Dealing with Run-Time Variability in Service Robotics: Towards a DSL for Non-Functional Properties

Juan F. Inglés-Romero, Alex Lotz, Cristina Vicente-Chicote, Christian Schlegel
How to improve the execution quality of a service robot acting in open-ended environments given limited onboard resources?

Example:

*Optimize coffee delivery service*

1. guarantee minimum coffee temperature (preference is to serve as hot as possible)
2. maximum velocity bound due to safety issues (hot coffee) and battery level
3. minimum required velocity depending on distance since coffee cools down
4. fast delivery can increase volume of coffee sales
Motivation

Focus so far in service robotics still mostly on:
• pure task achievement
• robot functionality
• how to do something

What cannot be ignored any longer:
• non-functional properties
  • quality of service
  • safety
  • energy consumption
  • …
• do it efficiently
  • which possibilities are better than others in terms of non-functional properties?
Motivation

**Robotics engineer / design-time**

- identify and enumerate all eventualities in advance???
- code proper configurations, resource assignments and reactions for all situations???

- **not efficient due to the combinatorial explosion of situations & parameterizations**
- **even the most skilled robotics engineer cannot foresee all eventualities**

**Robot / run-time:**

- just (re)plan in order to take into account latest information as soon as it becomes available???

- **complexity far too high when it comes to real-world problems**
  (not possible to generate action plots given partial information only while also taking into account additional properties like, e.g. safety and resource awareness)
Our Approach:
• Express variability at design-time
  • make it as simple as possible for the designer to express variability

• Bind variability at run-time based on the then available information
  • enable the robot to bind variability at run-time based on the then available information

• remove complexity from the designer by a DSL
• remove complexity from the robot’s run-time decision by modeling variability

We present:
• first version of a DSL to express variability in terms of non-functional properties
• integration into our robotic architecture
• real-world example
Our Approach

**Separation of concerns:**
- models (e.g. task net) describe *how* to deliver a coffee
- models specify *what is a good way (policy)* of delivering a coffee (e.g. in terms of non-functional properties like safety, energy consumption, etc.)

**Separation of roles:**
- designer at design-time: *provides* models
  - action plots with variation points to be bound later by the robot
  - policies for task fulfillment
  - problem solvers to use for binding variability
- robot at run-time: *decides* on proper bindings for variation points
  - apply policies
  - take into account current situation and context
Modeling Variability

Objective: Optimize service quality of a system (non-functional properties): power consumption, performance, etc.

- balance conflicting properties by minimizing overall cost function (constrained optimization problem)
  - property importance varies according to the current context → property priority
  - properties are expressed as functions of variation points → property definition

Inputs (context variables)
- the current robot state (task and resources)
- the environment situation

Outputs (variation point bindings)
- binding system variability (non conflicting with functionality)

adaptation rules:
- define direct relationships between context variables and variation points
- event-condition-action rules
- directly constrain the possible values of variation points according to current context

VML:
- Context definitions, variation point definitions, Properties and rules

05.11.2012
DSLRob 2012 / Schlegel
Modeling Variability

/* Data type definitions */
number percentType { range: [0, 100]; precision: 1; }
number velocityType { range: [100 600]; precision: 0.1; unit: "mm/s"; }

/* Contexts */
context ctx_battery : percentType;
context ctx_noise : percentType;

/* Adaptation rules */
rule low_noise : ctx_noise < 20 => speakerVolume = 35;
rule medium_noise : ctx_noise >= 20 & ctx_noise < 70 => speakerVolume = 55;
rule high_noise : ctx_noise >= 70 => speakerVolume = 85;

/* Properties */
property efficiency : percentType maximized {
  priorities: \( f(\text{batteryCtx}) = \max(\exp(-\text{batteryCtx}/15)) - \exp(-\text{batteryCtx}/15) \);
  definitions: \( f(\text{maxVelocity}) = \text{maxVelocity} \);
}

property powerConsumption : percentType minimized {
  priorities: \( f(\text{batteryCtx}) = \exp(-1 * \text{batteryCtx}/15) \);
  definitions: \( f(\text{maxVelocity}) = \exp(\text{maxVelocity}/150) \);
}

/* Variation points */
varpoint maximumVelocity : velocityType;
varpoint speakerVolume : percentType;
Execution Semantics

• M2M transformation from VML model into MiniZinc model
  • MiniZinc is currently supported by many constraint solvers
    • context variables => parameters
    • variation points => decision variables
    • adaptation rules / variation point dependencies => constraints
    • properties => cost function
  • we use
    • The G12 Constraint Programming Platform — University of Melbourne

\[ f(\text{ctx}, \text{vp}) = \sum_{\forall i} (-1)^{d_i} \cdot w_i(\text{ctx}) \cdot p_i(\text{vp}) \]
Integration into robotic architecture

Sequencer (SmartTCL: Task Coordination Language)

- Agenda Task-Net
- Knowledge Base
  - Task Net Library
  - Rules
  - Event Handler
  - Environment Model
  - Component Model

Interface: online reconfiguration (state, parameter, dynamic wiring); event/query management

Symbolic Planner
- e.g. Metric-FF, FF, LAMA

Constraint Solver
- e.g. solvers for MiniZinc, FlatZinc representations

Real-Time Analysis
- Checker

Simulation
- e.g. Gazebo, MORSE

Motion Planning
- Open Rave
Conclusions & Future Work

- VML enables designers to focus on modeling the adaptation strategies without having to foresee and explicitly deal with all the potential situations that may arise in real-world and open-ended environments.

- The variability, purposefully left open by the designers in the VML models, is then bound by the robot at run-time according to its current tasks and context (separation of roles and separation of concerns).

- We underpinned the applicability of our approach by integrating it into our overall robotic architecture and by implementing it in a sophisticated real-world scenario on our service robot Kate.

- For the future, we fully integrate VML into our SmartSoft MDSD toolchain.
Overall Vision: MDSD in Robotics...

- Use models for the entire life-cycle of the robot
- Models are refined step-by-step until finally they become executable
- Separate inside view (component builder) from outside view (system integrator)
- Separate stable execution container from implementational technologies (middleware, OS)
- Variation points: design-time (component builder, system integrator), runtime (robot)
  - Explicitly model variability for late binding (by system integrator and even by the robot at runtime)