A Comparative Study of Model Transformation Approaches through a Systematic Procedural Framework and Goal Question Metrics Paradigm

by

Shekoufeh Kolahdouz Rahimi

A Thesis Submitted for the Degree of Doctor of Philosophy

Department of Informatics
Kings College London

March, 2013
Acknowledgements

I would like to express my deepest appreciation to my supervisor, Dr Kevin Lano. Without his guidance and constant help this dissertation would not have been possible. I greatly appreciate both the time to invested and the ideas that contributed to my PhD, made the experience a truly productive one. I also express my warmest gratitude to my second supervisor, Iman Poernomo, for his endless source of ideas and motivating discussions.

It has been a pleasure to work with Dr Pieter Van Gorp. His technical excellence and guidance had a great impact on me. In addition, I would like to thank Dr Javier Troya and Suresh Pillay for their contribution in Chapter 6.

I also owe a great debt of gratitude to many people in the Department of Informatics at Kings who supported me during times of highs and lows in this period. Dr Laura Bennett, Dr Sanjay Modgil, Dr Elizabeth Black, Mathew Shaw and Young Anthony, not only for their generous support and advice during the writing up, but also for reading my chapters. Moreover, I would like to thank Dr Assel Akzhalova and Dr Gbolahan Williams, for their encouragement from early on. Above all, I appreciate Golnaz and Vida who made me feel the nature of true friendship.

My big thanks goes to my aunt Fereshteh and her family, Mojtaba, Golnoush, Farnoush and Mohammad Ali. I was embraced with a warm atmosphere, with infinite support in all aspects of my life, helping me miss home just a little bit less.

Thanks to my family for all their support, and constant encouragement over the years. To my parents for all their patience throughout this period. But above all, I appreciate my father for helping me realize my true potential, and my mother for her moral support and undying love. For my brothers, Hossein, and Javad I can not thank them enough for the continuous motivation they gave me. And most of all for my niece, Sheida and my nephews, Younes, Danial and Ali for making this experience full of memories of joy and happiness. I would also like to mention Marjan, Shabnam and Atefeh as they were there for their sisterly support.

This thesis is written in memory of my grandmother. I have no doubt that she would be very proud of me. For sure she raised me with a passion for science and a drive to succeed; for that I owe all of this to her.
Abstract

Model Driven Engineering has become a key Software Engineering approach, which aims at improving the cost-effectiveness and reusability of software by capturing the essential semantics of systems in models. By means of model transformations, these models can be analysed, improved, and mapped to executable implementations in a variety of languages and platforms. A large number of different transformation languages and tools, ranging from graph theoretic to relational, hybrid and imperative exist across the research community. A key problem in the current state of Model Driven Engineering is the lack of guidelines and techniques for measuring or improving transformation quality.

This thesis addresses this problem by defining a transformation quality framework based on the ISO/IEC 9126 international software quality standard. The framework is validated on different transformation languages using diverse case studies.

The case studies highlight the problems with the specification and design of particular categories of model transformation, and provide challenging examples by which model transformation languages and approaches can be compared. The evaluation procedure provides clear guidelines for suitability of selected transformation approaches on specific transformation problem by identifying the advantage and disadvantage of each approach.

keywords: Model Driven Engineering, Model Transformation, Quality Model, Metrics
Publications

Throughout this PhD, over twenty publications have been produced of which the following are related to this thesis:


# Contents

Acknowledgements i  
Abstract ii  
Publications iii  

## 1 Introduction

1.1 Overview 2  
1.2 Motivation 4  
1.3 Research methodology 4  
1.4 Overall aims of the thesis 5  
1.5 Contributions 6  
1.6 Overall thesis structure 7  
1.7 Detailed synopsis 7  

## 2 Background and Related Work

2.1 Model Driven Engineering 10  
2.1.1 The Model Driven Universe 10  
2.1.2 Models in MDE 11  
2.1.3 Metamodel in MDE 13  
2.1.4 Model Transformation in MDE 15  
2.1.5 Model Driven Architecture in MDE 16  
2.1.6 Eclipse Modelling Framework in MDE 18  
2.1.7 Discussion 18  
2.2 Model Transformation 19  
2.2.1 Specification of Model Transformation 19  
2.2.2 Directionality of Model Transformation 20  
2.2.3 Different purpose for Model Transformation 21  
2.2.4 Categorisation of Model Transformation 22  
2.2.5 Analysis of Model Transformations 23  
2.2.6 Discussion 26  
2.3 Different Approaches for Model Transformations 26
CONTENTS

4.9 Viatra Solution for the First Case Study ........................................ 113
  4.9.1 Evaluation properties for Viatra ............................................ 115
4.10 QVT-R Solution for the Second Case Study ................................. 117
4.11 Kermeta Solution for the Second Case Study ............................... 118
4.12 ATL Solution for the Second Case Study .................................... 119
4.13 UML-RSDS Solution for the Second Case Study ............................ 121
4.14 Viatra Solution for the Second Case Study .................................. 125
4.15 Comparison of approaches ..................................................... 126
  4.15.1 Evaluation summaries ..................................................... 135
4.16 Conclusion ............................................................................. 141

5 Case Study 3: Comparative Evaluation of Transformation Approaches on the Model Migration Case Study ............................................. 143
  5.1 Third Case Study: Migration of the UML Activity Diagrams from
    version 1.4 to UML 2.2 .............................................................. 144
  5.2 Evaluation Criteria .................................................................. 146
  5.3 Test Case ............................................................................. 149
  5.4 UML-RSDS ........................................................................ 149
    5.4.1 Specification of migration task in UML-RSDS .................... 150
    5.4.2 Evaluation properties for UML-RSDS ............................... 154
  5.5 Epsilon Flock ...................................................................... 157
    5.5.1 Specification of migration task in Epsilon Flock ............... 158
    5.5.2 Evaluation properties for Flock ........................................ 160
  5.6 GrGen.NET ........................................................................ 162
    5.6.1 Specification of migration task in GrGen.NET ............... 162
    5.6.2 Evaluation properties for GrGen ...................................... 164
  5.7 Comparison of approaches .................................................... 166
    5.7.1 Evaluation summaries ..................................................... 169
  5.8 Conclusion ........................................................................... 172

6 Case Study 4: Comparative Evaluation of Transformation Approaches on the Quality Improvement case study ...................................... 175
  6.1 Forth Case study: Class Diagram Restructuring Transformation .... 176
CONTENTS

6.2 Evaluation criteria ....................................................... 178
6.3 Test cases ................................................................. 183
6.4 UML-RSDS solution for class diagram restructuring case study ... 186
   6.4.1 Evaluation properties for UML-RSDS ......................... 189
6.5 GrGen.NET solution for class diagram restructuring case study ... 193
   6.5.1 Evaluation properties for GrGen ................................. 195
6.6 Kermeta solution for class diagram restructuring case study ...... 198
   6.6.1 Evaluation properties for Kermeta ............................... 200
6.7 ATL solution for class diagram restructuring case study .......... 203
   6.7.1 Evaluation properties of ATL ................................... 207
6.8 QVT-Relational solution for class diagram restructuring case study 210
   6.8.1 Evaluation properties of QVT-Relational ...................... 214
   6.8.2 Transformation tool properties ................................. 216
6.9 Comparison of approaches .............................................. 218
   6.9.1 Evaluation summaries ........................................... 224
6.10 Conclusions ............................................................ 232

7 Conclusion .................................................................... 234
7.1 Overview of Thesis ....................................................... 235
7.2 Review of Contributions ................................................. 236
   7.2.1 Refinement and Re-expression transformations in Chapter 4 . 236
   7.2.2 Model Migration transformation in Chapter 5 ................... 240
   7.2.3 Quality Improvement transformation in Chapter 6 ............... 243
   7.2.4 Summary of Contributions ...................................... 247
7.3 Future Work ............................................................... 249
   7.3.1 Optimising model transformations using design patterns .... 250
   7.3.2 Applying the evaluation framework to other categories of model transformation ............................................. 251
   7.3.3 Further extension of evaluation framework .................... 252
   7.3.4 Integration into transML .......................................... 252
7.4 Concluding Remarks ..................................................... 253

Appendices .................................................................... 270
CONTENTS

A Transformation from Class diagram to relational database 271
  A.1 Kermeta .................................................. 271
  A.2 ATL ......................................................... 275
  A.3 QVT-R ..................................................... 277
  A.4 Viatra ..................................................... 279
  A.5 UML-RSDS ............................................... 282

B Input model for UML-RSDS in the Migration case study 285

C Implementation of Quality Improvement case study 288
  C.1 UML-RSDS ............................................... 288
  C.2 GrGen .................................................... 289
  C.3 Kermeta ................................................ 292
  C.4 ATL ....................................................... 305
  C.5 QVT-R ................................................... 307
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The relationship between frameworks in the Model Driven Universe</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Different procedures for representation of model</td>
<td>12</td>
</tr>
<tr>
<td>2.3</td>
<td>Meta-Object Facility</td>
<td>15</td>
</tr>
<tr>
<td>2.4</td>
<td>Model Transformation Architecture</td>
<td>16</td>
</tr>
<tr>
<td>2.5</td>
<td>Transformation between three different representations of a model</td>
<td>18</td>
</tr>
<tr>
<td>2.6</td>
<td>Different states of semantics in source and target model</td>
<td>23</td>
</tr>
<tr>
<td>2.7</td>
<td>A large number of Transformation Approaches in MDE</td>
<td>27</td>
</tr>
<tr>
<td>2.8</td>
<td>Class diagram metamodel</td>
<td>28</td>
</tr>
<tr>
<td>2.9</td>
<td>Relational database metamodel</td>
<td>29</td>
</tr>
<tr>
<td>2.10</td>
<td>Relational database metamodel used in ATL specification</td>
<td>36</td>
</tr>
<tr>
<td>2.11</td>
<td>McCall Quality Model in [100]</td>
<td>49</td>
</tr>
<tr>
<td>2.12</td>
<td>Boehm Quality Model in [17]</td>
<td>51</td>
</tr>
<tr>
<td>2.13</td>
<td>The Goal Question Metrics Paradigm</td>
<td>54</td>
</tr>
<tr>
<td>3.1</td>
<td>Framework for evaluation of model transformation. The outer ring represents</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td>the main characteristic, the next ring contains the relevant sub-character</td>
<td></td>
</tr>
<tr>
<td></td>
<td>istics and, the most inner ring shows the associated attributes for each</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sub-characteristic.</td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td>Tree to graph transformation metamodel</td>
<td>84</td>
</tr>
<tr>
<td>4.2</td>
<td>Class diagram metamodel</td>
<td>87</td>
</tr>
<tr>
<td>4.3</td>
<td>Relational database metamodel</td>
<td>88</td>
</tr>
<tr>
<td>4.4</td>
<td>Test Case 1 for evaluation of execution time in the first case study</td>
<td>93</td>
</tr>
<tr>
<td>4.5</td>
<td>Test Case 2 for evaluation of execution time in the first case study</td>
<td>94</td>
</tr>
<tr>
<td>4.6</td>
<td>Test Case 3 for evaluation of execution time in the first case study</td>
<td>95</td>
</tr>
<tr>
<td>4.7</td>
<td>Calling dependencies of QVT-Relational solution for the second case study</td>
<td>118</td>
</tr>
<tr>
<td>4.8</td>
<td>Calling dependencies of Kermeta solution for the second case study</td>
<td>120</td>
</tr>
<tr>
<td>4.9</td>
<td>Relational database metamodel used in ATL specification</td>
<td>121</td>
</tr>
<tr>
<td>4.10</td>
<td>Calling dependencies of ATL solution for the second case study</td>
<td>122</td>
</tr>
<tr>
<td>4.11</td>
<td>Calling dependencies of Viatra solution for the second case study</td>
<td>126</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>5.1</td>
<td>UML 1.4 Activity Graphs based on [112]</td>
<td>144</td>
</tr>
<tr>
<td>5.2</td>
<td>UML 2.2 Activity Diagrams based on [114]</td>
<td>145</td>
</tr>
<tr>
<td>5.3</td>
<td>Activity model to be migrated</td>
<td>150</td>
</tr>
<tr>
<td>5.4</td>
<td>Screen shot of UML-RSDS</td>
<td>155</td>
</tr>
<tr>
<td>6.1</td>
<td>Basic class diagram metamodel</td>
<td>176</td>
</tr>
<tr>
<td>6.2</td>
<td>Rule 1</td>
<td>177</td>
</tr>
<tr>
<td>6.3</td>
<td>Rule 2</td>
<td>178</td>
</tr>
<tr>
<td>6.4</td>
<td>Rule 3</td>
<td>178</td>
</tr>
<tr>
<td>6.5</td>
<td>UML 2 class diagram metamodel</td>
<td>181</td>
</tr>
<tr>
<td>6.6</td>
<td>Remove redundant inheritance</td>
<td>182</td>
</tr>
<tr>
<td>6.7</td>
<td>Remove multiple inheritance</td>
<td>182</td>
</tr>
<tr>
<td>6.8</td>
<td>Replace concrete superclasses by an abstract class and a new concrete subclass of this class</td>
<td>183</td>
</tr>
<tr>
<td>6.9</td>
<td>Test case 1</td>
<td>184</td>
</tr>
<tr>
<td>6.10</td>
<td>Test case 1 result</td>
<td>184</td>
</tr>
<tr>
<td>6.11</td>
<td>Test case 2</td>
<td>185</td>
</tr>
<tr>
<td>6.12</td>
<td>Expected result of test 2</td>
<td>185</td>
</tr>
<tr>
<td>6.13</td>
<td>Call graph of GrGen solution</td>
<td>197</td>
</tr>
<tr>
<td>6.14</td>
<td>Calling dependencies of Kermeta solution</td>
<td>201</td>
</tr>
<tr>
<td>6.15</td>
<td>Call graph of ATL solution (rule 1)</td>
<td>208</td>
</tr>
<tr>
<td>6.16</td>
<td>Call graph of QVT-R solution</td>
<td>216</td>
</tr>
<tr>
<td>7.1</td>
<td>Results of best solution for each case study. “H” denotes high, “M” denotes medium and “L” denotes low. The inner ring reflects the result of the Kermeta solution for the transformation of “tree to graph” in Chapter 4. This is then followed by the result of ATL for the “class to relational database” transformation in Chapter 4. The next ring then presents the Flock evaluation result for the migration of “UML 1.4 to UML 2.2” case study in Chapter 5. The forth ring from the center shows the result of GrGen for the quality improvement case study in Chapter 6.</td>
<td>250</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Transformation category based on alteration of the modelling language and the abstraction level between the source and target models</td>
<td>23</td>
</tr>
<tr>
<td>2.2</td>
<td>Classification category based on conversion of the abstraction level and the semantic relations between the source and target models</td>
<td>24</td>
</tr>
<tr>
<td>2.3</td>
<td>ISO/IEC 9126-1 quality characteristics in [54]</td>
<td>52</td>
</tr>
<tr>
<td>2.4</td>
<td>Summery of Related Work for Evaluation of Model Transformation. The framework column identifies if that particular research generates a framework for evaluation of model transformation. Scope reflects the number of investigated approaches, attribute identifies the kind of attributes, whether internal or external, validation refers to the existence of correlation between metrics value and quality attributes, evaluation type shows the subjective or objective way of evaluation and case study columns presents the number of investigated case studies</td>
<td>59</td>
</tr>
<tr>
<td>3.1</td>
<td>The ISO/IEC 9126-1 quality characteristics in [54]</td>
<td>64</td>
</tr>
<tr>
<td>3.2</td>
<td>Selected quality characteristics, subcharacteristics and attributes from the standard ISO/IEC 9126 quality model for evaluation of model transformation approaches</td>
<td>71</td>
</tr>
<tr>
<td>3.3</td>
<td>Goal Question Metrics paradigm for measurement of quality attributes in model transformation</td>
<td>72</td>
</tr>
<tr>
<td>3.4</td>
<td>Embeddability categories</td>
<td>76</td>
</tr>
<tr>
<td>3.5</td>
<td>Maturity categories</td>
<td>78</td>
</tr>
<tr>
<td>3.6</td>
<td>Fault tolerance categories</td>
<td>78</td>
</tr>
<tr>
<td>3.7</td>
<td>Extensibility categories</td>
<td>81</td>
</tr>
<tr>
<td>4.1</td>
<td>Evaluation criteria for the solutions of first and second case studies</td>
<td>89</td>
</tr>
<tr>
<td>4.2</td>
<td>Size categories for the first case study</td>
<td>90</td>
</tr>
<tr>
<td>4.3</td>
<td>Complexity categories for the first case study</td>
<td>90</td>
</tr>
<tr>
<td>4.4</td>
<td>Value range for Maximum Capability in horizontal level for the first case study</td>
<td>92</td>
</tr>
</tbody>
</table>
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5</td>
<td>Value range for Maximum Capability in vertical level for the first case study</td>
<td>92</td>
</tr>
<tr>
<td>4.6</td>
<td>Number of elements in each level of test cases in group 3 for evaluation of first case study</td>
<td>96</td>
</tr>
<tr>
<td>4.7</td>
<td>Evaluation criteria for QVT-Relational Solution for the First Case Study</td>
<td>97</td>
</tr>
<tr>
<td>4.8</td>
<td>Group 1-Test Cases results for QVT-Relational</td>
<td>99</td>
</tr>
<tr>
<td>4.9</td>
<td>Group 2-Test Cases results for QVT-Relational</td>
<td>99</td>
</tr>
<tr>
<td>4.10</td>
<td>Group 3-Test Cases results for QVT-Relational</td>
<td>100</td>
</tr>
<tr>
<td>4.11</td>
<td>Evaluation criteria for Kermeta Solution for the First Case Study</td>
<td>102</td>
</tr>
<tr>
<td>4.12</td>
<td>Group 1-Test Cases results for Kermeta</td>
<td>104</td>
</tr>
<tr>
<td>4.13</td>
<td>Group 2-Test Cases results for Kermeta</td>
<td>104</td>
</tr>
<tr>
<td>4.14</td>
<td>Group 3-Test Cases results for Kermeta</td>
<td>105</td>
</tr>
<tr>
<td>4.15</td>
<td>Evaluation criteria for ATL Solution for the First Case Study</td>
<td>106</td>
</tr>
<tr>
<td>4.16</td>
<td>Group 1-Test Cases results for ATL</td>
<td>108</td>
</tr>
<tr>
<td>4.17</td>
<td>Group 2-Test cases results for ATL</td>
<td>108</td>
</tr>
<tr>
<td>4.18</td>
<td>Group 3-Test cases results for ATL</td>
<td>109</td>
</tr>
<tr>
<td>4.19</td>
<td>Evaluation criteria for UML-RSDS Solution for the First Case Study</td>
<td>111</td>
</tr>
<tr>
<td>4.20</td>
<td>Group 1-Test Cases results for UML-RSDS</td>
<td>113</td>
</tr>
<tr>
<td>4.21</td>
<td>Group 2-Test cases results for UML-RSDS</td>
<td>113</td>
</tr>
<tr>
<td>4.22</td>
<td>Group 3-Test cases results for UML-RSDS</td>
<td>114</td>
</tr>
<tr>
<td>4.23</td>
<td>Evaluation criteria for Viatra Solution for the First Case Study</td>
<td>115</td>
</tr>
<tr>
<td>4.24</td>
<td>Size metrics of case studies</td>
<td>127</td>
</tr>
<tr>
<td>4.25</td>
<td>Structural Complexity of UML to relational database transformations</td>
<td>128</td>
</tr>
<tr>
<td>4.26</td>
<td>Complexity comparison for the first case study</td>
<td>129</td>
</tr>
<tr>
<td>4.27</td>
<td>Complexity comparison for the second case study</td>
<td>129</td>
</tr>
<tr>
<td>4.28</td>
<td>Execution time comparison for the first case study</td>
<td>130</td>
</tr>
<tr>
<td>4.29</td>
<td>Modularity measures for class to relational database</td>
<td>131</td>
</tr>
<tr>
<td>4.30</td>
<td>Modularisation of Call graphs for UML to Relational database case study</td>
<td>132</td>
</tr>
<tr>
<td>4.31</td>
<td>Development effort comparison for the tree to graph transformation</td>
<td>133</td>
</tr>
<tr>
<td>4.32</td>
<td>Correctness measures</td>
<td>133</td>
</tr>
<tr>
<td>4.33</td>
<td>Summary of usability survey question responses for the first case study</td>
<td>134</td>
</tr>
<tr>
<td>4.34</td>
<td>Usability summaries for the first case study</td>
<td>134</td>
</tr>
</tbody>
</table>
4.35 Quality characteristics for QVT-R. The first sets of ticks are used to show the evaluation of the first case study and the second for the evaluation of the second one. ........................................... 136
4.36 Quality characteristics for Kermeta. The first sets of ticks are used to show the evaluation of the first case study and the second for the evaluation of the second one. ........................................... 137
4.37 Quality characteristics for ATL. The first sets of ticks are used to show the evaluation of the first case study and the second for the evaluation of the second one. ........................................... 138
4.38 Quality characteristics for UML-RSDS. The first sets of ticks are used to show the evaluation of the first case study and the second for the evaluation of the second one. ........................................... 139
4.39 Quality characteristics for Viatra. The first sets of ticks are used to show the evaluation of the first case study and the second for the evaluation of the second one. ........................................... 140
4.40 Overall ranking of approaches for the first case study .................. 140
4.41 Overall ranking of approaches for the second case study ............... 141

