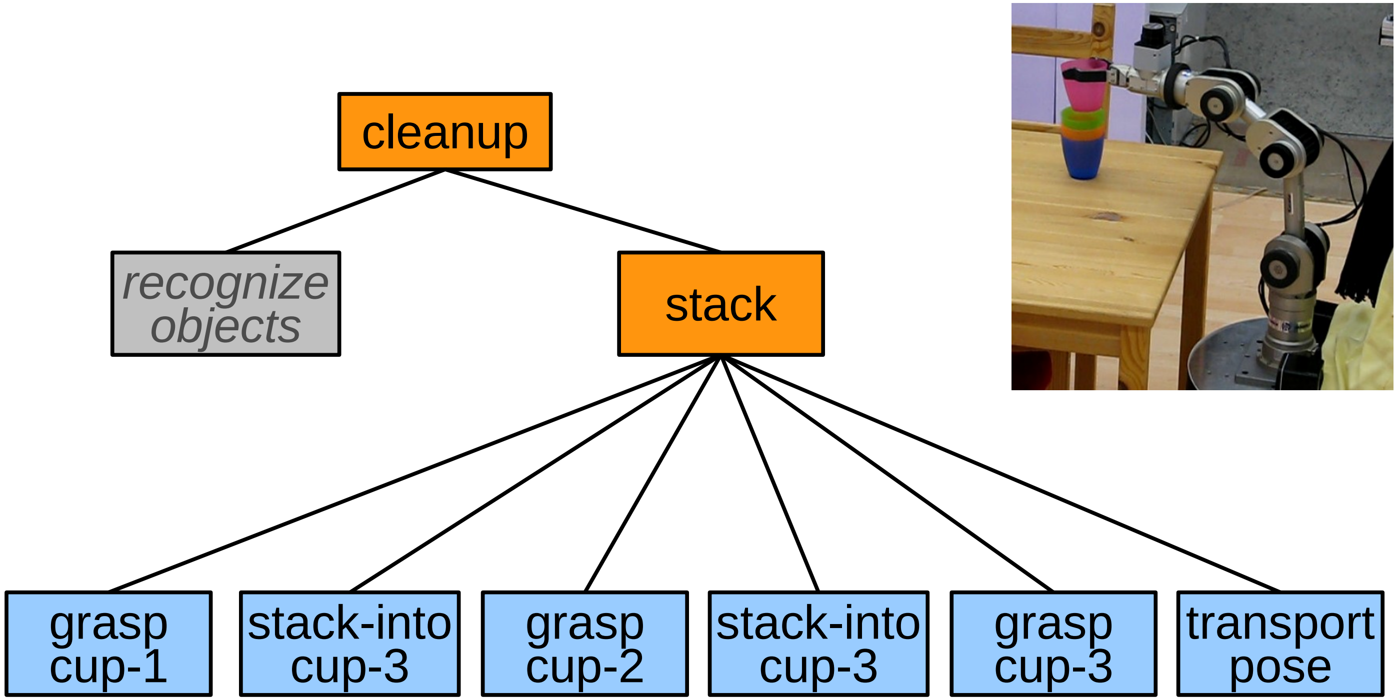
Details Task-Nets and Task Execution: SmartTCL

Run-time: managing execution variants, symbolic planner



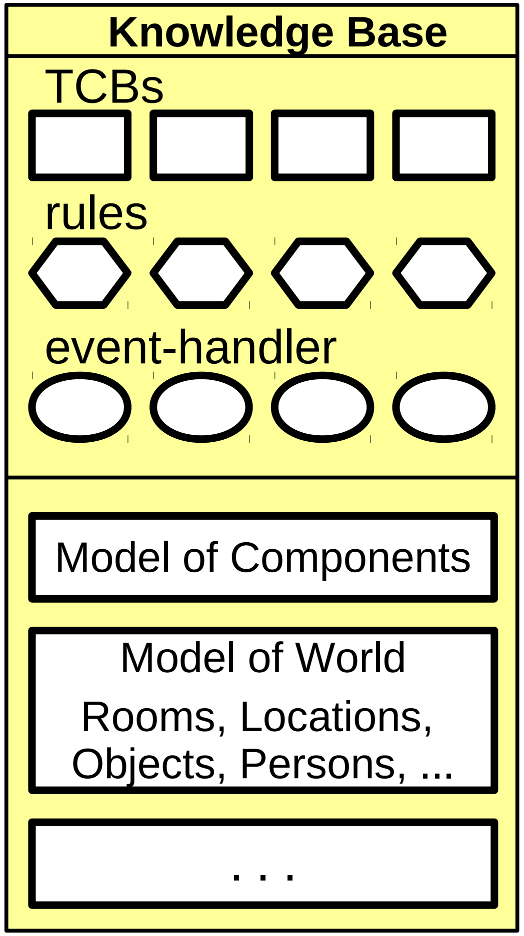
Details Task-Nets and Task Execution: SmartTCL

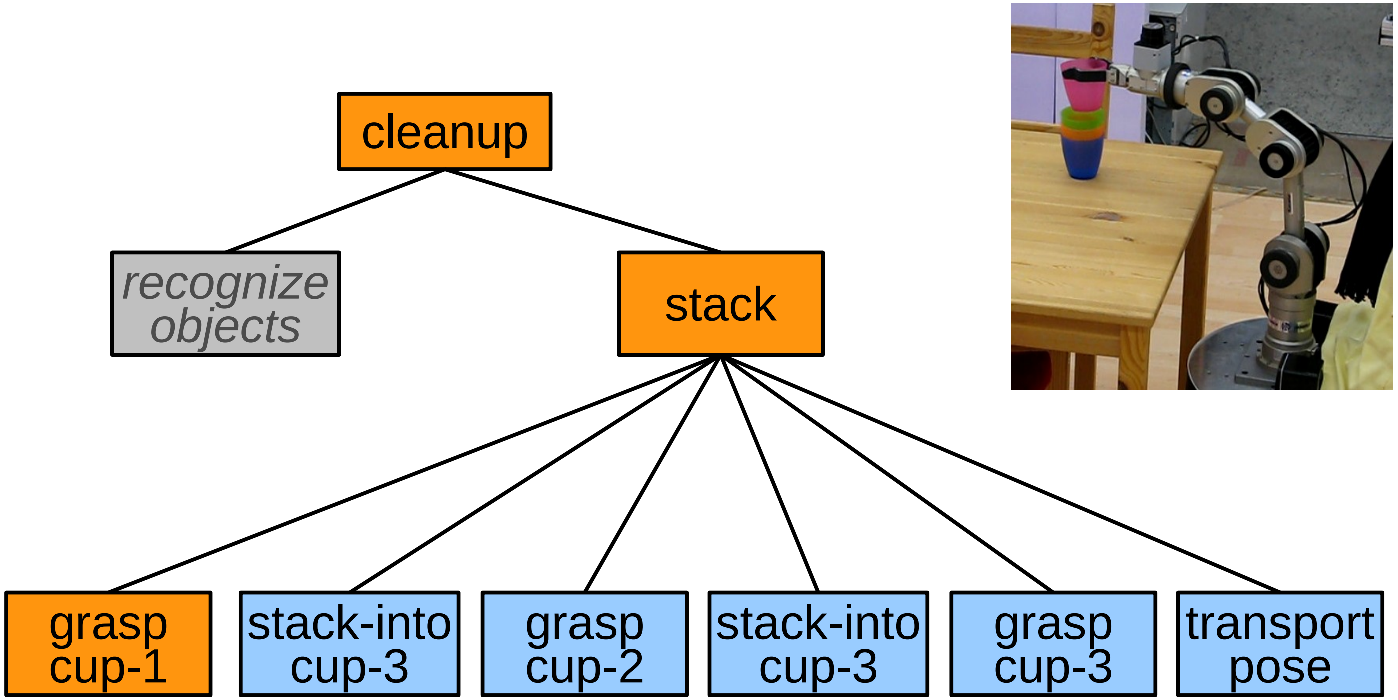
Run-time: managing execution variants, symbolic planner



Details Task-Nets and Task Execution: SmartTCL

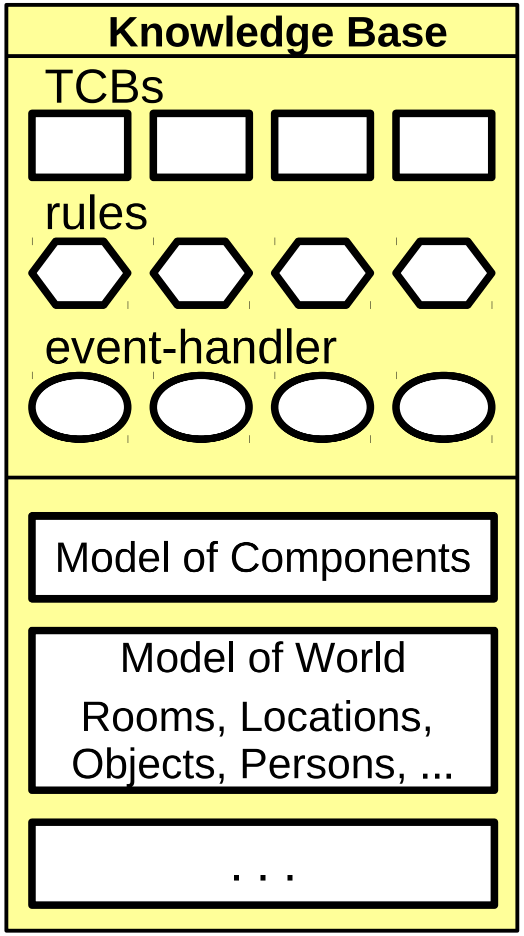
Run-time: managing execution variants, rules to recover from contingencies

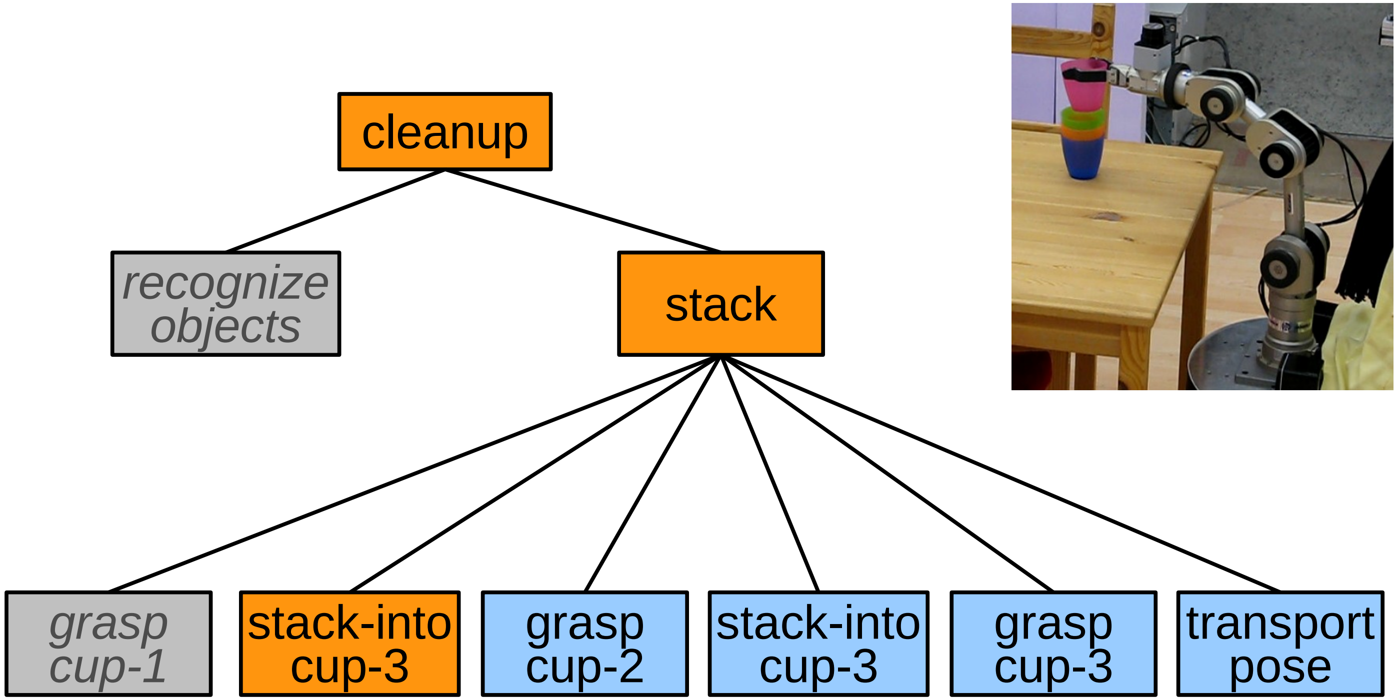




Details Task-Nets and Task Execution: SmartTCL

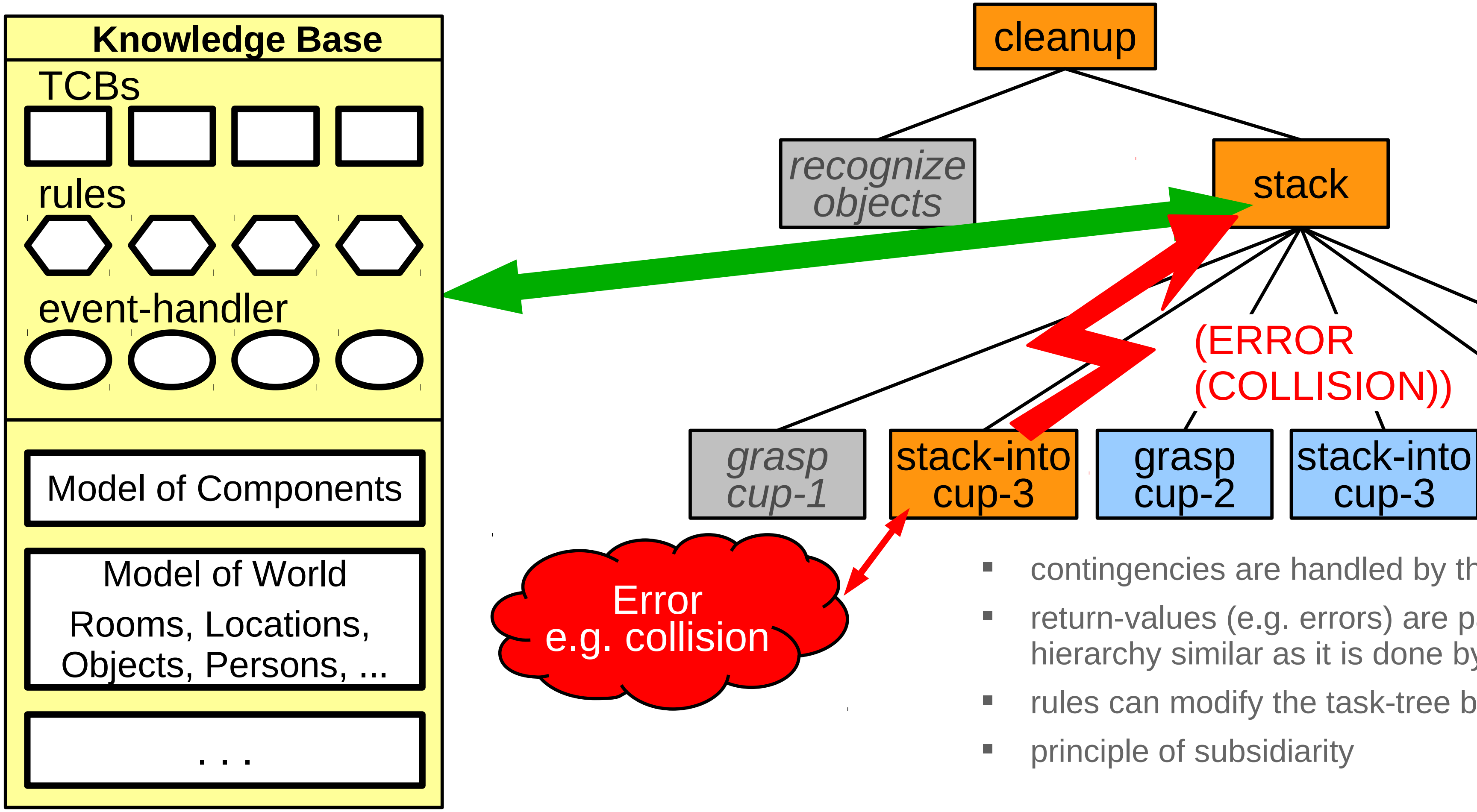
Run-time: managing execution variants, rules to recover from contingencies





Details Task-Nets and Task Execution: SmartTCL

Run-time: managing execution variants, rules to recover from contingencies

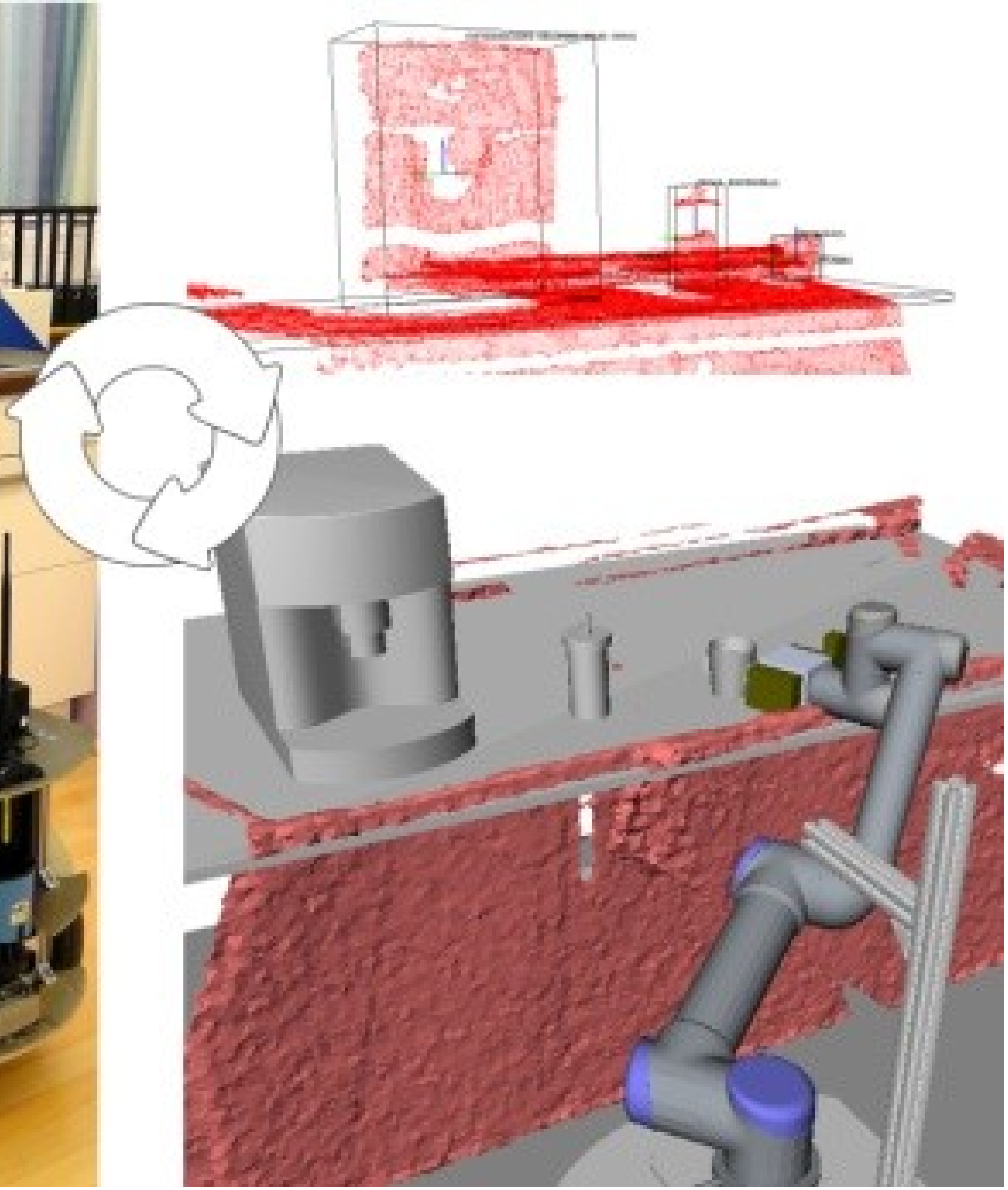


- contingencies are handled by the parent node
- return-values (e.g. errors) are passed along the node hierarchy similar as it is done by catch-throw mechanisms
- rules can modify the task-tree by adding or deleting nodes
- principle of subsidiarity

- *as rules are associated to the parent node (stack), the contingency handling works independent from the concrete plan which was generated*
- *rules “know” whether to repair the plan locally or to delete the plan and generate a new one.*

The Robotics Butler Scenario

Intel Core2Duo P8800
4 GB RAM



„A coffee with sugar, please“

Example: Pour sugar into coffee mug

```
; tcb-pour-sugar
(define-tcb (tcb-pour-sugar => ?cupId)
      (rules (rule-get-obj-no-obj-pour-sugar
             rule-sugar-grasp-obj-failed
             rule-sugar-grasp-obj-collision
             rule-sugar-position-object-failed
             rule-trigger-sugar-position-object-failed
             rule-manipulator-transport-no-path
             rule-manipulator-transport-collision))
      (action (
            (format t "==========================>>> tcb-pour-sugar ~%")))
      (plan (
            (tcb-load-env-into-openrave-location kitchen-unit-1)
            (tcb-obj-recog-setup kitchen-unit-1)
            (tcb-obj-recog kitchen-unit-1 => ?envID)
            (tcb-get-obj-id ?envID SUGAR-DISPENSER => ?sugarId)
            (tcb-get-obj-id ?envID IKEA-CUP-SOLBRAEND => ?cupId)
            (tcb-say "I pour the sugar into the cup.")
            (tcb-grasp-object real ?sugarId)
            (tcb-get-sugar-cup-pose ?cupId => ?x1 ?y1 ?z1)
            (tcb-position-object real ?sugarId ?x1 ?y1 ?z1 -20 nil t)
            (tcb-manipulator-angles => ?j1 ?j2 ?j3 ?j4 ?j5)
            (tcb-manipulator-pose-direct ?j1 ?j2 ?j3 ?j4 5.5 NO_OP) ;; 5.5 = 315 Grad
            (tcb-get-obj-pose-kb ?sugarId => ?x2 ?y2 ?z2)
            (tcb-position-object real ?sugarId ?x2 ?y2 ?z2 0 t t)
            (tcb-grasp-object real ?cupId)
            (tcb-manipulator-pose transport))))
```

```
(define-rule (rule-sugar-position-object-failed)
      (tcb (tcb-position-object real ?obj-id ?x ?y ?z ?roll ?open ?level))
      (return-value (ERROR (POSITIONING FAILED)))
      (action (
            (format t "RULE: (ERROR (POSITIONING FAILED)) ~%")
            (tcl-delete-plan)
            (tcl-push-plan :plan `(
                  (tcb-clear-env-openrave)
                  (tcb-load-env-into-openrave-location kitchen-coffee-1)
                  (tcb-get-obj-pose-kb ?sugarId => ?x2 ?y2 ?z2)
                  (tcb-position-object real ?sugarId ?x2 ?y2 ?z2 0 t t)
                  (tcb-trigger-return (ERROR (POUR-SUGAR-FAILED)))))
            '(ERROR (POUR-SUGAR-FAILED)))))
```

