Model Driven Software Development in Service Robotics – *It really works !*

Prof. Dr. Christian Schlegel

Computer Science Department
University of Applied Sciences Ulm

http://www.zafh-servicerobotik.de/ULM/index.php
http://www.hs-ulm.de/schlegel
http://smart-robotics.sourceforge.net/

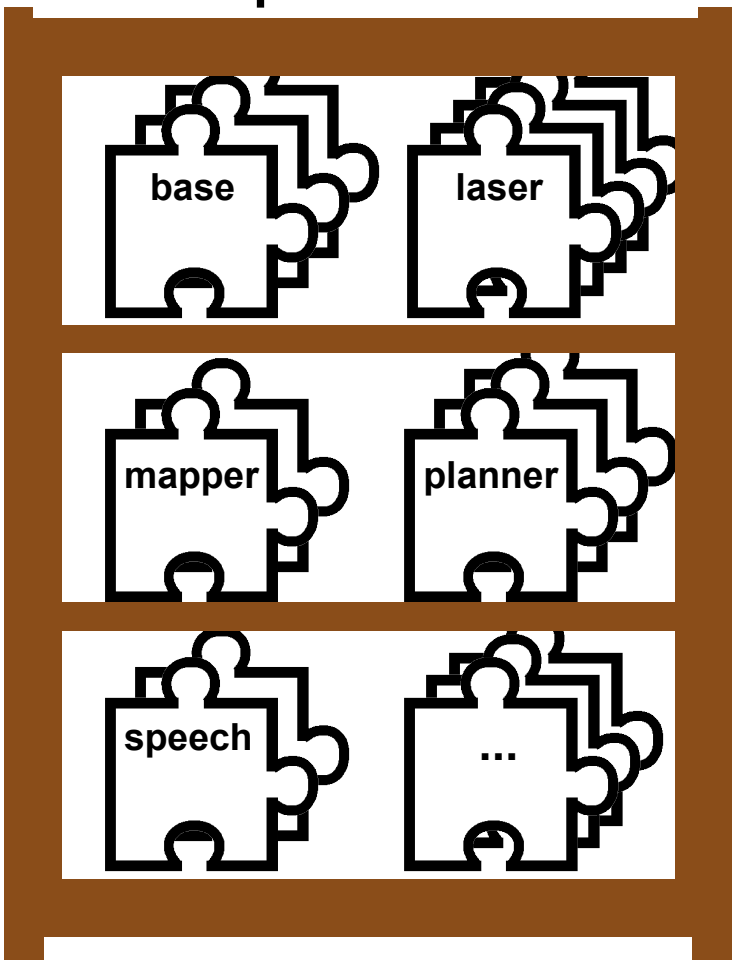
AG Schlegel: M.Sc. Siegfried Hochdorfer, B.Sc. Alex Lotz, B.Sc. Matthias Lutz, B.Sc. Dennis Stampfer, B.Sc. Andreas Steck, B.Sc. Jonas Brich, B.Sc. Manuel Wopfner

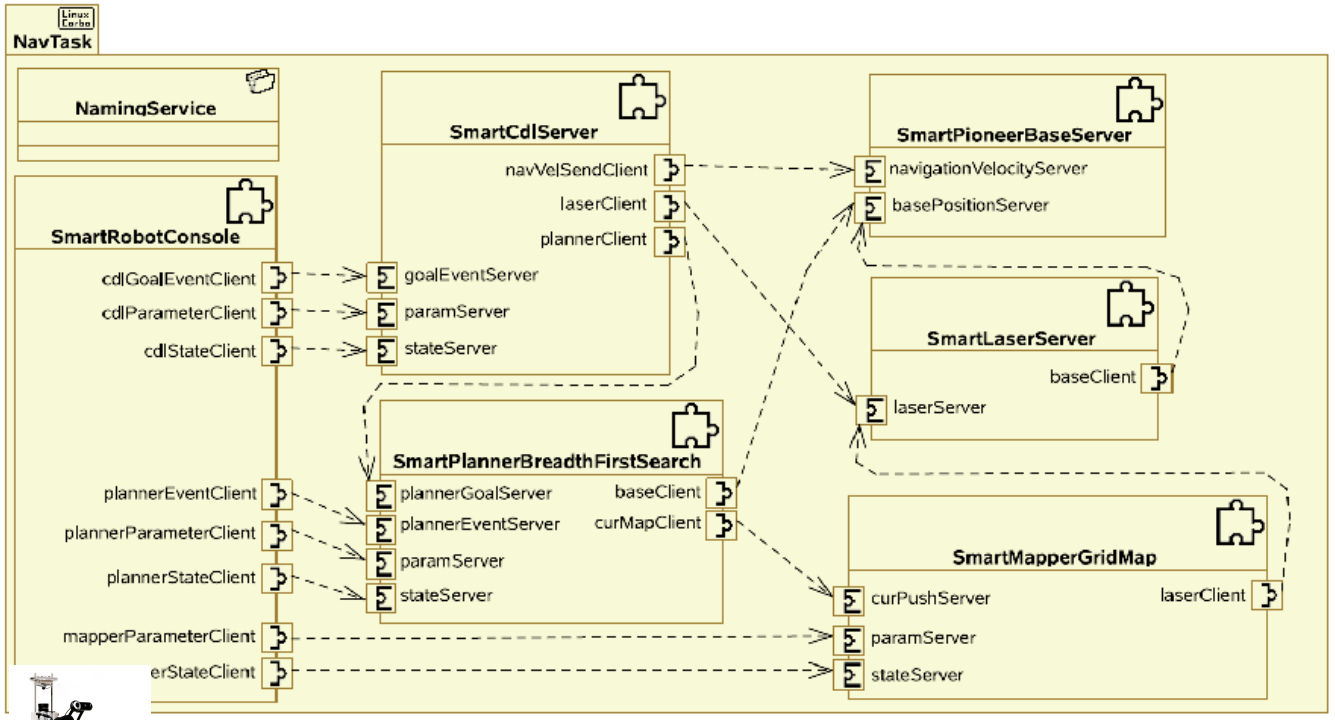
Hochschule Ulm

service robotik

Model Driven Software Development
Example: Navigation Task

Component Shelf

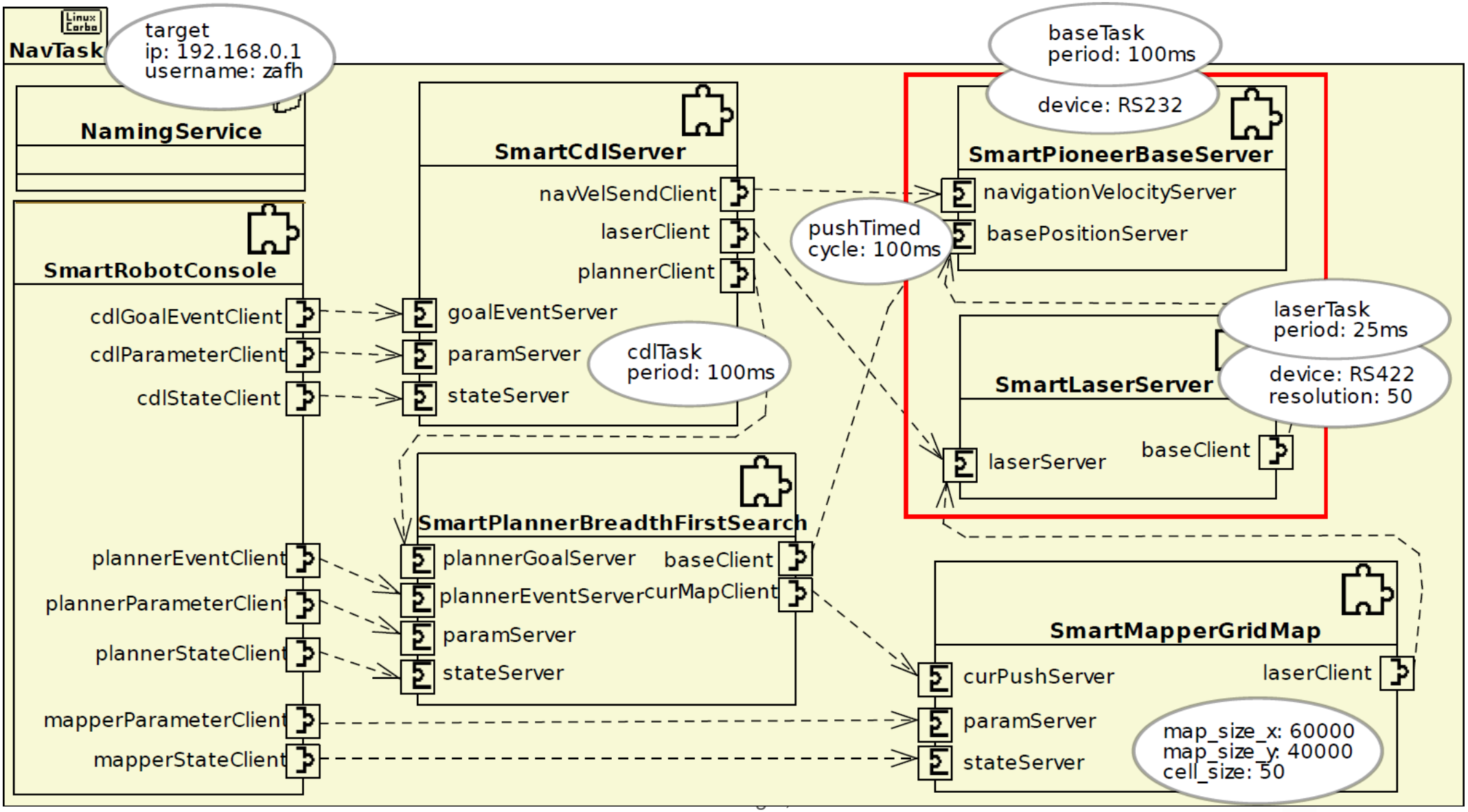




Hochschule Ulm

service robotik

Model Driven Software Development
Example: Navigation Task



Approach: Design Abstraction

We need a systematic engineering approach for robotics software!

- robots are complex systems that depend on systematic engineering
- so far fundamental properties of robotic systems have not been made detailed enough nor explicit (e.g. QoS)
- tremendous code-bases (libraries, middleware, etc.) coexist without any chance of interoperability and each tool has attributes that favors its use

➔ ***rely, as for every engineering endeavour, on the power of models***

➔ ***nowadays, robotics functionality is foremost based on software***

➔ ***make the step towards MDSD***





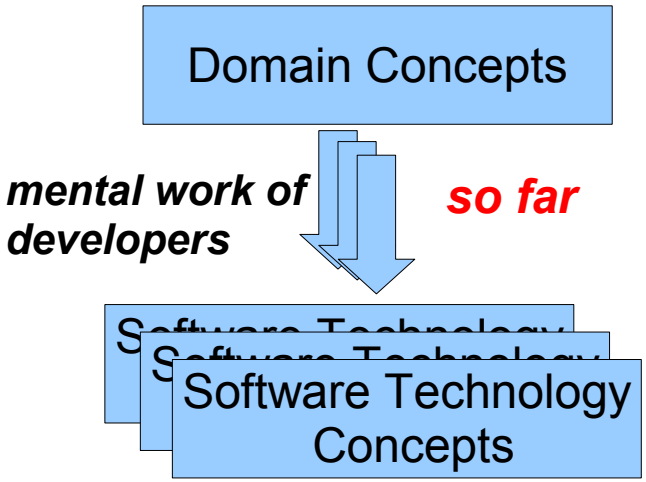


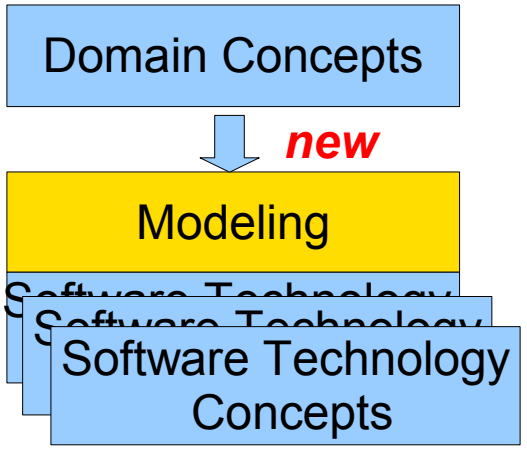
etc.

Hochschule Ulm

Approach: 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***

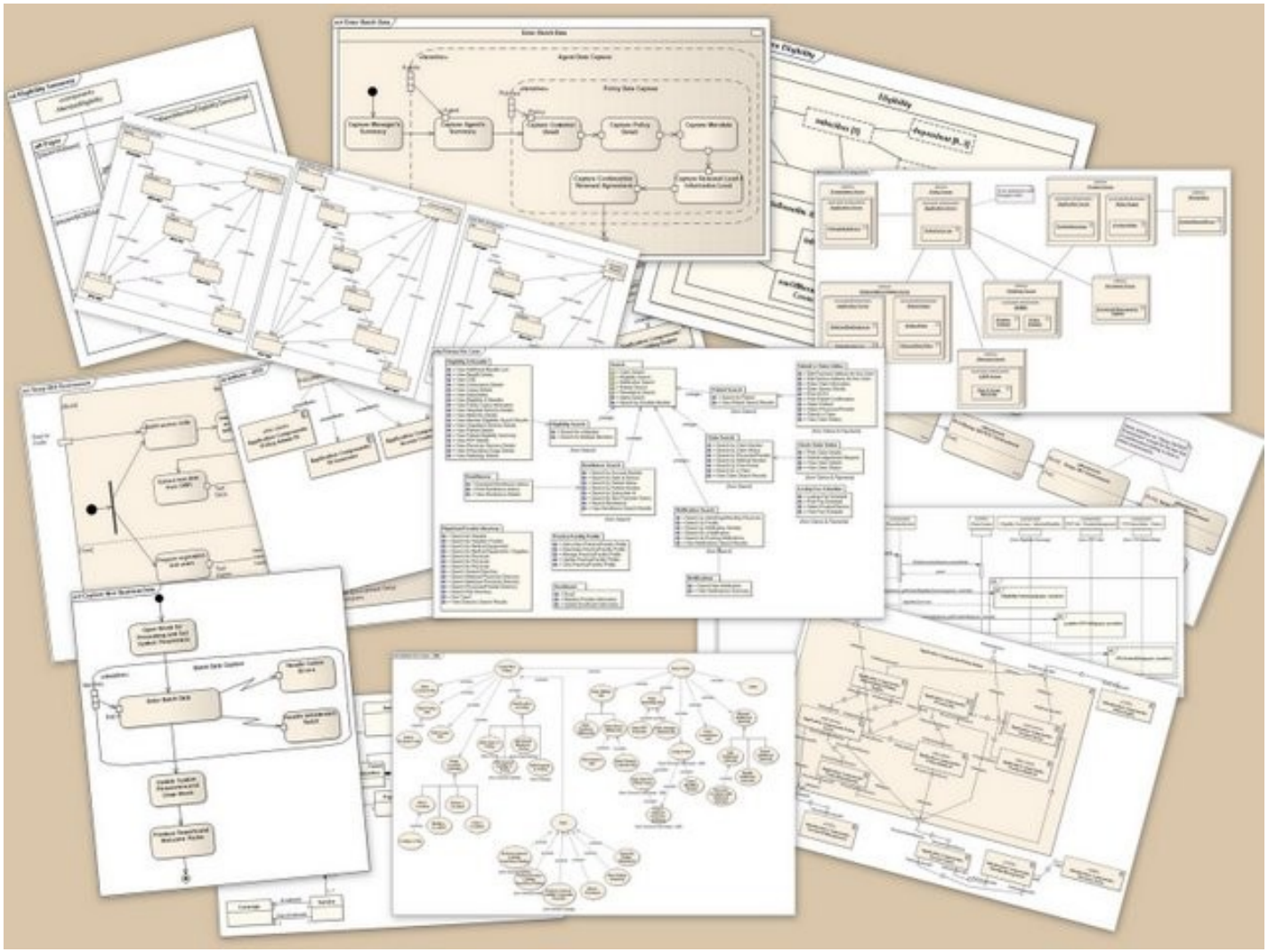




service robotik

ZAFH

What are Models / Modeling Languages good for ?



