Towards Model-Driven Software Development in Robotics: Motivation, Perspectives, Benefits, Challenges

Prof. Dr. Christian Schlegel
Fakultät Informatik
Hochschule Ulm
Germany

Alex Lotz

Matthias Lutz

Dennis Stampfer

me


ZAFH
servicerobotik

The Software Challenge in Robotics…

http://www.robotshop.com/blog/en/robot-humor-dinner-comic-524

The Software Challenge in Robotics…

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")

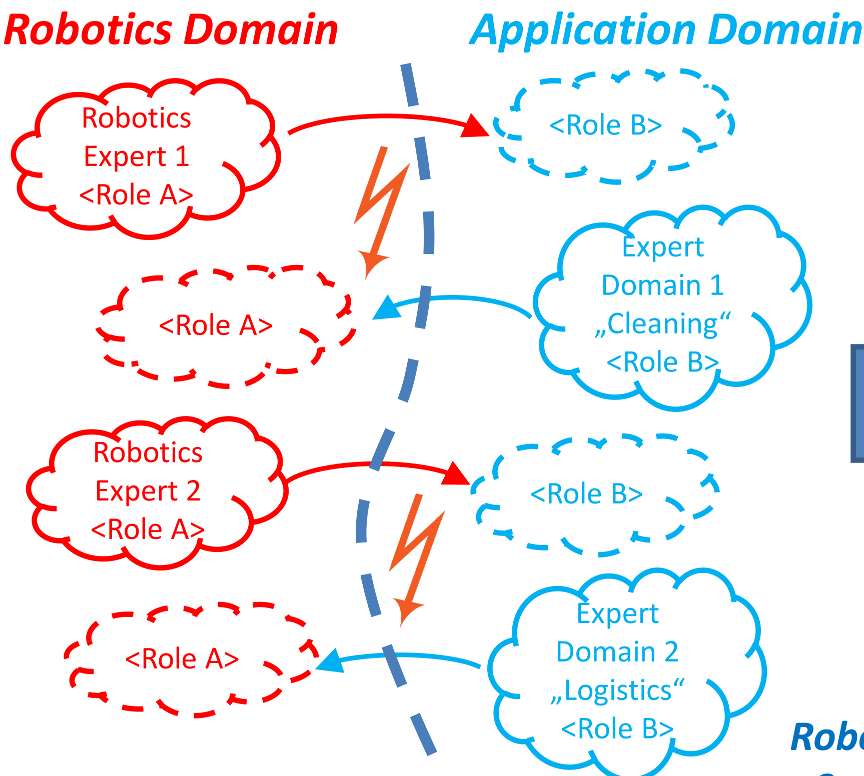
=> SWE already is bottleneck towards implementing service robotic applications in an economic and feasible way

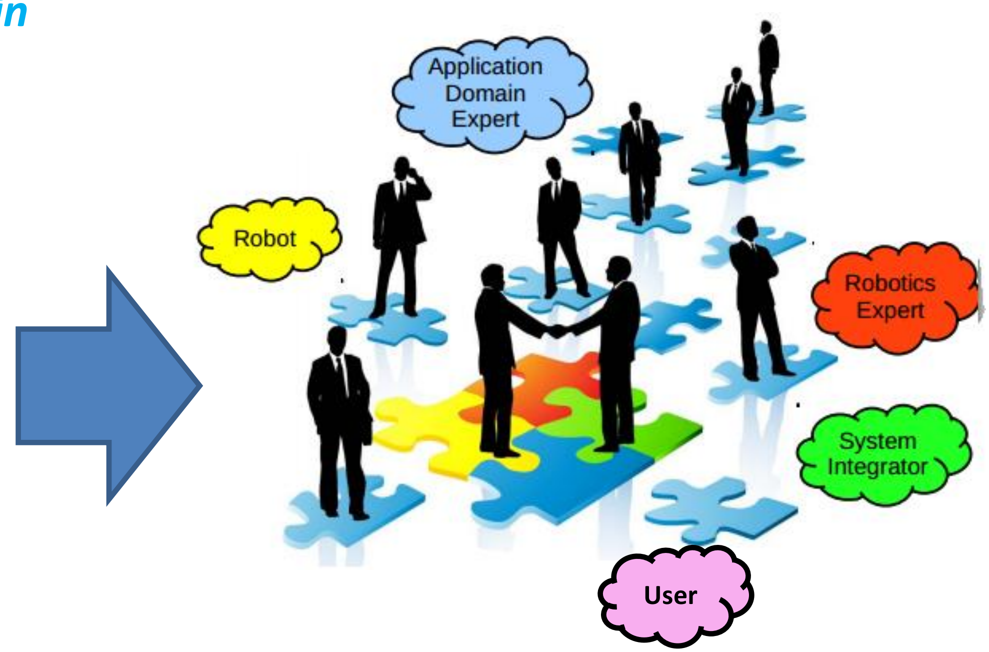
=> SWE is a major hurdle when it comes to developing markets for service robots and economic success of service robotic applications

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
- prepare for a robotics software business ecosystem

Towards a Software Business Ecosystem in Robotics…



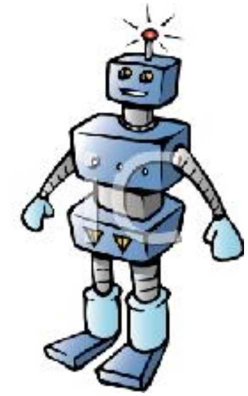


Robotics Business Ecosystem

- ***Separation of Roles (in order to support specialization)***
- ***Separation of Concerns (in order to reduce complexity)***
  - computation (functionality)
  - communication (exchange data between entities)
  - configuration (parameters at component and system level, wiring)
  - coordination (orchestration, resource management, wiring)
- ***MDSD (Model-Driven SW Development)***

Is Robotics Different from other Domains???

AUTOSAR AUTomotive Open System ARchitecture



- the *differences* 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

- the *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 ist disposal

⇒ the best matching between current situation, proper robot behavior and resource assignment becomes overwhelming even for the most skilled robot engineer!

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

How to improve the execution quality of a service robot acting in open-ended environments given limited onboard resources?

Example: ***Optimize coffee delivery service***

1. guarantee minimum coffee temperature (preference is to serve as hot as possible)
2. maximum velocity bound due to safety issues (hot coffee) and battery level
3. minimum required velocity depending on distance since coffee cools down
4. fast delivery can increase volume of coffee sales

Robotics engineer / design-time

- identify and enumerate all eventualities in advance???
- code proper configurations, resource assignments and reactions for all situations???

➢ *not efficient due to the combinatorial explosion of situations & parameterizations*

➢ *even the most skilled robotics engineer cannot foresee all eventualities*

Robot / run-time:

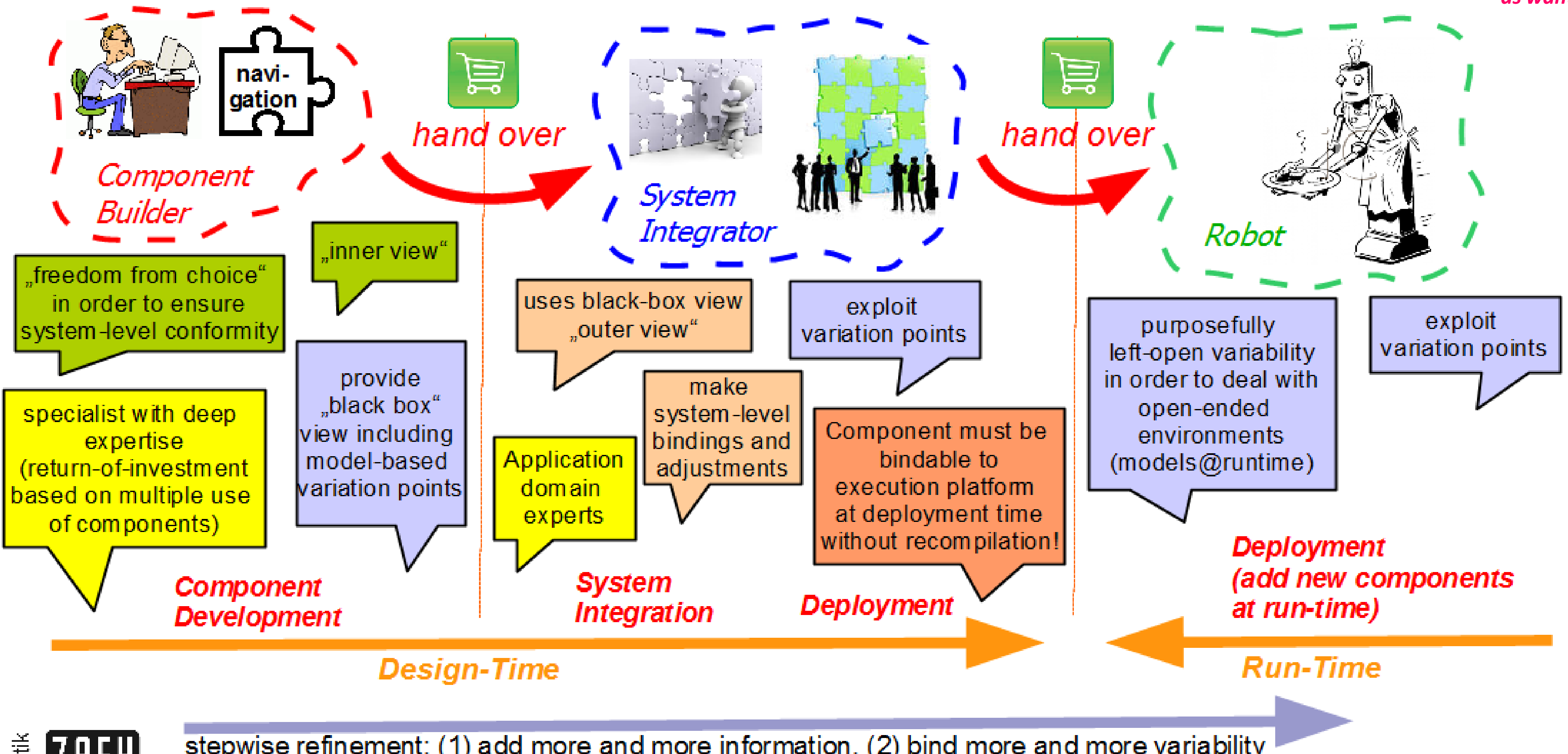
- just (re)plan in order to take into account latest information as soon as it becomes available???

