# Model Driven Software Development in 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/

Siegfried Hochdorfer, Alex Lotz, Matthias Lutz, Dennis Stampfer, Andreas Steck, Jonas Brich, Manuel Wopfner

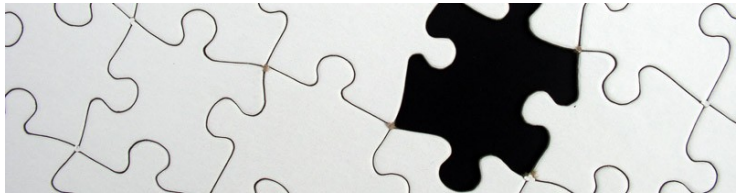
Hochschule Ulm

# Model Driven Software Development
# Introduction and Motivation

## What is this talk about?

- not just another software framework
- not just another middleware wrapper

➔ we have plenty of those ...



## But

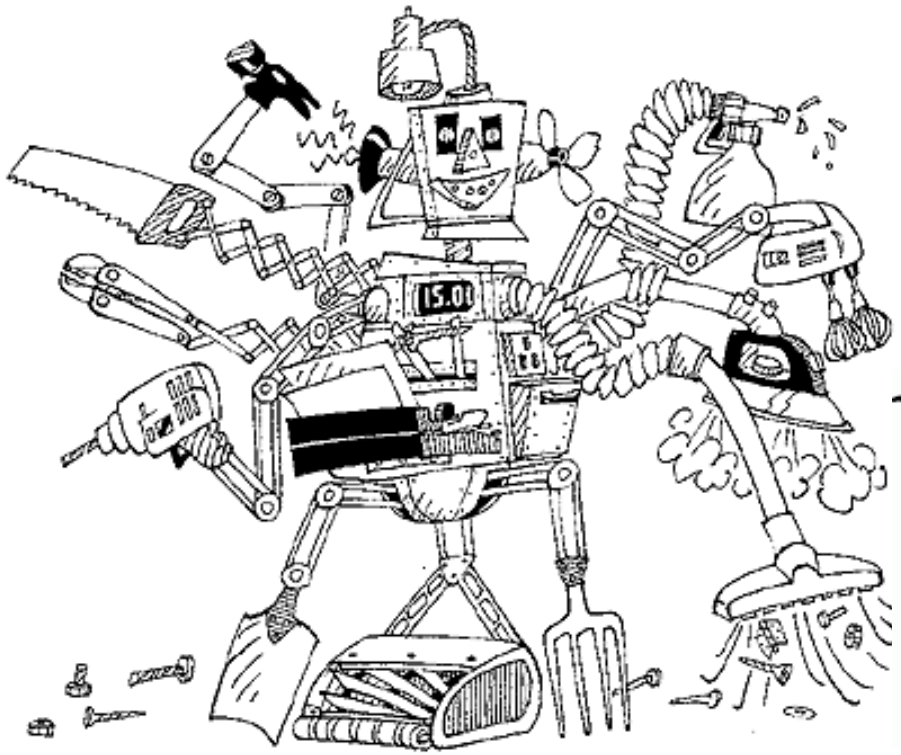
- separation of robotics knowledge from short-cycled implementational technologies
- providing sophisticated and optimized software structures to robotics developers not requiring them to become a software expert

## How to achieve this?

- make the step from code-driven to model-driven designs
- there are open source tools, standards etc. also useful in robotics!

# Model Driven Software Development
# Introduction and Motivation

A development process often applied in robotics ...

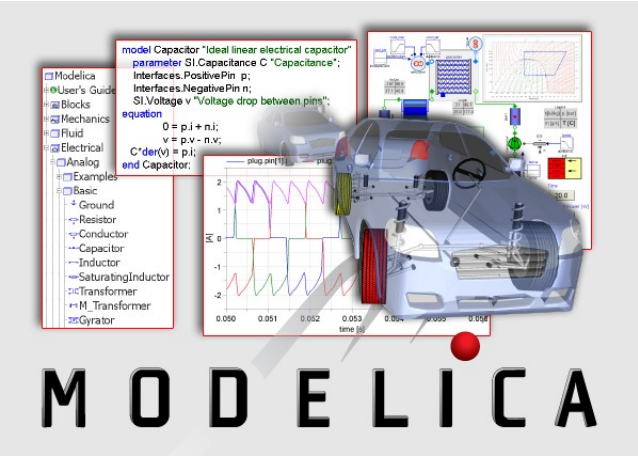


service robotik

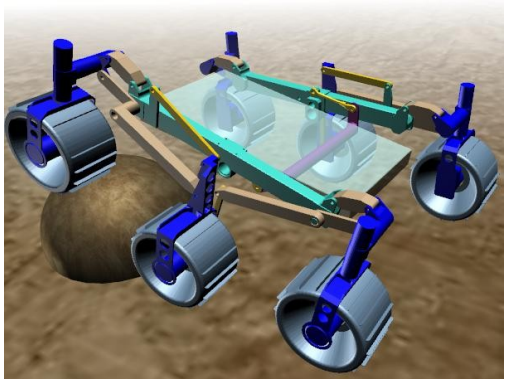
# Model Driven Software Development
# Introduction and Motivation

Systematic engineering processes look different ...

... also in case of software intensive systems







UML MARTE

Hochschule Ulm

service robotik

# Model Driven Software Development
# Introduction and Motivation

## 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***





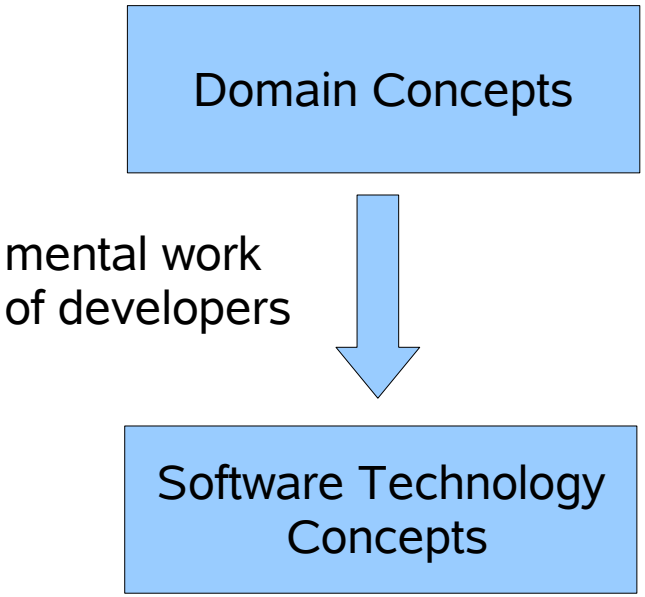
etc.

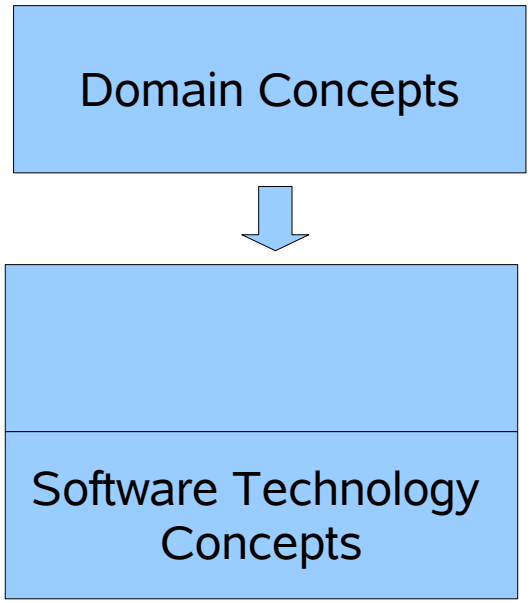
Hochschule Ulm

# Model Driven Software Development
# Introduction and Motivation

## What is Model Driven Software 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



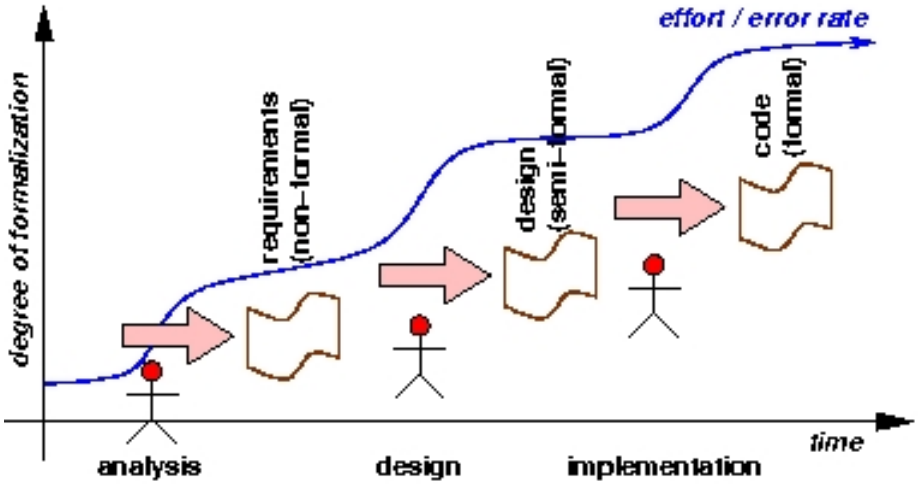



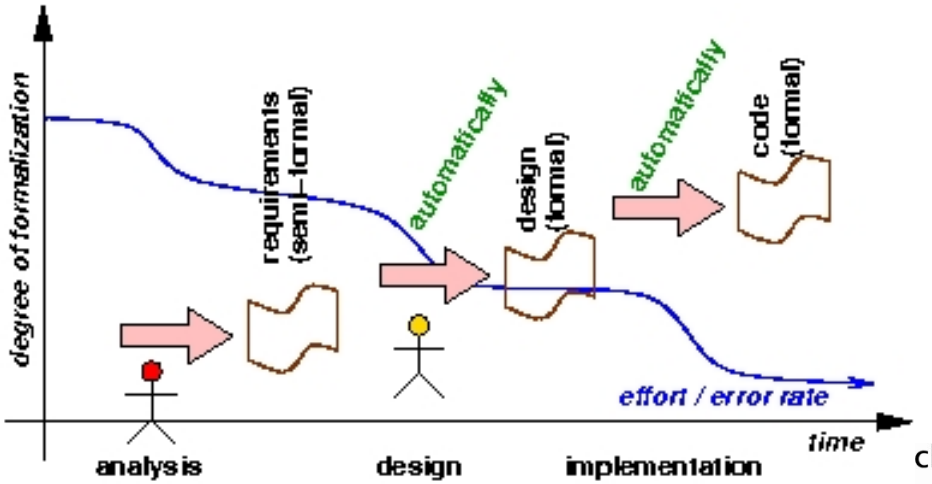

http://www.voelter.de/services/mdsd.html

# Model Driven Software Development
# Introduction and Motivation

## Modelling + Formalization = Solution of all Problems?

- "the earlier the formalization, the more steps can be automated" => is it true?
- ***what is about software architecture and target platform?***
  - need also be available in formalized form to be accessible by transformations!
  - that is exactly what is required by **MDA** in form of **PIM** and **PSM**
  - ***both, the software architecture as well as the platforms are formally described by models***



