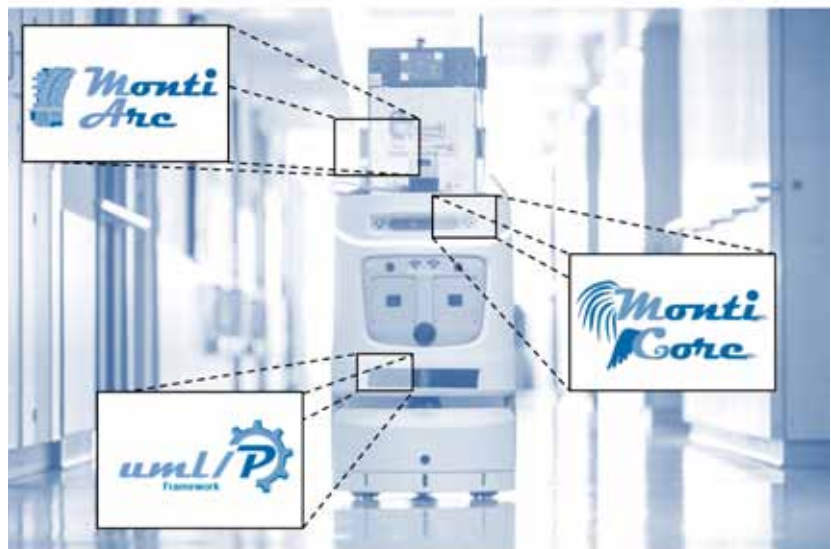


Kai Adam, Arvid Butting, Robert Heim,
Oliver Kautz, Jérôme Pfeiffer,
Bernhard Rumpe, Andreas Wortmann

Modeling Robotics Tasks for Better Separation of Concerns, Platform- Independence, and Reuse

Results of the iserveU Federal
Research Project



Aachener Informatik-Berichte,
Software Engineering

Band 28

Hrsg: Prof. Dr. rer. nat. Bernhard Rumpe

Aachener Informatik-Berichte, Software Engineering

herausgegeben von
Prof. Dr. rer. nat. Bernhard Rumpe
Software Engineering
RWTH Aachen University

Band 28

**Kai Adam, Arvid Butting, Robert Heim, Oliver Kautz,
Jérôme Pfeiffer, Bernhard Rumpe, Andreas Wortmann**
RWTH Aachen University

Modeling Robotics Tasks for Better Separation of Concerns, Platform-Independence, and Reuse

Results of the iserveU Federal Research Project

Shaker Verlag
Aachen 2017

Bibliographic information published by the Deutsche Nationalbibliothek

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at <http://dnb.d-nb.de>.

Copyright Shaker Verlag 2017

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the publishers.

Printed in Germany.

ISBN 978-3-8440-5319-7

ISSN 1869-9170

Shaker Verlag GmbH • P.O. BOX 101818 • D-52018 Aachen

Phone: 0049/2407/9596-0 • Telefax: 0049/2407/9596-9

Internet: www.shaker.de • e-mail: info@shaker.de

This research has partly received funding from the German Federal Ministry for Education and Research under grant no. 01IM12008C. The responsibility for the content of this publication is with the authors.

Abstract

Deploying robotics applications requires expertise from multiple domains, including general software engineering and the application domain itself. Consequently, successful robotics applications are developed by teams of software experts, robotics experts, and application domain experts. The conceptual gap between application domain challenges and implementation domain solutions gives rise to accidental complexities from solving problem domain challenges with programming domain details. This complicates development and may lead to failure. Domain and robotics experts are rarely software engineering experts. Their involvement into the software engineering of reusable robotics applications requires that they become software experts or that implementation details can be abstracted.

Model-driven engineering reduces the conceptual gap by leveraging models to primary development artifacts. Models usually are more abstract than programming language artifacts and enable to use robotics vocabulary or application domain vocabulary. This supports domain experts in formulating solutions in established known vocabulary. Model transformations can embody the software engineering expertise required to translate such models into robust and reusable programming language solutions. Different target technologies can be addressed by different transformations, which decouples logical robotics tasks from heterogeneous platforms and ultimately produces deployable solutions. Proper separation of expert concerns is crucial to enable transformations and ultimately improve engineering of robotics applications. Component-based software engineering has proven useful for the integration of domain-specific solutions and system extensibility. Here, software components encapsulate functionality in reusable black boxes. That these components usually are programming language artifacts gives rise to accidental complexities again. Models of architecture description languages lift components to the model level. Component models can serve as building blocks of complex systems and can be translated into implementations for different target technologies as well. Combining model-driven engineering with component-based software engineering enables to provide robotics experts and domain experts with appropriate modeling languages, as well as software engineering experts with means to decouple their concerns while ensuring integration of their solutions.

