Poster: Model-based Run-time Variability Resolution for Robotic Applications

Luca Gherardi* and Nico Hochgeschwender†,
*Institute for Dynamic Systems and Control
ETH Zurich, Switzerland
†Department of Computer Science,
Bonn-Rhein-Sieg University, Sankt Augustin, Germany

***Abstract*—In this paper we present our ongoing work on *Robotics Run-time Adaptation (RRA)*. RRA is a model-driven approach that addresses robotics runtime adaptation by modeling and resolving run-time variability of robotic applications.**

I. Introduction and Motivation

The development of robotics applications that are executed in possibly partially unknown and dynamic environments is a challenging task. In fact, it requires the developers to cope with a huge amount of variability, which relates to the software functionalities, the robot hardware, and the environment. This variability greatly affects the architecture design: two different selections of variants for the same set of variation points typically lead to two different architecture configurations. Even tough robotics software variability can be partially addressed by integrating Software Product Lines techniques with robotics-specific meta-models and tools (as demonstrated in our previous works [1] [2]), two main problems are still open. First, some variation points cannot be addressed at deployment-time because their resolution requires information about the state of the robot and the environment (from now on called context), which is available only at run-time. For example, the choice of the obstacle avoidance algorithm can be better performed at run-time, when the density and the dynamics of the obstacles are known. Second, not all the variants of a specific variation point can be foreseen at design-time or in other case they are too many (possibly infinite) to allow their enumeration.

These issues lead to the need of extending our previous work to (a) resolve only part of the variation points at deployment-time, (b) postpone the resolution of the remaining variation points at run-time, (c) continuously reason about the run-time variability to choose the configuration (i.e. variants selection) that best suits the context and provides the best QoS. According to this view the resolution of the deployment-time variability allows the deployment of a specific application (i.e. architecture configuration) belonging to the product line, whereas the resolution of the run-time variability allows the dynamic adaptation of the application architecture with the objective of addressing the changes in the context.

In what follows, we outline some of the key elements of *Robotics Run-time Adaptation (RRA)*, our model-driven approach for modeling and resolving robotics run-time variability. The contribution of this work are: (a) a set of robotics-specific meta-models that explicitly define the run-time variability and the rules for its run-time resolution, (b) a set of algorithms and tools that reason about the aforementioned information to adapt the software architecture, (c) the integration of the proposed approach with our previous works for the resolution of deployment-time variability [1], and (d) the evaluation of the approach by means of a robotics specific case study. The meta-models, the algorithms, and the tools are openly available with the *RRA Toolchain* (`https://github.com/lucaghera/RRA`).

II. Model-based Run-time Variability Resolution

Figure 1 shows how the run-time variability resolution approach presented in this paper (right part) has been integrated with the deployment-time approach presented in [1] (left part). The run-time resolution is designed as a feedback loop, which is continuously executed and includes the following steps.

A. Run-time Resolution Initialization

The first step is the initialization, which is based on three inputs. The first one is the *Deployment-time Configured System Model (DCSM)*. It is the output of the deployment-time variability resolution and represents the run-time architecture template. The DCSM specifies the set of components that can be deployed throughout the application execution and the initial connections between them. At run-time this template is continuously reconfigured (e.g. activating and deactivating components, changing connections, setting configuration parameters) to provide the configuration that best suits the context. The reconfiguration of the *DCSM* is stored into the *Run-time Configured System Model (RCSM)*, which reflects the software architecture running the robot control system. Any change made to this model is replicated on the software architecture. The second input is the *Feature Model (FM)* [3]. It represents the functional variability of the application and contains deployment-time and run-time features, which differ for the binding time of the variation points encoded with the features. Finally the last input is the selection of *deployment-time* features that was used for the deployment-time variability resolution and that led to the DCSM.

B. Context Monitoring

The second step is performed by the *Context Monitor (CM)* and consists of collecting *Context Dependent Measurements*

978-1-4799-1934-5/15 $31.00 © 2015 IEEE
DOI 10.1109/ICSE.2015.269

(CDMs), i.e. measurements the value of which depends on the state of the context (e.g. obstacles velocity). The context monitor is the result of a model-to-text transformation, which is based on the *Context Dependent Measurements Model (CDMM)*. Robotics systems use CDMs to take important decisions on the next actions. The acquisition of CDMs is typically hard-coded in the components that reason about them. We claim that explicitly modeling this information and having a dedicated component for its collection allows for a higher reusability of the components that use the information.