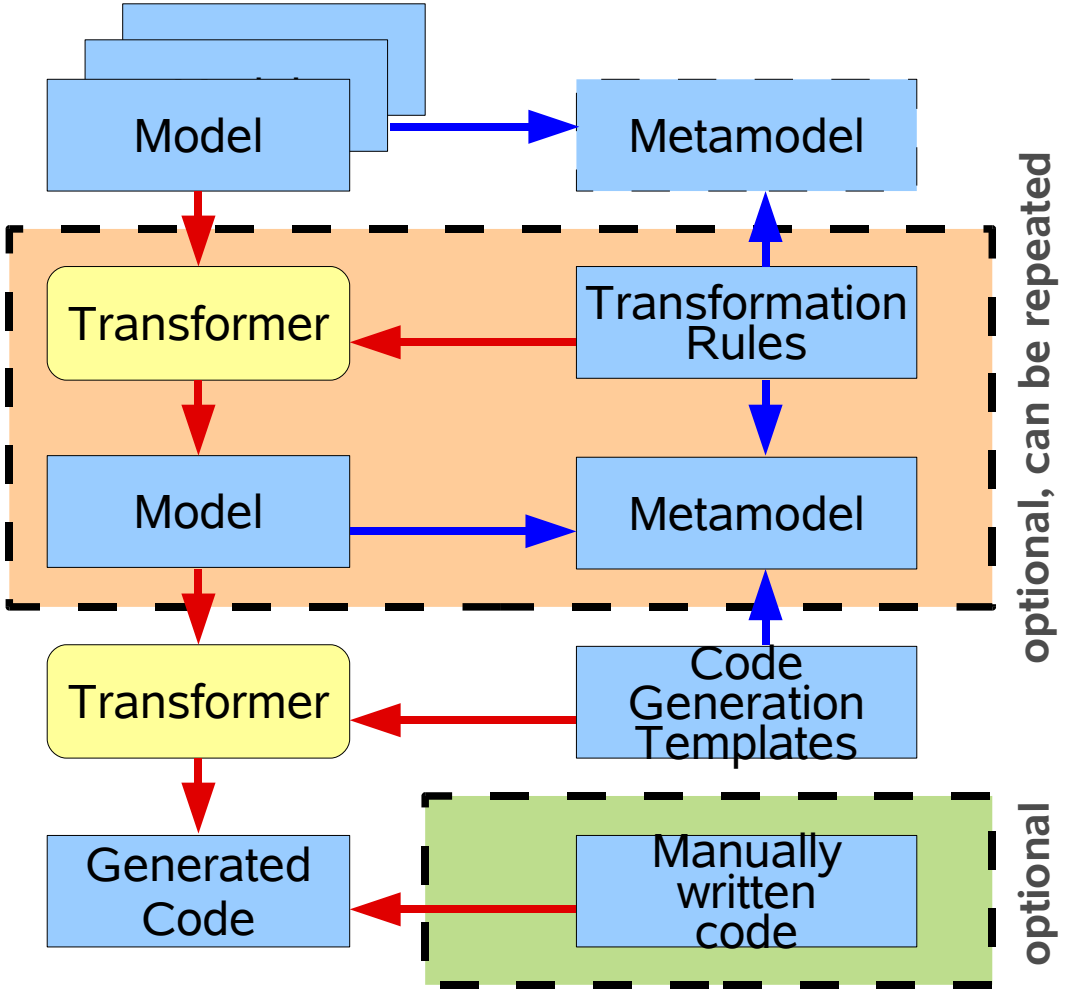




Petrasch, Meinberg: Model Driven Architecture

# Model Driven Software Development
# Introduction and Motivation

## How MDSD works

- Developer develops model(s) based on certain metamodel(s), expressed using a DSL
- Using code generation templates, the model is transformed to executable code
  - alternative: interpretation
- Optionally, the generated code is merged with manually written code
- One or more model-to-model transformation steps may precede code generation




http://www.voelter.de/services/mdsd.html

# Model Driven Software Development
# Introduction and Motivation

## Why is Model Driven Software Development important in robotics?

- Software development is too ***complex*** and too ***expensive***

... because:

- there is ***too little reuse***
- ***technology changes*** faster than developers can learn
- knowledge and practices are ***hardly captured explicitly*** and made available for reuse
- domain experts cannot understand all the ***technology stuff*** involved in software development

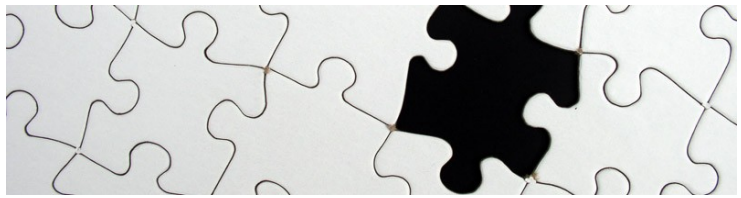
# Model Driven Software Development
# Introduction and Motivation

## Why is Model Driven Software Development important in robotics?

- 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

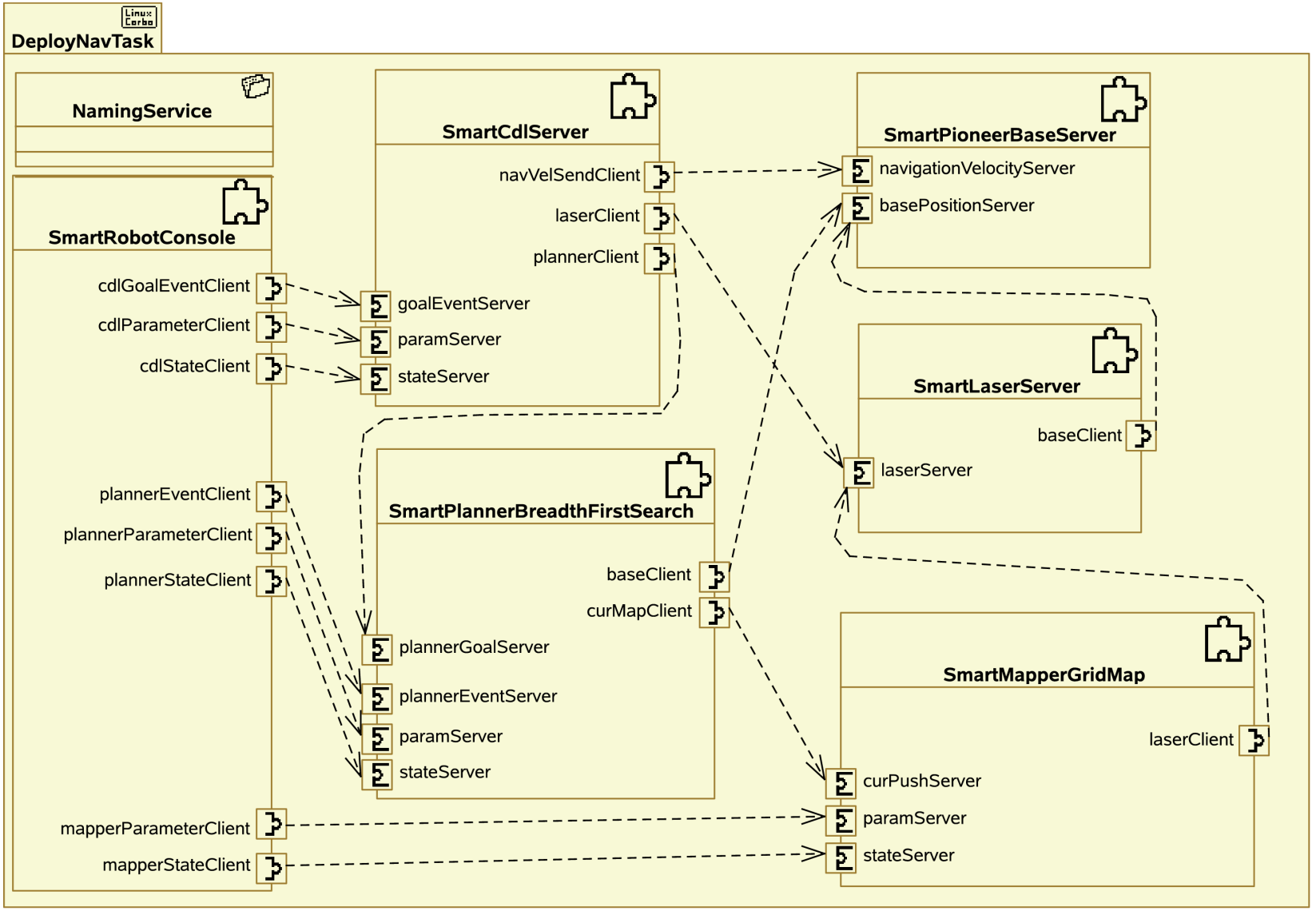
**Engineering the software development process in robotics is one of the basic necessities towards industrial-strength service robotic systems**



Hochschule Ulm

service robotik

# Model Driven Software Development
## Example / Navigation Task





Hochschule Ulm

# Model Driven Software Development
# Idea and Approach

## What is different in robotics?

- ***not*** the huge number of different sensors and actuators
- ***not*** the various hardware platforms
- ***not*** real-time requirements etc.

***but***

- the context and situation dependant reconfiguration of interactions
- a prioritized assignment of restricted resources to activities again depending on context and situation

## vision for the next steps in robotics:

- resource-awareness at all levels to be able to adequately solve tasks given certain resources

# Model Driven Software Development
# Idea and Approach

## That sounds good but give me an example ...

***we made some very simple but pivotal decisions while dealing with component based systems that then proved to pave the way towards MDSD:***

- **granularity level for system composition:**
  - loosely coupled components
  - services provided and required
- **strictly enforced interaction patterns between components**
  - precisely defined semantics of intercomponent interaction
  - these are policies (and can be mapped onto any middleware mechanism)
  - separate component internals from externally visible behavior

  ➔ *independent of a certain middleware*

  ➔ *enforce standardized service contracts between components*
- **minimum component model to support system integration**
  - dynamic wiring of the data flow between components
  - state automaton to allow for orchestration / configuration

  ➔ *ensures composability / system integration*

# Model Driven Software Development
# Idea and Approach

## That sounds good but give me an example ...

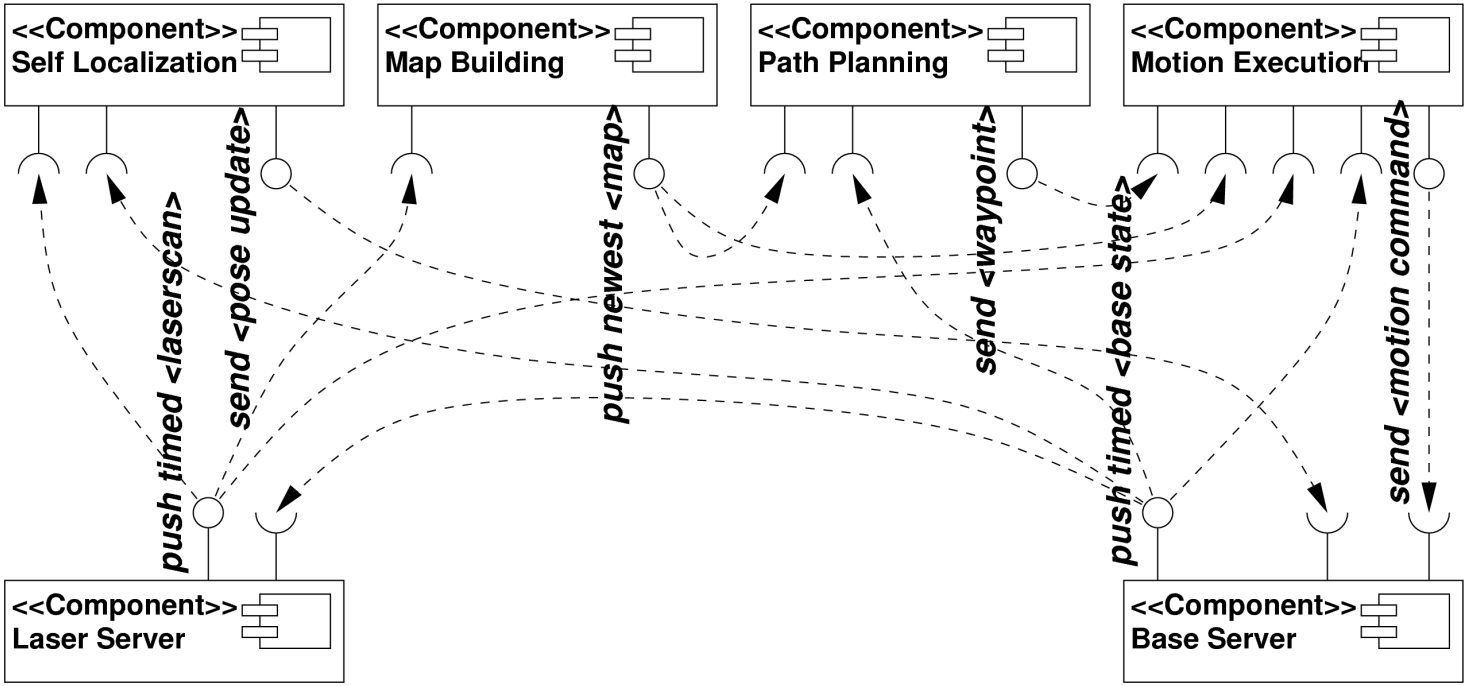
- **execution environment**
  - tasks (periodic, non-periodic, hard real-time, no realtime), synchronization, resource access

