SE | RWTH AACHEN UNIVERSITY

RWTH Aachen University
Software Engineering Group

MontiCore 5
Language Workbench

Edition 2017

Bernhard Rumpe,
Katrin Hölldobler

http://www.se-rwth.de/
http://www.monticore.de/

Aachener Informatik-Berichte,
Software Engineering Band 32

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

RWTH Aachen University
Software Engineering Group

MontiCore 5 Language Workbench

Edition 2017

Bernhard Rumpe,
Katrin Hölldobler

http://www.se-rwth.de/
http://www.monticore.de/

Aachener Informatik-Berichte, Software Engineering

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

Band 32

RWTH Aachen University Software Engineering Group

Bernhard Rumpe / Katrin Hölldobler
(Herausgeberische Koordinierung)
RWTH Aachen University

MontiCore 5 Language Workbench

Edition 2017

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 978-3-8440-5713-3
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

Foreword

MontiCore is a language workbench, which is developed since 2004. We have started its development because at that time the available tools for model management where often very poor in its functionalities and also not extensible, but closed shops. In 2004 the first version of the UML/P was published (and is now available as [Rum16, Rum17]) demonstrating that the conglomerate of languages that the UML is made of can be substantiated with useful transformation, refinement and refactoring techniques. Code and test code generation as well as flexible combination of language fragments, such as OCL within Statecharts or Class Diagrams for typing in Component and Connector Diagrams, were the techniques of primary interest. However, hard coded modeling tools where not helpful in realizing these techniques. This was the original motivation for MontiCore that can also be found in the foundational theses in [Kra10, Völ11].

Later, it became apparent that UML will be complemented by several domain specific languages (DSLs) that will be connected to software development or execution in various ways. The definition of DSLs encounters the same difficulties as the definition of the UML faced, i.e., they are often built from scratch, reuse is pretty bad, the same concepts get different syntactic shapes. Thus, combining DSLs is rather impossible. We therefore extended the focus of MontiCore to become a general language workbench that allows to define languages and language fragments and to derive as much as possible from an integrated and therefore compact definition.

In this version of the MontiCore Tutorial and Reference Manual the core facilities of MontiCore are described. Extensions are available through various projects either using or enhancing MontiCore. We have a sophisticated technique to generate transformation languages and their transformation engines based on DSLs [HRW15, AHRW17, RRW15, HHRW15, Wei12], we have explored tagging languages [Loo17, MRRW16, GLRR15], various forms of the UML and its derivatives [Sch12, Wor16, Hab16, Rei16, Rot17] and plenty specific languages.

Despite MontiCore is an academic tool to explore modeling and meta-modeling techniques, after 14 years of development, it has reached an extraordinary strength and is thus increasingly used in industrial projects, like energy management [Pin14], as well as in scientific projects of entirely different nature, such as simulation of city scenarios for autonomous driving [Ber10] or human brain modeling [PBI+16]. MontiCore, however, does not primarily focus comfort, e.g., graphical editing, but advanced functionality for model-based analysis or synthesis of software intensive systems as well as quick textual editing for experienced users.

We would like to thank all current and former members of our group as well as all students and apprentices who helped to develop MontiCore in its current shape. Namely, we would

like to thank Kai Adam, Professor Dr. Christian Berger, Vincent Bertram, Arvid Butting, Anabel Derlam, Imke Drave, Robert Eikermann, Timo Greifenberg, Dr. Hans Grönniger, Dr. Tim Gülke, Dr. Arne Haber, Guido Hansen, Olga Haubrich, Lars Hermerschmidt, Dr. Christoph Herrmann, Gabi Heuschen, Steffen Hillemacher, Oliver Kautz, Carsten Kolassa, Dr. Anne-Therese Körtgen, Thomas Kurpick, Evgeny Kusmenko, Dr. Holger Krahn, Dr. Stefan Kriebel, Achim Lindt, Dr. Markus Look, Professor Dr. Shahar Maoz, Matthias Markthaler, Dr. Klaus Müller, Dr. Pedram Mir Seyed Nazari, Antonio Navarro Pérez, Nina Pichler, Dr. Claas Pinkernell, Dr. Dimitri Plotnikov, Deni Raco, Dr. Holger Rendel, Dr. Dirk Reiss, Dr. Daniel Retkowitz, Dr. Alexander Roth, Dr. Martin Schindler, David Schmalzing, Steffi Schrader, Dr. Frank Schroven, Christoph Schulze, Igor Shumeiko, Brian Sinkovec, Simon Varga, Dr. Steven Völkel, Dr. Ingo Weisemöller, Michael von Wenckstern, and Dr. Andreas Wortmann. The individual contributions to MontiCore and its derivatives resulted in numerous publications[1]. Special thanks go to Marita Breuer and Galina Volkova, who maintain and extend MontiCore, as well as to Sylvia Gunder, who ensures that all financial and project activities supporting our language workbench project MontiCore are running perfectly.