➢ *complexity of (re)planning is far too high when it comes to real-world problems*

(not possible to generate action plots given partial information only while also taking into account additional properties like, e.g. safety and resource awareness)

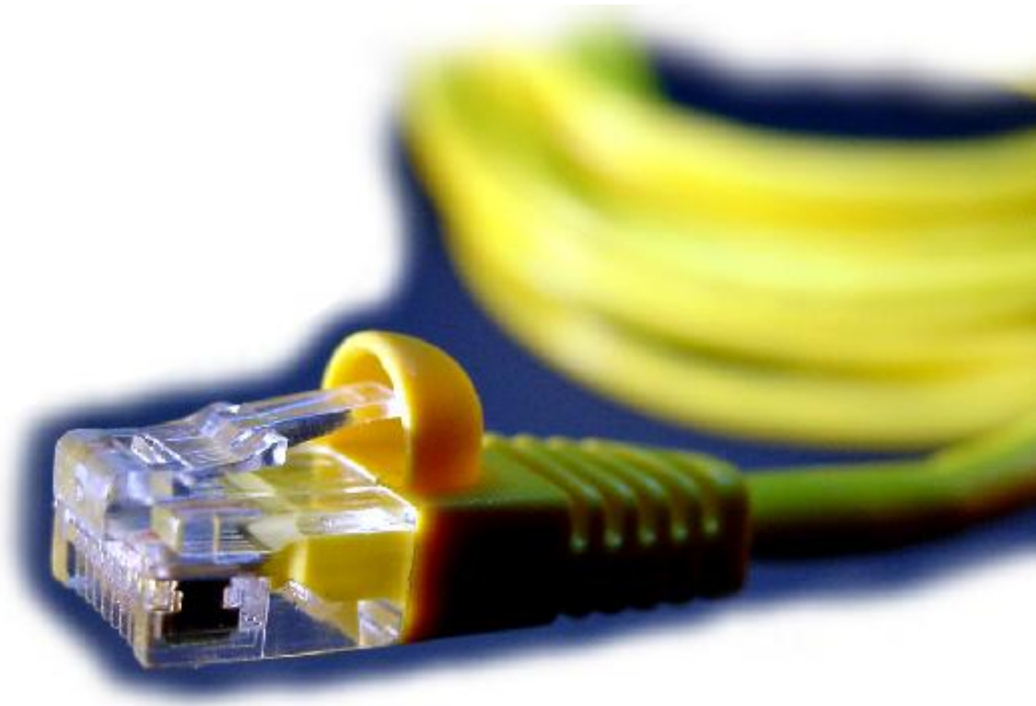
Towards a Software Business Ecosystem in Robotics…

- *Use models for the entire life-cyle of the robot*
- *Models are refined step-by-step until finally they become executable*
- *Separate inside view (component builder) from outside view (system integrator)*
- *Separate stable execution container from implementational technologies (middleware, OS)*
- *Variation points: design-time (component builder, system integrator), runtime (robot)*
  - *Explicitly model variability for late binding (by system integrator and even by the robot at runtime)*



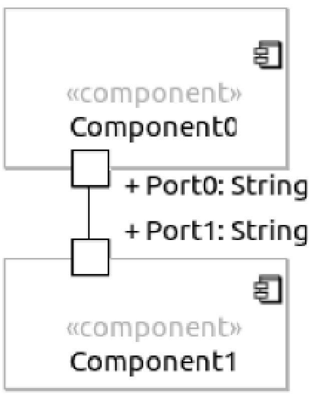
A Robotics Software Component Model as Basis…
… **Communication Patterns** as Major Ingredients

- Communication
  - Send
  - Query
  - PushNewest
  - PushTimed
- Coordination / Configuration
  - State (Lifecycle)
  - DynamicWiring
  - Parameter
  - Event
  - Monitoring

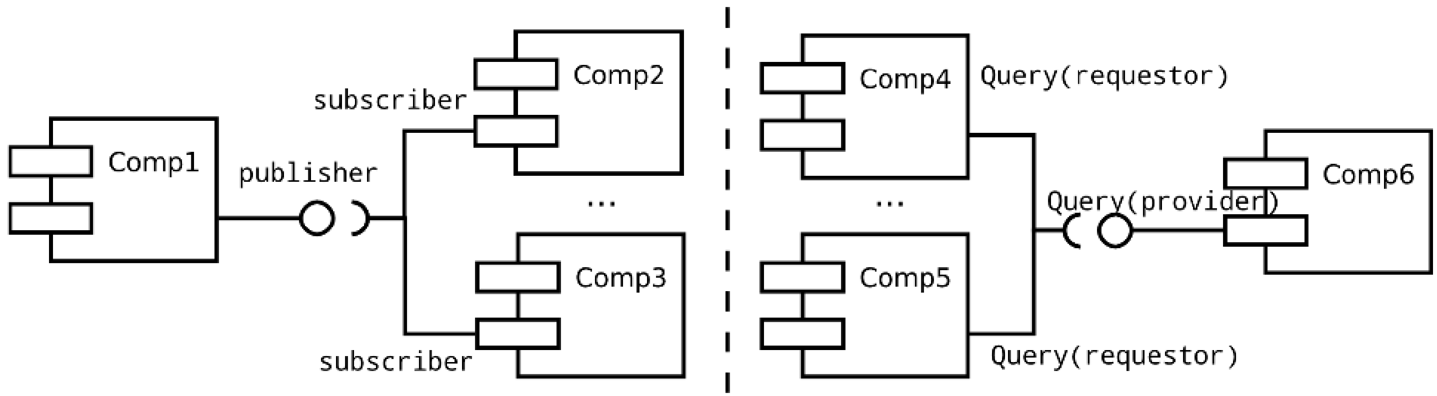


Robotics Software Component Model + MDSD…

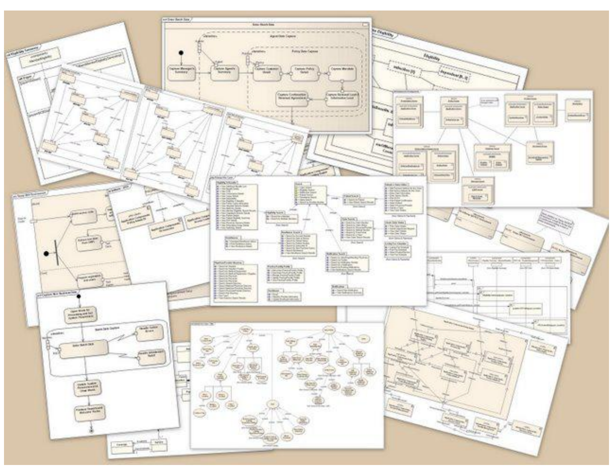
UML:



Robotics:



But: Communication Semantics? Protocols? Policies? QoS? Interface (inside of component)? Internal Buffers/Handlers?



Wikipedia Commons / Kishorekumar 62

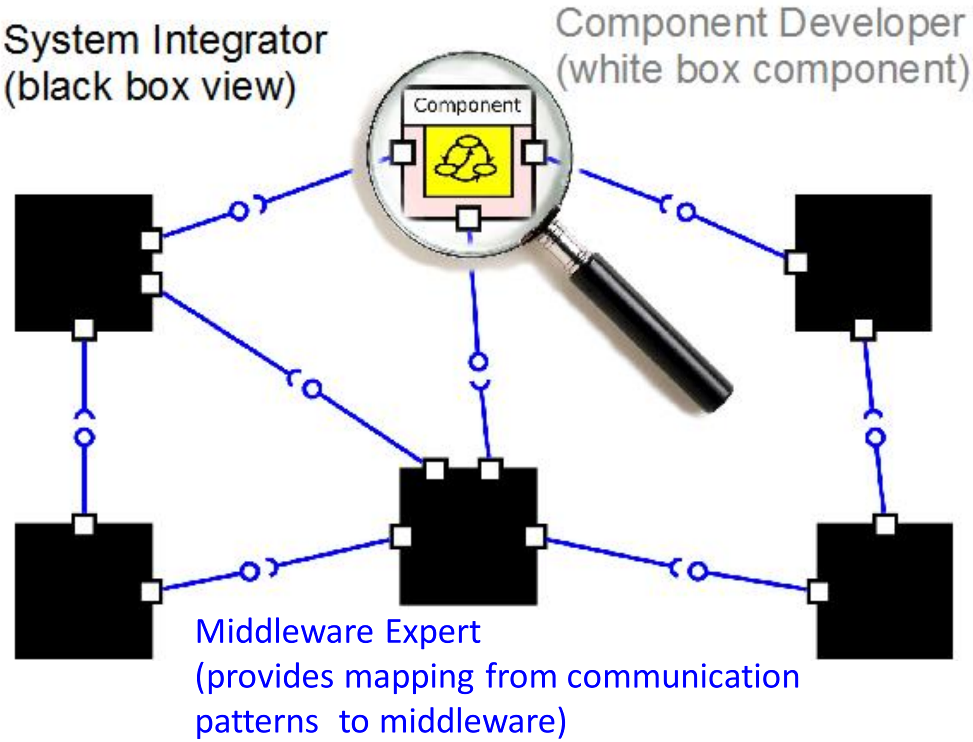
Robotics Software Component Model + MDSD…

Gain control over component hull

all relevant properties and parameters explicated at the component hull

*Think **SOA** rather than message centric:*

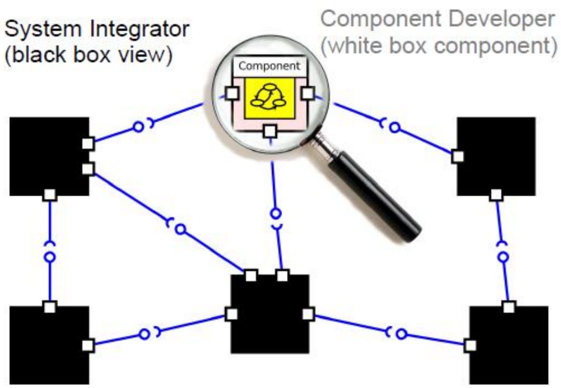
A SOA (service-oriented architecture) has to ensure that services don‘t get reduced to the status of interfaces, rather they have an identity of their own



Principles of good service design: [Sprott&Wilkes, 2004]

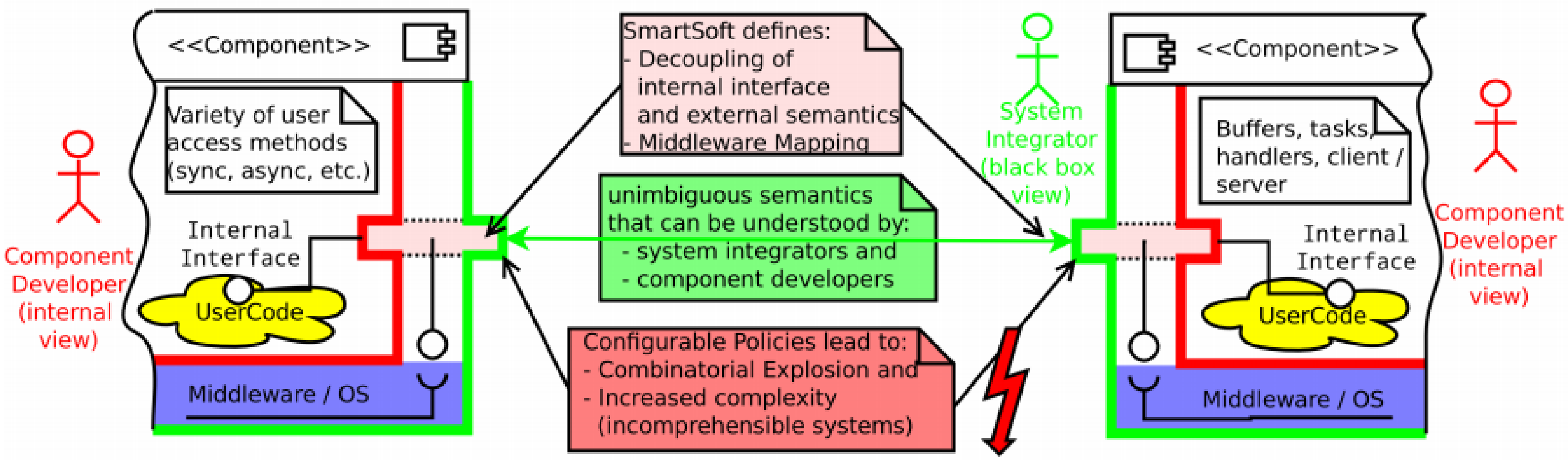
- **reusable**: use of service, not reuse by copying of code / implementation
- **abstracted**: service is abstracted from the implementation
- **published**: precise, published specification functionality of service interface, not implementation
- **formal**: formal contract between endpoints places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

Robotics Software Component Model + MDSD…



Communication:
- *Send*
- *Query*
- *Push Newest*
- *Push Timed*

Coordination / Configuration:
- *State (Lifecycle)*
- *Dynamic Wiring*
- *Parameter*
- *Event*
- *Monitoring*



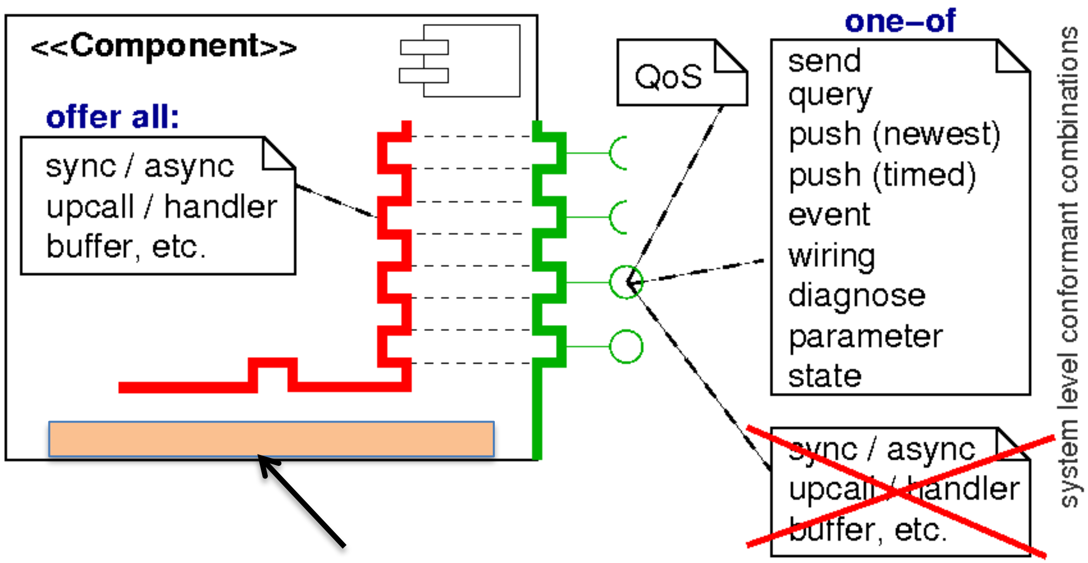
We decide on a small set of commnuication patterns, each binding a reasonable and consistent configuration and give them names!