  ➔ *components explicitly allocate resources like processing power and communication bandwidth from the underlying HAL*

  ➔ *again, can be mapped onto different operating systems*

- **design policy for component behavior:**
  - principle of locality: a component solely relies on its own resources
  - example: ***QUERY***
    - maximum response time as attribute of service provider
    - client can only *ask* (attribute in request object) for faster response as guaranteed by QoS of service provider
    - server would respond with a VOID answer in case it rejects requested response time
    - it is the client that then decides what is next
  - example: ***PUBLISH/SUBSCRIBE***
    - service provider agrees upon QoS as soon as subscription got accepted
    - again, it is a matter of policy whether this already requires hard guarantees or whether we also accept notifications about not being able to hold the schedule

# Model Driven Software Development
# Idea and Approach




| | |
|---|---|
| send | CHS::SendClient, CHS::SendServer, CHS::SendServerHandler |
| query | CHS::QueryClient, CHS::QueryServer, CHS::QueryServerHandler |
| push newest | CHS::PushNewestClient, CHS::PushNewestServer |
| push timed | CHS::PushTimedClient, CHS::PushTimedServer, CHS::PushTimedHandler |
| event | CHS::EventClient, CHS::EventHandler, CHS::EventServer, CHS::EventTestHandler |
| wiring | CHS::WiringMaster, CHS::WiringSlave |

Hochschule Ulm

# 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]*

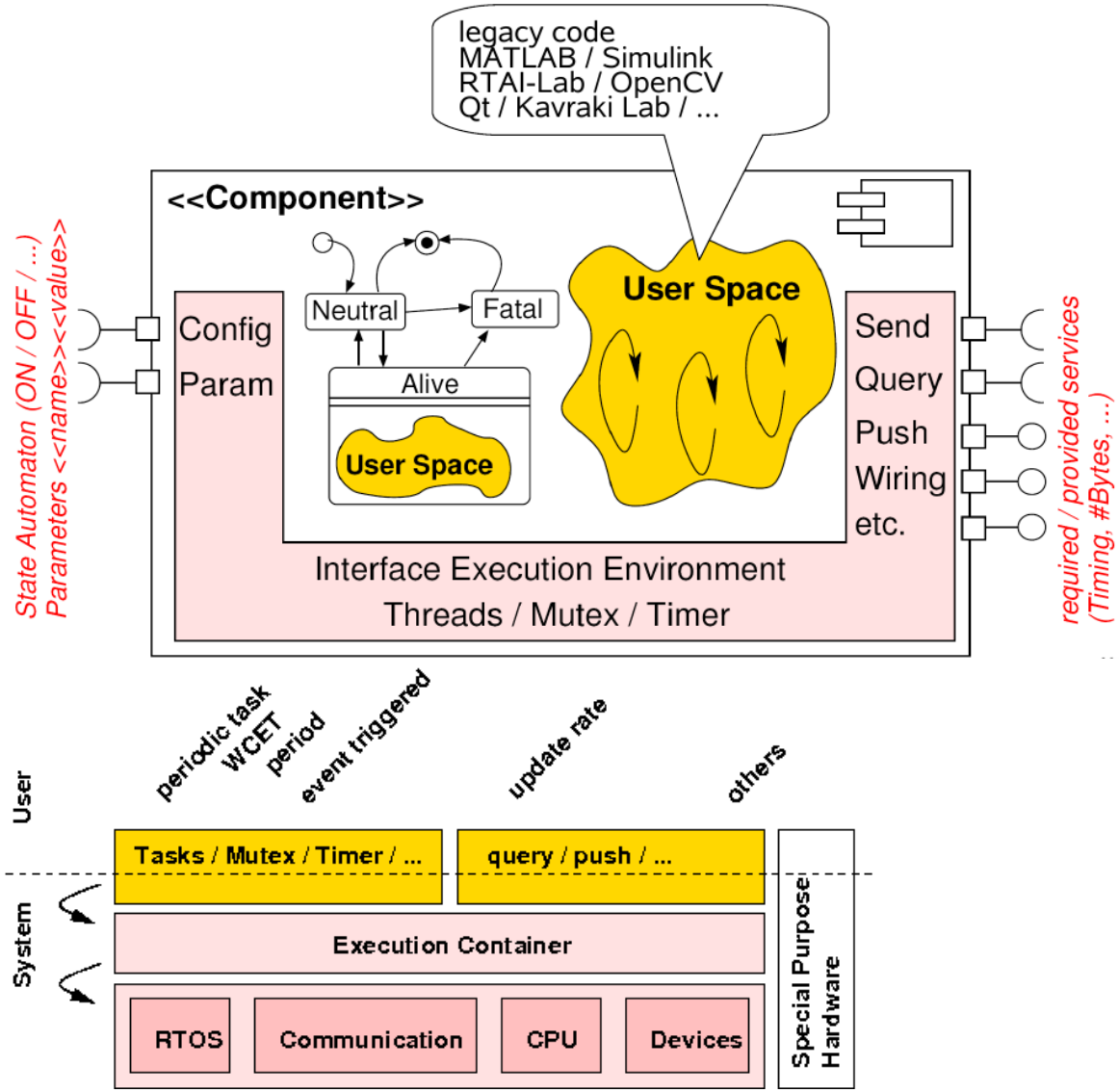
+ add(:WiringSlave*, port:const string&) : StatusCode throw()
+ remove() : StatusCode throw()

+ 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
# Idea and Approach

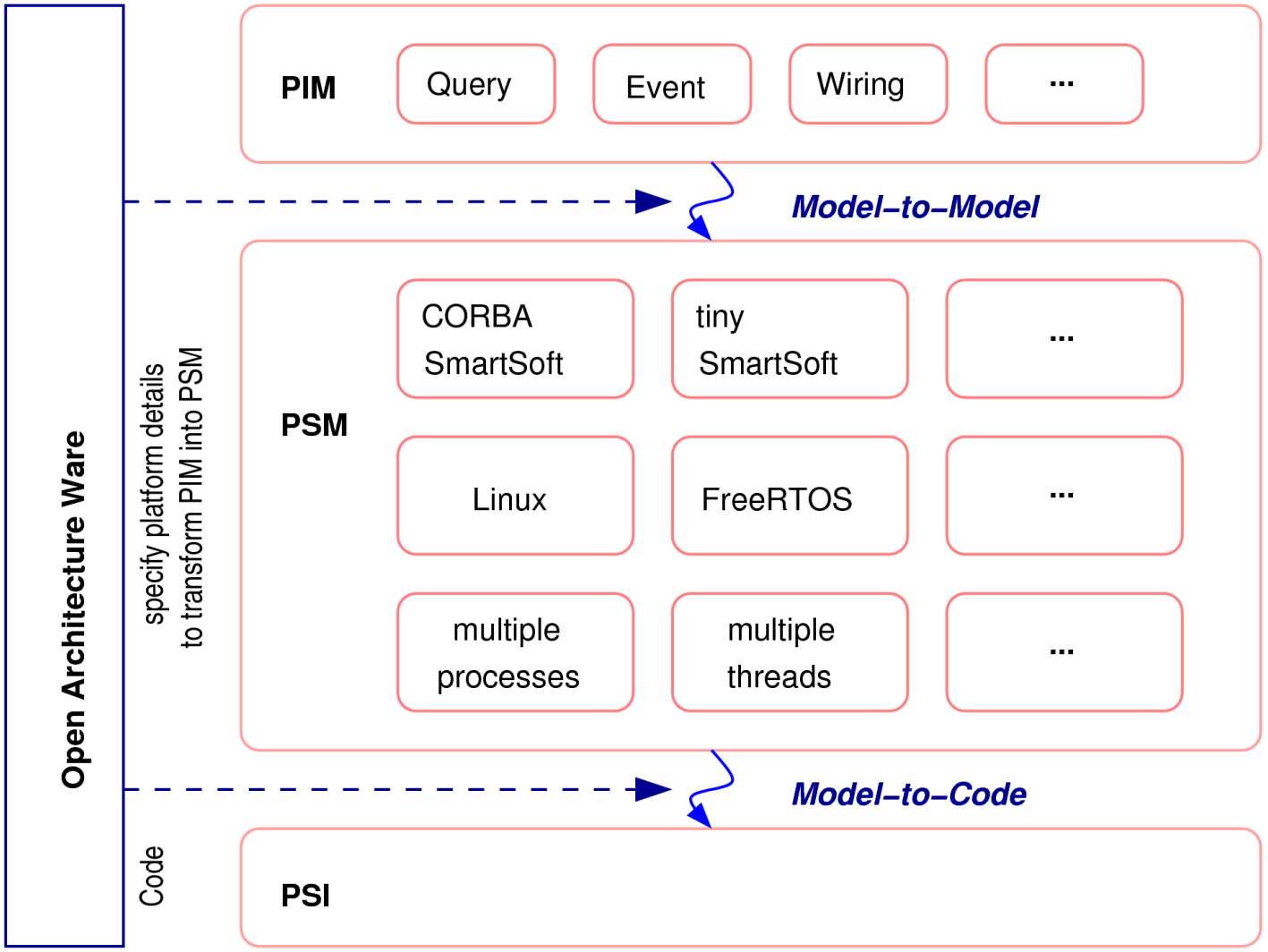



- Communication Objects
  - marshalling
- Communication Patterns
  - downward interface / interal
    - invisible to user
    - is handled by MDSD toolchain
    - can be mapped onto different middleware systems
      - » ACE Reactor / Acceptor etc.
      - » CORBA
      - » 0MQ message system
      - » global variables
  - upward interface / user
    - no adjustments at user visible API
- Tasks etc.
  - obviously, no guarantees when mapped onto no-realtime systems etc.