- UML notations: unified ?

- *Semantics* matters (more than syntax)
- *Useful* semantics imply *constraints* on designers
- *Heterogeneity* may be better than *generality*
- Modeling languages that are not executable, or where the execution semantics is vague or undefined provide only low benefit

Wikipedia Commons / Kishorekumar 62

Example: Component-based Development Process

- we focus on concurrent components that communicate via ports
  - as one might describe in SysML, AADL, or UML Component Diagrams and Communication Diagramms

- consider SysML flow ports
  - seems like message passing
  - but what kind of message passing?
  - to make this *executable*, we need some decisions
    - » nonblocking writes could mean unbounded memory for buffers
    - » is unbounded memory part of the meaning of this model?
    - » such questions matter!
  - caution: even the professionals can mess up semantics

- Dimensions of variability
  - identify / support / ...
  - achieve architectural level conformance / integration etc.

[Edward A. Lee, Berkeley]

Example: Component-based Development Process

- in case we leave too much variability and decisions without explications and without clear semantics to the developer:
  - „Portability“ then apparently means that it runs on different platforms, not that it has the same behavior on different platforms

- semantics
  - the semantics of a modeling language is the foundation for the models
  - *weak foundations result in less useful models*
  - unless we commit to a semantics, the model means different things to different observers!

[Edward A. Lee, Berkeley]

- does „more general“ always mean „better“?
  - obsessive flexibility can lead to languages like Jisp++

```
List myList
     = new List((cons (cdr foo->bar)).elements());
```

  - usable design practice implies “freedom *from* choice“
    [Alberto Sangiovanni-Vincentelli, on platform-based design]

Model Driven Software Development in Robotics

- useful modeling languages with strong semantics
  - useful executable modeling languages impose *constraints* on the designer
  - the constraints may come with benefits
  - we have to stop thinking of constraints as a universal negative!!
  - freedom *from* choice!!!

[Edward A. Lee, Berkeley]

- what is different in robotics compared to other domains like e.g. automotive, avionics, embedded systems, complex software systems, other engineering disciplines?
  - hypercomplex systems
  - deviation between design-time and run-time optimality

- what is a suitable starting point for model-driven software development in robotics?
  - component-based design
  - service-oriented architecture
  - → ***master component-hull by a model-driven approach to ensure architectural level conformance and interoperability to cope with system complexity***

Model Driven Software Development
Idea and Approach

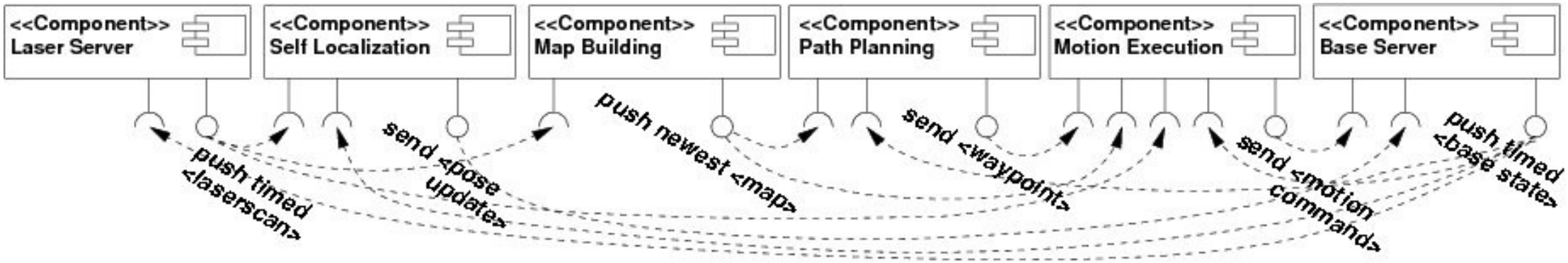
Precisely Defined Services:
services are defined by an interaction pattern and communication objects

The SmartSoft Interaction 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
state	activate/deactivate component
wiring	dynamic component wiring

These patterns are sufficient since they offer request/response interaction as well as asynchronous notifications and push services.

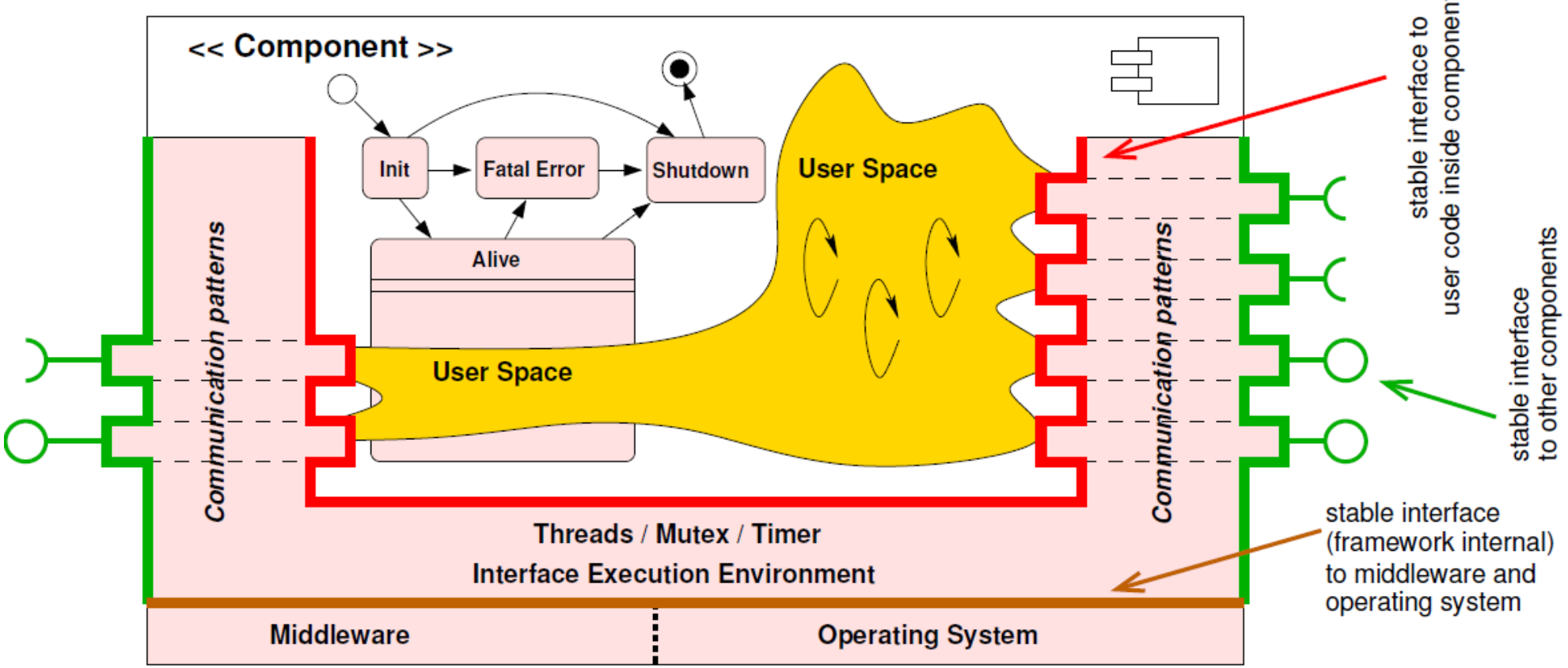
These patterns and their semantics are independent of any kind of middleware but ensure a certain type of architectural granularity



service robotik

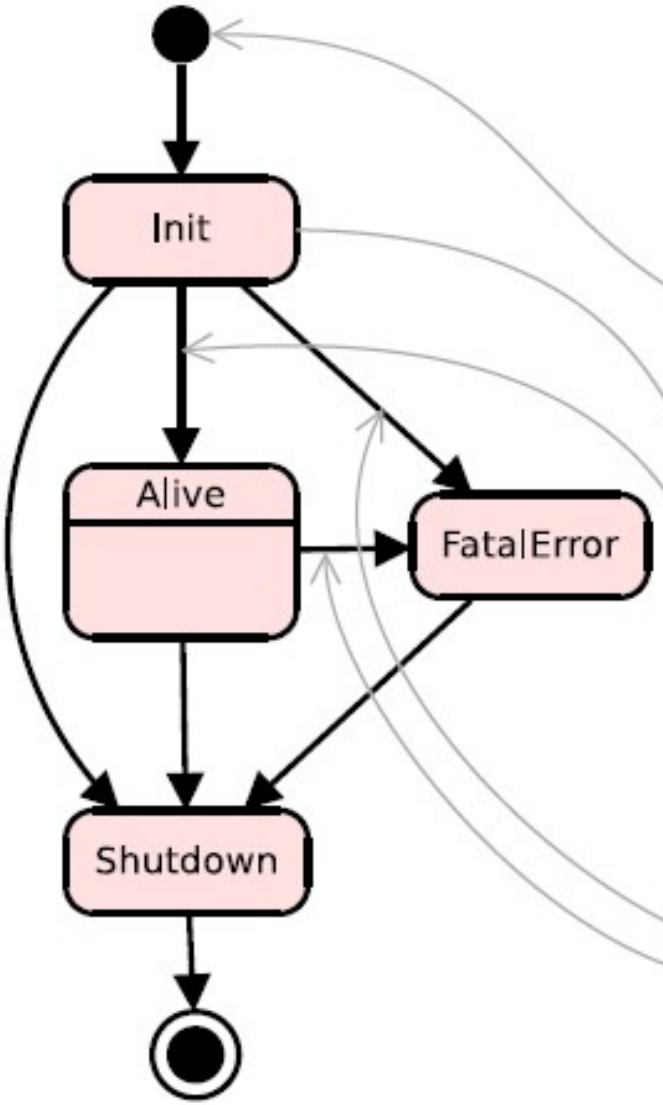
Model Driven Software Development
Idea and Approach

- service-oriented component approach with loose couplings



service robotik

Model Driven Software Development
Idea and Approach

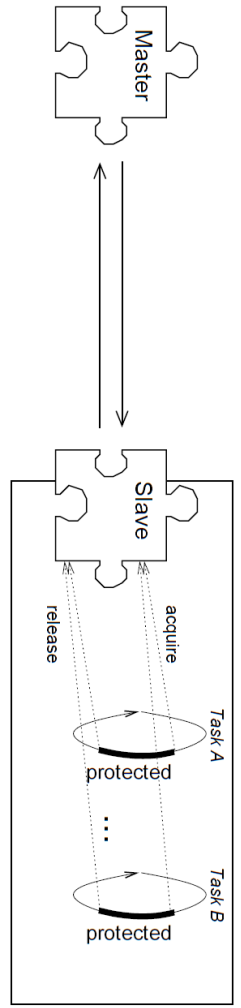



```
// include the SmartSoft framework and Communication-Object(s)
#include <smartSoft.hh>
#include <commExampleTime.hh>

CHS::PushNewestClient<CHS::CommExampleTime> *time_client;

class MyTask: public CHS::SmartTask
{
// ...
};


int main(int argc, char *argv[])
{

  try {
    // create and initialize SmartSoft-Component-Hull
    CHS::SmartComponent comp("ExampleComponent", argc, argv);

    // create and initialize component's CommunicationPort(s)
    time_client = new CHS::PushNewestClient<CHS::CommExampleTime>(&comp);

    // create Task(s) for user defined executions context(s)
    MyTask task;

    // try to (re)connect all ServiceRequestor(client) - ports (static configuration)
    std::string server = "SomeServer";
    std::string service = "SomeService";
    while(time_client->connect(server, service) != CHS::SMART_OK)
    {
      std::cout << "Connection to server: " << server << "; service: " <<
           service << "; FAILED! sleep(1) and trying reconnect..." <<
           std::endl;
      ACE_OS::sleep(1);
    }

    // start task(s) to execute
    task.open();

    // start component management (with Reactor-Event-Loop)
    comp.setState("Alive");
    comp.run();
  } catch (std::exception &e) {
    std::cerr << e.what() << std::endl;
    return 1;
  }catch(...) {
    std::cerr << "Uncought exception..." << std::endl;
    comp.setState("FatalError");
    return -1;
  }

  return 0;
}
```

