Model-Driven Software Systems Engineering in Robotics: Covering the Complete Life-Cycle of a Robot

Christian Schlegel, Alex Lotz, Matthias Lutz, Dennis Stampfer, Juan F. Ingles-Romero, Cristina Vicente-Chicote

16 Sept. 2013, Koblenz
Cooperative Robot Butler Scenario

http://www.servicerobotik-ulm.de/

http://www.youtube.com/user/RoboticsAtHsUlm

Service Robotics Ulm
autonomous mobile service robots

16.09.2013
Christian Schlegel, Alex Lotz et al.
Towards a Software Business Ecosystem

Robotics Business Ecosystem:
- Separation of Roles
- Separation of Concerns
- MDSD (Model-Driven SW Developm.)
Towards a Robotics Software Component Model

- **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

- **Think business ecosystem**:
  Share risks and efforts between different stakeholders, reduce costs and development time and increase robustness and quality of products

- **Think model driven**:
  Provide a black-box view for components with explicated services, properties and configurations
Component Builder

- Specialist with deep expertise (return-of-investment based on multiple use of components)
- "freedom from choice" in order to ensure system-level conformity

Component Development

- "inner view"
- "black box" view including model-based variation points

Application domain experts

System Integrator

- "outer view"
- uses black-box view
- make system-level bindings and adjustments

exploit variation points

Deployment

- Component must be bindable to execution platform at deployment time without recompilation!!
- purposefully left-open variability in order to deal with open-ended environments (models@runtime)

Deployment (add new components at run-time)

Robot

- exploit variation points

Design-Time

Run-Time

stepwise refinement: (1) add more and more information, (2) bind more and more variability
Communication Patterns*

not missing guidance inside components
but flexible interface consistent with outer semantics
to ease the job of the component 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 execution container

not variety outside where it affects system integration, but:

- stable and distinct communication characteristics for each communication pattern
- avoid complexity of combinatorial explosion of policies, mechanisms, etc.
- ensure system level conformance (avoid distributed system deadlocks, etc.)
- avoid incompatible port variants of the same service


Service Robotics Ulm
autonomous mobile service robots

16.09.2013 Christian Schlegel, Alex Lotz et al.
Model-Driven Software Development Toolchain
Robotics Software Component Model + MDSD

- component life-cycle
- monitoring/debugging
- separated internal interface from outer communication characteristics
- middleware abstraction

Metamodel

Service Robotics Ulm
autonomous mobile service robots

16.09.2013 Christian Schlegel, Alex Lotz et al. slide 8
MDSD Toolchain

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

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

**SmartTCL**
(Task Coordination Language)

**VML**
(Variability Modeling Language)

Domain Specific Languages

- Software component
- execution container

Meta-Model

Eclipse-Toolchain
- component developer

Eclipse-Toolchain
- system integrator
- deployment

Real robot in real world

How to get the coffee to the customer as hot as possible?

Service Robotics Ulm
autonomous mobile service robots

16.09.2013 Christian Schlegel, Alex Lotz et al.
System Architecture: Managing Execution Variants at Run-Time

- **SmartTCL:**
  Managing Variability in Task Sequencing

- **VML:**
  Managing Variability in Task Execution Quality
Sequencer orchestrates the system

The sequencing layer with SmartTCL:
- bridges between continuous processing and event-driven task execution
- orchestrates software components in the system
- assigns decision spaces to components
- involves dedicated expert components such as a symbolic planner for runtime bindings of designed variability
- uses a knowledge base to resolve symbols
- coordinates analysis, simulation and planning capabilities
SmartTCL

On the left:
(a) select between alternatives at run-time
(b) handle contingencies
(c) delete, add or replace sub trees at run-time

On the right: example for a refinement/expansion of the task “cleanup-table”
Non-Functional Properties at Run-Time

The robot needs to trade-off different non-functional properties such as **safety** and **performance** in order to select appropriate execution variants (in this case which coffee machine to use).
Variability Modeling Language (VML)*

**VML model**

- **Context Variables:** context a, b, ...
- **Properties:**
  - Property A: \( f(a) = \ldots \) influence on y
  - Property B: \( f(b) = \ldots \) influence on y
- **Variation Points:** varpoint x, y, ...
- **Adaptation Rules:**
  - cond a, cond b, ...
  - rule a
  - rule b
  - ...

**design-time:**
the designer provides the models (action plots with variation points to be bound later by the robot, policies for task fulfillment, problem solvers to use for binding of variability).

**run-time:**
the robot decides at run-time on proper bindings for variation points by applying the policies taking the current context into account.

---

System Architecture: Mapping Execution Variants at Run-Time

Integration of “Variability in Task Sequencing” and “Variability in Task Execution Quality”

(a) SmartTCL handles a contingency by exchanging a sub-tree

(b) SmartTCL uses a symbolic planner to refine a sub-tree

(c) VML as a service on demand

(d) VML as continuous service
Open Challenges and Future Work

- Enable designers to explicate the desired quality-of-service which the robot achieves at run-time by trading off different execution variants
- Extend the mechanisms for black-box handover from one role to another
- Link between S/W models (component settings, resources) and robot behavioral models (task nets) supported by MDSD approaches
- Improve the overall development workflow with different roles which refine the overall system model step by step
- Further improve the handover of knowledge and efforts between design-time and run-time
Thank you for your attention!

Any Questions?