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 → 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

Diagram showing a distributed system with connected sensors, a PC, and an embedded device.
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

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**
- Requirement Analysis
- High Level Design
- Detailed Specifications
- Code Generation
- Unit Testing, Formal Verification
- Simulation, Debugging, Formal Verification
- Operational Testing, Maintenance

**Components (based on framework)**
- Hardware Model, Application Model

© A. Knoll, S. Barner, M. Geisinger, M. Rickert. k@tum.de
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

Framework

HAL

Kinematics, Dynamics, Geometry

Hardware Model

Verification

Application Model

Simulation

Debugging

Code Generation

Deployment

Robots Technology Framework (RTF)

math

util

xml

hal

kin

mdl

gd

Scene Graph Abstraction

Motion Planning

Operational Space Control

HAL

Kinematics, Dynamics, Geometry

Verification

Application Model

Simulation

Debugging

Code Generation

Deployment

Code templates
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
Hardware Model

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
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 Libraries: Comprehensive Set of Functions

- Basic mathematics

\[ \dot{X}_A = \begin{bmatrix} E & 0 \\ -E r x & E \end{bmatrix} \]

\[ \begin{bmatrix} a_{i,j} & \cdots & a_{i,j} \\ \vdots & \ddots & \vdots \\ a_{k,j} & \cdots & a_{k,j} \end{bmatrix} \]

\[ x + y \cdot i + z \cdot j + w \cdot k \]

- Kinematics and dynamics

\[ \dot{x} = J \dot{q} \]

\[ \tau = M(q) \ddot{q} + V(q, \dot{q}) + G(q) \]

\[ \ddot{q} = M^{-1}(q) \left[ \tau - V(q, \dot{q}) - G(q) \right] \]

- Scene-graph abstraction / Collision detection

- Motion planning / Operational space control
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
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
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

VRLab

Free, but not open source
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
Development Status
EasyLab & Framework Libs

- 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

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)

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

Selected Publications