Initialize

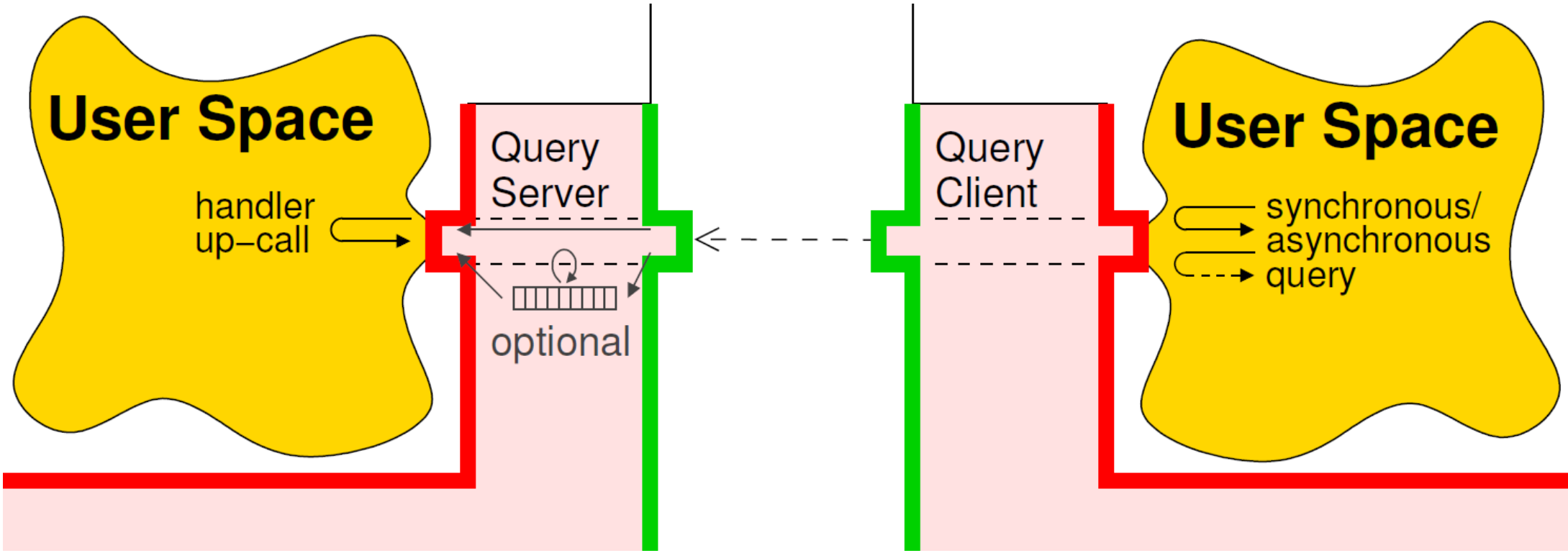
Error

state automaton interacts with communication patterns



Hochschule Ulm

Model Driven Software Development
Idea and Approach



Model Driven Software Development / Insertion / Technical Details /

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]*

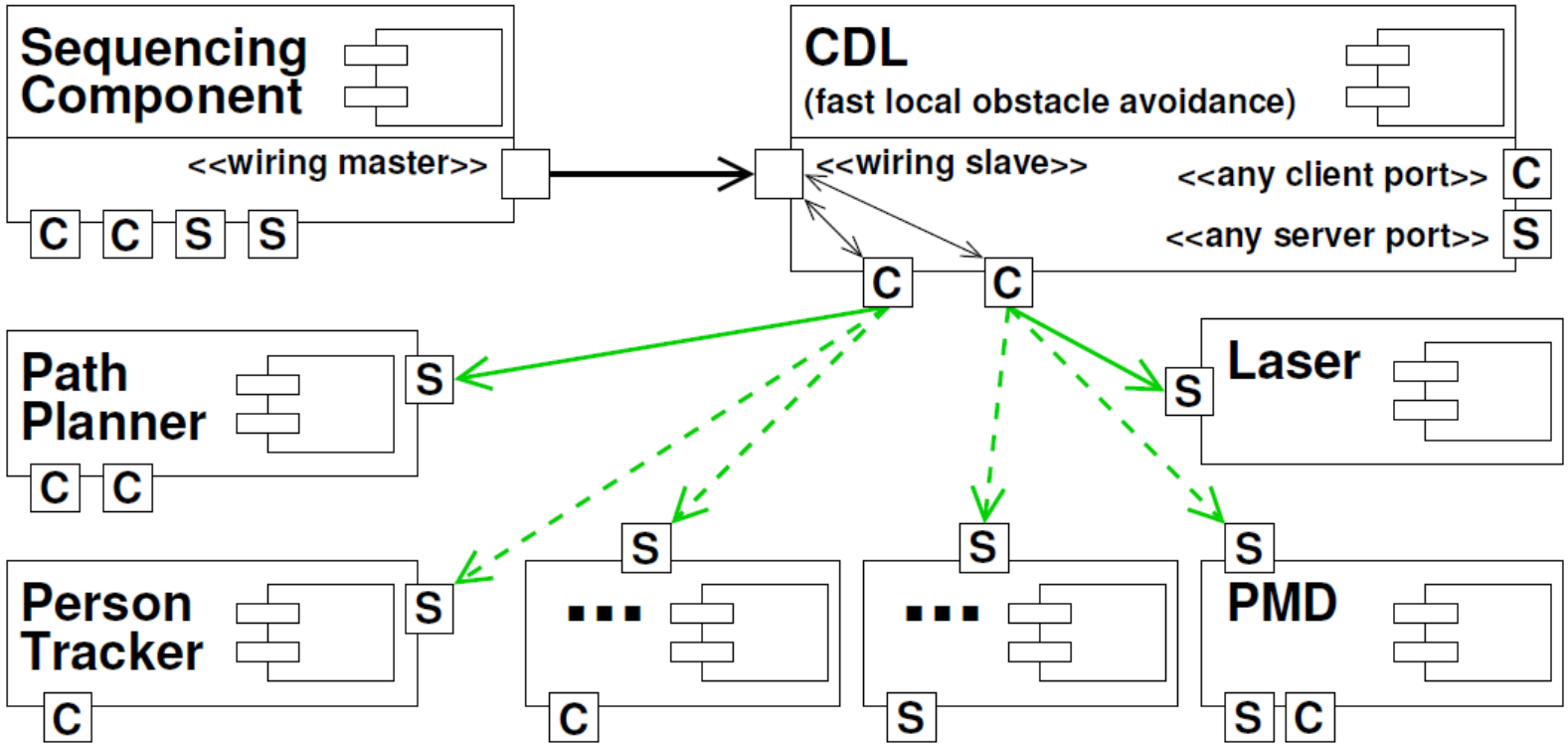
+ add(:WiringSlave*, port:const string&) : StatusCode throw()
+ remove() : StatusCode throw()

+ 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()

Model Driven Software Development
SmartMDSD

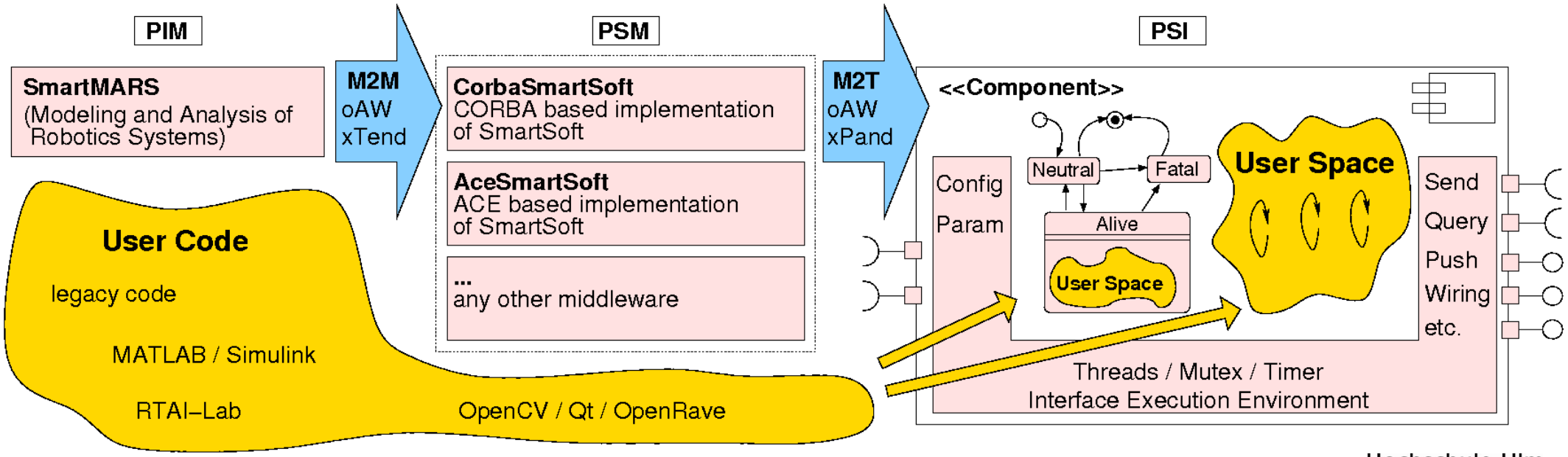


- a wiring master port can be used to rearrange the client connections of components at run-time
- e.g. the CDL component can receive
  - its intermediate waypoints either from a path planner or a person tracker
  - its distance information from either the laser or the PMD component

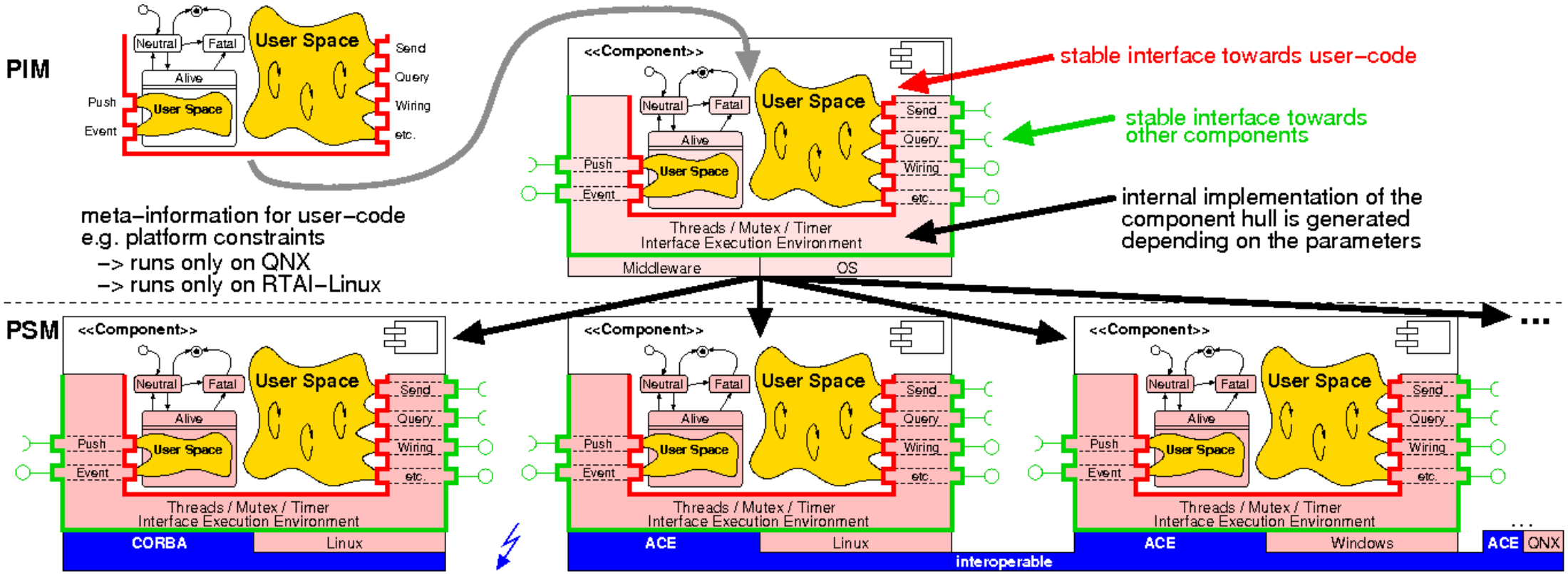
Model Driven Software Development SmartMDSD

Illustration of our development process

- UML 2.0 profile for robotics component model
- covers component development, system composition, deployment
- based on standards: UML 2.0, Open Architecture Ware, Eclipse, etc.
- different runtime platforms, middleware systems etc.