Example: Pour sugar into coffee mug

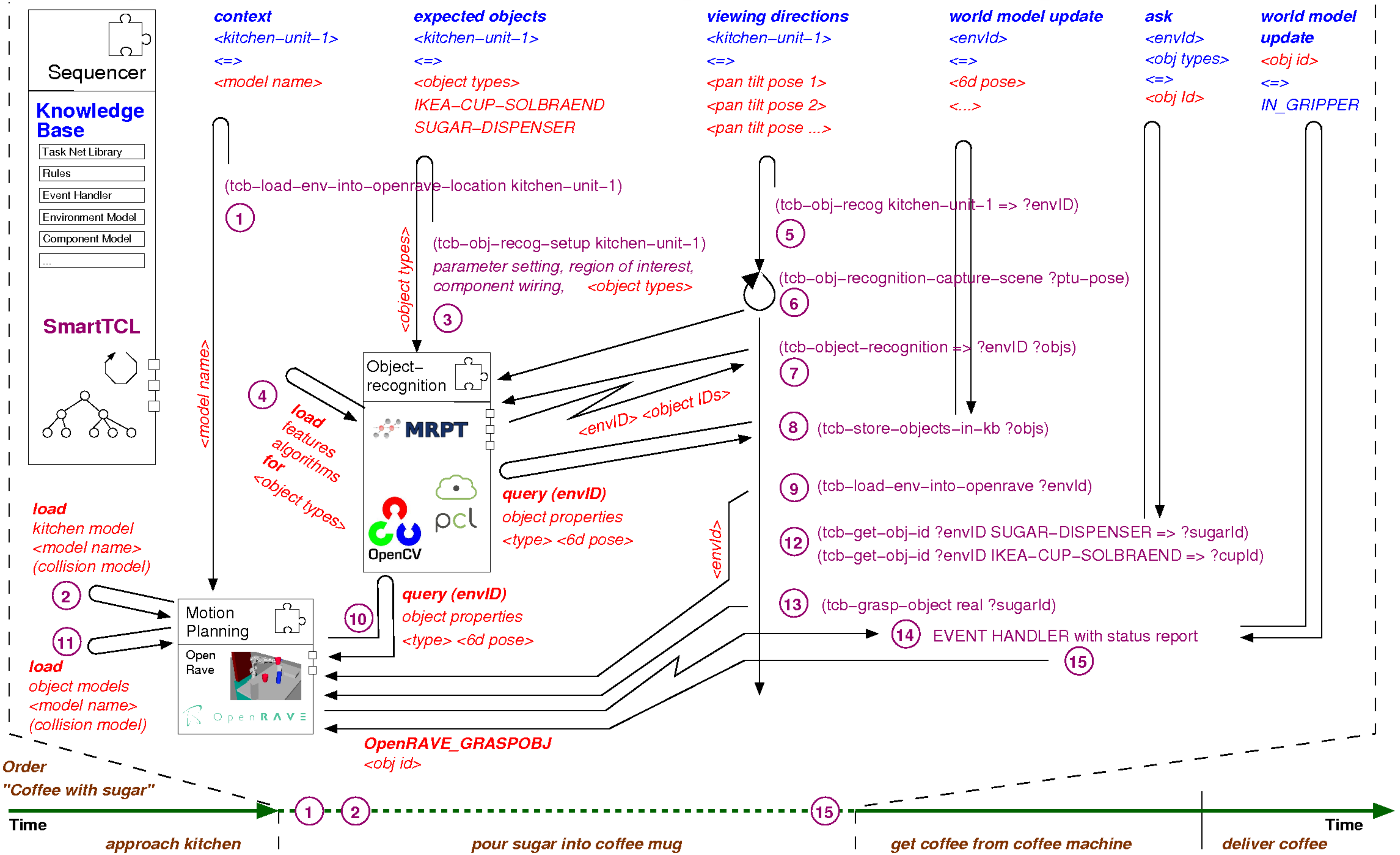
```
;; tcb-grasp-object
(define-tcb (tcb-grasp-object real ?obj-id)
  (action (
    (format t "==========>>> tcb-grasp-object real ~d ~%" '?obj-id)
    (let* ((obj (tcl-kb-query :key '(is-a id)
                              :value '((is-a object)(id ?obj-id))))
           (pose (get-value obj 'pose))
           (simple-grasping (get-value obj 'simple-grasping))
           (speech (get-value obj 'speech)))
    (tcl-state :server 'openrave :state "neutral")
    (format t "simple grasping ~s~%" simple-grasping)
    (tcl-param :server 'openrave :slot 'GRASPING_SIMPLE
                                 :value simple-grasping)
    (tcl-param :server 'openrave :slot 'PARALLELIZATION_ON)
    (tcl-state :server 'manipulator :state "trajectory")
    (tcl-state :server 'openrave :state "trajectory")
    (tcl-activate-event :name 'evt-traj
                        :handler 'handler-grasping
                        :server 'manipulator
                        :service 'manipulatorevent
                        :mode 'continuous)
    (tcl-activate-event :name 'evt-grasp-openrave
                        :handler 'handler-grasping-openrave
                        :server 'openrave
                        :service 'trajectoryevent
                        :mode 'continuous)
    (format t "pose: ~s ~%" pose)
    (tcl-send :server 'openrave
              :service 'trajectory
              :param (append (list 'POSE) pose (list 'OPEN_BEFORE_CLOSE_AFTER)))
    '(SUCCESS ())))))
```

```
;; handler-grasping
(define-event-handler (handler-grasping)
  (action (
    (format t "=========================>>> HANDLER GRASPING ~s ~%~%" (tcl-event-message))
    (cond
      ;; ok
      ((equal (tcl-event-message) '(goal reached))
        (format t "=========================>>> goal reached !!! obj-id ~s ~%" '?obj-id)
        (tcl-state :server 'openrave :state "neutral")
        (tcl-param :server 'openrave :slot 'OPENRAVE_GRASPOBJ :value '?obj-id)
        (tcl-kb-update :key '(is-a id)
                       :value `((is-a OBJECT) (id ,?obj-id) (status IN_GRIPPER)))
        (tcl-abort)
        '(SUCCESS ()))

      ;; slipped out
      ((equal (tcl-event-message) '(goal reached gripper empty))
        (format t "=========================>>> goal reached gripper empty !!! obj-id ~s ~%" '
        (tcl-send :server 'tts
                  :service 'say
                  :param (format nil "Oh sorry, it seems that the object slipped out of my gri
        (tcl-state :server 'openrave :state "neutral")
        (tcl-param :server 'openrave :slot 'OBJ_DELETE :value '?obj-id)
        (tcl-kb-update :key '(is-a id)
                       :value `((is-a OBJECT) (id ,?obj-id) (status NOT_GRASPABLE)))
        (tcl-abort)
        '(ERROR (GRASPING FAILED)))

      ;; collision
      ((equal (tcl-event-message) '(collision))
        (format t "=========================>>> collision detected !!! obj-id ~s ~%" '?obj-id)
        (tcl-send :server 'tts
                  :service 'say
                  :param (format nil "Oh sorry, it seems that I collided with an object." ))
        (tcl-kb-update :key '(is-a id)
                       :value `((is-a OBJECT) (id ,?obj-id) (status NOT_GRASPABLE)))
        (tcl-abort)
        '(ERROR (GRASPING COLLISION)))

      ;; out of range
      ((equal (tcl-event-message) '(value out of range))
```

servicerobotik ZAFH

Hochschule Ulm

Example: World model / Update / Management



servicerobotik ZAFH

Hochschule Ulm

Details Active Object Recognition

Use Cases?

Task-nets for conditional
reative task execution

Overall principles
behind our work

Needs?

Model-driven software
development for robotics

Benefits?

Learning from
demonstration for
manipulation

Technology?

Goals?

Achieve suitability
for everyday life of
service robots

Resource-aware SLAM

Active object recognition with information-driven sensor placement

Approach?

- focus on tools for systematic engineering of service robotic applications (e.g. MDSD)
  - separation of roles
  - separation of concerns
- focus on robust and efficient key functionalities and components
  - extending and merging so far separated techniques

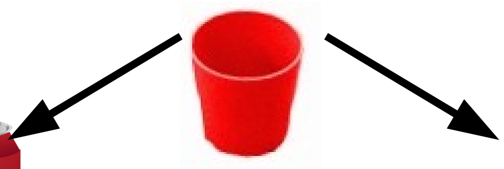
Multi-Modal Object Recognition

Robust object recognition is a mandatory prerequisite for many applications

- Object recognition means...
  - Detecting possible objects
  - Identifying objects
  - Know their position and orientation
- Robust in our terms means...
  - Reliable
  - Fast (enough)
  - Situation dependent
- Difficult in everyday environments
  - Real challenge: similar objects

Combining algorithms

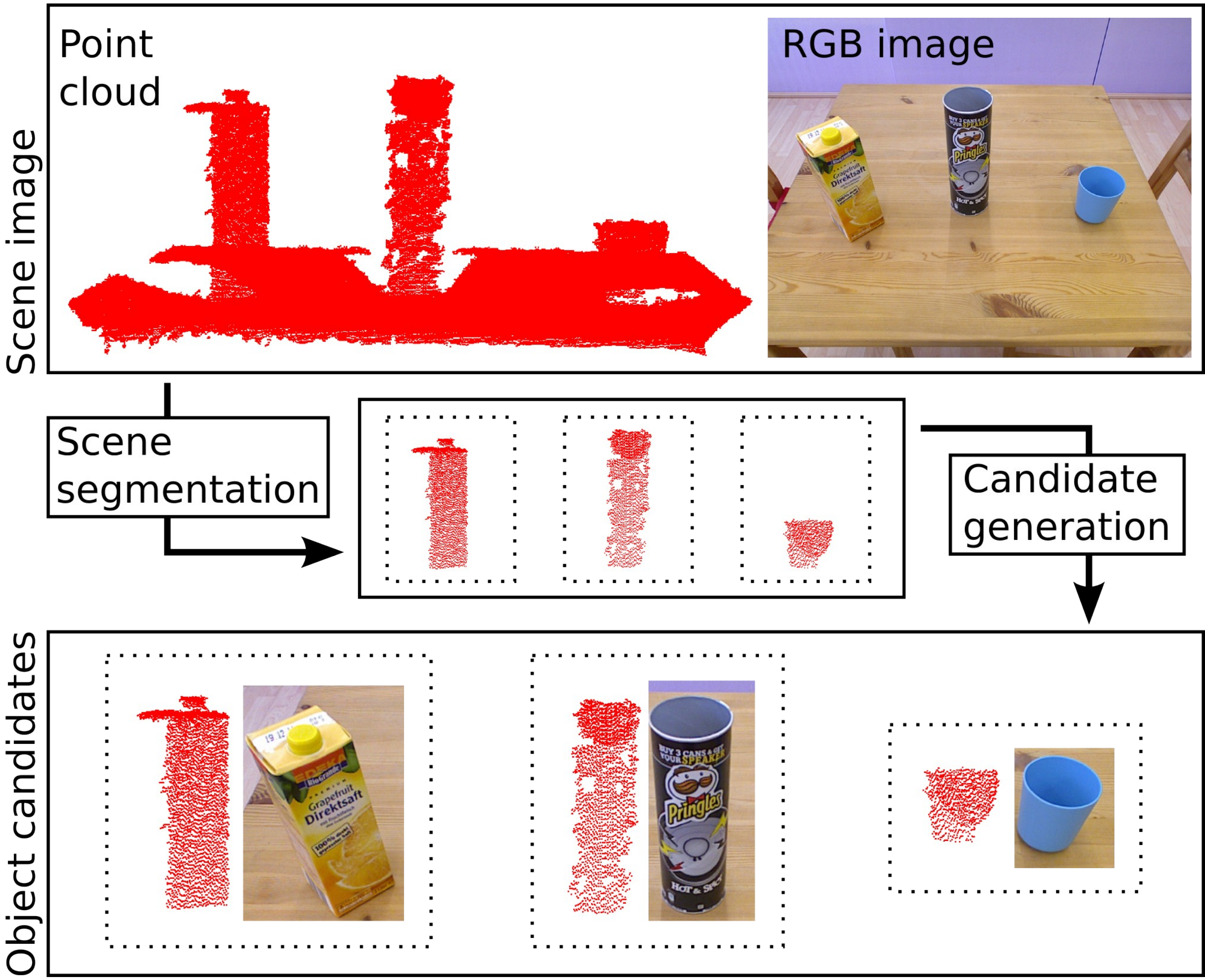
- Recognition by comparing features
  - Color
  - Texture
  - Shape
- Many different classes of objects
  - Requires many different features
- Recognition by combining features
  - Improves robustness



Multi-Modal Object Recognition

- Microsoft Kinect Kamera
  - Depth information for every pixel (3D pointcloud)
  - Color image
  - Cheap

- Detecting objects
  - Detecting table top in pointcloud
  - Cluster points above table top
  - Cut objects from RGB scene image



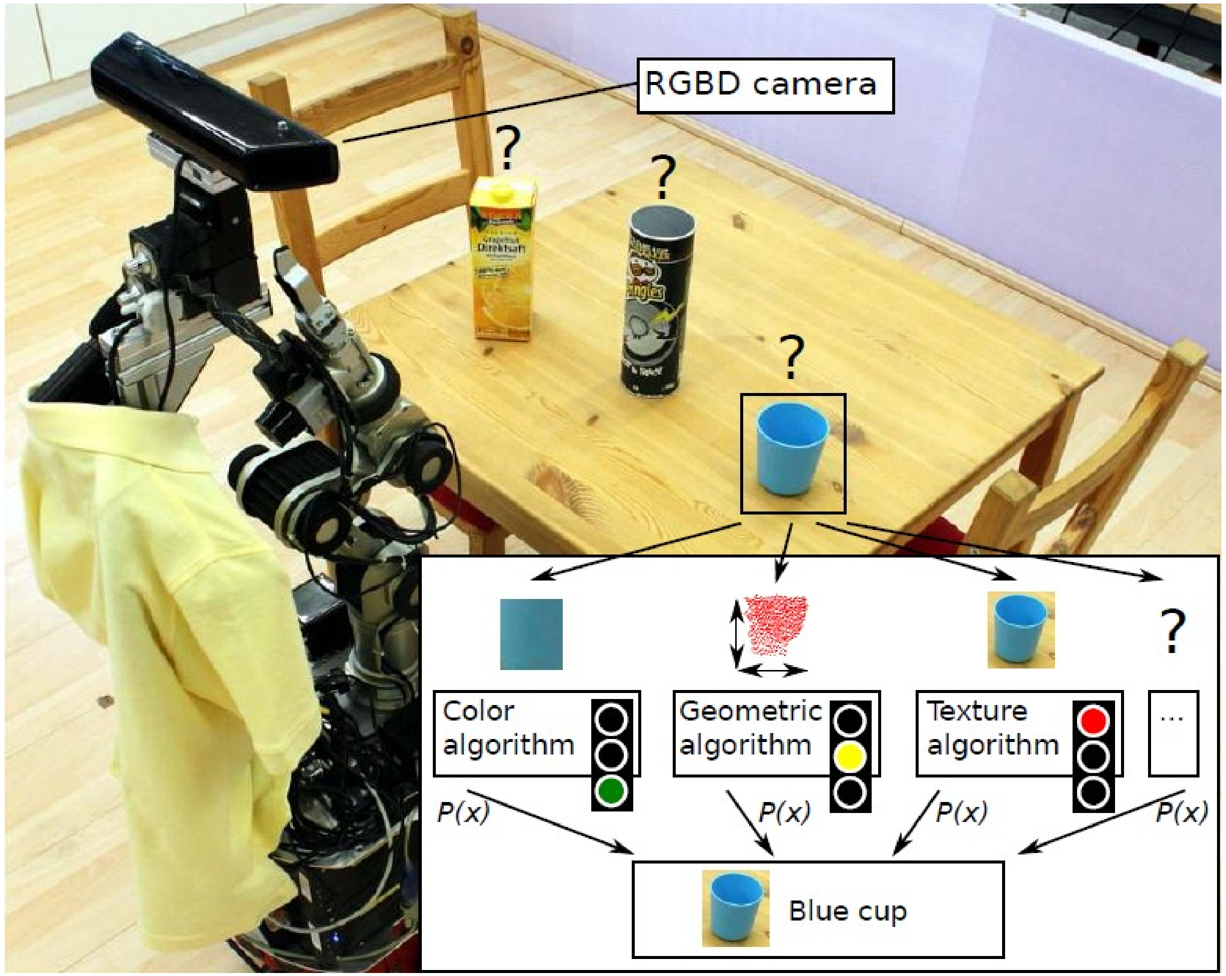
Multi-Modal Object Recognition

Combining algorithms

- Run multiple algorithms on every object candidate
  - Many algorithms available
  - Reduces recognition problem for the single algorithm
  - Requires an interface to collect and interpret the results

- Algorithms report recognition result as probability
  - Tells how reliable recognition is
  - Semantic interface

- Fusing the single hypotheses to a final result
  - Probabilistic
  - Use established probabilistic methods

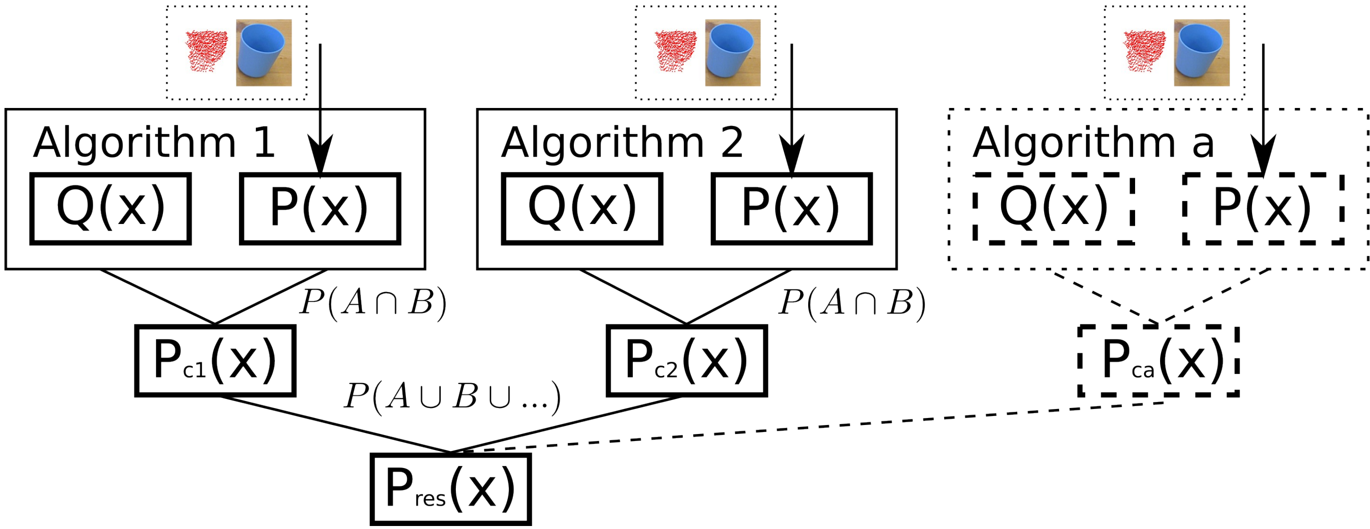


Multi-Modal Object Recognition

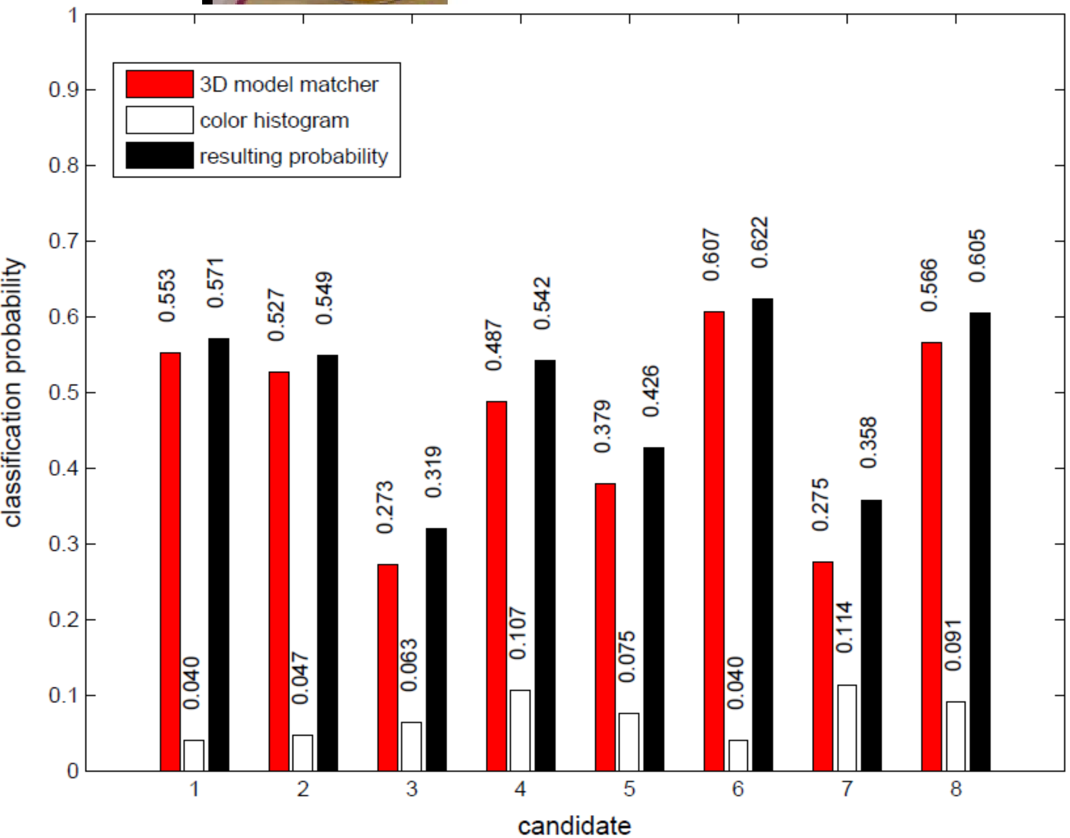
Combining Algorithms

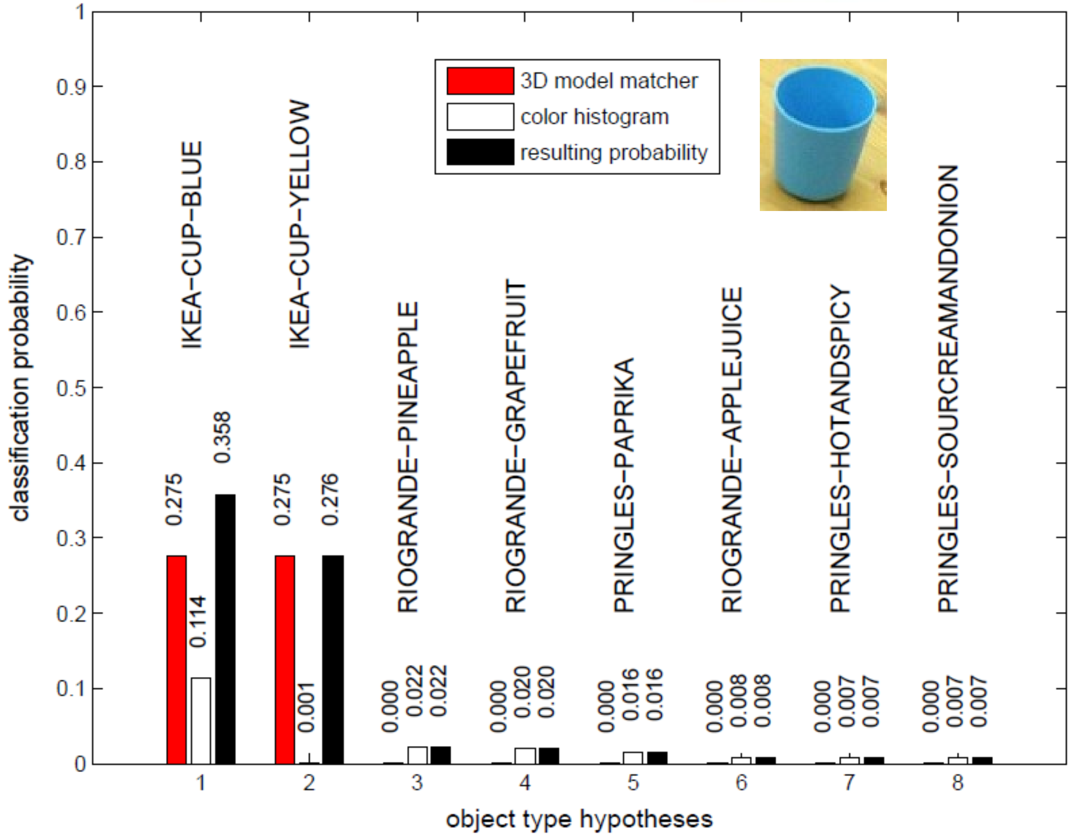
- A measure of quality is required
  - Defines the quality of an algorithm
  - How useful is an algorithm to recognize an object?
  - Probabilistic
  - Depending on the object