5.1 Mapping from UML 1.4 activity diagrams to UML 2.2 .................. 146
5.2 Evaluation criteria for solutions of third case study ..................... 147
5.3 Size categories for migration transformation .................................. 148
5.4 Complexity categories for migration transformation ...................... 148
5.5 Extensions for migration case study ............................................. 149
5.6 Evaluation criteria of UML-RSDS solution for migration case study . 156
5.7 Evaluation criteria of Flock solution for migration case study ......... 160
5.8 Evaluation criteria of GrGen solution for migration case study ....... 164
5.9 Size comparison for migration case study ..................................... 166
5.10 Complexity comparison for migration case study ......................... 167
5.11 Modularity measures for migration case study ............................. 167
5.12 Development Effort ................................................................. 168
5.13 Correctness measures ............................................................... 168
5.14 Quality characteristics of UML-RSDS for the migration case study . 171
5.15 Quality characteristics of Epsilon Flock for the migration case study . 172
LIST OF TABLES

5.16 Quality characteristics of GrGen.NET for the migration case study . 173
5.17 Overall ranking of approaches for the migration case study . . . . . 173

6.1 Evaluation criteria for solutions . . . . . . . . . . . . . . . . . . . . 179
6.2 Size categories . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 180
6.3 Complexity categories . . . . . . . . . . . . . . . . . . . . . . . . . . 180
6.4 Maximum capability categories . . . . . . . . . . . . . . . . . . . . . 182
6.5 Evaluation criteria for UML-RSDS solution . . . . . . . . . . . . . . . 190
6.6 Test case results for UML-RSDS . . . . . . . . . . . . . . . . . . . . . 192
6.7 Maximum capability test results . . . . . . . . . . . . . . . . . . . . . 193
6.8 Evaluation criteria for GrGen solution . . . . . . . . . . . . . . . . . 196
6.9 Test case results for GrGen . . . . . . . . . . . . . . . . . . . . . . . . 197
6.10 Maximum capability test results (GrGen) . . . . . . . . . . . . . . . . 198
6.11 Evaluation criteria for Kermeta solution . . . . . . . . . . . . . . . . 200
6.12 Test case results for Kermeta . . . . . . . . . . . . . . . . . . . . . . . 202
6.13 Maximum capability test results . . . . . . . . . . . . . . . . . . . . . 203
6.14 Evaluation criteria for ATL solution . . . . . . . . . . . . . . . . . . 207
6.15 Test case results for ATL . . . . . . . . . . . . . . . . . . . . . . . . . 209
6.16 Maximum capability test results . . . . . . . . . . . . . . . . . . . . . 209
6.17 Evaluation criteria for QVT-R solution . . . . . . . . . . . . . . . . . 215
6.18 Test case results for QVT-R . . . . . . . . . . . . . . . . . . . . . . . . 216
6.19 Maximum capability test results . . . . . . . . . . . . . . . . . . . . . 217
6.20 Size comparison for quality improvement case study . . . . . . . . . 218
6.21 Complexity comparison for quality improvement case study . . . . . 220
6.22 Modularity measures for quality improvement case study . . . . . . 220
6.23 Development effort for quality improvement case study . . . . . . . 221
6.24 Correctness measures for quality improvement case study . . . . . . 222
6.25 Summary of usability survey question responses for quality improve-

ment case study . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 223
6.26 Usability summaries for quality improvement case study . . . . . . 223
6.27 Correlation results of measures . . . . . . . . . . . . . . . . . . . . . . . 224
6.28 Quality characteristics of UML-RSDS for quality improvement case study 225
6.29 Quality characteristics of GrGen for quality improvement case study . 226
6.30 Quality characteristics of Kermeta for quality improvement case study ........................................... 227
6.31 Quality characteristics of ATL for quality improvement case study ......................................................... 228
6.32 Quality characteristics of QVT-R for quality improvement case study .................................................... 229
6.33 Overall ranking of approaches for the quality improvement case study ...................................................... 230

7.1 Overall ranking of approaches for the first case study in terms of the ISO 9126 characteristics ....................... 239
7.2 Overall ranking of approaches for the second case study in terms of the ISO 9126 characteristics ...................... 240
7.3 Overall ranking of approaches for the migration case study in terms of the ISO 9126 characteristics ................. 242
7.4 Overall ranking of approaches for the quality improvement case study in terms of the ISO 9126 characteristics .... 246
7.5 Summary of the attributes values of the transformation approaches for each case study. In the first ("tree to graph transformation"), second ("class to relational database transformation") and fourth case studies ("quality improvement transformation"), the ticks correspond to the UML-RSDS (U), Kermeta (K), ATL (A), QVT-R (Q), Viatra (V) (first and second case studies)/GrGen (G) (fourth case study) approaches. For the third case ("UML 1.4 to UML 2.2 transformation") the ticks used indicate UML-RSDS (U), Flock (F) and GrGen (G) ........................................... 248
Introduction

This research concerns the topic of Model Driven Engineering (MDE). MDE is a revolutionary paradigm that focuses the attention of technology developers from low level implementations to the more abstract level modelling. Models allow designers to view a system’s structure or behaviour at a higher level of abstraction. They improve the productivity by capturing the relevant information from the current perspective [67][21] [42]. Model transformation is a central element in MDE, being used to map one or more software engineering models into one or more target models [20]. A large number of model transformation notations and tools have been defined. With an increasing demand for the use of model transformation in both the research and industrial communities, it is essential to measure the quality of transformation approaches according to a standard framework. Measuring the different features of transformation approaches will make their benefits more apparent and understandable.

This chapter provides an introduction to the work presented in this thesis. The next section discusses the concept of model transformation. Following this, the motivation and the overall aims of the thesis are presented. The contributions of the thesis are summarized in the next section. Lastly, the structure of the thesis and a detailed synopsis of each chapter are given.
1.1 Overview

Model transformation is an automatic procedure for construction of target model from a source model with respect to a transformation specification [67]. A transformation specification describes acceptable input models and appropriate output models by identifying the metamodels to which a model needs to conform. A Metamodel is an abstraction of a model and identifies its necessary structure and features. In addition, transformation specifications contain a set of rules to describe how a model in the source language can be transformed into a model in a target language [67]. Transformation tools generate output from the input model by applying the transformation specification. Transformation tools differ in their input, output and notations.

Model transformations are employed for a variety of reasons within the MDE community. They are used to improve model quality, to systematically apply design patterns for refactoring, to refine models from platform-independent forms to platform-specific forms, to migrate models in response to metamodel evolution, and generally to translate the semantic content of a model from one language to that of another. The models considered may be graphically constructed using graphical languages such as the Unified Modelling Language (UML) [114], or can be textual notations such as programming languages or formal specification languages (B [30], Z [138]).

Transformations can be implemented using a conventional programming language, such as Java, or by using a special-purpose transformation language, such as Kermeta [35], Atlas Transformation Language (ATL) [7], Query View Transformation (QVT) [117] or Epsilon Transformation Language (ETL) [72]. Such languages provide specialised syntax and language constructs to define transformations as collections of transformation rules. Individual rules usually map a specific kind of source element to corresponding target elements. These special purpose languages are more interesting from the MDE perspective, as they are developed for the purpose of model transformation.

A major distinction between transformation mechanisms is whether they rely on graph transformation (eg., Viatra [154]) or declarative (eg., QVT-R[117]), imperative (eg., Kermeta [35]) or hybrid (ATL [7]) styles. In declarative style languages, transformations are described abstractly as mathematical relations between source
and target models, while in imperative ones, transformations are defined as programs which explicitly define the details of how a source model is transformed into a target model. In principle, the declarative style has the advantage that transformations can be described more clearly and concisely, omitting the details of strategies for selecting and modifying model elements, and avoiding issues concerning the order of applying the rules on specific elements. The imperative style on the other hand makes it easier and more direct to implement the transformation in an executable form.

Numerous approaches for model transformation have been proposed. However, there is a lack of information regarding their practical application. Therefore, it is essential to measure the quality of transformation approaches. The measurement makes the benefit of an approach more understandable and helps to answer a variety of questions associated with any transformation approach. In addition, it allows us to determine the strengths and weaknesses of a particular approach for performing a specific task [38].

“Measurement” in MDE is the process of assigning a quantitative value to the attributes of software products in order to describe and compare them [106]. The International Organisation for Standardization (ISO) has defined a number of standard frameworks for evaluating software quality [54, 56]. Software quality describes the desirable attributes of software products. The ISO/IEC 9126-1 standard, provides comprehensive classification of software qualities with a structured set of characteristics and sub-characteristics [54]. The framework can be applied for measurement of transformation approaches by selecting the features of interest. The metrics are a clear evidence for measurement. There are a variety of mechanisms for defining metrics in the literature. The Goal Question Metrics is a procedural framework for defining metrics based upon a set of measurable goals [11]. It represents the goals of measurement and addresses those with quantifiable questions. Following that, a set of metrics is generated to answer those goals.

Measurement of different features of a transformation approach is more clear if the evaluation is carried out on a specific case study for each category of model transformation use.
1.2 Motivation

As transformations have become more widely applied, they have also become large and complex software systems in their own right, to which MDE can be applied [87, 52]. Characteristics such as, functionality of transformation approaches, the efficiency, maintainability and usability of transformations implemented by means of particular approaches, the reliability of transformation tools, and portability of transformation specifications, have become important factors in selecting an approach for a given transformation problem category.

There have been a number of publications comparing characteristics of model transformation approaches on different case studies [3] [146] [145] [65]. However, there is little work that considers quantitative measurement comparison across different transformation languages and styles, based on a standard framework. Therefore, a suitable broad-based evaluation framework is needed to compare and assess the benefits and disadvantages of particular transformation approaches for specific categories of transformation problems.

1.3 Research methodology

Recent developments in the field of model transformation have led to a number of different approaches and tools. However, there is little agreement on the suitability of different approaches for each individual application. Therefore, our main objective is to investigate the appropriateness of transformation approaches for tackling specific problems in the MDE community and demonstrating the utility of approaches according to a standard evaluation framework. The comparison will help in choosing a suitable tool for a particular transformation task. The research presented in this thesis consists of:

*An investigation into the practical application of transformation approaches, by measuring their relevant quality attributes through a systematic procedural framework.*

where:

**Practical application** will be evaluated through case studies. Each case study highlights the issues and problems with the specification and design of a specific cat-
1.4. OVERALL AIMS OF THE THESIS

egory of model transformation, and provides a challenging example by which model
transformation languages and approaches can be compared. The selected case studies
tackle refinement, re-expression and quality improvement transformation problems.

Transformation approaches will be considered for the purpose of comparison in
this thesis. Approaches will be selected from different language categories. Five
transformation languages on the refinement and re-expression case studies will be
compared. In addition, three approaches on a migration task will be evaluated.
Lastly, five transformation tools will be examined on the quality improvement case
study.

Quality attributes are the factors that affect the behaviour of a transformation
language. They will be selected from the list of specified attributes in the standard
ISO/IEC 9126-1 software quality model.

A systematic procedural framework will provide a step by step formalism to gener-
ate metrics corresponding to each individual quality attribute of interest. A frame-
work will be introduced for the measurement of model transformation approaches in
this thesis.

1.4 Overall aims of the thesis

This thesis focuses on the comparison of several established model transformation
languages from different language categories (GrGen, Viatra: graph transformation
[60] [154], Kermeta: imperative [35], QVT-R: declarative [117], ATL: hybrid [62],
UML-RSDS: general purpose MDE tool [94]) upon transformation problems which
are typical for re-expression, refinement, migration and quality-improvement restruc-
turing transformations.

This thesis has the following complementary aims:

• To define a systematic procedural framework for measurement of different fea-
tures in model transformation approaches.

• To carry out evaluation of transformations using the framework across several
1.5. CONTRIBUTIONS

categories of transformation use.

- To provide guidelines for the selection of model transformation approaches for particular transformation tasks based on the evaluation result.

- To investigate possible enhancements and extensions of model transformation approaches for tackling particular problem domains.

The comparison focuses on the characteristics of Functionality, Reliability, Usability, Efficiency, Maintainability and Portability from the standard ISO/IEC 9126-1 quality model [19].

1.5 Contributions

By systematically comparing and evaluating the selected transformation approaches on the case studies, according to the ISO/IEC 9126-1 standard, clear guidelines for the appropriateness of different types of transformation approaches for re-expression, refinement, migration and restructuring types of transformations will be provided. Indication of the thesis chapter in which contributions are detailed are given in brackets.

The main contributions of this thesis are as follows:

1. I propose a novel methodology for evaluating transformation approaches (Chapter 3).

2. I design/choose case studies for the purpose of comparison in model transformation (Chapters 4, 5 and 6).

3. I apply transformation approaches with different perspectives to the case studies (Chapters 4, 5 and 6).

4. I provide clear guidelines for determining the appropriateness of transformation approaches for solving particular case studies, using the evaluation framework in Chapter 3 (Chapters 4, 5 and 6).

5. I propose possible improvements and modifications of transformation languages and techniques to make them applicable to particular case studies (Chapters 4, 5 and 6).
1.6 Overall thesis structure

In order to satisfy the stated aims, the thesis is organised into seven chapters. In addition to the introduction, Chapter 2 provides the literature review on model transformation, Chapter 3 proposes the methodology for evaluating transformation approaches and Chapters 4, 5 and 6 are each organised into five main parts. The parts are as follows:

1. A Problem statement to specify the case study and its different assumptions, properties and constraints.
2. Evaluation criteria for the assessment of the case study solution.
3. Test cases with different scenarios to check the behaviour of the transformation in extreme cases.
4. Solutions to the case study with the list of evaluation properties.
5. Comparison of the approaches with respect to the evaluation criteria.
6. Conclusion.

Chapter 7 summarises the outcomes of the research, and outlines the areas of possible future work for this thesis.

1.7 Detailed synopsis

Chapter 2 reviews the concept of Model Driven Engineering and particularly model transformation. In addition a clear description of commonly-used transformation languages in the community is provided. The refinement example of transformation from “class diagram to relational database” is used, to have a clear description of notation and specification of these approaches. Moreover, the chapter highlights the advantage of the ISO/IEC 9126-1 quality model as a fundamental framework for evaluation of model transformation. Finally, the thesis is placed in the context of previous surveys and comparisons of transformation approaches in the related work section.
Chapter 3 presents software measurements as an essential part of software engineering. The chapter then proposes a methodology for the evaluation of transformation approaches based on the ISO software quality standard 9126-1. Following that the Goal Question Metrics paradigm uses to generate metrics in a purposeful way.

Chapter 4 illustrates a comparison of five different transformation specification approaches on two case studies. The well-known “tree to graph” case study is an example of a re-expression transformation, while the “class diagram to relational database” transformation is a refinement transformation. The effectiveness of transformation approaches are analysed according to the methodology of Chapter 3.

Chapter 5 describes a UML-RSDS transformation solution to tackle the migration problem which transforms UML activity diagrams from version 1.4 to 2.2. The UML-RSDS tool is then compared with the Flock migration tool and the graph rewriting tool GrGen, based on the methodology of Chapter 3.

Chapter 6 compares five established model transformation languages upon a typical example of a quality-improvement restructuring transformation. Such a transformation iteratively rewrites a model in-place to improve particular quality characteristics or to impose some property upon the model structure. A comprehensive evaluation is then generated according to the methodology of Chapter 3.

Chapter 7 concludes the work in this thesis by giving an overview of each chapter. The contributions of the thesis is presented by using tables with an overall ranking of the approaches for each case study. Finally, it outlines potential areas of future work.
2

Background and Related Work

The previous chapter presented the principle aims of this thesis and highlighted the importance of evaluation in Model Driven Engineering (MDE) and particularly in model transformation. The main aim of this thesis is to develop a systematic framework for evaluating transformation approaches and subsequently determining their suitability for specific problem domains. In order to provide a comprehensive comparative study, a profound understanding of the fundamental concepts and widely-used transformation approaches in the community is required. This chapter provides a detailed overview of the principles of MDE with focus on model transformation.