C. Run-time Feature Selection

The third step is performed by the *Adaptation Engine (AE)*, which reasons about the context in which the robot is operating and produces the system configuration that best suits it. *CDMs* are one of the three inputs of this component. The second input are the *Context Independent Measurements (CIMs)*, measurements the value of which is always the same and do not depend on the context. The third input are the *Non-Functional Requirements (NFRs)*, which express constraints that must be respected in order to achieve a goal. We propose to model CIMs and NFRs in the *Feature Model* by enriching features for which the developer wants to specify *CIMs* or *NFRs* with attributes. To achieve this goal we extended the Feature meta-model presented in [1]. Each attribute specifies four properties a name, a data type, and a value. In addition *NFR* attributes also specify one a the following tags: min, max, avg, and count (e.g. max indicates that the attribute value is the maximum allowed for the NFR represented by the attribute).

These three inputs are used by the *AE* to produce the more appropriate selection of run-time features. The logic of the *AE* is modeled by means of the *Adaptation Model (AM)*, which specifies how context measurements trigger a set of actions that generate the *AE* output. The *AM* is defined in terms of a set of atomic rules, which consist of a name, a condition under which the rule is fired, and an action. Rule conditions compose *CDMs*, *CIMs*, and *NFRs* by means of logical, relational and math operators. The possible actions are feature selection and feature de-selection. We defined a Domain Specific Language for the *AM* and integrated it in an Eclipse plugin.

D. Run-time Architecture Adaptation

The fourth step is performed by the *Run-time Resolution Engine (RRE)*, which uses the selection of run-time features produced in the previous step to adapt the system's architecture. This component is configured by means of the *Run-time Resolution Model (RRM)*, which defines how the architectural variability of *RCSM* has to be resolved according to the output of the *AE*. The *RRM* associates to each run-time feature a set of transformations (e.g. changing the connections between components' inputs and outputs), which have to be performed when the feature is selected. The *RRE* applies the transformations not only to the architectural model (*RCSM*) but also to the architecture of the running system. This is done by invoking an adaptation API, which provides functions to execute the transformations on the running architecture.

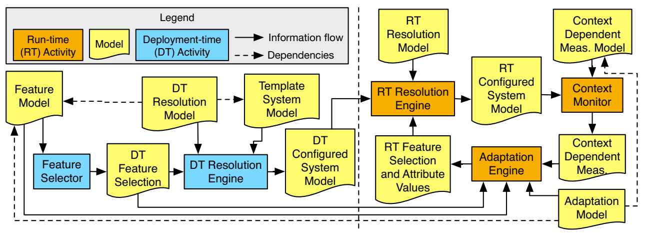

Fig. 1. An overview of the model-based variability management approach.

III. Implementation and Evaluation

In order to evaluate the approach we implemented a simple case study where a simulated quadrocopter flies along a circular trajectory. As shown in [4] a quadrocopter can fly also when some of its motors break. This can be achieved by replacing the standard vehicle controller with a specific *Proploss* controller, which allows the vehicle to recover and then follow the desired trajectory. To switch the controller and adapt the architecture, we have implemented the RRA's approach described here. Based on sensor data (i.e. expected and measured current) the *CM* infers the number of motors that are working properly (CDM). The *AE* uses this CDM to resolve a variation point made of two variants (features `Proploss` selected or not). Based on this selection the *Runtime Resolution Engine* adapts the architecture as follows. When the feature `Proploss` is not selected, the architecture is not modified. Otherwise the controller component is replaced with the *Proploss*. The case study has been executed on a standard laptop and the adaptation was performed in 0.5 seconds. This was the time required to process the context information, stop the standard controller and launch the *Proploss* controller. The adaptation is fast enough to recover a flying vehicle, which is a time-critical operation. A brief video of the case study is available at `lucagherardi.it/RRA`.

IV. Conclusion

We presented RRA, our model-based approach for run-time adaptation of robotics system. As RRA does not modify the run-time architecture directly, adaptation design is abstracted from the architectural level to the functional level. This allows developers to decouple the adaptation rules from the architecture. Additionally, the models describing the architecture, the functional variability, and the adaptation rules are orthogonal (i.e. they don't depend on each other). Therefore developers can easily change one of these models without affecting the others.

References

[1] L. Gherardi, "Variability modeling and resolution in component-based robotics systems," Ph.D. dissertation, University of Bergamo, 2013.
[2] L. Gherardi and D. Brugali, "Modeling and Reusing Robotic Software Architectures: the HyperFlex toolchain," in *International Conference on Robotics and Automation (ICRA)*, 2014.
[3] K. Kang, "Feature-oriented domain analysis (FODA) feasibility study," DTIC Document, Tech. Rep., 1990.
[4] M. W. Mueller and R. D'Andrea, "Stability and control of a quadrocopter despite the complete loss of one, two or three propellers," in *International Conference on Robotics and Automation (ICRA)*, 2014.