- Fusing the algorithm results considering the recognition probability and algorithm quality



Multi-Modal Object Recognition - Results



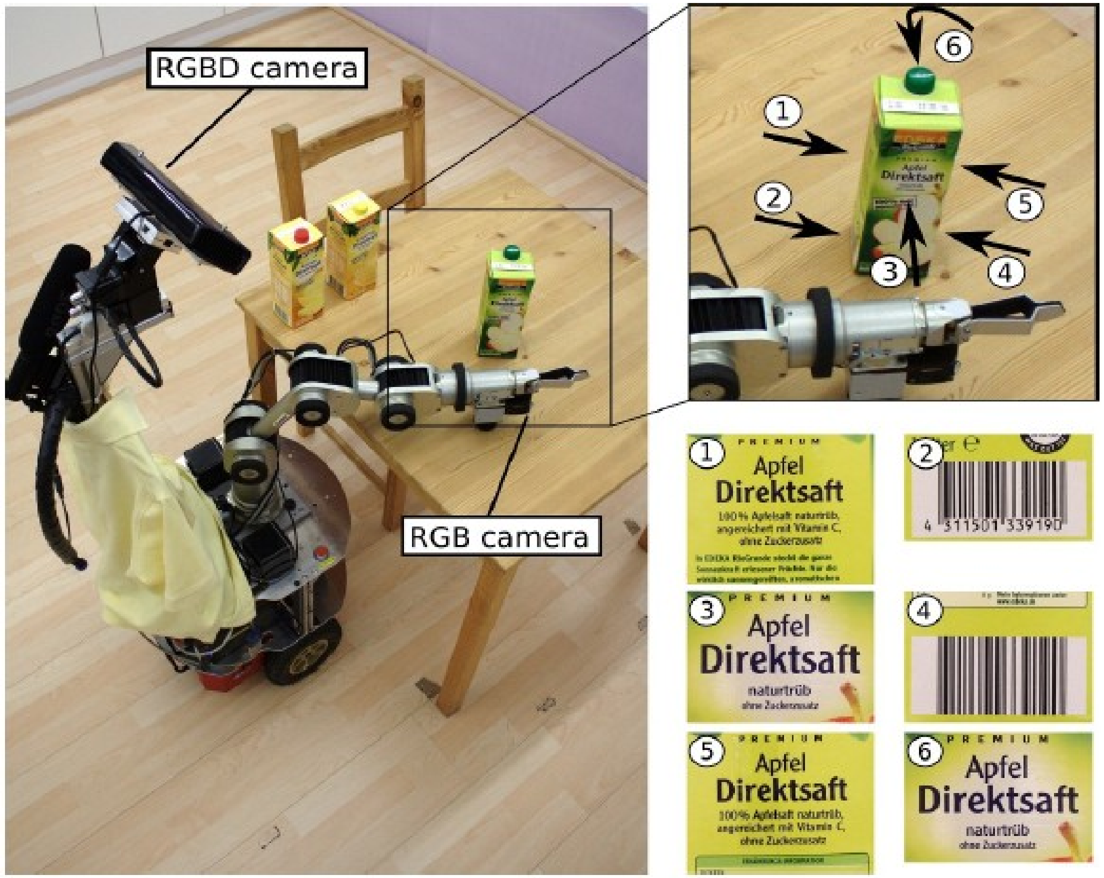


servicerobotik ZAFH

Hochschule Ulm

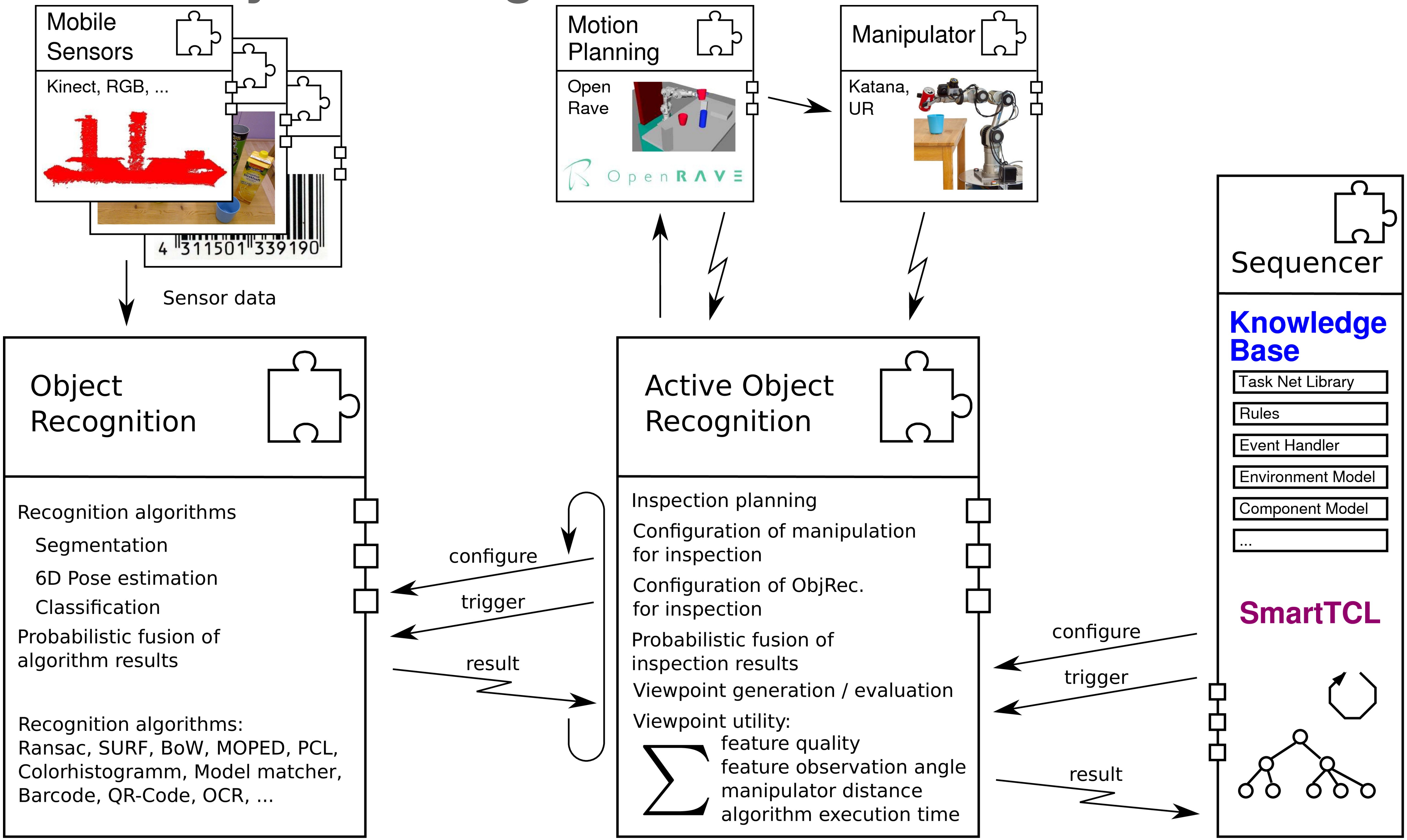
Active Object Recognition: Motivation

- Performance of “scene recognition” often not enough
  - Low / insufficient quality of sensor data,
  - Bad perspective,
  - Occlusion, etc.



- Idea
  - Active acquisition of new sensor data from other views on the object
  - Use images from camera mounted on manipulator

Active Object Recognition: Structural Overview

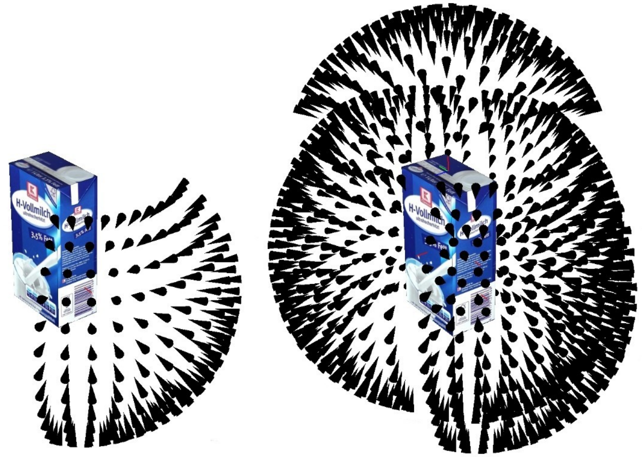


servicerobotik

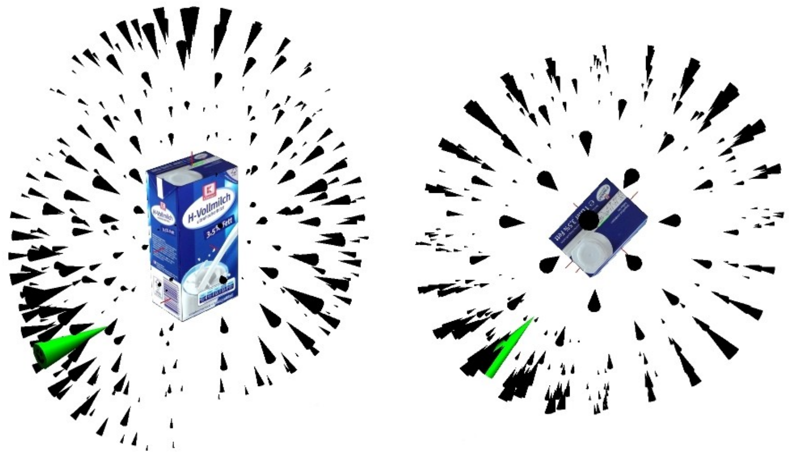
Hochschule Ulm

Active Object Recognition

- Extending object recognition by manipulation
  - Requires taking the environment into account
- Recognition process:
  - 1: Object recognition on full scene
  - 2: Generate viewpoints
  - 3: Select one viewpoint
  - 4: Simulate and manipulate
  - 5: Run object recognition on new data
  - 6: Include new observation
- Probabilistic fusion of results
- Repeats as necessary
  - Required certainty configurable
  - e.g. juice vs. medicine



Viewpoints generated for a milk box

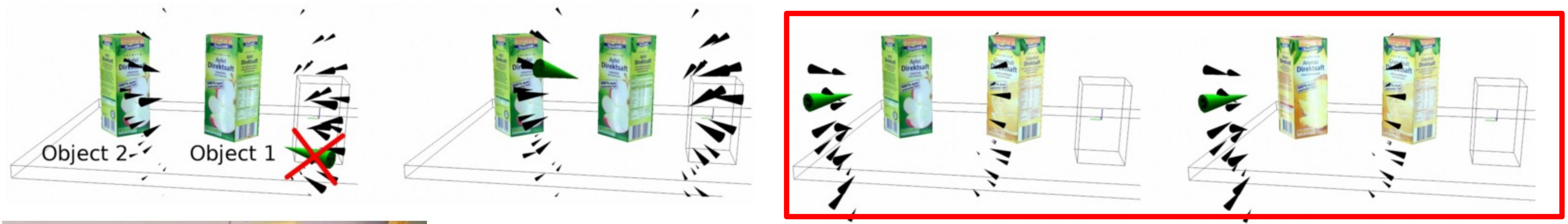


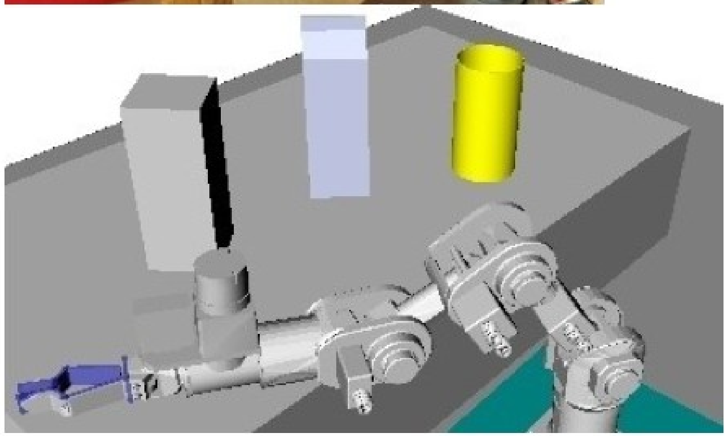
Evaluated viewpoints

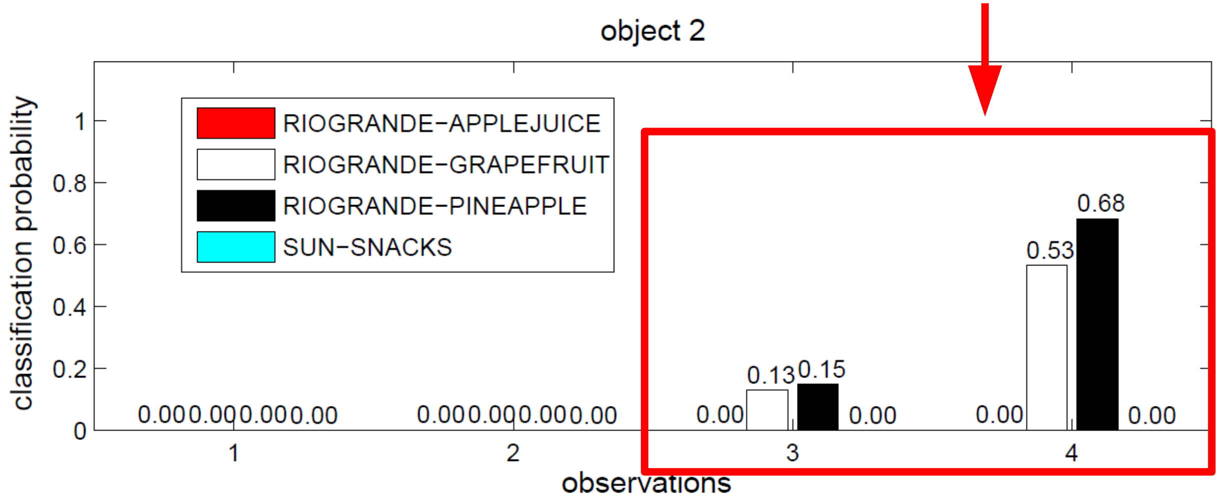
Viewpoint generation and selection

- *Generating camera positions to point at features on the object*
  - *Barcodes*
  - *Texts/labels*
- *Evaluate viewpoints*
  - *The quality of a feature seen from viewpoint*
  - *Estimated quality of sensor data*
  - *Effort for manipulation*
- *Camera is positioned at “best” viewpoint with the highest utility*
- *All objects in the scene are considered for manipulation*

Active Object Recognition: Experiment







Observation 1: look at object 1 => manipulator collision
Observation 2: look at object 1 => successfully classified
Observation 3: look at object 2 => not sufficient classification quality
Observation 4: look at object 2 => now sufficient classification quality

Overview

Use Cases?

Overall principles behind our work

Needs?

Benefits?

Goals?

Achieve suitability for everyday life of service robots

Task-nets for conditional reative task execution

Model-driven software development for robotics

Learning from demonstration for manipulation

Technology?

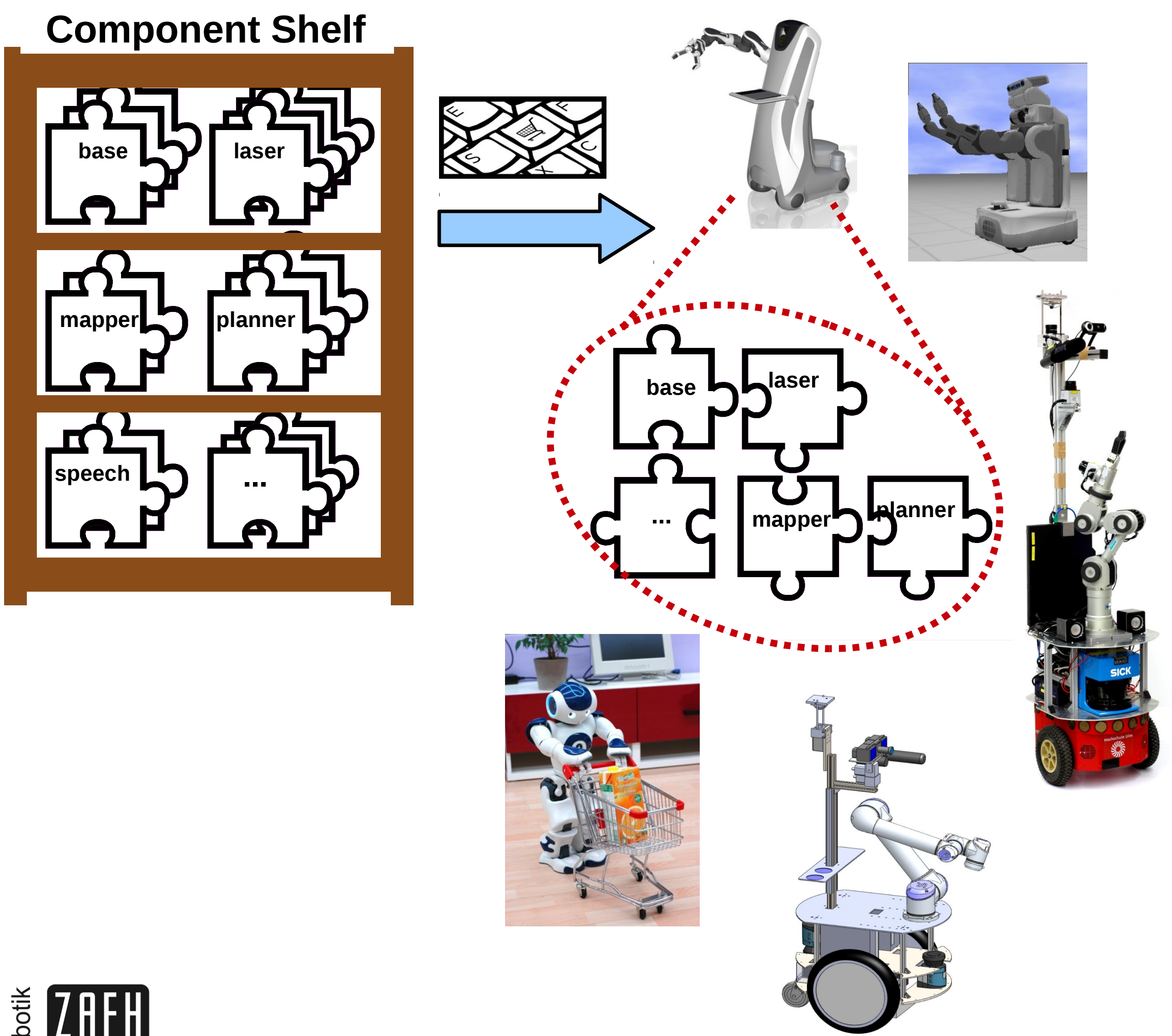
Resource-aware SLAM

Active object recognition with information-driven sensor placement

Approach?

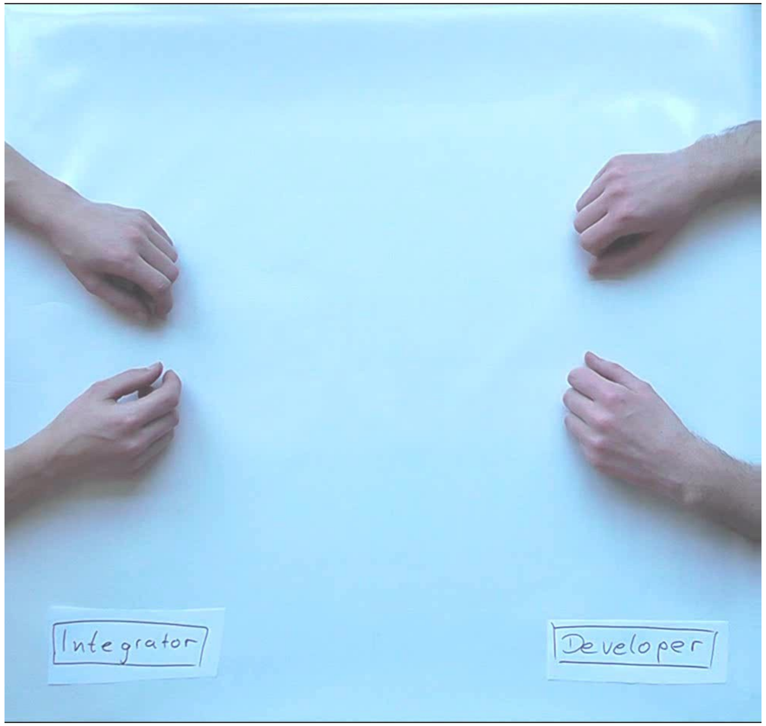
- focus on tools for systematic engineering of service robotic applications (e.g. MDSD)
  - separation of roles
  - separation of concerns
- focus on robust and efficient key functionalities and components
  - extending and merging so far separated techniques

Software Concepts for Service Robots: Model-Driven Software Development

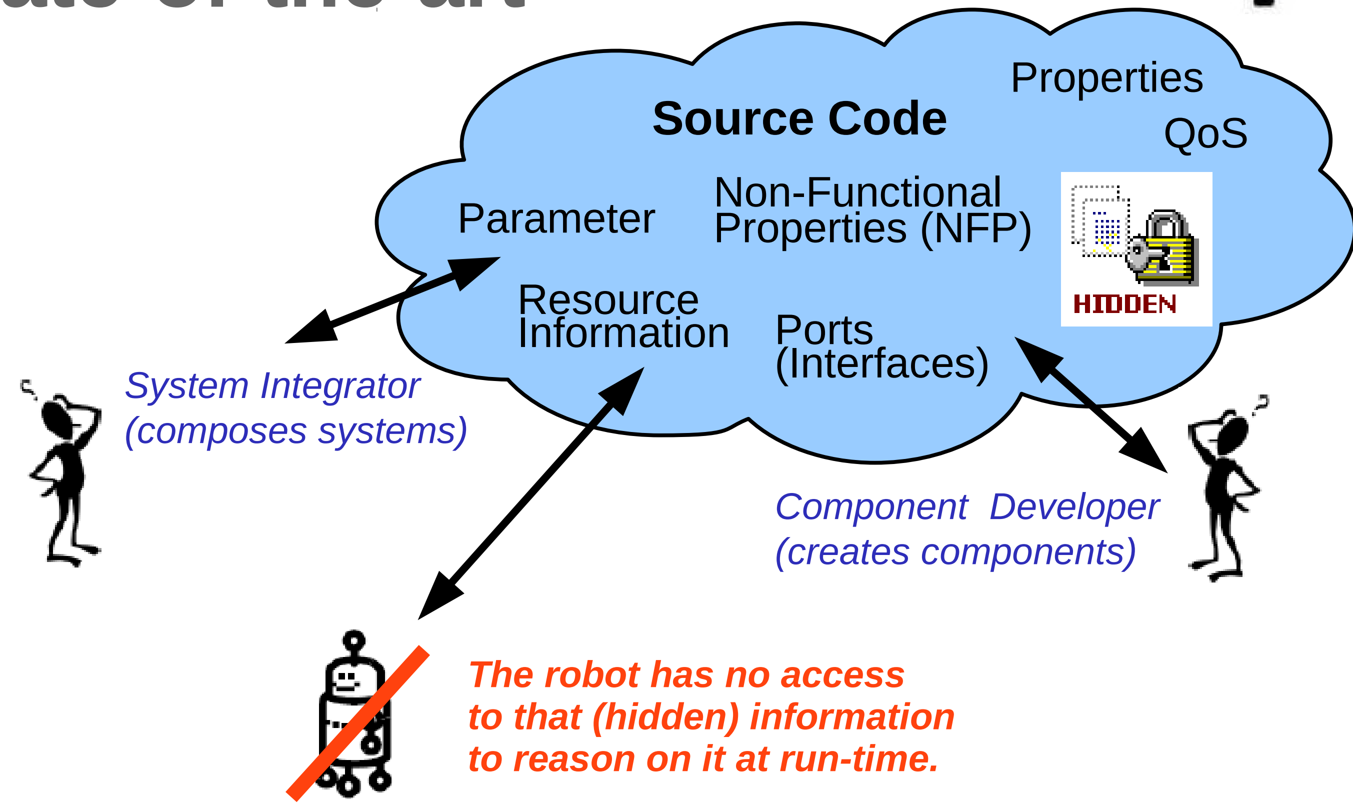


Software Concepts for Service Robots: Current Situation / State-of-the-art

?