The chapter is divided into four sections. First, an introduction to the primary elements in MDE is presented. Second, a more detailed account of model transformation is given. Third, the most well-known transformation approaches in the MDE community are described, with the help of the common transformation example of “class diagram to relational database”. Finally, a brief introduction to the suitable quality model for evaluation of model transformation, through the comparative evaluation of existing quality models, is given. Overall, the main contributions of this chapter are to explain different features of model transformation, to introduce commonly-used transformation languages, to identify the lack of evidence for selecting an appropriate approach to tackle particular problem domains and to introduce an appropriate quality model for evaluating model transformation.
2.1 Model Driven Engineering

Demand for operating in diverse environments of distributed and embedded computing, communicating through different interaction methods and adapting to changes in operating environment requires complex software systems. Developing such systems using current code-centric technologies is not a trivial task despite the advancement of programming languages. It is because of the existence of a wide gap between problem and implementation domains. Current research in the area of MDE focuses on reducing the gap by using different modelling technologies to specify problems at a high level of abstraction and using transformation technologies that support systematic transformation of problem-level abstraction to software implementation. Model Driven Engineering (MDE) is a software development methodology established with the aim of raising the level of abstraction in program specification and increasing automation in program development [21] [42]. The higher design level and separation of specification from implementation improves the productivity, portability, interoperability and maintainability of software systems [67]. In order to understand the benefits of MDE in developing complex software systems, a clear view of its nature and underlying components are required. This section presents the fundamental elements of MDE and explains their connection to practical applications. Topics include the model-driven universe, definition of models, metamodels, model transformation, Model Driven Architecture and the Eclipse Modelling Framework.

2.1.1 The Model Driven Universe

The model driven universe contains many different abbreviations which leads to confusion between the various paradigms. This section gives explanations of the main acronyms and their relationship.

The MDE framework reduces the complexity of the implementation level by raising the level of abstraction. MDE focuses on engineering processes for the development of software rather than pure development activities. An instance of MDE is Model Driven Development (MDD), a framework which uses models as the central element in the development process. The aim is to have the implementation automatically generated from the models. Model Driven Architecture (MDA) is a subset of MDD.
2.1. MODEL DRIVEN ENGINEERING

Figure 2.1: The relationship between frameworks in the Model Driven Universe proposed by the Object Management Group (OMG). The MDA paradigm uses OMG standards for the purpose of development [20]. Figure 2.1 shows the relationship between MDE, MDD and MDA.

Models are a primary element in MDE and it is therefore essential to have a clear understanding of this concept. The following sections provide an explanation of a model and subsequently metamodelling technologies in MDE.

2.1.2 Models in MDE

A model is considered to be the central element of MDE. Where model originally comes from the Latin word *modulus*, meaning a small measure [76]. Models have various definitions from different research perspectives. Seidewitz considers a model to be a “set of statements about some system under study” [133], where a statement is an expression about the system that can be considered either true or false.

Starfield states that “a model is a representation of a concept. The representation is purposeful: the model purpose is used to abstract from the reality the irrelevant details” [97]. Kleppe defines a model as “a description of a system written in a well-defined language” [67]. He specifies a well-defined language as one with well-defined form (syntax), and meaning (semantics), which is suitable for automated interpretation by a computer. Bezivin defines a model as “a simplification of a system built with an intended goal in mind. The model should be able to answer questions
Figure 2.2: Different procedures for representation of model

in place of the actual system” [15]. Lano denotes a model to be “a representation of some artifact, which is used to analyse or design properties of that artifact” [83]. He proposes that a model only represents the interesting aspects to the modeller which are expressed in a precise graphical or textual notation. Brown expresses that “models provide abstractions of a physical system that allow engineers to reason about that system by ignoring extraneous details while focusing on the relevant ones” [23].

It is apparent from the definitions above that a model needs to abstract away from the details of a system and requires a language to define the system. In the MDE field, a model is a formal representation of the function, behaviour and structure of a system under study. The formal representation of a model may not appear unless there is a well-defined syntax and semantic language. Models may be graphically constructed using languages such as the Unified Modelling Language (UML) [115], or can be textual notations such as programming languages or formal specification languages (B [1], Z [138], [58], etc). Figure 2.2 shows different procedures for specifying a model in a modelling language.
Two classifications for modelling languages are given. Domain-Specific Languages (DSLs) are designed uniquely for the specific domain or company. However, General-Purpose Modelling Languages (GPMLs) can be applied to any domain. For instance, UML is considered as a suitable language to model a variety of sectors.

The OMG specified the UML in 1997 [111]. Ever since, UML has been the most common modelling language in the software development community. UML captures important characteristics of the system and visualizes them by applying a graphical language. Object Constraint Language (OCL) is part of the UML standard and is employed for specification of constraints in the UML model with a declarative nature [110]. OCL notations specify constraints in different forms; an invariant, a post-condition, a pre-condition or a guard. Invariants are sets of constraints that apply to all the instances of a UML model and guards are the constraints that specify the pre-conditions and must be met for the occurrence of transition. post-conditions are evaluated to determine whether they are true (or not) after the execution of the operation. The operation is failed if the post-condition is not evaluated to true.

In order to provide comprehensive representation of the system it is essential to have an expressive model. The higher level model is required to define the structure, semantics, constraints and processes to form a model. The initialization of the higher model generates the model of interest. A metamodel is an alternative concept in MDE for representation of an abstraction of a model while highlighting its various features.

2.1.3 Metamodel in MDE

A metamodel describes the structure of models and is specified with sets of syntactic and semantic constraints in a modelling language. The definition of the modelling language is provided at a metamodel level [20]. Metamodelling is an important concept for the design of modelling languages. A system can be defined with a model and the model of the system can be described with an additional model. Similarly, the hierarchical process is applicable to a metamodel and a model of the metamodel. A precise and consistent metamodel provides a convenient procedure for representation of a model. In addition, it is essential for the valid model at each level to conform to its metamodel and satisfies its constraints [118].
2.1. MODEL DRIVEN ENGINEERING

The Meta Object Facility (MOF) is a framework for metamodelling, defined by OMG, to facilitate interoperability between MDE tools. The MOF framework was originally generated in UML, for the purpose of generating a metamodel. The framework comprises four levels to represent the abstraction of a model. Figure 2.3 illustrates the four levels of MOF framework. Level M0 is the domain of run-time instances and contains the actual real world objects (in this example, a `t1:Tree`). Level M1 contains the domain of the user models such as the specific class diagrams. Models written in UML are an example of this layer (Tree and its attributes). The M2 level is the level of language definition such as the class diagram language. The most prominent example of this layer is the UML metamodel. It defines the concepts being applied in level M1, such as `Class` and `Property`. Finally, level M3 is the domain of metamodelling notations. This level contains the language used by the MOF to generate metamodels. Similarly, the concepts employed at level M2 (the sole class concept in this example) is specified it this layer. At each level, a model or structure is an instance of a structure at the next higher level. The MOF framework uses XML Metadata Interchange (XMI) to exchange the data between different layers [116]. XMI is an OMG standard for saving UML models in XML format [156]. XML has been developed in W3C [157] as a technology for representation and replacement of the data. Development of metamodelling languages based on the standard MOF metamodeling framework provides consistency between modelling concepts. In addition, it increases the interoperability between other MDE tools.

For different reasons, models are often required to transform from one representation to another one. The transformed models conform to the constraints of the source metamodel or in many cases need to conform to a new metamodel in the target. Model transformation is the process of converting a model to a different representation while preserving the constraints of the corresponding metamodel. The model and metamodel are considered to be the fundamental elements in model transformation. Next an introduction to the process of model transformation in MDE is given.
2.1. MODEL DRIVEN ENGINEERING

![Diagram showing the relationships between M0 (Run-time instances), M1 (User Model), M2 (UML), and M3 (MOF).]

Figure 2.3: Meta-Object Facility

2.1.4 Model Transformation in MDE

Model transformation is a crucial element in MDE. It provides mapping between different models by taking the source models and transforming them to the target models. The basic definition of a transformation is in the metamodel level while the transformation applies to the elements in the model level. The model level elements conform to the corresponding metamodels in the source and target. The transformation specification describes the process in which the source elements are transformed to the target elements, using a set of rules. A large number of transformation tools and approaches have been defined across the MDE community. Transformations differ in their input model, output model, specification notation and style. Some languages follow standard notations such as UML and MOF and some may have their own representation. A transformation tool is applied to generate a target model.
from the source model using the transformation specification. Figure 2.4 illustrates the general architecture of the transformation paradigm. The comparison of model transformation approaches is the main focus of this thesis.

### 2.1.5 Model Driven Architecture in MDE

The Object Management Group (OMG) was originally formed in 1989 to promote standards in data interchange upon different object-oriented languages. The aim was to promote innovation in software applications by reducing their cost and complexity. Model Driven Architecture (MDA) is one of the major frameworks that has been defined by this group as the realization of MDE [104, 67, 66] and has become the common framework of various organisations. The MDA framework provides comprehensive support for the software development process including analysis, design and implementation. Moreover, it satisfies the standards developed by OMG such
2.1. MODEL DRIVEN ENGINEERING

as UML and MOF. In the MDA framework, models are defined at different levels of abstraction, according to their objectives. The three levels of abstraction for models are: Computational Independent Model, Platform Independent Model and Platform Specific Model.

A Computational-Independent Model (CIM) is an informational view point model which describes domains and requirements of the system without including computational implementation. It represents a system from a high perspective and does not show details of the system structure. This view point provides the connection between experts of domain and experts of design. A model in this case represents familiar concepts and does not require a particular modelling language [43].

A Platform-Independent Model (PIM) is a formal specification of the structure and function of a system that abstracts away technical details. The specification and design models of a software system or business system is independent of the construct of the modelling language in terms of domain concept and implementation. This model takes into account the parts of the system that will be computerised, but does not determine the technological platform that will be supported by the implementation.

A Platform-Specific Model (PSM) describes the specification and design model of a system for the specific target platform. The direct implementation is possible from this view point as sufficient information about functionality of the system is available.

Model Transformation is the main artefact in MDA which enables the automated transformation between each representation. A CIM captures the general requirements from the domain and the model transformation is applied to generate the PIM. This level of model contains the complete specification without any information about implementation. Similarly, a model transformation is executed to generate the PSM with sufficient functionality of the system. Finally, a further application of model transformation produces executable code from the PSM (Figure 2.5).
2.1. MODEL DRIVEN ENGINEERING

![Diagram](image)

Figure 2.5: Transformation between three different representations of a model

2.1.6 Eclipse Modelling Framework in MDE

The Eclipse Development Environment is a model based environment in MDE that has been developed by the Eclipse Foundation [40]. The aim of this organisation is to increase interoperability of the tools by providing a unified platform. An open source Eclipse Modelling Framework (EMF) is the core framework developed by this organisation. In this platform, the metamodel is based on the modelling language of Ecore [49]. Ecore is an Eclipse implementation of the Essential Meta-Object Facility (EMOF). It enables the development of metamodels in the Eclipse platform called Ecore models. EMF provides a specific facility to manipulate models pragmatically and presents models in the tree based editor. The input models in this framework are in XMI format [116]. Although EMF has been developed as an independent modelling framework, the last version has supported MOF 2.0 and an OCL validation feature.

A considerable number of transformation tools have been developed under the EMF platform. Most of the transformation approaches that will be investigated in this thesis provide interoperability with EMF.

2.1.7 Discussion

This section introduced the principle concepts of MDE which are used throughout the remainder of this thesis. A model is a primary element in MDE, providing a formal representation of the function, behaviour and structure of a system. The abstraction of a model is provided by a metamodel, where metamodels highlight the features of the initial model. The Unified Modelling Language (UML) is a general purpose language developed by the OMG group, which is used to define models and metamodels. Model transformation is used to transform models to different
2.2 Model Transformation

Model transformation was previously introduced as a key element of MDE and MDA. In this section the concept is discussed in more detail. Model transformation is employed for a variety of reasons within MDE. These include, to improve model quality, to systematically apply design patterns for refactoring, to refine models from platform-independent forms to platform-specific, to migrate models in response to metamodel evolution, and generally to translate the semantic content of a model from one language to that of another [93]. A number of transformation approaches and tools have been developed; however there is no sufficient evidence for selecting appropriate ones for any given problem domain.

2.2.1 Specification of Model Transformation

This section presents a more in-depth insight into the transformation specification. The transformation process is composed of four principle components; transformation tool, transformation specification, input/output models and input/output metamodels. The transformation specification, determines appropriate source and target models by identifying the metamodel to which a model must conform. Additionally, the transformation specification explicates the procedure of the transformation by using sets of transformation rules [67]. The transformation rules are defined in various styles. The style indicates what needs to be transformed and how it should be done. The transformation tool executes the transformation specification and applies the rules inside it and consequently produces a target model which has conformance with the target metamodel.

Semantically, model transformations are relations from one or more entire models representation according to particular needs. It is a central element in the MDA framework. The sequential transformation process reduces the gap between problem and implementation domains. The Eclipse Modelling Framework (EMF) provides a unified platform for defining metamodels. If the transformation language supports EMF models, then interoperability with other transformation languages will increase. Model transformation is discussed further in the next section.
to one or more alternative entire models, but such a global description is impractical for non-trivial languages and transformations. Instead, model transformations are specified in terms of relations between individual elements in the source model(s) and individual elements in the target model(s). The model-to-model relation is then derived from some composition of these individual relations [69].

Transformation approaches have different styles for expressing the specification and rules of the process. Three general styles for presenting transformation specification have been proposed; declarative, imperative and hybrid.

**Declarative** transformations are described abstractly as mathematical relations between source and target models [2]. Two categories of declarative styles are the relational and graph transformation. A relational transformation is based on relations between source and target elements, while a graph transformation is based on representing transformations between two languages as graph patterns and rewrites.

**Imperative** transformations are defined as programs which explicitly define the details of how a source model is transformed into a target model [35].

**Hybrid** style is a combination of declarative and imperative approaches, e.g., a wide-spectrum specification language in which a declarative description can be refined within the same notation into a program-like description [72].

The declarative style has the advantage that (in principle), transformations can be described more clearly and concisely, omitting the details of strategies for selecting and modifying model elements, and avoiding issues of ordering of application of rules on specific elements. The imperative style on the other hand makes it easier and more direct to implement the transformation in an executable form. The hybrid style attempts to combine the other two styles in order to obtain the advantages of both.

### 2.2.2 Directionality of Model Transformation

The definition of transformation specification can be applicable in one direction or two direction. The unidirectional transformation applies in one direction from source model to the target model, while bidirectional transformation may be executed in both directions. The bidirectional transformation is useful when two models need to be synchronised, and the consistency between models need to be preserved. It means any changes from the source or target model needs to be reflected on other side [140].
2.2. MODEL TRANSFORMATION

The transformation mode is whether forward or backward. In forward transformation the direction of transformation is from source models to target models (like refinement transformation), while in backward transformation the direction is from target models to source models (like reverse engineering transformation) [53].

2.2.3 Different purpose for Model Transformation

Transformation process is applied for different reasons within MDE. The brief categorisation of some common transformation problems are presented below. These categories are not exclusive, and concern how the transformation is being used [83].

**Quality Improvement** removes unnecessary elements and refactors the structure of a model while remaining in the same level of abstraction to improve software quality characters such as understandability, modifiability, reusability, modularity and adaptability.

**Enhancement** extends the model elements while preserving the properties of existing elements and remaining in the same level of abstraction.

**Refinement** of a CIM to PIM or PIM to a PSM. A typical example is a code generation where the source code is translated into the bytecode or executable code.

**Specialisation** excludes some specific situations by strengthening the constraints in the model.

**Re-expression/migration** defines a model in the same level of abstraction but in a different language (a different metamodel).

**Abstraction** reuses a model by abstracting PIM from PSM. A typical example is reverse engineering which extracts a higher level of specification from a lower-level one.

**Generalisations** extends the applicability of models for more situations through weakening the constraints of a model.

**Design Pattern** proposes the design pattern to improve part of the model.

Transformations are either Model-to-Model (M2M) or Model-to-Text (M2T). In M2M transformations, input and output elements are models, while in M2T transformations, the output element is text.
2.2.4 Categorisation of Model Transformation

In this section a brief summary of transformation classification is provided. There have been a number of publications defining classification for transformation approaches. Mens and Van Gorp in [102] applied a multi-dimensional taxonomy to categorise tools, techniques or formalisms for model transformations based on their common qualities. A classification of model transformation approaches based on features is given in [34]. This has defined a general terminology for describing model transformation approaches. A broad range of classification factors are covered and two examples of transformations are presented. The relevant features of model transformation are now discussed in this section. Descriptions of common classifications in model transformation enhances the understanding of problem domains in Chapters 4, 5 and 6.

The alteration of abstraction level and the semantics of source and target models produce two classes of model transformations [102]. The primary step in the transformation process is to define the source and target models with the modelling language. Based on the language of the source and target models a distinction can be made between endogenous and exogenous transformations. In an endogenous or rephrasing transformation the identical modelling language is applied for developing the source and target models. However, in exogenous transformation the source and target models are expressed in different languages.

The level of abstraction in the source and the target models may change. In horizontal transformations, the source and target models preserve the abstraction level, while in vertical transformation, the source and target models reside at different abstraction levels. Table 2.1 illustrates some examples of transformation problem types for possible combinations of these allocations. (The explanation of different transformation types are provided in Section 2.2.3). For instance, for quality improvement transformations, the source and target models are generated with an identical modelling language and then remain on the same level of abstraction.

On the other hand, conversion of both the abstraction level and the semantic relations between the source and target models, introduces new classification for model transformation. Figure 2.6 shows the various possibilities of semantic alterations from the source to the target model during a transformation process. Note that
2.2. MODEL TRANSFORMATION

<table>
<thead>
<tr>
<th>Endogenous</th>
<th>Vertical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quality Improvement</td>
<td>Refinement in same language</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exogenous</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Re-expression (Language migration)</td>
<td></td>
<td>Code generation-Reverse Engineering</td>
</tr>
</tbody>
</table>

Table 2.1: Transformation category based on alteration of the modelling language and the abstraction level between the source and target models.

<table>
<thead>
<tr>
<th>M1</th>
<th>M2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Semantics of M1 and M2 are equivalent

<table>
<thead>
<tr>
<th>M1</th>
<th>M2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Semantics of M2 is weaker than M1

<table>
<thead>
<tr>
<th>M1</th>
<th>M2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Semantics of M2 is stronger than M1

<table>
<thead>
<tr>
<th>M1</th>
<th>M2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

M2 is an extension of M1

Figure 2.6: Different states of semantics in source and target model

the semantics of the target model may be stronger or weaker than semantics of the source model. Moreover, the transformation may enhance the semantics in the target model. Table 2.2 shows the possible combination of these feature values. In this table transformation types are encapsulated according to each division. For instance, in an enhancement type of transformation, the source and target are in the same abstraction level, while the target is an extension of the source model.