We also would like to thank the authors or co-authors of several chapters, who partially described their contribution directly in this reference manual.

In addition, we would like to thank all reviewer for their intense reviews of certain chapters (chapters are mentioned in brackets): Kai Adam (Ch. 1,9,14,19), Vincent Bertram (Ch. 15,17), Arvid Butting (Ch. 7,9), Imke Drave (Ch. 1,2,3,4,12,16), Robert Eikermann (Ch. 7,8,9,10), Timo Greifenberg (Ch. 3), Guido Hansen (Ch. 4,6,13), Olga Haubrich (Ch. 2,17), Steffen Hillemacher (Ch. 5,12), Oliver Kautz (Ch. 3,6,9,11), Evgeny Kusmenko (Ch. 7), Achim Lindt (Ch. 2,8), Nina Pichler (Ch. 18), Deni Raco (Ch. 7,10), David Schmalzing (Ch. 3,6,11), Christoph Schulze (Ch. 15,19), Igor Shumeiko (Ch. 5,12), Brian Sinkovec (Ch. 9,18), Simon Varga (Ch. 4,6,13), Michael von Wenckstern (Ch. 3,13,14) and Dr. Andreas Wortmann (Ch. 7,9).

To all readers of this manual: We hope you enjoy reading this manual and trying out our language workbench MontiCore as well as the tools generated with MontiCore. In case you have any suggestions or questions do not hesitate to contact us.

Aachen, 23.12.2017

Katrin Hölldobler, Bernhard Rumpe

[1] www.se-rwth.de/publications/

Contents

1 Introduction **1**
1.1 MontiCore Language Workbench . . . 2
1.2 Notational Conventions . . . 3
1.3 Textual Modeling . . . 4
1.4 Methodical Considerations: Agile Modelling . . . 5

2 Getting Started **7**
2.1 Prerequisites . . . 7
2.2 How to Install MontiCore for Command Line . . . 8
2.2.1 Installation . . . 8
2.2.2 Run MontiCore . . . 8
2.2.3 Compile and Run the Target . . . 9
2.2.4 MontiCore Commandline Arguments . . . 10
2.3 Using MontiCore in Eclipse . . . 11
2.3.1 Setting up Eclipse . . . 11
2.3.2 Importing the Example . . . 12
2.3.3 Running MontiCore . . . 12
2.4 Using MontiCore in IntelliJ IDEA . . . 13
2.4.1 Setting up IntelliJ IDEA . . . 13
2.4.2 Importing the Example . . . 14
2.4.3 Running MontiCore . . . 14

3 Architecture of a Model Processor **17**
3.1 Structure of a Model Processor - External View . . . 17
3.2 Internal Architecture of a Generator . . . 18
3.3 Tool Workflow On The Top Level . . . 19

4 MontiCore Grammar for Language and AST Definitions **23**
4.1 Lexical Tokens . . . 24
4.1.1 Definition of Tokens using Regular Expressions . . . 25
4.1.2 Actions to Process a Token . . . 26
4.1.3 Predefined Tokens . . . 27
4.2 Productions . . . 29
4.2.1 Terminals . . . 31
4.2.2 Enumeration . . . 31
4.2.3 Nonterminals . . . 32
4.2.4 Interface Nonterminals . . . 32
4.2.5 Extending Nonterminals . . . 33
4.2.6 Abstract Nonterminals . . . 34

4.2.7 Starting Nonterminal . . . 35
4.2.8 Infix Operations and Priorities . . . 35
4.2.9 Restricting the Cardinality of a Nonterminal . . . 37
4.2.10 Symbols and Scopes . . . 37
4.2.11 Passing Code to the ANTLR . . . 38
4.3 Context Conditions for the MCG Language . . . 39
4.4 Semantic Predicates and Actions . . . 44
4.5 Editor assistance for the MCG Language . . . 44
4.6 EBNF for the MCG Language . . . 44