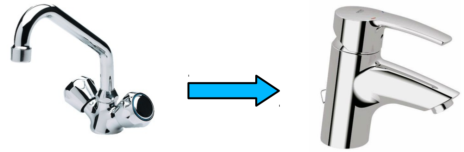


System Integrator *Component Developer*



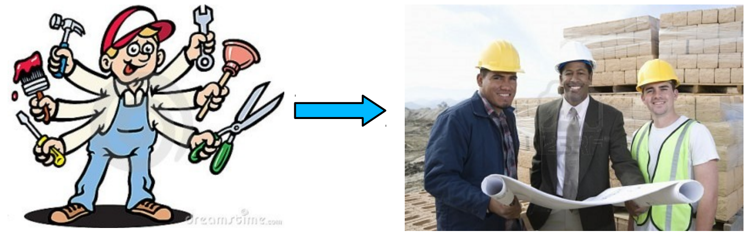
- relevant information is hidden in source files (parameters, properties, ports, resource information)
  => source code has to be analyzed
- no explicit descriptions of properties of software building blocks
  => no black box reuse possible
- **domain experts (e.g. cleaning business) need to become robotics experts (or vice versa)**

Software Concepts for Service Robots: Current Situation / State-of-the-art



No separation of concerns ***(in order to reduce complexity)***

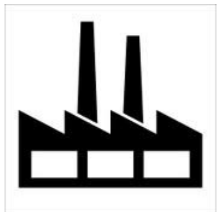
- computation,
  communication,
  configuration (parameters at component and system level),
  coordination (orchestration, resource management)



No separation of roles ***(in order to support specialization)***

- end users, system integrators,
  component developers, framework developers

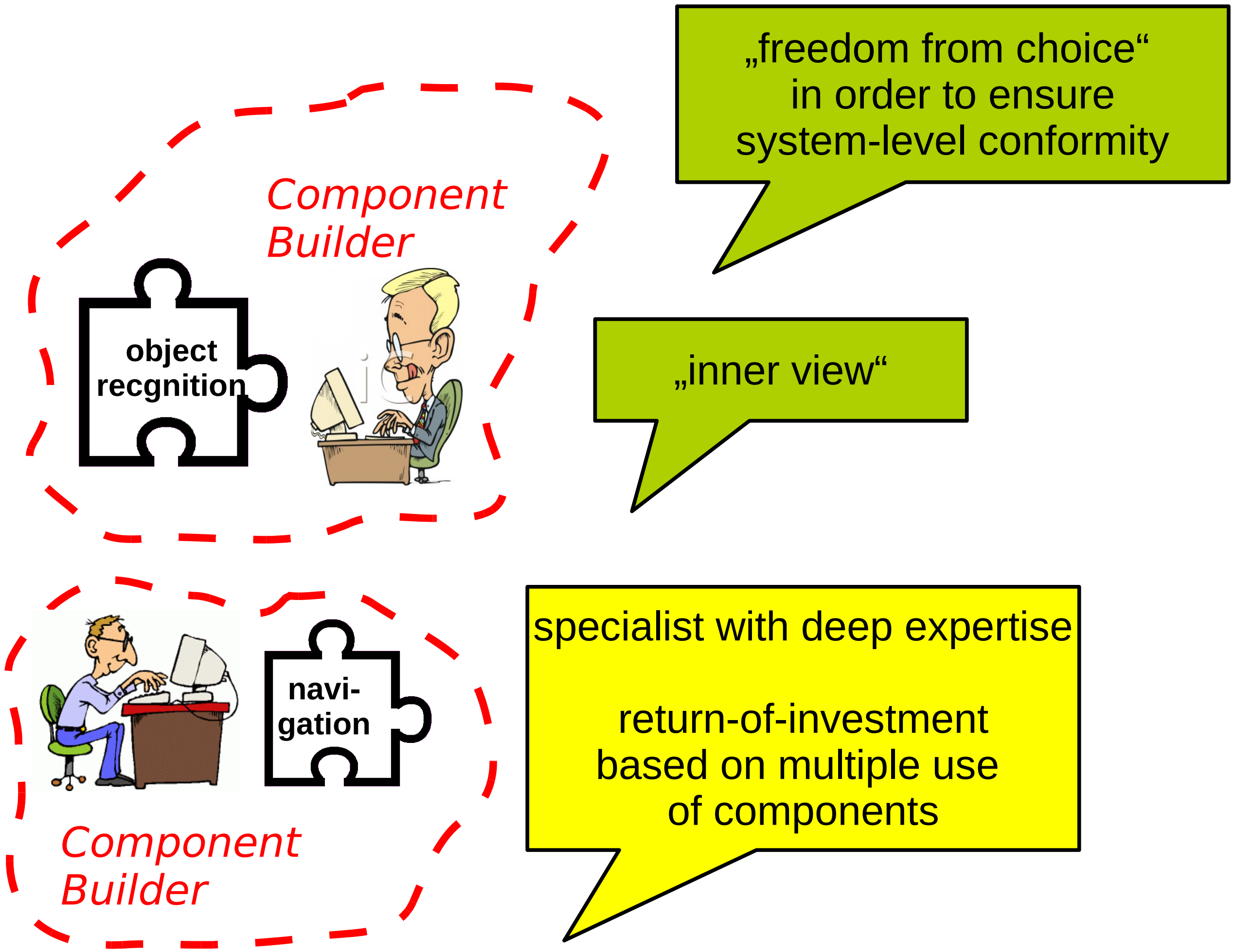
Academia so far circumvented this challenge by not separating between component builders and system integrators



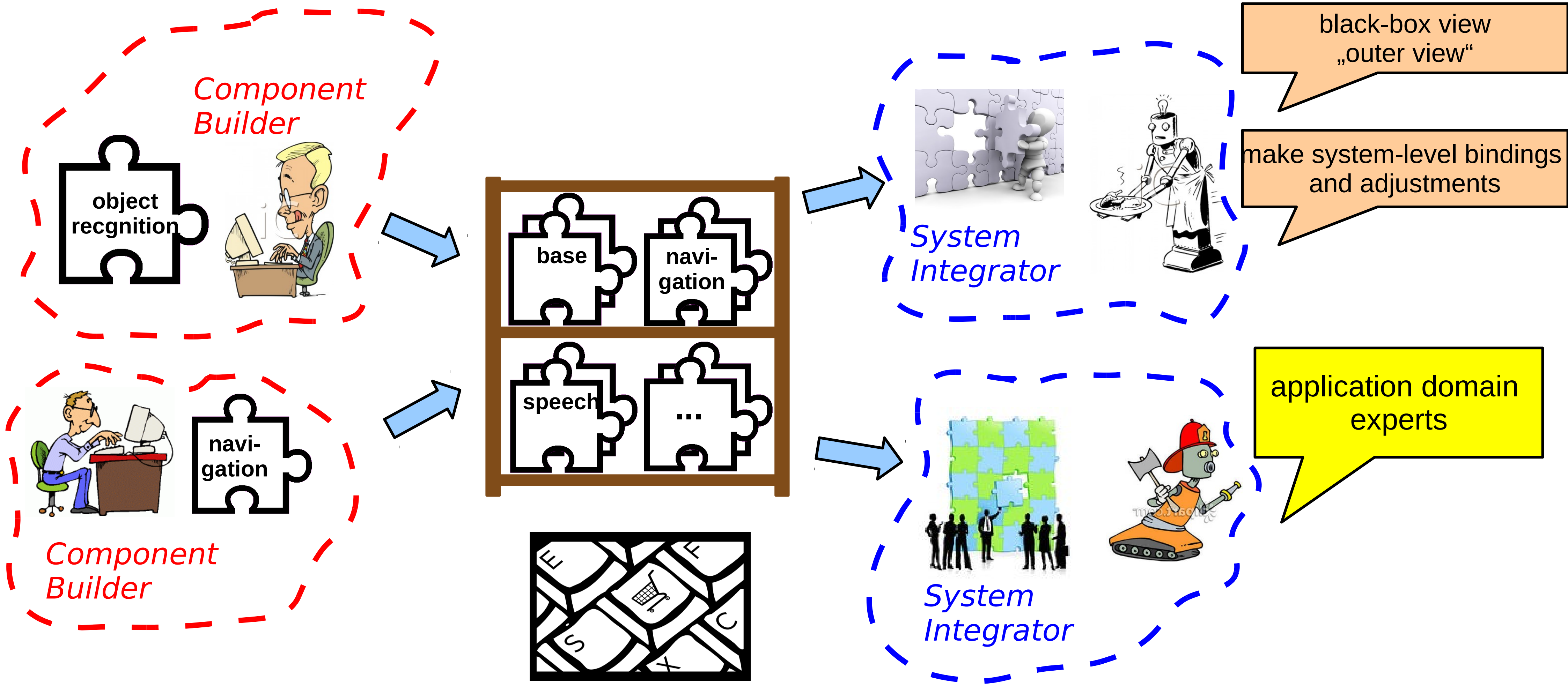
Separation of roles and separation of concerns is essential for successful markets

- lower risks, share efforts, provide second source, reduce costs, reduce development time, reduce time-to-market, increase robustness, increase quality, …

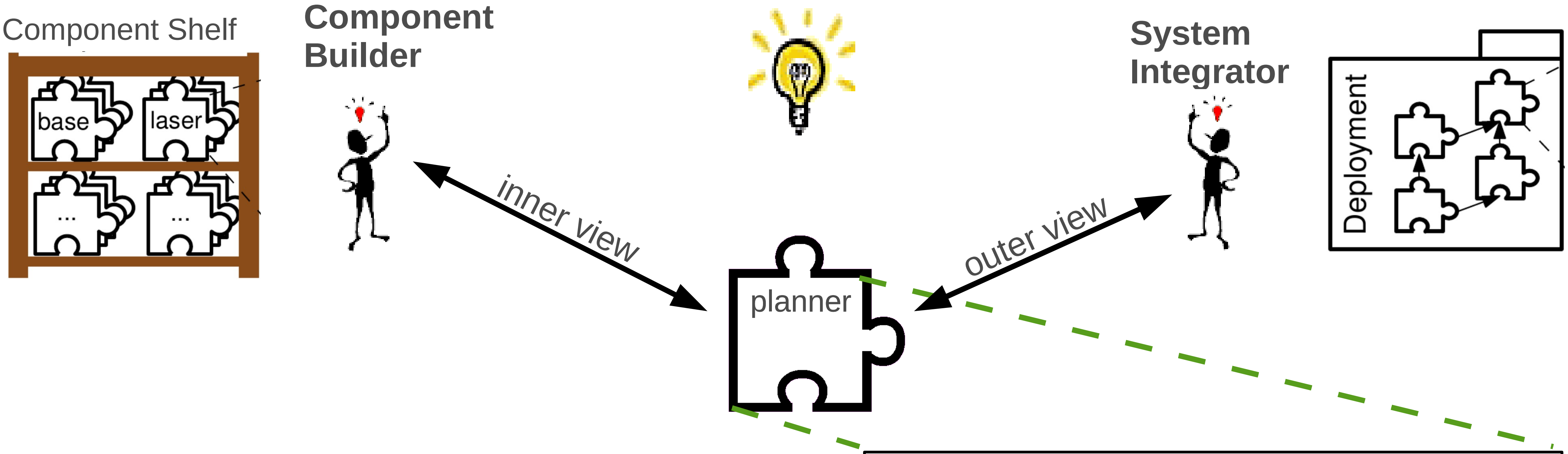
Software Concepts for Service Robots: What we want to have: separation of roles, concerns



Software Concepts for Service Robots: What we want to have: separation of roles, concerns



Software Concepts for Service Robots:
What we want to have: separation of roles, concerns



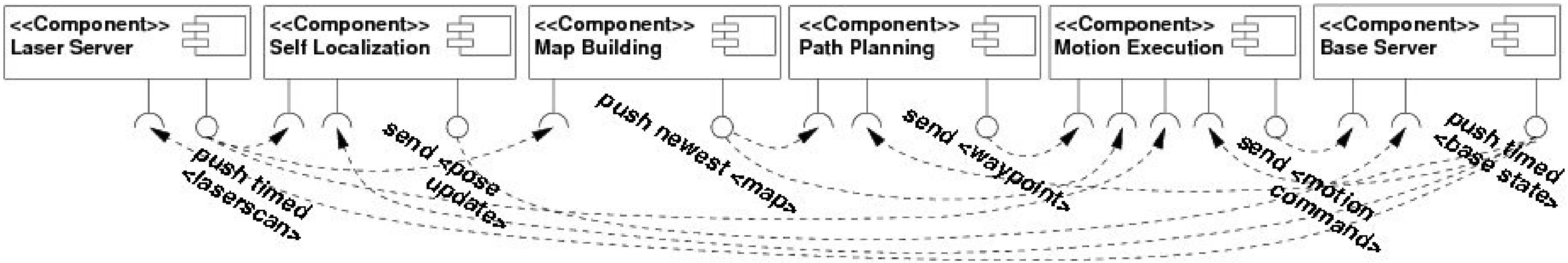
Name: planner
Ports:
mapClient : pushNewestClient<gridmap>
goalServer : pushNewestServer<goal>
plannerEvent : eventServer<goalStatus>
state : stateServer
param : parameterServer
...
States:
active, neutral, ...

Parameters, Properties, Ports (Interfaces), Resource Information, …

ARE EXPLICATED IN THE MODELS
AND ARE NOT HIDDEN IN THE SOURCE FILES

Software Concepts for Service Robots: Where to start?

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

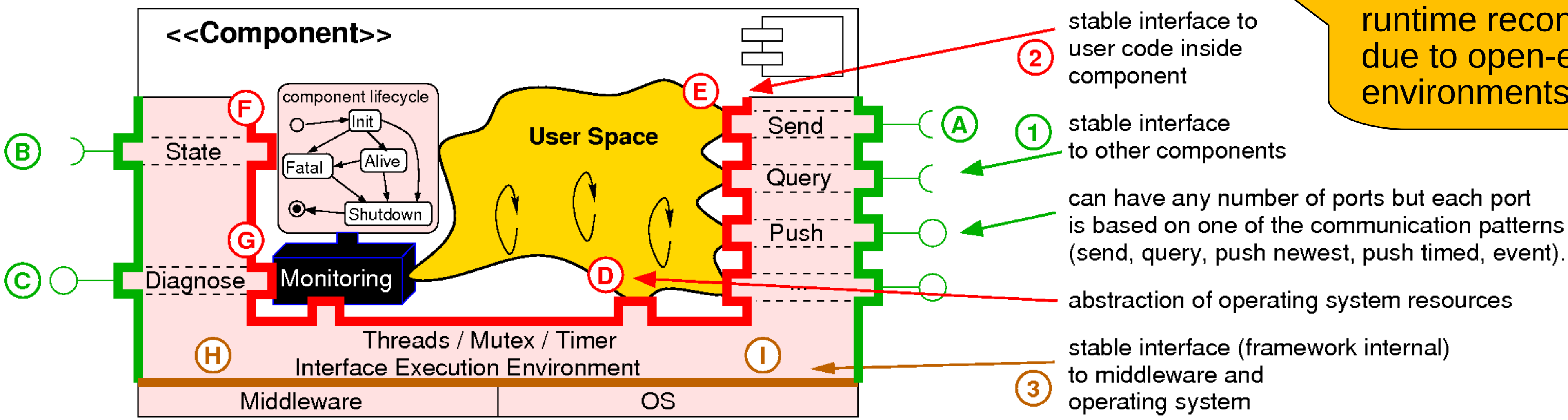


- Separating the roles of the component builder, system integrator and the robot requires to identify, specify and explicate stable structures as well as variation points each role can rely on.
- These stable structures and variation points build the ground for a model-based representation. Representing 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.

Approach: service-oriented component model
=> master component hull by MDSD

– Separate inside view (component builder) from outside view (system integrator)
– Separate stable execution container from implementational technologies (middleware, OS)
– MDSD to generate component hull ensures compliance at the component and system level while giving freedom within a component

Difference in robotics: runtime reconfiguration due to open-ended environments



- *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*

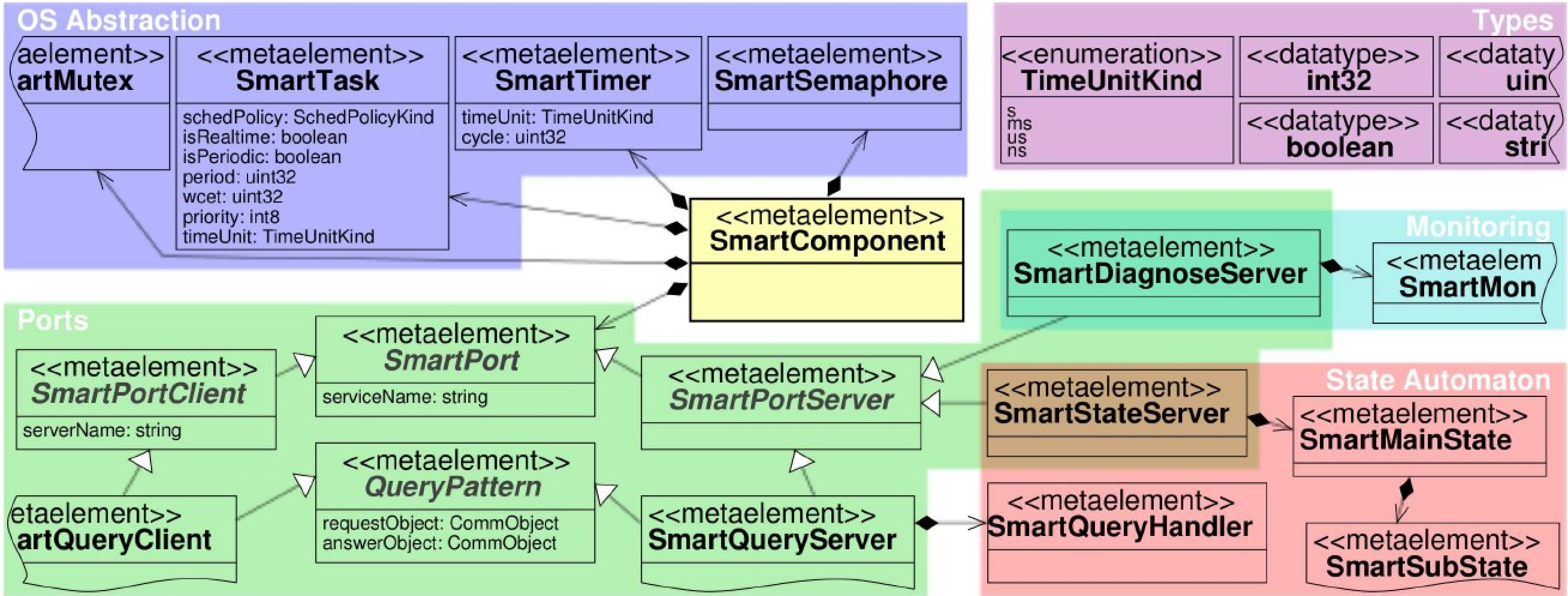
The SmartSoft Communication Patterns

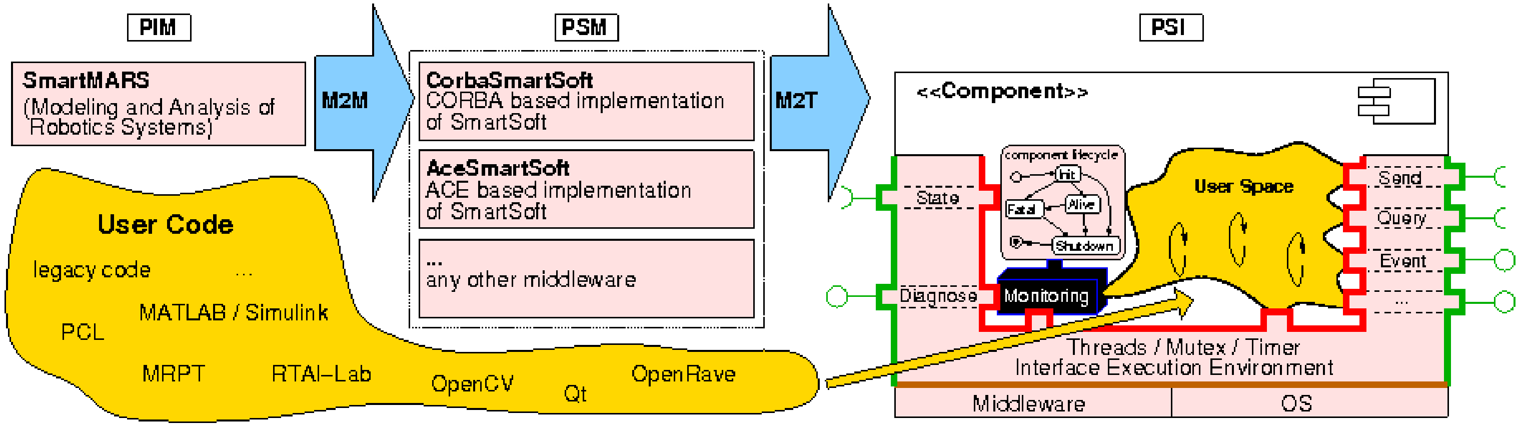
send	one-way communication
query	two-way request/response
push newest	1-to-n distribution
push timed	1-to-n distribution
event	asynchronous conditioned notification