Hochschule Ulm

# Model Driven Software Development
# SmartMDSD



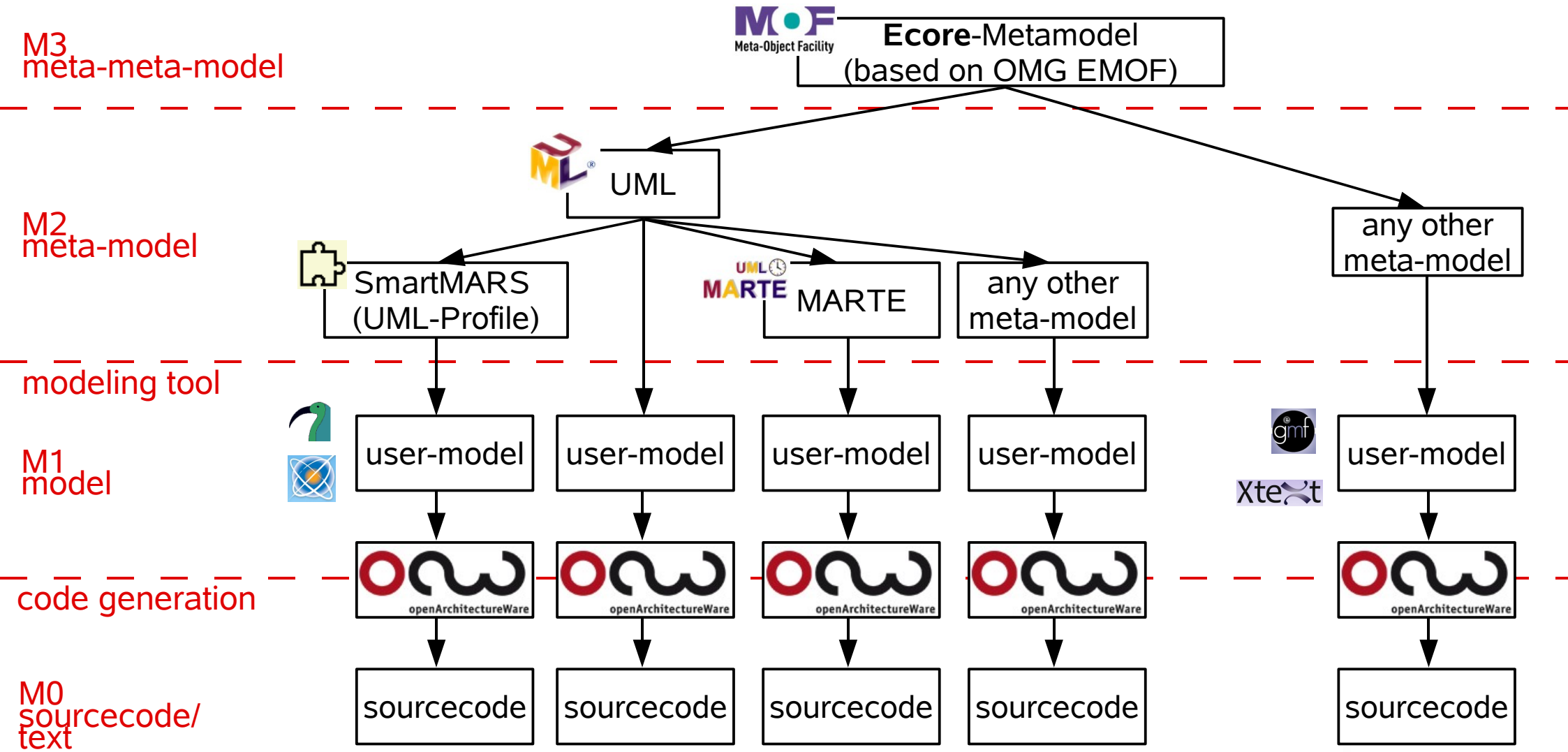
# Model Driven Software Development SmartMDSD


## Benefits of this 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
- SmartSoft provides the perfect granularity for system design, component development, composability, freedom within components, tool support etc.

Hochschule Ulm

# MDSD
## / EMF /


M3
meta-meta-model
Ecore-Metamodel
(based on OMG EMOF)
M2
meta-model
UML
SmartMARS
(UML-Profile)
MARTE
any other
meta-model
any other
meta-model
modeling tool
M1
model
user-model
user-model
user-model
user-model
user-model
code generation
openArchitectureWare
openArchitectureWare
openArchitectureWare
openArchitectureWare
openArchitectureWare
M0
sourcecode/
text
sourcecode
sourcecode
sourcecode
sourcecode
sourcecode

# Model Driven Software Development
## / MDSD basics /

### MDSD core building blocks:

- domain analysis
- meta modelling
- model-driven generation
  (and: model transformations, model-to-model, model-to-text)
- template languages
- domain-driven framework design

### Are MDSD models the same as requirements / analysis models?

- they can be, but in general, they are not
- analysis / requirements models are ***non-computational***, MDSD models are ***computational***
- formalizing requirements is beneficial since requirements become unambigious
- ***MDSD models are no "paperwork"***, they ***are*** the solution which is translated into code automatically

http://www.voelter.de/services/mdsd.html

# Model Driven Software Development
## / MDSD basics /

**Three basic viewpoints:**

- ***Type Model***: Components, Interfaces, Data Types
- ***Composition Model***: Instances, „Wirings“
- ***System Model***: Nodes, Channels, Deployments

**Generated stuff:**

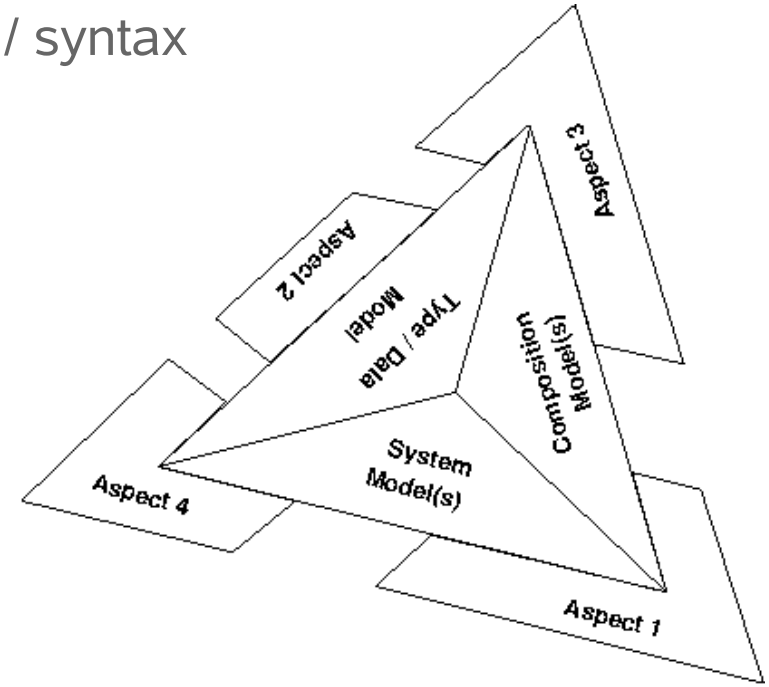
- base classes for component implementation
- build scripts
- descriptors
- remoting infrastructure
- persistence
- etc.

http://www.voelter.de/services/mdsd.html

# Model Driven Software Development
## / MDSD basics /

**Aspect models:**

- often, the described three viewpoints are not enough, ***additional aspects*** need to be described
- these go into ***separate aspect models***, each describing a well-defined aspect of the system
  - each of them uses a suitable DSL / syntax
  - the generator acts as a weaver
- Typical ***examples*** are
  - persistence
  - security
  - timing, QoS in general
  - packaging and deployment
  - diagnostics and monitoring

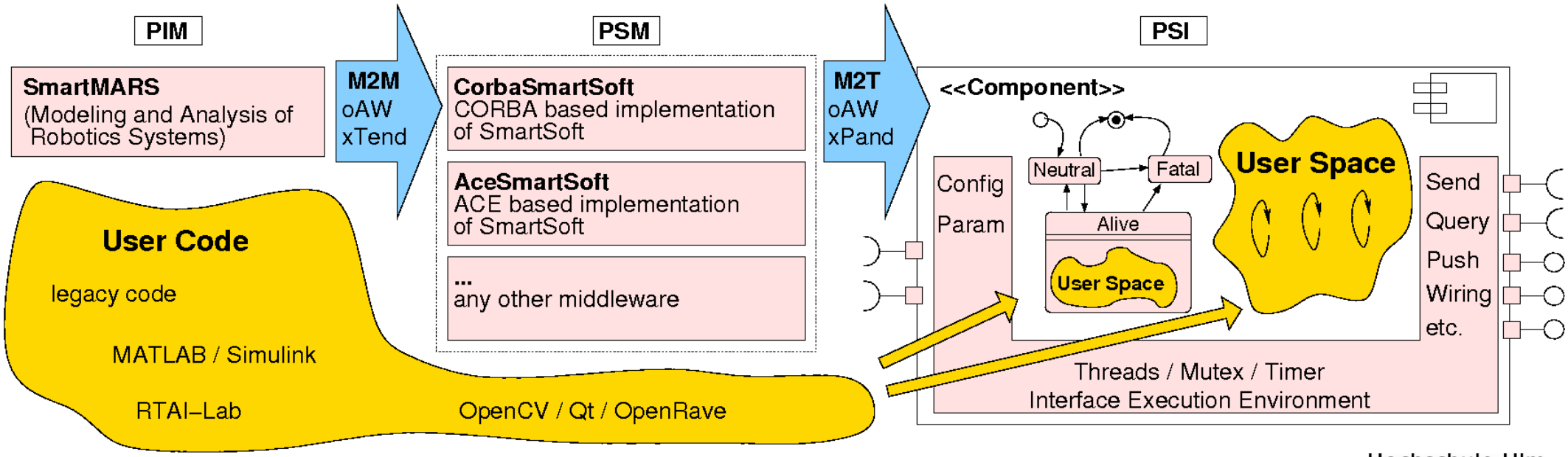



http://www.voelter.de/services/mdsd.html

# 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

**PIM**

**SmartMARS** – Metamodel

(**M**odeling and **A**nalysis of **R**obotics **S**ystems)

**M2M**
oAW
xTend
check

**PSM**

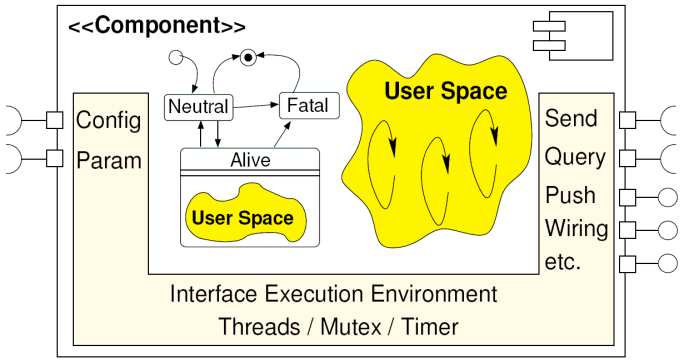
**CorbaSmartSoft**
CORBA based implementation of SmartSoft

**AceSmartSoft**
ACE based implementation of SmartSoft

**...**
any other middleware

**M2T**
oAW
xPand
check

**PSI**




- UML2-Profile
- platform independent stereotypes
  - SmartComponent
  - SmartTask
  - SmartMutex
  - SmartQueryServer
  - SmartEventClient
  - ...

- UML2-Profile
- platform specific stereotypes

has to be created by a middleware expert

The user space can contain arbitrary code and libraries

The user space remains the same independent of the different platform specific models

