Software Development Workflow in Robotics

Alois Knoll

Simon Barner, Michael Geisinger, Markus Rickert

Robotics and Embedded Systems

Department of Informatics

Technische Universität München

ICRA 2009 Workshop

Open Source Software in Robotics

Underlying Observations

- Robots are one of the most complex class of technological products we have developed – they include all kinds and large numbers of
  - diverse actuators, sensors, controllers, communication systems;
  - at different time-scales and with different needs for bandwidth to result in systems;
  - with highest requirements of safety, reliability, availability, fault-tolerance
  - for a large variety of behaviours – typically in uncertain/unknown environments.
- Integrating all these components to a given specification is extremely demanding and would not only justify but necessitate the use of the most advanced software engineering tools there are
- **Yet**, this system integration challenge is almost completely ignored by roboticists, and we program our robots almost from scratch – this approach has not future!
- Systems typically get stuck at "lab sample stage": they are brittle, error-prone, and never robust …

What would be necessary …

- **Re-usability** of components (to compose the robot system starting from appropriate levels of abstraction)

- Simple **configuration** of those components – instead of low-level programming

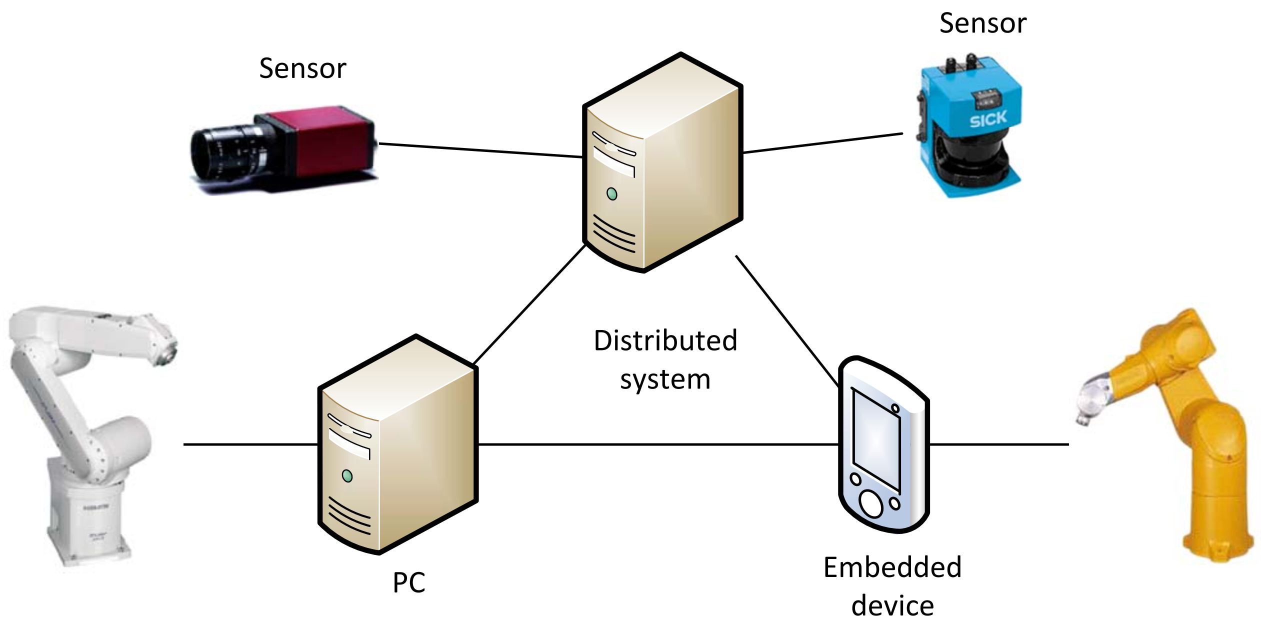
- Possibility for **specification of the desired behavior** at the highest levels instead of "wiring in" pre-defined schemata at various levels of the software architecture

… and how this could be approached:

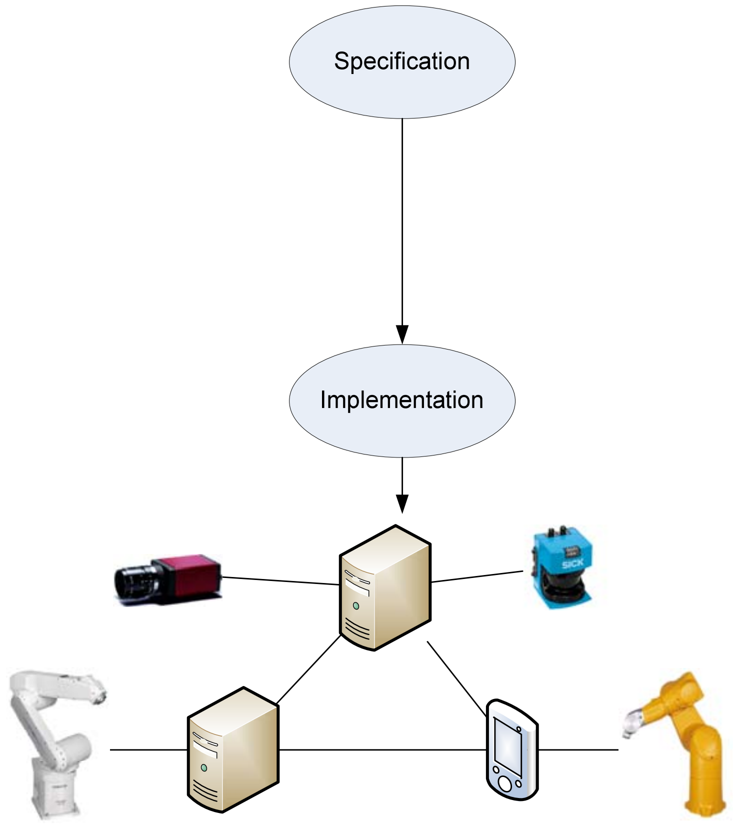
- Apply state-of-the art SW-engineering methodology
- Use state-of-the art tools and tool chains $\rightarrow$ there are lots of tools for different purposes available in the open domain
- Provide a “meta-architecture” for the software design process – so as to allow competing tools to be used at each stage and let the community decide which one will win

➢ Instead of discussing and deciding about programming languages, drivers and operating systems, we should deal with:
  - **Specifications**: descriptions of tasks and desired system behaviour
  - **Models**: abstractions from concrete hardware and software designs – describe **what** the components do
  - **Interfaces**: describe **how** the components interact