Model Driven Software Development
SmartMDSD



Hochschule Ulm

service robotik

Model Driven Software Development
SmartMDSD

*PlayerPushTimedHandler.cc

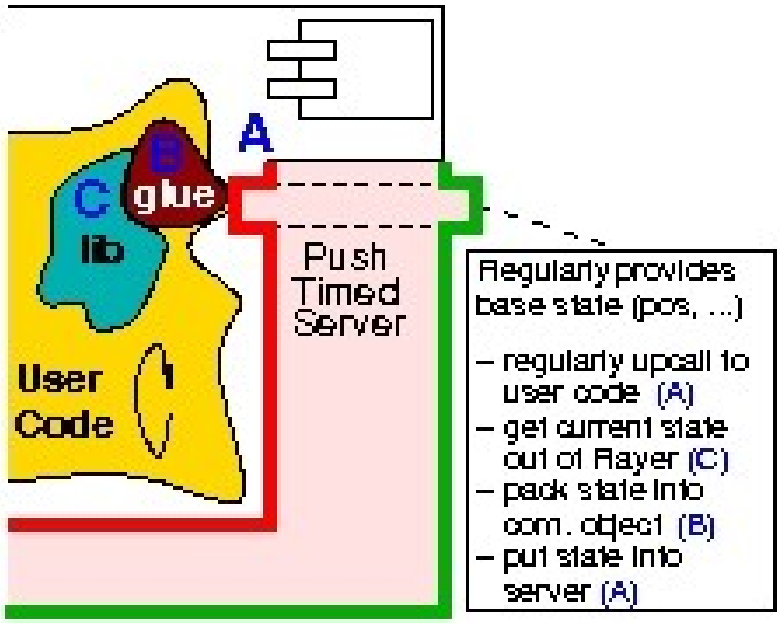
```
#include <iostream>

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

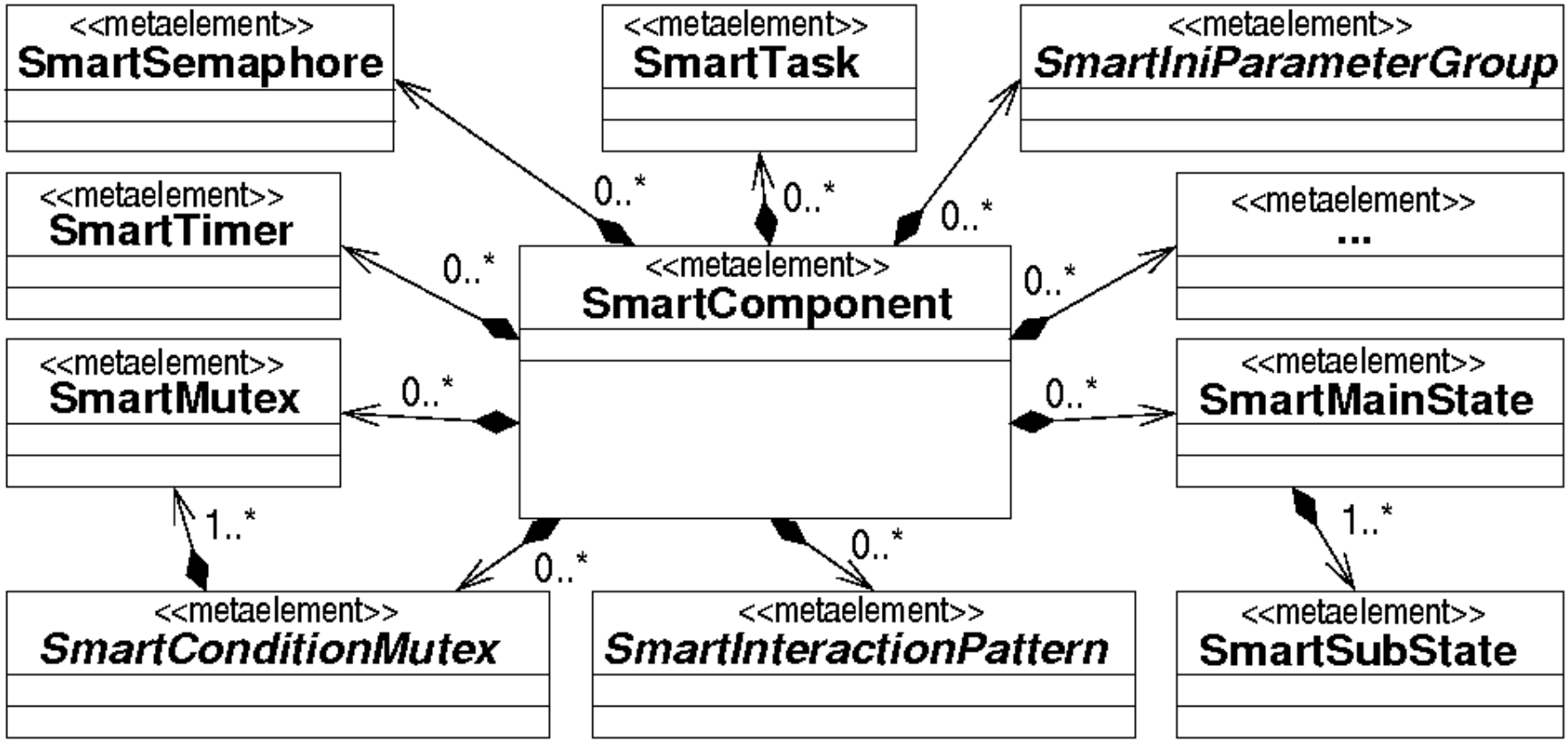
C

B

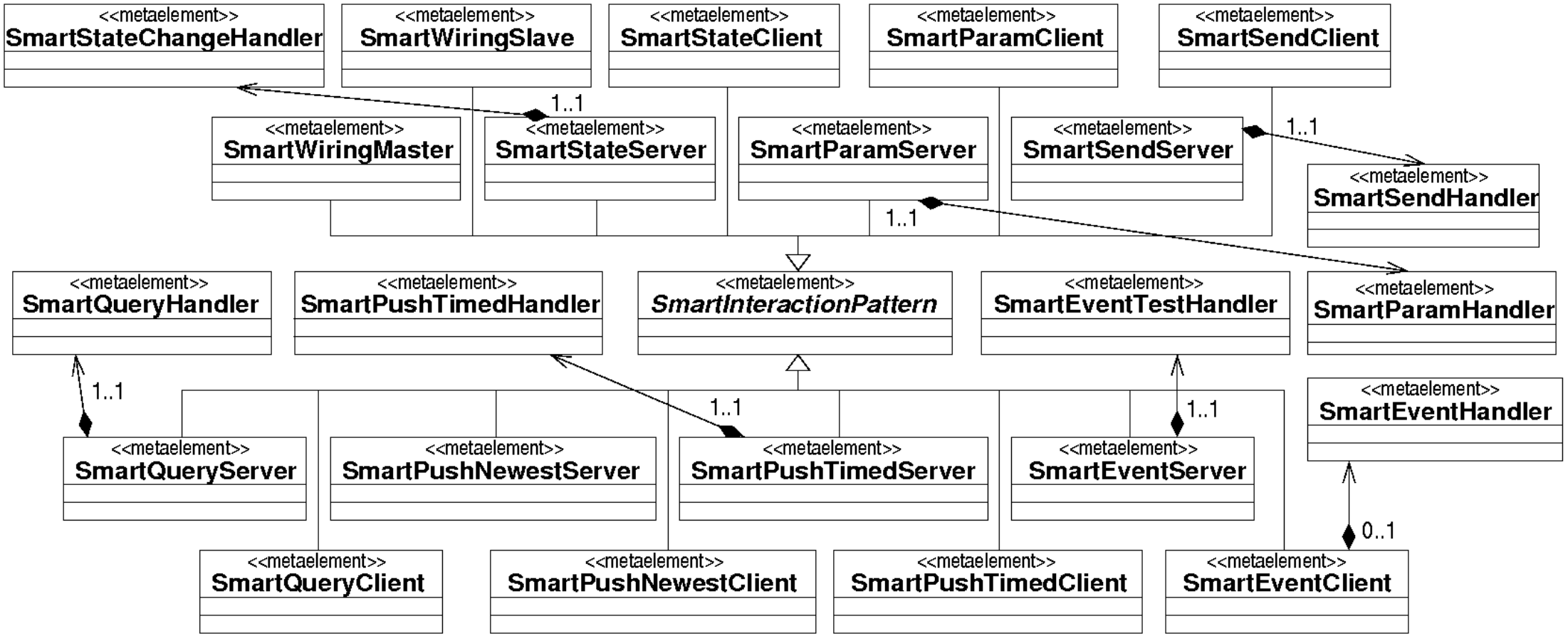
A



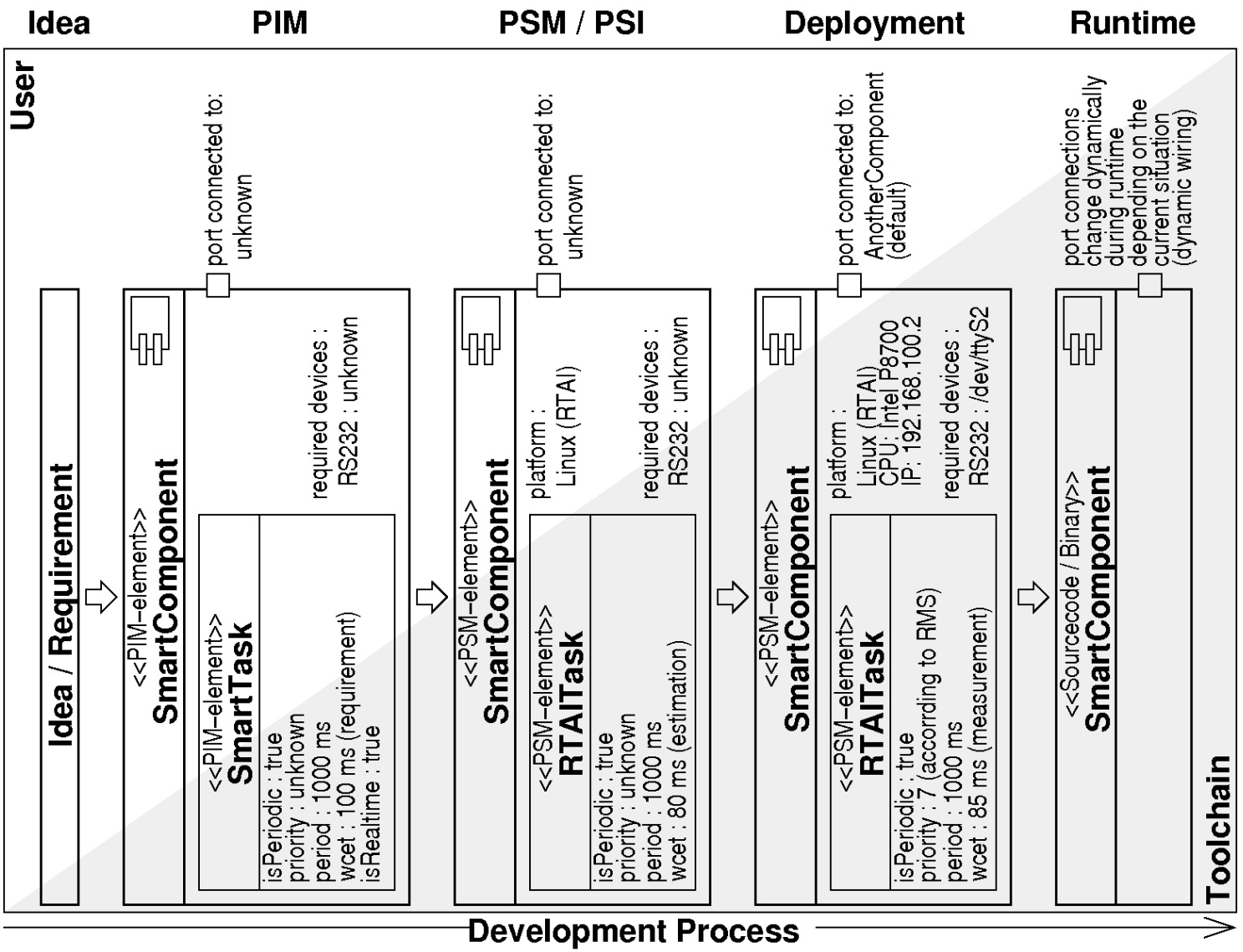
Model Driven Software Development
Metamodels (partial view)



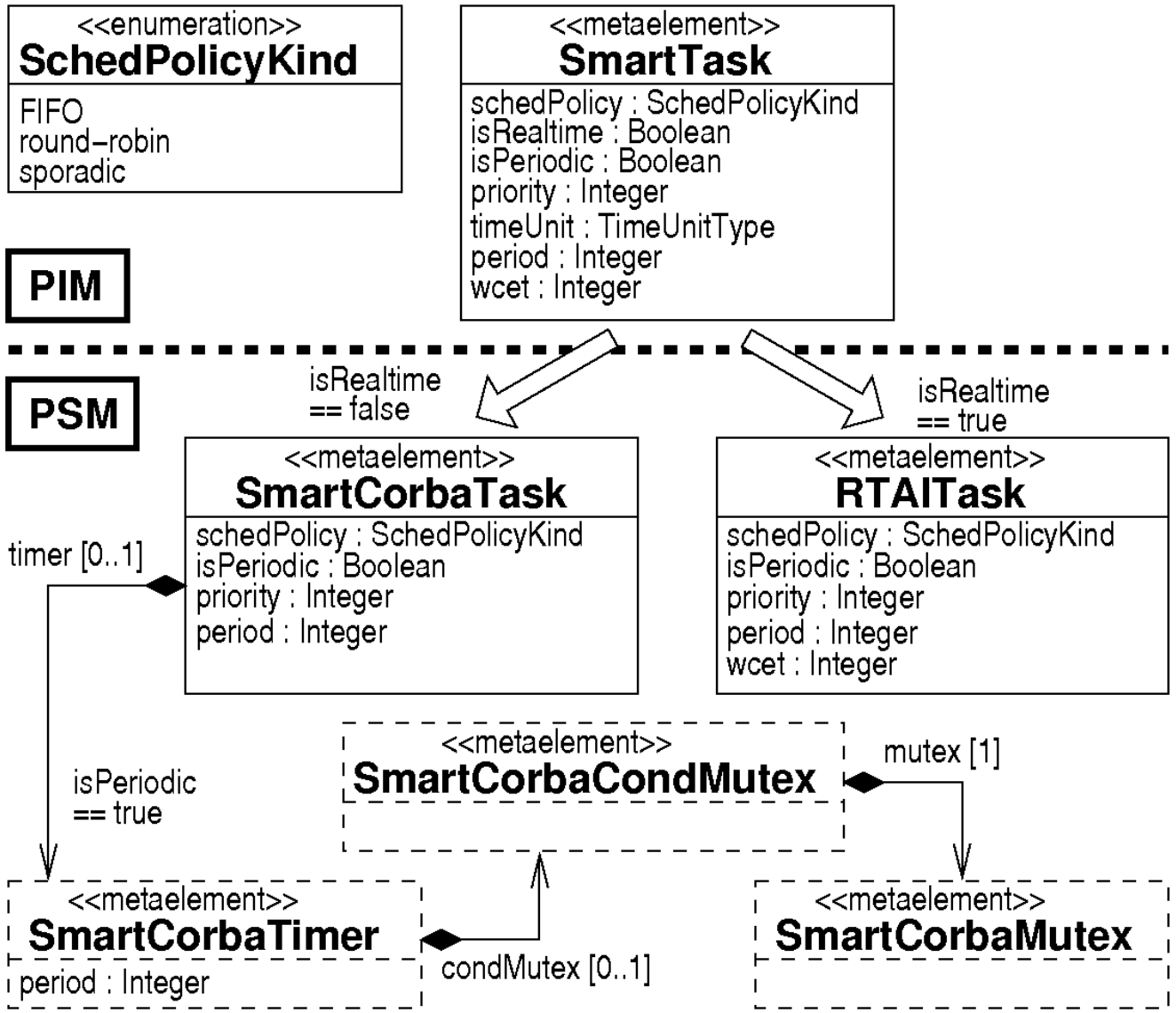
Model Driven Software Development
Metamodels (partial view)



Model Driven Software Development
SmartMDSD



Model Driven Software Development
Examples / SmartMDSD / Real-Time Task



service robotik

Model Driven Software Development
SmartMDSD

Benefits of our development process:

- get rid of hand-crafted single unit service robot systems
- compose them out of standard components with explicitly stated properties
- be able to reuse / modify solutions expressed at a model level
- take advantage from the knowledge of software engineers that is encoded in the code transformators
- be able to verify properties (or at least provide conformance checks)