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

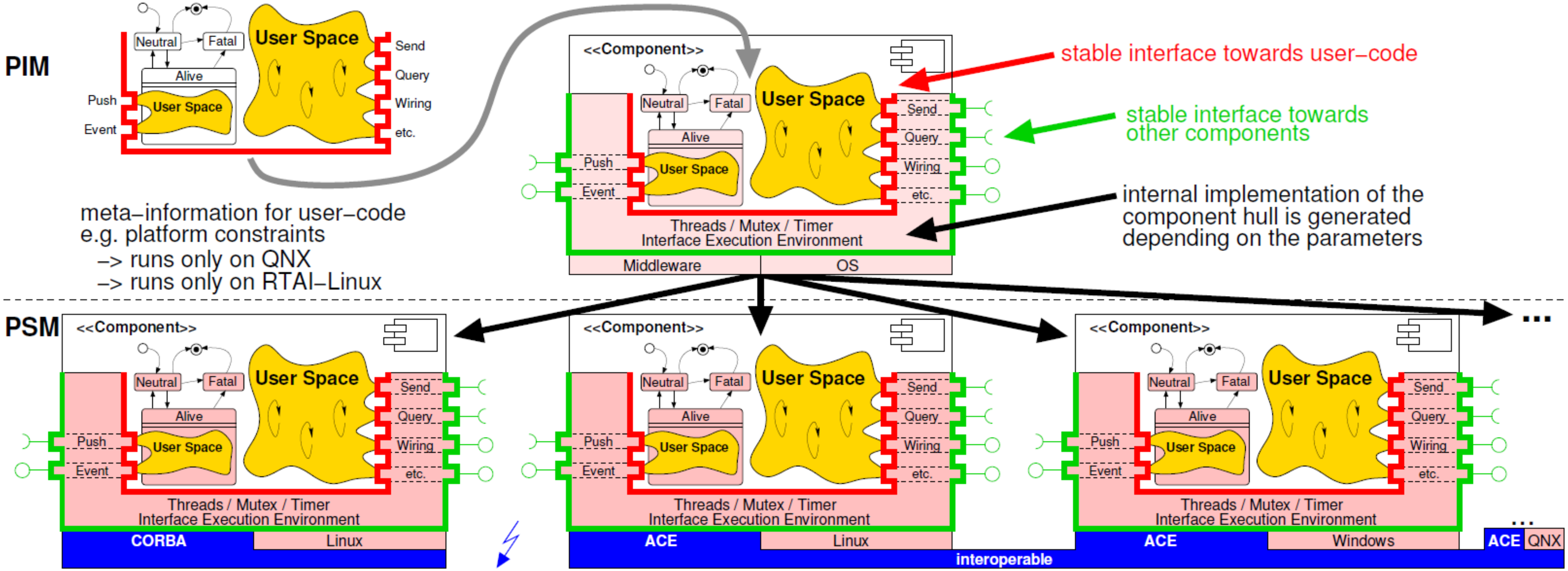




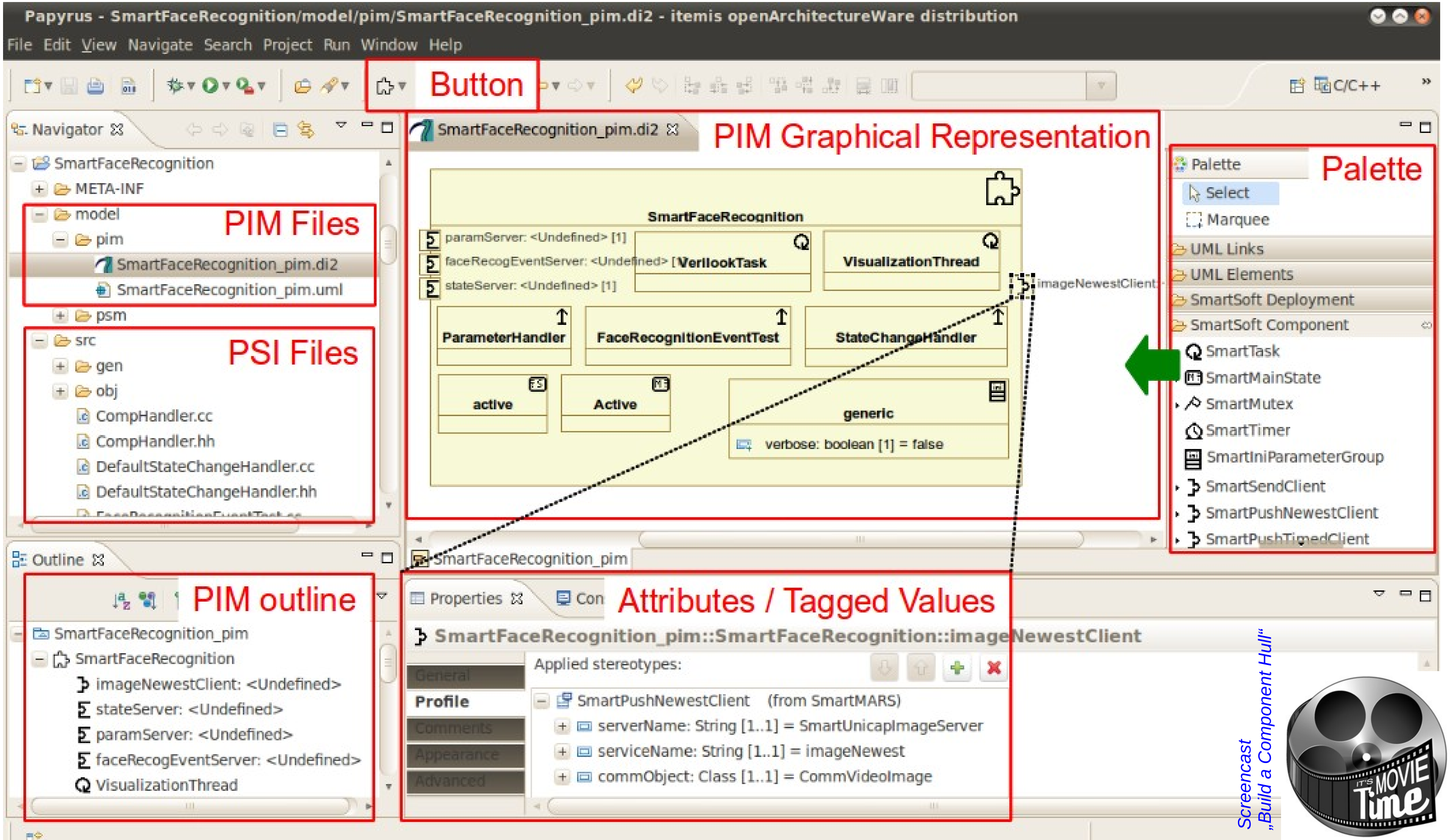
servicerobotik

Hochschule Ulm

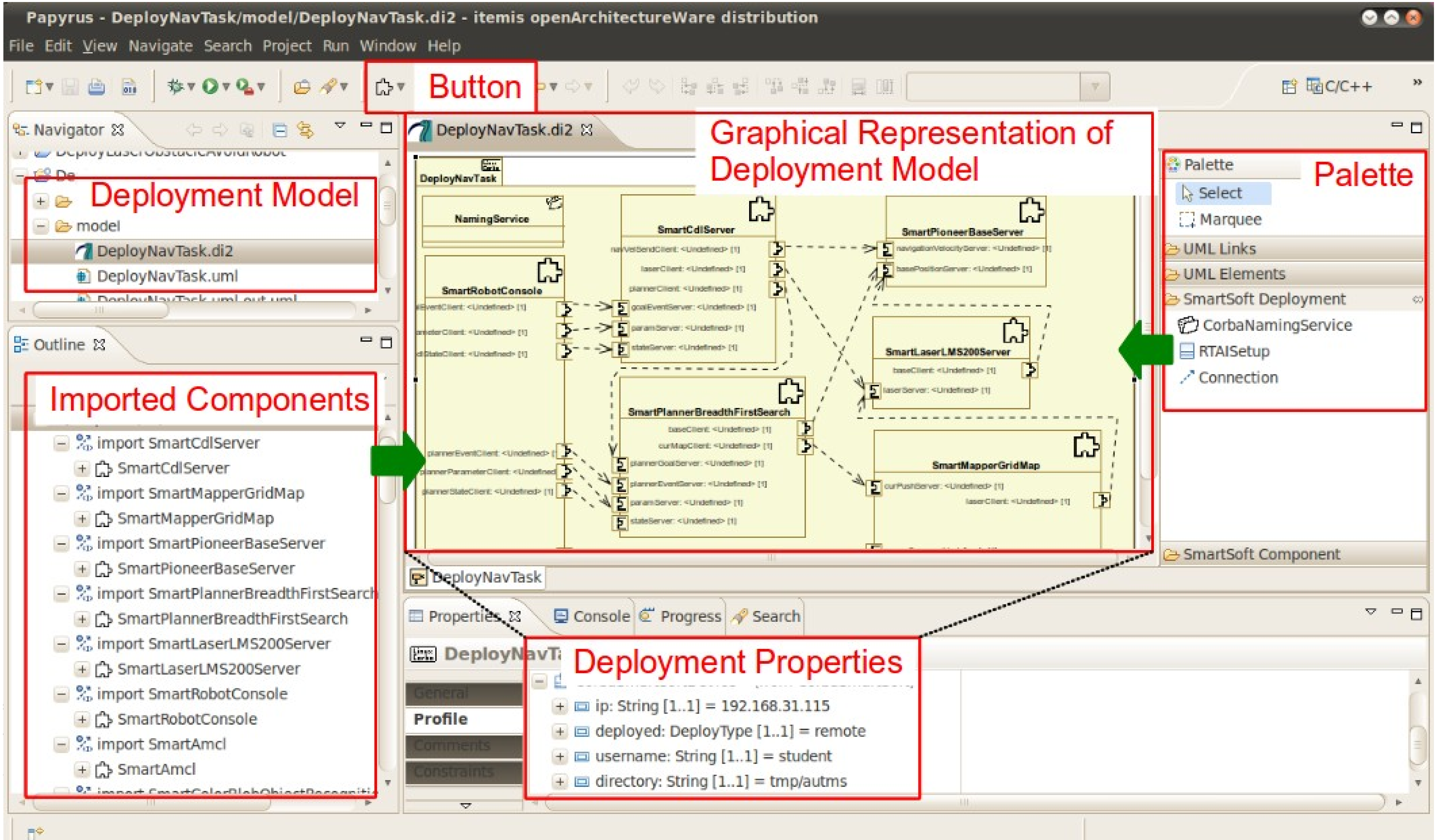
The SmartSoft Component Model
Mapping to different Middlewares



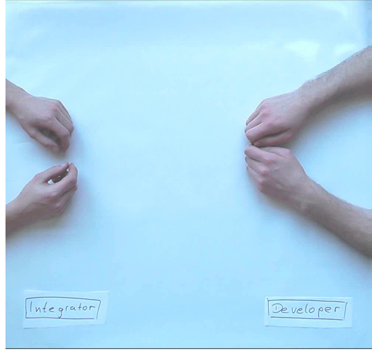
Model-Driven Software Development: Component Builder View

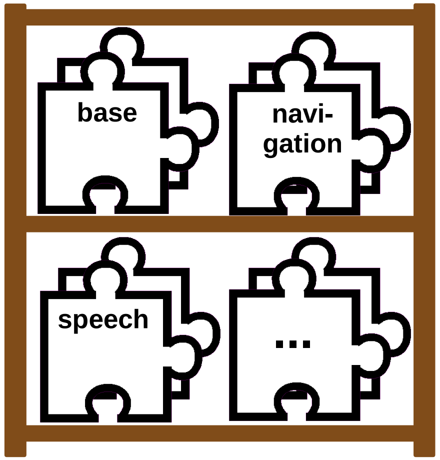


Model-Driven Software Development: System Integrator View



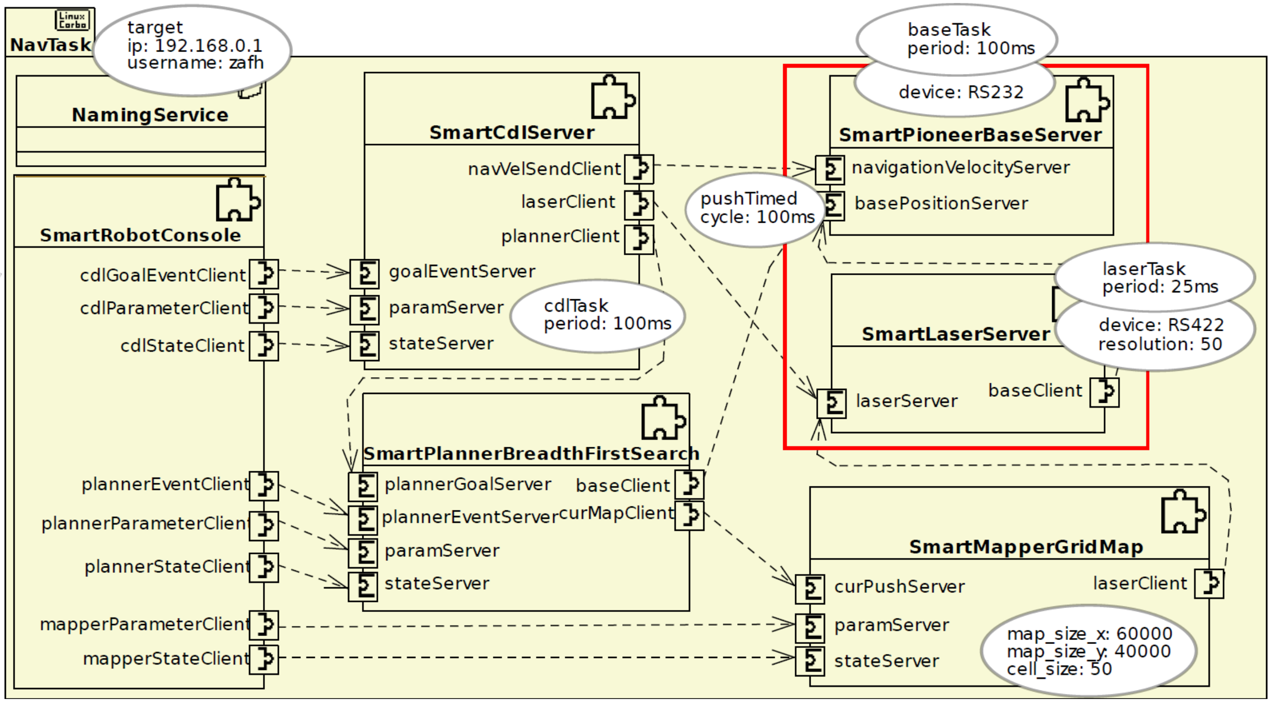
Model-Driven Software Development: System Integrator View

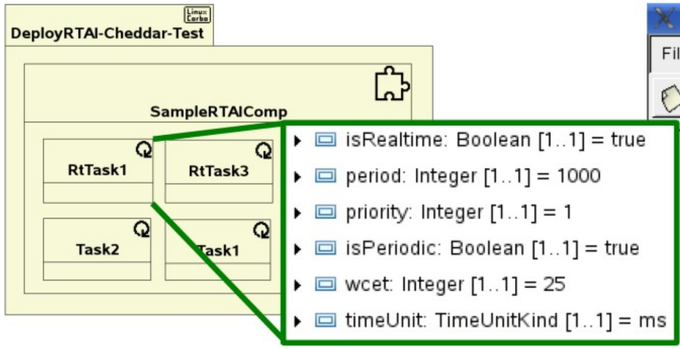


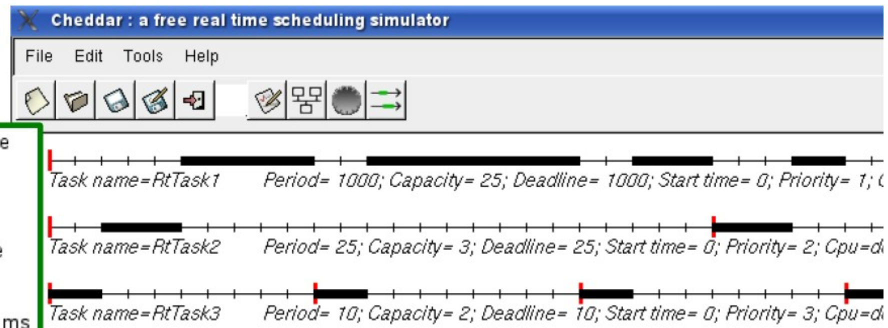


Component Shelf
Reusable Components

System Level Properties / Bindings / Conformance Checks

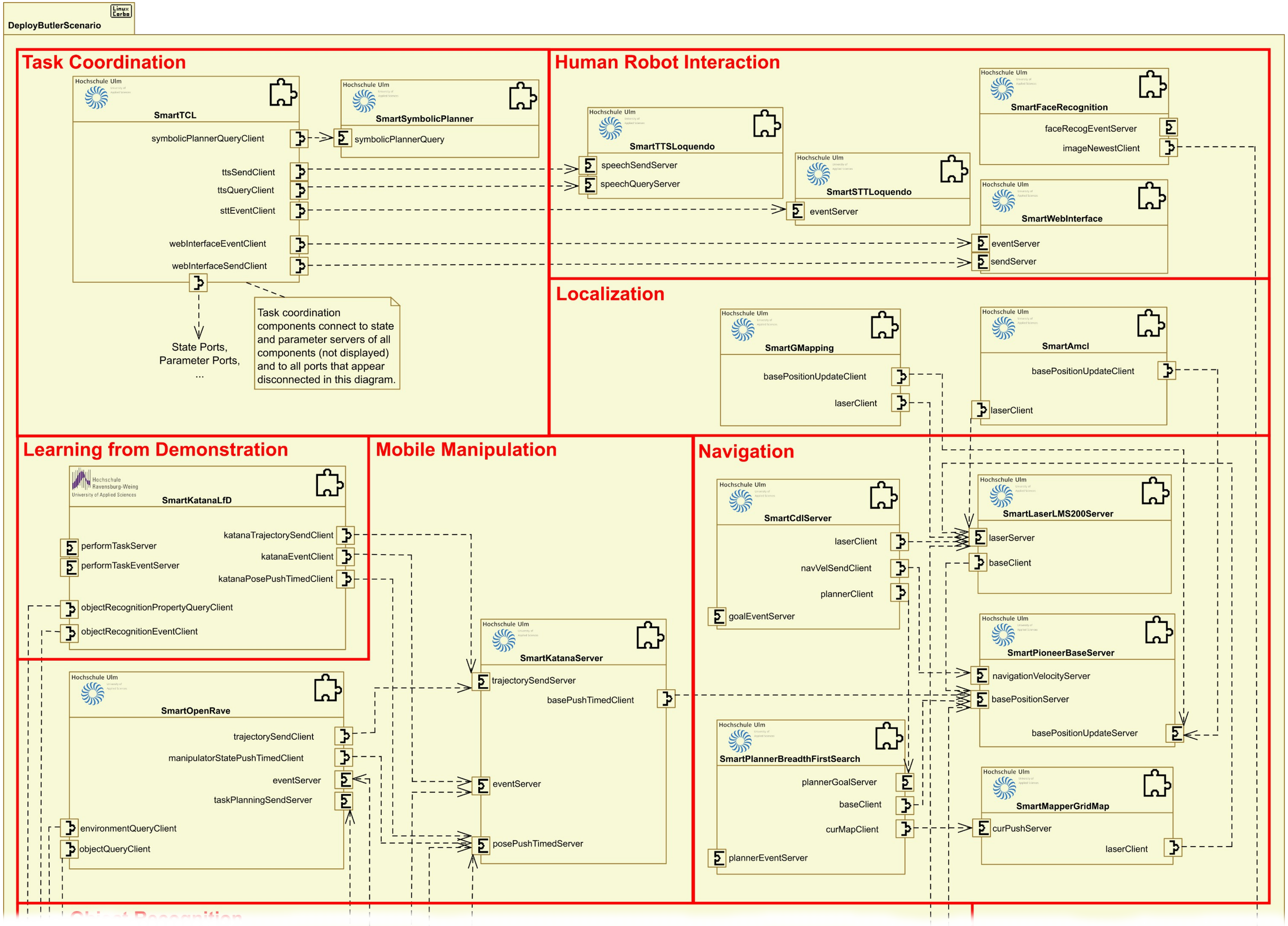


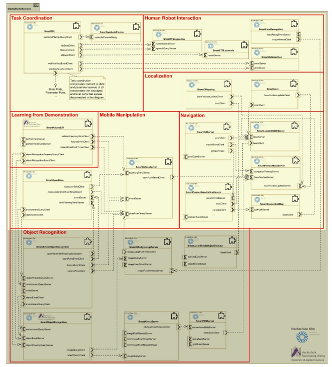




Model-Driven Software Development

System Integrator View – Butler Scenario

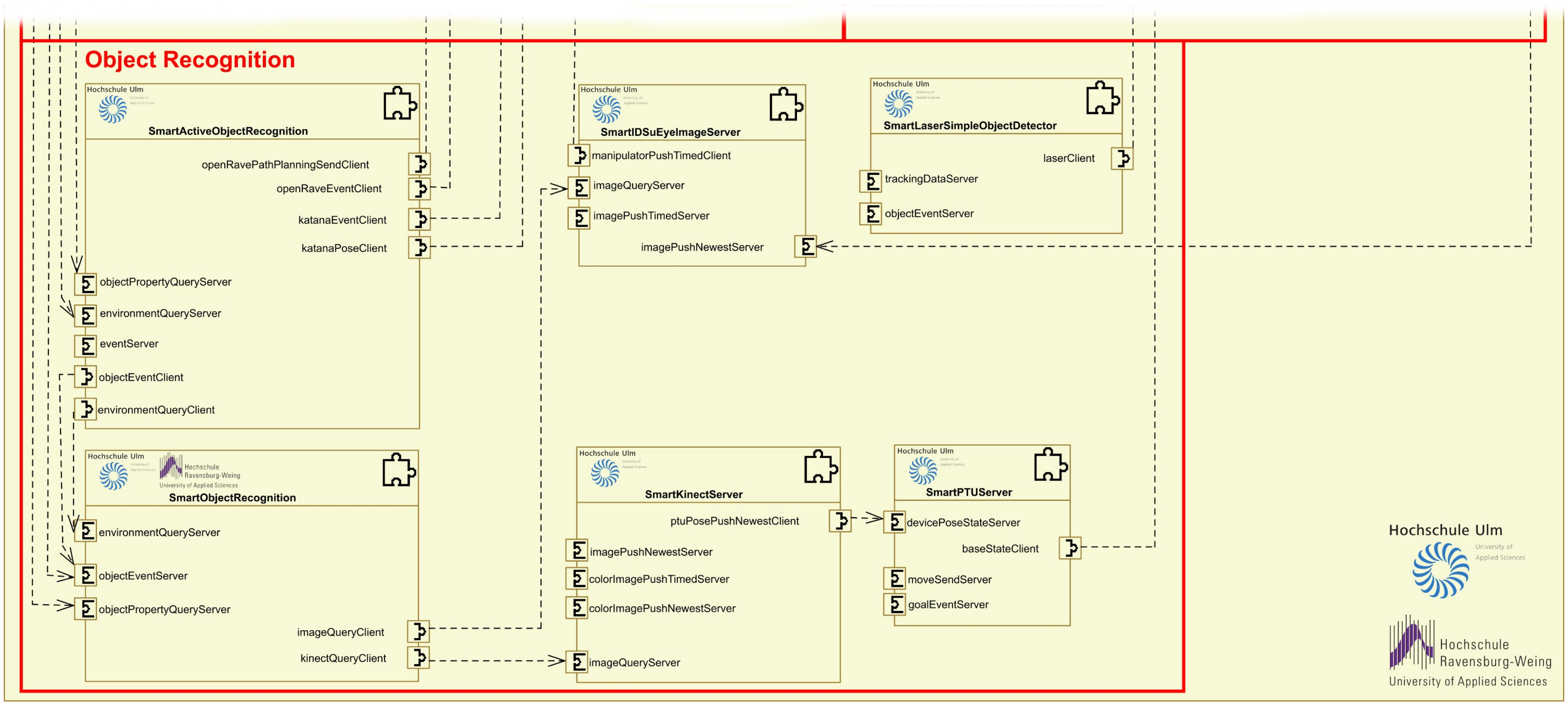


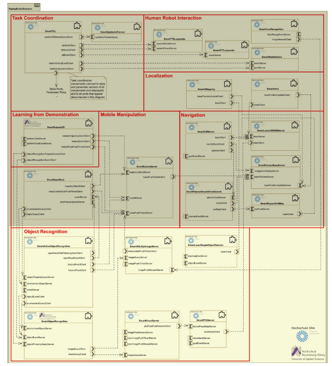


Hochschule Ulm

Model-Driven Software Development

System Integrator View – Butler Scenario





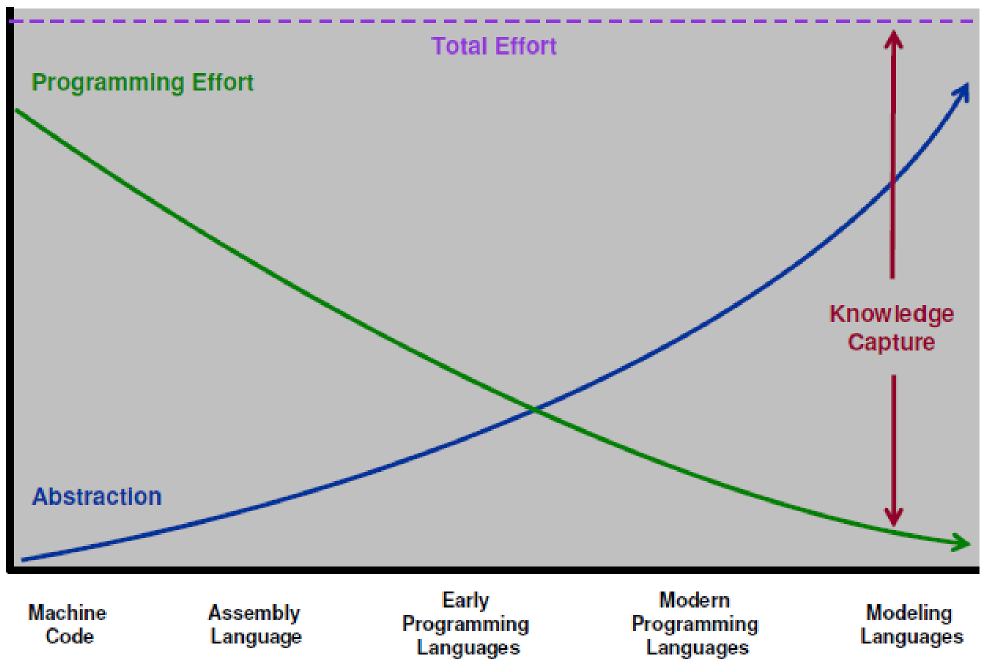
Model-Driven Software Development: The future of a service robotics market