We present a collection of domain-specific languages to describe service robotics applications. They enable formulating domains, tasks, actions, and properties of a robot and its environment free of GPL complexities. Their models are translated into component implementations of a MontiArcAutomaton [RRRW15] software architecture model and interpreted at system runtime. The architecture executes tasks via their transformation to Planning Domain Definition Language models, solves these, and executes the resulting plans with a robotics middleware. Leveraging separation of concerns and abstraction of modeling languages, this reduces the effort of describing robot tasks, facilitates extension

of the system with new components, and decouples logical task solving from the robot platform. This supports integration of domain experts and reuse of infrastructure constituents in different contexts and with different platforms.

Contents

1	Introduction	1
2	Methodology	3
3	Example	7
4	Modeling Languages	11
4.1	Application Language	11
4.1.1	Language Elements	11
4.1.2	Well-formedness Rules	12
4.2	Domain Modeling	12
4.3	Entity Language	12
4.3.1	Language Elements	13
4.3.2	Well-formedness Rules	14
4.4	Task Language	15
4.4.1	Language Elements	15
4.4.2	Well-formedness Rules	16
4.5	Goal Language	16
4.5.1	Language Elements	17
4.5.2	Well-formedness Rules	18
4.6	Discussion	18
5	System Architecture	21
5.1	Component & Connector Architecture	22
5.2	Reference Architecture	23
5.2.1	Remote Operator	24
5.2.2	Task Processor	24
5.2.3	Data Types	27
5.2.4	Controller Component	28
5.2.5	Planner Component	32
5.2.6	InitialStateProvider Component	32
5.2.7	PlanVerifier Component	33
5.2.8	ActionExecuter Component	33
5.2.9	PropertyCalculator Component	34
5.3	Runtime Environment	34
5.3.1	Model Realization Base Classes	34
5.3.2	Robot and World Interfaces	35

5.3.3	System-Wide Logging	36
5.4	Generators for iserveU Models	36
5.5	Application-Specific Architecture Parts	39
5.5.1	SmartSoftUsageDeployer Component	40
5.5.2	ResponseListener Component	40
5.5.3	Robot and World Implementation	40
5.5.4	SmartSoft Communicator	41
5.5.5	Communication Protocol	41
5.5.6	Knowledge Base	42
5.5.7	Architecture Starter	43
5.6	Exemplary Execution Excerpts	44
5.6.1	Setting	44
5.6.2	Successful Deliver	44
5.6.3	Path Temporarily Blocked	46
5.6.4	Path Permanently Blocked, Remote Operator Successful	47
5.6.5	Path Permanently Blocked, Remote Operator Not Successful	48
5.6.6	Abort via Desktop UI	48
5.6.7	Abort via Tablet UI	51
5.7	Discussion	52
6	Translating Domain Expert Models to Robot Plans	53
6.1	Intra-Language Model-to-Model Transformations	54
6.2	Transformations to PDDL	56
6.2.1	Transformation of CD Models to PDDL Types and Predicates	56
6.2.2	Transformation of Entity Properties to PDDL Predicates	56
6.2.3	Transformation of Entity Actions to PDDL Actions	58
6.2.4	Transformation of Goals to PDDL Problems	60
6.3	Discussion	62
7	Human-Robot Interaction	65
7.1	Desktop User Interface	65
7.2	Tablet User Interface	69
7.2.1	Activities in Different Scenarios	69
7.2.2	Tablet Communication	74
8	Deployment	77
8.1	Deployment Scenario	77
8.1.1	Logistics Domain	79
8.1.2	Rooms World Entity	79
8.1.3	Transport Robot Entity	79
8.1.4	Transport Tasks	80
8.1.5	Transport Goals	82
8.1.6	Integrating Handcrafted Code	84

8.2	Starting SmartSoft	85
8.3	Deployment of Architecture and User Interfaces	85
8.3.1	Starting the iserveU Server	86
8.3.2	Deploying Reference Architecture and Desktop UI	86
8.3.3	Starting the Tablet User Interface	87
8.4	Generalizing the Infrastructure	87
8.4.1	Adapting Models	88
8.4.2	Transfer to Other Platforms	90
9	Related Work	93
9.1	Modeling Languages	93
9.2	Integrating Modeling Languages with Planning	94
9.3	Robotics Reference Architectures	94
10	Conclusion	95
	Bibliography	97
	List of Figures	103
	List of Listings	106
	Index of Abbreviations	109