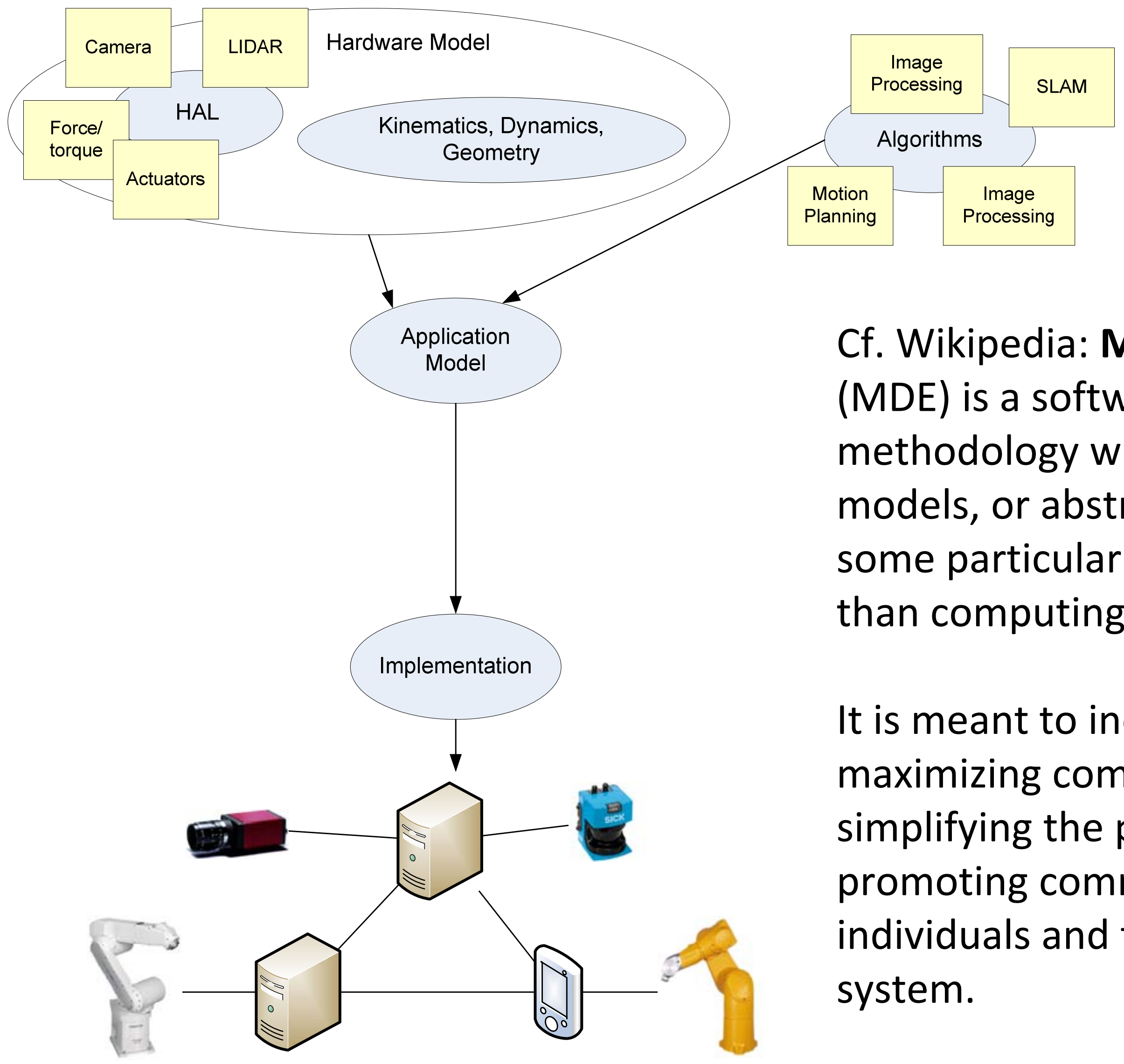
Traditional Development Process



Traditional Development Process



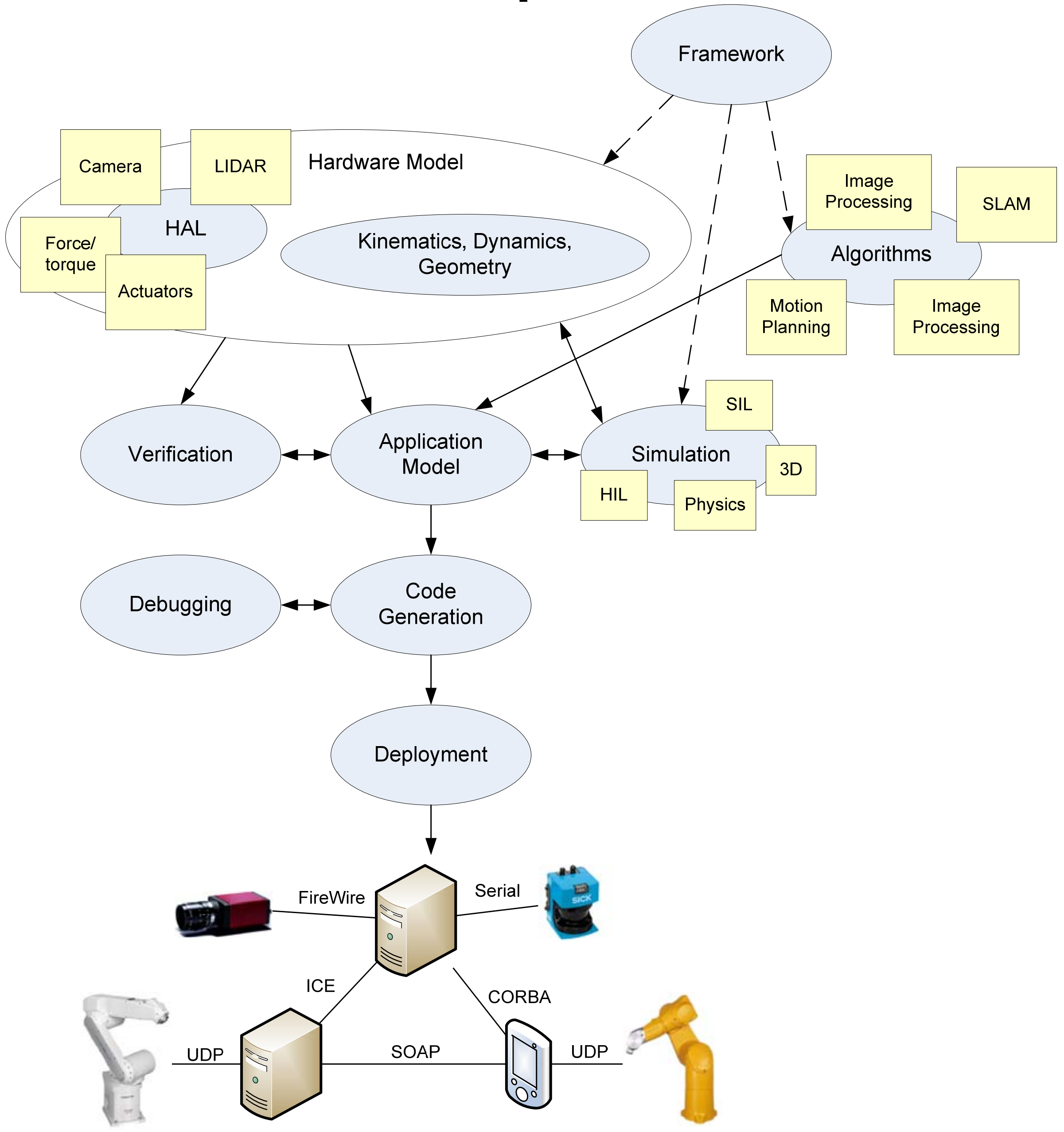
Model-Based Development Process



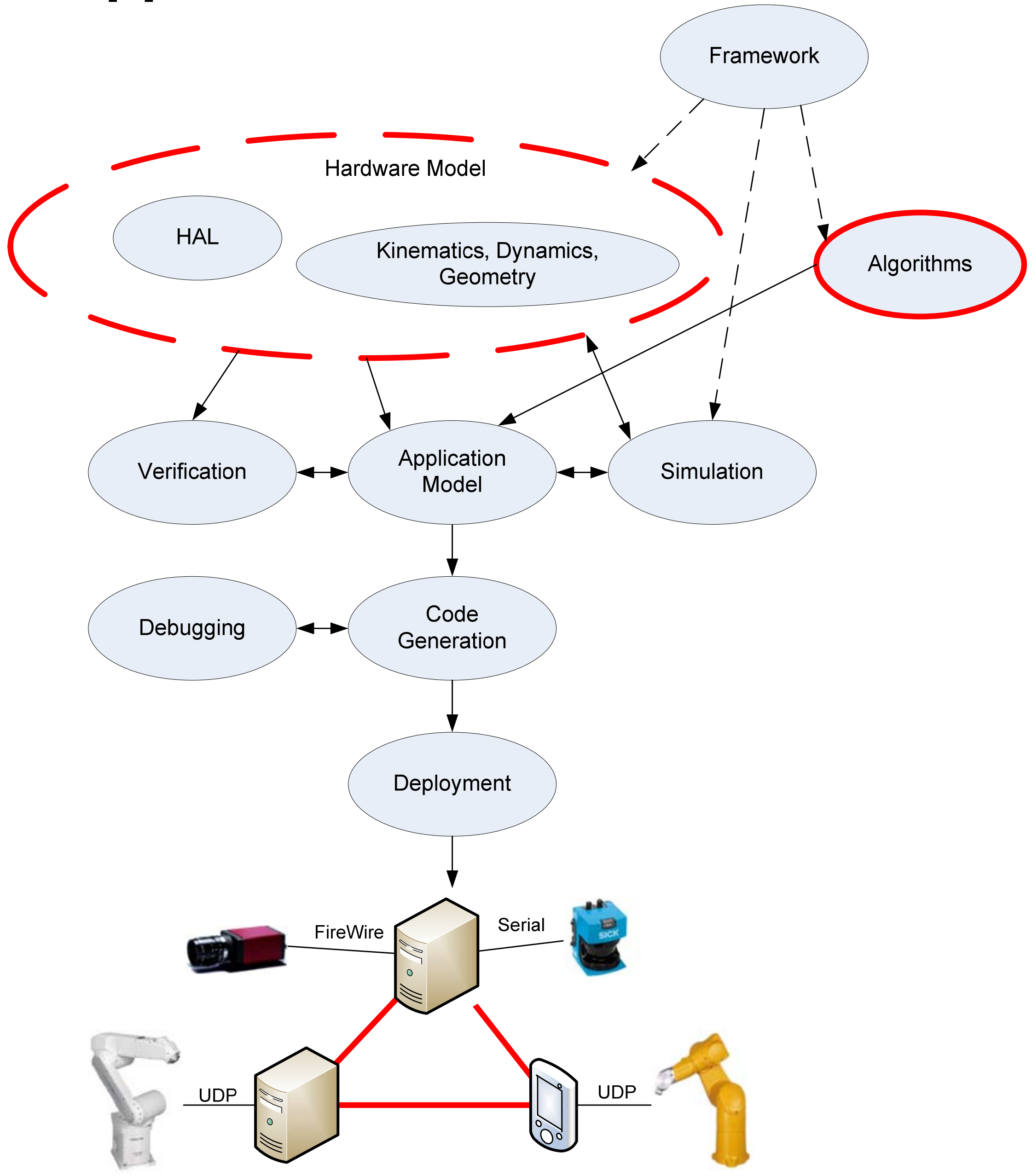
Cf. Wikipedia: **Model-driven engineering** (MDE) is a software development methodology which focuses on creating models, or abstractions, more close to some particular domain concepts rather than computing (or algorithmic) concepts.

It is meant to increase productivity by maximizing compatibility between systems, simplifying the process of design, and promoting communication between individuals and teams working on the system.

Refined Model-Driven Development Process



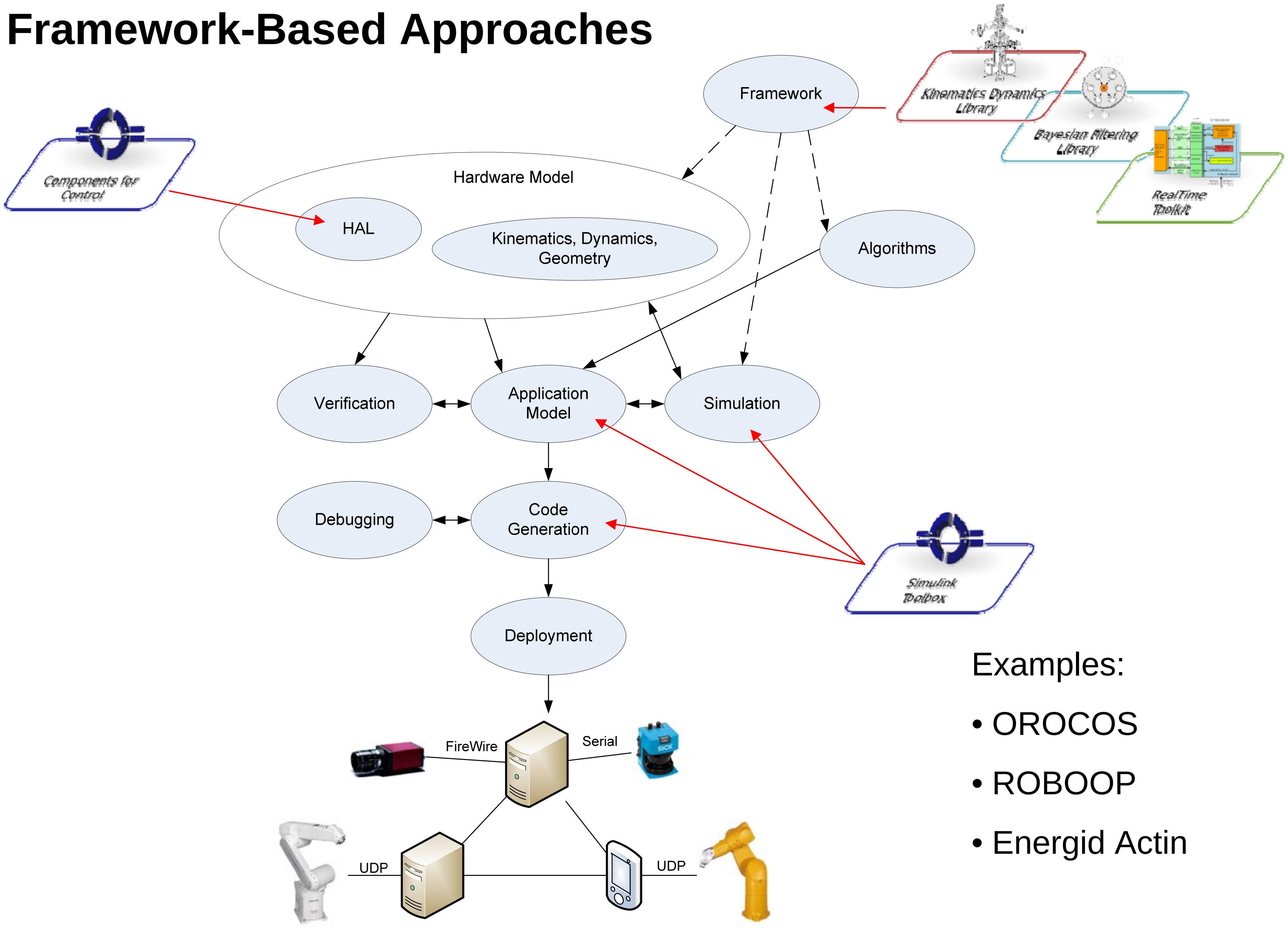
Middleware Approaches



Examples:

- ORCA
- YARP

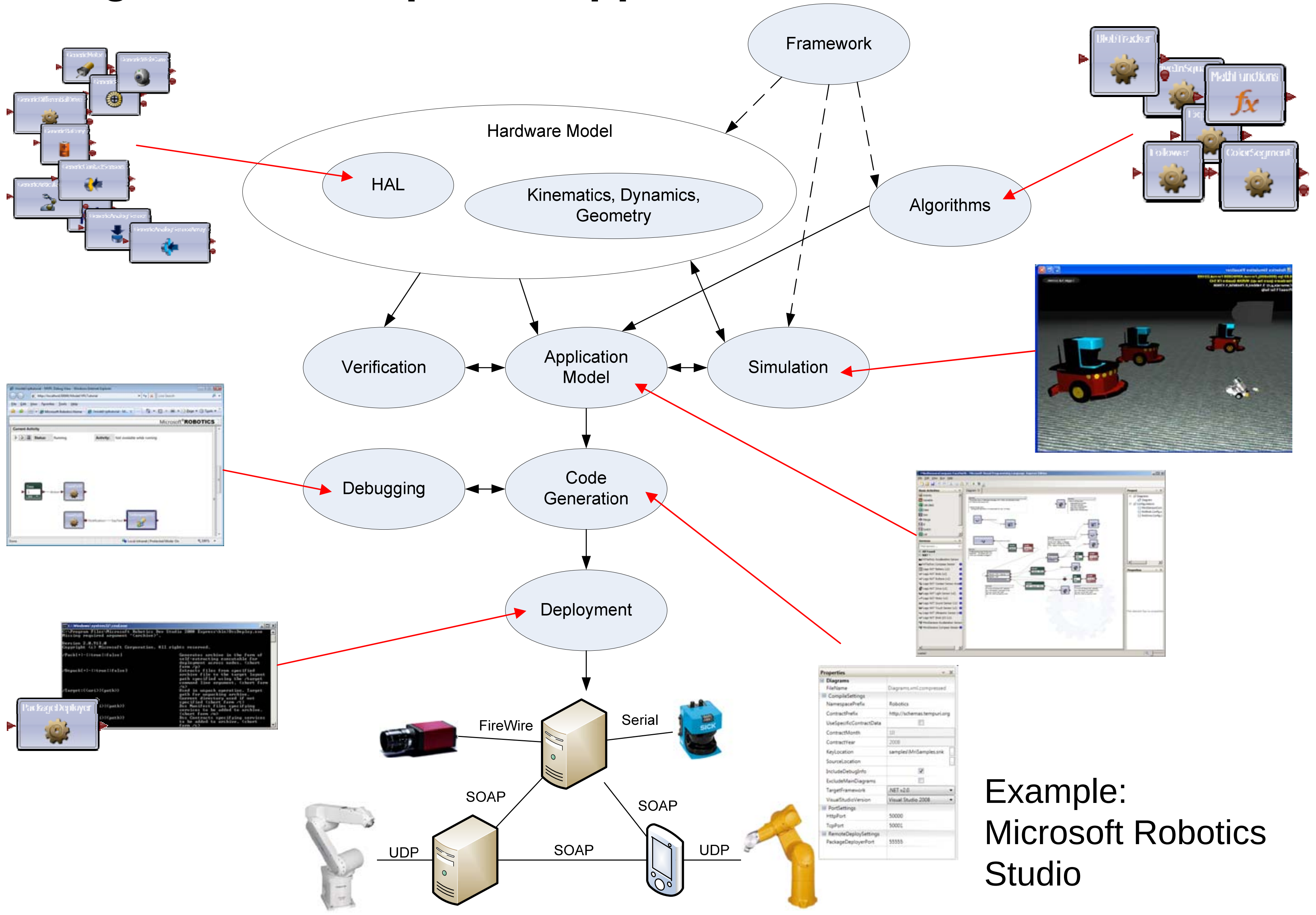
Framework-Based Approaches



Examples:

- OROCOS
- ROBOOP
- Energid Actin

Integrated Development Approaches

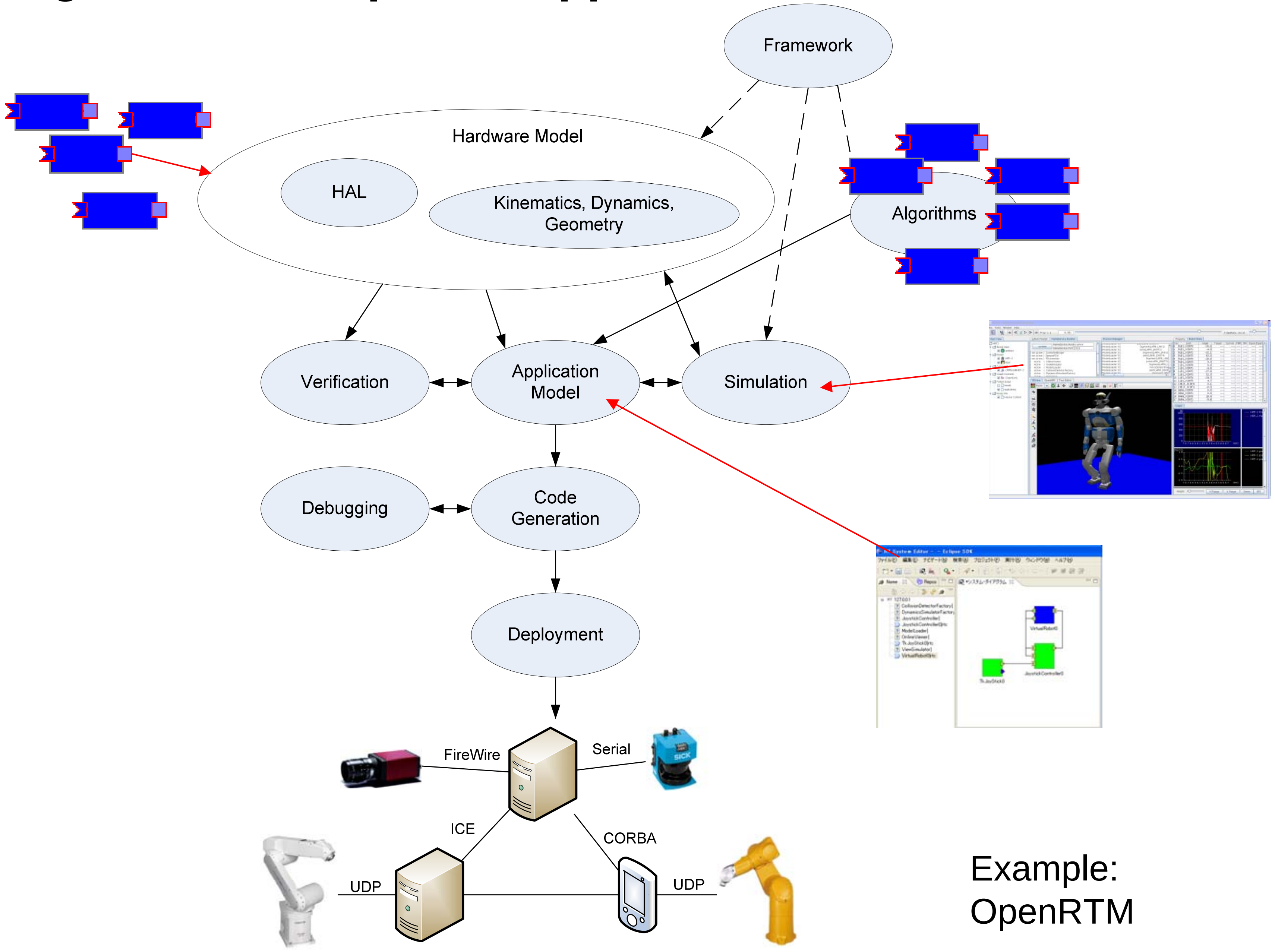


```
C:\Program Files\Microsoft Robotics Dev Studio 2008 Express\bin\DssDeploy.exe
Missing required argument '<archive>'.

Version 2.0.913.0
Copyright (c) Microsoft Corporation. All rights reserved.

/Pack[+|-|(true|false)]                Generates archive in the form of
                                       self-extracting executable for
                                       deployment across nodes. (Short
                                       form /p)
/Unpack[+|-|(true|false)]              Extracts files from specified
                                       archive file to the target layout
                                       path specified using the /target
                                       command line argument. (Short form
                                       /u)
/Target:(<uri>)|(path))                Used in unpack operation. Target
                                       path for unpacking archive.
                                       Current directory used if not
                                       specified (Short form /t)
                  )|(path))            Dss Manifest files specifying
                                       services to be added to archive.
                                       (Short form /m)
                  )|(path))            Dss Contracts specifying services
                                       to be added to archive. (Short
                                       form /c)
```

Properties	
Diagrams	
FileName	Diagrams.xml.compressed
CompileSettings	
NamespacePrefix	Robotics
ContractPrefix	http://schemas.tempuri.org
UseSpecificContractData	☐
ContractMonth	10
ContractYear	2008
KeyLocation	samples\MriSamples.snk
SourceLocation	
IncludeDebugInfo	☑
ExcludeMainDiagrams	☐
TargetFramework	.NET v2.0
VisualStudioVersion	Visual Studio 2008
PortSettings	
HttpPort	50000
TcpPort	50001
RemoteDeploySettings	
PackageDeployerPort	55555

Example:
Microsoft Robotics Studio

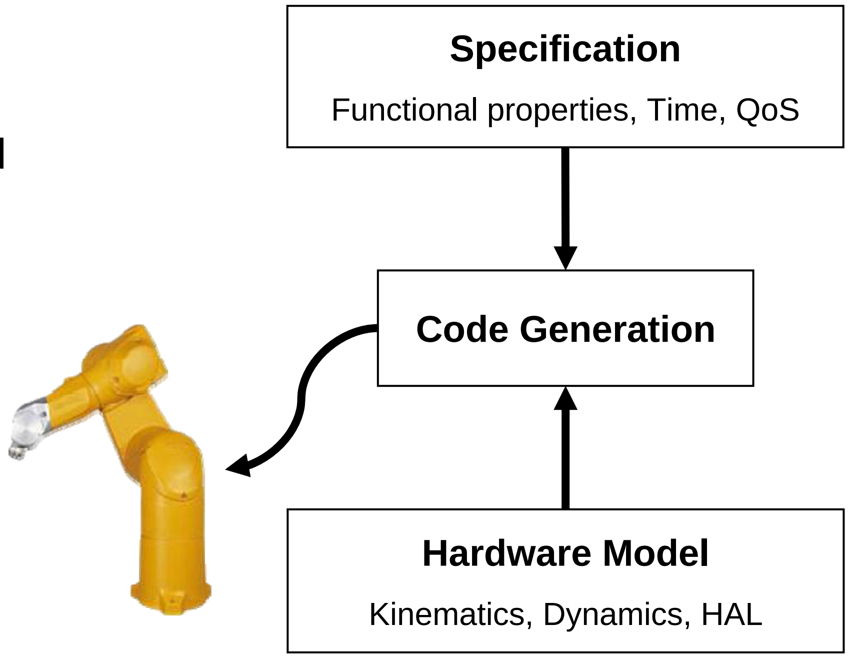
Integrated Development Approaches



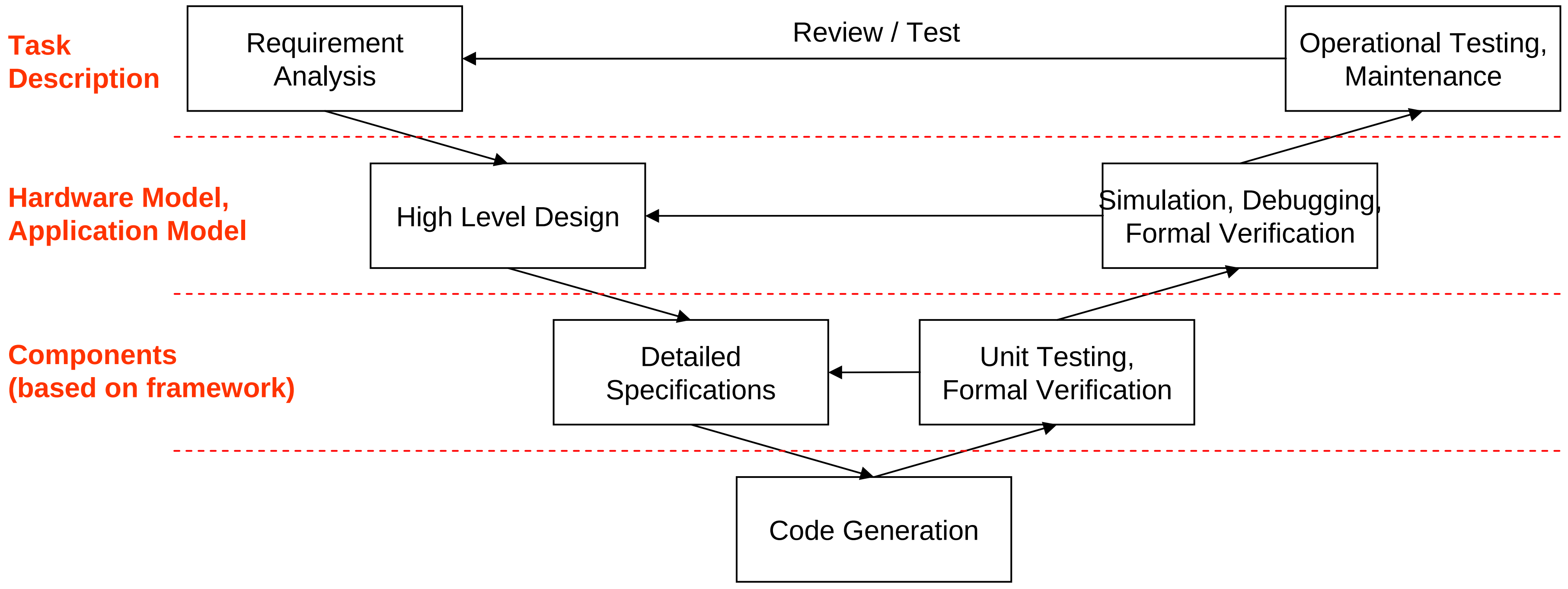
Example:
OpenRTM

Robotics Technology Workflow

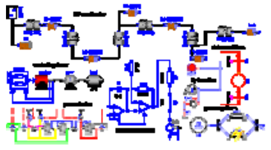
- A meta tool chain developed at **TUM** in cooperation with **AIST Tsukuba** for the complete development cycle
  - High-level behavior description
  - Framework libraries at middle layer
  - Performance-optimized code at low level
  - Additional tools for verification, simulation, debugging and deployment
- Support for many platforms
  - Embedded systems
  - Real-time capability
  - HAL with device hierarchy (actuators, sensors, etc.)
- Important features
  - Extensibility in all dimensions, open standards
  - Real-time capability
  - Reference implementations of all tools are/will be freely available

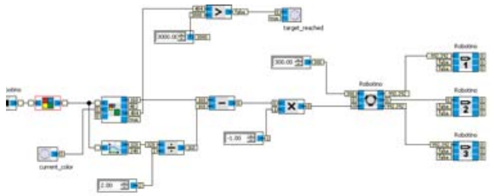


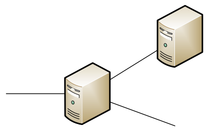
Robotics Software Engineering Process


Task Description
Hardware Model, Application Model
Components (based on framework)
Requirement Analysis
Review / Test
Operational Testing, Maintenance
High Level Design
Simulation, Debugging, Formal Verification
Detailed Specifications
Unit Testing, Formal Verification
Code Generation