2.2.5 Analysis of Model Transformations

Ideally, any specification language for model transformations should support validation, modularity, verification, and the implementation of transformations. Analysing transformation specification according to these criteria enhances the evidence and knowledge for selecting the tool for a particular problem.

Validation enables the analysis of the specification to ensure it represents the
2.2. MODEL TRANSFORMATION

<table>
<thead>
<tr>
<th>Same abstraction level</th>
<th>M2 Lower</th>
<th>M2 Higher</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2 semantically Stronger</td>
<td>Specialization</td>
<td>Refinement</td>
</tr>
<tr>
<td>M2, M1 equivalent</td>
<td>Re-expression/ Quality improvement</td>
<td>Refinement/ Re-expression</td>
</tr>
<tr>
<td>M2 semantically weaker</td>
<td>Generalization</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2.2: Classification category based on conversion of the abstraction level and the semantic relations between the source and target models

correct intended transformation [79], [78]. Validating the specification of a model transformation can include the following:

1. Inspection to check that the specification correctly expresses the intended transformation.

2. Animation of the specification to check that it correctly specifies the transformation on the particular test cases.

3. Static analysis of the description to check properties such as syntactic correctness, completeness and consistency.

4. Testing of the specification is a main way to analyse the transformation in the industry. One of the key challenges for testing model transformations is the construction of test cases.

Modularity provides the possibility of combining two or more model transformation specifications to form a new transformation. It is the key property which supports the other three properties. Transformations described as single large monolithic relations can not be easily understood, analysed or implemented. Instead, if transformations can be decomposed into appropriate smaller units, these parts can be (in principle) more easily analysed and implemented, and the analysis and implementation of the complete transformation can be composed from those of its parts [33, 77, 130].
2.2. MODEL TRANSFORMATION

Verification proves that a transformation is semantically correct, i.e. that all constraints of the source model remain true in the target model (possibly under some interpretation) [6, 131, 10].

Implementation automatically generates an efficiently executable implementation of the transformation, which is correct with respect to the specification.

The analysis of specification languages in MDE have been addressed in different studies. Verification of OCL pre/post conditions in the specification can be carried out by translating them to the notation of B and using proof tools in a toolkit for B to verify that the transformation preserves constraints of the starting model in the transformed model [1]. In [28] a method for analysis of declarative transformations is developed. This method is based on the automatic extraction of OCL invariant defined implicitly in the transformation specification. A number of invariant-based verification properties have been defined to increase degrees of confidence about transformation correctness. The method is applied to Triple Graph Grammars [5] and QVT [117] transformation tools as a proof of concept. Kuster in [80] proposed three techniques for constructing test cases that show presence of errors in model transformation. The techniques have been developed while implementing a set of five model transformations for business-driven development [68] which are used in an MDE approach for business process modelling. The specification can be considered as a special way for analysing model transformation. Alloy is a textual, declarative modelling language based on first-order relation logic [4]. UML2Alloy is a tool which enables integration of the graphical modelling language of UML and the formal specification language of Alloy [18]. It provides automated generation of Alloy from UML by transformation of OCL statements. The Alloy analyser can then be used to conduct fully automated analysis of a model transformation specification represented in it [57]. In addition, Simons in [134] explored the validation process of an abstract syntax of modelling notation. The abstract syntax is presented in Alloy through different approaches. Similarly, in this research the verification is checked by the Alloy analyser tool. UML Specification Environment (USE) [122] allows validation of UML and OCL models by constructing snapshots to represent system states at a particular point in time with objects, attribute values, and links. In addition, a Snapshot Sequence Language (ASSL) [45] constructs snapshots of specification in a declarative procedure by identifying the desired properties of a particular snapshot.
Validation, modularity, and verification are the crucial criteria for evaluation of transformation approaches and are investigated in the comparison framework in Chapter 3.

### 2.2.6 Discussion

This section focused on the different features of model transformation. Transformation specification is described in the metamodel level and applies in the model level. The style of the specification varies between languages. The transformation applies within MDE for various reasons. According to the reasons different classification is provided. This will help us to have a clear understanding of the problem domains. The modelling language, level of abstraction and semantic relation between source and target models may be changed during the transformation process. Analysing a transformation specification enables us to evaluate its usability, improves its productivity and ensures its reliability.

### 2.3 Different Approaches for Model Transformations

Transformations can be implemented using conventional programming languages, such as Java, or by using a special-purpose transformation language. A large number of formalisms have been proposed for the definition of model transformations (Figure 2.7): the pure relational approach of [2], graphical description languages such as graph grammars [41] [29] or the visual notation of QVT [117], hybrid approaches such as Epsilon [72], and also implementation-oriented languages such as Kermeta [35].

Graph transformation or graph rewriting has been a major and traditional approach toward model transformation. A large number of tools choose graphs as the underlying mechanism for the transformation engine. Graph transformation technology allows the expression of endogenous and exogenous transformations in addition to specifying horizontal and vertical ones [28]. The main idea of graph transformation is the rule based modification of graphs. Graph rules match elements of the Left hand side of the graph, and replace them with elements from the Right hand side. Each language has its own mechanism for specifying the constraints of the graph. Graph transformation is considered a favourable approach to deal with model
2.3. DIFFERENT APPROACHES FOR MODEL TRANSFORMATIONS

Figure 2.7: A large number of Transformation Approaches in MDE

transformation. This is because it specifies a transformation explicitly. Additionally, grouping, composing, extensibility and modularity has been supported within graph transformation approaches [103].

In this section an introduction to some well-known model transformation approaches is provided. A commonly-used example of transformation from “class diagram to relational database schema” is employed to have a deeper understanding of the notations and the specification styles in each approach.

2.3.1 Transformation from Class Diagram to Relational Database

The transformation from “class diagram to relational database schema” is a well-known example of model-to-model transformation from PIM to PSM. Different solutions to tackle this problem can be found via Kermeta [35], Viatra [154], ATL [7], KMTL [2], and other model transformation languages. The transformation maps a data model expressed in UML class diagram notation to the more restricted data modelling language of relational database schemas. The source of the transformation is a class diagram, conforming to the metamodel of Figure 2.8. Part of the metamodel is presented here and the complete metamodel will investigate in Chapter 4.
This metamodel consists of classes having a name which inherited from abstract class \textit{UMLModelElement}. The class \textit{Class} contains a set of attributes of type \textit{Attribute}. Additionally, it has a \textit{general} reference pointing to the superclasses for modelling inheritance relations. There is a class \textit{PrimitiveDataType}, which models primitive data types. The class \textit{Attribute} has the reference \textit{owner} pointing to the the belonging class. Both \textit{Class} and \textit{PrimitiveDataType} classes inherit from \textit{Classifier} class. The \textit{Classifier} declares the type of attribute.

The target is the relational database, conforming to the metamodel shown in Figure 2.9. Similar to the class diagram metamodel, it is not a complete metamodel. Relational database metamodel consists of the class having a name which inheritance from abstract class \textit{RModelElement}. The class \textit{Table} consists a set of column of type \textit{Column}. The class \textit{Column} has the reference \textit{owner} pointing to the the belonging table. Additionally, there is an alternative class \textit{Key}. The class \textit{Table} consists a set of \textit{keys}. Moreover, the class \textit{Column} and \textit{Table} has reference \textit{key} pointing to the belonging key. Additionally, a \textit{Key} may consist of a set of \textit{columns}.

In order to have a better understanding of the underlying concepts in each ap-
2.3. DIFFERENT APPROACHES FOR MODEL TRANSFORMATIONS

proach, the specification of two levels of this transformation example is included for each solution. In this transformation, attributes in the source model are mapped to columns in the target model. In addition, classes are transformed to tables and primary keys. Depending on the type of attribute in the source model such as PrimitiveDataType or Class, different levels of transformation implementation may be presented.

Query View Transformation Language

The Query View Transformation (QVT) approach was developed as a standard transformation language by the Object Management Group [117]. The language has a hybrid nature. The declarative part is split into user-friendly Relational and Core languages. The Relational language enables complex object pattern matching [136]. The Core language only support pattern matching over a flat set of variables. In addition, QVT has an imperative style specification language called Operational. The Operational mapping can be used to implement the rules that are not applicable in a Relational language [51]. In this thesis, the QVT Relational language is used for the
2.3. DIFFERENT APPROACHES FOR MODEL TRANSFORMATIONS

purpose of comparison in three case studies.

The *Relational* language (QVT-R) specifies a transformation as a set of relations between models. The semantics of relations are defined in a combination of English and first order predicate logic. The transformation is named after the keyword *transformation*. Following that, the source and target models, and metamodels are declared. The relationship between the elements of the candidates model is defined by two or more domains and pairs of *when* and *where* predicates. The object models are marked as *enforce* or *checkonly*. The notation *enforce* means that the rule is enforced by modifying a model (source or target) where necessary. *Checkonly* indicates that the model is checked, but not modified by the rule. The *when* clause specifies the conditions under which the relationship needs to hold, and the *where* clause specifies the condition that must be satisfied by all the model elements participating in the relation. In order to execute any rule, the source model elements need to satisfy sets of constraints. QVT-R considers relations to be in one of two levels: *top-level* and *non-top-level*. The *top-level* relation rules need to be held for execution of transformation, but *non-top-level* relations are executed if they are called in a *where* clause of another relation. *Top-level* relations are distinguished by the keyword *top*. Each relation consists of left and right-hand side object models [98]. The published specification of [117] for transformation of “class diagram to relational database” is used in this section [117].

The *top* relation rule of, *ClassToTable* shows in listing 2.1. This rule filters all the objects of type *Class* in the UML model, and eliminates the ones of its kind that are not set as *Persistent*. It then gets a binding to the value of the *name* property for all the classes that are not filtered out due to a mismatch with other properties. The variable *cn* (line 3) is free and not assigned to any value, this variable then binds to the value of name for the sets of selected classes (line 5). For creating an object in the target model the new pattern must be defined. This pattern specifies the need for creating an object with *name*, *column*, and *primaryKey* properties. The values are set in lines 6, 7 and 8. The *primary key* is used here to define a set of properties of a class that uniquely identifies an object instance of the class in a model. At the time of object creation the *key* is checked, if the matching is found corresponding to the *key* of the selected *class*, a new object is created when a matching object does not existed.
2.3. DIFFERENT APPROACHES FOR MODEL TRANSFORMATIONS

Listing 2.1: Class to Table transformation in QVT-R

The non-top relation of AttributeToColumn is called in the previous rule. This rule generates columns for the attributes through calling three other rules (lines 7, 8, 9 of listing 2.2).

Listing 2.2: Attribute to Column transformation in QVT-R

Listing 2.3 presents the code for mapping PrimitiveAttribute to Column. The PrimitiveAttributeToColumn rule binds the attributes of class with the primitive data type (lines 4-6) and maps them to the table with the same name and sqltype(lines 7,8).

Listing 2.3: PrimitiveAttribute to Column transformation in QVT-R

Apart from Eclipse plug-ins, the QVT Relational is executable in Medini QVT tool. The Medini QVT is a model-to-model transformation tool [101], developed
2.3. DIFFERENT APPROACHES FOR MODEL TRANSFORMATIONS

based on QVT Relational standards. It executes the QVT specification in a powerful engine. This tool integrates with Eclipse, and the metamodels are generated in the Ecore format. Moreover, a graphical debugger and editor are provided for this tool.

Kermeta Transformation Language

Kermeta was designed at INRIA to be the core language of a model-oriented platform [35]. It is a procedural language, built upon the type system of MOF. Transformations are defined as sets of classes and operations in this language. It extends the MOF metamodel by not only declaring the aspects of the model, but also providing strategies for manipulating the models in situations such as static semantics, dynamic semantics and model transformations. For instance, the static composition operator, `require`, enables extending the existing elements of the metamodel and integrating the new elements of metamodel automatically with the original metamodel. The composed model is type-checked to ensure it satisfies the intended needs of the case study. This is an extremely useful feature in the view of metamodel engineering. Kermeta provides a classic way for specification of usual statements such as `blocks`, `loops` and `comments`. The textual syntax uses several keywords like `class`, `attribute` and `reference`. `Classes` are defined in a similar fashion to Java, and placed into brackets. `Operations` are defined within the body of the `class`.

To implement the transformation from “class diagram to relational database” in Kermeta, the class, `Class2RDBMS` is defined in line 1 of the listing 2.4. Moreover, the operation `transform` is specified (line 3) inside this class. First, `Persistent class` is mapped to the `table` in the target model (lines 5, 6, 7) with the same `name` (line 8). The keyword `result` is used to store the value that will be returned by the operation (line 9). In line 9 the new table is added to the list of tables. In languages like ATL and QVT-R elements are stored automatically, while in Kermeta we need to explicitly specify it. Following that, `columns` are generated by calling another operation as `createColumns` (line 14). The `class2table` is a reference of Trace (reference `class2table : Trace<Class,Table>`).

```plaintext
1 class Class2RDBMS
2 {
3   operation transform(inputModel : ClassModel) : RDBMSModel is do
4     // Create tables
5     getAllClasses(inputModel).select{ c | c.isPersistent }.each{ c |
6       stdio.writeln("Create Table " + c.name)
7       var table : Table init Table.new
```
2.3. DIFFERENT APPROACHES FOR MODEL TRANSFORMATIONS

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td><code>table.name := c.name</code></td>
</tr>
<tr>
<td>9</td>
<td><code>result.table.add(table)</code></td>
</tr>
<tr>
<td>10</td>
<td><code>}</code></td>
</tr>
<tr>
<td>11</td>
<td><code>// Create columns</code></td>
</tr>
<tr>
<td>12</td>
<td>`getAllClasses(inputModel).select { c</td>
</tr>
<tr>
<td>13</td>
<td><code>stdio.writeln(&quot;Create Columns for table &quot; + c.name)</code></td>
</tr>
<tr>
<td>14</td>
<td><code>createColumns(class2table.getTargetElem(c), c, &quot;&quot;)</code></td>
</tr>
<tr>
<td>15</td>
<td><code>}</code></td>
</tr>
<tr>
<td>16</td>
<td><code>end</code></td>
</tr>
</tbody>
</table>

Listing 2.4: Create Table for Classes in Kermeta

The `createColumns` operation in listing 2.5, generates columns in the target model for all the attributes by calling alternative operation `createColumnsForAttribute`.

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>operation createColumns(table : Table, cls : Class, prefix : String) is do</code></td>
</tr>
<tr>
<td>2</td>
<td><code>// add all attributes</code></td>
</tr>
<tr>
<td>3</td>
<td>`getAllAttributes(cls).each { att</td>
</tr>
<tr>
<td>4</td>
<td><code>createColumnsForAttribute(table, att, prefix)</code></td>
</tr>
<tr>
<td>5</td>
<td><code>}</code></td>
</tr>
<tr>
<td>6</td>
<td><code>end</code></td>
</tr>
</tbody>
</table>

Listing 2.5: Create Columns for Attributes in Kermeta

To generate columns corresponding to the attributes in the source model, the operation `createColumnsForAttribute` is called. The implementation of this rule presents in listing 2.6. This operation creates columns for attributes which have a `PrimitiveDataType` (line 4). It assigns the name of selected attributes to the name of columns by adding a prefix to it (line 6). This column is then added to the list of columns in the table (line 8). The rest of the code in this section corresponds to generation of foreign key in the table and I have not included it in this section.

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>operation createColumnsForAttribute(table : Table, att : Attribute, prefix : String) is do</code></td>
</tr>
<tr>
<td>2</td>
<td><code>// The type is primitive : create a simple column</code></td>
</tr>
<tr>
<td>3</td>
<td><code>stdio.writeln(&quot;Create column for attribute &quot; + att.name + &quot;, prefix = &quot; + prefix)</code></td>
</tr>
<tr>
<td>4</td>
<td><code>if PrimitiveDataType.isInstance(att.type) then</code></td>
</tr>
<tr>
<td>5</td>
<td><code>var c : Column init Column.new</code></td>
</tr>
<tr>
<td>6</td>
<td><code>c.name := prefix + att.name</code></td>
</tr>
<tr>
<td>7</td>
<td><code>c.type := att.type.name</code></td>
</tr>
<tr>
<td>8</td>
<td><code>table.cols.add(c)</code></td>
</tr>
<tr>
<td>9</td>
<td><code>if att.isPrimary then table.pkey.add(c) end</code></td>
</tr>
<tr>
<td>10</td>
<td><code>end</code></td>
</tr>
</tbody>
</table>

Listing 2.6: Create Columns for Attributes in Kermeta

Kermeta is distributed as an Eclipse plug-in which includes an editor for Kermeta programs with syntax colouring and code completion capabilities. The metamodels are generated as Ecore files [139], and models can then be originated from them. Kermeta is currently a very good compromise between a general-purpose language such as Java, and a specific model transformation language such as QVT. As many
2.3. DIFFERENT APPROACHES FOR MODEL TRANSFORMATIONS

developers are used to object-oriented languages it is easy for them to learn and apply the Kermeta language. The traceability of Kermeta needs to be specified by the developer and it is not generated automatically by the tool like QVT-R.

AtlanMod Transformation Language

AtlanMod Transformation Language (ATL) is a model to model transformation language developed by AtlanMod team (previously called ATLAS Group) [7]. It enables a user to define the transformation specification in a declarative and imperative style. The models in ATL are compatible with the Ecore model and the metamodels conform to the MOF metamodel. A module corresponds to the transformation specification in ATL and generates the target model from the source model. A module composes of a mandatory header section, an import section, a number of helpers and some transformation rules. The header section provides the name of the transformation module by using module keyword. The optional import section declares which ATL libraries have to be imported, the keyword uses is used for its definition. The main components of each module are helpers and rules which provide the functionality of the transformation specification. Helpers make it possible to define factorized ATL code that can be called from different points of an ATL transformation. Two kinds of helpers has been defined in ATL: the operational and attribute helpers [62].