→ Now system integrators understand the system and component developers know how to use the services!

Communication Patterns:
Mastering the link between component inside / outside view

not variety outside where it affects system integration, but

- *avoid complexity of combinatorial explosion of policies, mechanisms etc.*
- *ensure system level conformance (avoid distributed systems deadlocks etc.)*
- *avoid incompatible port variants of the same service*



Middleware / OS binding:
SmartSoft, OROCOS, ROS, RobotML, …
ACE, CORBA, DDS, Linux, Windows, …

not variety outside where it affects system integration

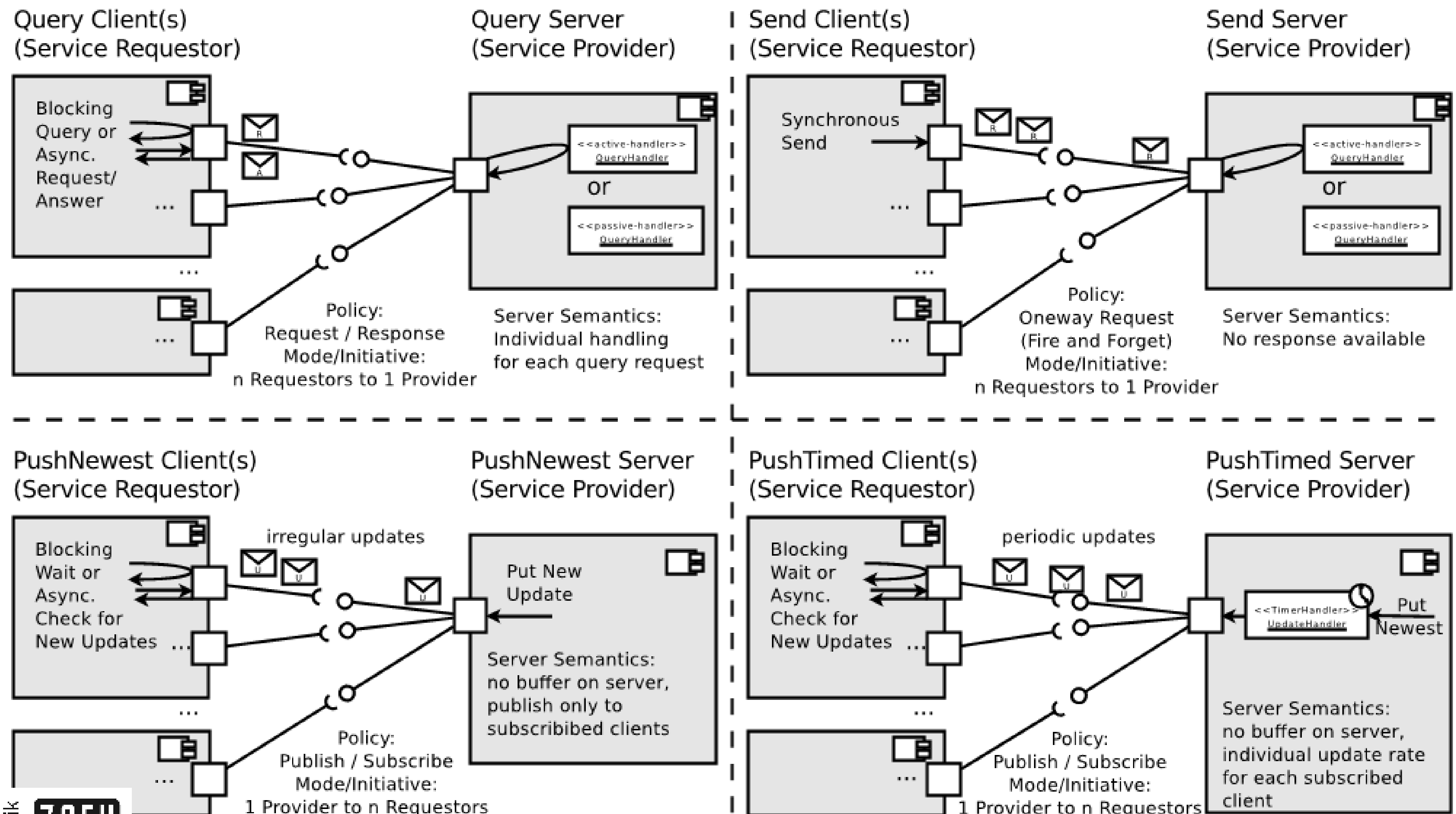
but variety inside a component to ease job of developer:

- *give freedom to use desired access methods (sync, async, upcall, etc.)*
- *give freedom to install desired processing (passive, thread-pool, pipeline, buffers, etc.)*

not early platform binding

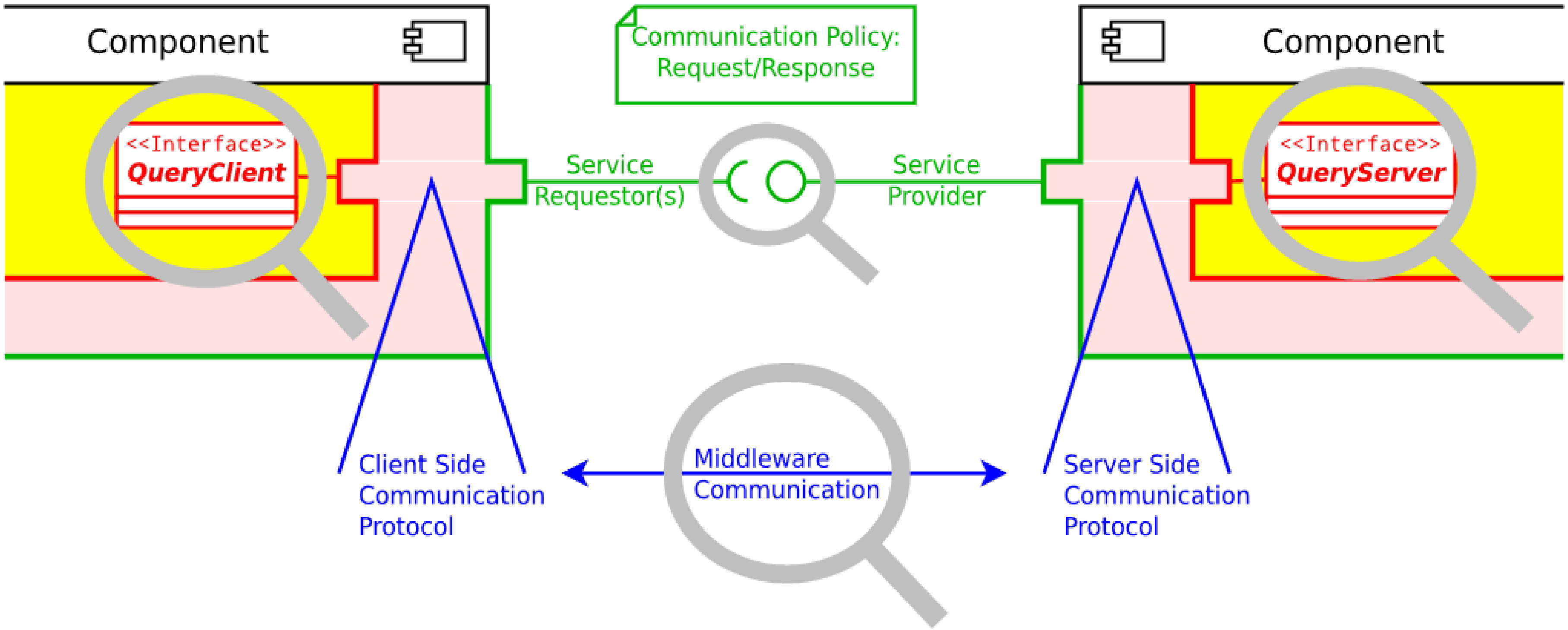
but late linking to implementation of execution container

Overview Communication: Query / Send / Push Newest / Push Timed (there are 5 more patterns for coordination / configuration)