Typical Target Workflow

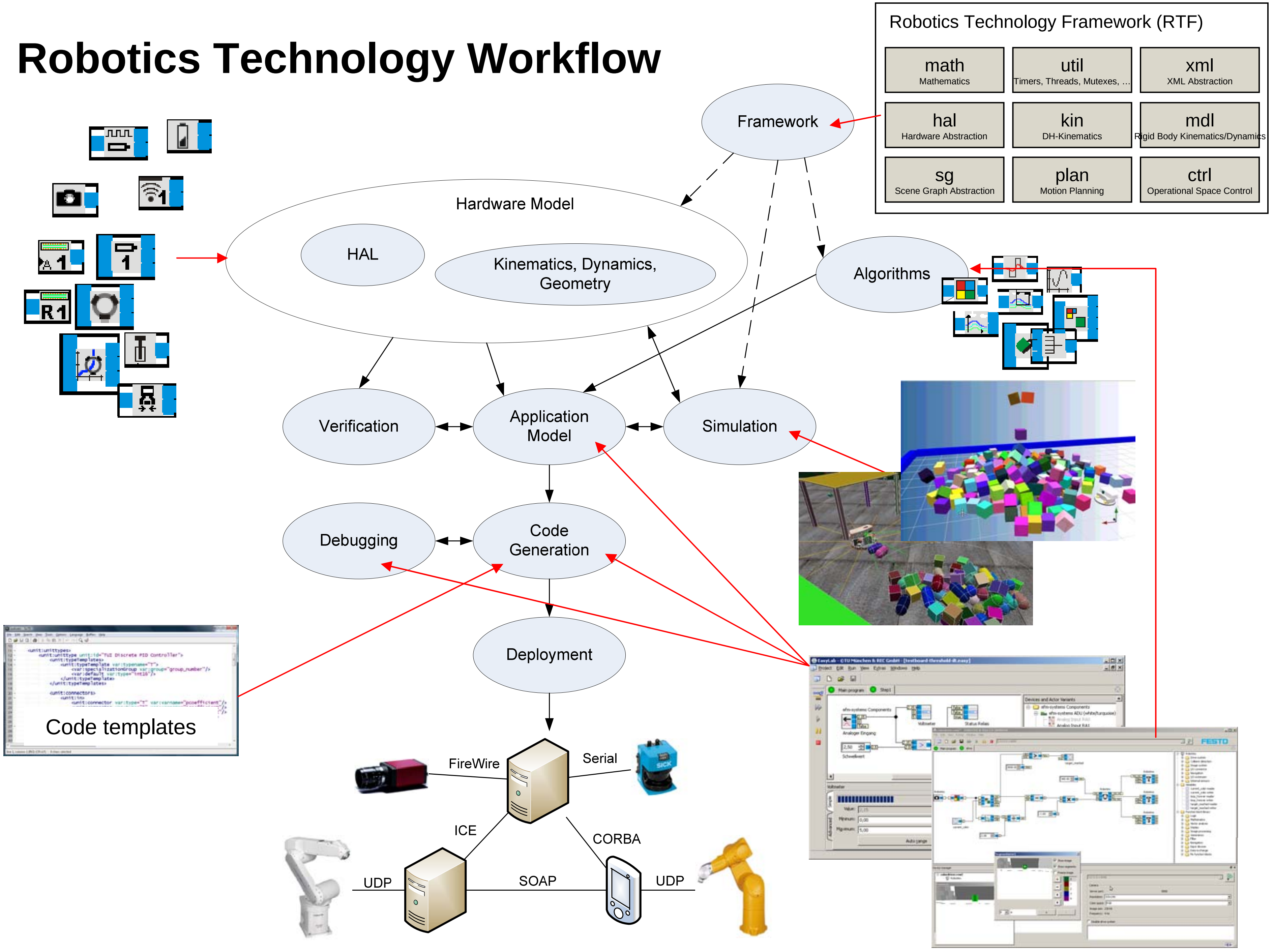






1. Define new robot system
   - Specify physical properties: kinematics, dynamics, geometry
   - Add support for robot’s devices to hardware abstraction layer: Device drivers for sensors and actuators
2. Define behavior of robot system
   - Add new components to architecture: image processing, motion planning, task-based descriptions, …
   - Define temporal constraints and application logic
3. Simulation and verification
4. Code generation, deployment and debugging

Robotics Technology Workflow

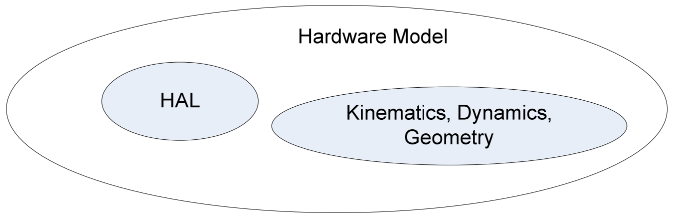


```
<unit:unittypes>
    <unit:unittype unit:id="TUI Discrete PID Controller">
        <unit:typeTemplates>
            <unit:typeTemplate var:typename="T">
                <var:specializationGroup var:group="group_number"/>
                <var:default var:type="int16"/>
            </unit:typeTemplate>
        </unit:typeTemplates>

        <unit:connectors>
            <unit:in>
                <unit:connector var:type="T" var:varname="pcoefficient"/>
```

A word about models …

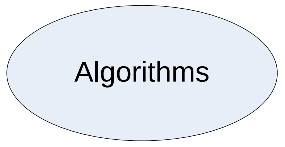
- Hardware components
  “What do we have?”
  - Kinematics, dynamics, geometry, gear mechanics, …
  - Hardware abstraction layer (HAL)
    - Device drivers
    - Operating system
    - Target architecture
  - Container for meta-information to cover various aspects of system

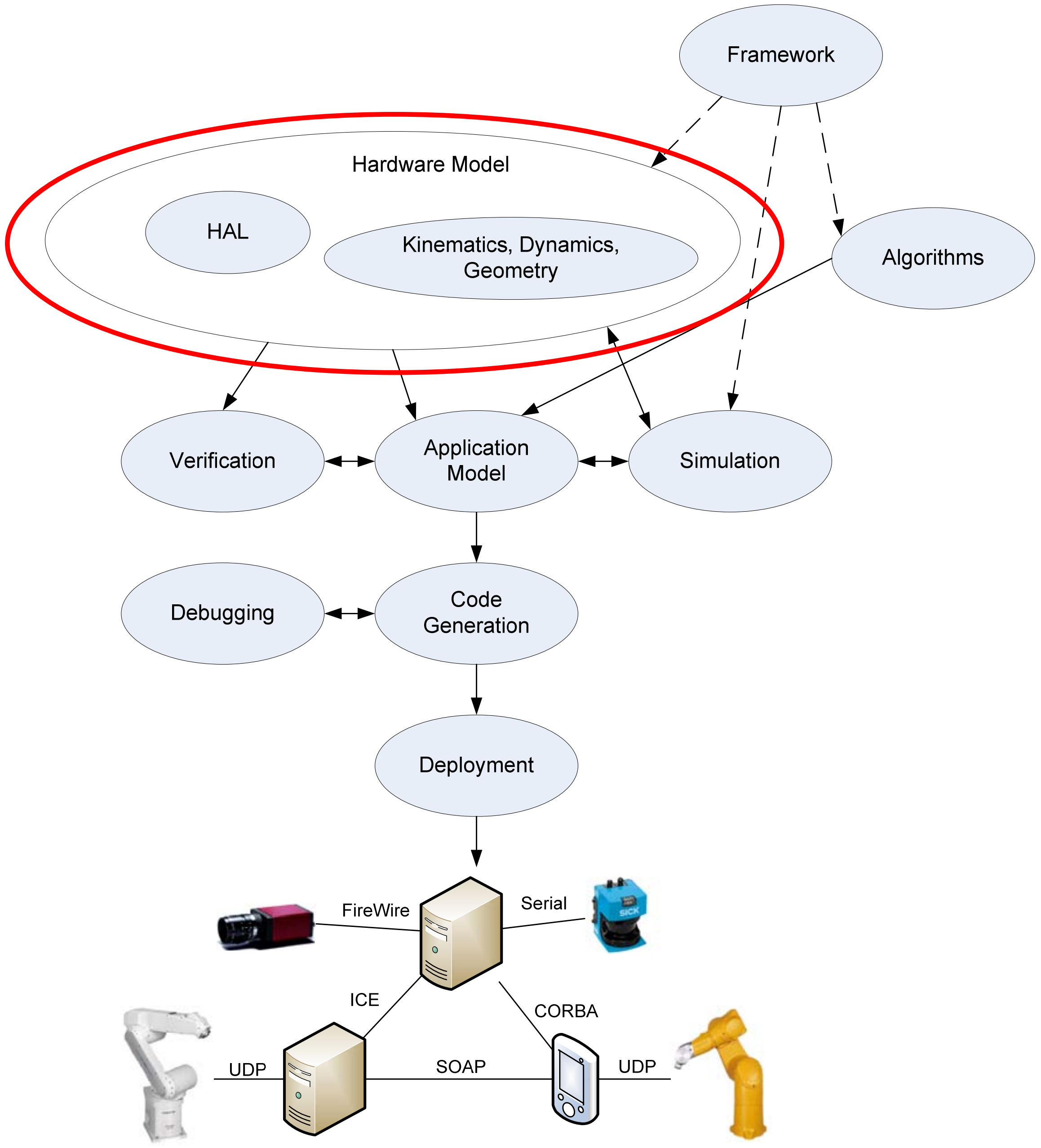
→ Modeling of physical properties



- Algorithmic components
  “What do we want to do?”
  - Basic signal processing
  - Behavior based programming
  - Abstract task specification

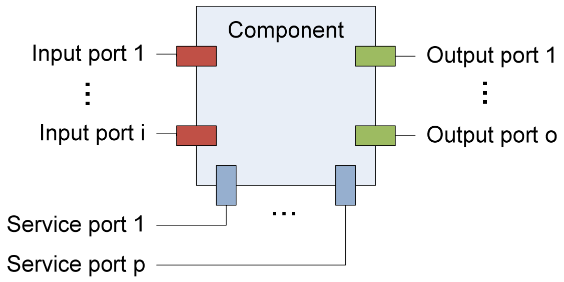
→ Modeling of functional properties

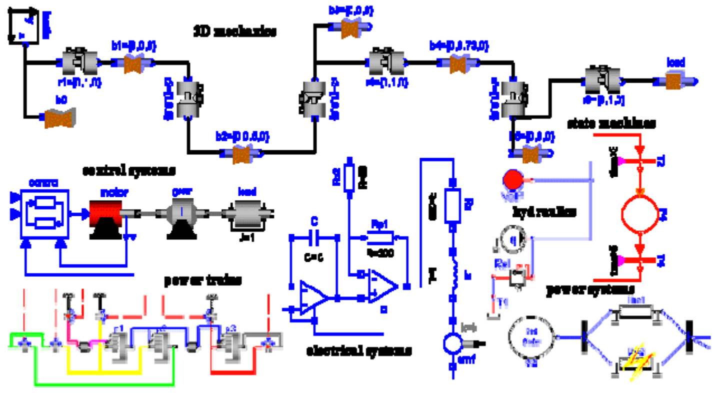


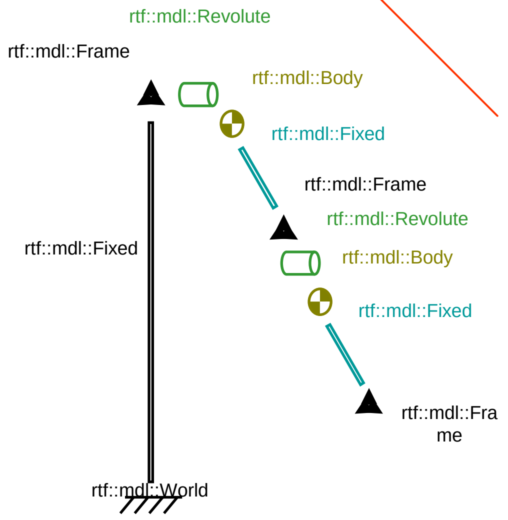


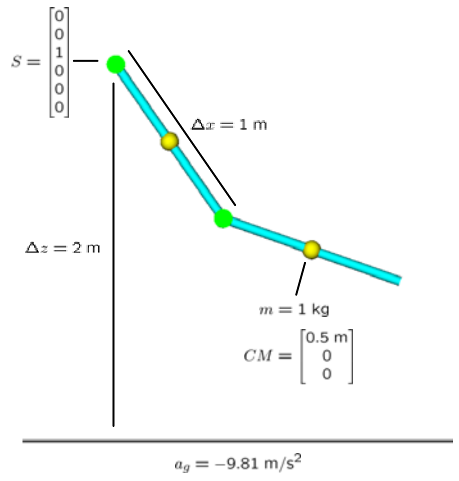
Framework
Hardware Model
HAL
Kinematics, Dynamics, Geometry
Algorithms
Verification
Application Model
Simulation
Debugging
Code Generation
Deployment
FireWire
Serial
ICE
CORBA
UDP
SOAP
UDP