The ATL language generates the transformation from read-only source models to the write-only target models. The rules have declarative and imperative parts. The declarative part of ATL is divided into matched rules and lazy rules. Both parts specify the relation between source patterns and target patterns. Lazy rules have the same structure as matched rules, but they are only applied if they called by other rules. The imperative structure is also embedded in ATL’s structure for specifying complex cases. Two imperative constructors are provided in this tool: called rules and action blocks. The called rules are called by other rules in a procedural style, while action block provides a sequence of imperative statements. The action block part can be used within matched rules or called rules. The keyword rule is used to introduce a matched rule. It consists of two mandatory sections (the source and the target patterns), and two optional sections (the local variables and the imperative). The source metamodel of a matched rule is defined after the keyword from. This means that the target
2.3. DIFFERENT APPROACHES FOR MODEL TRANSFORMATIONS

The target pattern of a matched rule is introduced by the keyword to. When the source pattern of the rule is matched, the target model generates correspondence to the target metamodel. The optional imperative section can be specified by using the keyword do. It will be executed after the initialization of the target elements generated by the rule. ATL provides two execution modes, the default mode and the refining mode. In the default mode it is required to specify all elements of the target model, while in refining mode only the elements that are changed during a transformation are specified and the unchanged elements appear automatically in the target model. The refining mode specifies the transformation by means of matched rules and not lazy rules, or imperative sections. In addition, modification or deletion of a specific element in this mode is not trivial. Although it is applicable to some refinement cases, it still needs further improvement. The comprehensive explanation to the limitation of refining mode by using an example is provided in Chapter 6.

For the transformation of “class diagram to relational database” in ATL language, the module Class2Relational is defined in listing 2.7. For the ATL the relational database metamodel contains a set of Columns and has a reference to its keys. There are two references as owner and keyOf for class Column that pointing to the Table it belongs to and of which it is part of the key (in case it is a key). Moreover, Column has a reference to Type. The relational database metamodel that used in ATL specification is shown in Figure 2.10.

The source and target metamodel are specified as Class and Relational (line 2) in this example. The helper is defined for this transformation and is called objectIdType by the keyword def (line 3). The rule Class2Table (line 6) maps each class element (line 8) to a table element (line 10) with the same name (line 11). The col reference set, is defined to contain all columns that have been created for a single value attribute (line 12). The key reference set has to contain a pointer to the mentioned key (line 14). An attribute instance has to be created as a key (line 16) and its name has to be set to objectId (line 17). In addition, its type reference has to reference a type with the name Integer (line 18). Columns are generated in the target model from attribute elements of class metamodel in the source. This part of the code is not defined explicitly. The code below shows the first part of the transformation in this example.
2.3. DIFFERENT APPROACHES FOR MODEL TRANSFORMATIONS

![Relational database metamodel used in ATL specification](image)

Listing 2.7: Class to Relational Database transformation in ATL

```
1 module Class2Relational;
2 create OUT : Relational from IN : Class;
3 helper def : objectIDType : Relational!Type =
4 Class!DataType.allInstances()->
5 select ( e | e.name = 'Integer')->first();
6 rule Class2Table {
7     from
8     c : Class!Class
9     to
10     out : Relational!Table {
11         name <- c.name,
12         col <- Sequence {key}->
13         union(c.attr->select(e | not e.multiValued)),
14         key <- Set {key}
15     },
16     key : Relational!Column {
17         name <- 'objectID',
18         type <- thisModule.objectIDType
19     }
20 }
```

The rule `DataTypeAttribute2Column` in listing 2.8, takes the `classes` whose attribute is a `single data type` and not `multi valued` (lines 3, 4). It then generates a `column` in the `table` corresponding to the selected `attribute` in the source (line 7) with the same `name` (line 8) and `type` (line 9).
2.3. DIFFERENT APPROACHES FOR MODEL TRANSFORMATIONS

Listing 2.8: DataTypeAttribute to Column transformation in ATL

ATL is supported in OCL, and developed over the Eclipse platform. The models are handed through EMF [139] and MDR (Meta Data repository) [109]. These enable models to support the Ecore models [26] and the MOF 1.4 semantics. Moreover, the ATL debugger and ATL editor are provided for this approach in Eclipse.

Epsilon Transformation Language

The Epsilon Transformation Language (ETL) is a hybrid transformation language based on the Epsilon Transformation Language (Epsilon) platform [70]. The main model management tasks, such as model validation, model comparison, model transformation and model merging, are all supported in Epsilon platform, through different languages. Alternative examples of the languages in this platform are the Epsilon Object Language (EOL) and the Epsilon Validation Language (EVL). The EOL language provides sets of reusable model management tasks and the EVL language provides validation for constraints of models and metamodels. ETL is based on Epsilon Object Language (EOL) [74], and applies the imperative features of EOL for complex transformation. The language has the advantage of model modification, multiple model access and conventional programming constructs (variables, loops, branches, etc). The distinguishing feature of ETL is the support of models with diverse metamodels and technologies by using the Epsilon Model Connectivity (EMC) layer which generates a uniform interface [72].

ETL transformations are organised by sets of modules with a number of rules and operations inside it. Rules are declared with their name, one source, and one or more target elements. The rules can be independent or be an extension of other transformation rules. It is possible to assign applicability of the rules to the particular elements in the source model by defining a guard. The guard can be specified
2.3. DIFFERENT APPROACHES FOR MODEL TRANSFORMATIONS

optionally by using EOL expression or a block of EOL statements. In addition, a
rule has EOL statements for the logical explanation of property values of the target
model. This is called the body of a statement, and contains the main transformation
specification. Furthermore, a number of pre and post blocks of EOL can be included
in ETL modules and transformation rules. These blocks can be executed before and
after transformation rules. The name of each rule is declared after the the keyword
rule and the source and target parameters are identified after the transform and to
keywords, respectively. The extends keyword is used to represent the rule that is
extended. Furthermore, EOL expressions for defining guards are pointed out after
a colon, and a block of statements are encapsulated in curly braces. Pre and Post
blocks are specified with pre and post keywords and an optional name is assigned
for them. In addition, statements for pre and post blocks can be specified in curly
braces. Lastly, the main body of the rule is presented in EOL statements.

The transformation specification from “class diagram to relational database” in
ETL language is shown in listing 2.9. The rule is called Class2Table (line 1) and
following that the source and target metamodels are presented as Class (line 2) and
Column (line 3). The main specification of transformation is defined in EOL state-
ments. Firstly, it maps the name of a class in the source model to the name of a
table in the target model (line 4). It then assigns the primary key to the table ele-
ments, and adds it to the list of columns of the table (lines 5-9). In addition, this
implementation considers if the class is a specialisation of other classes, it generates
a foreign key for it and points it to the primary key of the parent class. This part of
the code is not included in this section as we only focus on the first part of the code.

```
1 rule Class2Table
2   transform c : OOP!Class
3 to t : DB!Table, pk : DB!Column {
4   t.name := c.name;
5   t.database := db;
6   pk.name := t.primaryKeyName();
7   pk.type := 'INT';
8   t.columns.add(pk);
9   t.primaryKeys.add(pk);
10 }
```

Listing 2.9: Class to Table transformation in ETL

The attributes are divided into a single-value and a multi-value attributes. The rule
SingleValuedAttribute2Column in listing 2.10 transforms the single-value attributes
(line 4) into the columns (line 5) of the database table, while preserving its name
(line 7). The owner of the class is transformed to the list of tables (line 8), and
finally the type of the class is transformed to the type of table by calling toDbType() operation (line 9).

```
1  −− Transforms a single-valued attribute
2  −− to a column
3 rule SingleValuedAttribute2Column
4     transform a : OOAtribute
5     to c : DBColumn {
6       guard : not a.isMany
7       c.name := a.name;
8       c.table := a.owner;
9       c.type := a.type.name.toDbType();
10  }
```

Listing 2.10: SingleValuedAttribute to Column transformation in ETL

Epsilon provides Eclipse integration for all the languages in this platform. It enables the specification to be executed and debugged in a user-friendly environment. In addition, an editor, an outline view, and launch configuration interfaces are provided for each language [73].

**Epsilon Flock Transformation Language**

Epsilon Flock is a model transformation language tailored for model migration [128]. In particular, all those model elements that have not yet been affected by metamodel evolution are copied via Flock from an original to a migrated model. The user needs to specify the migration strategy for those model elements that do not conform to the evolved metamodel. Flock is built on top of Epsilon and can be used with a range of modelling technologies. The rule in Flock combines imperative and declarative parts. It performs the transformation by sets of modules which are inherent from EOL modules. Flock modules comprise a number of rules. Each rule has an original metamodel type (originalIType) and can optionally specify a guard. In this language, rules are ordered according to their position in the Flock source file. The rules begin with the keywords migrate or delete. Following that, the original metamodel type is identified. The when keyword is used to distinguish guards in the specification. The main specification of transformation is provided with EOL statements.

The Flock transformations language focuses on migration examples and the implementation of transformation from “class diagram to relational database” is not provided with this approach. However, the structure of specification and execution is similar to the ETL approach. Similar to other Epsilon languages, Flock has an Eclipse plug-in.
2.3. DIFFERENT APPROACHES FOR MODEL TRANSFORMATIONS

Reactive System Development Support Transformation Language

The Reactive System Development Support (UMLRSDS) language is a general model-driven software development approach developed at King’s College London [94]. This language is based on UML and MOF, and provides the necessary specification notations for model transformation definition, at different levels of abstraction. In addition, there exists established formal semantics for parts of the language to support verification and the synthesis of executable code. In this approach individual transformation rules are specified as operations using pre/post pairs in the OCL notation. The precondition of the rule identifies when it is applicable, and to which elements of the source model. The postcondition identifies what changes to elements and connections should be made in the target model. Each transformation is defined as a UML use case (Chapter 16 of [115]), Asm are the preconditions of the use case, Cons are the postconditions. Logically, the transformation is interpreted as achieving the conjunction of the postconditions, under the assumption that the conjunction of the preconditions hold at its initiation. The structure and organisation of the constraints will also be used to guide the practical implementation of the transformation, but this process should usually be accomplished with minimal user involvement.

The individual constraints of a use case, and the entire use case, have both a logical and a procedural interpretation [86] [89]. Constraints are used as the basis of transformation specifications because they have the key advantage that they can be understood without knowledge of the execution semantics of a particular tool. In contrast, even the most declarative style of transformation rule specification, in QVT-Relations or a graph transformation language, requires knowledge of the particular rule scheduling and selection strategies of the language. Visual specifications of rules can be more usable than textual constraints; however there is currently no widely-accepted visual equivalent of OCL. Visual representations could be used to document and explain the formal constraints.

The transformation of “class diagram to relational database” considers the mapping use case in the UML-RSDS tool. The transformation relates attributes in the source model to columns in the target model with the same name, and likewise maps persistent classes to tables with keys. For each persistent class in the source model, there is a unique table representing the class in the target model, with columns for
2.3. DIFFERENT APPROACHES FOR MODEL TRANSFORMATIONS

each owned attribute. In this specification an extra attribute are introduced in Class and Table as umlId and rdbId. These attributes provides unique identifier to look up the target object corresponding to the source object. The formal specification of the transformation as a single global relation between the source and target languages is shown below.

For each persistent attribute in the source model, there is a unique column in the target model, of corresponding type:

\[
\forall a : \text{Attribute} \cdot a.\text{owner.kind} = '\text{Persistent}' \implies \exists_1 \text{cl} : \text{Column} \cdot \text{cl.rdbId} = a.\text{umlId} \land \\
\text{cl.name} = a.\text{name} \land \text{cl.kind} = a.\text{kind} \land \\
(a.\text{type.name} = '\text{INTEGER}' \implies \text{cl.type} = '\text{NUMBER}') \land \\
(a.\text{type.name} = '\text{BOOLEAN}' \implies \text{cl.type} = '\text{BOOLEAN}') \land \\
(a.\text{type.name} \neq '\text{INTEGER}' \land a.\text{type.name} \neq '\text{BOOLEAN}' \implies \text{cl.type} = '\text{VARCHAR}')
\]

The mapping particularly applies to the persistent classes in the source model.

\[
\forall c : \text{Class} \cdot c.\text{kind} = '\text{Persistent}' \implies \exists_1 t : \text{Table} \cdot \text{t.rdbId} = c.\text{umlId} \land t.\text{name} = c.\text{name} \land \\
t.\text{kind} = '\text{Persistent}' \land \\
\text{Column}[c.\text{attribute}.\text{umlId}] \subseteq t.\text{column}
\]

The Column[c.attribute.umlId] is a set of columns corresponds to the attribute of C and those who already created by previous rule.

UML-RSDS transformation approach uses standard UML and OCL so that developers do not need to learn a new notation to specify model transformations and enabling to analyse the specification with UML tools and OCL checkers [121]. In addition, the notation has formal semantics which support verification. UML-RSDS is not restricted to single-target, single-source transformations; mappings may involve any number of metamodels in the source and target. The process of integrating UML-RSDS to the Eclipse platform is under development.
2.3. DIFFERENT APPROACHES FOR MODEL TRANSFORMATIONS

Visual Automated Model Transformations

Visual Automated model TRansformations (VIATRA) is a transformation tool developed at the Budapest University of Technology and Economics [154]. The aim is to provide a language which supports the whole life-cycle of engineering model transformations, including the specification, design, execution, validation and maintenance. It provides general means to specify and implement various transformations between models through the paradigm of graph transformation. The idea is to combine graph transformation and Abstract State Machines (ASM) into a single specific procedure. In addition, Viatra supports models to code transformation and allows designers to customize existing code generators within the same transformation framework.

The pattern is used as a basic concept for defining transformations in Viatra. It has collections of model elements with sets of constraints which are defined by attribute conditions. The transformations, models, and metamodels are stored in the so-called model space. It provides a convenient way for capturing languages and models from different perspectives by using Visual Precise Metamodelling (VPM) approach [153]. The VPM language consists of two basic elements: the entity (a generalization of MOF package) and the relation (a generalization of MOF association end) [9, 14]. The metamodel composes VPM entities, relations and functions in the Viatra approach. It can be created by the visual editor of Viatra as sets of XMI files or generated by hand as a textual VTML file. Moreover, it is also possible to use pre-defined UML models and transform them to VPM metamodels by a simple model transformation program. To build-up models in a VPM model space, an interface is needed to connect the framework components to the actual modelspace.

A combination of several constructs, generates an expressive language to develop both model to model transformations and code generators. Constraints and conditions of the model are defined by graph patterns. In addition, the ASM describes the control structures. The ASM machine contains sets of rules, where ASM rules provide the specification of the transformation. The rules have different types in this specification. The sequential (seq) rule provides an execution of the whole rules inside it in the particular order. The random rule selects, non-deterministically rules, from the list of specified rules. In addition, the log rule notices a warning or an error message into the Eclipse Error Log. The ASM variables are also used inside the machine.
2.3. DIFFERENT APPROACHES FOR MODEL TRANSFORMATIONS

The *let* rule declares a variable and initializes it with a term. This rule is then usable inside the scope of *let*. The *call* parameter applies to call other ASM rules within the same machine. The *forall* syntax is used to select particular variables which satisfy a Boolean condition. If a new element is created during a transformation a keyword *new* is used to represent the container of new element. The complete explanation of the syntax can be found in the Viatra manual [154].

Two levels of transformation from “class diagrams to relational database” is shown below. Firstly, the source and target metamodels and a trace metamodel for recording mappings between the domains are created. Then a set of transformations are defined to realize the mapping from the UML domain to the relational domain. The main rule is defined in the listing 2.11 with the name of input parameters consisting of the class model, the reference model and the database model. The call *initModels* (line3) initialize the models if it satisfies the input metamodels. The rest of graph transformation rules such as *attrOfStringTypeR* are applied in the *forall* constructs. A *skip* represents an “empty” or “do nothing” instruction and it passes the execution to the next rule.

```
1 rule main (in UMLStr ,in RefStr ,in DBStr) = seq
2 { // (1.1)
3  call initModels(UMLStr, RefStr, DBStr); 
4  forall C below models("uml") with apply class2tableR(C) do skip; // (1.2)
5  forall C below models("uml") , A below models("uml")
6  with apply attr2columnR(C, A) do skip; // (1.3)
7  forall A below models("uml") with apply attrOfStringTypeR(A) do skip; // (1.4)
8  forall A below models("uml") with apply attrOfIntTypeR(A) do skip; // (1.5)
9  forall A below models("uml") with apply assoc2tableR(A) do skip; // (1.6)
10 }
```

Listing 2.11: Main rule for transformation of Class diagram to Relational database in Viatra

The *class2tableR* graph transformation rule is called in the main rule. Listing 2.12 presents the implementation of this rule. It creates a new *table* (line 12) and assigns a *column id* to the *table* (line13). The *column id* sets as a *primary key* and creates the mapping reference entity. Finally, a notification with the fully qualified *name* of the *class* and the *table* is printed out (line17).

```
1 g rule class2tableR (in Cls) = { // (2.1)
2  precondition pattern lhs (Cls, ClsNM) = { 
3   Class(Cls) below models("uml")
4   NamedElement_name(N1, Cls, ClsNM)
5   String(ClsNM) below models("uml")
6 } }
7 action {
8   let T = undef in
9  
10 } 
```

43
2.3. DIFFERENT APPROACHES FOR MODEL TRANSFORMATIONS

Listing 2.12: Class to Table transformation in Viatra

The `attr2columnR` graph transformation rule in listing 2.13 filters out all the pairs that are not associated with the class-property pairs in the pre-condition. Similarly, the action part of the rule creates a new `column` in the `table` with the same `name` as `class` element, and also establishes the mapping reference entity.

Listing 2.13: Attribute to Column transformation in Viatra

Viatra is executable on the open source Eclipse tool. The model transformations are specified in a mathematically precise way, and transformations are automated such that the target models can be derived fully and automatically.
Graph Rewrite Generator Transformation Tool

The Graph Rewrite Generator Tool (GrGen.Net) is a general purpose graph rewrite system developed at University of Karlsruhe [59]. The aim was to find patterns in graph based intermediate representations by using compiler constructions. Though the aim of the tool was to facilitate certain compiler construction problems, in the meantime GrGen.NET has grown into a general purpose tool for graph rewriting. The tool is a combination of two groups of components. The first group offers the basic functionality of the system by compiling GrGen declarative graph rewrite rule specifications into highly efficient .NET assemblies. The second offers a rapid prototyping environment, which supports graphical and stepwise debugging of programmed rule applications. GrGen can be used in model transformations, computer linguistics, modern compiler construction, or any other tasks that work with a number of linked objects. Traditionally these tasks are handled by pointer structures which have a low level of abstraction. GrGen generates the code automatically from higher level of nodes which are connected by edges, and rewrite rules of the patterns [61, 44].