- significant savings in terms of man power for setting new and different robots into service
- "separation of roles and concerns" supported by a model-driven software approach can initiate a shift in robotics from technology driven exploitation to use-case driven exploitation of robotics technology
- design-abstraction can bridge the gap between academia and industry => future perspective:

- *collaborate* at the level of meta-models, models, software tools, etc.
- *compete* at the level of implementations, specialized frameworks, proprietary functionalities, etc.
- allow for a symbiotic eco-system of large companies and SMEs, specialists and integrators

D.J. Hoch, W. Huhn, U. Naeher, A.E. Zielke: The Race to Master Automotive Embedded Systems Development, McKinsey, 2006

- productivity gain in model-based software engineering is estimated to be about 30%
- MBD is one of the most influential levers and can lead to step-change increase in both productivity and quality



Summary

Open-ended environments: a tremendous amount of situations

- how to spend scarce resources in a most appropriate way?
  - acting efficiently
  - achieve a high degree of robustness
  - maintain a high success rate in task fulfillment

Goal

- flexible response to dynamic environments
- complexity and variety of tasks: multi-purpose robot
- be able to put in as much knowledge about tasks as possible

Challenges: always deviation between design-time/run-time optimality

- even most skilled robotics engineer is not able at design-time to
  - identify and enumerate all eventualities in advance
  - properly code configurations, resource assignments, reactions

  (even not efficient at all due to the combinatoric explosion of possible situations and skill parameterizations)
- not possible just to (re)plan at run time in order to take into account latest information as soon as it becomes available

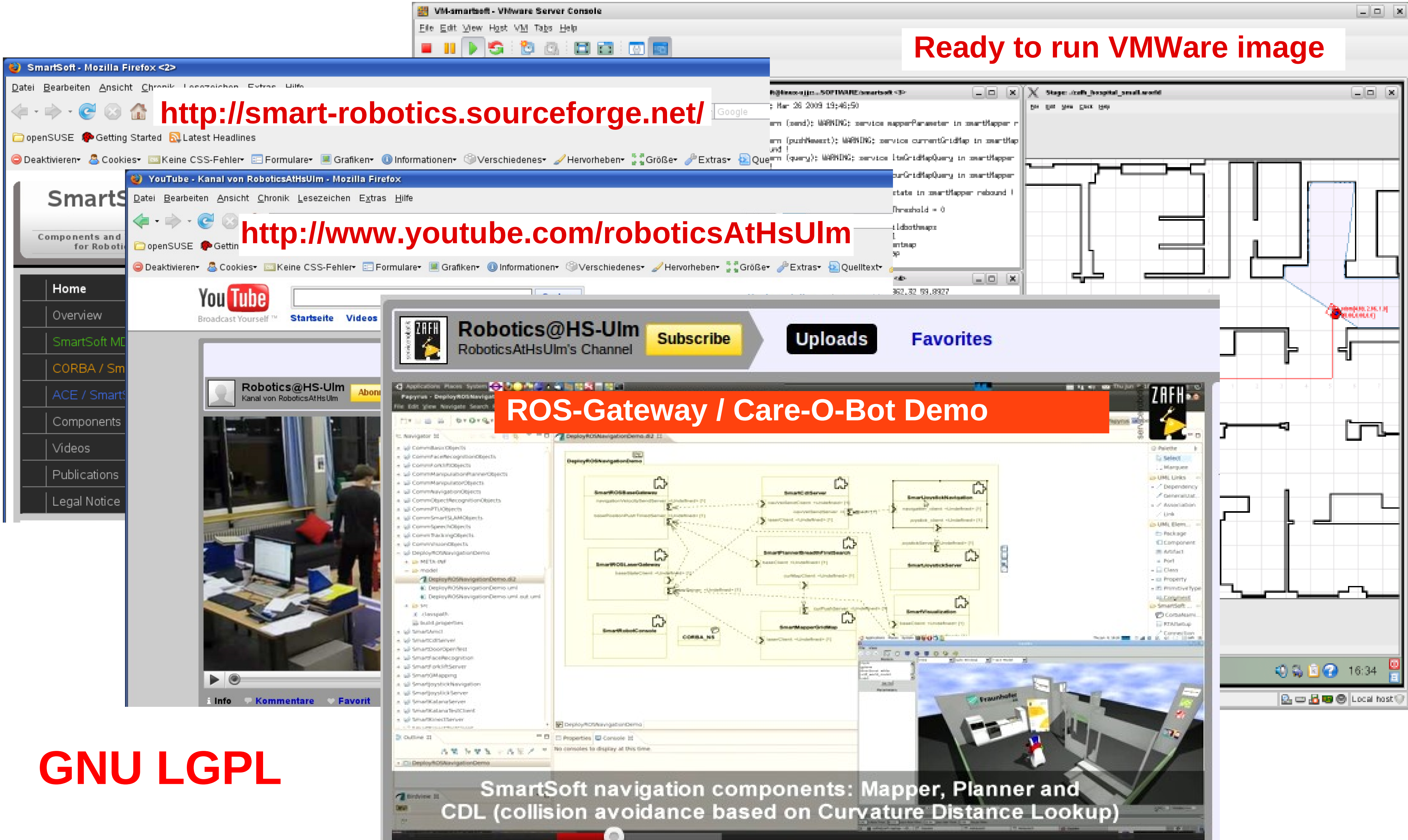
  (computational complexity of planning far too high when it comes to real-world problems, i.e. generate action plots given partial information while taking into account additional properties like safety and resources)

Motivation for a different approach:

- make it as simple as possible for the designer to express variability at design time
- robot needs to be able to bind variability at run time based on the then available information
- At design time, we also specify which problem solver (symbolic planner, constraint solver, etc.) to use to bind which variation point.
- At run time, the robot then involves the prearranged and dedicated problem solvers.

Overall, this improves task execution quality, optimizes robot performance and cleverly arranges complexity & efforts between design time and run time.

Handling of intellectual property rights
Available as Open Source



GNU LGPL

ZAFH Servicerobotik – what is that?

Center for Applied Research at Universities for Applied Sciences

- University of Applied Sciences Ulm
  - Prof. Schlegel
- University of Applied Sciences Ravensburg-Weingarten
  - Prof. Ertel, Prof. Voos
- University of Applied Sciences Mannheim
  - Prof. Ihme, Prof. Wirnitzer

http://www.zafh-servicerobotik.de/
http://www.zafh-servicerobotik.de/ULM/publikationen.php
http://servicerobotik.hs-weingarten.de/publikationen.php

Addendum

What is different in robotics?

- 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!*

- *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)*

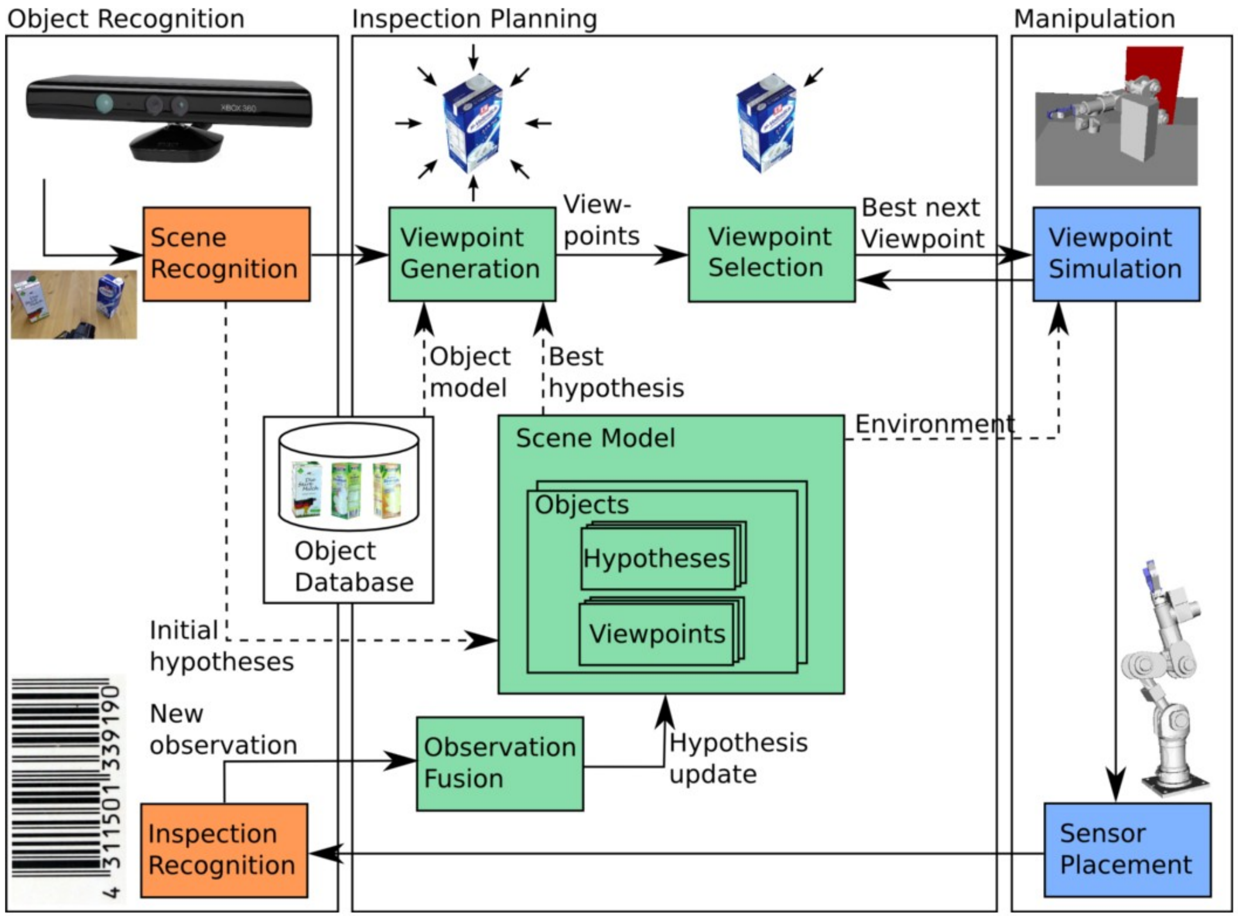
Active Object Recognition: Structural Overview

- Extending object recognition by manipulation
  - Requires taking the environment into account

- Recognition process:
  - 1: Object recognition on full scene
  - 2: Generate viewpoints
  - 3: Select one viewpoint
  - 4: Simulate and manipulate
  - 5: Run object recognition on new data
  - 6: Include new observation

- Probabilistic fusion of results

- Repeats as necessary
  - Required certainty configurable
  - e.g. juice vs. medicine



Software Concepts for Service Robots: Motivation and goals of research/development efforts

Motivation: ***Extensive software costs and high risks***

(see EFFIROB study / Fraunhofer IPA:
"efficient software engineering is decisive to lower development costs of service robotic applications")

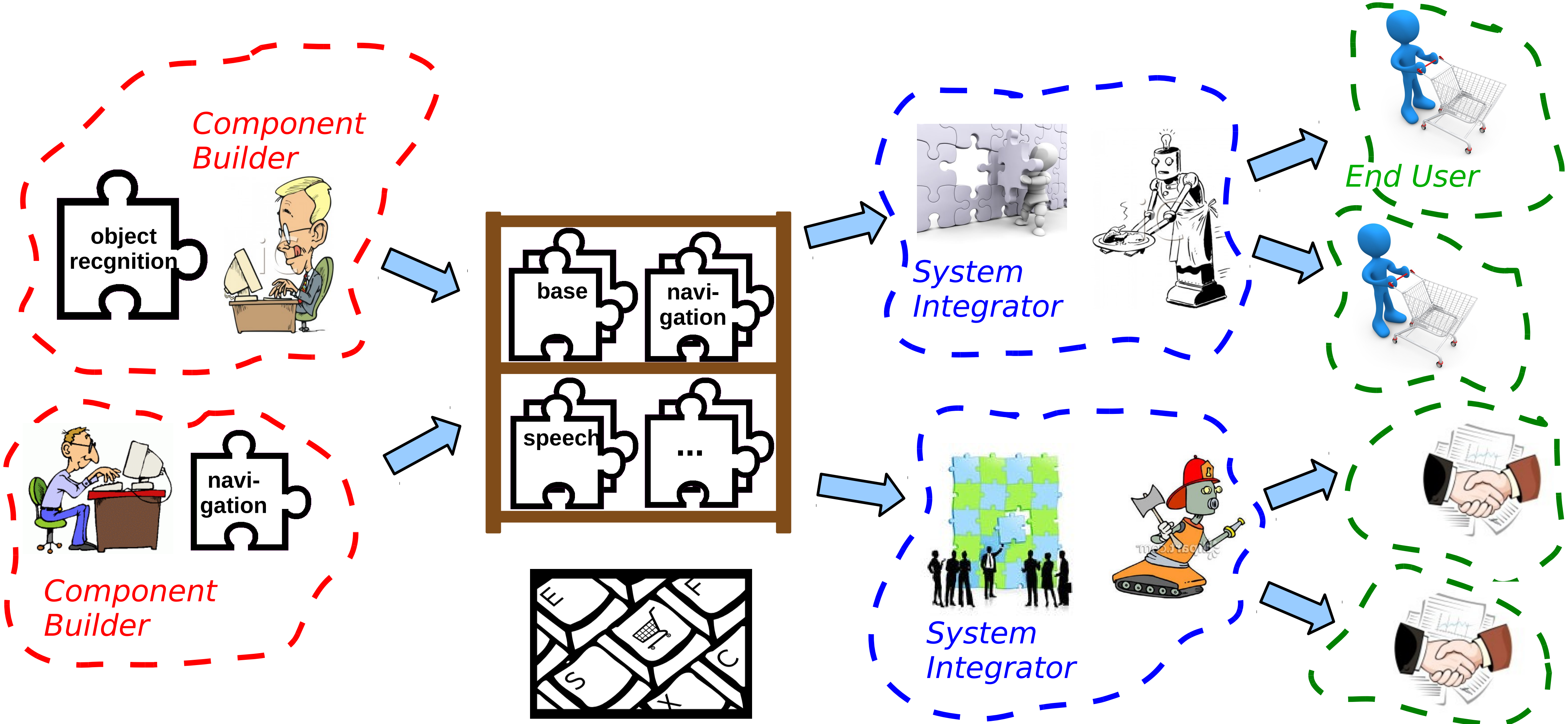
=> SWE already is bottleneck towards implementing service robotic applications in an economic and feasible way

=> SWE is a major hurdle when it comes to developing markets for service robots and economic success of service robotic applications

Goal:

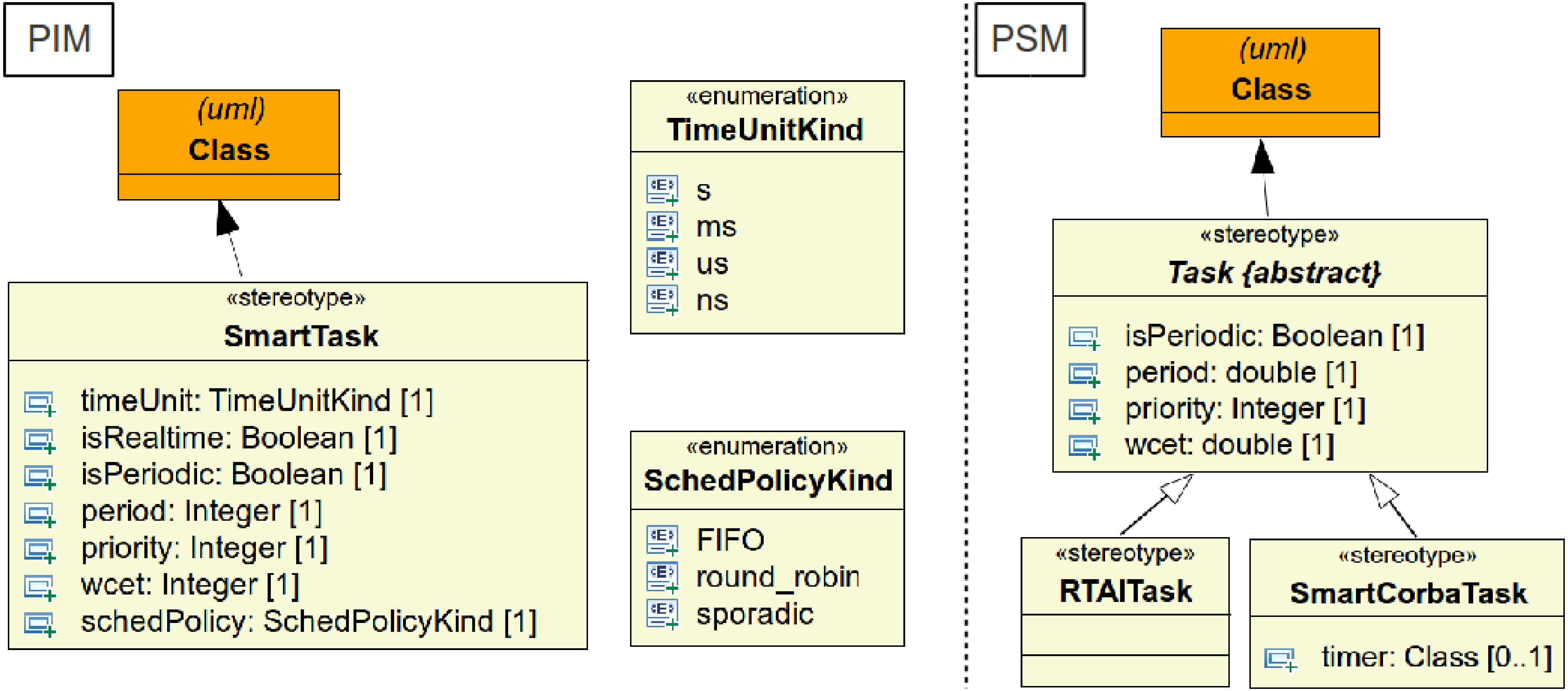
- reduce risks and costs of software development for advanced service robotic systems in order to make a step ahead towards economically feasible service robotic applications
- allow for re-use of software components
- plan ability of software components

Software Concepts for Service Robots: What we want to have: separation of roles, concerns



SmartSoft - MDSD details -

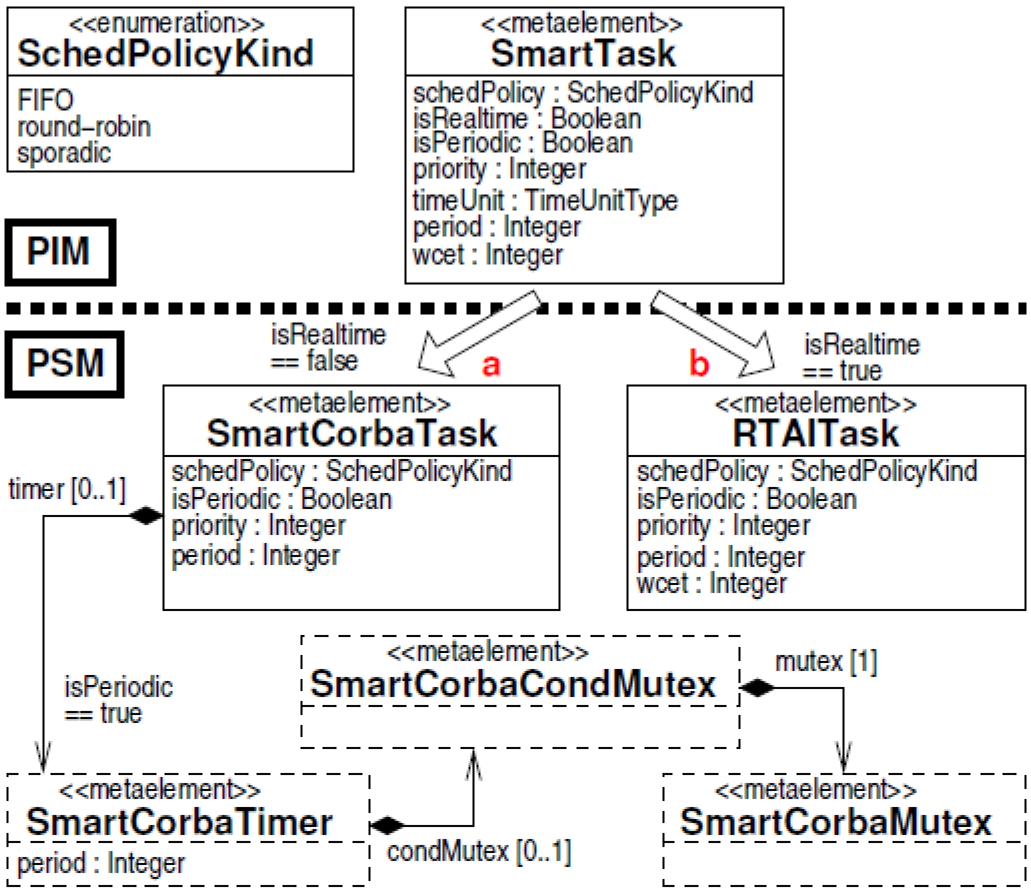
Model-Driven Software Development SmartMARS UML Profiles (PIM, PSM)