Just the component hull will be created

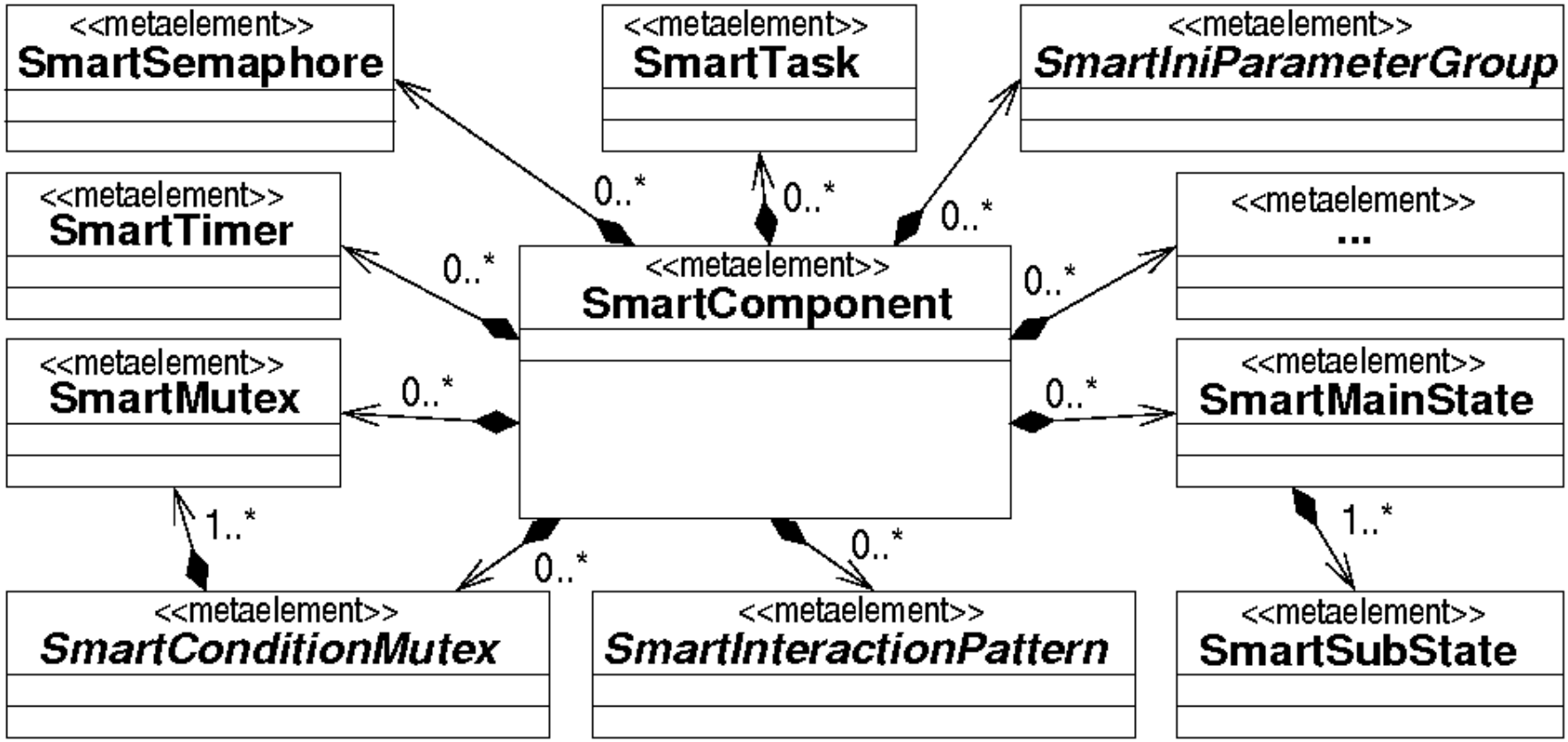
Hochschule Ulm

# Model Driven Software Development
# Idea and Approach

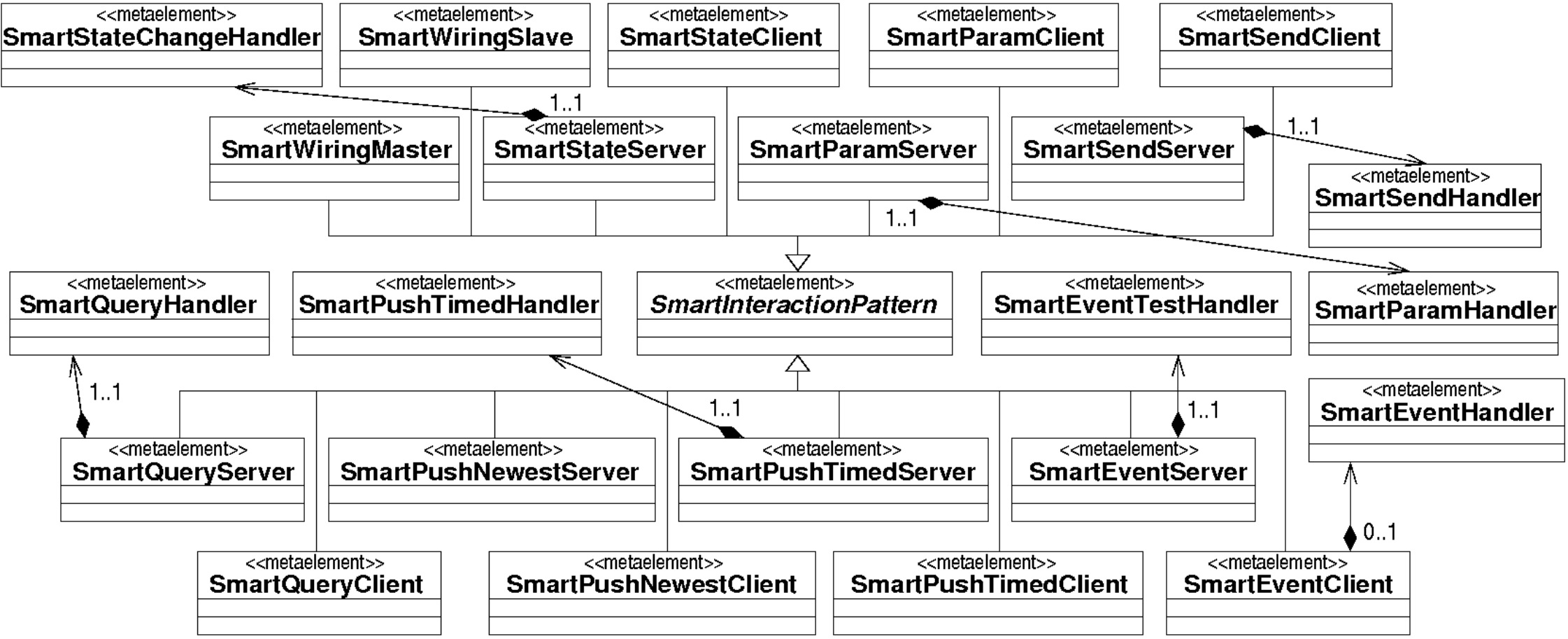
## What do we need within a component meta-model?

- Interaction Patterns
  - loosely coupled communication
  - independent of middleware
  - accessible to MDSD
- Parameterization and configuration ports
  - name / value pairs
  - dynamic wiring
  - reflection ?
- Abstraction of execution container
  - resource access via abstraction independent of implementational technology and OS
  - Tasks, Semaphore, CV, PCP, etc.
  - accessible to MDSD
- State automaton inside a component
  - share common states to support orchestration
  - allow for individual substates beneath basic state automaton
- and many others
  - authorization, encryption, etc.

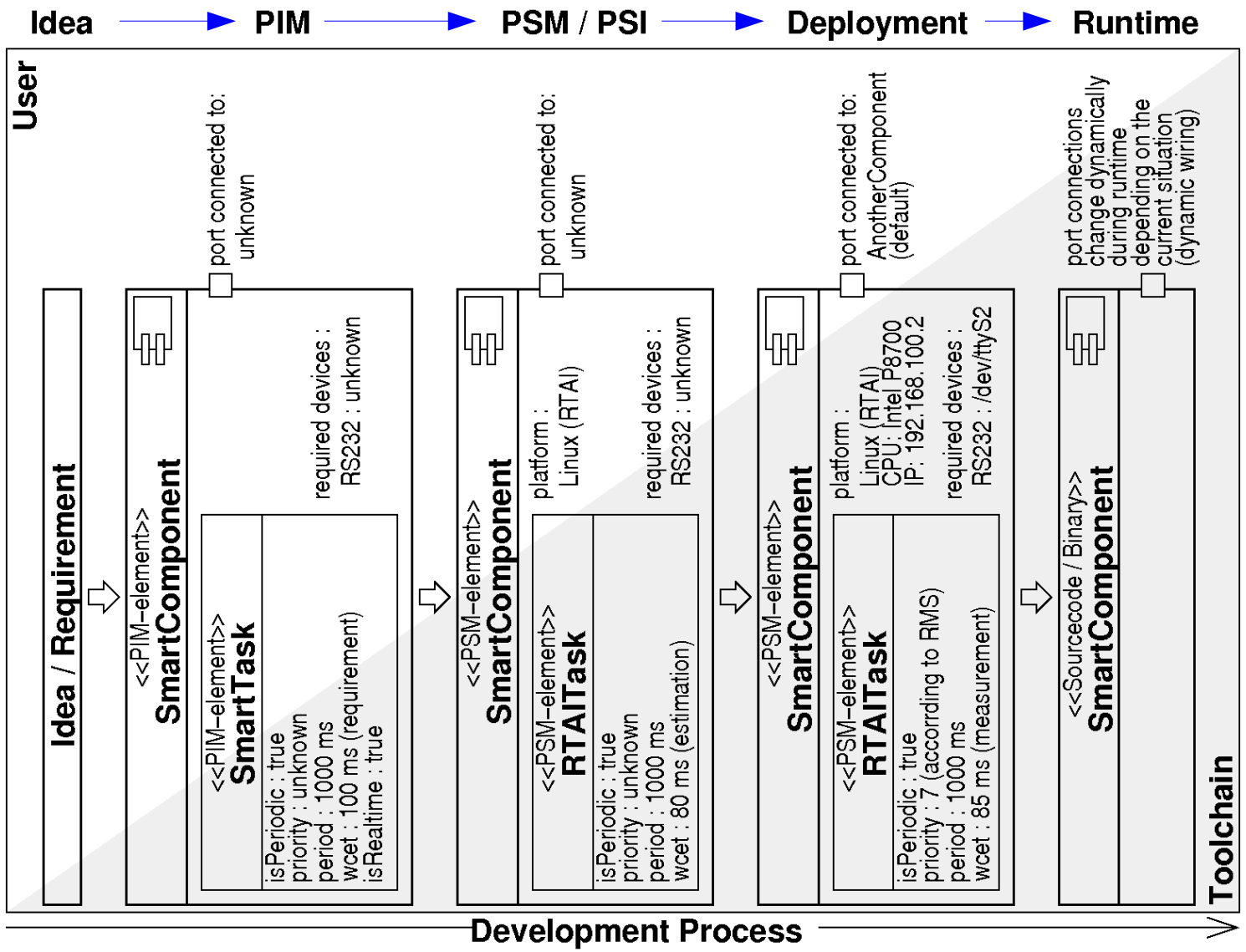
# Model Driven Software Development
# Metamodels (partial view)



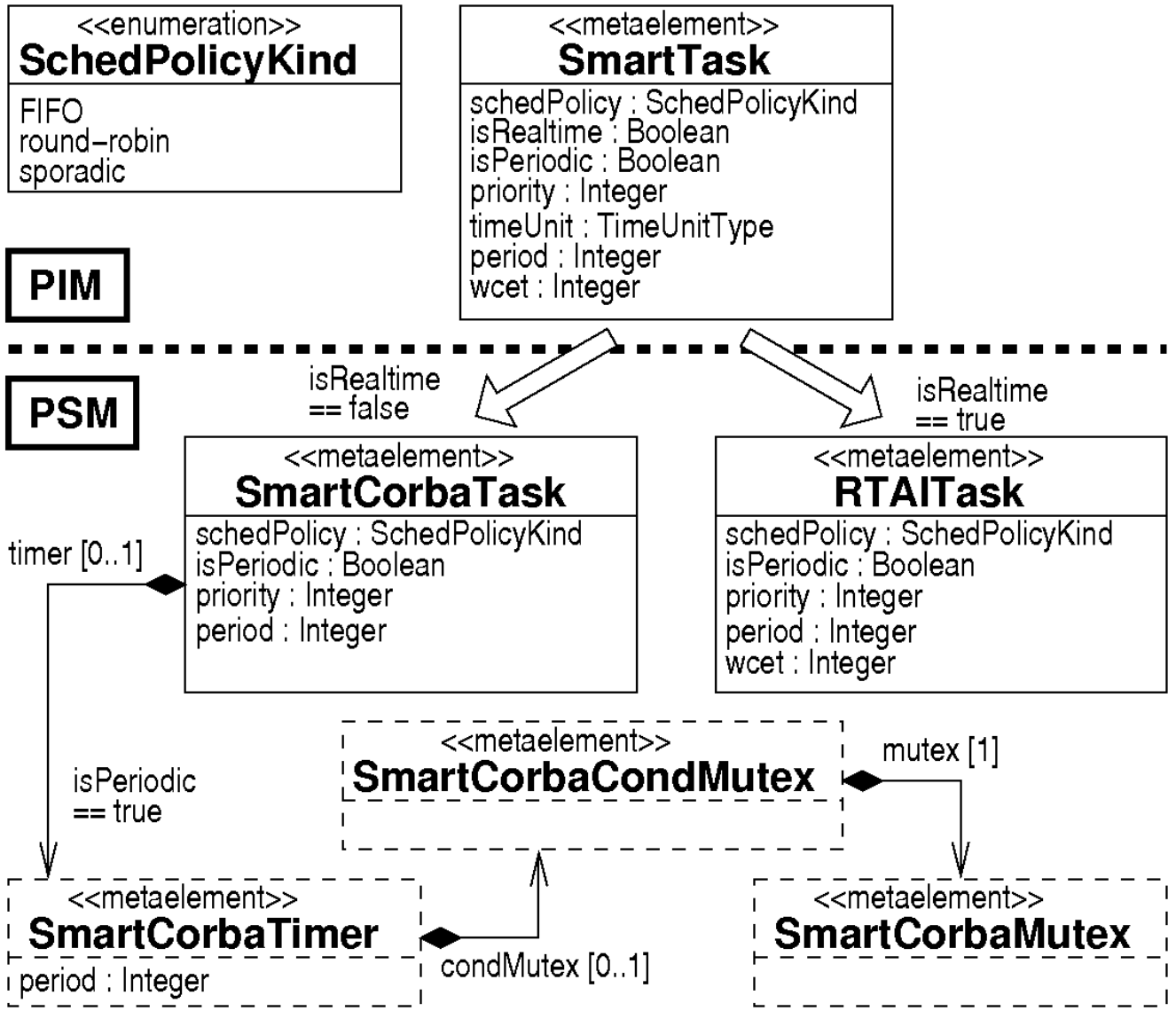
# Model Driven Software Development
# Metamodels (partial view)