Transformation is GrGen performs by sets of rules. The rules provide a specification of patterns in the form of replace or modify blocks. The graph pattern part identifies the matching elements and a rewrite part specifies the changing elements. The patterns are described in terms of node and edge declarations. The node and its type are presented in $n : t$ format. In addition, edge $e$ is declared with its source $x$, target $y$ and type $t$ by $x - e : t - > y$ syntax. Alongside, the rewrite part is presented in modify block. This block identifies the new graph elements or the old ones. The node is rewritten with $y : t < x >$ syntax. In this syntax a node $x$ is rewritten to $y$ with the new type $t$. Similarly, the edge is rewritten with $- y : t < x > - >$ syntax. The implementation of transformation from “class diagram to relational database” is not provided in GrGen. GrGen.NET has been publicly available since 2003. Since then fifty transformation case studies have been implemented using it. In addition, it has been integrated with the Eclipse platform.

2.3.2 Discussion

This section introduced commonly-used transformation languages. The two levels of the transformation example from “class diagram to relational database” are used
2.3. DIFFERENT APPROACHES FOR MODEL TRANSFORMATIONS

to have clear understanding of the approaches. It has been discussed that Kermeta is an imperative procedural language built upon the type system of MOF. Further, ATL as one of the well-known languages, constructing a transformation with a mixture of imperative and declarative style is detailed. Alongside the standard QVT transformation language is presented. It divides into three sub-languages each with specific styles of specification. Following that, ETL as a hybrid model transformation language and the Epsilon Flock migration tool, are also investigated. In addition, UML-RSDS as a general model-driven software development approach is explained. This approach provides the necessary specification notations for model transformation at different levels of abstraction. Furthermore, an introduction to some graph transformation approaches is provided. Viastra is a graph transformation tool developed to support the whole life-cycle of engineering model transformations, including the specification, design, execution, validation and maintenance. GrGen a general purpose graph rewrite system is also presented.

It can be seen that most of the approaches mentioned in this section provide a solution to the transformation of “class diagram to relational database” case study. However, it is not clear which of the approaches is the most suitable for performing this task. The reasoning behind the selection is different according to the criteria of interest for a specific organisation or research group. For instance, the ATL language may be more understandable from one perspective while, an alternative perspective may prefer UML-RSDS. The understandability factor is influenced by knowledge and familiarity of the user with specific notations. Moreover, the attractiveness of an approach varies between individuals and therefore it is not possible to assign an approach a specific measure for understandability criteria. Alternatively, one could claim that UML-RSDS is the best solution for performing this task due to the less number of lines of code and calling relations between the rules in the specification. In addition, it could be argued that the UML-RSDS approach is more modular and can be applied easily for other transformation tasks.

In some cases, the execution time of transformation tool is more important for the user and therefore a tool is selected which is compiled rather than interpreted. For industrial cases a tool with the capability of handling a large input model would be preferred. Another argument may be the development time. It may be important to know how much effort was used to develop the solution for tackling the transforma-
2.4 EVALUATION IN MDE

Model Driven Engineering has become an established technology and therefore it is essential to evaluate its quality in a similar way to the evaluation of object-oriented systems [46]. Research on quality of MDE can promote its usage for complex software systems. MDE is a model-centric technology and its quality attributes need to explain various aspects of modelling. In addition, model transformation applies in MDE to map models to new representations. Consequently, evaluating the quality of Model Driven Engineering consists of evaluating both the quality of models and model transformations. The quality of UML model has been assessed in different studies. For instance, Krogtie in [75] developed a framework for evaluating the quality of model and metamodel. It enabled to identify the weakness of UML model for improvement in future versions. Mohagheghi and Dehlen [106] provided an initial framework for defining and evaluating quality across different categories of MDE. This study also addressed the adaptation of the quality framework to model transformations. Measuring the quality of model transformation makes it concepts more visible and understandable.

Model transformation may need to be evaluated for different reasons such as measuring understandability, modifiability, usability, and interoperability. In order to analyse any software we need to identify the qualities that have influence on its functionality. A quality model is a paradigm for defining quality. It needs to specify in way that express the fundamental specification of qualities in software system. The
metrics can then developed to measure each quality element. Metrics give insights about the product to developers and users. The measurement can be processed subjectively or objectively. The user opinion is considered as a measure of assessment in subjective measurement, while objective measurement requires actual data and appropriate evidence for assessment [64] [38].

2.4.1 Quality Models

A large number of research has been conducted to generate a catalogue of features for software products. In addition, the International Organisation for Standardization (ISO) and the International Electrotechnical Commission (IEC), have defined a number of standards for evaluation of software quality [54]. In the following sections, four quality models are discussed and the suitable one as a foundation for generating a measurement framework of model transformation is identified.

**McCall Quality Model**

One of the first quality model was introduced by McCall in [100]. The framework is divided to measurable (right) and non-mearable (left) qualities. The quality model is presented in Figure 2.11. Metrics are then assigned to the measurable qualities in a subjective process. The McCall quality model is a comprehensive framework; however many of the metrics in this framework can only be measured subjectively. The subjective measurement does not provide sufficient evidence for assessment of model transformation. In order to evaluate model transformation in MDE, it is required to select a quality model with the comprehensive sets of measurable qualities that can be evaluated objectively.

**Boehm Quality Model**

Previously Boehm developed a clear and well-defined framework for analysing the characteristics of software quality [17]. This framework define an initial characteristic of quality as *general utility*. This quality is composed of *as-is utility, maintainability* and *portability*. The generic nature of these qualities forced to decompose them into the measurable characteristics. Metrics are then generated to assess each character. The Boehm’s quality model is shown in Figure 2.12. This framework has a hierarchical
2.4. EVALUATION IN MDE

Figure 2.11: McCall Quality Model in [100]
structure from general level quality to the more specific quality, which is of interest to the users and companies. This is a fundamental framework for quality model; however the top level quality aspects are more generic and inappropriate for measuring the quality of model transformation. There are not comprehensive qualities in the bottom of the hierarchy to cover different features of model transformation. Moreover, it is difficult to provide quantitative measure for some of the characteristics in this framework.

**Dromey Quality Model**

Dromey introduced a novel approach for software quality in [36]. He stated that the quality model varies according to the attribute of particular product. In this study he particularly developed a quality model framework for analysing the quality of software components. The proposed quality attributes have direct relation to the characteristics of software components. Dromey’s work is interesting; however it is not a trivial task to generate a framework individually for each particular elements in MDE. For instance, model transformation approaches have different styles and can be analysed from different perspectives. This results in having different frameworks for evaluation of each language.

**ISO/IEC 9126 Quality Model**

In 1991, the ISO/IEC 9126 quality model is introduced by The International Organisation for Standardization and the International Electrotechnical Commission [19]. This standard defines the comprehensive sets of quality attributes for evaluating software products. In addition, it provides a set of guidelines for the measurement of quality attributes. The ISO/IEC 9126 is applicable for the evaluation of any kind of software product. This framework defines quality models based on general characteristics of software, which are further refined into subcharacteristics. Moreover, the subcharacteristics are then decompose into attributes. The attributes are easier to understand and measure. The ISO/IEC 9126 is one of the most widespread quality standards with a hierarchical relation between characteristic, subcharacteristics and attributes. The metrics are then generated to measure the quality of attributes in the last level of hierarchy. This standard categorise qualities into *internal quality*,
2.4. EVALUATION IN MDE

Figure 2.12: Boehm Quality Model in [17]
2.4. EVALUATION IN MDE

external quality and quality-in-use. Internal attributes can be measured during the
development of software while external attributes are measured within performance
and testing of the software. Quality-in-use refers to the quality of a software product in a specific environment. Internal and external quality models are encapsulated
inside a single framework while an alternative model is introduced for quality-in-use. Table 2.3 enumerates the six internal/external quality characteristics defined in
ISO/IEC 9126 and their decomposition into subcharacteristics.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Subcharacteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Suitability</td>
</tr>
<tr>
<td></td>
<td>Accuracy</td>
</tr>
<tr>
<td></td>
<td>Interoperability</td>
</tr>
<tr>
<td></td>
<td>Security</td>
</tr>
<tr>
<td></td>
<td>Functionality compliance</td>
</tr>
<tr>
<td>Reliability</td>
<td>Maturity</td>
</tr>
<tr>
<td></td>
<td>Fault tolerance</td>
</tr>
<tr>
<td></td>
<td>Recoverability</td>
</tr>
<tr>
<td></td>
<td>Reliability compliance</td>
</tr>
<tr>
<td>Usability</td>
<td>Understandability</td>
</tr>
<tr>
<td></td>
<td>Learnability</td>
</tr>
<tr>
<td></td>
<td>Operability</td>
</tr>
<tr>
<td></td>
<td>Attractiveness</td>
</tr>
<tr>
<td></td>
<td>Usability compliance</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Time behavior</td>
</tr>
<tr>
<td></td>
<td>Resource utilisation</td>
</tr>
<tr>
<td></td>
<td>Efficiency compliance</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Analysability</td>
</tr>
<tr>
<td></td>
<td>Changeability</td>
</tr>
<tr>
<td></td>
<td>Stability</td>
</tr>
<tr>
<td></td>
<td>Testability</td>
</tr>
<tr>
<td></td>
<td>Maintainability compliance</td>
</tr>
<tr>
<td>Portability</td>
<td>Adaptability</td>
</tr>
<tr>
<td></td>
<td>Installability</td>
</tr>
<tr>
<td></td>
<td>Co-existence</td>
</tr>
<tr>
<td></td>
<td>Replaceability</td>
</tr>
<tr>
<td></td>
<td>Portability compliance</td>
</tr>
</tbody>
</table>

Table 2.3: ISO/IEC 9126-1 quality characteristics in [54]

There are many ways for identifying and defining metrics. The Goal Question
2.4. EVALUATION IN MDE

Metrics paradigm (GQM) is used in this evaluation framework to generate metrics corresponding to each attribute [11]. This paradigm is a process to formulate metrics for a particular task in a goal-oriented fashion. It enables the analytical evaluation with sets of pre-defined goals. The application of GQM has the following three hierarchical levels:

- **The conceptual level (Goal)** contains an abstract and non-quantitative goals of products, processes, or resources for specifying the objectives of the measurement.

- **The operational level (Question)** contains a set of concrete and quantifiable questions to characterize how the goal(s) should be attained.

- **The quantitative level (Metric)** contains objective or subjective quantitative metrics that are collected in order to answer the question(s).

In GQM, the goals of the measurement are identified. Several measurement goals may be pursued at the same time. The goals has no real meaning until some quantitative measure is associated with it. In this paradigm the goals are further refined to a set of quantifiable questions. Questions can be reused across several goals. Questions enable us to move from conceptual level to operational level. Finally, a set of metrics is associated with every question in order to answer it in a quantitative way. The same metrics may be associated to more than one question.

The GQM approach for measurement offers several advantages, writing goals allows us to focus on what the important issues are. Defining questions enables us to make the goal more specific and suggests metrics that are relevant to the goals. The resulting GQM lattice allows us to see the full relationship between goals and metrics. It determines what goals and metrics are missing or inconsistent, and provides a context for interpreting the data after it is collected. The three hierarchical levels of GQM is shown in Figure 2.13.

GQM has been introduced by Basili in 1994 and since then it has been used by software engineers in both research and academic. Basili further develop framework for measurement of software products and clarify the relationship between metrics and goals by using GQM paradigm [99]. Latum in [95] summerized the effect of GQM approach on improving measurement programs in the organization. In [150] the GQM
as a most common-used mechanism for goal-oriented software measurement is applied to the industrial case study. The GQM paradigm is integrated with a set of empirical hypotheses in [22] to define measures of product attributes in software engineering. Evaluating MDE features has also been investigated in [46]. Similarly, the Goal Question Metrics (GQM) is used in this work to identify potential metrics [11]. In [107] the application of GQM for various projects and organisations is presented. Further extension of GQM is provided in GQM+ by Basili [12]. The extended strategy provides mechanisms for explicitly linking software measurement goals, to higher-level goals of measurement. Following that the GQM+ is applied for different kinds of measurement including measurement-based IT-business alignment in [13]. In [137] feedback session and cost/benefit analysis are integrated in GQM methodology for industrial software measurement.

The ISO/IEC 9126 is an appropriate quality model for measurement of model transformations in this thesis. Firstly, it covers a wide range of quality attributes from internal to external perspectives. Secondly, the hierarchical structure of framework from quality model attributes provides a reliable process for measurement. Moreover, the provided guidelines for generation of metrics in this framework ease the task of
metrics calculation. Finally, this framework is applicable to any software product independent of its style and structure. The ISO/IEC 9126 quality model and GQM paradigm are used in Chapter 3 to generate a systematic comparison framework for model transformation.

2.4.2 Related Work for Evaluation of Model Transformation

There have been a number of publications comparing model transformation approaches on different case studies. One distinguishing study in this field was conducted at Eindhoven University of Technology. In [147] and [3], the importance of evaluating the quality of model transformation by comparing it to the traditional software in terms of reusability and changeability of requirements is highlighted. Amstel, in [147], argued the suitability of assessment techniques for evaluating the quality of model transformation from different perspectives. This study emphasis the effect of internal qualities on reusability and maintainability of model transformation. Here, direct and indirect assessment for evaluating the internal and external qualities of model transformation are explained. Amstel also discussed the difficulty of indirect measurement on exogenous and endogenous kinds of transformation by using metrics. Validation, as an alternative technique for assessing the properties of model transformation, is additionally mentioned in this work. Model checking is an example for these types of assessments [152]. He stated that validation investigates the preservation of certain behavioural properties in model transformation; however it does not guarantee the correctness of model transformation.

The work in [3] is given as an example of direct measurement of internal quality for model transformation. In this work the key factors that influence the internal quality of model transformations are presented and assessed by a specialized set of metrics on size, functionality, modularity and consistency. Six relevant quality attributes for evaluation of model transformations are proposed in this work. The comparison was based on the metrics that are generated from different case studies in ASF+SDF transformation language [149]. Here, the metrics are tailored automatically and then compared with the manual ones which has been generated by experts. A questionnaire is used to generate the the manual metrics. Consequently, a framework for analysis of model transformation approaches is generated and the relation between quality
models and metrics are justified in detail. For instance, the result shows the influence of size metrics on understandability and modifiability factors. It means the more lines of code in the specification, the more difficult is to understand and modify the transformation.

Alternative discussion for analysing maintainability of model transformation is presented in [146]. Objective measurement techniques such as metrics collection, dependency analysis and metamodel coverage are proposed as analysis techniques in this study. These techniques are focused on increasing the understandability and reusability of model transformation. Metrics increase analysis by giving an insight into a transformation specification. Dependency analysis illustrates the relation between different parts of transformation specification. In addition, model coverage identifies the relation between a model transformation and its metamodel. This particular technique is only applicable for analysis of model transformation. The tools are used to visualize call relations and metamodel relations. Here, the analysis techniques are investigated with different case studies and MDE projects. The main contribution of this work is to present the effect of analysis in improving understandability of model transformation.

Another evaluation study performed by Amstel is shown in [145]. This work presented a systematic comparison of performance features in three transformation language; ATL, QVT-R and QVT-O. Here, performance of model transformation is considered to be influenced by model, metamodel and transformation specification. The transformations with the identical algorithm are specified in this work and input models with increasing size applied to them. The executions are performed on the same environment and run three times (average time was taken). The result of comparison indicated the faster performance of ATL than QVTO and QVTR approaches in different scenarios.

Another studies regarding the evaluation of model transformation has performed at Karlsruhe University. Kapova in [65] identified the lack of quality metrics for evaluation of functional transformation languages. The work focused on analysis the maintainability feature of QVT-R language by using sets of metrics. 24 metrics from four types of size, relational, consistency and inheritance are defined to get insight into the specification of QVT-R language. The metrics are evaluated on three case studies.
2.4. EVALUATION IN MDE

Vignaga in [155] stated that the practical application of MDE is influenced by the quality of model transformations. Here, the sets of metrics for assessing the quality of ATL transformation language is introduced. The particular case study of ATL2Problem [8] is used to verify the non-structural constraints on the ATL definition of concrete transformations. The resulting model represents violated constraints. The quality attributes for assessing ATL in this work are based on the qualities specified in [148]. In addition to those attributes complexity and performance are also added to the list of quality attributes. ATL transformation considers as a module and the transitive closure of all imported libraries. A set of ATL helpers are encapsulated inside a library. Different metrics corresponding to each element of ATL language such as “number of source model” and “number of imported libraries” are defined. The relation of metrics to quality attributes is then presented. A metric may be related to more than one quality attribute. Interestingly, the table is generated to show the percentage of the metrics that affect each quality attribute.

In [123], Rose describes the result of a migration case study at the Transformation Tool Contest 2010 workshop, with nine graph and model transformation tools applied to a model migration problem (the mapping of UML state machines to activity diagrams). Transformation varied on different aspects such as style and domain. The solutions compared according to six criteria including correctness, conciseness, understandability, appropriateness, tool maturity and extensions. This paper considers correctness and tool maturity as the most important evaluation criteria. Scale and weight were assigned to each evaluation criteria. All solution experts perform a peer review of the other solutions and the results are analysed statistically. Afterwards, the statistics are investigated critically by experts. This enables to identify the evaluation criteria and properties of solution. Following that, the pros and cons of transformation tools according to the evaluation result is explained. In addition, it identifies the potential areas of improvement for each tool.

Alternative comparative study of model transformation is presented by Rose in [125]. He performed a comparative research study on four model migration tools (AML, COPE, Ecore2Ecore and Epsilon Flock) using an example of Petri net metamodel evolution and a comparison example involving the Eclipse Graphical Modeling Framework [49]. They conducted a single dimension research by selecting tools specialised for migration. The innovative feature of this research is the advocate of nine
quality attributes. However, these attributes are not based upon a standard framework. In order to perform the evaluation in this study each tool was used by another participants than its developer. The result of evaluations represents the best solution for the specific migration task, while identifying the weakness of the others. In addition, Rose et al. [128] compared Flock to other languages for model migration, including ATL, Ecore2Ecore and COPE using a petri net example. The comparison is focussed upon the capabilities and conciseness of the approaches for specifying migrations.

In [103], quality requirements are formulated for graph transformation tools and these are analysed without focusing on a specific case study. The qualities are not based on the quality model; however the evaluation of approaches are based on a taxonomy of features. Here, the level of automation by the tool, complexity and correctness are considered as important characteristics of model transformation. Four graph transformation languages are evaluated in this study.

In addition, a further paper based on subjective evaluations is [151], which shows the result of an earlier Transformation Tool Contest. 11 participants using graph-based tools attended the contest to perform a transformation from UML activity diagrams to formal CSP processes. Moreover, in the AGTIVE Tool Contest [120], 17 transformation tools applied to different problems in order to find the pros and cons of each approach. In [50] three model transformation approaches are compared from graph based (CGT, AGG) to hybrid (ATL) on a refactoring example: the removal of unstructured cycles from UML activity graphs. Taentzer et al. [143] generated a taxonomy for graph transformation tools by focusing on AGG, AToM, VIATRA2 and VMTS, using the commonly-used example of transforming from “class diagrams to relational databases”, but without considering any quality attributes.