5 Abstract Syntax Tree 51
5.1 Mapping Nonterminals to the AST . . . 51
5.2 Interface and Abstract Nonterminals . . . 52
5.3 Extending Nonterminals . . . 54
5.4 Extending the Abstract Syntax Implementation . . . 55
5.5 Terminals in the AST . . . 57
5.6 Enumerations . . . 58
5.7 ASTNode: A Base Interface for AST Classes . . . 58
5.8 Generated ASTNode Subclasses . . . 61
5.9 Node Construction Using the Node Builder Mill . . . 66
5.10 Handwritten Extension of AST Classes and Node Builders . . . 70
5.10.1 Handwritten Extension of AST Classes . . . 70
5.10.2 Handwritten Extension of AST Builders and Mills . . . 72

6 Parser Generation and Use 75
6.1 Generating a Parser and a Lexer . . . 75
6.2 Interface of the Generated Parser Classes . . . 78
6.3 Executing a Generated Parser . . . 79

7 Language Composition 81
7.1 Introduction to Language Composition . . . 81
7.2 Language Composition at a Glance . . . 84
7.3 Grammar Constructs for Language Composition . . . 87
7.3.1 Component Grammar . . . 88
7.3.2 External Nonterminals . . . 88
7.3.3 Importing and Extending Grammars . . . 90
7.4 Language Inheritance . . . 90
7.4.1 Redefining Productions of Grammars . . . 91
7.4.2 Extending Multiple Grammars . . . 93
7.5 Language Embedding . . . 94
7.6 Composing the Builder Infrastructure . . . 95
7.7 Composing Parsers . . . 96
7.8 Conservative Extension . . . 97
7.8.1 Conservative Extension of the Concrete Syntax . . . 98
7.8.2 Access-Conservative Extension of the Abstract Syntax . . . 99
7.8.3 Modification-Conservative Extension of the Abstract Syntax . . . 100

7.8.4 AST Signatures Causing Java Type Errors 101

8 Visitors for AST Traversal 103
8.1 Visitor Interface of a Language 103
8.2 Predefined Visitor Variants 107
8.2.1 Inheritance Visitor 107
8.2.2 Delegator Visitor 109
8.2.3 Parent Aware Visitor 109
8.3 Visitors for Composed Languages 110
8.3.1 Visitor for Language Inheritance and Extension 110
8.3.2 Visitor for Language Inheritance – an Alternative 113
8.3.3 Visitors for Compositional Language Embedding 114

9 Symbol Management Infrastructure 121
9.1 Introduction to Symbol Table Concepts 121
9.2 Defining Symbols 123
9.2.1 Generated Classes For Symbols 124
9.2.2 RTE Classes For Symbols 125
9.3 Defining Scopes 127
9.3.1 Generated Classes For Scopes 127
9.3.2 RTE Classes For Scopes 128
9.3.3 Artifact Scope and Global Scope 129
9.4 Collaboration between AST, Symbol, and Scope 130
9.5 Using Symbols 130
9.6 Instantiating Symbol Tables 131
9.6.1 RTE Classes For Symbol Table Building 132
9.6.2 Generated Classes For Symbol Table Building 132

10 Context Conditions 133
10.1 Context Condition Infrastructure 134
10.2 Implementation of Context Conditions 137
10.3 Testing Context Conditions 139
10.3.1 Testing a Context Condition on a Valid Model 139
10.3.2 Testing a Context Condition on an Invalid Model 140

11 Design Pattern Used and Invented for MontiCore 143
11.1 Static Delegator Design Pattern 143
11.2 RealThis Object Composition Pattern 145
11.3 Template Hook Pattern 147

12 FreeMarker 149
12.1 The FreeMarker Template Languages 149
12.2 Expressions in FreeMarker 151
12.3 Control Directives in FreeMarker 152
12.4 FreeMarker Drawbacks 154

13 Generator Engine using Flexible Templates **155**
13.1 Methodical Considerations . . . 155
13.2 Generator API . . . 157
13.3 Configuring the Generation Process . . . 159
13.4 MontiCore APIs for Templates . . . 162
13.4.1 Shortcuts: Aliases in Templates . . . 162
13.4.2 The Template Controller . . . 163
13.4.3 Logging within a Template . . . 169
13.4.4 Variables in the Templates with GlobalExtensionManagement . . . 169
13.5 Hook Points for Adaptation . . . 171
13.5.1 The Concept of Hook Points . . . 171
13.5.2 Forms of Hook Points . . . 174
13.5.3 Defining Explicit Hook Points in Templates . . . 177
13.5.4 Binding Hook Points . . . 178
13.5.5 Replacing and Decorating Hook Points . . . 179
13.5.6 HookPoint Replacement and Decoration Strategy . . . 180