Hardware Model





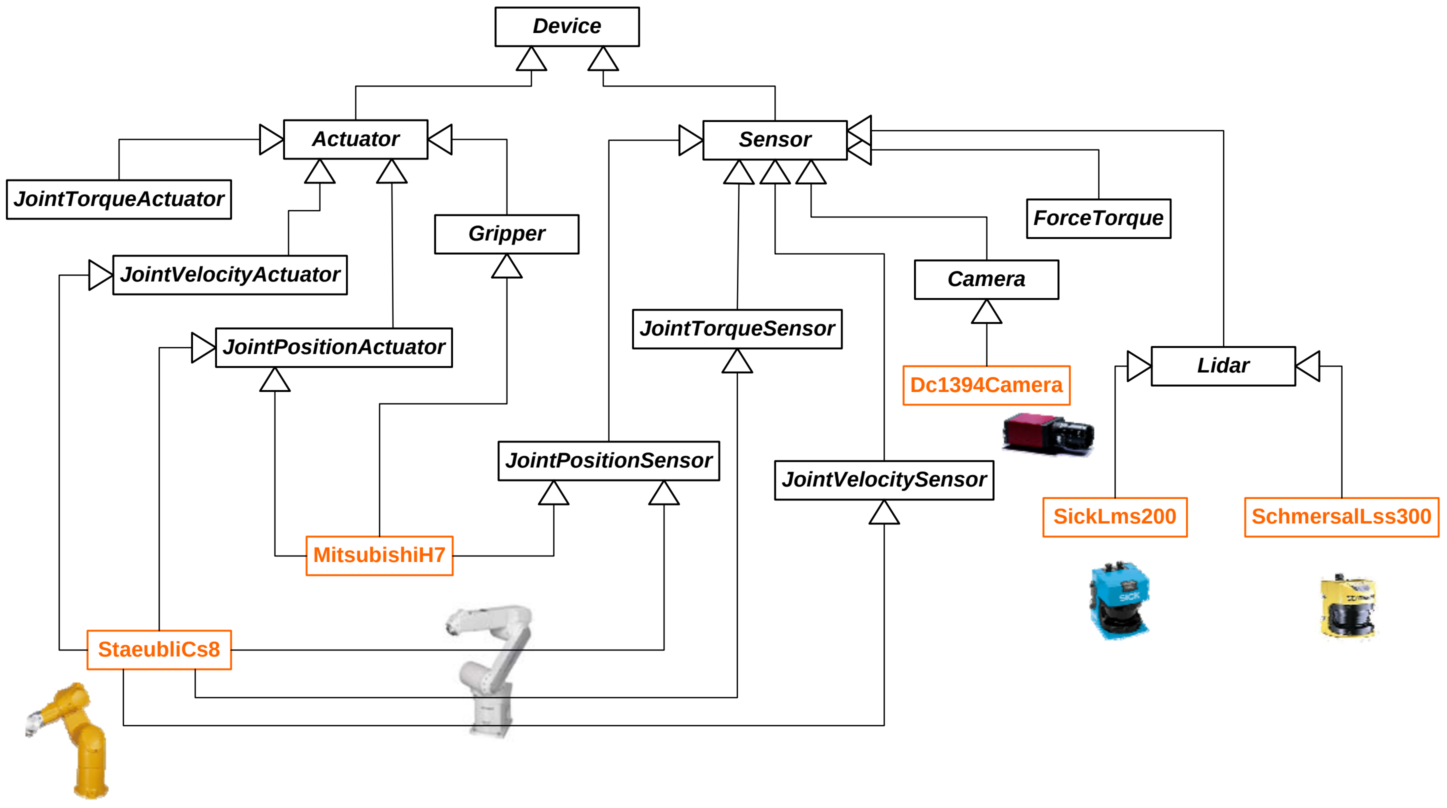


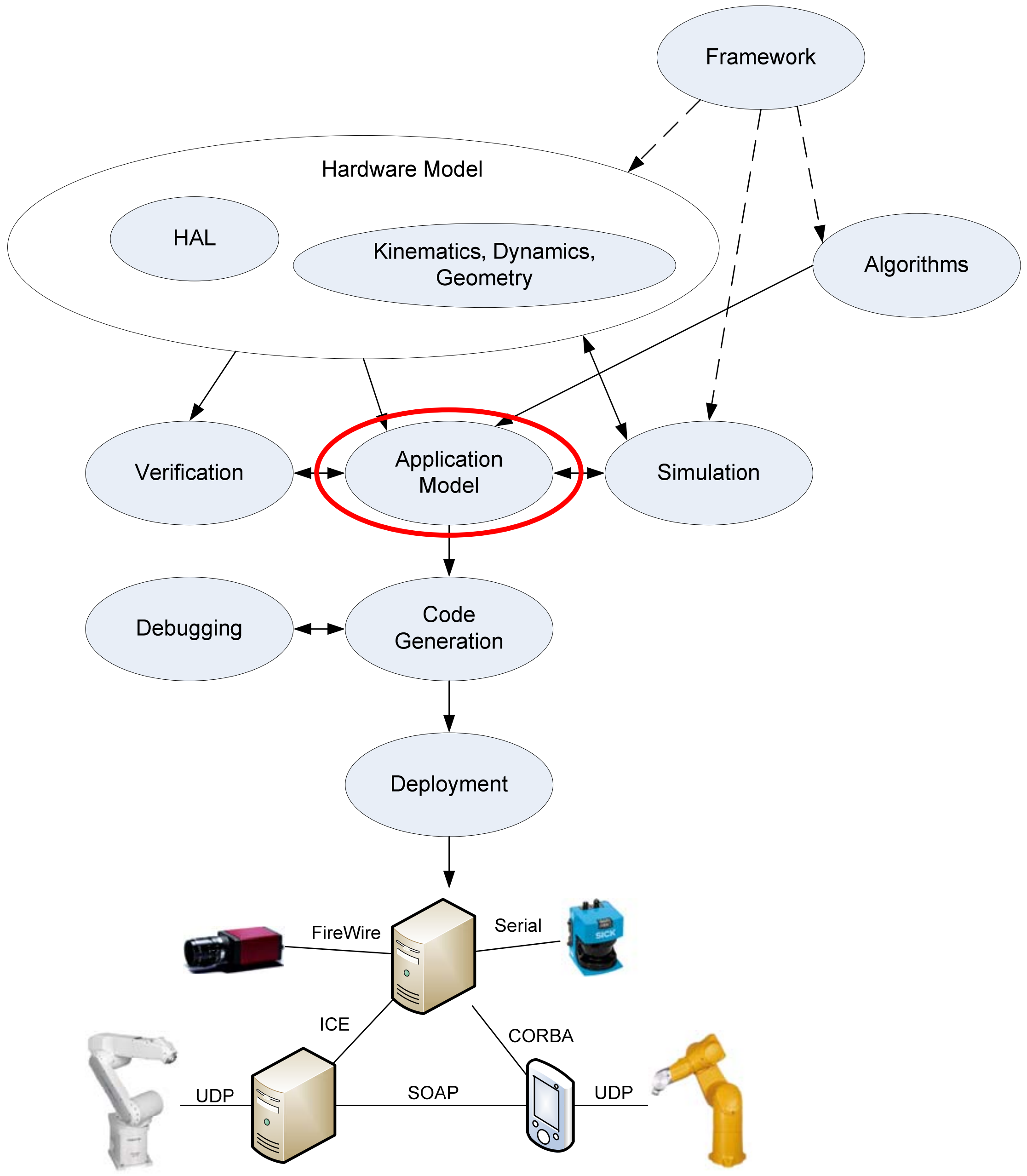

$S = \begin{bmatrix}0\\0\\1\\0\\0\\0\end{bmatrix}$
$\Delta x = 1$ m
$\Delta z = 2$ m
$m = 1$ kg
$CM = \begin{bmatrix}0.5\text{ m}\\0\\0\end{bmatrix}$
$a_g = -9.81$ m/s^2

Contains:

- Specification of physical properties for verification, including functional data and implementation (kinematics, dynamics, geometry, gear mechanics, …)
- Drivers for actual control of corresponding hardware devices

HAL: Common Interface to Platform Functionality

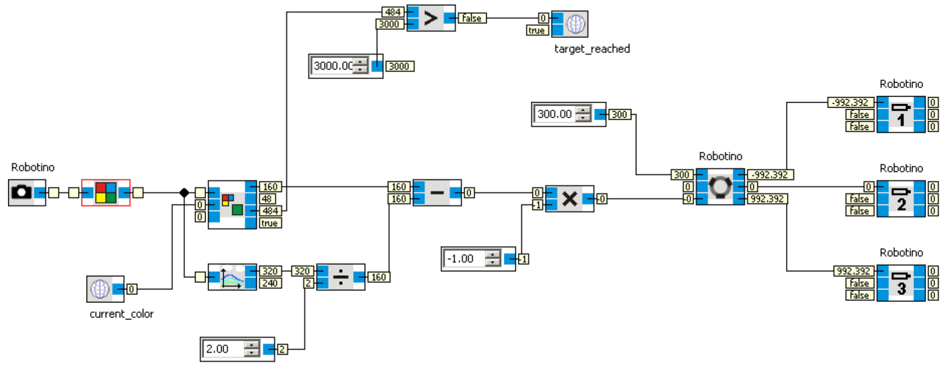


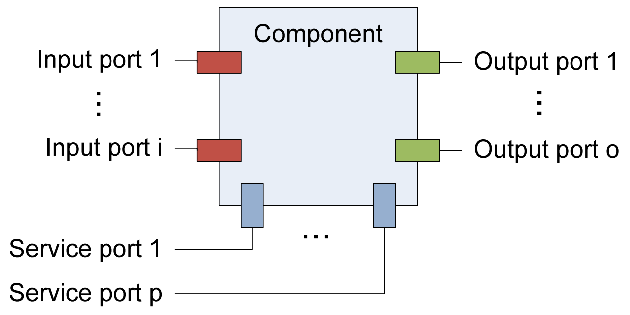

Framework
Hardware Model
HAL
Kinematics, Dynamics, Geometry
Algorithms
Verification
Application Model
Simulation
Debugging
Code Generation
Deployment
FireWire
Serial
ICE
CORBA
UDP
SOAP
UDP

Application Model

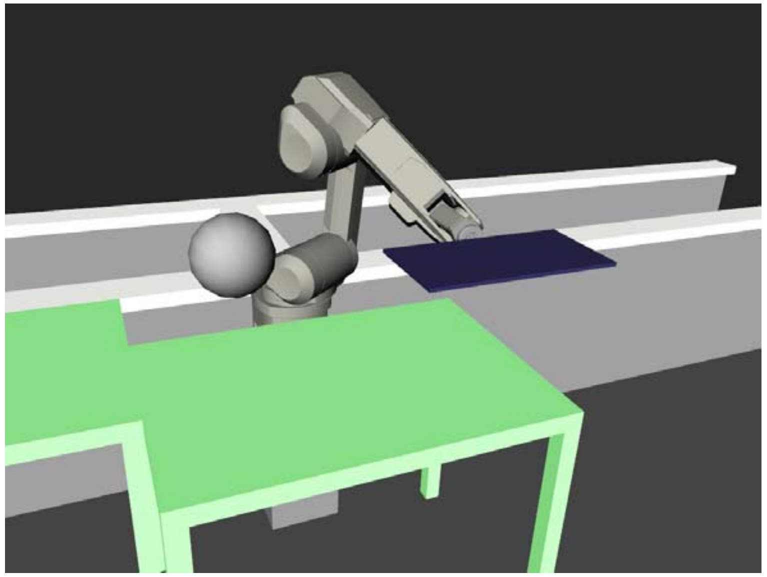
Description of **functional behavior** at three levels:

- Low-level controllers
  (Pneumatic cylinder, inverted pendulum, …)
- Behavior-based controllers
  (Line-follow, “find-the-ball”, …)
- Abstract task descriptions