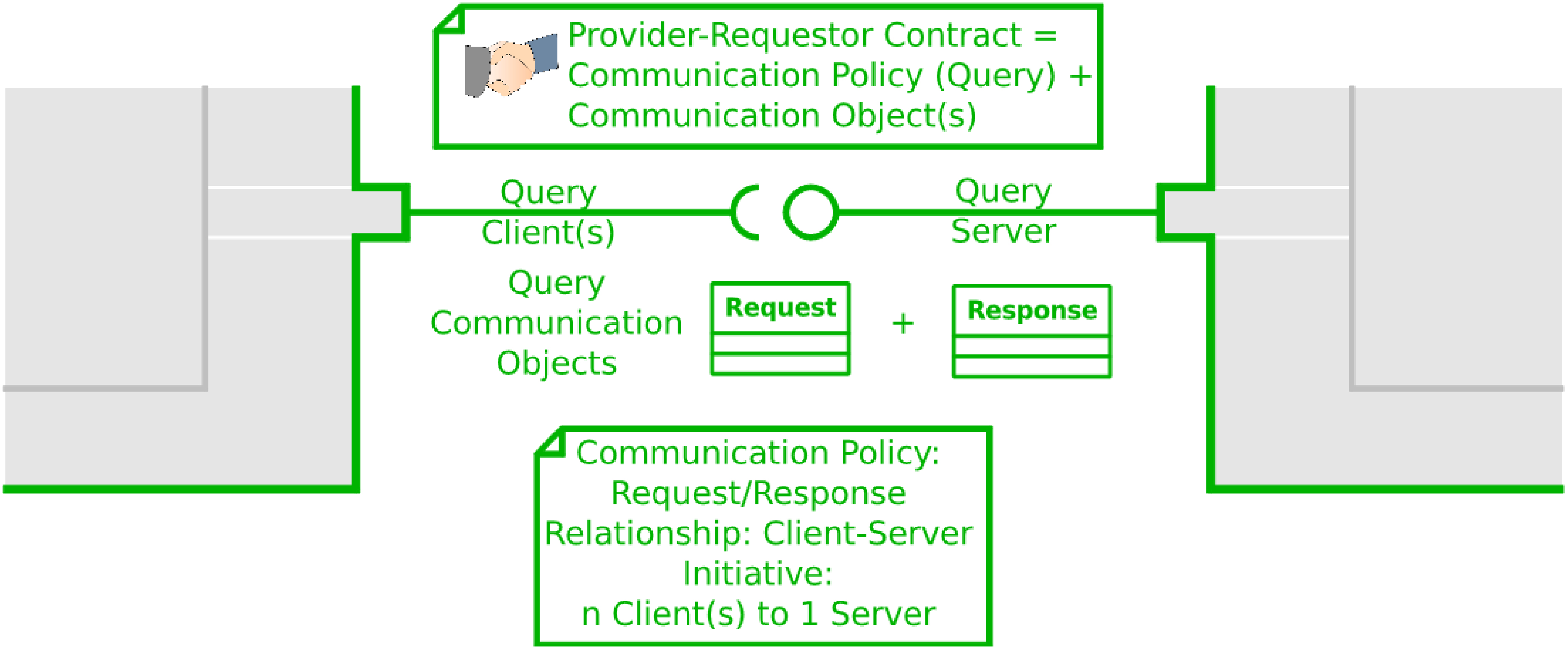
Example Query Pattern / Overview

Example Query Pattern Overview Different Views



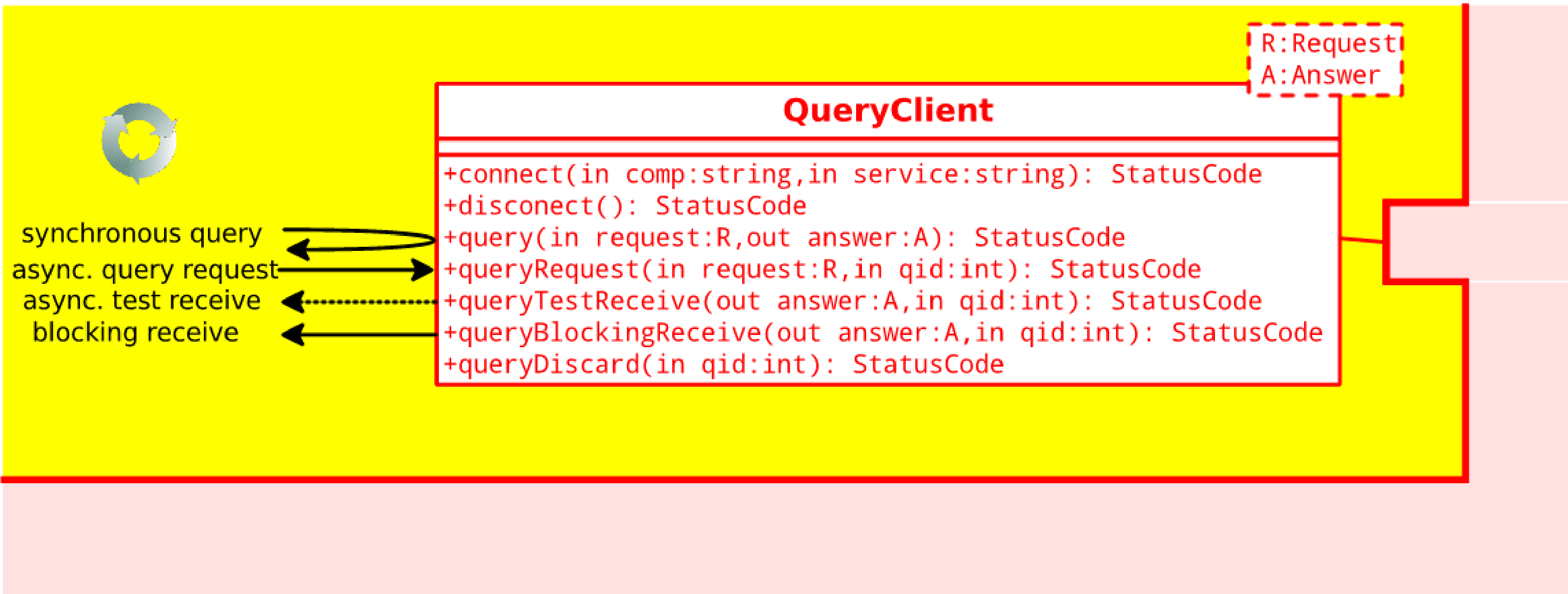
Example Query Pattern / External View

Component as Black-Box
System Integrator View



Example Query Client / Internal View

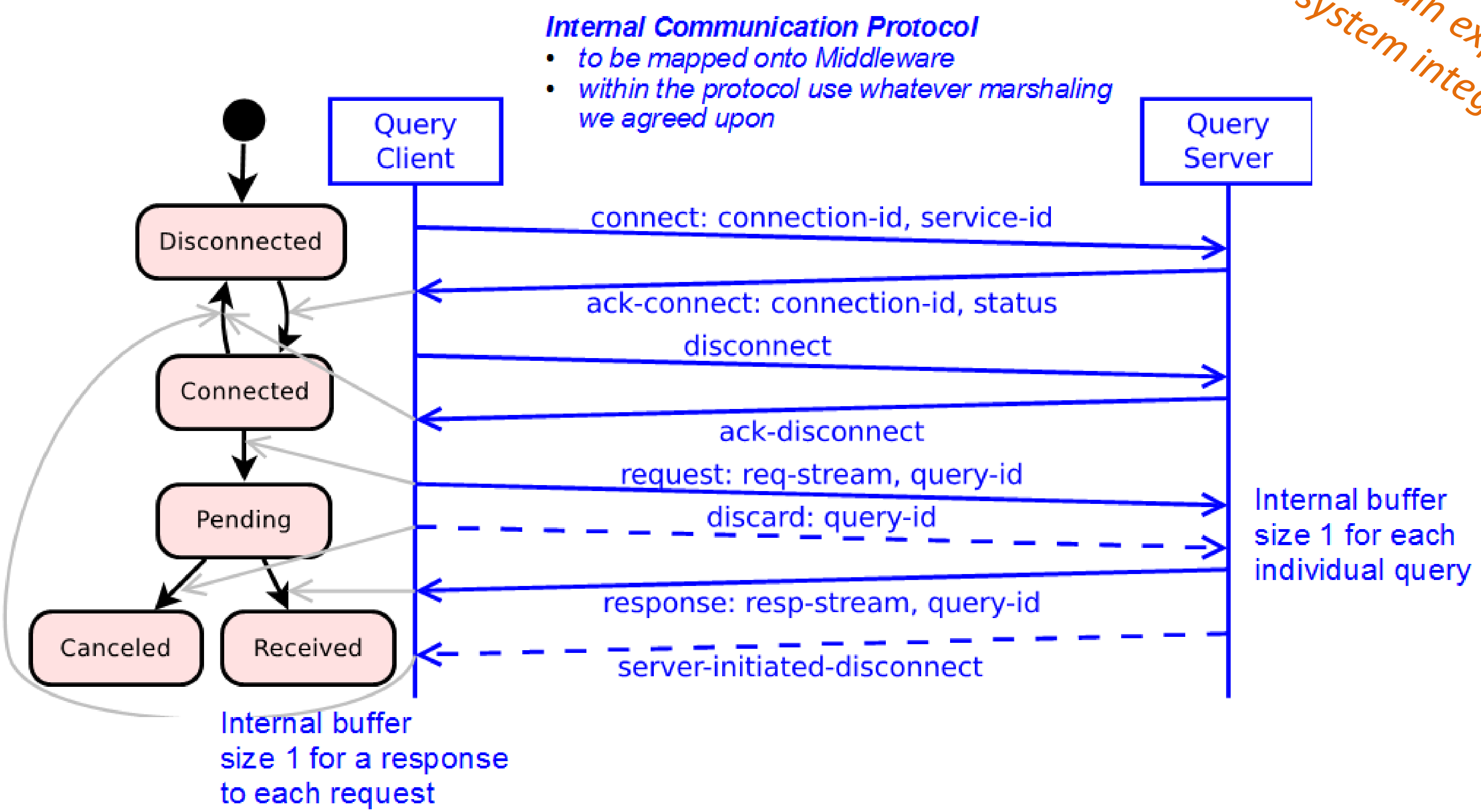
Component as White-Box
Component Developer View



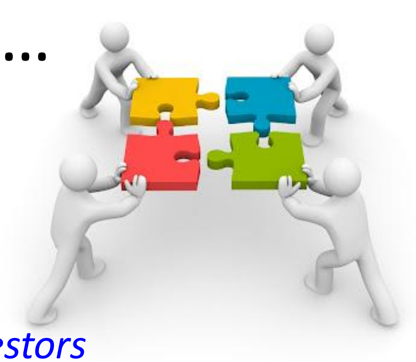
StatusCode	query	query Request	queryTest Receive	queryBlocking Receive	query Discard
OK	✓	✓	✓	✓	✓
disconnected	✓	✓	✓	✓	✗
canceled	✓	✗	✗	✓	✗
no-data	✗	✗	✓	✗	✗
wrong ID	✗	✗	✓	✓	✓
error	✓	✓	✓	✓	✓

Example Query Pattern: Mapping of Message Protocol onto Middleware

Framework Builder View
- software engineering expert
- not visible to
  - robotics expert
  - domain expert
  - system integrator



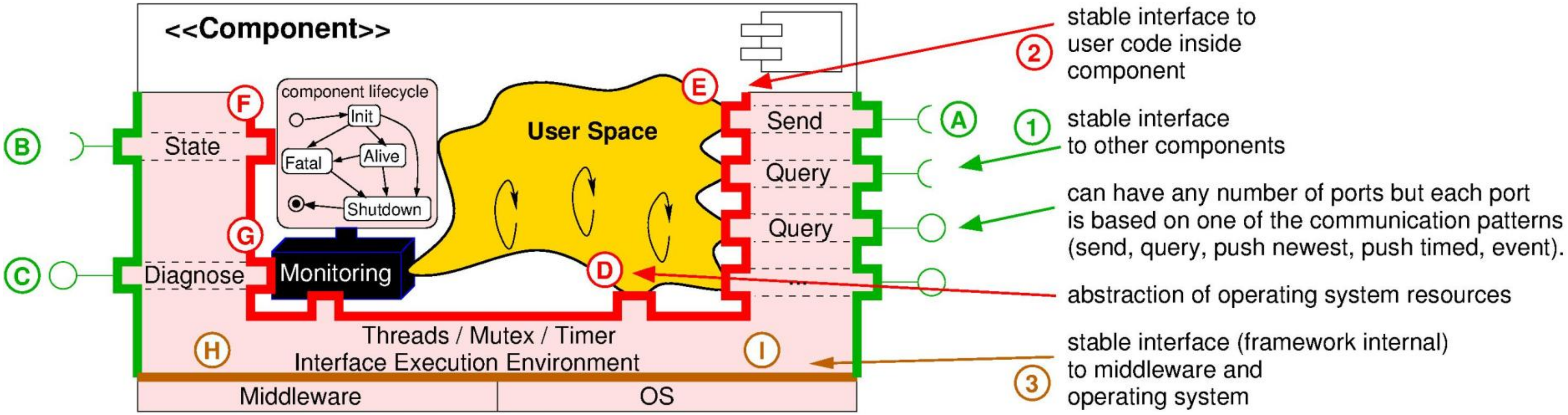
Policies and Strategies behind SmartSoft Services…
…towards QoS and Resource Awareness



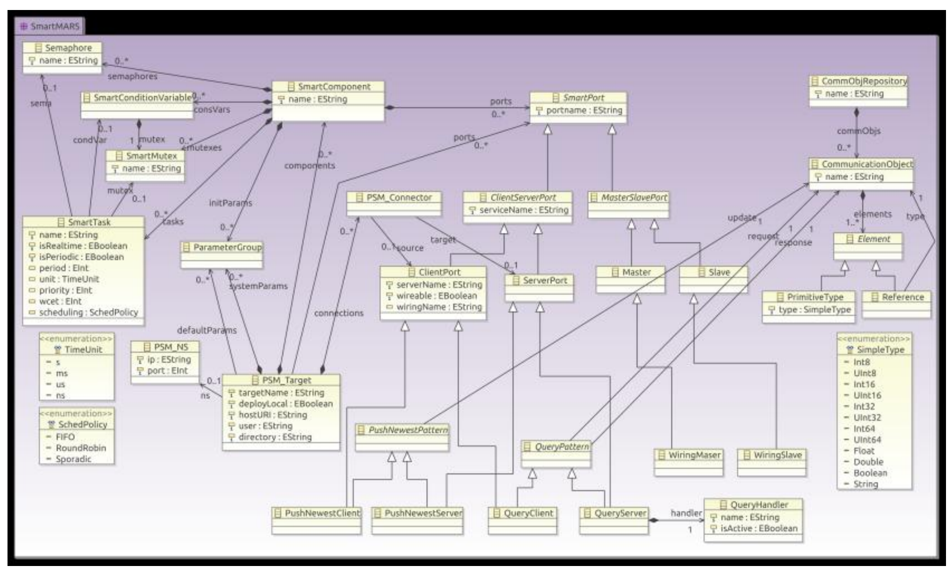
- ***clients of services are not allowed*** *to make any assumptions about offered services beyond the announced characteristics*
- ***service providers are not allowed*** *to make any assumptions about service requestors (like e.g. their maximum rate of requests)*