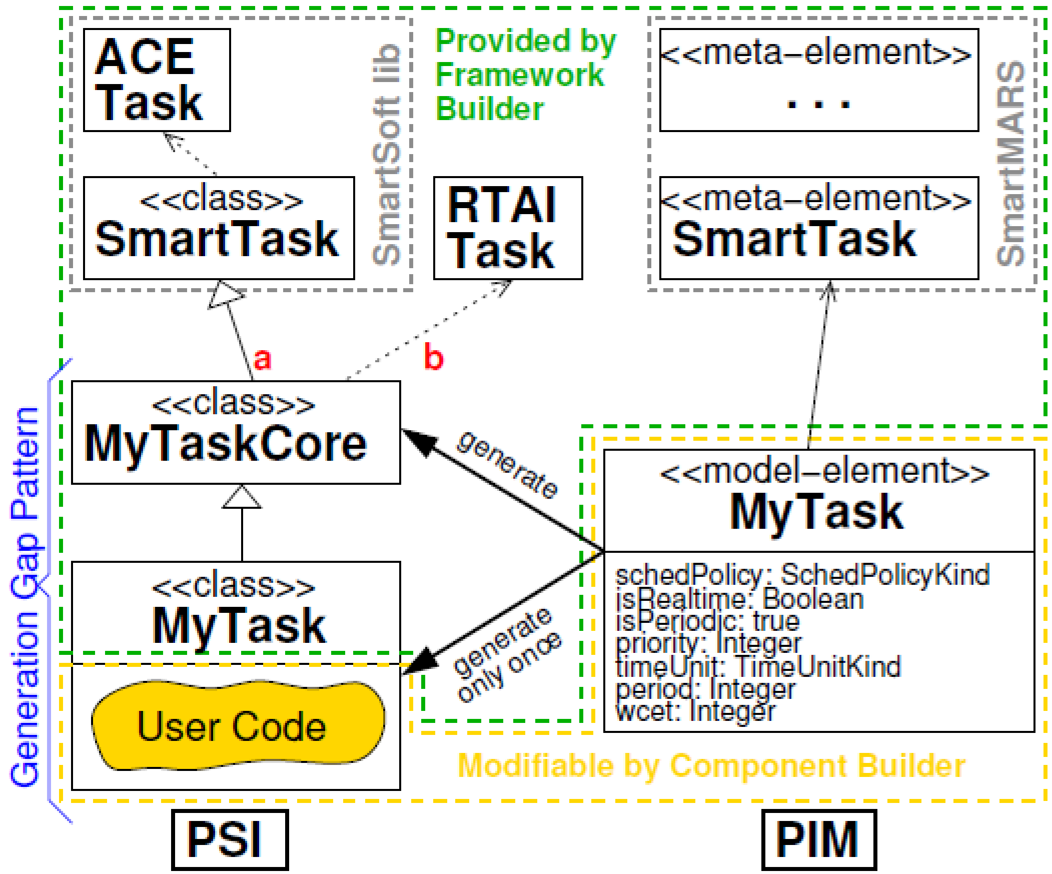


excerpts of UML Profile created with Papyrus UML (left PIM, right PSM)

SmartSoft - MDSD details -

Model-Driven Software Development
Model Transformation + Code Generation





Transformation PIM into PSM

Generation Gap Pattern

SmartSoft - MDSD details -

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

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"

SmartSoft - MDSD details -

Model-Driven Software Development
PSM to PSI

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

SmartSoft - approach -

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

SmartSoft - approach -

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.

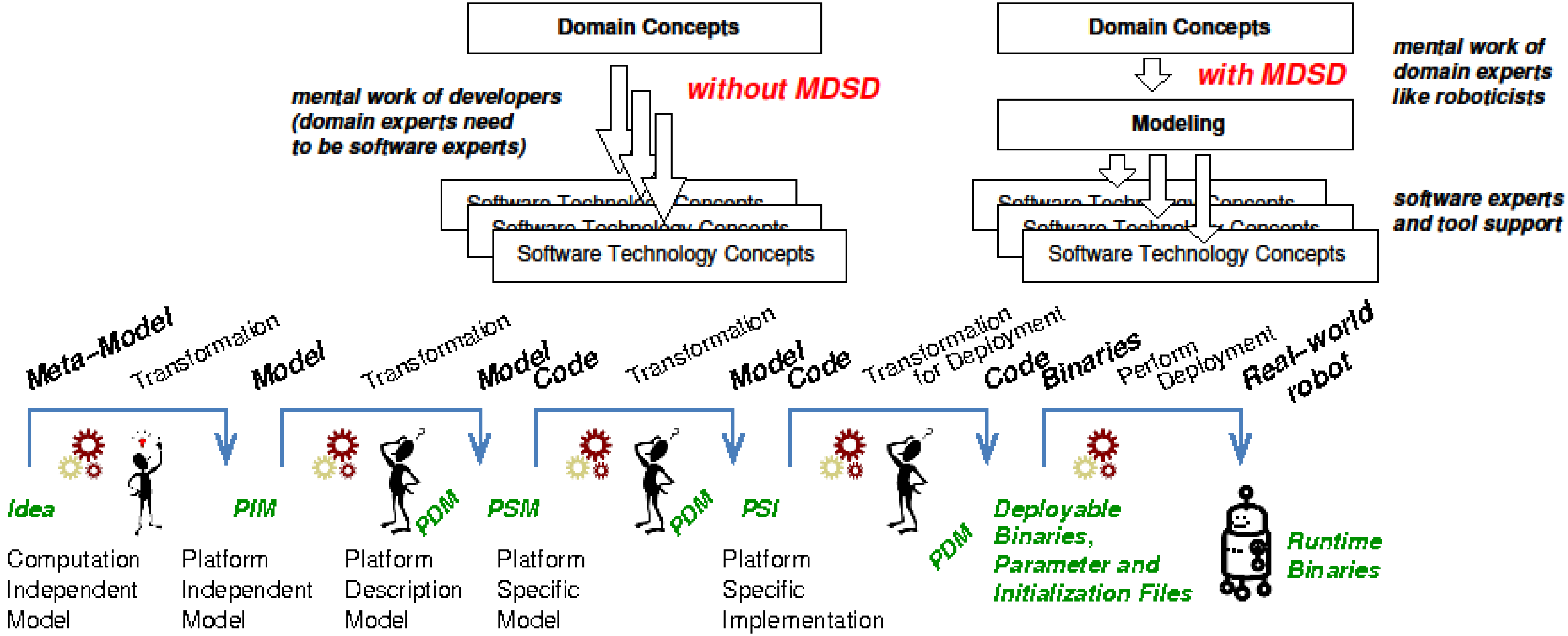
With SOA, it is critical to implement processes that ensure that there are at least two different and separate processes - for providers and consumers (Sprott & Wilkes, 2004).

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)

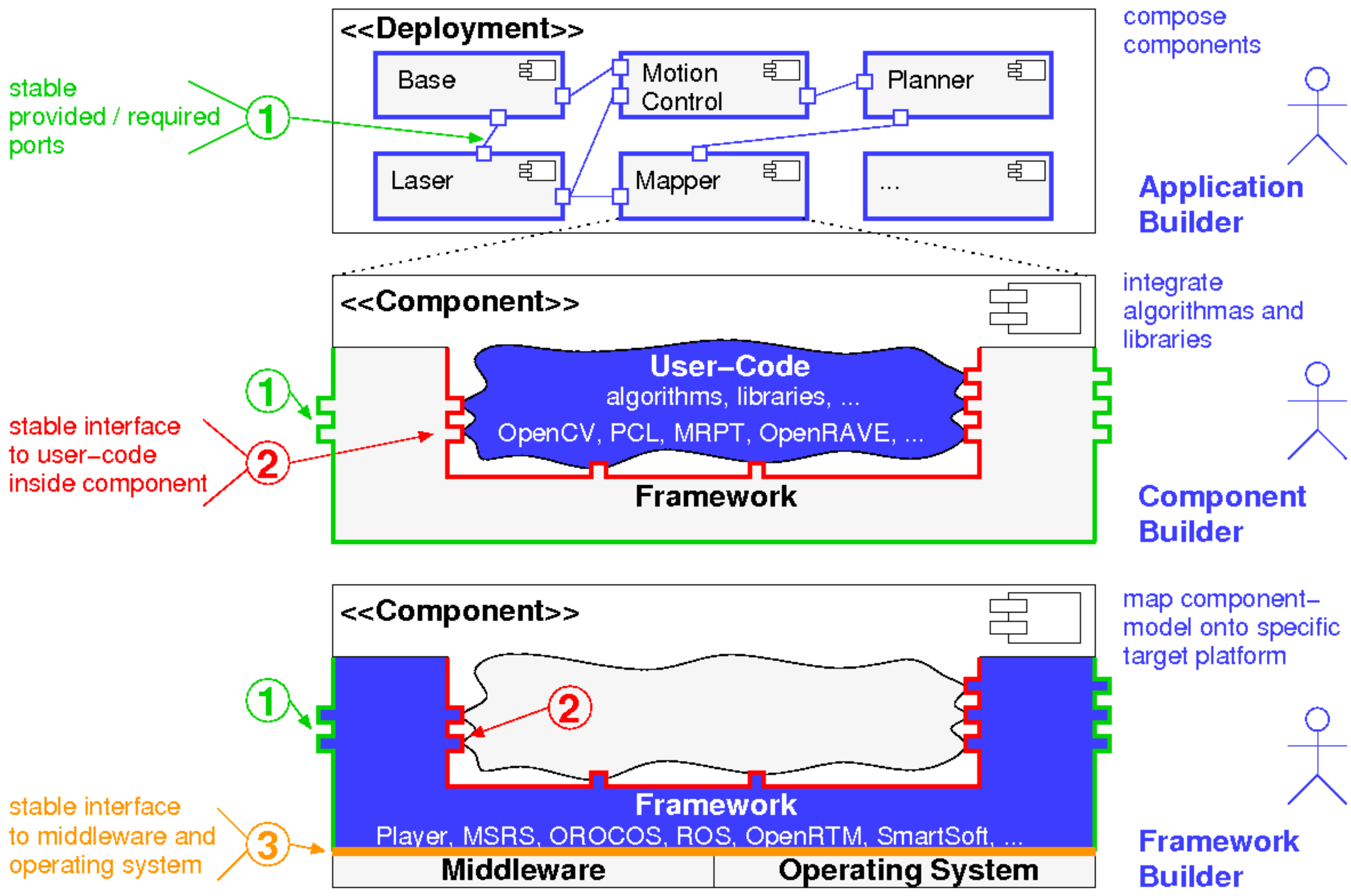
Where to start?
MDSD – Model-Driven SW Development

- 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



SmartSoft
- approach -

The SmartSoft Component Model
Stable Interfaces



System–Level (S)
- build system
- reuse components
- black–box view on components

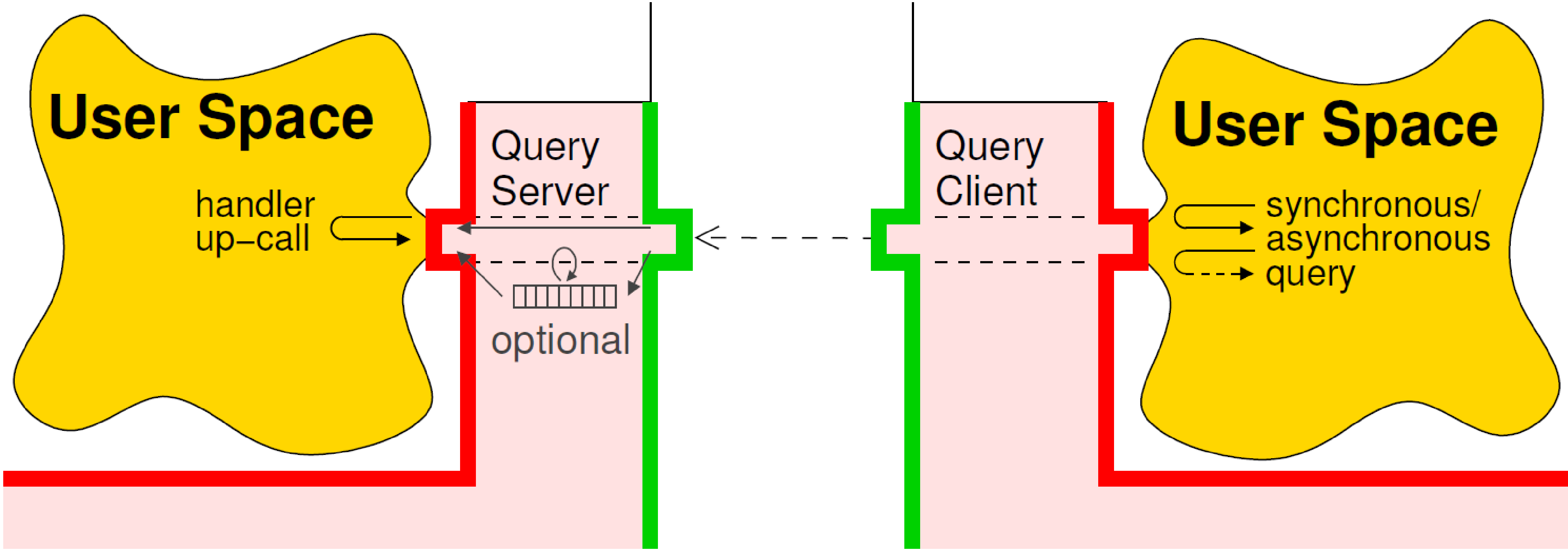
Component–Level (C)
- build component
- use framework
- black–box view on framework

Framework–Level (F)
- build framework
- abstract middleware and OS details
- black–box view on middleware / OS

servicerobotik ZAFH

Hochschule Ulm

The SmartSoft Component Model
Stable Interfaces

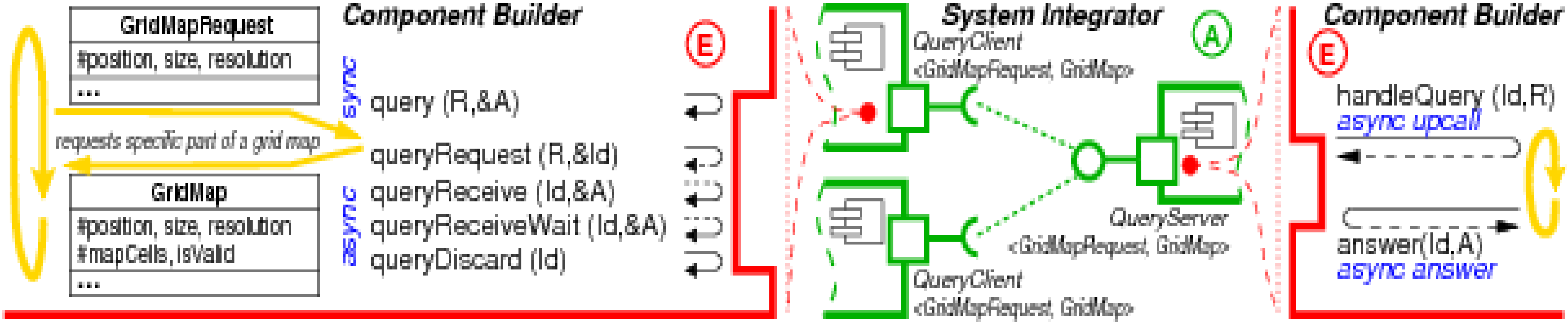


SmartSoft - details -

The SmartSoft Component Model
Stable Interfaces

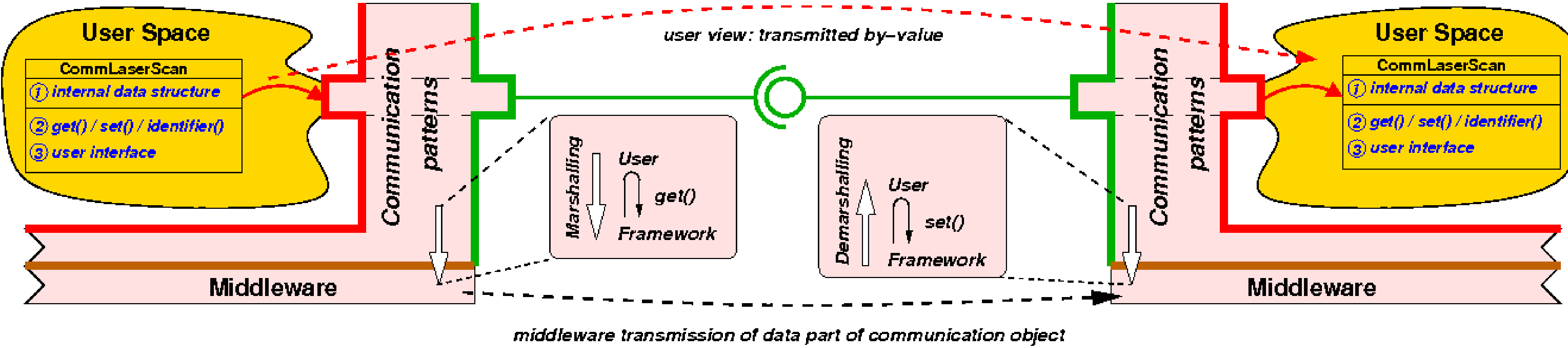
Query Client	R A
+ QueryClient(:SmartComponent*) throw(SmartError) + QueryClient(:SmartComponent*, server:const string&, service:const string&) throw(SmartError) + QueryClient(:SmartComponent*, port:const string&, slave:WiringSlave*) throw(SmartError) + *~QueryClient() throw() [virtual]* + add(:WiringSlave*, 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



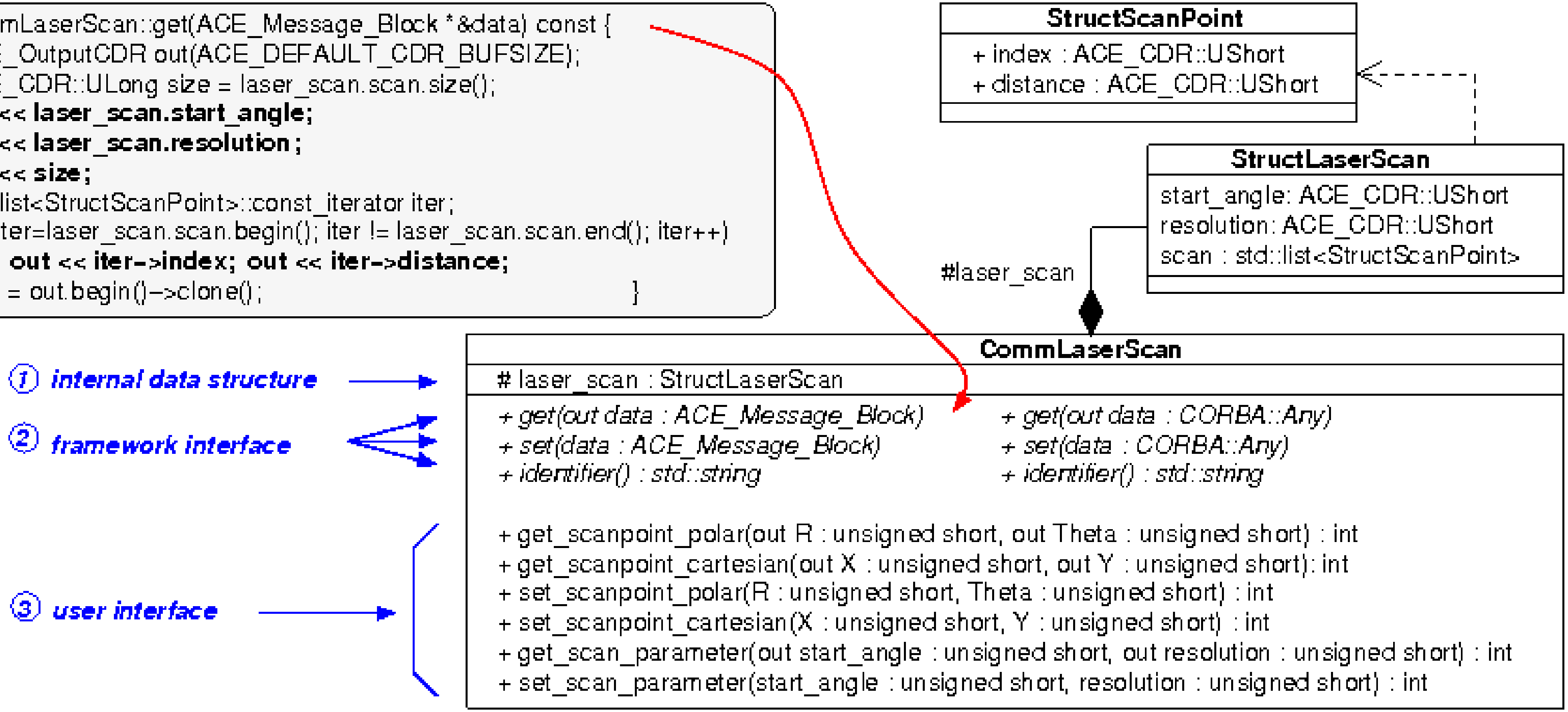
SmartSoft
- details -

SmartSoft
Technical Details



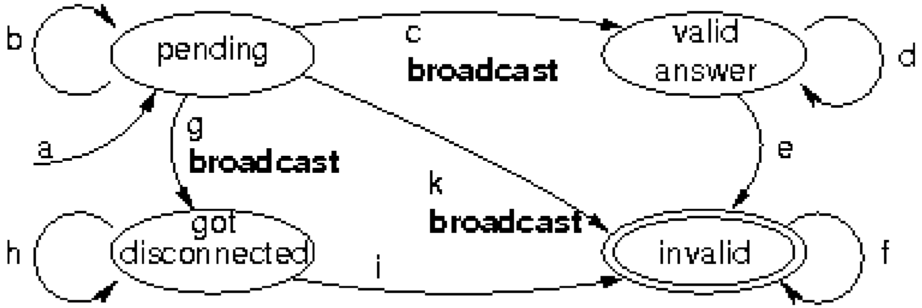
SmartSoft
Technical Details

```
void CommLaserScan::get(ACE_Message_Block *&data) const {
        ACE_OutputCDR out(ACE_DEFAULT_CDR_BUFSIZE);
        ACE_CDR::ULong size = laser_scan.scan.size();
        out << laser_scan.start_angle;
        out << laser_scan.resolution;
        out << size;
        std::list<StructScanPoint>::const_iterator iter;
        for (iter=laser_scan.scan.begin(); iter != laser_scan.scan.end(); iter++)
                out << iter->index; out << iter->distance;
        data = out.begin()->clone();                                     }
```