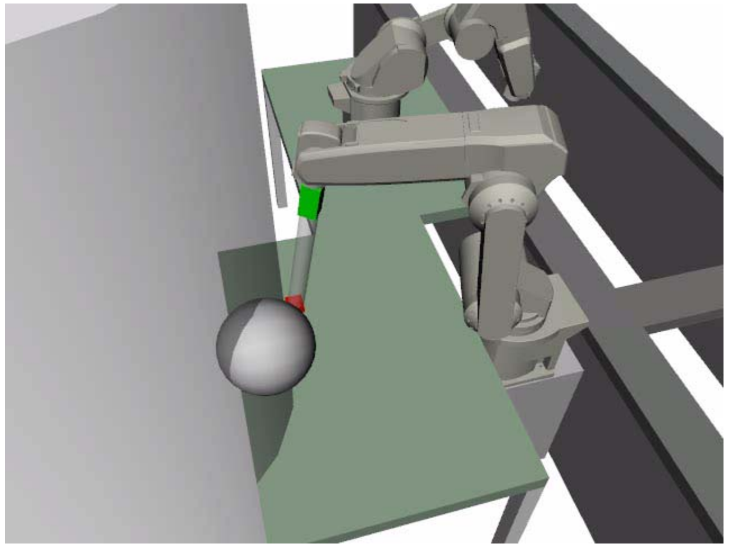


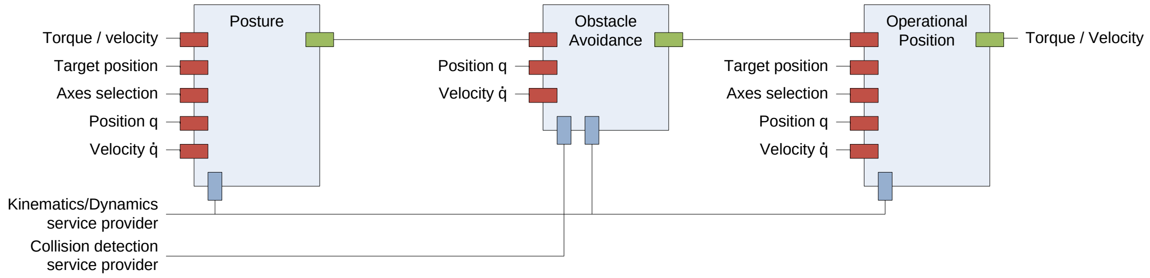
Application Model: Task Description

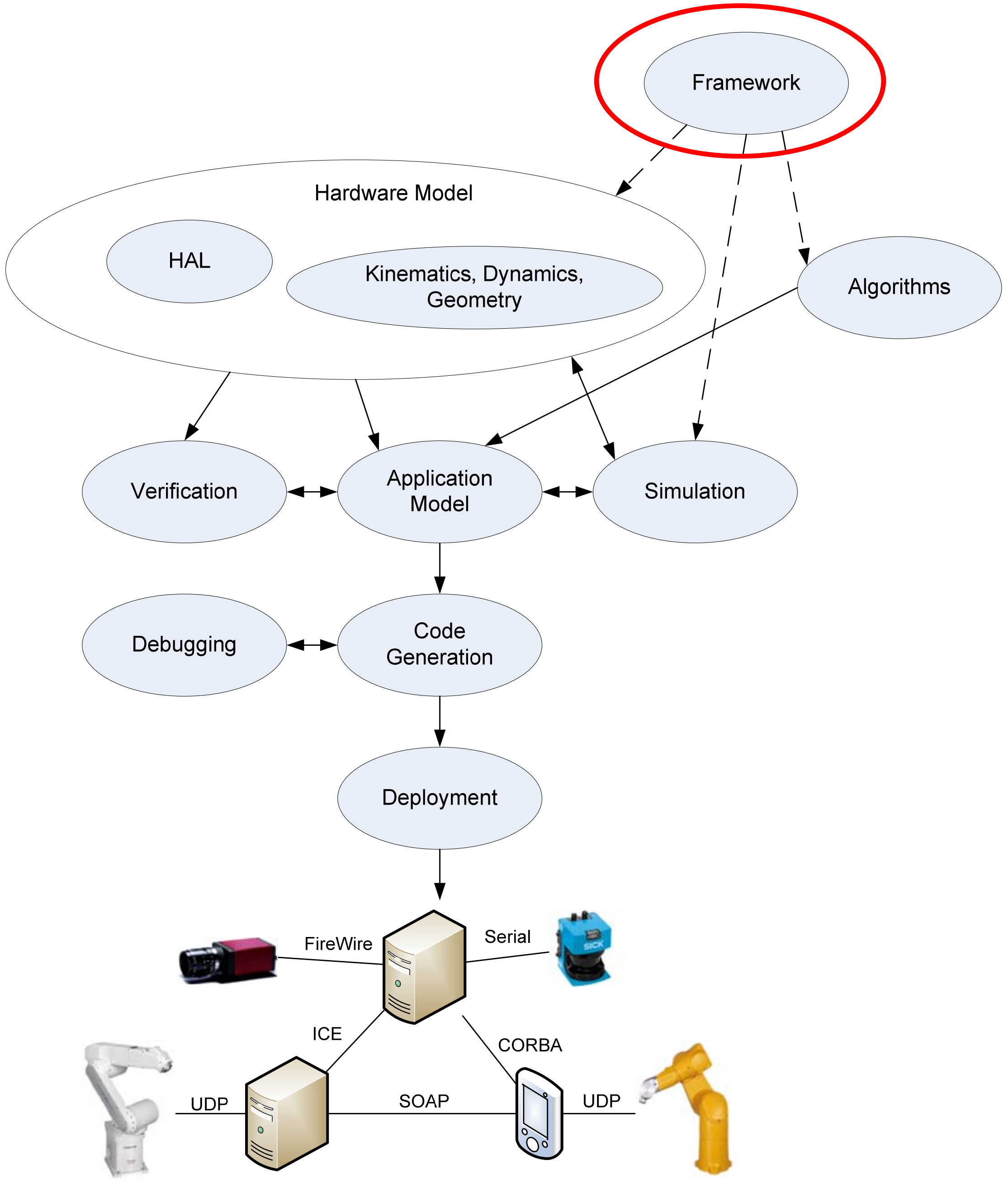


Example using the same graphical blocks representations:

- Holding orientation and height of tablet while avoiding collision and keeping posture (A/B/C fixed, X/Y free)
- Holding TCP position while avoiding collision and keeping posture (X/Y/Z fixed, A/B/C free)






Framework
Hardware Model
HAL
Kinematics, Dynamics, Geometry
Algorithms
Verification
Application Model
Simulation
Debugging
Code Generation
Deployment
FireWire
Serial
ICE
CORBA
UDP
SOAP
UDP

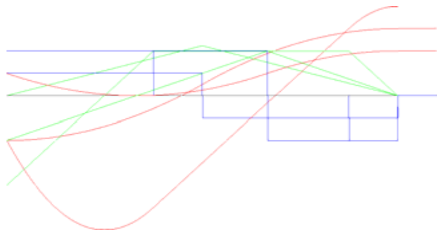
Framework Libraries: Comprehensive Set of Functions

- Basic mathematics

$$^{B}X_A = \begin{bmatrix} E & 0 \\ -Er\times & E \end{bmatrix}$$

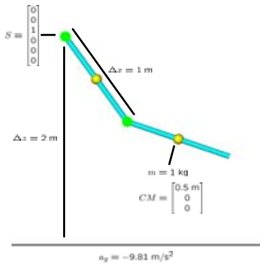
$$\begin{bmatrix} a_{i,j} & \cdots & a_{i,l} \\ \vdots & \ddots & \vdots \\ a_{k,j} & \cdots & a_{k,l} \end{bmatrix}$$

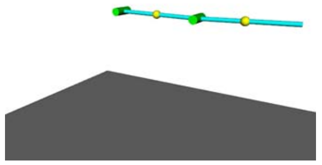
$$x + y \cdot i + z \cdot j + w \cdot k$$



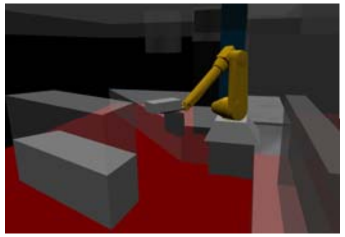
- Kinematics and dynamics

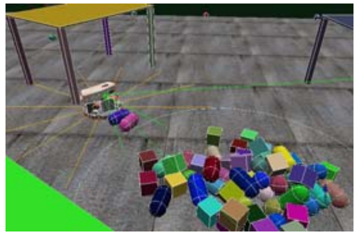
$$\dot{x} = J\dot{q}$$
$$\tau = M(q)\ddot{q} + V(q,\dot{q}) + G(q)$$
$$\ddot{q} = M^{-1}(q)[\tau - V(q,\dot{q}) - G(q)]$$


$S = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$
$\Delta x = 1$ m
$\Delta z = 2$ m
$m = 1$ kg
$CM = \begin{bmatrix} 0.5\text{ m} \\ 0 \\ 0 \end{bmatrix}$
$a_g = -9.81$ m/s^2

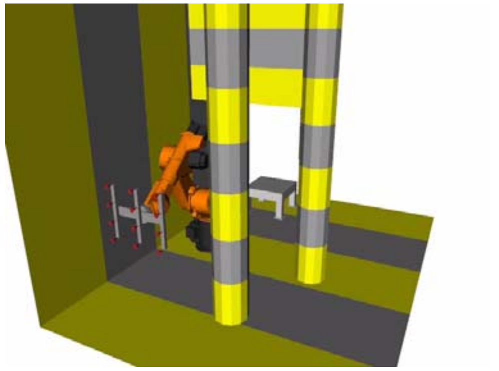


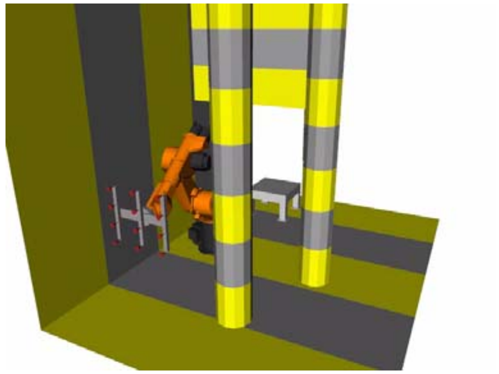
- Scene-graph abstraction / Collision detection