  - As long as there are no further quality-of-service attributes, the service provider accepts all incoming requests and guarantees to answer all accepted requests
  - However, only the service provider knows about its resources available to process incoming requests and clients are not allowed to impose constraints on the service provider (a request might provide further non-committal hints to the service provider like a request priority)
  - Thus, the service provider is allowed to provide a NIL answer in case it is running out of resources (beyond guaranteed QoS) to answer a particular request
  - In consequence, all service requestors must always be prepared to get a NIL answer when performing requests beyond acknowledged QoS
  - A service requestor is also not allowed to make any assumptions about the response time as long as according QoS attributes are not set by the service provider
  - However, if a service provider announces to answer requests within a certain time limit, one can rely on getting at least a NIL answer before the deadline. If a service provider announces to process requests within a certain time limit, one can rely on getting a complete answer before the deadline
  - If a service requestor depends on a maximum response time although this QoS attribute is not offered by the service provider, it needs to use client-side timeouts with ist request

Robotics Software Component Model + MDSD…



Meta-Model



Robotics Software Component Model + MDSD…

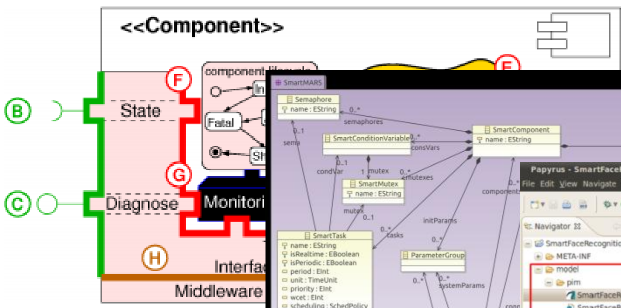
Screencast
Simple Navigation

SmartMDSD
(service oriented component model)
- Meta-Model
- Toolchain

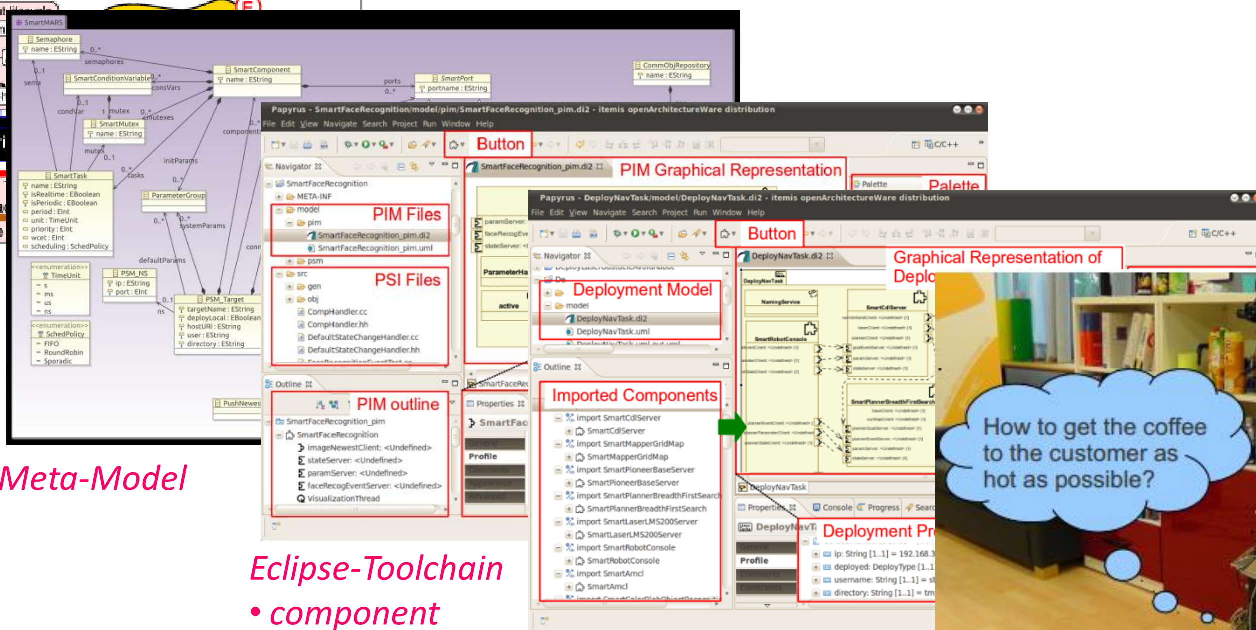
SmartSoft
(implementation)
- CORBA / SmartSoft
- ACE / SmartSoft

SmartTCL
(Task Coordination Language [DSL])

VML
(Variability Modeling Language [DSL])



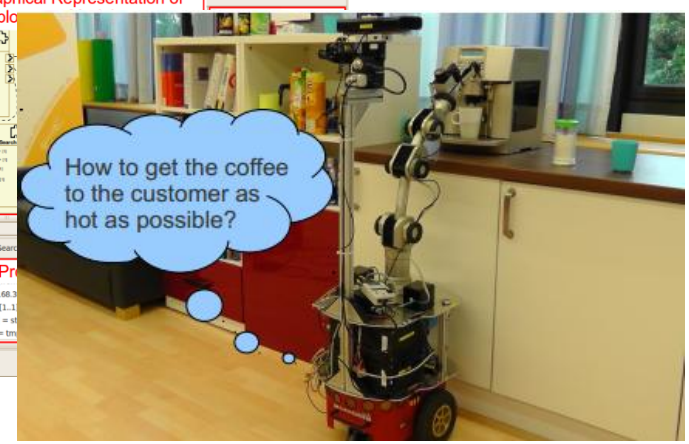
- *Software component*
- *execution container*



Meta-Model

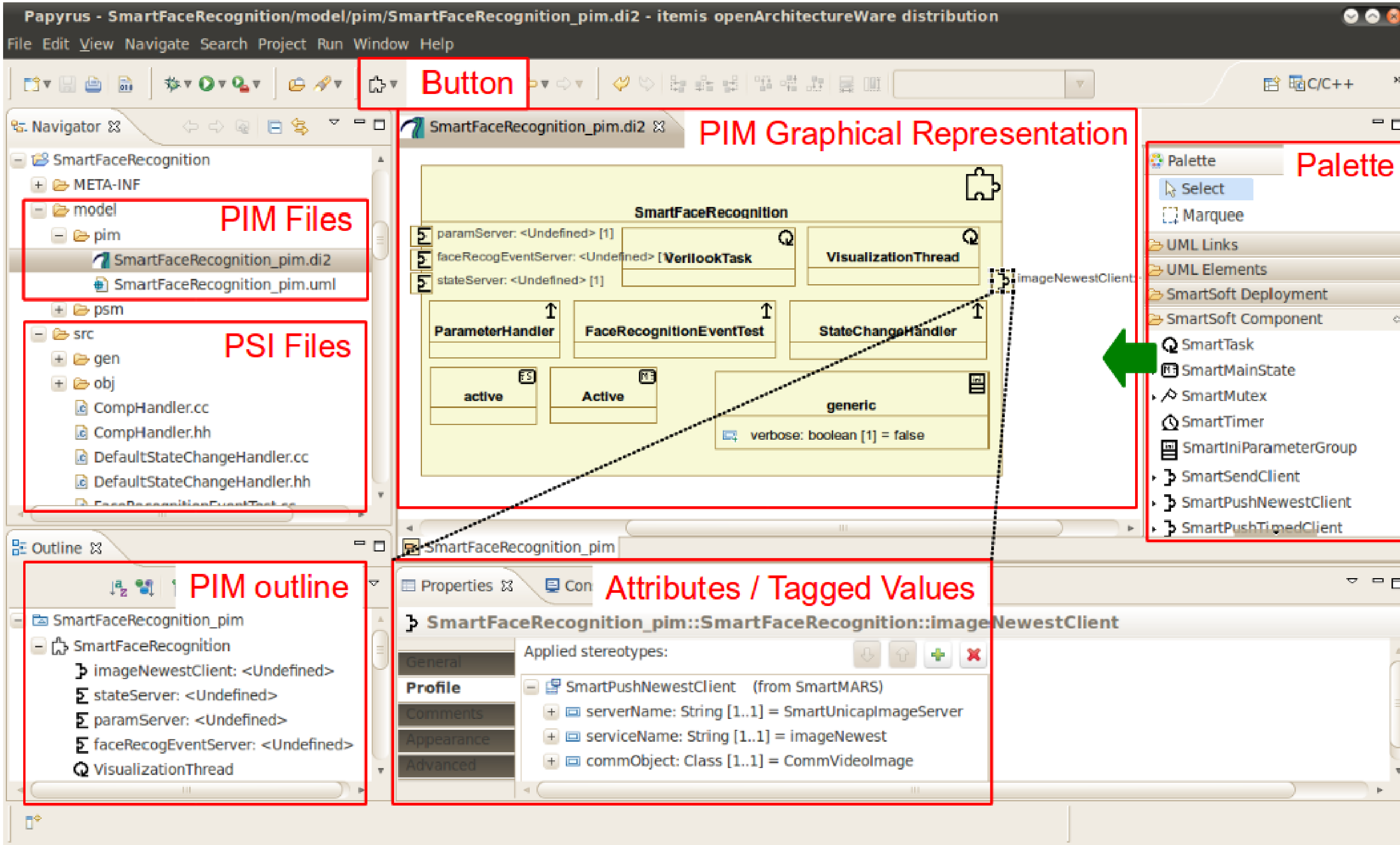
Eclipse-Toolchain
- *component developer*

Eclipse-Toolchain
- *system integrator*
- *deployment*

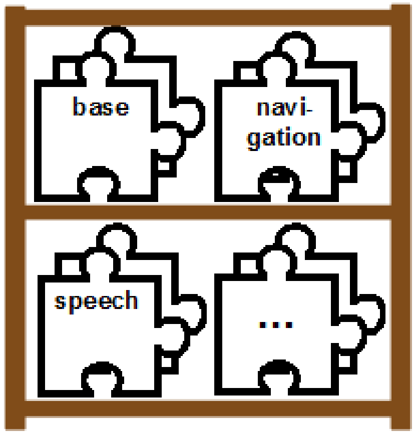


Real robot in real world

Component Builder View...