- be able to address resource awareness !!

and many many more good reasons

Engineering the software development process in robotics is one of the basic necessities towards industrial-strength service robotic systems

Hochschule Ulm

service robotik

ZAFH

Model Driven Software Development SmartMDSD

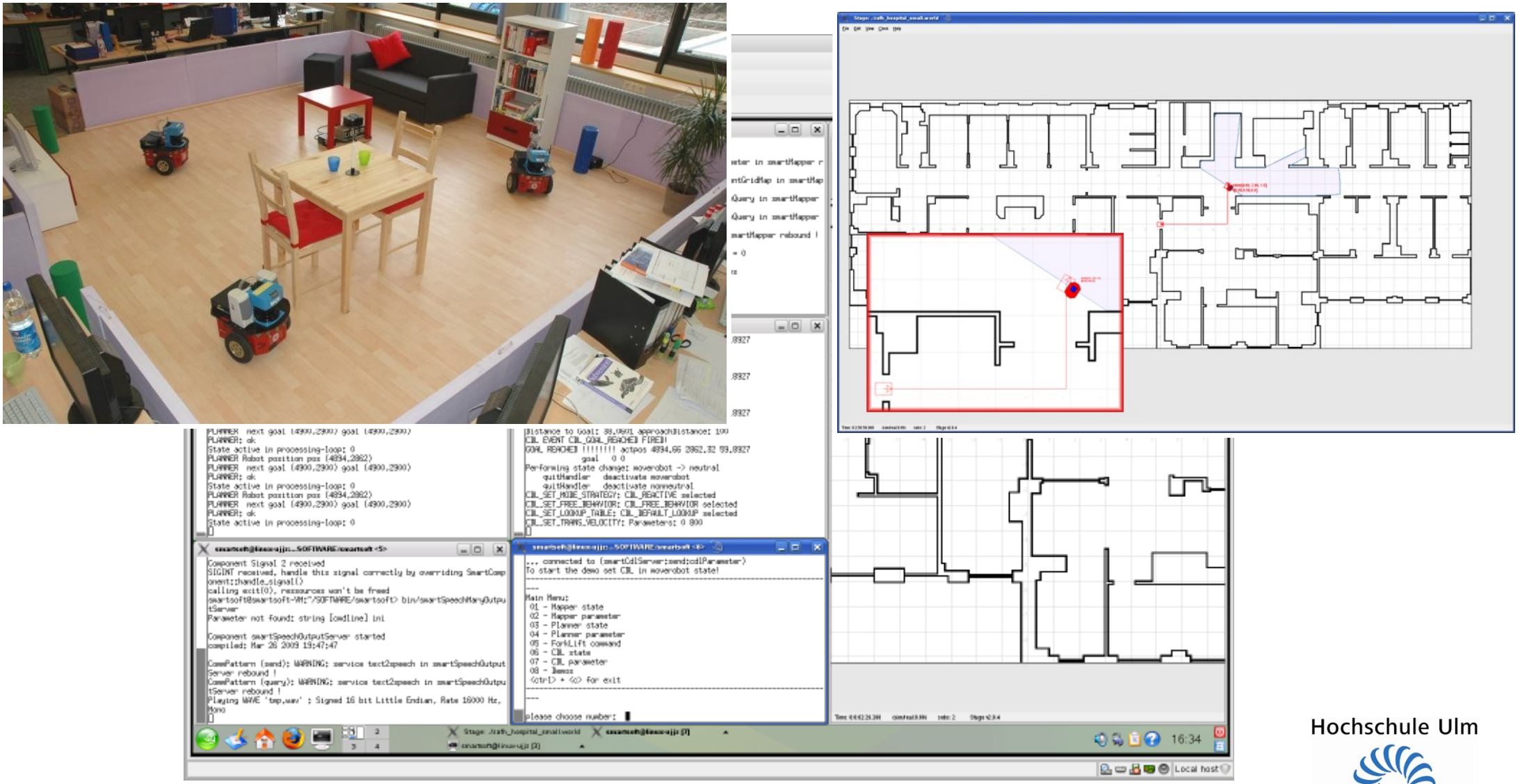
Experience made with our development process

- systematically handle integration of systems of the complexity of service robots and to overcome plumbing
- tools like OpenArchitectureWare, Eclipse etc. are matured enough to be used in robotics
- there are many experienced people out there being already familiar with the tools, can start immediately using them and can just focus on robotics
- design patterns, best practices, approved solutions can be made available within the code generators such that even novices can immediately take advantage from that coded and immense experience
- provides the perfect granularity for system design, component development, composability, freedom within components, tool support etc.

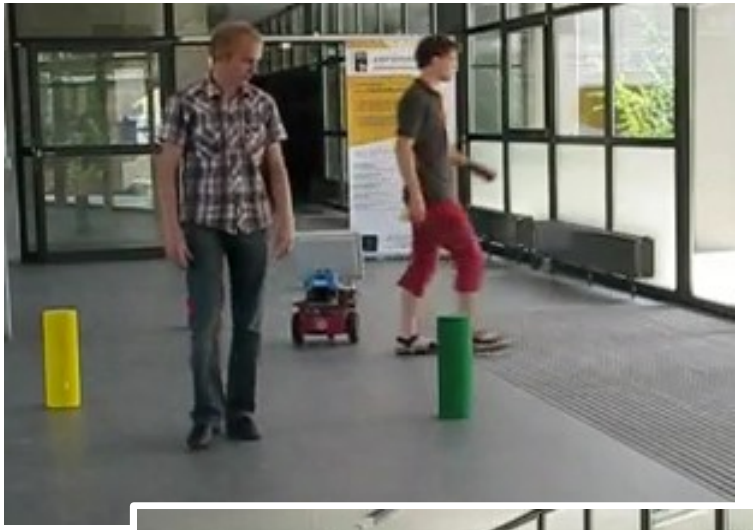
Hochschule Ulm

service robotik

Model Driven Software Development