Table 2.4 summarises the representative works for comparison of model transformation, in terms of framework, scope, attribute, validation, evaluation type and case study. The framework column identifies if that particular research generates a framework for evaluation of model transformation. Scope reflects the number of investigated approaches, attribute identifies the kind of attributes, whether internal or external, validation refers to the existence of correlation between metrics value and quality attributes, evaluation type shows the subjective or objective way of evaluation and case study columns presents the number of investigated case studies.
2.5 Conclusion

This chapter investigated the background and related literature that underpin the remainder of this thesis and can be summarised as follows. The main aim of MDE is to
shift the focus of software development activity from coding to modelling. It provides a systematic use of models as primary artefacts. A metamodel defines a language to form a model. Model transformation maps models onto different representations. A model conforms to its metamodel before and after the transformation. There exists different specification languages to define model transformations, each with its own style for specifying the transformation. Transformations are categorised into different types according to the transformation usage. In addition, alteration of semantic level and abstraction level of source and target languages formalise different classifications. Analysing model transformations for example with regard to complexity and validity, makes its concept model visible and understandable. A large number of model transformations and notations exists across the research community. However, there is no guideline on how to select the appropriate one for a particular case study. Evaluation of model transformation simplifies the task of selection by providing quantitative evidence.

Overall, it has been identified in this chapter that evaluating model transformation lacks a comprehensive systematic framework for selecting the appropriate transformation language according to the quantitative measurement of quality attributes. In this thesis a systematic evaluation framework is generated from a standard ISO/IEC 9126 quality model to validate different transformation languages using various case studies. First, the evaluation framework is specified in the next chapter.
3
Proposed Methodology for Evaluating Transformation Approaches

3.1 Introduction

Software measurement is an essential part of software engineering, developed to quantify the attributes of the characteristics of software products [39]. The measurement of software make its concepts more visible and understandable, while highlighting the importance of improvement in software products. Measuring model transformation in Model Driven Engineering (MDE) is a critical subject of study which helps to identify, improve and select appropriate specification tools for any particular transformation task. This chapter presents the systematic framework for evaluation of model transformation. The framework is developed according to the standard ISO/IEC 9126 quality model [19] [54] and Goal Question Metrics paradigm [11].

In this chapter first, the motivation for developing the evaluation framework for measurement of model transformation is explained. Second, the quality attributes of the ISO/IEC 9126 in terms of model transformation is described. Finally, we describe the Goal Question Metrics paradigm and the unified metrics for measurement of model transformation in this thesis.
3.2 Motivation for Developing Evaluation Framework for Model Transformation

Evaluation of model transformation approaches have been presented in different publications. In Chapter 2 the representative works has been described. Typical studies conducted at Eindhoven University are a substantial attempt toward evaluation of model transformation [147] [3]. However, the proposed quality attributes in all these studies were not initialized from the standard software quality model. Developing a comparison paradigm according to a standard quality model, establishes a firm foundation for the comparison process. Amstel in [3] has an arbitrary selection of six characteristics. These characteristics are not selected systematically and are from the different levels of the ISO/IEC 9126 quality model. In addition, the generated metrics in the study change according to the type of transformation and case study 1. It would be desirable to develop unified metrics that are applicable to all the transformation languages rather than using large experimental case studies, and generating metrics that are applicable to a specific language. In addition, from the view point of companies or industries it is impractical and costly to go through all the metrics and find the most suitable ones. Moreover, it is difficult to find many experts in the ASF-SDF transformation language, while the main stream languages like Kermeta and ATL are broadly used across the research community.

The work by Kapova in [65] is an interesting research towards the evaluation of functional transformation languages. However, the evaluation process is performed particularly for the QVT-R transformation language and does not consider other functional languages. Considering the wider range of transformation languages provides us with more insights into other declarative languages and enables the selection of the most appropriate ones for a particular task. This work focuses on the maintainability of model transformation and other important characteristics such as usability and efficiency are not considered. In addition, the process of generating metrics does not follow the systematic procedural framework.

The proposed metrics in [155] are particularly applicable to ATL transformation

\[\text{For instance, the number of matching conditions per equations does not have any meaning for some of the languages like Kermeta}\]
3.3. Evaluation Framework for Model Transformation

and there is not enough evidence for its usefulness for other transformation languages. In addition, assessing ATL on a single case study does not give sufficient evidence for measurement and comparison. This study evaluates both internal and external properties of ATL language, however the quality attributes are not considered according to the standard quality model. The work would be more interesting if the comparison was conducted on different specifications of a single case study. It then would have provided a better insight about the best specification of ATL for a specific task.

Rose in [123] evaluated a wide ranges of transformation languages with different perspectives on a single case study. However, the metrics are generated subjectively in this work. An objective measurement of the quality attributes gives more reliable evidence for comparison than the subjective measurement. The criteria of evaluation in this work is limited and is not based on a standard quality model. Moreover, this work concentrated on the measurement of internal attributes.

In this thesis a novel systematic framework for comparison of transformation languages is introduced. The framework selects relevant characteristics for evaluation of model transformation from the standard ISO/IEC 9126 quality model. The framework in this thesis covers distinctive characteristics of model transformation including Functionality, Reliability, Usability, Efficiency, Maintainability and Portability. Importantly, subcharacteristics such as Adaptability and changeability are considered, which is not evaluated in previous comparison studies. Additionally, the Goal Question Metrics (GQM) paradigm is used to produce metrics following a systematic procedure. In this paradigm the attributes of interest are considered as the goal of measurement, goals are then refined to a set of quantifiable questions and finally metrics are generated to answer individual questions. The metrics have generic nature and are applicable to different transformation languages, independent of their styles. The systematic framework in this work enables us to compare various transformation languages and tools on complex case studies. The comparison focuses on properties of both transformation language and transformation tool.

3.3 Evaluation Framework for Model Transformation

The International Organisation for Standardization (ISO) has defined a set of ISO and ISO/IEC standards related to software quality [54, 56]. In the previous chapter
the ISO/IEC 9126 quality model is described and its suitability for measurement of model transformation is identified. This framework defines quality models based on general characteristics of software, which are further refined into subcharacteristics. Table 3.1 enumerates the six quality characteristics defined in ISO/IEC 9126 and their decomposition into subcharacteristics.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Subcharacteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td><strong>Suitability</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Accuracy</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Interoperability</strong></td>
</tr>
<tr>
<td></td>
<td>Security</td>
</tr>
<tr>
<td></td>
<td><strong>Functionality compliance</strong></td>
</tr>
<tr>
<td>Reliability</td>
<td>Maturity</td>
</tr>
<tr>
<td></td>
<td><strong>Fault tolerance</strong></td>
</tr>
<tr>
<td></td>
<td>Recoverability</td>
</tr>
<tr>
<td></td>
<td>Reliability compliance</td>
</tr>
<tr>
<td>Usability</td>
<td><strong>Understandability</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Learnability</strong></td>
</tr>
<tr>
<td></td>
<td>Operability</td>
</tr>
<tr>
<td></td>
<td><strong>Attractiveness</strong></td>
</tr>
<tr>
<td></td>
<td>Usability compliance</td>
</tr>
<tr>
<td>Efficiency</td>
<td><strong>Time behavior</strong></td>
</tr>
<tr>
<td></td>
<td>Resource utilisation</td>
</tr>
<tr>
<td></td>
<td>Efficiency compliance</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Analysability</td>
</tr>
<tr>
<td></td>
<td><strong>Changeability</strong></td>
</tr>
<tr>
<td></td>
<td>Stability</td>
</tr>
<tr>
<td></td>
<td>Testability</td>
</tr>
<tr>
<td></td>
<td>Maintainability compliance</td>
</tr>
<tr>
<td>Portability</td>
<td><strong>Adaptability</strong></td>
</tr>
<tr>
<td></td>
<td>Installability</td>
</tr>
<tr>
<td></td>
<td>Co-existence</td>
</tr>
<tr>
<td></td>
<td>Replaceability</td>
</tr>
<tr>
<td></td>
<td>Portability compliance</td>
</tr>
</tbody>
</table>

Table 3.1: The ISO/IEC 9126-1 quality characteristics in [54]
3.3. EVALUATION FRAMEWORK FOR MODEL TRANSFORMATION

3.3.1 Model transformation quality factors

Relevant characteristics and subcharacteristics for evaluation of model transformation approaches can be selected from the ISO/IEC 9126-1 quality framework. The ‘software product’ in this case is the transformation language and its associated tools and methods. The quality of the transformation approach not only concerns the direct quality of its tool support, but the potential quality of transformations developed using the approach. We will incorporate both aspects into our evaluation framework. The selected subcharacteristics for evaluation of model transformation are presented in bold in Table 3.1.

Functionality

The ISO/IEC 9126 defines functionality as “the capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.”

For a transformation approach, we interpret functionality as the capability of the transformation approach to define and implement transformations which meet the stated and implied needs of the transformation requirements. The subcharacteristics of this quality are as follows, based on the corresponding definitions of the subcharacteristics in [54]:

- **Suitability**: the capability of a transformation approach to provide an appropriate means to express the functionality of a transformation problem, at an appropriate level of abstraction, and to solve the transformation problem correctly and with acceptable use of resources (developer time, computational resources, etc).

- **Accuracy**: the capability of the transformation approach to provide a correct transformation implementation.

- **Interoperability**: the capability of a transformation approach to support transformation composition and to be used within other transformation and software development environments.
3.3. EVALUATION FRAMEWORK FOR MODEL TRANSFORMATION

- Functionality compliance: the adherence of the transformation language and tool to standards. For example, the selected approaches in this thesis partially or completely use the OCL 2.0 standard notations in their specification languages. Moreover, being close to programming language like Java is another positive factor for functionality compliance.

We evaluate suitability using a set of attributes which have quantifiable measures. The abstraction level of a transformation approach, and the relative size and complexity of specifications defined using the approach are factors which influence the suitability of the approach: the higher the abstraction level, the more direct it is to express transformation requirements in the transformation language, reducing the likelihood of errors being introduced at this stage. Both the size and complexity of a transformation specification also affect suitability: the larger the size and complexity, the more difficult it is to analyse the specification and to verify its correctness with respect to requirements and to verify its internal consistency. The development effort involved in specifying and implementing a transformation in an approach is a further important factor in measuring the suitability. As the amount of human resources which needs to be spent on the development of transformation tasks increases, the less suitable the approach becomes. The relative effectiveness of transformations produced by the approach in solving the transformation problem is also considered significant. An effective transformation is complete and produce optimal result. Moreover, the computational resources the transformations require to execute is an alternative important factor. The higher these requirements are, the less suitable is the approach.

Accuracy refers to the capability of an approach to achieve correctness and completeness of the transformation specification and implementation. Correctness includes syntactic correctness to ensure the construction of target models which satisfy the language constraints of the target language (the conformance of target model to target metamodel), and ensuring the termination and confluence of the transformation implementation. Accuracy also includes completeness. A complete transformation carries out all required functionality of the transformation, and correctly handle and process the input models. The solution can be optimal or non-optimal.

Interoperability means that the transformation approach should be interoperable
3.3. EVALUATION FRAMEWORK FOR MODEL TRANSFORMATION

with other model-driven development environments and tools. This is difficult to evaluate in general, however we consider that two important factors in achieving a good level of interoperability is being able to exchange model data with the widely-used Eclipse environment for MDE, and having a notation close to well known notations such as the OCL (Object Constraint Language). The transformation approach should also support the embeddability of transformations developed by the approach within larger transformation processes, developed with the same or different approaches.

Security is not generally a quality of interest for model transformation approaches.

Reliability

This characteristic is defined as “the capability of the software product to maintain a specified level of performance when used under specified conditions” in [54].

Important factors for reliability of transformations are the maturity of the model transformation approach (in terms of a history of successful use), and the fault tolerance capabilities of transformations produced by the approach.

- Maturity: the history of successful use of transformation tool and the number of case studies implemented within the tool.

- Fault tolerance: covers how well the transformations produced by the approach detect errors in input models, and errors in processing, and the capability to produce useful error messages.

Maturity is evaluated based on the number of years an approach has been publicly available and the number of case studies to which it has been successfully applied.

Fault tolerance is evaluated by identifying if the transformations successfully process models which fail to satisfy one or more assumptions of the transformation.

Recoverability is not the subcharacteristics of interest for model transformation. Moreover, reliability compliance is not considered as there are no reliability standards for model transformation.

Usability

This is defined in [54] as “The capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions”. For
3.3. EVALUATION FRAMEWORK FOR MODEL TRANSFORMATION

transformation approaches this can be interpreted as referring to the clarity of transformation specifications defined using an approach, and the simplicity of use of the transformation tools for an approach.

- Understandability: how easy a transformation specification is to comprehend.
- Learnability: the degree to which the transformation language and tool can be learnt in a reasonable timescale and with reasonable effort.
- Attractiveness: how acceptable is the language and tool for the user.

A survey is one possible way to measure understandability, learnability and attractiveness of transformation approaches.

Due to the time limitation the operability of model transformation is not evaluated in this work, however it is considered as a future extension of framework. In addition, the usability compliance is not evaluated in this framework as there are no standard for it.

Efficiency

This characteristic is defined in [54] as “The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions.” For transformation approaches this is the capability of the transformation approach to specify efficient transformations.

Here we consider specifically:

- Time behavior: the capability of implemented transformations to provide appropriate response and processing times. We measure the execution times for processing input models of different sizes (model loading and output times are not included), when the transformations are executed as the only application consuming significant resources on the study platform.

The efficiency of each transformation implementation is investigated by measuring the execution time on different input model sizes.

Furthermore, a series of models of increasing sizes are input to each implementation to check the maximum size of model which can be processed in a reasonable time.
3.3. EVALUATION FRAMEWORK FOR MODEL TRANSFORMATION

Resource utilisation and efficiency compliance are important factors for measuring the efficiency of model transformation and will be considered as further extension of framework.

Maintainability

In [54] this characteristic is defined as “the capability of the software product to be modified”. We interpret this as the capability of transformation specifications and implementations in an approach to be modified. Modifications may include corrections, improvements or changes in requirements and functional specifications. In ISO 9126 there are five subcharacteristic. However, we consider specifically the Changeability as the main subcharacteristic. Moreover the Testability is also interested for the evaluation of model transformation and will be investigated for the future evaluations.

- Changeability: the capability of the transformation approach to enable a specified modification of a transformation to be effectively implemented.

Changeability is considered to be one of the most important properties of transformation approaches in the research community [48]. We measure changeability indirectly by measuring size, complexity and modularity. It is a long standing opinion that size, complexity and modularity have influence on maintainability [119]. This factors have influence on navigating the part that need to be changed. A larger size of specification generally results in a less flexible transformation, likewise for increased complexity. In addition, the greater the modularity of the specification, the easier it should be to change the transformation.

We particularly focus on changeability as representative of other subcharacteristics, however the other subcharacteristics have influence on maintainability of model transformation and will be considered as future extension of framework. The maintainability compliance is not considered for evaluation of model transformation as there are no standard for it.

Portability

This characteristic is defined as “the capability of the software product to be transferred from one environment to another” in [54]. This concerns how much work
3.3. EVALUATION FRAMEWORK FOR MODEL TRANSFORMATION

is required to port a software product from one environment to another. We have selected Adaptability from the list of subcharacteristics. In principle Adaptability can have quantitative measurement, but in this thesis it is evaluated subjectively and through estimation. Further quantitative investigation of this approach will be considered as future work.

- Adaptability: the capability of the transformation tool to be adapted for different specified environments without applying actions or means other than those provided for this purpose for the tool considered.

We measure the subcharacteristic of adaptability by considering how much effort is required to extend a transformation to a more general case of the transformation task. The evolution of source or target metamodels is a common environmental change which affects transformations.

In this work we selected adaptability as representative of other subcharacteristics. However, the other subcharacteristics will be considered as future extension of framework. The portability compliance is not considered for evaluation of model transformation as there are no standard for it.

Table 3.2 summarizes the chosen characteristics, subcharacteristics and their corresponding measurable attributes. One attribute may be related to more than one quality factor.

3.3.2 The use of Goal Question Metrics paradigm for Evaluation of Model Transformation

There are many ways for identifying and defining metrics. The Goal Question Metrics (GQM) paradigm is a goal-oriented approach which supports the measurement process in the software engineering domain [11].

In GQM, the goals of the measurement are identified. Several measurement goals may be pursued at the same time. Goals are refined to a set of quantifiable questions, questions can be reused across goals. Finally, a set of metrics is associated with every question in order to answer it in a quantitative way. The same metrics may be associated to more than one question.
### 3.3. EVALUATION FRAMEWORK FOR MODEL TRANSFORMATION

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Subcharacteristic</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Suitability</td>
<td>Abstraction level</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Effectiveness</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Development effort</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Execution time</td>
</tr>
<tr>
<td>Accuracy</td>
<td></td>
<td>Correctness</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Completeness</td>
</tr>
<tr>
<td>Embeddability</td>
<td></td>
<td>Embeddable in transformation process</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Close to well-known notation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Eclipse plug-ins</td>
</tr>
<tr>
<td>Functionality</td>
<td>Compliance</td>
<td>Close to well-known notation</td>
</tr>
<tr>
<td>Reliability</td>
<td>Maturity</td>
<td>History of use</td>
</tr>
<tr>
<td></td>
<td>Fault tolerance</td>
<td>Run-time checks</td>
</tr>
<tr>
<td>Usability</td>
<td>Understandability</td>
<td>Survey</td>
</tr>
<tr>
<td></td>
<td>Learnability</td>
<td>results</td>
</tr>
<tr>
<td></td>
<td>Attractiveness</td>
<td></td>
</tr>
<tr>
<td>Efficiency</td>
<td>Time behaviour</td>
<td>Execution time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum capability</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Changeability</td>
<td>Size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modularity</td>
</tr>
<tr>
<td>Portability</td>
<td>Adaptability</td>
<td>Extensibility</td>
</tr>
</tbody>
</table>

Table 3.2: Selected quality characteristics, subcharacteristics and attributes from the standard ISO/IEC 9126 quality model for evaluation of model transformation approaches.