SmartSoft
- details -

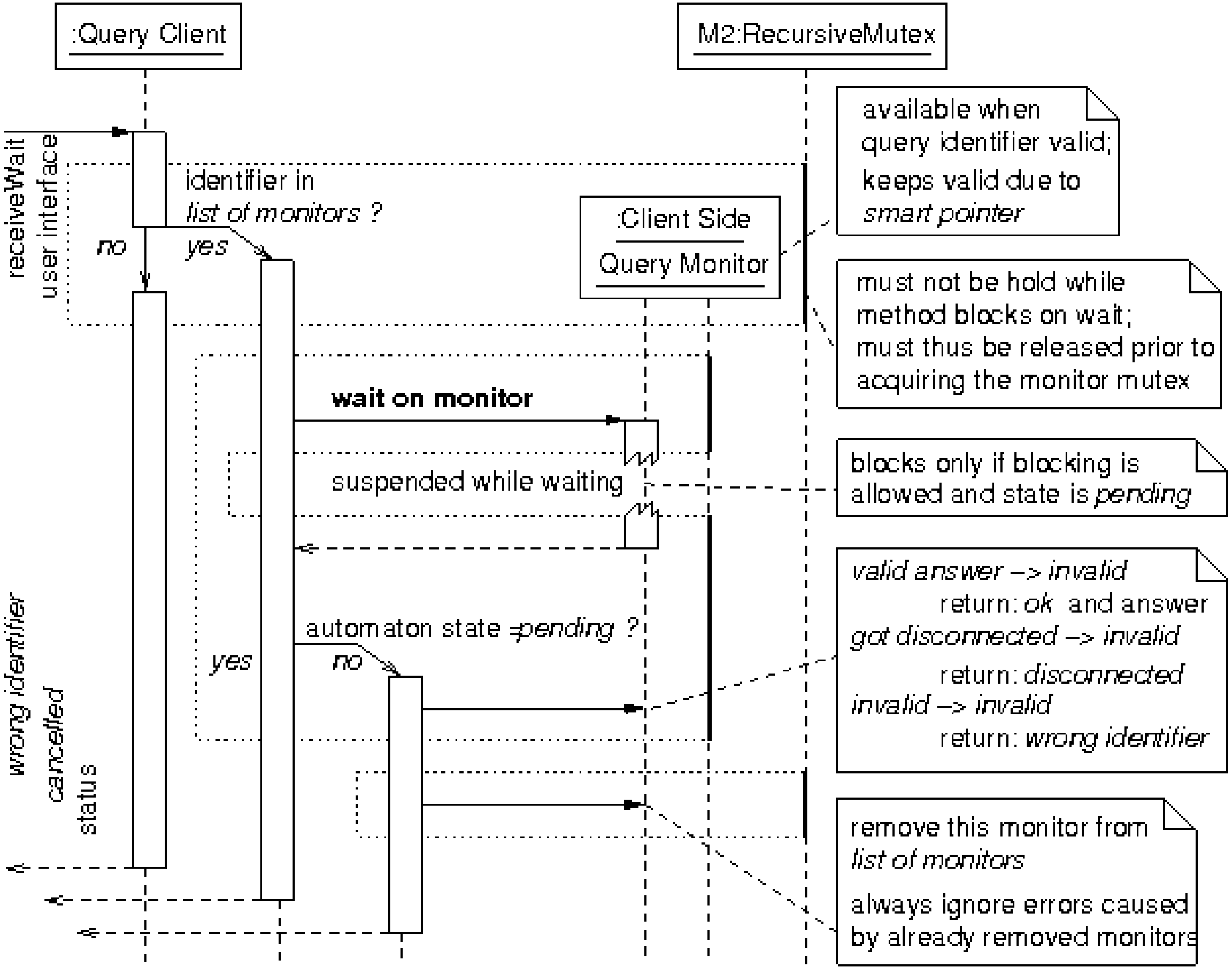
SmartSoft
Technical Details

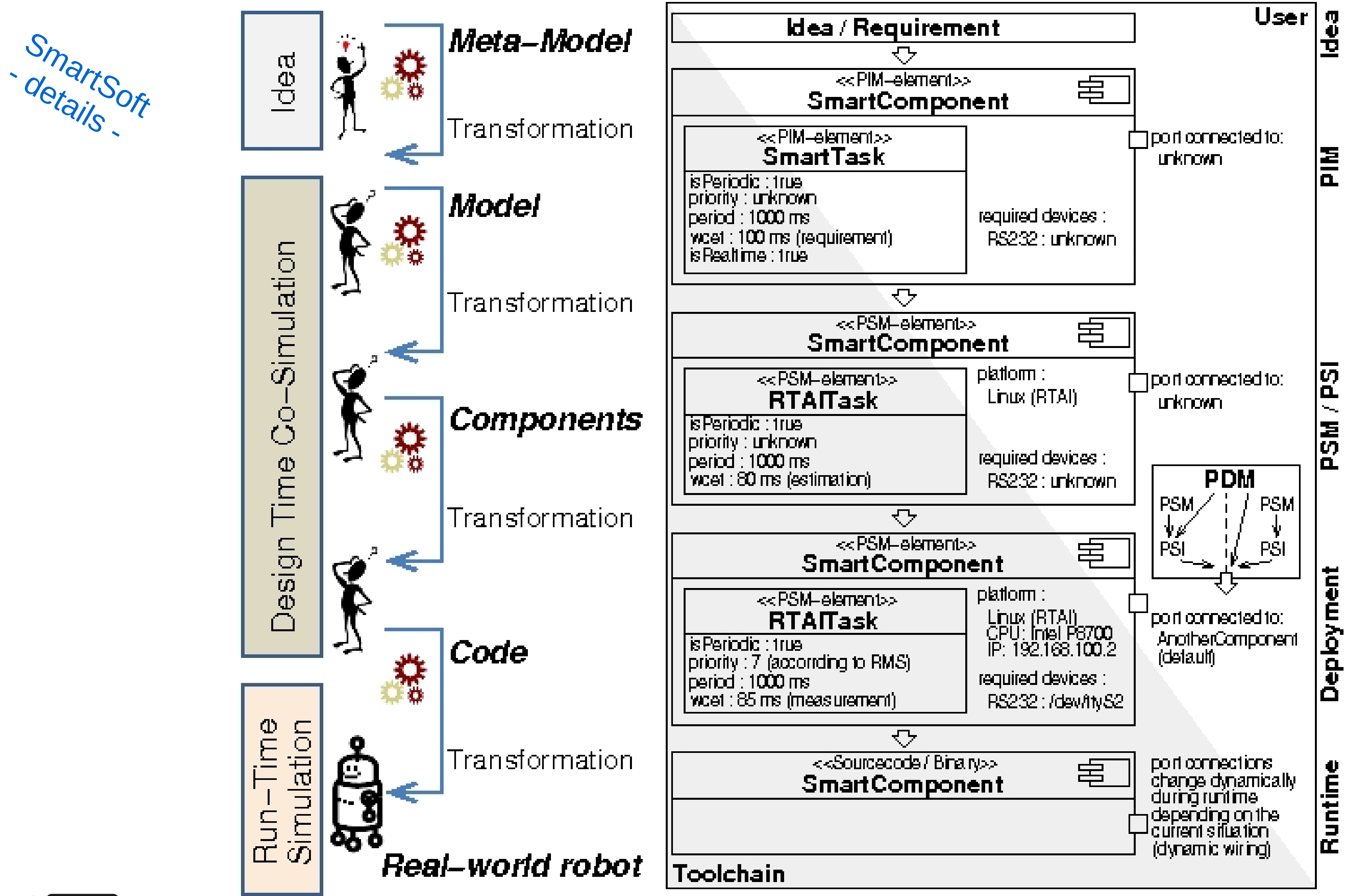


Query: client side state automaton (per request)

SmartSoft
- details -

SmartSoft
Technical Details





SmartSoft
- details -
Idea
Design Time Co-Simulation
Run-Time Simulation
Meta-Model
Transformation
Model
Transformation
Components
Transformation
Code
Transformation
Real-world robot
User
Idea / Requirement
<<PIM-element>>
SmartComponent
<<PIM-element>>
SmartTask
isPeriodic : true
priority : unknown
period : 1000 ms
wcet : 100 ms (requirement)
isRealtime : true
required devices :
RS232 : unknown
port connected to:
unknown
<<PSM-element>>
SmartComponent
<<PSM-element>>
RTAITask
isPeriodic : true
priority : unknown
period : 1000 ms
wcet : 80 ms (estimation)
platform :
Linux (RTAI)
required devices :
RS232 : unknown
port connected to:
unknown
PDM
PSM
PSI
PSM
PSI
<<PSM-element>>
SmartComponent
<<PSM-element>>
RTAITask
isPeriodic : true
priority : 7 (according to RMS)
period : 1000 ms
wcet : 85 ms (measurement)
platform :
Linux (RTAI)
CPU: Intel P8700
IP: 192.168.100.2
required devices :
RS232 : /dev/ttyS2
port connected to:
AnotherComponent
(default)
<<Sourcecode / Binary>>
SmartComponent
port connections
change dynamically
during runtime
depending on the
current situation
(dynamic wiring)
Toolchain
Idea
PIM
PSM / PSI
Deployment
Runtime

SmartSoft
- details -

Model Driven Software Development
Glueing User Code / Legacy Code

*PlayerPushTimedHandler.cc

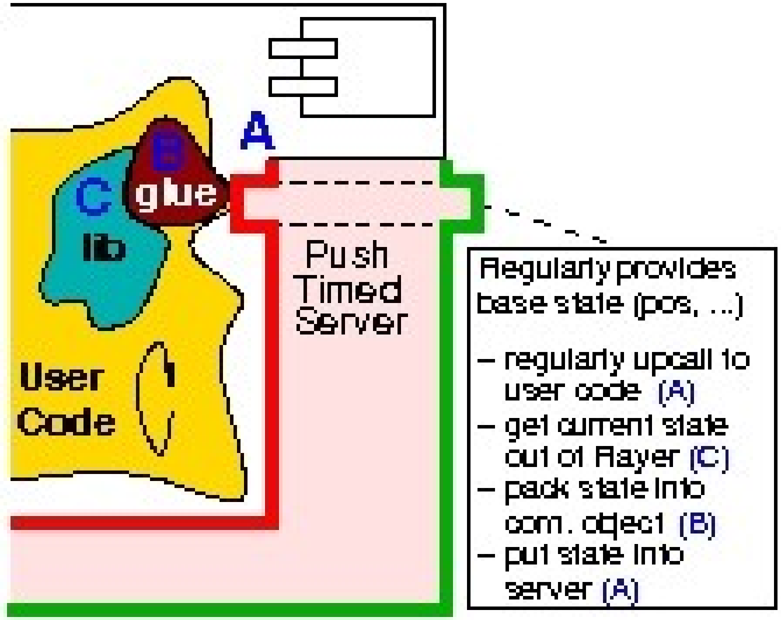
```
#include <iostream>

void PlayerPushTimedHandler::handlePushTimer(
        CHS::PushTimedServer<Smart::CommBaseState> & server) throw()
{
    COMP->PlayerClientMutex.acquire();
    COMP->robot->ReadIfWaiting();
    double xPos = COMP->position_2d_proxy->GetXPos();
    double yPos = COMP->position_2d_proxy->GetYPos();
    double aPos = COMP->position_2d_proxy->GetYaw();
    double xSpeed = COMP->position_2d_proxy->GetXSpeed();
    double ySpeed = COMP->position_2d_proxy->GetYSpeed();
    double yawSpeed = COMP->position_2d_proxy->GetYawSpeed();
    COMP->PlayerClientMutex.release();

    base_position.set_x(xPos, 1.0);
    base_position.set_y(yPos, 1.0);
    base_position.set_base_alpha(aPos);
    base_velocity.set_v(((xSpeed + ySpeed) / 2.0) * 1000.0);
    base_velocity.set_omega_base(yawSpeed);
    base_state.set_base_position(base_position);
    base_state.set_base_velocity(base_velocity);

    server.put(base_state);
```

A

C

B

A



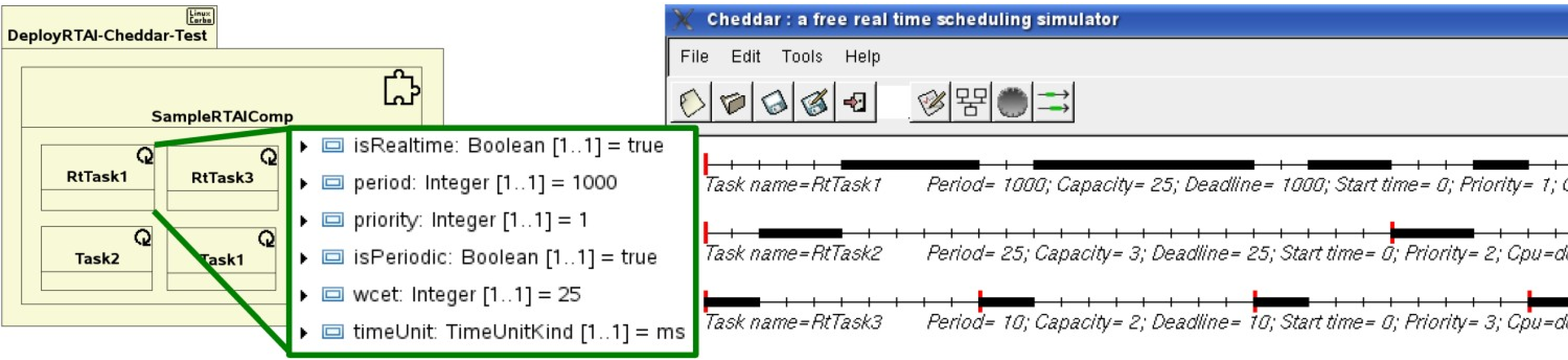
SmartSoft - details -

Model-Driven Software Development System-Integrator View / Deployment

System Level Properties / Bindings / Conformance Checks

Resource Awareness and Quality of Service

- Example: Schedulability Analysis (CHEDDAR)

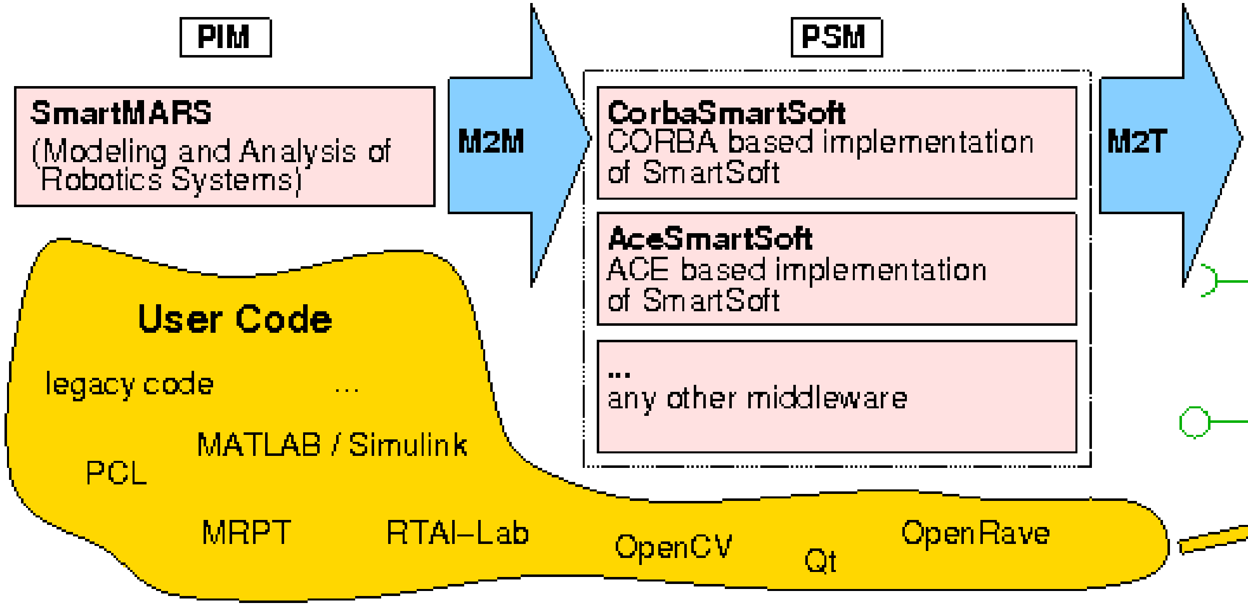


SmartSoft
- details -

Model-Driven Software Development SmartMDSD

Illustration of the Development Process

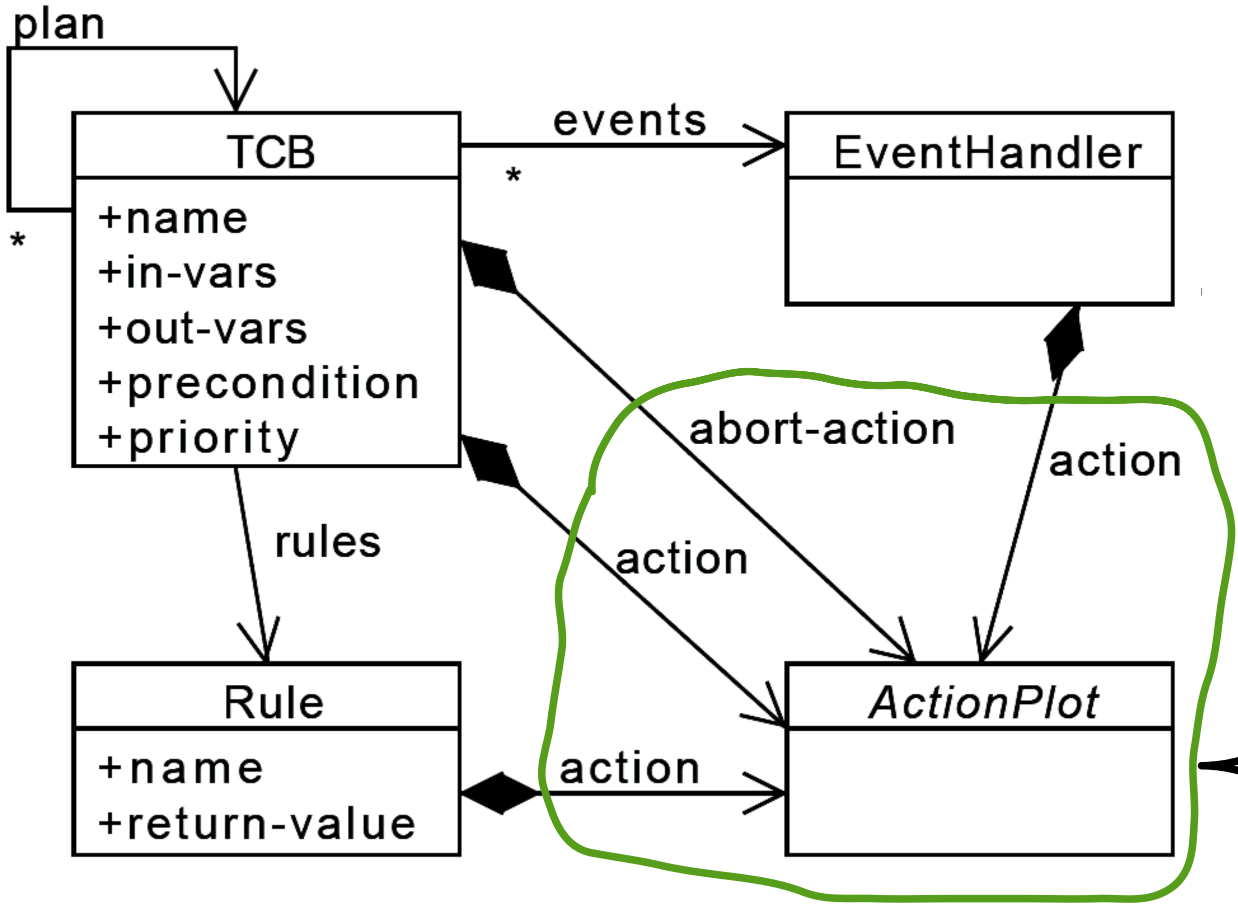
- Implemented as UML 2.0-Profile for Robotics Software Components
- supports Component Development, System Integration, Deployment
- based on standards: UML 2.0, Papyrus, Eclipe Modeling Project, etc.
- different Runtime-Platforms, Middleware-Systems etc.



2-step transformation workflow (framework builder view)

SmartTCL

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



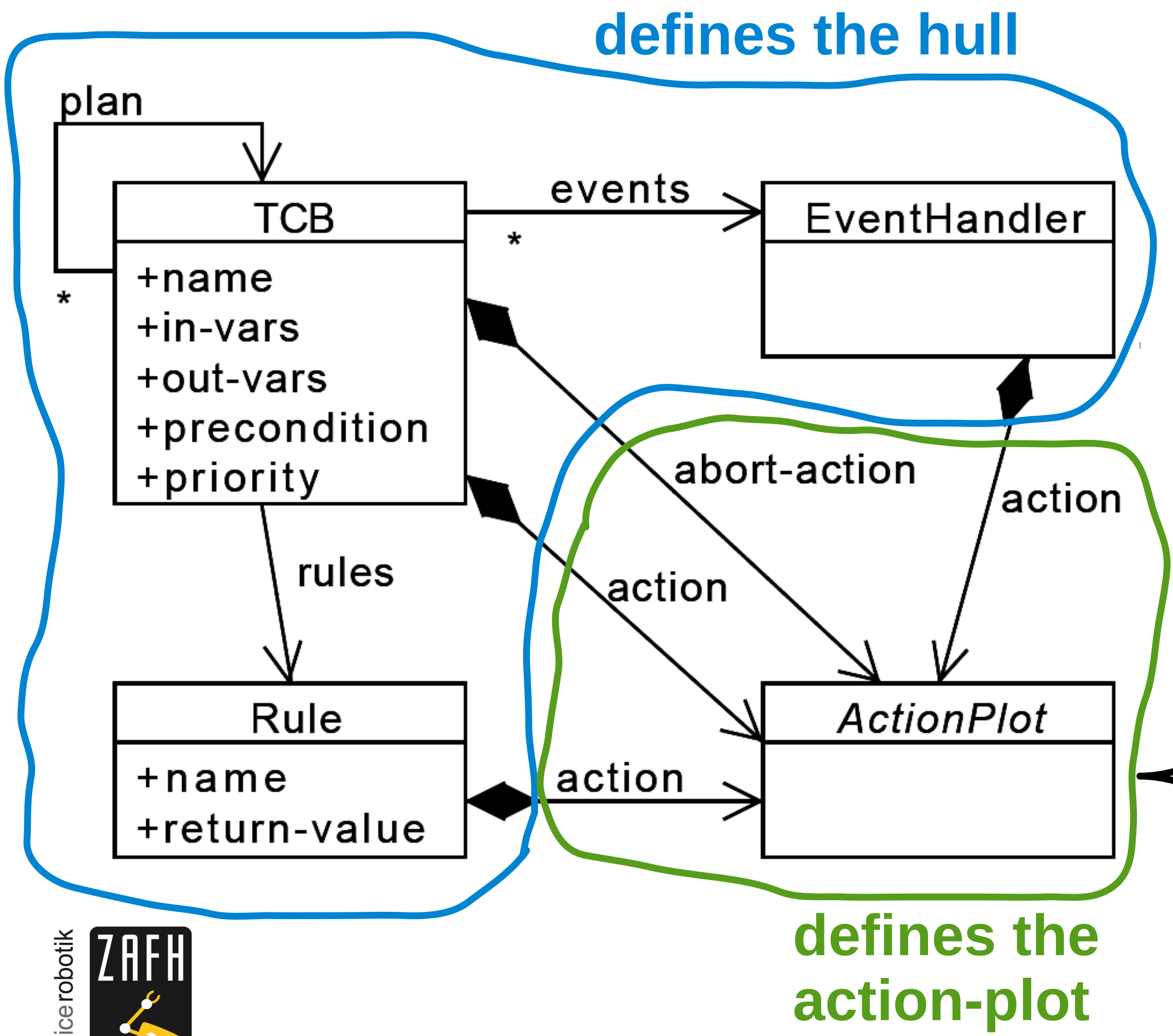
defines the action-plot

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
  - ...

SmartTCL

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
  - ...