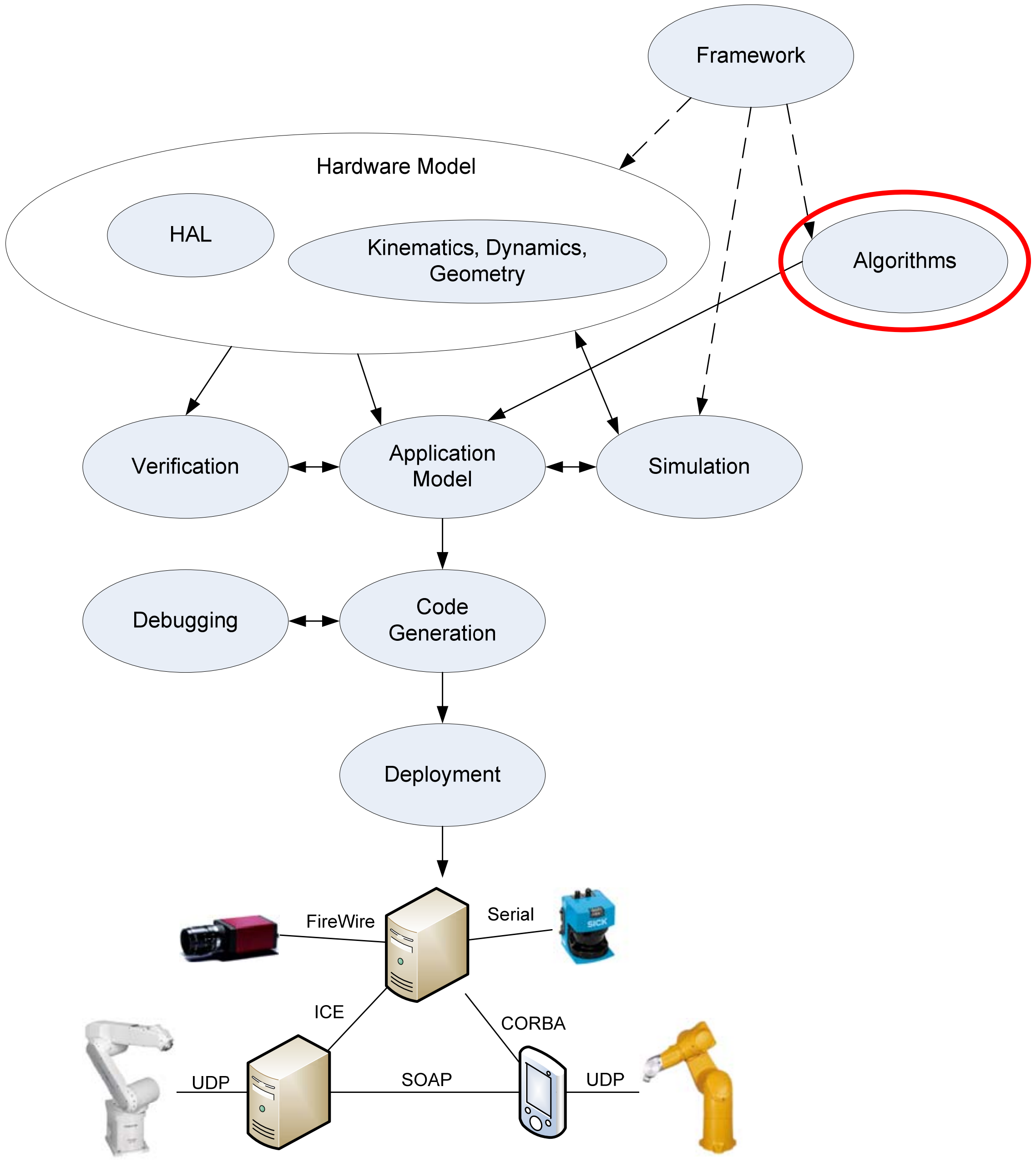




- Motion planning / Operational space control

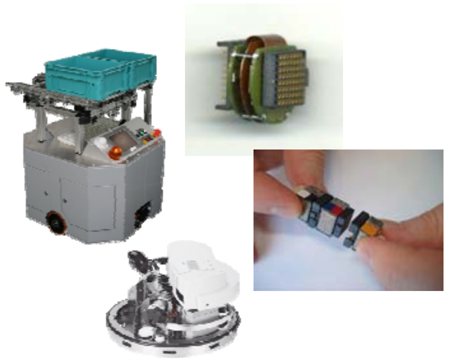


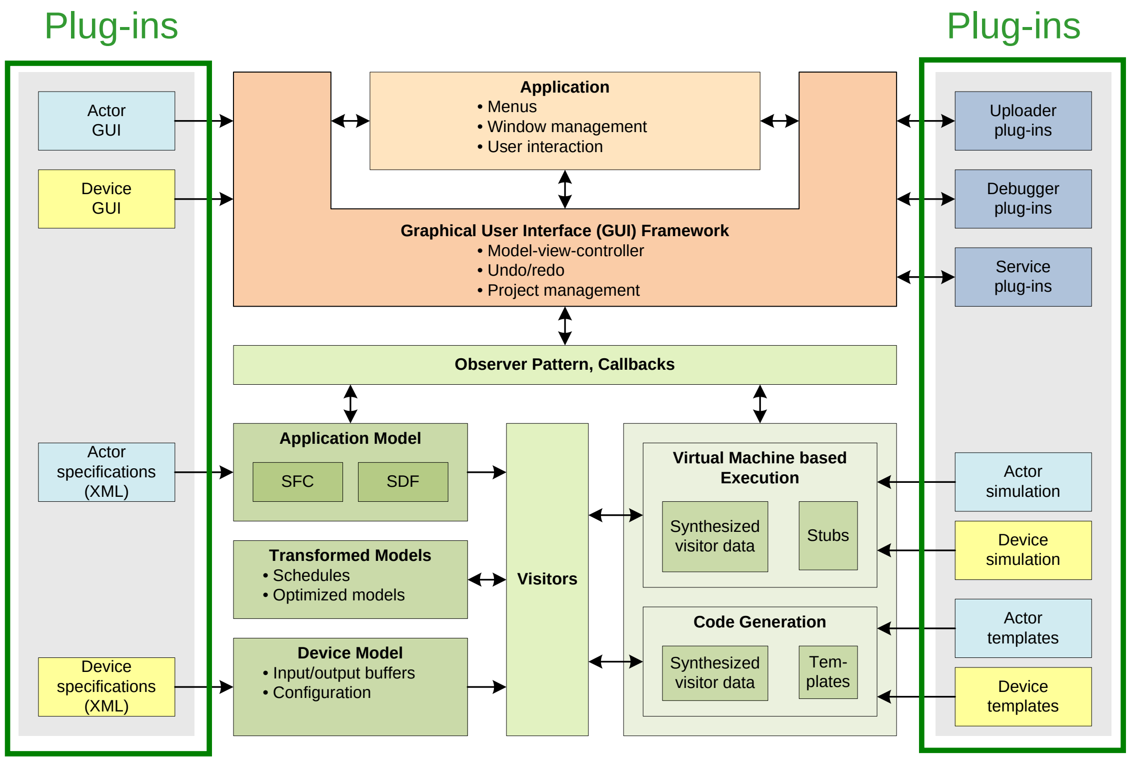



Framework
Hardware Model
HAL
Kinematics, Dynamics, Geometry
Algorithms
Verification
Application Model
Simulation
Debugging
Code Generation
Deployment
FireWire
Serial
ICE
CORBA
UDP
SOAP
UDP

Component architecture

- Definition of new components (plug-in system)
  - Hierarchical composition
  - Atomic components
- Support for new target platforms
  - Abstraction layers for hardware platforms

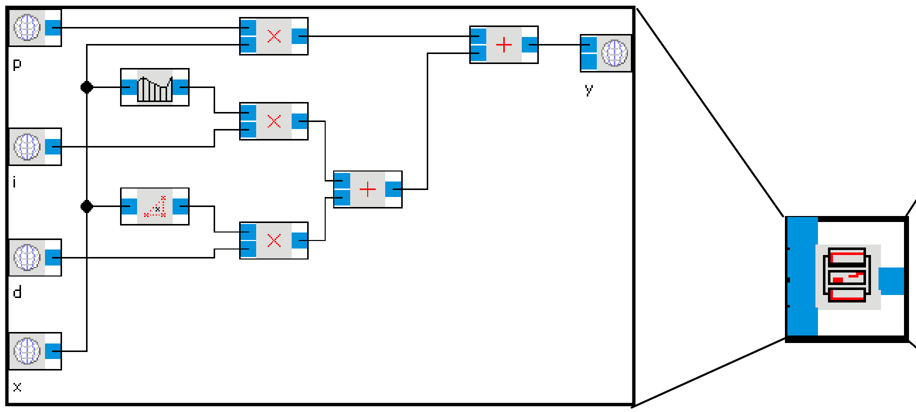


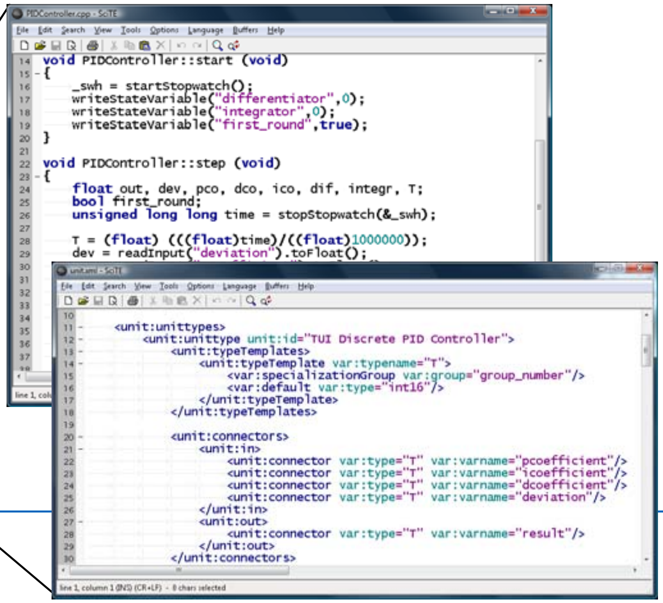


Creation of new Components

- Hierarchical composition
  - "Grouping" of existing components
  - Advantage
    - No extra code templates have to be specified
  - Disadvantage
    - No completely new functionality can be added

- Atomic components
  - Specification of interfaces (XML)
  - Code templates and plug-ins
  - Advantage
    - Full flexibility
  - Disadvantage
    - More time consuming




PIDController.cpp - SciTE
File Edit Search View Tools Options Language Buffers Help
unit.xml - SciTE
line 1, column 1 (INS) (CR+LF) - 0 chars selected

```
void PIDController::start (void)
{
    _swh = startStopwatch();
    writeStateVariable("differentiator",0);
    writeStateVariable("integrator",0);
    writeStateVariable("first_round",true);
}

void PIDController::step (void)
{
    float out, dev, pco, dco, ico, dif, integr, T;
    bool first_round;
    unsigned long long time = stopStopwatch(&_swh);

    T = (float) (((float)time)/((float)1000000));
    dev = readInput("deviation").toFloat();
```

```
    <unit:unittypes>
        <unit:unittype unit:id="TUI Discrete PID Controller">
            <unit:typeTemplates>
                <unit:typeTemplate var:typename="T">
                    <var:specializationGroup var:group="group_number"/>
                    <var:default var:type="int16"/>
                </unit:typeTemplate>
            </unit:typeTemplates>

            <unit:connectors>
                <unit:in>
                    <unit:connector var:type="T" var:varname="pcoefficient"/>
                    <unit:connector var:type="T" var:varname="icoefficient"/>
                    <unit:connector var:type="T" var:varname="dcoefficient"/>
                    <unit:connector var:type="T" var:varname="deviation"/>
                </unit:in>
                <unit:out>
                    <unit:connector var:type="T" var:varname="result"/>
                </unit:out>
            </unit:connectors>
```

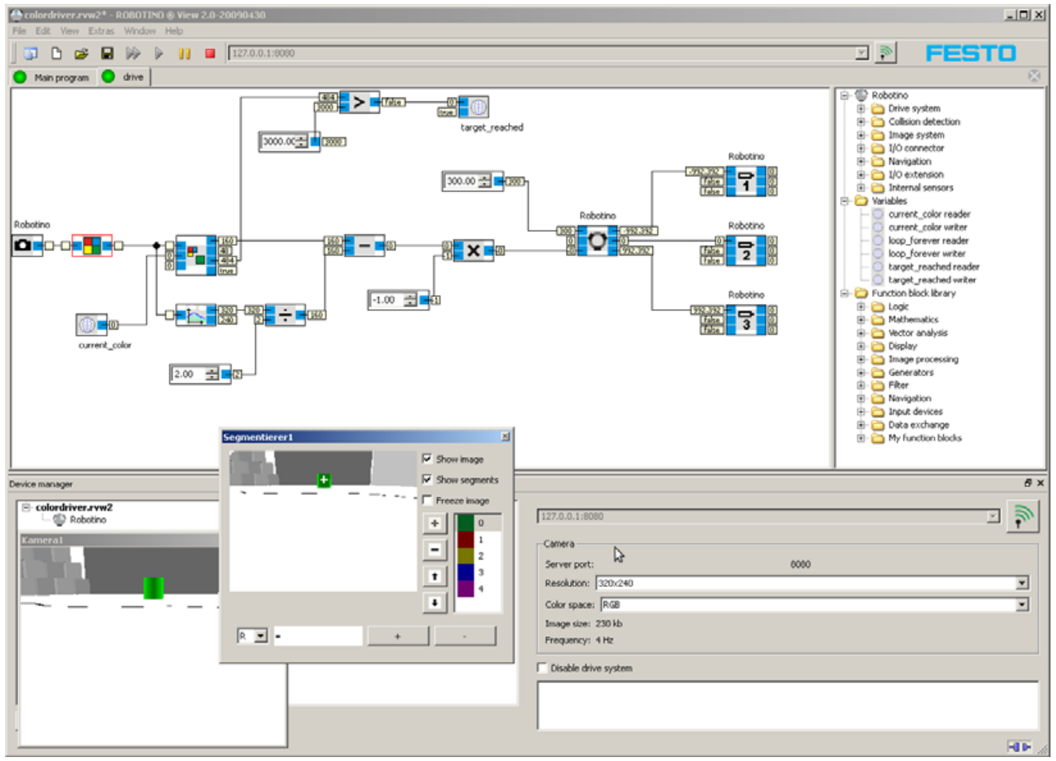
Tools for:

Application Model Development

Simulation

Debugging

Code Generation



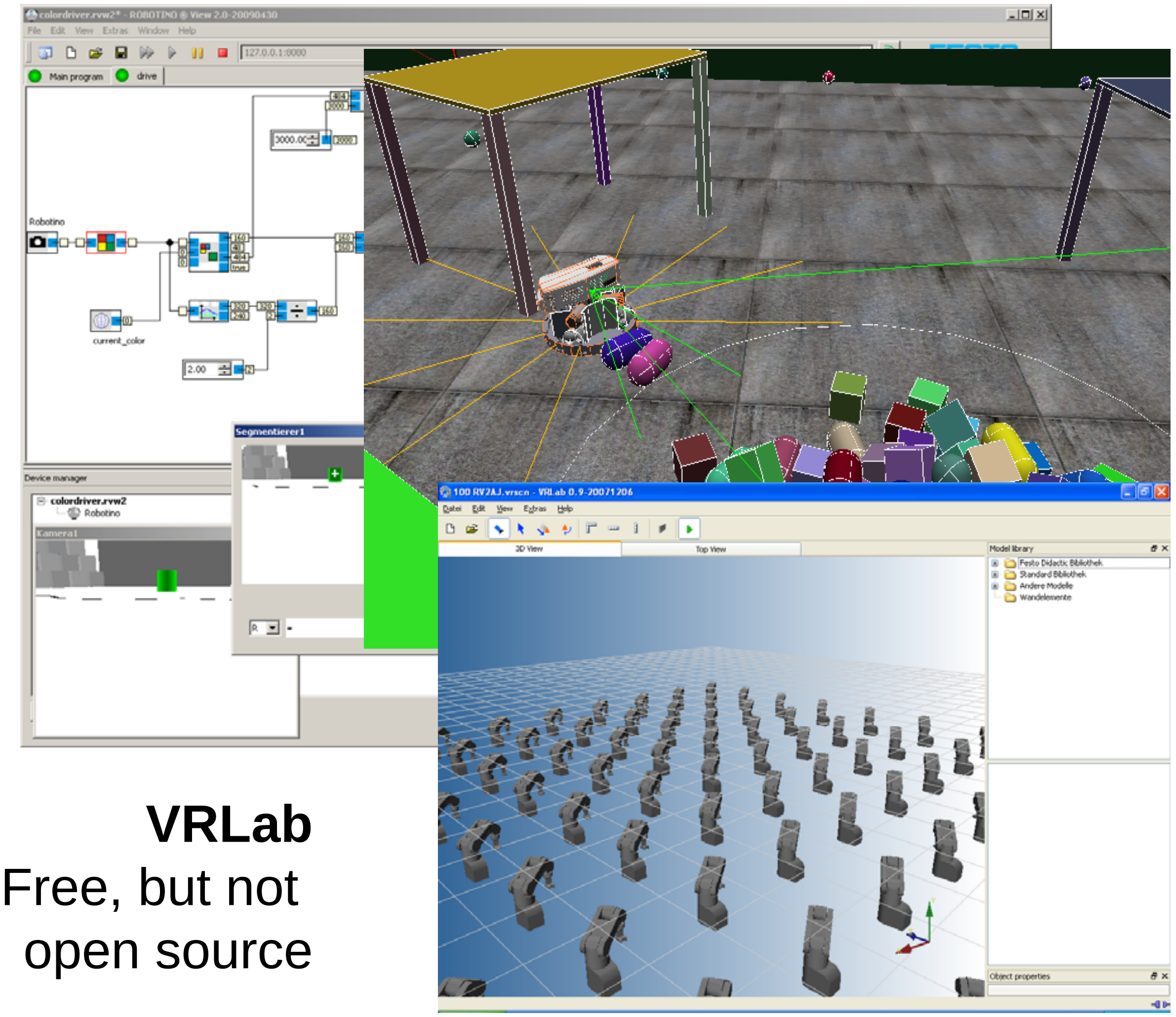
RobotinoView 2.0

Integrated Tool for Programming Robotino

Implements:
- *Application Model Development*
- *VM-based execution and Debugging*