# Model Driven Software Development SmartMDSD



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



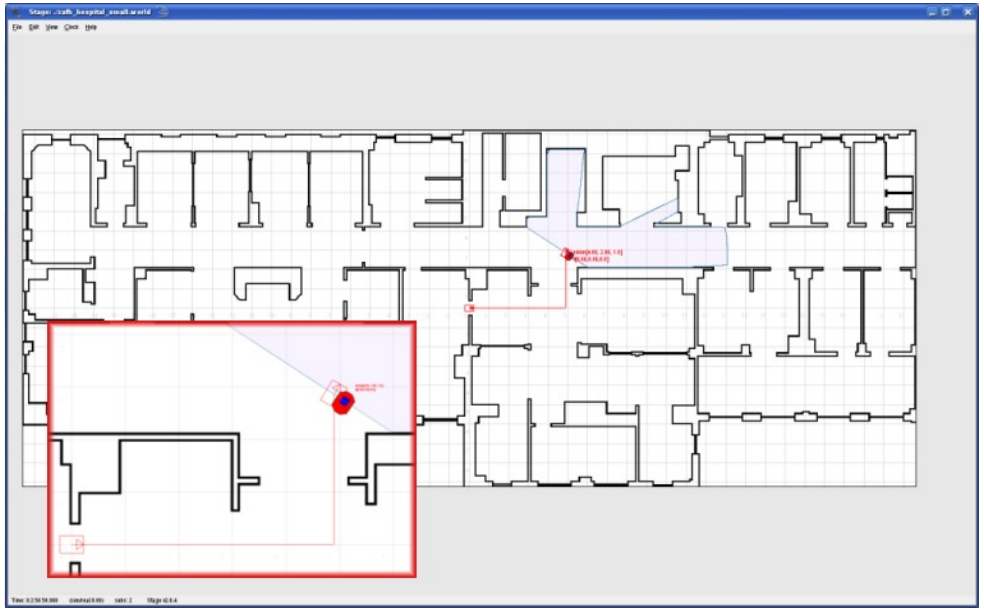
# Model Driven Software Development
# Extensions towards QoS / real-time

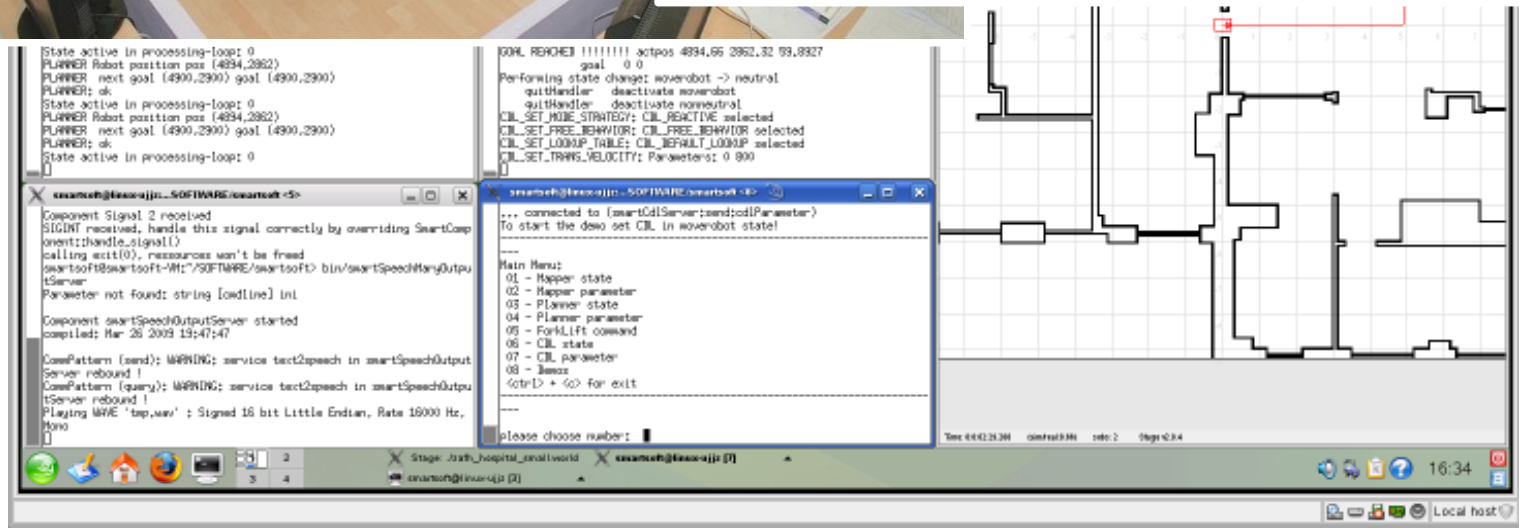
- already available
  - have timing parameters within communication objects as part of request to server
  - server can then response with void answer in case it cannot meet the deadline
  - interface to CHEDDAR timing analysis for RMA / dependent on PSM
- next step
  - timeout-parameters at user-interface of interaction methods
    - 0 infinity / no timeout
    - others timeout
    - *since interaction patterns are standalone entities, these timings are easily implemented locally without server interaction (see principle of locality)*
    - *have these parameters within UML component model*
- next step
  - at deployment of components
    - map ports (and their messages) onto hard real-time communication systems where needed (like Realtime-Ethernet)
  - extend this towards general resource awareness