14 Integrating Handwritten Code **183**
14.1 Integration of Handwritten Code . . . 183
14.2 Adaptation of Generated Code by Subclassing . . . 184
14.3 Adaptation of Generated Code using the TOP Mechanism . . . 185

15 Error Handling, Logging and Reporting **189**
15.1 Where to find Concrete Help for an Error, Warning, or other Message . . . 189
15.2 Errors, Warnings and Log Messages . . . 189
15.2.1 Errors . . . 190
15.2.2 Warnings and Information . . . 191
15.2.3 Form of Errors, Warnings and Log Messages . . . 191
15.3 The Error and Logging Component . . . 193
15.4 Logging Configurations in MontiCore . . . 195
15.4.1 Selecting one of the given Configurations . . . 196
15.4.2 Using a Custom logback Configuration . . . 197
15.4.3 Initializing the Log within Java . . . 197
15.4.4 Providing a Custom Log Implementation . . . 197
15.5 Reports . . . 198
15.5.1 Where to Find Reports . . . 198
15.5.2 How to Configure Reporting . . . 198
15.5.3 Identifiers contained in the Reports . . . 199
15.5.4 List of the Reports . . . 201
15.6 For Developers: How to Deal with Errors and Warnings . . . 204

16 MontiCore Configuration with Groovy **205**
16.1 Generate the Standard Classes . . . 206
16.2 MontiCore Base Class for Groovy Scripts . . . 208
16.2.1 Methods Available within Groovy Scripts . . . 209
16.2.2 Variables Available within Groovy Scripts . . . 210

16.2.3 Available preimported Classes within Groovy Scripts 211

17 Some MontiCore Grammars Explained 213
17.1 Component Grammar MCBasics.mc4 213
17.2 Component Grammar StringLiterals.mc4 215
17.3 Component Grammars for Numbers 217
17.3.1 Component Grammar MCNumbers.mc4 217
17.3.2 Component Grammar MCHexNumbers.mc4 219
17.4 Component Grammar MCBasicTypes1.mc4 220
17.5 Component Grammars for UML Languages 221
17.5.1 Component Grammar UMLStereotype.mc4 221
17.5.2 Component Grammar Cardinality.mc4 222
17.5.3 Component Grammar Completeness.mc4 223
17.5.4 Component Grammar UMLModifier.mc4 223
17.6 Component Grammar MCExpressions.mc4 224

18 Some Example Languages 229
18.1 A Simple Automaton Language 229
18.2 Hierarchical Automaton 235
18.3 A Language for Automata with Invariants 236
18.4 Scannerless Parsing to Handle Complex Tokens 238
18.4.1 Parsing with Whitespaces 239
18.4.2 Temporarily Parsing with Whitespaces 240
18.4.3 Preventing Whitespaces between Tokens 241
18.5 Tip: Testing Grammars and their Models 242
18.6 Questionnaire Language 244

19 Developer's View on MontiCore 247
19.1 MontiCore's GitHub Repository 248
19.1.1 External Developers and Forks 249
19.1.2 MontiCore's Maven Projects 250
19.2 Other Source Code Locations 251

20 Further Reading and Related Work from the SE Group, RWTH Aachen 253
20.1 Agile Model Based Software Engineering 253
20.2 Generative Software Engineering 253
20.3 Unified Modeling Language (UML) 254
20.4 Domain Specific Languages (DSLs) 254
20.5 Software Language Engineering 254
20.6 Modeling Software Architecture & the MontiArc Tool 255
20.7 Compositionality & Modularity of Models 255
20.8 Semantics of Modeling Languages 256
20.9 Evolution & Transformation of Models 256
20.10 Variability & Software Product Lines (SPL) 256
20.11 Cyber-Physical Systems (CPS) 257
20.12 State Based Modeling (Automata) 257
20.13 Robotics 258

20.14 Automotive, Autonomic Driving & Driver Assistance 258
20.15 Energy Management 258
20.16 Cloud Computing & Enterprise Information Systems 259

List of Figures **261**

Listings **265**

References **269**

Index **283**