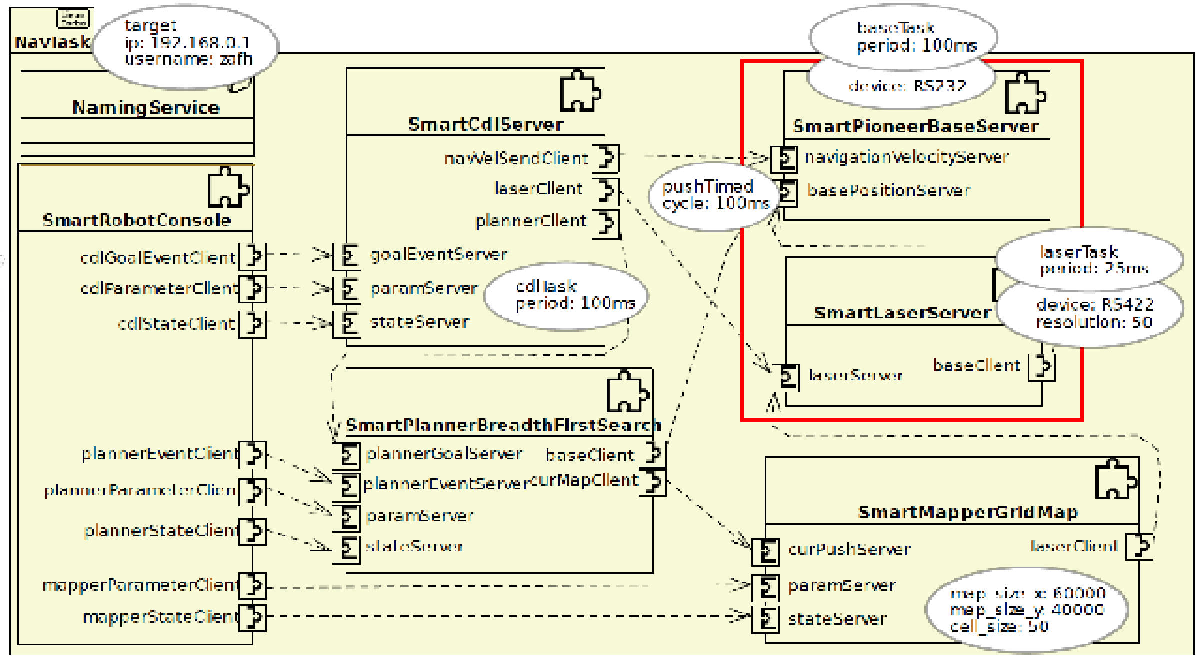


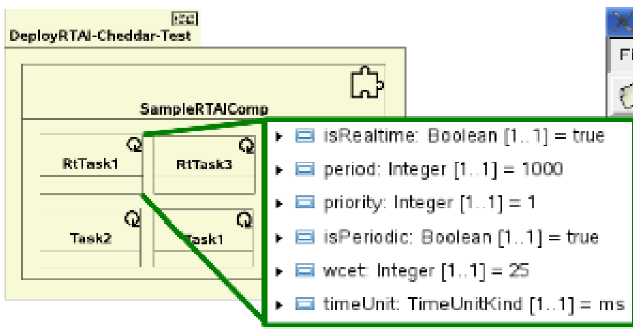
System Integrator View…

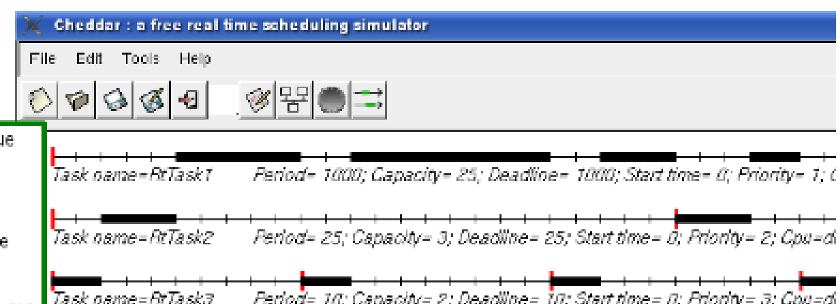


Component Shelf
Reusable Components

System Level Properties / Bindings / Conformance Checks

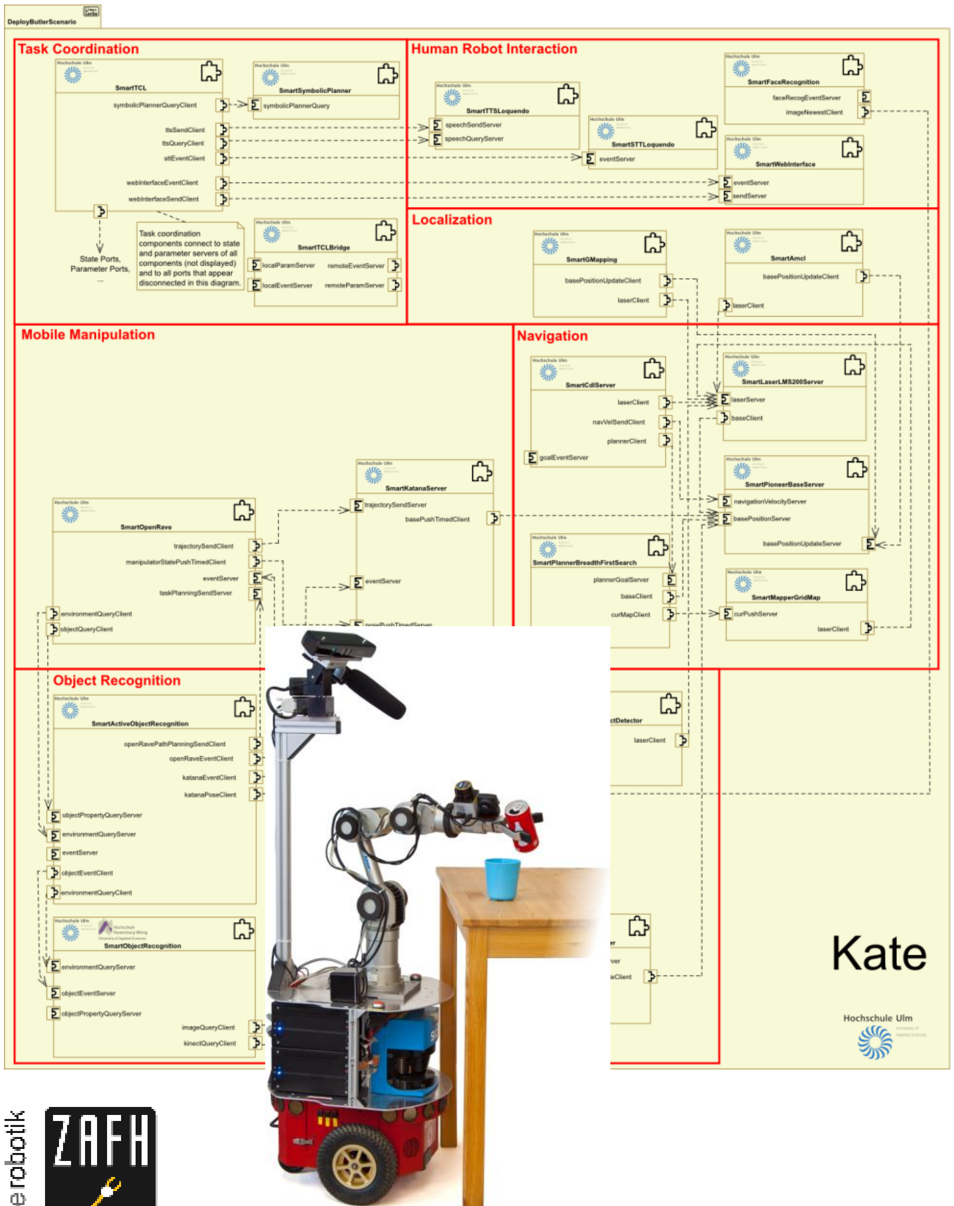


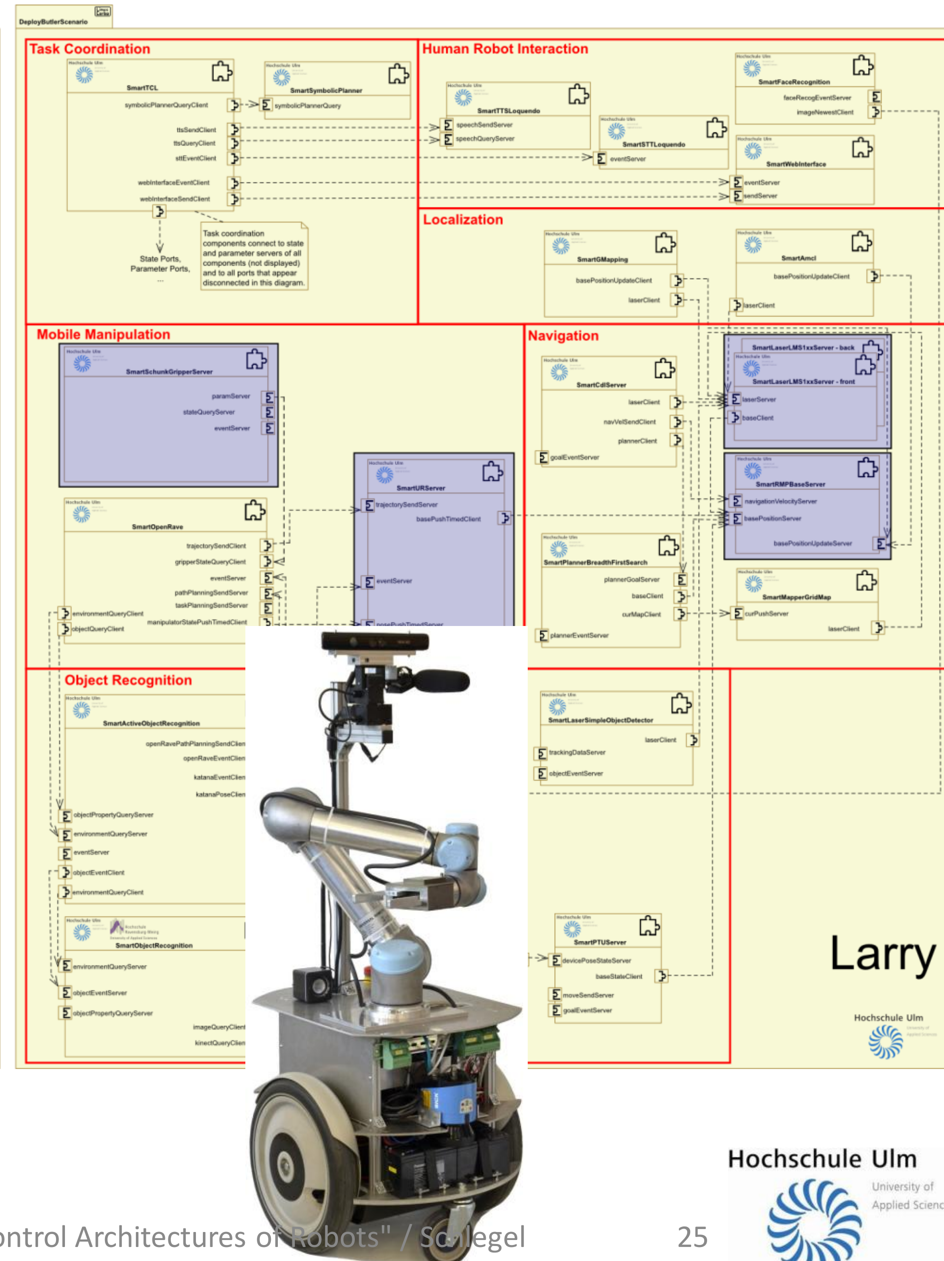




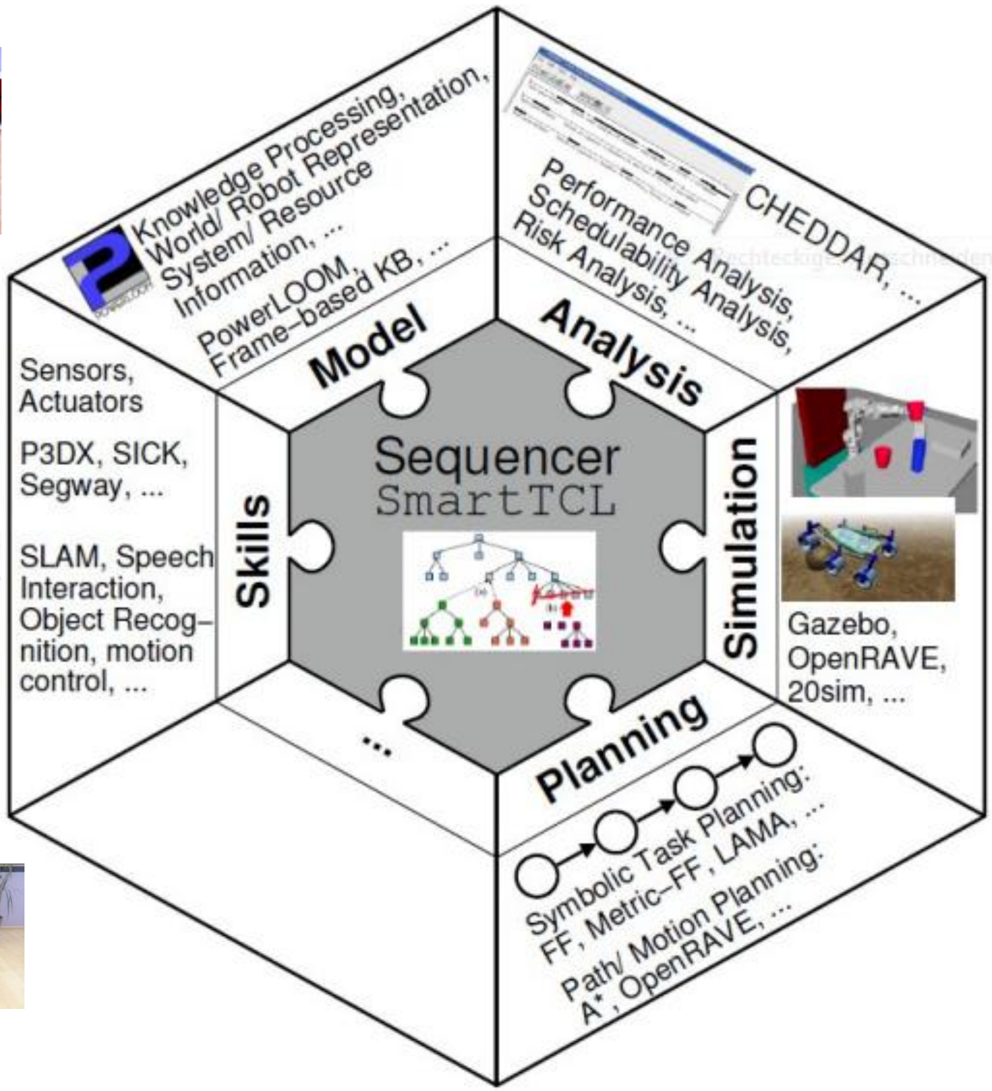

Intel Core2Duo P8800
4 GB RAM
SICK

Movie
Collaborative Butler Scenario





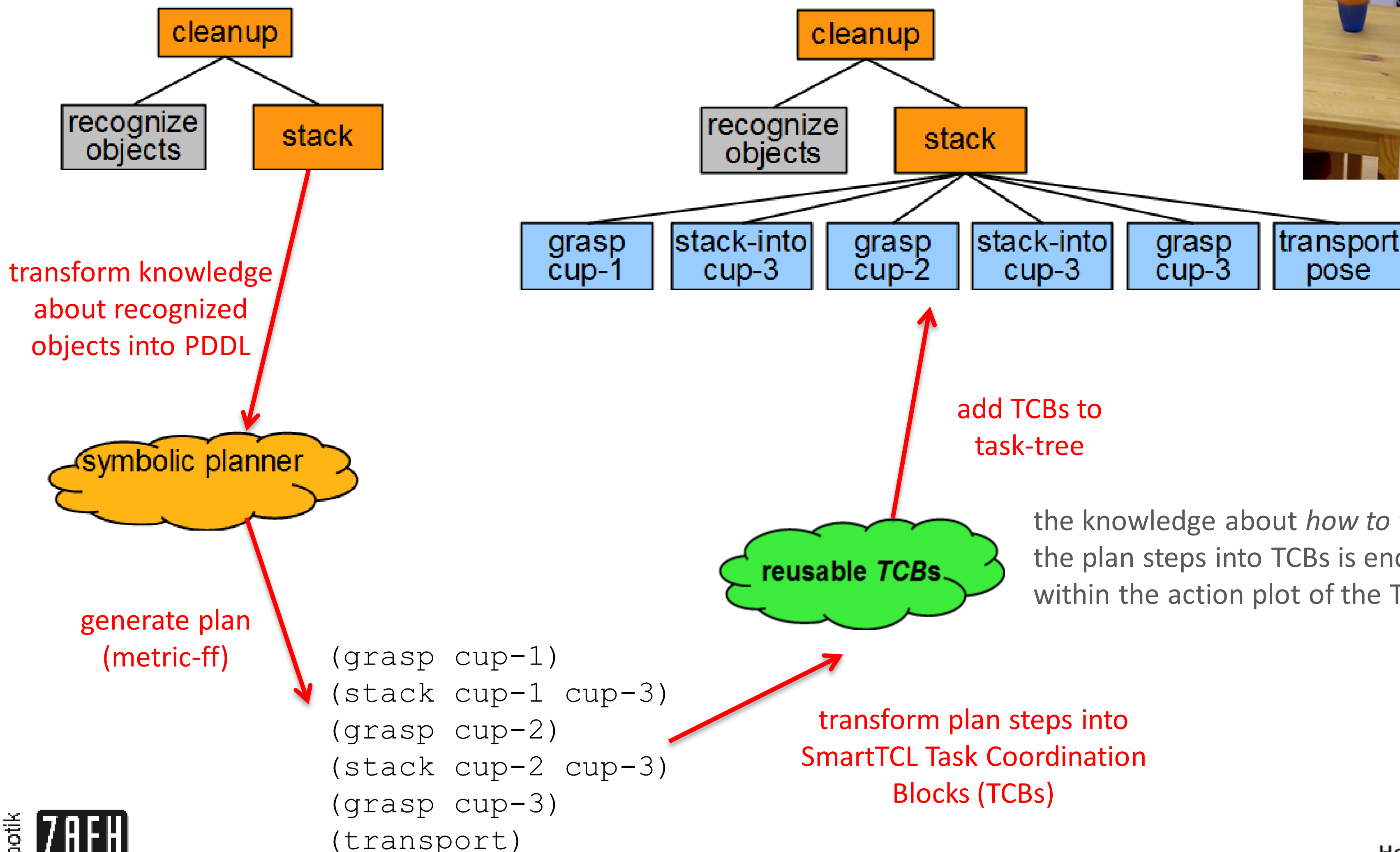
Robotic Architecture / Software Architecture



Sequencing Layer with SmartTCL:

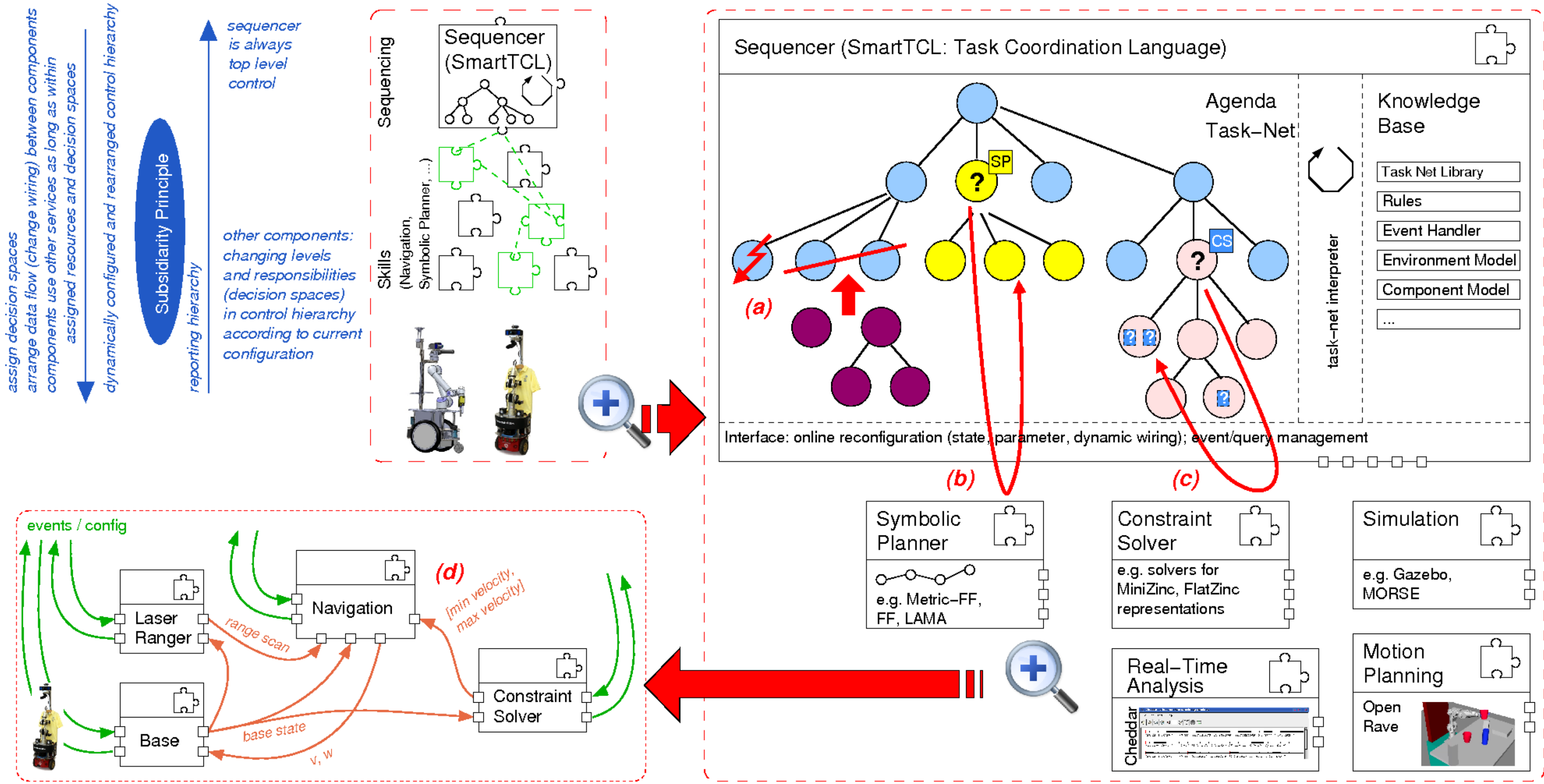
- bridges between continuous processing and event-driven task execution
- orchestrates the software components in the system
- assigns decision spaces to components
- involves dedicated experts for run-time binding of designed variability
- coordinate analysis, simulation and planning capabilities
  - send parameters and configurations
  - switch components on/off to manage resources
  - change the wiring between the components
  - query information and wait for events

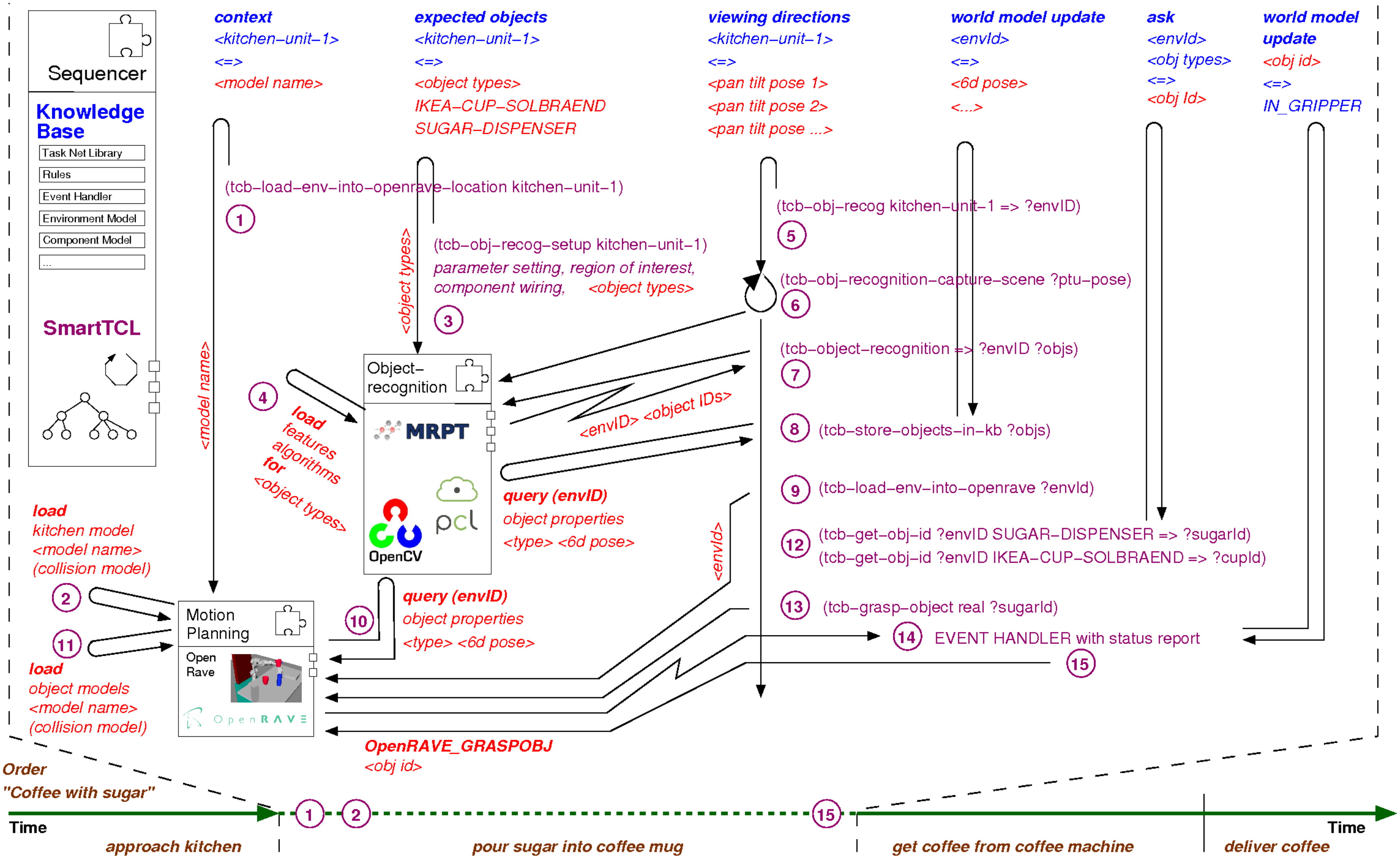
Robotic Architecture / Software Architecture