***incrementally extend Meta-Models to cover more and more aspects of robotics***

service robotik

# Model Driven Software Development
# Examples etc.

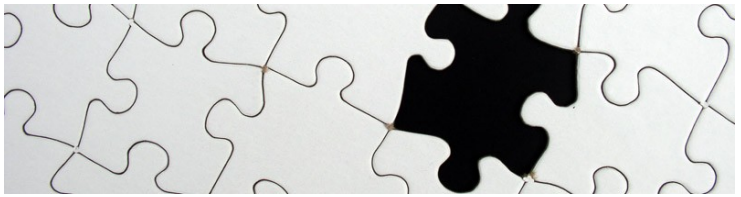






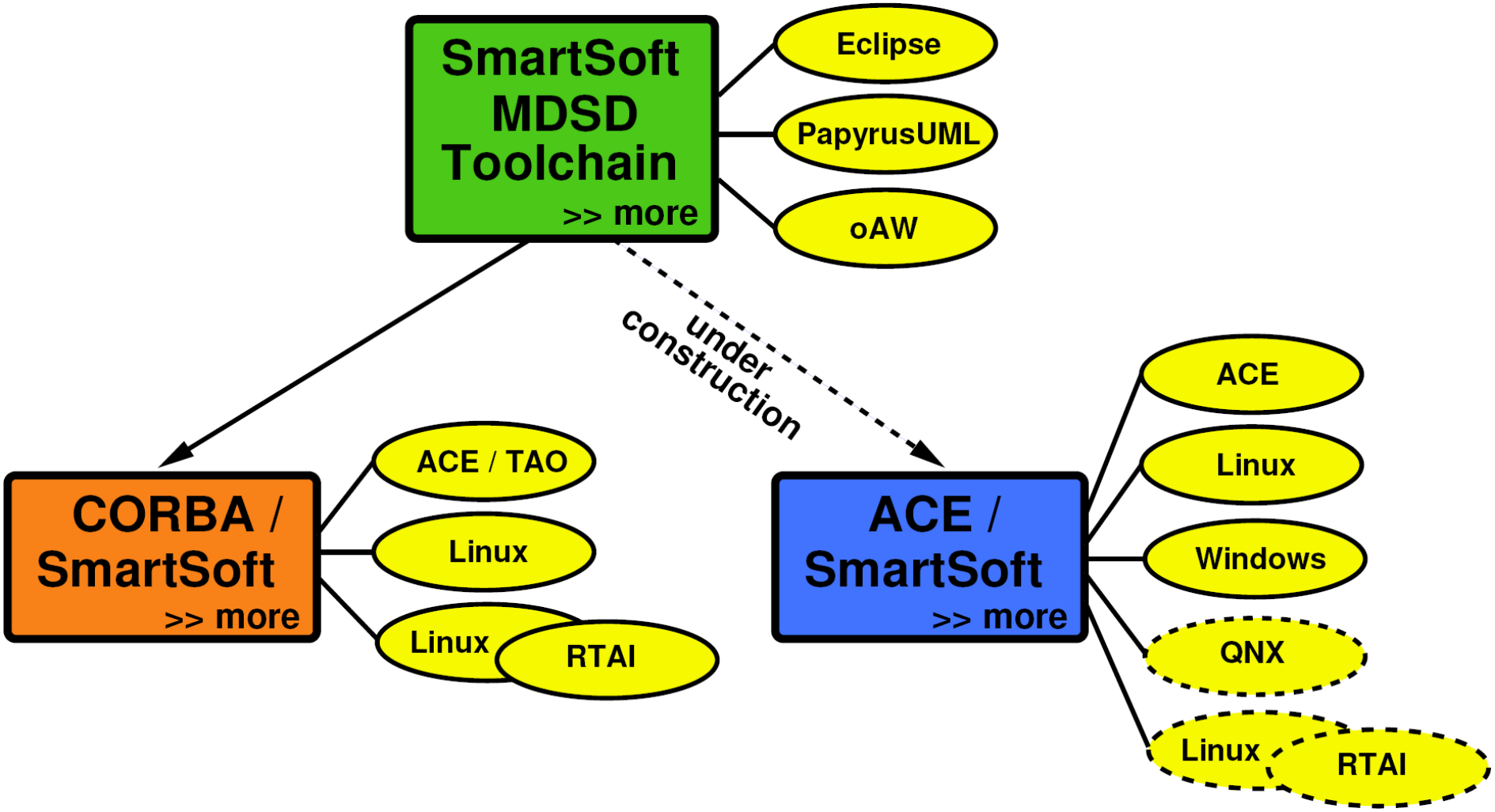
Hochschule Ulm

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



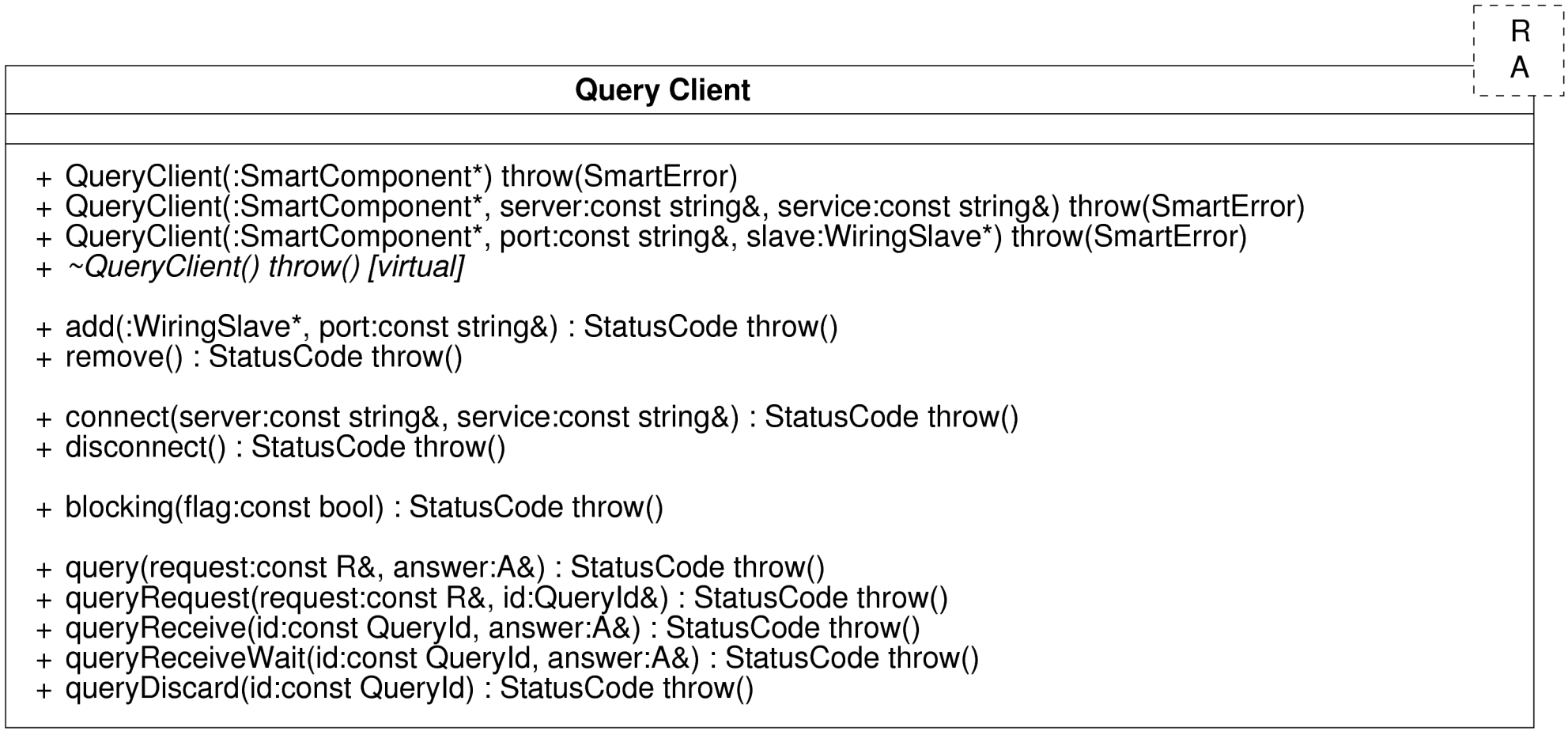
- Toolchain based on Open Architecture Ware
  - fully integrated into Eclipse
  - **http://www.openarchitectureware.org/**
- MDSD Toolchain Example
  - PIM: SmartMARS robotics profile (Modeling and Analysis of Robotics Systems)
  - PSM: SmartSoft in different implementations but with the same semantics !
  - 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
  - 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

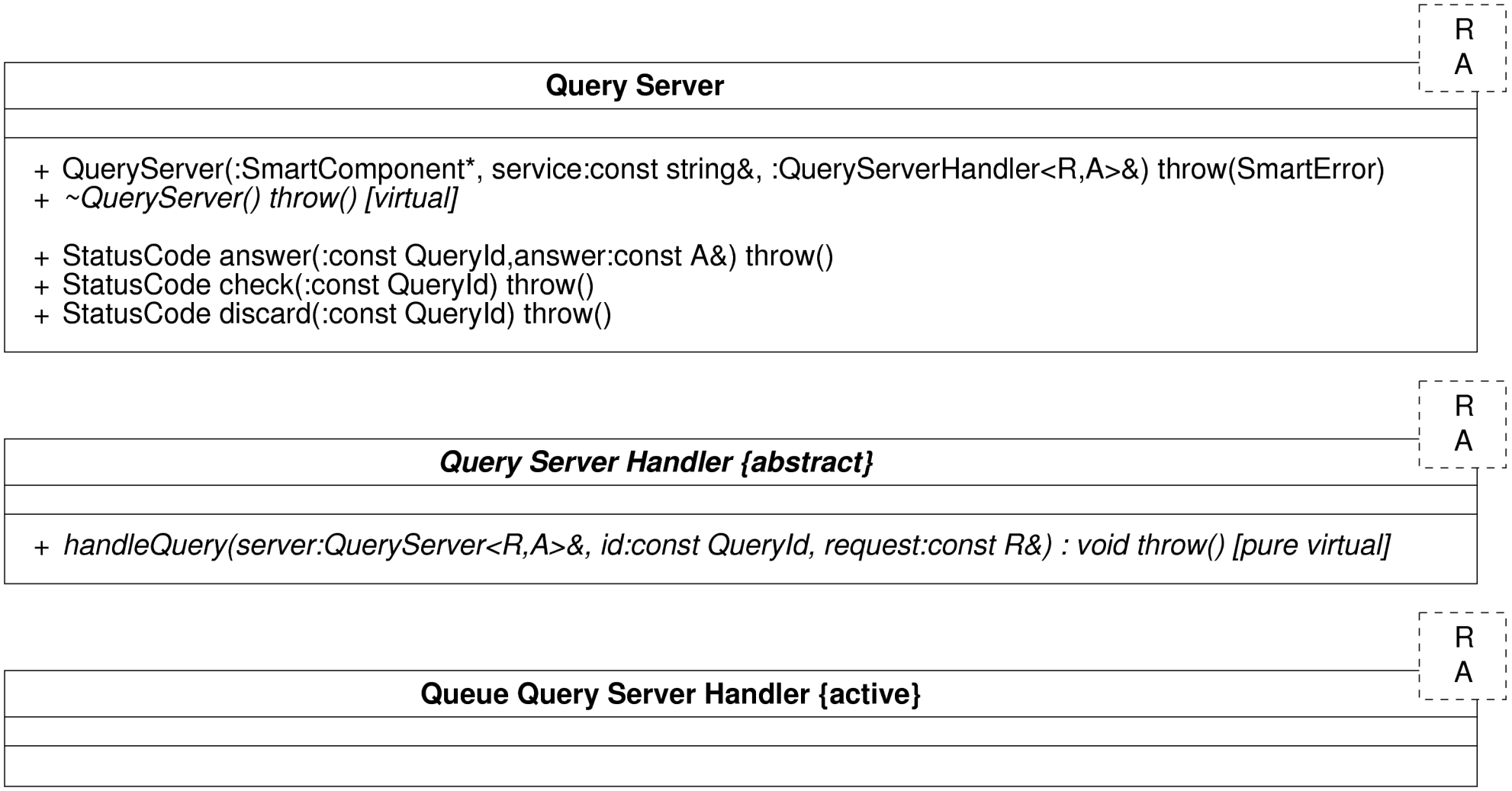


# Addendum

# Model Driven Software Development
## / Technical Details /



# Model Driven Software Development
## / Technical Details /

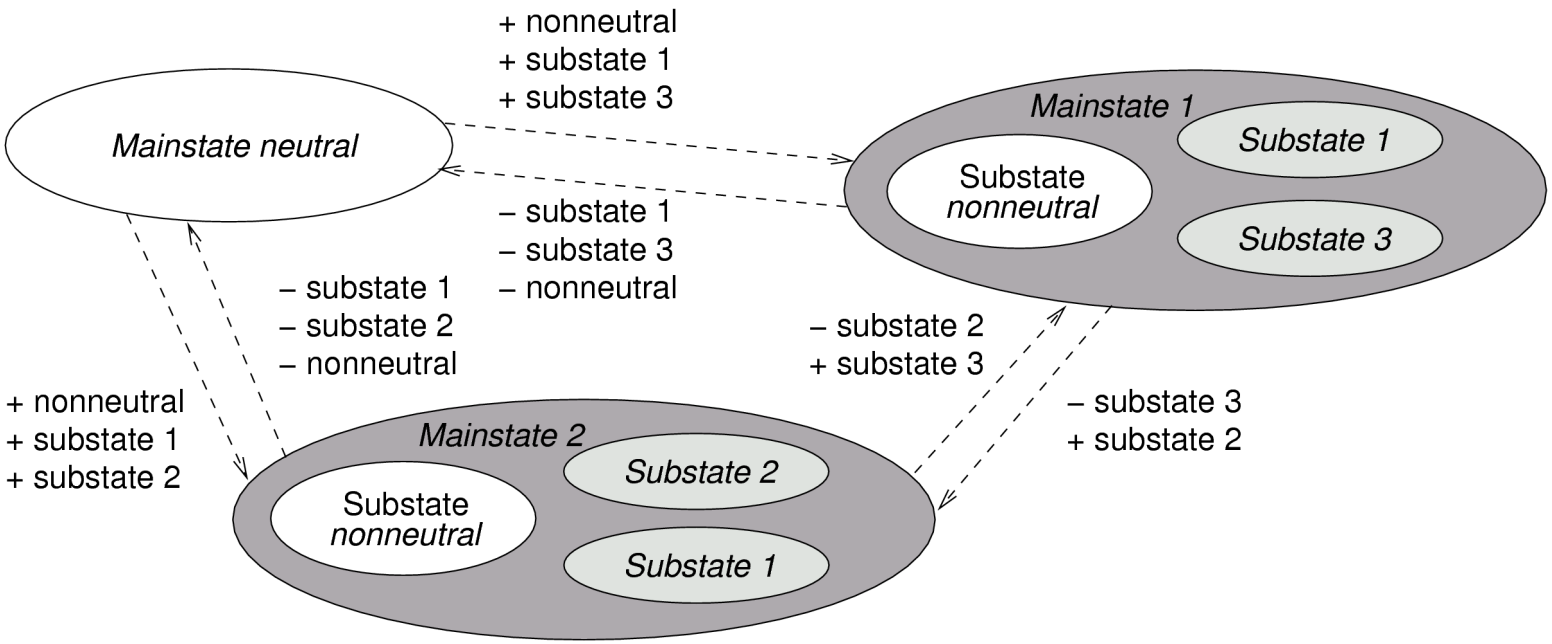


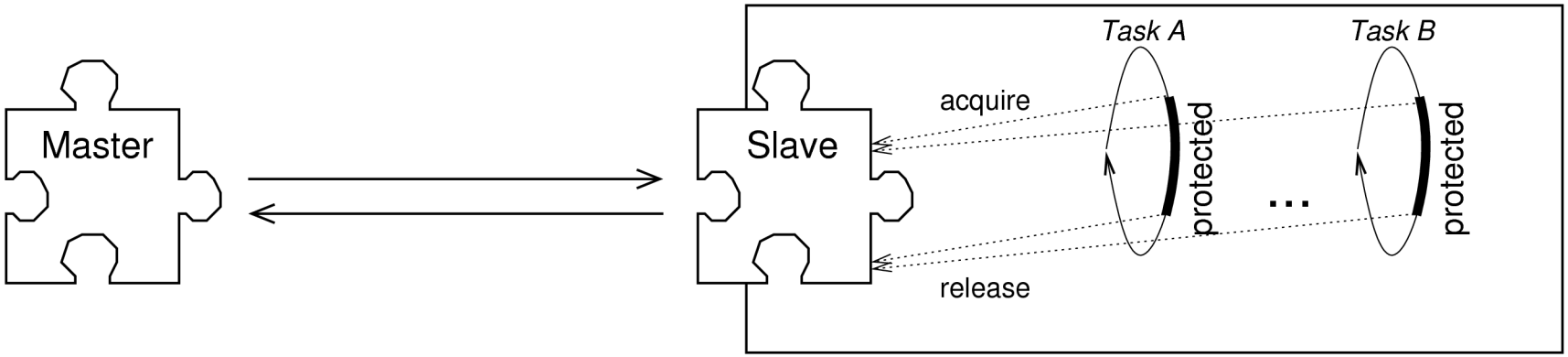
# Model Driven Software Development / Technical Details /

| Wiring Master |
| --- |
| |
| + WiringMaster(:SmartComponent*) throw(SmartError)<br>+ *~WiringMaster() throw() [virtual]*<br><br>+ blocking(flag:const bool) : StatusCode throw()<br><br>+ connect(slavecmpt:const string&, slaveprt:const string&, servercmpt:const string&, serversvc:const string&) : StatusCode throw()<br>+ disconnect(slavecmpt:const string&, slaveprt:const string&) : StatusCode throw() |

| Wiring Slave |
| --- |
| |
| + WiringSlave(:SmartComponent*) throw(SmartError)<br>+ *~WiringSlave() throw() [virtual]* |