Free,
but not open source

Available soon, current version is RobotinoView 1.7



VRLab
Free, but not open source

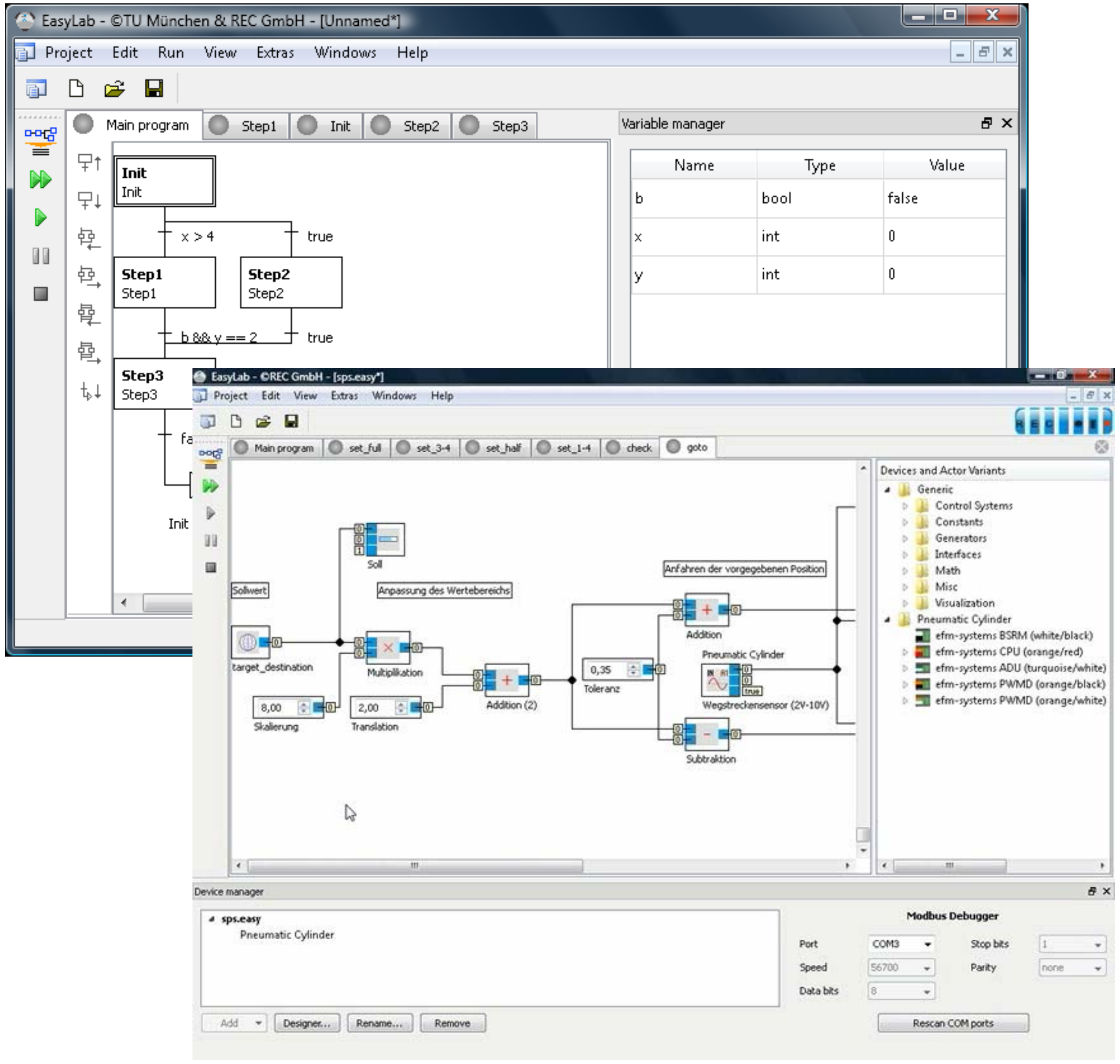
RobotinoSim

Physics simulation for Robotino
Simulator integrated with RobotinoView

Based on Ageia PhysX core (2005)

RobotinoView can switch between real Robotino hardware and physics simulation

Free,
but not open source



Name	Type	Value
b	bool	false
x	int	0
y	int	0

8,00 · 2,00 · 0,35

Port: COM3 · Speed: 56700 · Data bits: 8 · Stop bits: 1 · Parity: none

EasyLab 1.0

Integrated Tool for Programming Embedded Devices

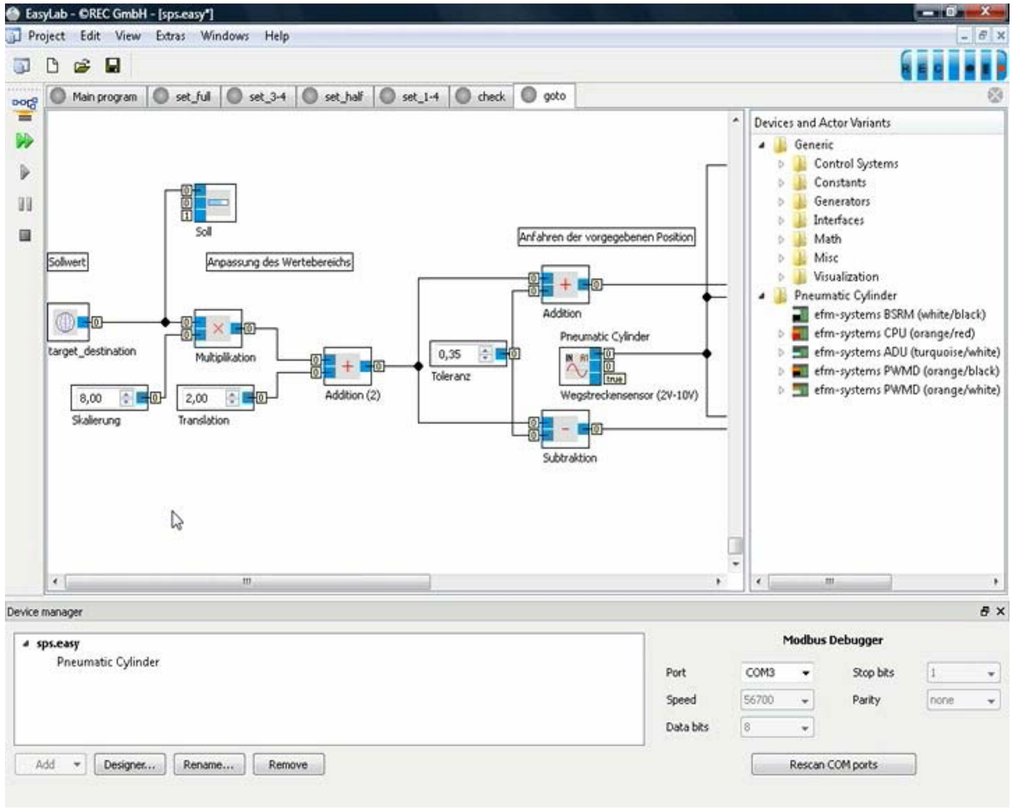
Implements:
- *Application Model Development*
- *Live Debugging*
- *Code Generation*

Kernel (models, code generation, simulation):
- free, open source.
- Partly shared with RobotinoView 2.0

GUI free, but closed source

Debugging

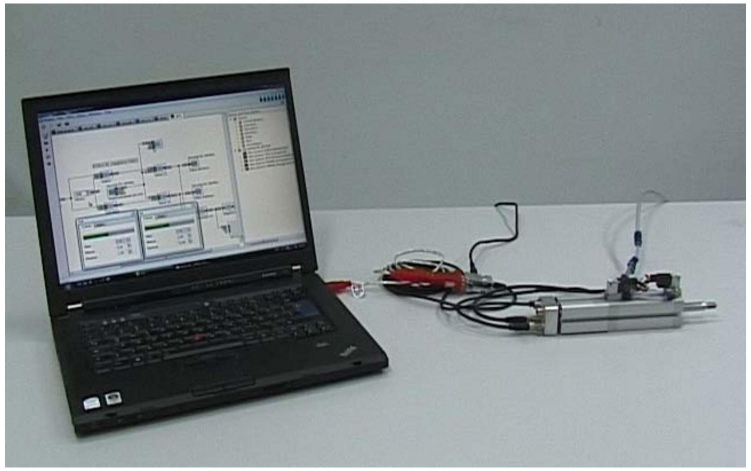
- Data exchange over standard transport layers
- Inspection and manipulation of model attributes
- Step-wise execution



8,00 Skalierung; 2,00 Translation; 0,35 Toleranz

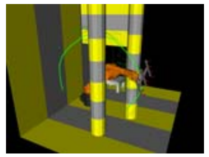
Modbus Debugger

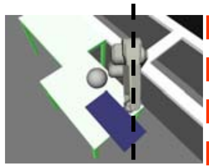
Port: COM3
Speed: 56700
Data bits: 8
Stop bits: 1
Parity: none



Development Status

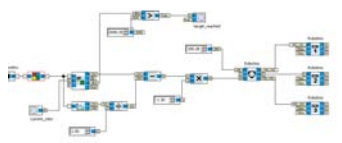
EasyLab & Framework Libs

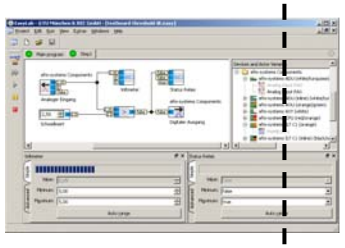




$$\begin{bmatrix} a_{i,j} & \cdots & a_{i,l} \\ \vdots & \ddots & \vdots \\ a_{k,j} & \cdots & a_{k,l} \end{bmatrix}$$

$$x + y \cdot i + z \cdot j + w \cdot k$$





- Open standards and specifications
- Reference implementations for
  - Models
    - Combined hardware model
    - Distributed systems
  - Verification
  - Deployment
- Domain-specific component libraries
  - Advanced image processing (tracking, …)
  - SLAM
  - Operational space control
- Overall integration of tools

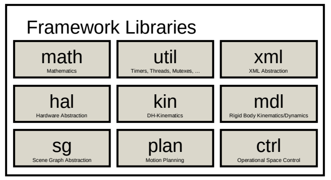
2004

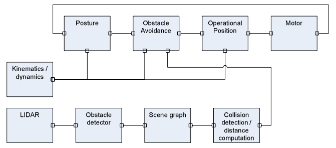
now

First full release: **1-November-2009**

Conclusion

- Architecture for integrated robotic development tool chain
- Model-driven approach
  - Comprehensive picture of a robot system:
    Physical properties and hardware-software interaction
  - Code generation
  - Simulation and verification
- Extensible architecture provides support at all layers
  - Framework libraries for efficient definition of new components
  - Pre-defined component libraries contain domain specific functionality
- Various programming paradigms provide the right level of abstraction
  - Conventional libraries as basic building blocks
    (mathematics, HAL, image processing, …)
  - Task-based programming
  - Behavior-based specification of high-level tasks
- Open source implementations that cover many areas of the presented architectures are available





Questions?

Further Information

- See ICRA CD for details on framework libraries
- See http://www6.in.tum.de/Main/ResearchEasyKit for EasyLab and http://www.easy-kit.de/ (German)
- See http://www.openrobotino.org/ and http://www.festo-didactic.com/

For first release see: http://www6.in.tum.de/

Selected Publications

- Michael Geisinger, Simon Barner, Martin Wojtczyk, and Alois Knoll. A software architecture for model-based programming of robot systems. In German Workshop on Robotics 2009 (GWR09), Braunschweig, Germany, June 2009.
- Simon Barner, Michael Geisinger, Christian Buckl, and Alois Knoll. EasyLab: Model-based development of software for mechatronic systems. In IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications, pages 540-545, Beijing, China, October 2008