the knowledge about *how to transform* the plan steps into TCBs is encoded within the action plot of the TCB *stack*

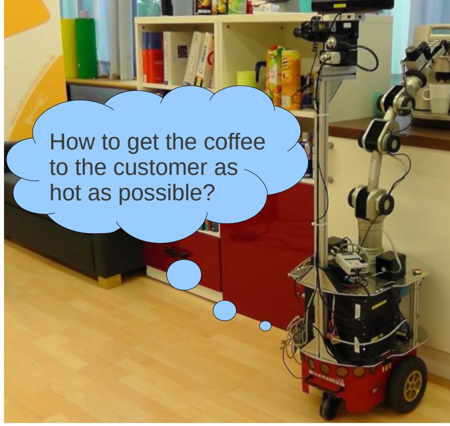
Robotic Architecture / Software Architecture





Sequencer
Knowledge Base
Task Net Library
Rules
Event Handler
Environment Model
Component Model
...
SmartTCL
context
<kitchen–unit–1>
<=>
<model name>
expected objects
<kitchen–unit–1>
<=>
<object types>
IKEA–CUP–SOLBRAEND
SUGAR–DISPENSER
viewing directions
<kitchen–unit–1>
<=>
<pan tilt pose 1>
<pan tilt pose 2>
<pan tilt pose ...>
world model update
<envId>
<=>
<6d pose>
<...>
ask
<envId>
<obj types>
<=>
<obj Id>
world model update
<obj id>
<=>
IN_GRIPPER
(tcb–load–env–into–openrave–location kitchen–unit–1)
1
(tcb–obj–recog–setup kitchen–unit–1)
parameter setting, region of interest,
component wiring, <object types>
3
<object types>
<model name>
(tcb–obj–recog kitchen–unit–1 => ?envID)
5
(tcb–obj–recognition–capture–scene ?ptu–pose)
6
(tcb–object–recognition => ?envID ?objs)
7
<envID> <object IDs>
8 (tcb–store–objects–in–kb ?objs)
9 (tcb–load–env–into–openrave ?envId)
12 (tcb–get–obj–id ?envID SUGAR–DISPENSER => ?sugarId)
(tcb–get–obj–id ?envID IKEA–CUP–SOLBRAEND => ?cupId)
13 (tcb–grasp–object real ?sugarId)
14 EVENT HANDLER with status report
15
Object–recognition
MRPT
pcl
OpenCV
4
load features algorithms for <object types>
query (envID)
object properties
<type> <6d pose>
<envId>
load
kitchen model
<model name>
(collision model)
2
11
load
object models
<model name>
(collision model)
Motion Planning
Open Rave
OpenRAVE
10
query (envID)
object properties
<type> <6d pose>
OpenRAVE_GRASPOBJ
<obj id>
Order
"Coffee with sugar"
Time
1
2
15
Time
approach kitchen
pour sugar into coffee mug
get coffee from coffee machine
deliver coffee

Modeling Variability / Run-Time Variability



The Approach

- Express variability at design-time
    - make it as simple as possible for the *designer* to *express variability*
- Bind variability at run-time based on the then available information
    - enable the *robot* to *bind variability* at *run-time* based on the then available information
- *remove complexity from the designer by a DSL*
- *remove complexity from the robot's run-time decision by modeling variability*

Separation of concerns:

- models (e.g. task net) describe ***how*** to deliver a coffee
- models specify ***what is a good way (policy)*** of delivering a coffee
  (e.g. in terms of non-functional properties like safety, energy consumption, etc.)

Separation of roles:

- designer at design-time: provides models
    - action plots with variation points to be bound later by the robot
    - policies for task fulfillment
    - problem solvers to use for binding variability
- robot at run-time: decides on proper bindings for variation points
    - apply policies
    - take into account current situation and context

Modeling Variability / Run-Time Variability

Objective: ***Optimize service quality of a system (non-functional properties):***
power consumption, performance, etc.
balance conflicting properties by minimizing overall cost function
(constrained optimization problem)

- property importance varies according to the current context → ***property priority***
- properties are expressed as functions of variation points → ***property definition***

Inputs
(context variables)

- the current robot state (task and resources)
- the environment situation

VML:
Context definitions,
variation point definitions,
Properties and rules

Outputs
(***variation point*** bindings)

- binding system variability (non conflicting with functionality)

adaptation rules:

- define direct relationships between context variables and variation points
- event-condition-action rules
- directly constrain the possible values of variation points according to current context

Modeling Variability / Run-Time Variability


```
/* Data type definitions */
number percentType { range: [0, 100];    precision: 1; }
number velocityType { range: [100 600]; precision: 10; unit: "mm/s"; }

/* Contexts */
context ctx_battery : percentType;
context ctx_noise : percentType;

/* Adaptation rules */
rule low_noise : ctx_noise < 20 => speakerVolume = 35;
rule medium_noise : ctx_noise >= 20 & ctx_noise < 70 => speakerVolume = 55;
rule high_noise : ctx_noise >= 70 => speakerVolume = 85;

/* Properties */
property efficiency : percentType maximized {
        priorities: f(batteryCtx) = max(exp(-batteryCtx/15)) - exp(-batteryCtx/15);
        definitions: f(maxVelocity) = maxVelocity; }

property powerConsumption : percentType minimized {
        priorities: f(batteryCtx) = exp(-1 * batteryCtx/15);
        definitions: f(maxVelocity) = exp(maxVelocity/150); }

/* Variation points */
varpoint maximumVelocity : velocityType;
varpoint speakerVolume   : percentType;
```

Context variables

Adaptation rules

Properties

Variation points

Modeling Variability / Execution Semantics

- M2M transformation from *VML model* into *MiniZinc model*
  - *MiniZinc is currently supported by many constraint solvers*
    - *context variables => parameters*
    - *variation points => decision variables*
    - *adaptation rules / variation point dependencies => constraints*
    - *properties => cost function*
  - *we use*
    - **The G12 Constraint Programming Platform — University of Melbourne**

$$f(ctx, vp) = \sum_{\forall i} (-1)^{d_i} \cdot w_i(ctx) \cdot p_i(vp)$$

minimize
maximize

normalized priority function

normalized definition function

PPP Topic Group Proposal

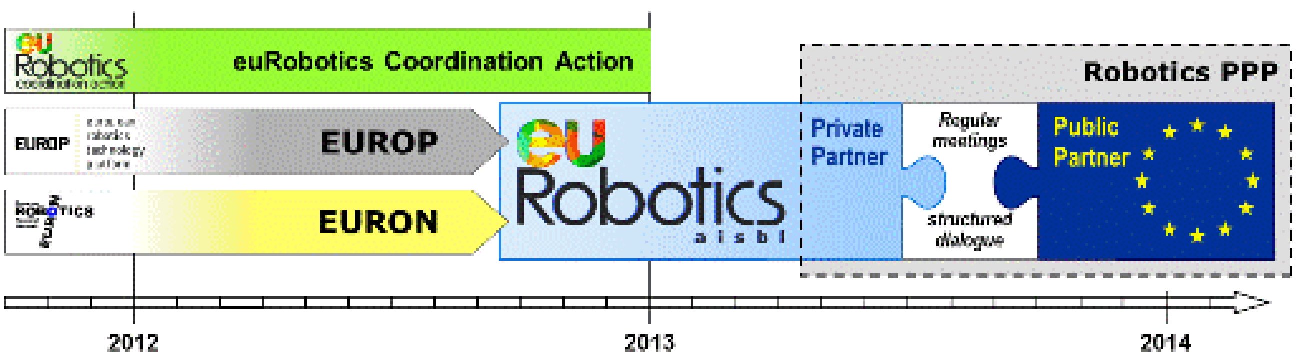
Proposal for setting up a Topic Group within the European Robotics Public Private Partnership on **„Software Systems Engineering in Robotics“**

Path from Research to Development to Innovation

- Achieve and provide structures in robotics software development and software systems integration and software tools along a business ecosystem (with support for separation of roles)
- Lower the barrier for participating in a robotics software business ecosystem by disseminating model-driven software development tools beyond the robotics experts to application domain experts, system integrators, end-users etc.
- We expect the very same positive effects of a business ecosystem for the software challenge in robotics as has been seen already in business ecosystems of other high-tech domains.

Milestones

- **Milestone 1: Driven by current academic group**
  - establish software systems engineering in robotics as a first-class research discipline
- **Milestone 2: Towards Economic Sustainability and Persistence**



Challenges / Current Work in Progress
(unstructured hints)

- **come up with Meta-Models, Models as Robotics Community Activity**
  - explicate robotics body of knowledge independent from implementational technologies
  - collaborate at the level of meta-models ⇔ compete at the level of implementations
- **Model-Driven Software Development**
  - workflow to support *separation of roles* and *separation of concerns*
  - ***from*** a standard linear workflow ***towards*** a stepwise-refinement approach
    - ***not just*** linear PIM => PSM => PSI ***but*** Component Developer => System Integrator => Robot
      - each role binds variability at PIM, PSM, PSI level
        - e.g. early partial binding of H/W (closed source library, sensor mounting)
        - e.g. late binding of underlying execution platform (OS, middleware like ACE, DDS) as object libraries (see *generation gap pattern* used in *SmartSoft* templates)
- **Black box handover from one role to the next**
  - variability modeling
    - transformation from design-time model to run-time exploitable model
  - resource modeling and QoS modeling
    - QoS attfibutes at the communication patterns
    - Relationships between QoS settings (twice maximum speed ⇔ three times processing power)
  - deployment
    - mapping / matching S/W resource requirements with H/W platform model
- **Link between S/W model (component settings, resources) and robot behavioral model (Task Nets)**
  - generic task nets ⇔ skills with component configurations ⇔ S/W component model

Addendum / Some Links

SmartMDSD / SmartSoft

- http://smart-robotics.sf.net/
- http://smart-robotics.sourceforge.net/mdsdSmartSoft/index.php
- http://smart-robotics.sourceforge.net/corbaSmartSoft/index.php
- http://smart-robotics.sourceforge.net/aceSmartSoft/index.php

- http://www.youtube.com/roboticsathsulm
- http://www.zafh-servicerobotik.de/ULM/publikationen.php

- http://www.intechopen.com/articles/show/title/robotic-software-systems-from-code-driven-to-model-driven-software-development
- http://www.iconceptpress.com/download/paper/101215045543.pdf