# Model Driven Software Development
## / Technical Details /



**State Master**

+ StateMaster(:SmartComponent*) throw(SmartError)
+ StateMaster(:SmartComponent*, server:const string&, service:const string&) throw(SmartError)
+ StateMaster(:SmartComponent*, port:const string&, :WiringSlave*) throw(SmartError)
+ *~StateMaster() 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()

+ setStateWait(state:const string&) : StatusCode throw()
+ getCurrentStateWait(state:string&) : StatusCode throw()
+ getMainStatesWait(states:list<string>&) : StatusCode throw()
+ getSubStatesWait(mainstate:const string&,states:list<string>&) : StatusCode throw()

---

**State Slave**

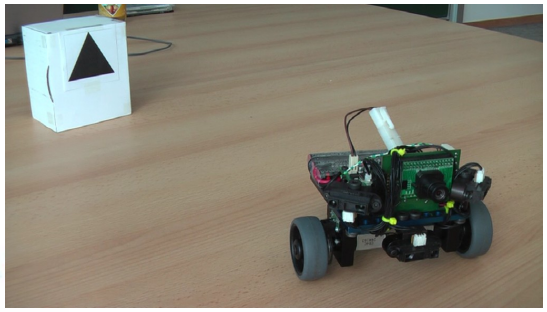
+ StateSlave(:SmartComponent*, service:const string&, :StateChangeHandler&) throw(SmartError)
+ *~StateSlave() throw() [virtual]*

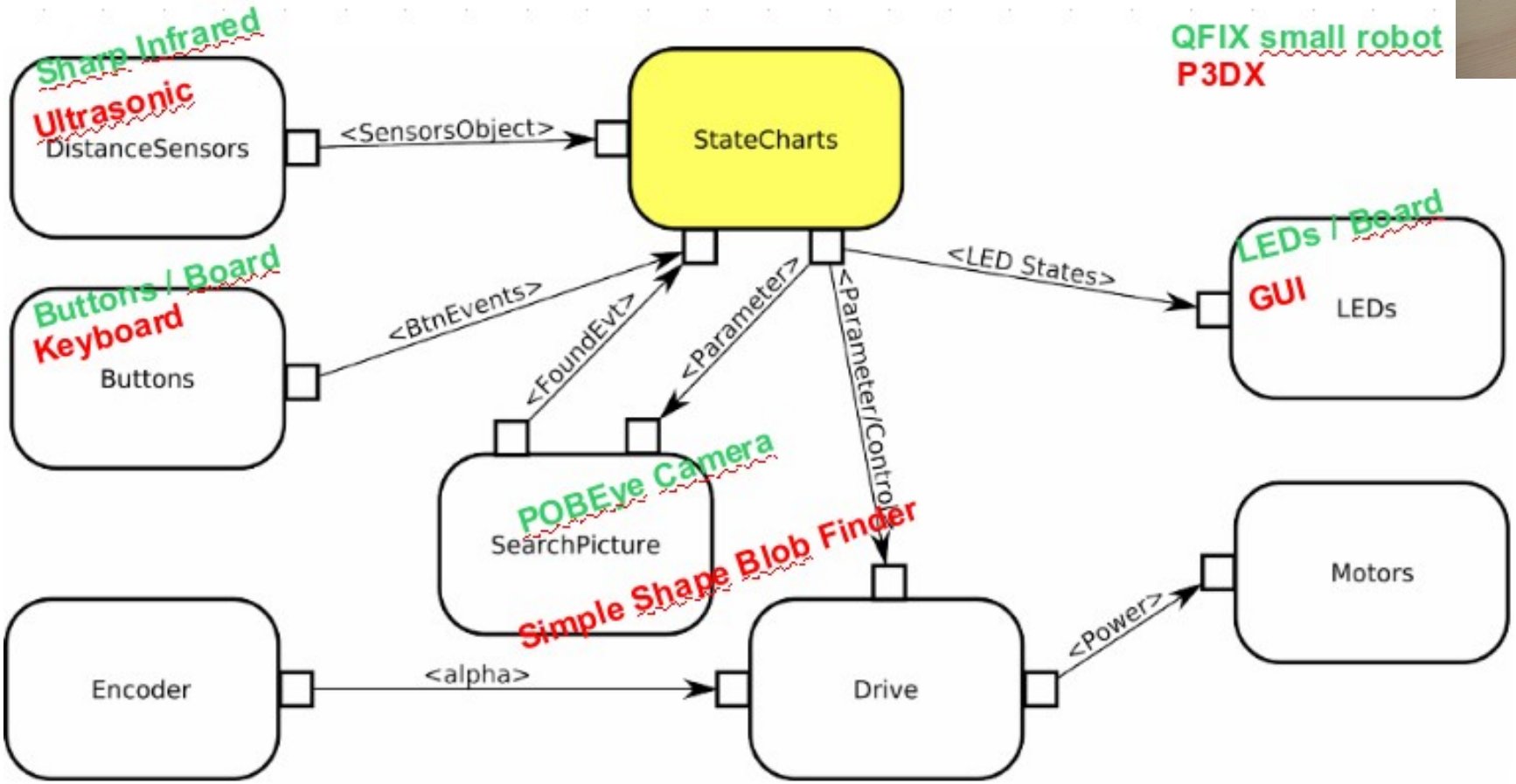
+ defineStates(mainstate:const string&, substate:const string&) : StatusCode throw()
+ activate() : StatusCode throw()
+ acquire(substate:const string&) : StatusCode throw()
+ tryAcquire(substate:const string&) : StatusCode throw()
+ release(substate:const string&) : StatusCode throw()

***State Change Handler {abstract}***

+ *handleEnterState(state:const string&) : void throw() [pure virtual]*
+ *handleQuitState(state:const string&) : void throw() [pure virtual]*

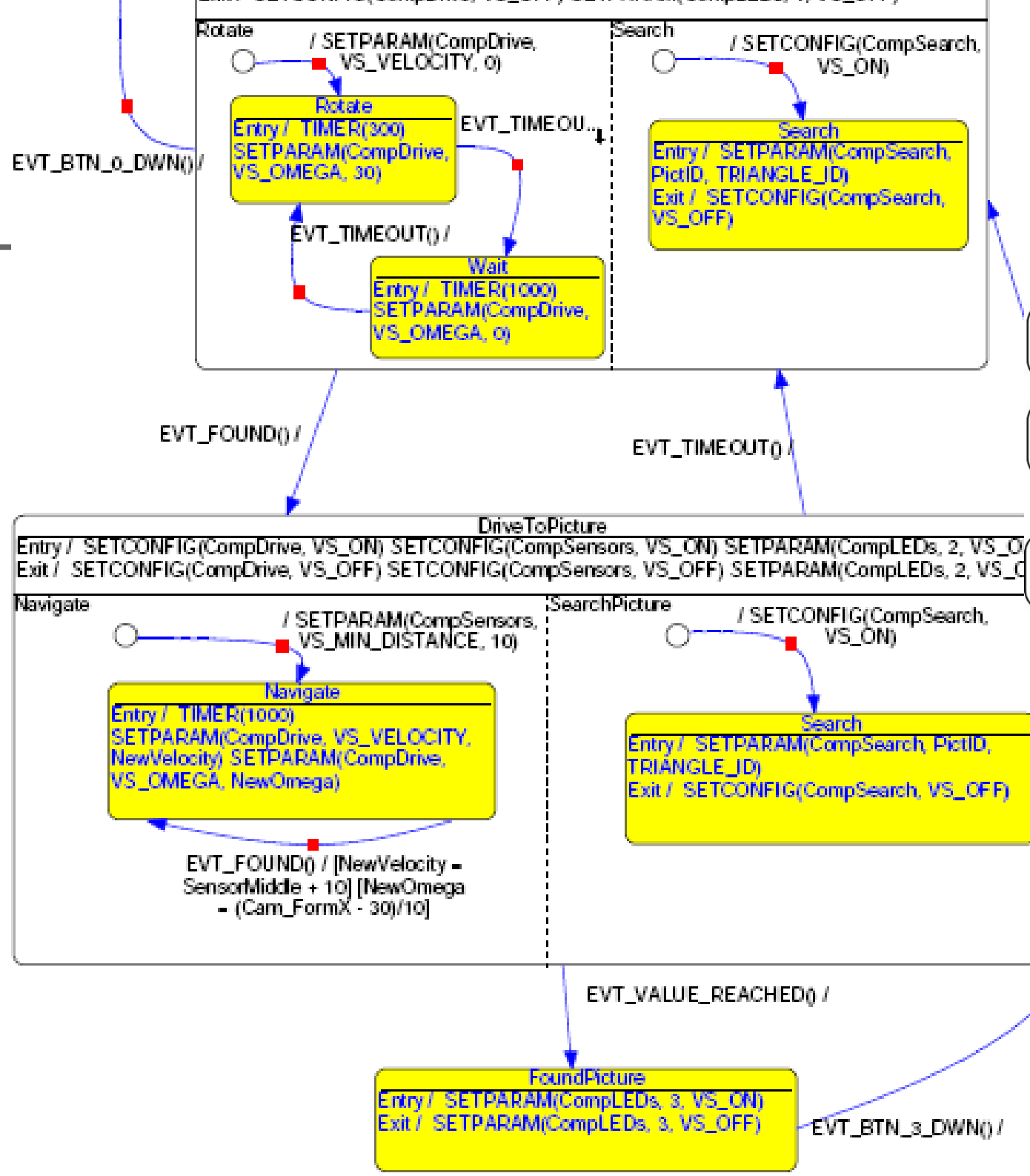
# Model Driven Software Development Example

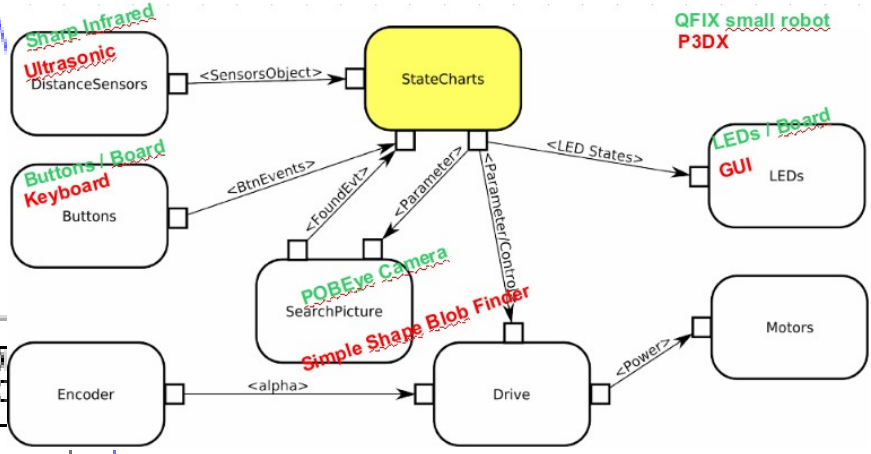





Hochschule Ulm

# elopment




Hochschule Ulm

# Model Driven Software Development
# Idea and Approach