Examples / Simulation Player/Stage



Example: “Follow Me”
Reuse for RoboCup@Home

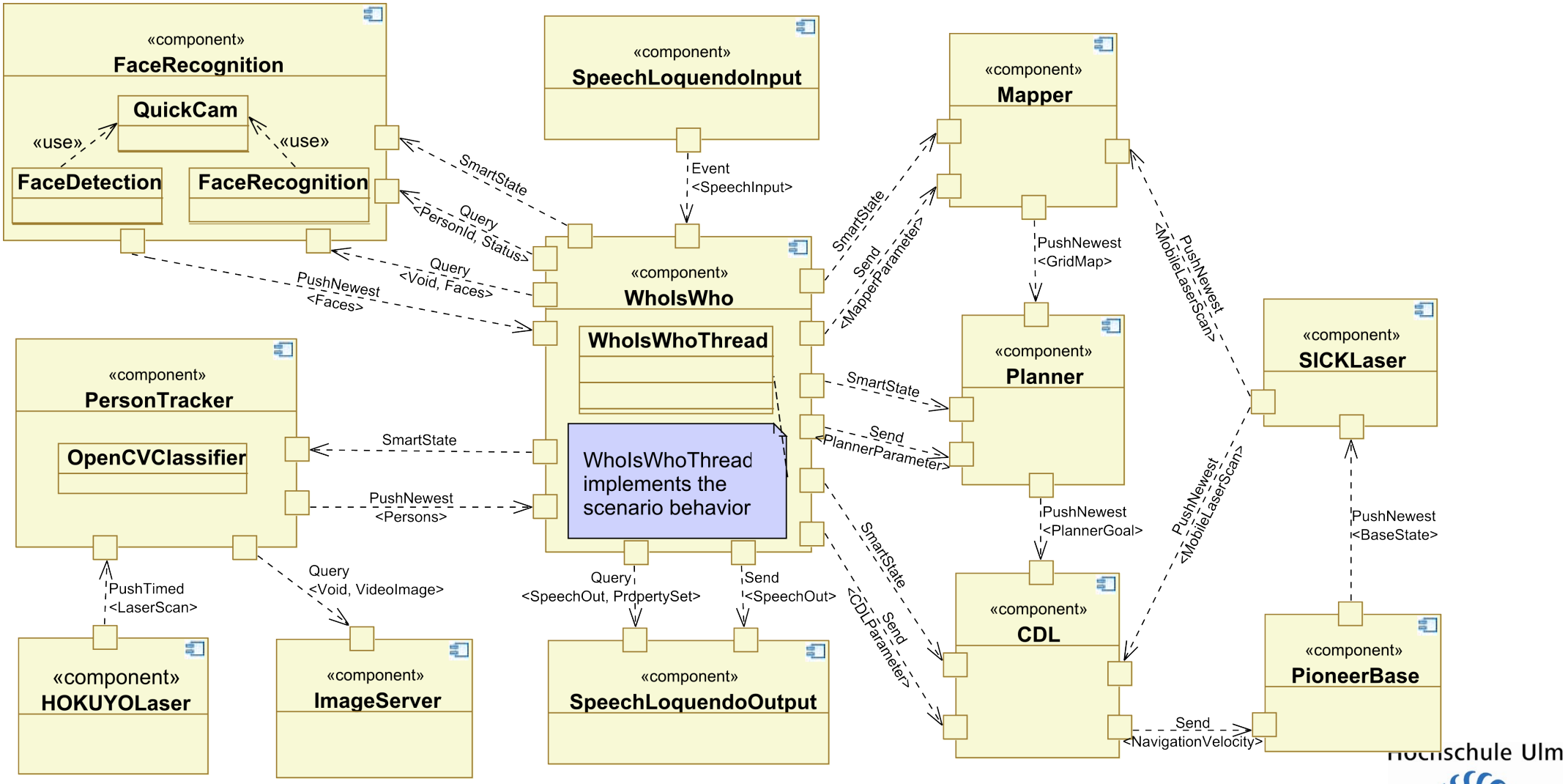


watch on youtube:

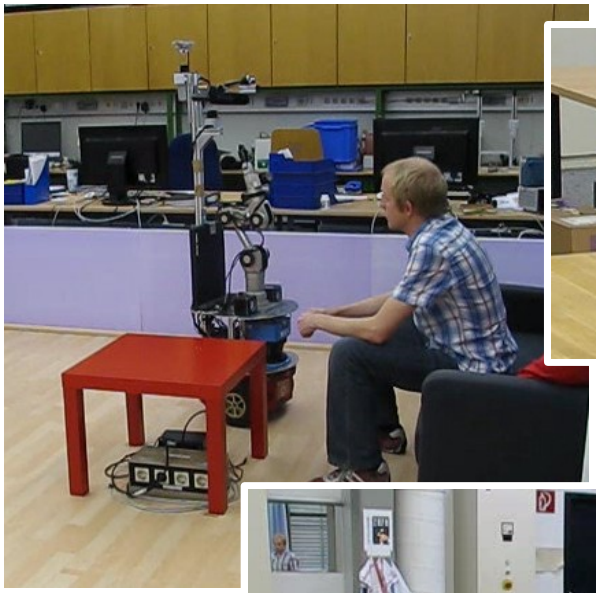
http://www.youtube.com/roboticsathsulm
http://www.youtube.com/watch?v=-xvO_bbrb-I
http://www.youtube.com/watch?v=grf-_-32oY0

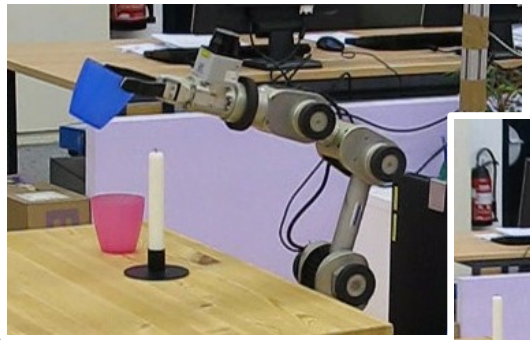
Hochschule Ulm

Model Driven Software Development
Examples / Robocup@Home / “WhoIsWho“



Example: “Cleanup Table Scenario”





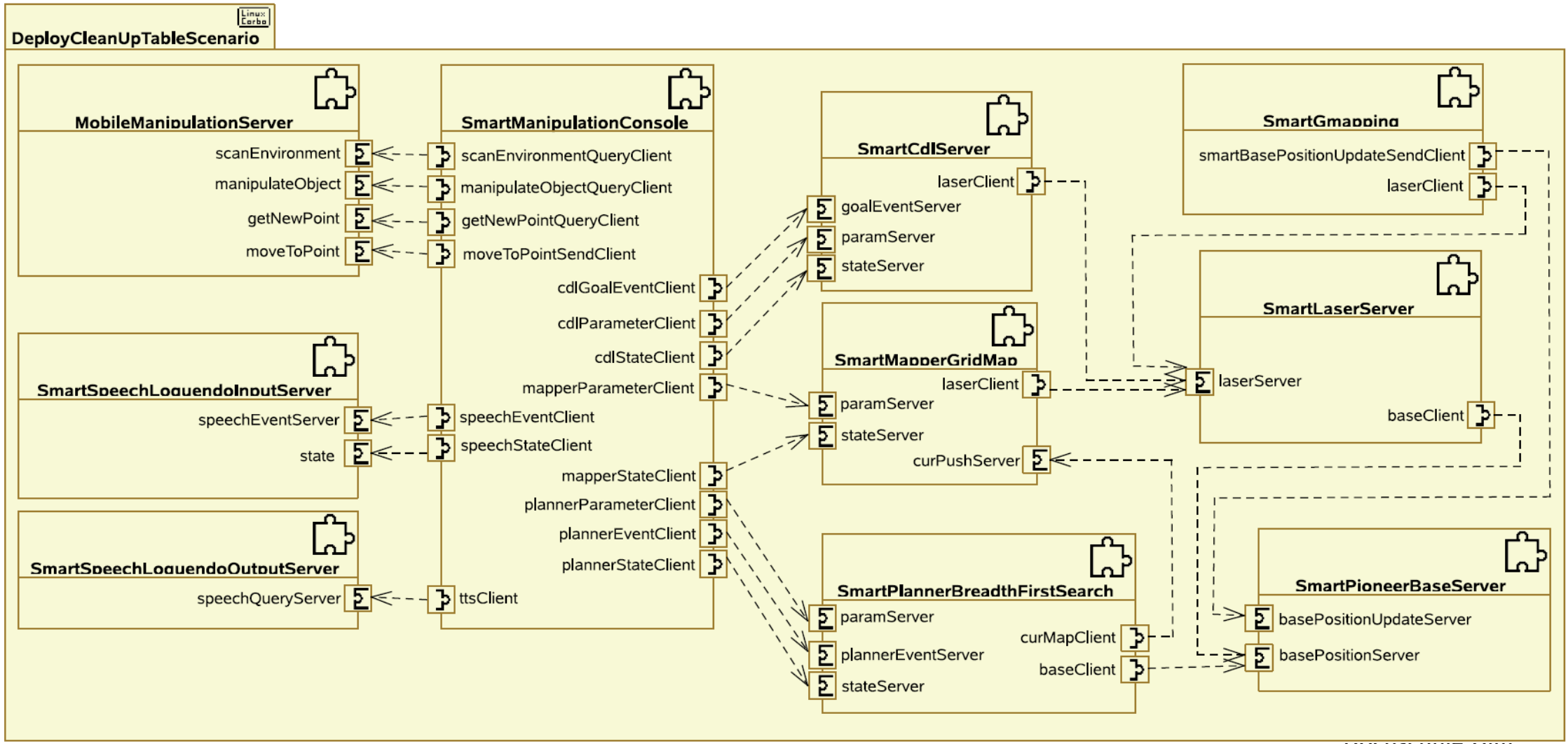
deployment of COTS components

complex real world scenario
in everyday environment

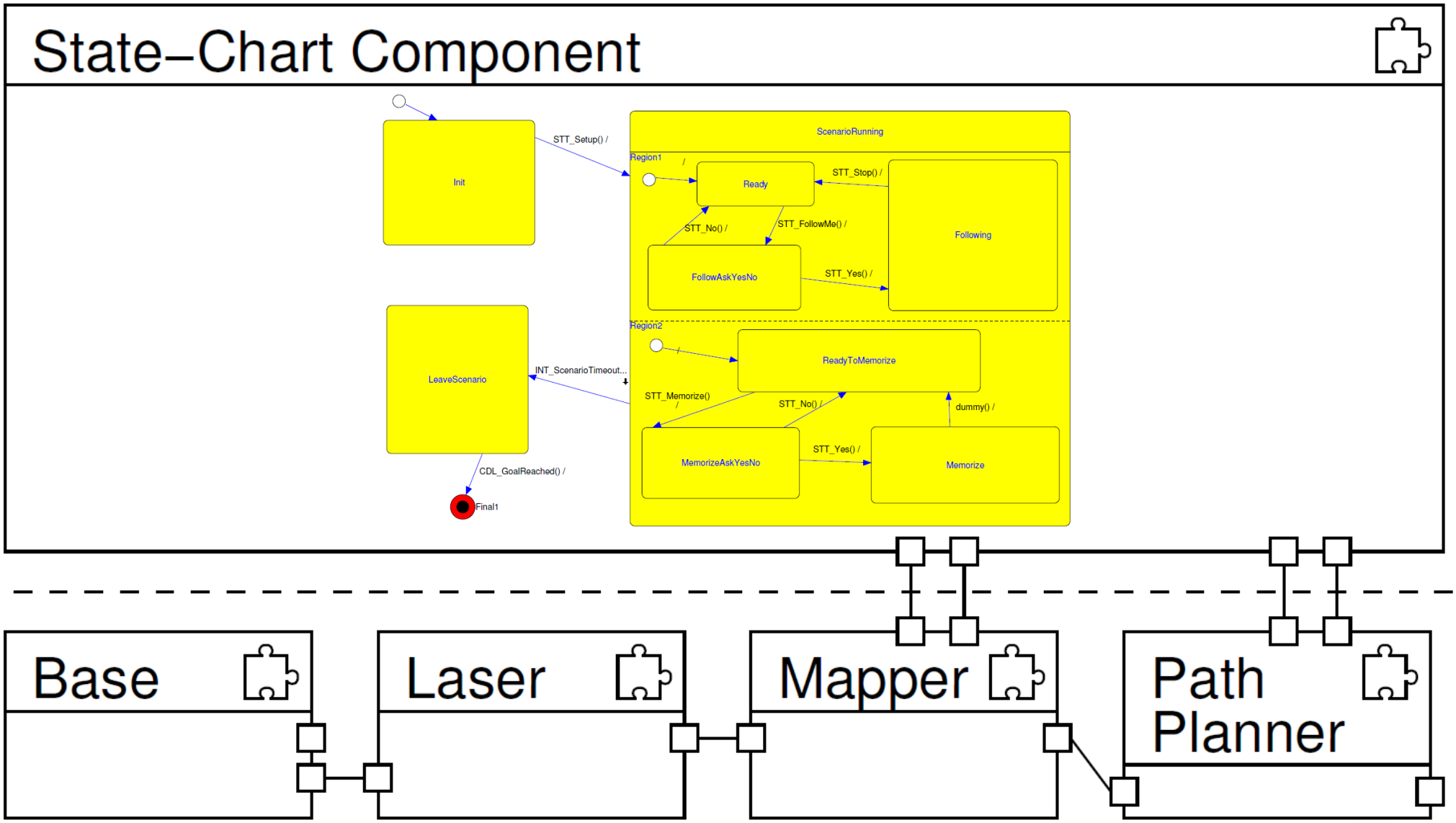
robot cleans up the table

Hochschule Ulm

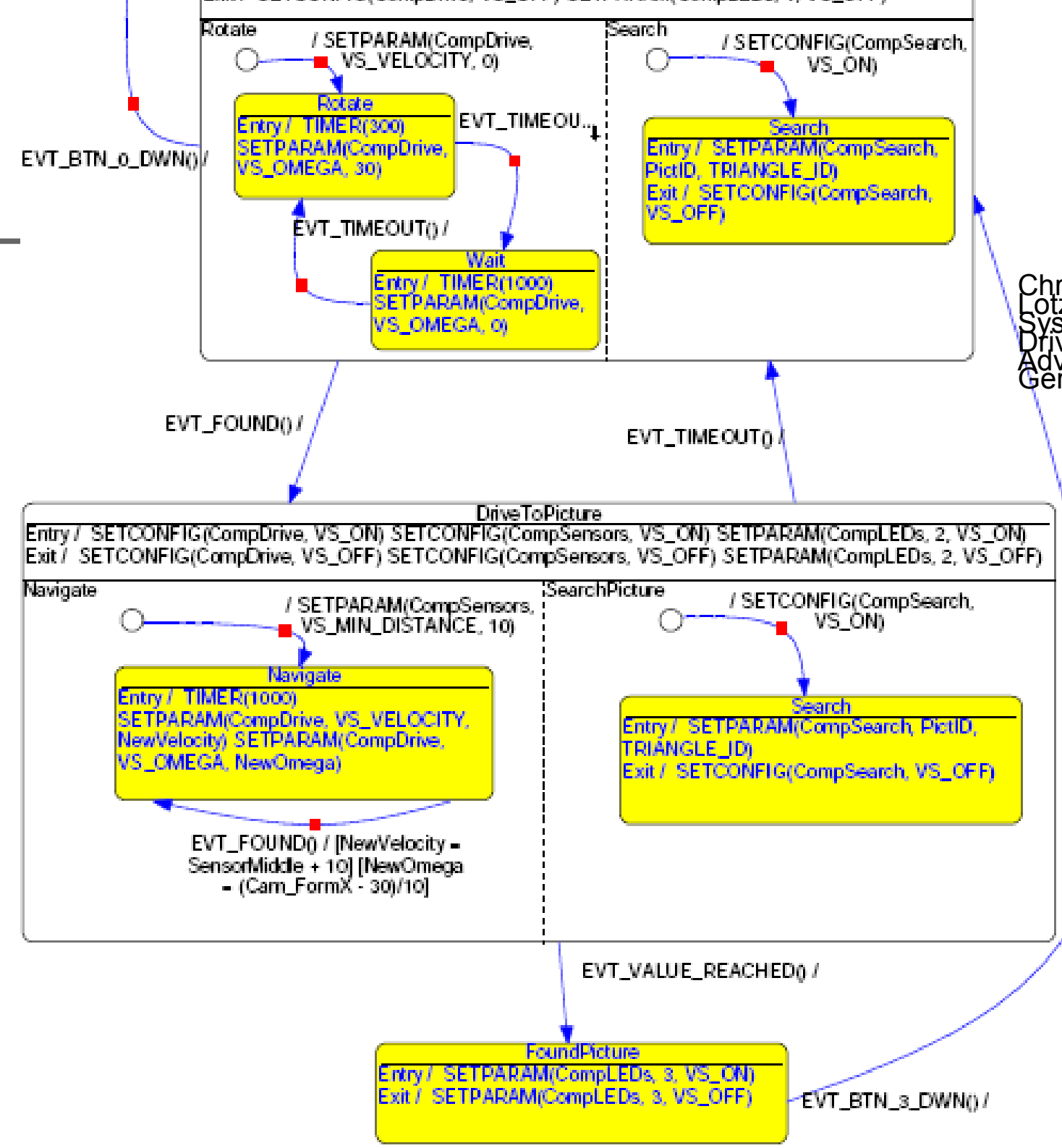
Example: “Cleanup Table Scenario”



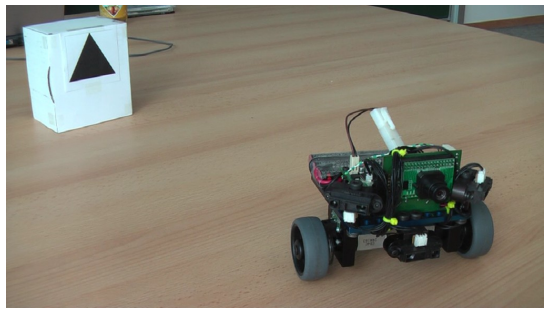
Orchestration of Components using State Charts Dynamic Online Reconfiguration



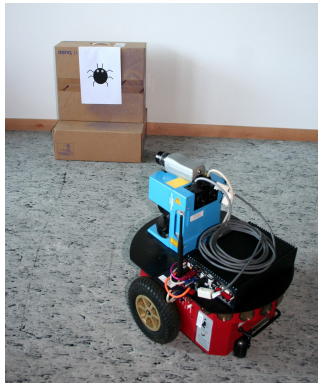
ual State)



Christian Schlegel, Thomas Haßler, Alex Lotz and Andreas Steck, "Robotic Software Systems: From Code-Driven to Model-Driven Designs", In Proc. 14th Int. Conf. on Advanced Robotics (ICAR), Munich, Germany, 2009.



Qfix – ATMega 8Bit (FreeRTOS)

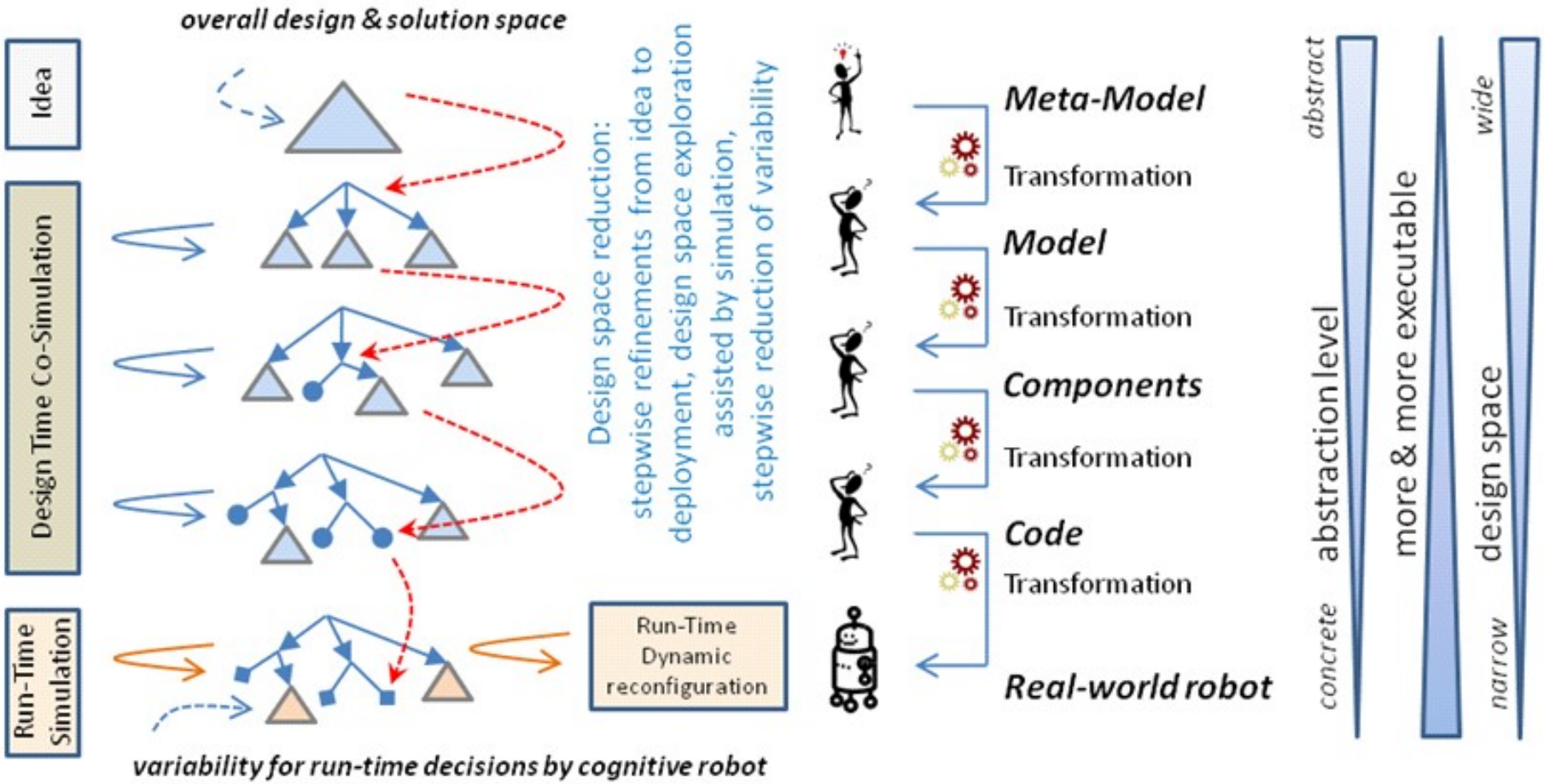


P3DX – x86 (Linux)

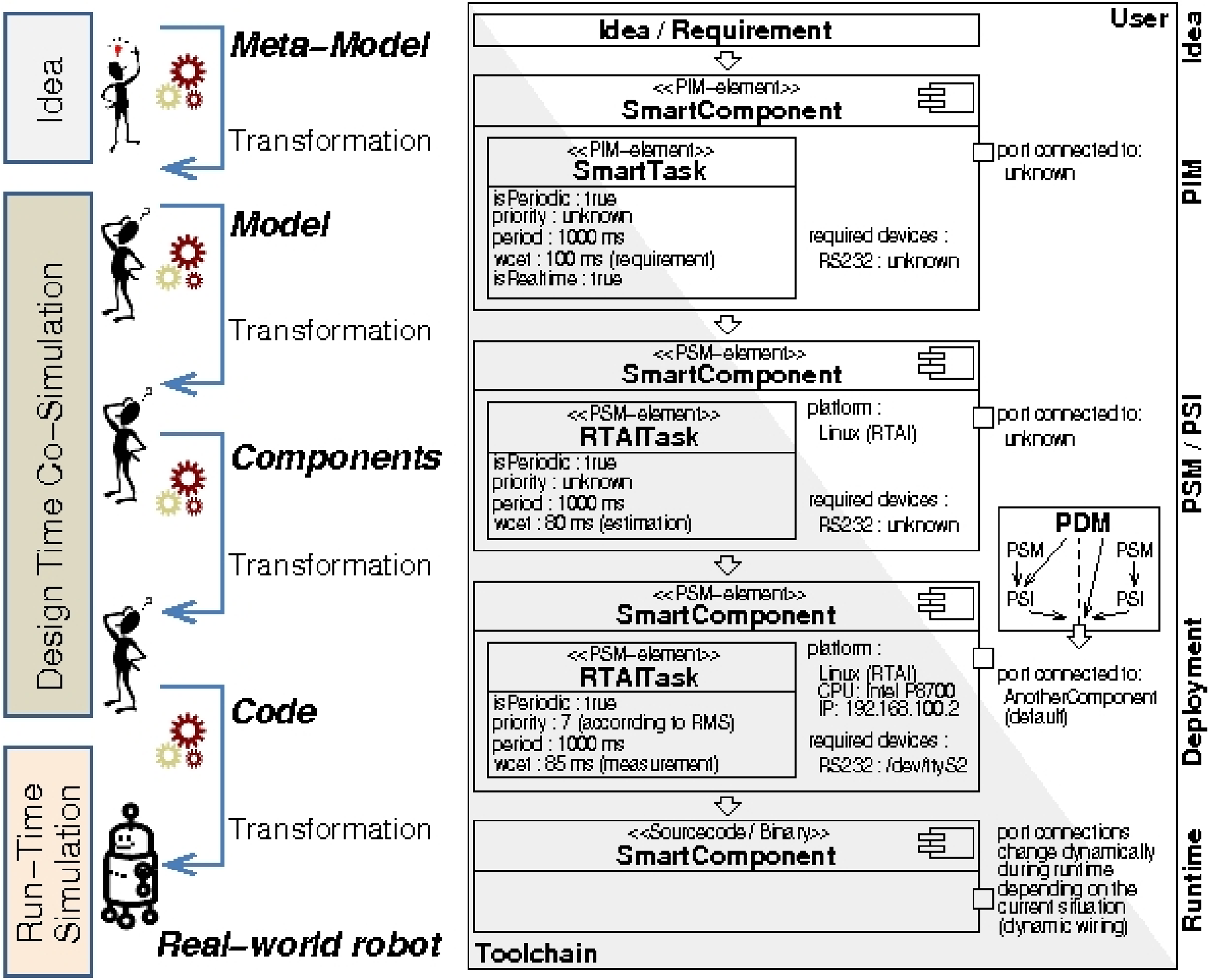
Hochschule Ulm

service robotik

Approach: Design Abstraction



Hochschule Ulm

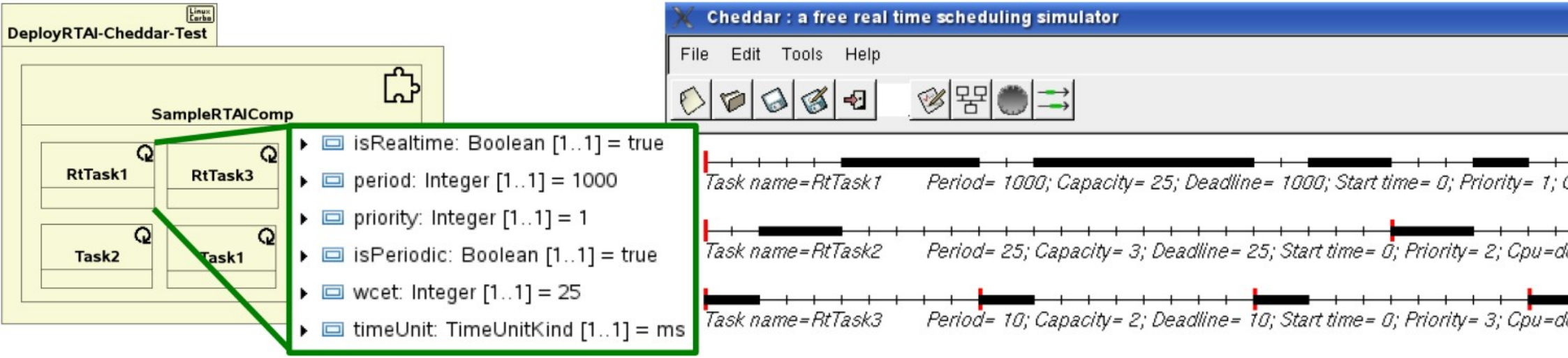


Hochschule Ulm

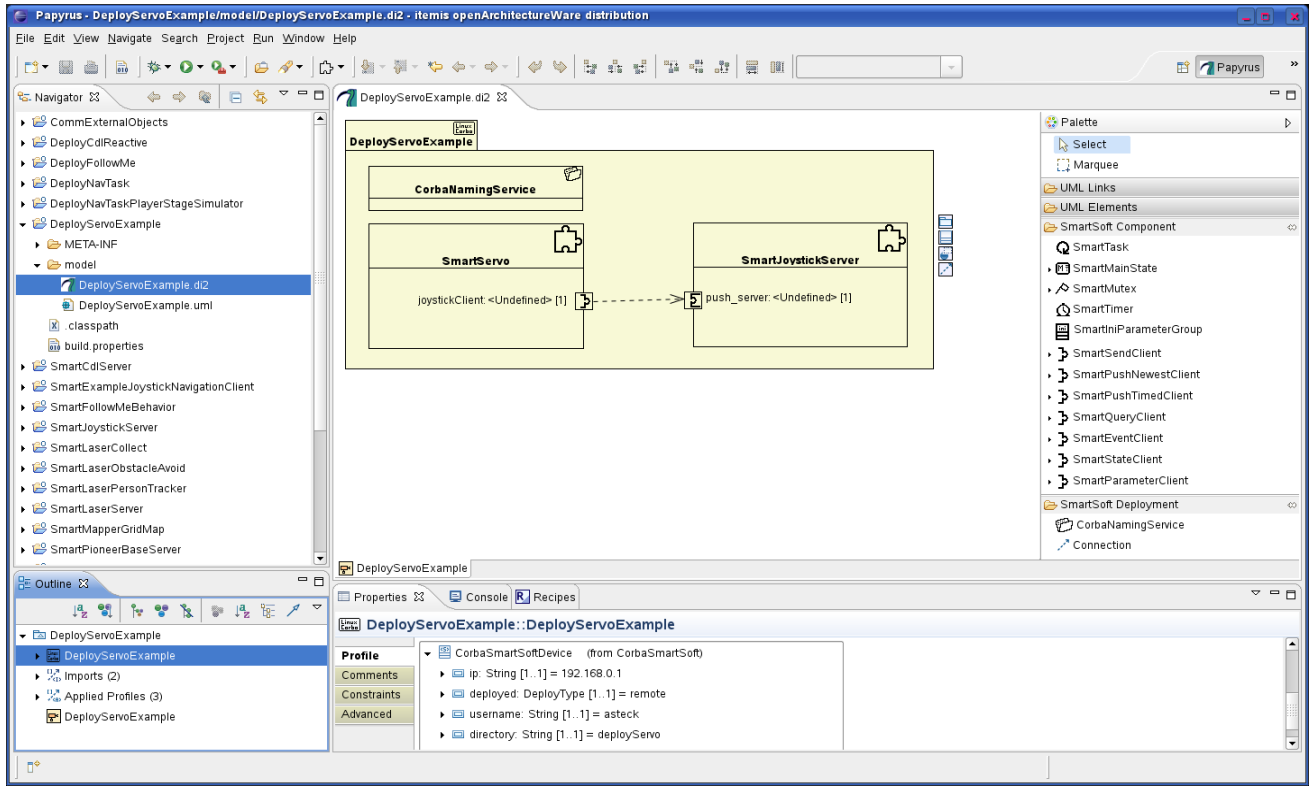
Approach: Design Abstraction

Resource Awareness and Quality of Service

- Example: Schedulability Analysis (CHEDDAR)



SmartSoft MDSD Toolchain



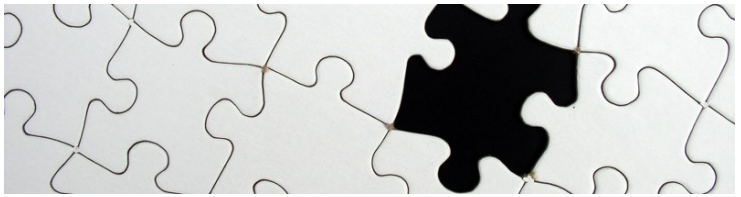
Eclipse

openArchitectureWare

PapyrusUML

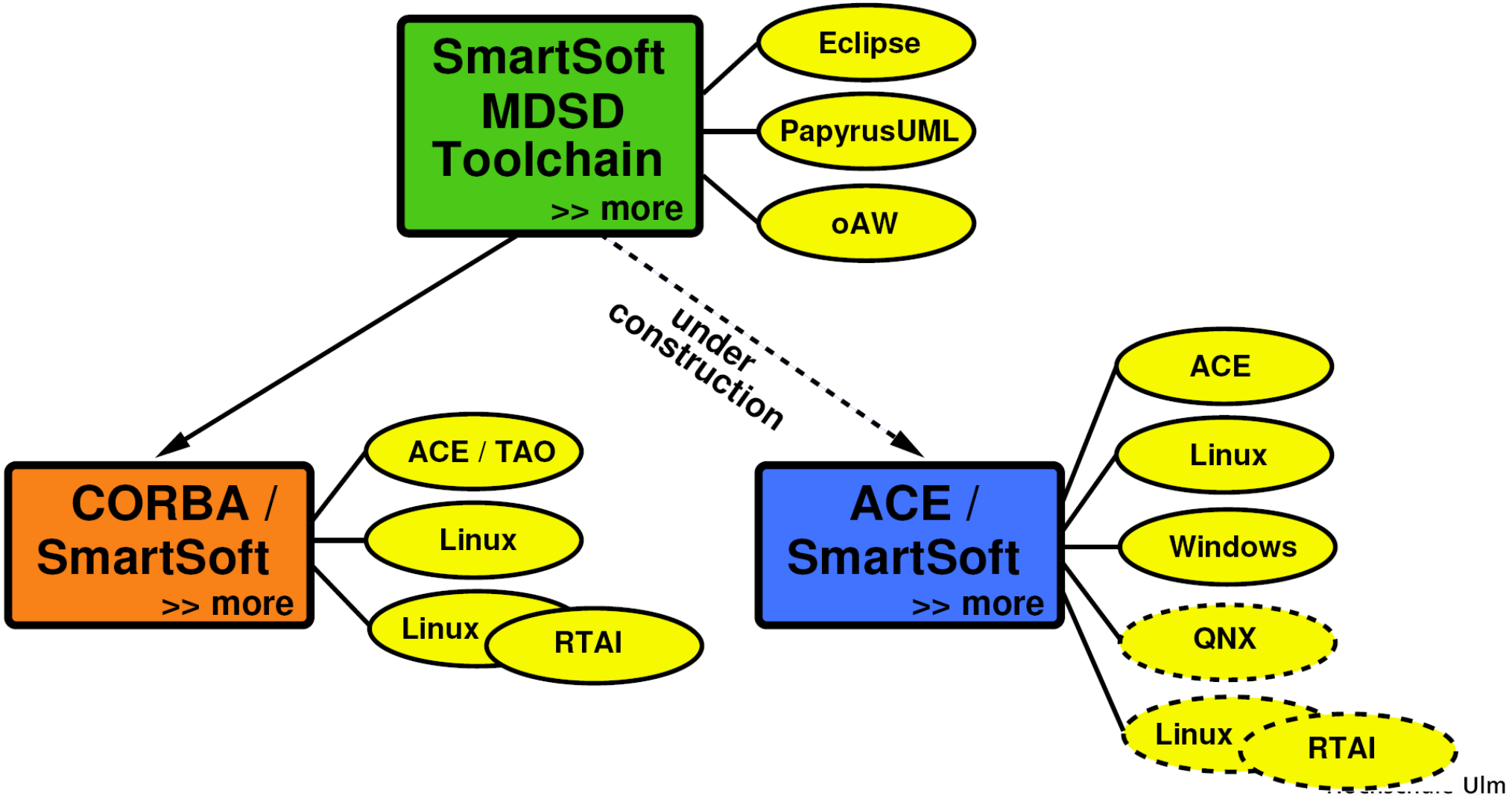
Hochschule Ulm

Model Driven Software Development
It is all available (LGPL) ...

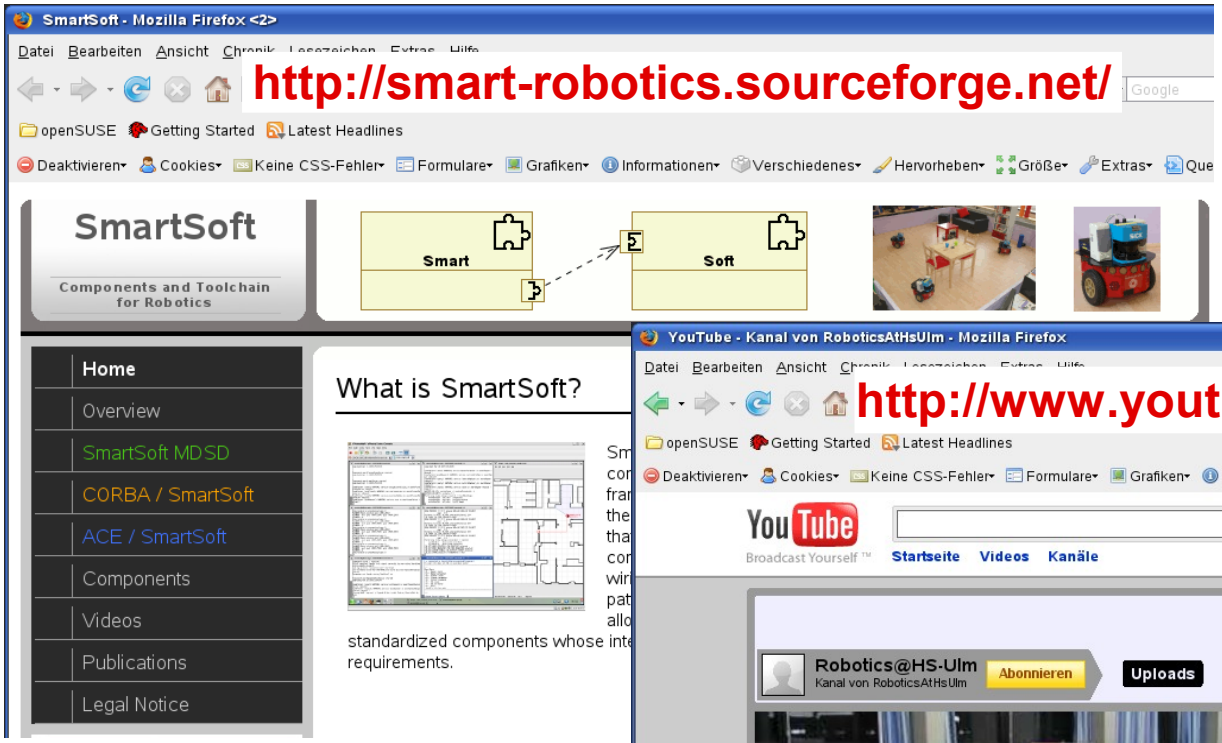


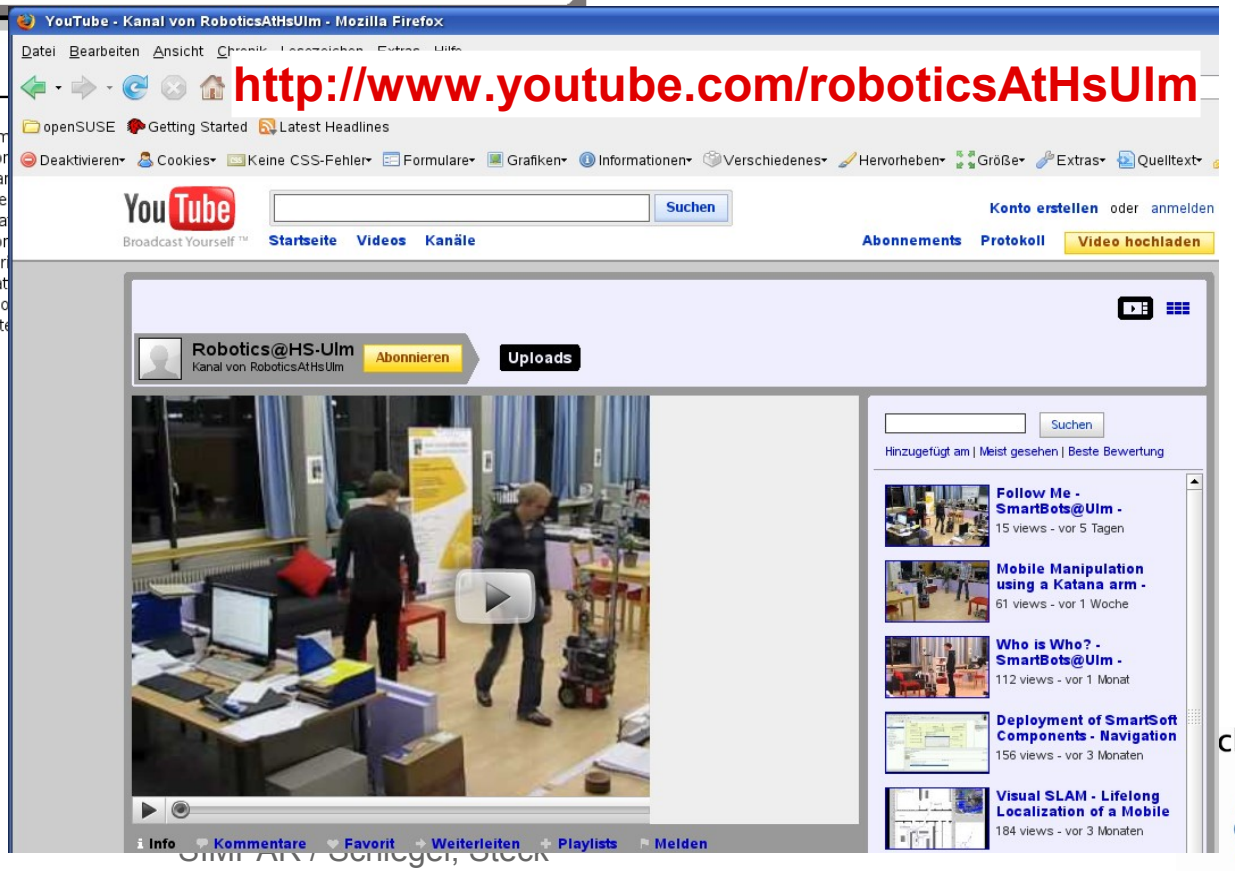
- Toolchain based on Open Architecture Ware
  - fully integrated into **Eclipse**
  - *http://www.openarchitectureware.org/*
- MDSD Toolchain Example
  - PIM: SmartMARS robotics profile (Modeling and Analysis of Robotics Systems)
  - PSM: SmartSoft in different implementations but with the same semantics !
  - can be easily adapted to different profiles / profile extensions / PSMs
- Short Summary on SmartSoft [LGPL]
  - ***http://smart-robotics.sourceforge.net/***
  - ***http://www.zafh-servicerobotik.de/ULM/en/smartsoft.php***
  - ***http://www.youtube.com/user/RoboticsAtHsUlm***
  - CORBA (ACE/TAO) based SmartSoft
    - on sourceforge with various robotics components and simulators etc.
    - in use in research and industry
  - ACE (without CORBA) based SmartSoft
    - on sourceforge [Linux, Windows]
    - in use in research and industry
  - oAW Toolchain for SmartSoft
    - on sourceforge ***(including Screencasts and Tutorials)***

SmartSoft MDSD Toolchain



SmartSoft MDSD Toolchain Links



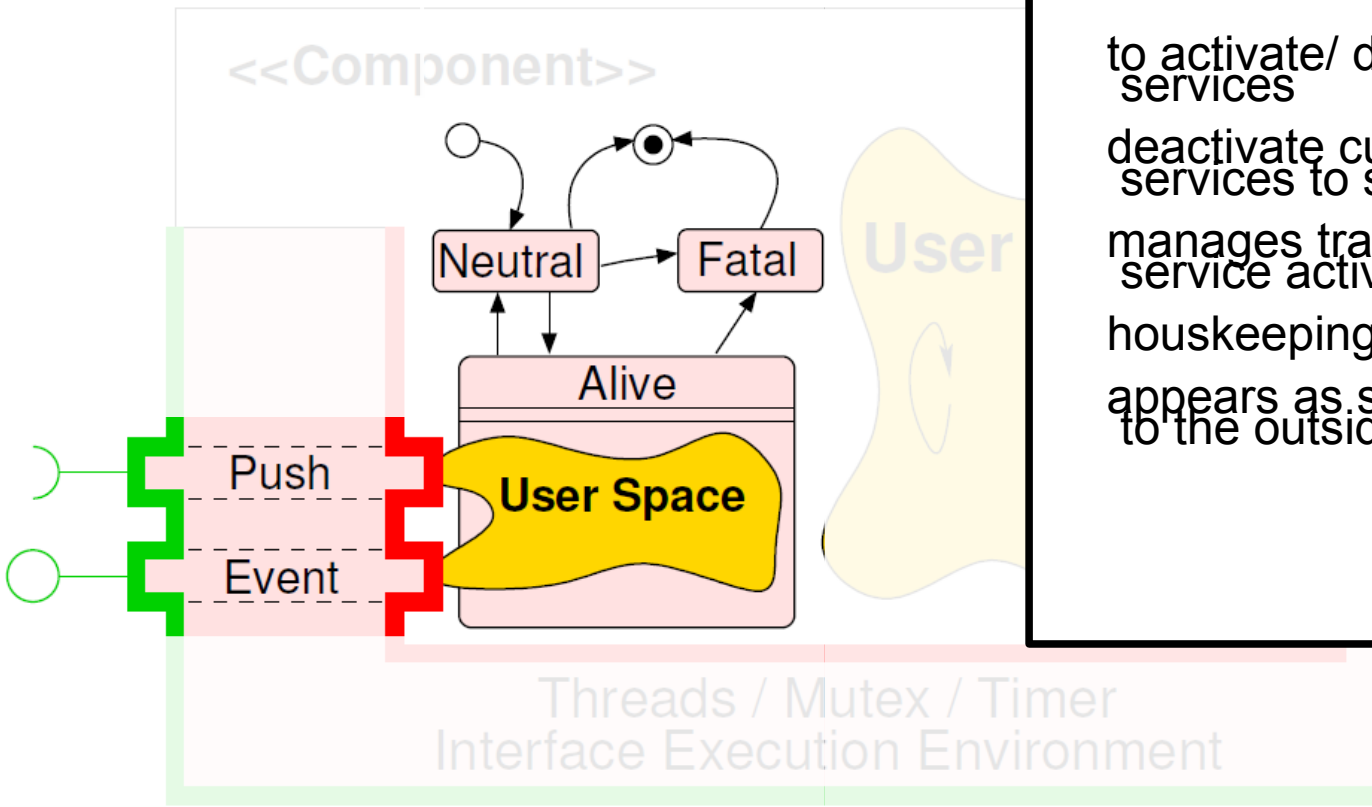


service robotik

ZAFH

Addendum

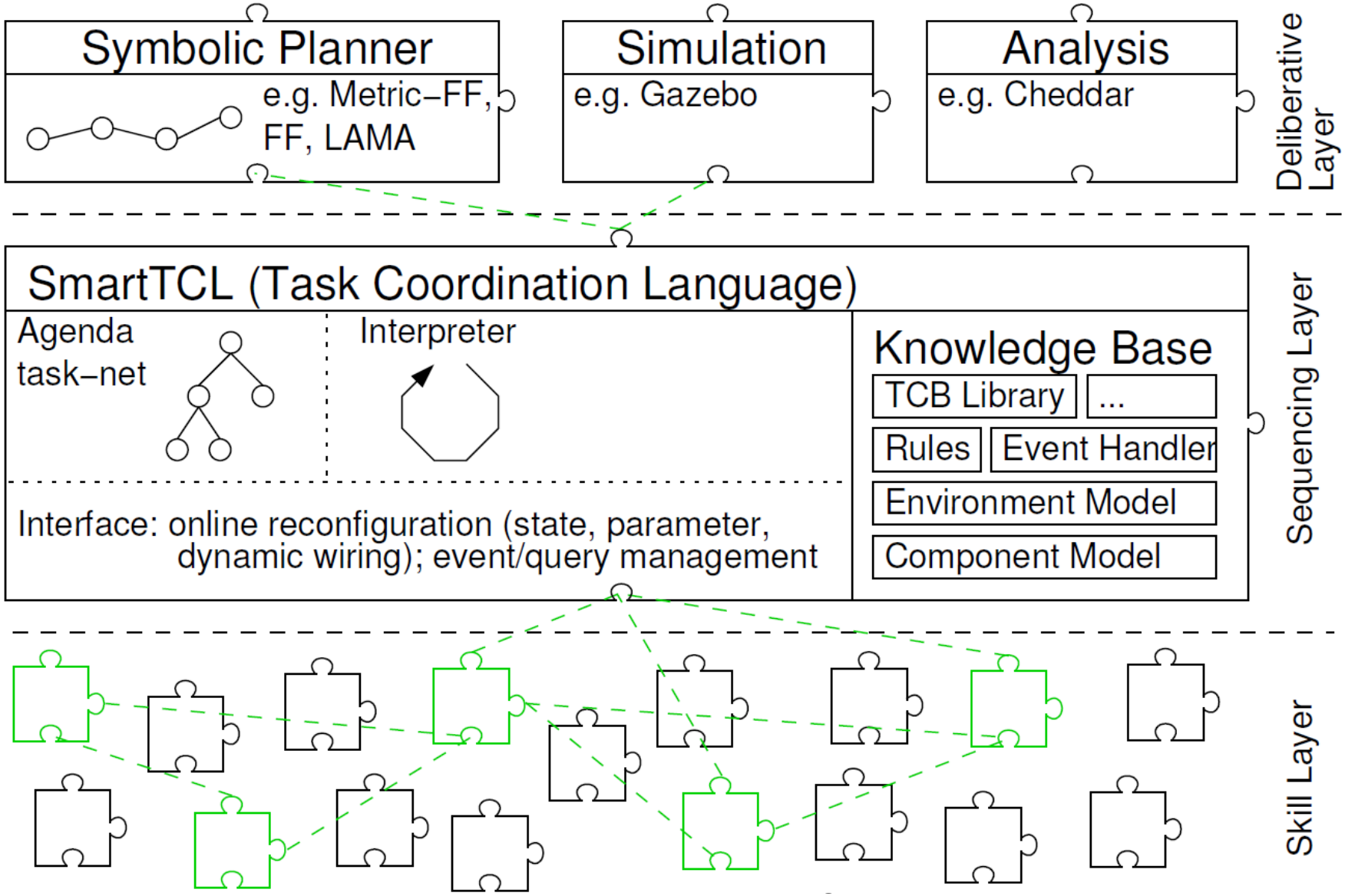
Resource Awareness
State Pattern – State Automaton



- to activate/ deactivate component services
- deactivate currently unused services to save resources
- manages transitions between service activations
- houskeeping activities
- appears as stateless interface to the outside

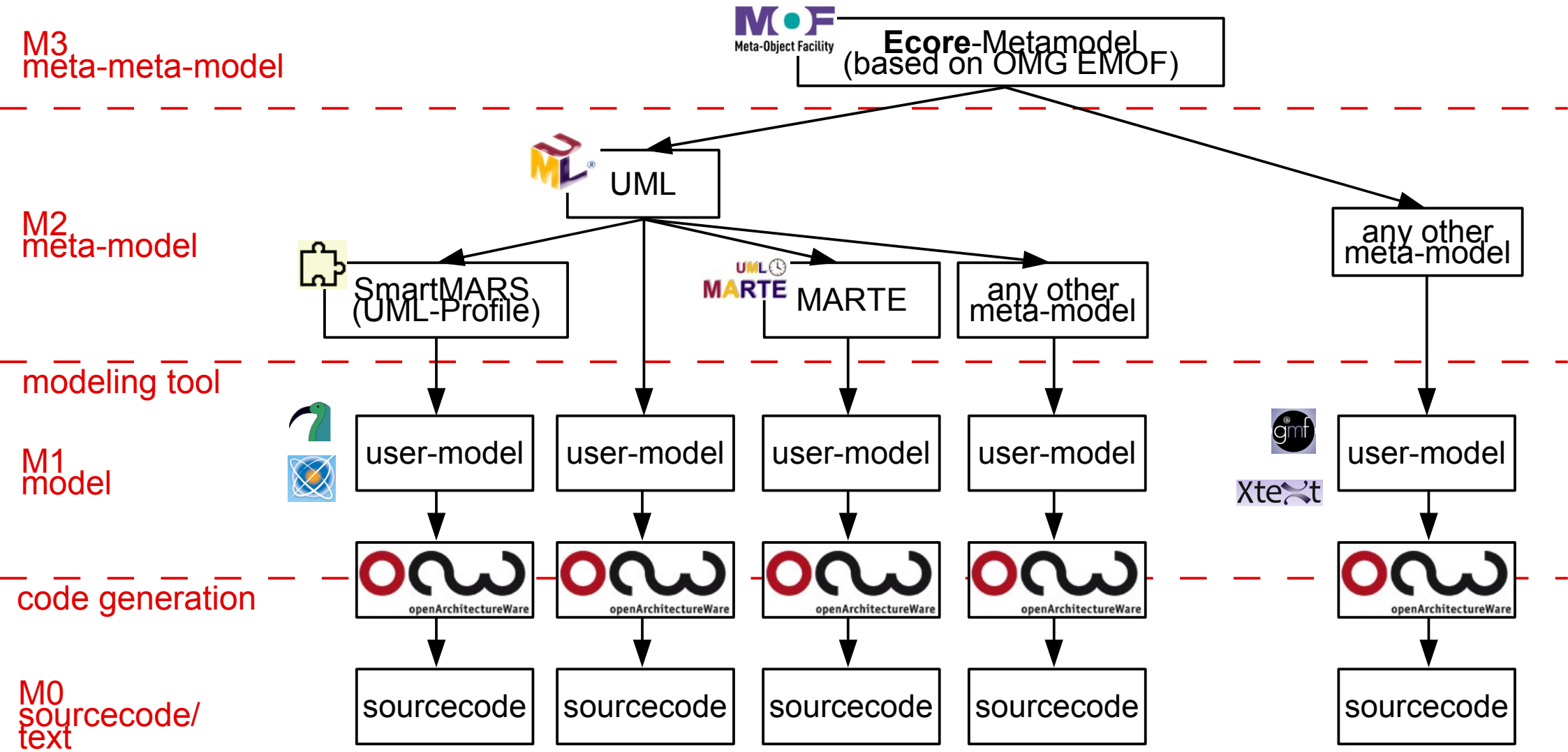
service robotik

The Three Layer Robot Control Architecture based on the SMARTSOFT Concepts



Hochschule Ulm

Eclipse Modeling Framework
Meta-Modeling



SmartSoft MDSD Toolchain
Generation Gap Pattern