The GQM approach for measurement offers several advantages. For example, writing goals allows us to focus on what the important issues are. Defining questions enables us to make the goal more specific and suggests metrics that are relevant to the goals. The resulting GQM lattice provides the full relationship between goals and metrics. It determines what goals and metrics are missing or inconsistent, and representing a context for interpreting the data after it is collected. Table 3.3 gives the GQM decomposition for the quantitative model transformation evaluation attributes from Table 3.2, considered as GQM goals.
### 3.3. EVALUATION FRAMEWORK FOR MODEL TRANSFORMATION

<table>
<thead>
<tr>
<th>Goal Level</th>
<th>Question</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstraction Level</td>
<td>What is the abstraction level of the transformation?</td>
<td>Declarative, Hybrid, Imperative</td>
</tr>
<tr>
<td>Size</td>
<td>What is the overall size of specification?</td>
<td>Lines of code (LOC), Non-commented lines of code (NCLOC), Size in KIL</td>
</tr>
<tr>
<td>Complexity</td>
<td>How complex is the transformation specification?</td>
<td>Syntactic complexity, Total number of calls, Total number of recursive calls, Maximum depth of recursive loop, Maximum call depth</td>
</tr>
<tr>
<td>Effectiveness</td>
<td>Does the transformation produce optimal result?</td>
<td>Metrics for each test case</td>
</tr>
<tr>
<td>Development Effort</td>
<td>What is the development time of transformation?</td>
<td>Development time for transformation specification/implementation in person-minute, Future measurement</td>
</tr>
<tr>
<td>Execution time</td>
<td>What is the execution time of transformation on each test case?</td>
<td>Metrics for each test case, Future measurement</td>
</tr>
<tr>
<td>Correctness</td>
<td>How correct is the transformation?</td>
<td>Termination of transformation implementation, Confluence of transformation implementation, Correct target models constructed by transformation implementation</td>
</tr>
<tr>
<td>Completeness</td>
<td>How complete is the transformation?</td>
<td>Completeness for each test case</td>
</tr>
<tr>
<td>Interoperability</td>
<td>How effectively can the transformation be used within a larger transformation?</td>
<td>Both internal and external composition support, Only internal composition support, No convenient mechanism for composition</td>
</tr>
<tr>
<td>Closeness</td>
<td>How close is the transformation specification to a well-known notation?</td>
<td>Common syntax and semantics, Variant syntax and semantics, No similarity</td>
</tr>
<tr>
<td>Interoperability with Eclipse</td>
<td>Does the transformation have complete integration to Eclipse?</td>
<td>Complete integration, Integration via exported/imported data file, No integration</td>
</tr>
<tr>
<td>Maturity</td>
<td>What is the history of use of transformation approach?</td>
<td>Less than four years, Between 4 and 8 years, More than 8 years, Future measurement</td>
</tr>
<tr>
<td>Fault tolerance</td>
<td>Can the transformation produce useful error message?</td>
<td>Error message within the specification, Error message not within the specification, Lacks fault detection</td>
</tr>
<tr>
<td>Understandability</td>
<td>How understandable is the transformation specification/implementation?</td>
<td>Survey</td>
</tr>
<tr>
<td>Learnability</td>
<td>How learnable is the transformation specification/implementation?</td>
<td>Survey</td>
</tr>
<tr>
<td>Attractiveness</td>
<td>How attractive is the transformation implementation?</td>
<td>Survey</td>
</tr>
<tr>
<td>Maximum Capability</td>
<td>What is the largest model that can be processed?</td>
<td>Number of elements in model</td>
</tr>
<tr>
<td>Modularity</td>
<td>How well-factored is specification into modules?</td>
<td>Factoring of repeated sub-expressions, Factorisation</td>
</tr>
<tr>
<td>Cohesiveness</td>
<td>How cohesive is the specification?</td>
<td>Sum of interconnectivity between distinct modules (coupling), Sum of interconnectivity inside each individual module (cohesion)</td>
</tr>
<tr>
<td>Extensibility</td>
<td>How much effort is required to adapt the transformation to extended meta-model?</td>
<td>Amount of effort (subjective measurement)</td>
</tr>
</tbody>
</table>

Table 3.3: Goal Question Metrics paradigm for measurement of quality attributes in model transformation
3.3. EVALUATION FRAMEWORK FOR MODEL TRANSFORMATION

The third columns of Table 3.3 shows the metrics for assessing the transformation solutions. In cases where a numeric value is not appropriate for an attribute (such as abstraction level, maturity) a three-point scale low, medium, high is used to summarise the relative values of attributes.

The goals, questions and metrics corresponding to measurement of model transformation are explained below.

Goal 1 : Abstraction Level

Question : What is the abstraction level of the transformation approach?
Metrics : Abstraction level depends on the style of specification. The higher the abstraction level, the more direct it is to express transformation requirements in the transformation language, reducing the likelihood of errors being introduced at this stage. In addition, it is less complicated to analyse the transformation. The abstraction level is high for declarative solutions, medium for declarative-imperative solutions, and low for primarily imperative solutions.

Goal 2 : Size

Question : What is the overall size of the transformation? Metrics : The size of transformation is measured by counting the total lines of code. Line counts have deficiencies as measures, but they provide a quick comparison of the relative size of different transformation specifications. The comparison of lines of code of different transformation languages on the same problem will tell us about verbosity of languages. The developers are trying to have concise language. In this evaluation the lower the lines of code the better is the verbosity and conciseness of the approach. In addition, the total number of non-commented lines and the size of source file (KB) for each transformation language are also considered as important factors of measurement. This factors will be evaluated in future comparison of transformation languages.

The classification of size is calibrated based on distribution of values of solutions for each problem domains. It is different for each case study.
Goal 3 : Complexity

**Question** : How complex is the transformation specification?

**Metrics** : The sum of the scores for syntactic complexity and structural complexity generates the overall complexity of transformation. Syntactic complexity is due to the complexity of the expressions used within the specification. It is measured by summing the number of occurrences of operators, features and entities in the transformation definition. Structural complexity of a transformation is measured by the number of calls, the number of recursive calls and the maximum call depth in the transformation definition. The classification of complexity is different for each case study. In this evaluation the lower the complexity the more functional and maintainable is the approach.

Goal 4 : Effectiveness

**Question** : Does the transformation produce optimal result?

**Metrics** : The effectiveness of a transformation considers if an optimal result is presented from complete solution. If the transformation generates correct result but produce extra unnecessarily elements, it is not optimal and effective. The higher the effectiveness the more is the functionality of transformation approach.

Goal 5 : Development Effort

**Question 1** : What is the development time of transformation?

**Question 2** : What is the level of familiarity of the developers with the syntax of the language?

**Question 3** : How long does it take to learn the syntax of the languages?

**Question 4** : What is the development time of each test case?

**Metrics** : Development effort for the case study in each transformation approach is measured in person-minutes. The developers of transformation specification for each case study in this thesis are at the same level of expertise. For the first case study the developer was beginner. The time for understanding the problem is not included in the measure of development effort. Moreover, all the test cases were generated in UML-RSDS tool in the XMI format. The first question is the focus of measurement in this thesis. The rest of metrics will investigate for future comparison of model
transformation approaches. The classification of development time for particular transformation problem depends on distribution of solution values. The lower the development time the more functional is the transformation approach. For each case study the attributes are fixed, but the categorisation is changed according to the actual results.

Goal 6 : Execution Time

**Question 1** : What is the execution time of transformations on each test case?
**Question 2** : What is the loading and unloading time of model from the transformation tools?

**Metrics** : Execution time of the approaches is considered as one of the measurement factors. Execution time of the transformation implementation in this thesis does not include the loading and unloading of models from the transformation tool. This will be considered as a future evaluation factor. The classification of execution time is different for each particular transformation problem. The lower the execution time of transformation tool the more functional and efficient is the approach.

Goal 7 : Correctness

**Question** : How correct is the transformation?

**Metrics** : Correctness is divided into syntactic correctness, termination and confluence. Syntactic correctness includes the capability to establish the constraints of the target metamodel of a transformation. This means the conformance of target models to the target metamodel. The transformation has confluence if the effect of the transformation is independent of any alternative orders of application of transformation rules which are allowed by the specification. In addition, transformation has termination if it starts from a valid source model and terminate in a valid final model, through a finite number of computational steps. The classification of correctness is given by an average of three separate 5-point measures for syntactic correctness, termination and confluence. The higher the measure of correctness the more functional is the approach.
Goal 8: Completeness

**Question**: How complete is the transformation?

**Metrics**: A complete transformation carries out all required functionality of the transformation, and correctly handle and process the input models. For the case studies except the update-in-place transformation in Chapter 6, there is not a distinction between effectiveness and completeness. The correct transformation in Chapter 6 can be completed or not completed (Completeness). In addition, the correct transformation can be competed and being optimal or non-optimal (Effectiveness).

Goal 9: Embeddability

**Question**: How effectively the transformation can be used within a larger transformation in the same language or across different languages?

**Metrics**: Interoperability consists of Embeddability. It means how effectively the transformation can be reused within a larger transformation process. External composition relates to the case when individual transformations are composed while in internal composition the internal rules of a particular transformation are composed. The support for embeddability is classified according to Table 3.4. It is high if the language supports both internal and external composition, medium if it provides only internal or external support and low for no convenient mechanism for composition. In some languages it is not possible to embed the top rule of transformation as a rule of another transformation and transformation do not naturally compose. The higher the embeddability the more functional is the transformation approach.

<table>
<thead>
<tr>
<th>Property</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Both internal and external composition support</td>
<td>High</td>
</tr>
<tr>
<td>Only internal or external composition support</td>
<td>Medium</td>
</tr>
<tr>
<td>No convenient mechanism for composition</td>
<td>Low</td>
</tr>
</tbody>
</table>

Table 3.4: Embeddability categories
3.3. EVALUATION FRAMEWORK FOR MODEL TRANSFORMATION

Goal 10: Closeness

**Question**: How close is the transformation specification/implementation to a well-known notation?

**Metrics**: Closeness to a well-known notation, is measured by a three-point scale: *high* for a common syntax and semantics to a well-known notation (such as OCL, programming language or any established standard); *medium* for a variant syntax and/or variant semantics; *low* for no similarity. The higher the closeness to a well-known notation the more functional is the approach.

Goal 11: Interoperability with Eclipse

**Question**: Does the transformation have integration to Eclipse?

**Metrics**: Interoperability with Eclipse is *high* for complete integration; *medium* for interoperability via exported/imported data files only; *low* for no interoperation mechanism.

Goal 12: History of use

**Question 1**: What is the history of use of transformation approach?

**Question 2**: What is the number of problems that have been tackled with a specific transformation approach?

**Metrics**: Maturity of transformation approach is measured according to the years it has been publicly available. In addition, the number of problems that have been tackled with a specific transformation approach is considered. Maturity of languages/tools is classified according to Table 3.5. Maturity is *low* for the approach which has been available less than 4 years, *medium* for the ones available between 4 and 8 years and *high* for more than 8 years. These ranges are normalised based on the maturity values of the case study approaches. The higher is the maturity the more reliable is the transformation approach.

Goal 13: Fault Tolerance

**Question 1**: Can the transformation produce useful error message?

**Question 2**: Can the transformation detect invalid input model?
3.3. EVALUATION FRAMEWORK FOR MODEL TRANSFORMATION

<table>
<thead>
<tr>
<th>Public Availability</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less than 4 years</td>
<td>Low</td>
</tr>
<tr>
<td>4-8 years</td>
<td>Medium</td>
</tr>
<tr>
<td>More than 8 years</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 3.5: Maturity categories

<table>
<thead>
<tr>
<th>Property</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error message (in terms of the specification)</td>
<td>High</td>
</tr>
<tr>
<td>Error message (not in terms of the specification)</td>
<td>Medium</td>
</tr>
<tr>
<td>Lacks fault detection</td>
<td>Low</td>
</tr>
</tbody>
</table>

Table 3.6: Fault tolerance categories

**Metrics** : Fault tolerance (robustness) considers if there are facilities to handle the runtime checks. It is *high* if the tool provides run-time checks and shows message that locate specification (rules) where error acquires; It is *medium* where error message is shown, but not within the specification. It is usually happen for transformation tools that are compiled rather than being interpreted. If there is no fault detection strategy the fault tolerance is *low*. In this evaluation we only focus on the first question. The support for fault tolerance is classified according to Table 3.6. The higher is the fault tolerance the more appropriate and reliable is the transformation approach.

**Goal 14, 15, 16 : Understandability, Learnability, Attractiveness**

**Question** : How understandable / learnable / attractive the transformation specification/implementation?

**Metrics** : Understandability is measured through a survey, together with learnability and attractiveness. People with different expertise were asked to fill in an on-line questionnaire. The form contains five questions. The first question identifies the level of knowledge of the participant. The rest of the questions are as follows:

- To what degree do the rules of the transformation satisfy the case study description / How easy is it to relate the informal to the formal specification? (Understandability)
3.3. EVALUATION FRAMEWORK FOR MODEL TRANSFORMATION

- How well structured is the transformation specification? (Attractiveness)
- How attractive is the specification notation to read? (Attractiveness)
- How much effort is needed to understand the transformation? (Learnability)

In addition to these questions we include a small test case to assess the detailed understanding of the participant: the participant needs to explain where in the code a particular aspect of the transformation is dealt with. This is a factor for understandability. The difference between this score for detailed understanding and the level of initial knowledge is also taken as a learnability factor.

Due to the time limitation of participants I could not include the actual comprehension tests and extensibility test to have more accurate measure of usability. This will be considered as future extension of work.

Goal 17 : Maximum capability

**Question**: What is the largest model that can be processed?

**Metrics**: Test cases with increasing size apply to the transformation approaches to measure the maximum capability of the tools. The classification of maximum capability is different for each case study. The higher is the maximum capability the tool the more efficient it is.

Goal 18 : Modularity

**Question 1**: How well-factorized is specification into modules?

**Question 2**: How cohesive is the specification?

**Metrics**: Two individual factors have influence on measuring modularity of transformation. (1) factorisation: the percentage of unique sub-expressions of rules, at or above a certain minimum syntactic complexity. Syntactic complexity is due to the complexity of the expressions used within the specification. It is measured by summing the number of occurrences of operators, features and entities in the transformation definition. It is three for the first case study and seven for the other case studies investigated in this thesis. (2) the proportion of calls which are internal to modules (100% if there are no calls of any kind), versus the proportion of calls between modules (0% if no calls).
3.3. EVALUATION FRAMEWORK FOR MODEL TRANSFORMATION

For example in order to measure the factorisation in the example below, the minimum syntax complexity is set to 3.

\[(a + b) \times (a + b)\]

The expressions with minimum size are \((a + b)\), \((a + b)\) and \((a + b) \times (a + b)\). There are two repetitions of \((a + b)\) expression while the \((a + b) \times (a + b)\) expression is unique. The total factorisation for this expression is one over three.

An alternative example is shown below:

\[(a + b) \times (a - b)\]

In this example the expressions are \((a + b)\), \((a - b)\) and \((a + b) \times (a - b)\). The number of unique expression is 3 and factorization is 100 % (three over three).

In order to measure the coupling and cohesion, transformation specification is encapsulated into different modules. The calls inside each module represents cohesion and the calls between modules reflects coupling. The transformation approach has high modularity if both cohesion and factorisation are high, low if both are low, and medium otherwise.

It is a well-known software engineering fact that size, complexity and modularity effect the changeability and therefore maintainability of model transformation. These factors have influence on navigating the part that needs to be changed or modified. It has to be mention that we consider coupling and cohesion within the module and not within the rule. Modularity within the rule is considered as a future extension for the evaluation task. The greater the modularity of the specification, the easier it should be to change the transformation and therefore the higher is the maintainability.

Goal 19 : Extensibility

Question : How much effort is required to adapt the transformation to an extended metamodel?

Metrics : Extensibility considers how much effort is required to adapt the transformation to an extended metamodel. The extensibility depends on both the language and the case study. If the transformation is modular for a specific case study, it is more convenient to extend it. The support for extensibility is classified according to...
3.4. CONCLUSION

<table>
<thead>
<tr>
<th>Property</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension can be done in modular manner without substantial effort</td>
<td>High</td>
</tr>
<tr>
<td>Possible in principle, requires substantial extension of specification</td>
<td>Medium</td>
</tr>
<tr>
<td>Impractical due to effort required</td>
<td>Low</td>
</tr>
</tbody>
</table>

Table 3.7: Extensibility categories

Table 3.7. The higher is the extensibility the more portable is the transformation approach.

Figure 3.1 indicates the framework for comparison of model transformation approaches in this thesis. The outer ring represents the main characteristic, the next ring contains the relevant sub-characteristics and finally, the most inner ring shows the associated attributes for each sub-characteristic. One attribute may be related to more than one quality factor in this framework. The framework will apply for evaluation of different case studies in this thesis. The list of evaluation criteria for each case study presents in each chapter. The classification of high, medium and low for quantitative measurement of attributes are calibrated based on the distribution of values in each problem domain (or case study).

3.4 Conclusion

This chapter presented the systematic procedural framework for the evaluation of model transformations. The framework is based on the standard ISO/IEC 9126 software quality model. The use of Goal Question Metrics paradigm in this framework enables us to decompose attributes of model transformation and systematically generate appropriate metrics. The metrics have a unified nature and are applicable to transformation languages with various styles and structures. In general, this framework provides a skeleton for evaluation of different transformation approaches on complex case studies, based on the standard quality model.
Figure 3.1: Framework for evaluation of model transformation. The outer ring represents the main characteristic, the next ring contains the relevant sub-characteristics and, the most inner ring shows the associated attributes for each sub-characteristic.
Case Study 1 and 2: Comparative Evaluation of Model Transformation Approaches on the Re-expression and Refinement case studies

This chapter presents a comparison of five different transformation specification approaches including, (1) the imperative language of Kermeta [35], (2) the declarative QVT-Relational language [117], (3) the hybrid ATL language [62], (4) the general purpose UML-RSDS tool [94] and (5) the graph transformation language of Viatra [154] on the well-known “tree to graph” and “class diagram to relational database” transformations. The effectiveness of the approaches are analysed with the framework proposed in Chapter 3. This chapter is based on a paper that has been published in the International Journal of Software and Informatics in 2012 [93].

In Sections 4.1 and 4.2 we describe the case studies in detail. Section 4.3 defines the criteria for evaluation of two case studies. Section 4.4 presents test cases for evaluation of transformation tools on the first case study. The solutions of QVT-Relational, Kermeta, ATL, UML-RSDS and Viatra for “the tree to graph” transformation are then demonstrated. In Section 4.15 we compare the different solutions for both case studies. Finally, Section 4.16 gives conclusions.
4.1. FIRST CASE STUDY: TREE TO GRAPH TRANSFORMATION

This transformation relates tree objects in the source model to node objects in the target model with the same name, and defines that there is an edge object in the target model for each non-trivial relationship from a tree node to its parent. Figure 4.1 shows the source and target metamodels of the transformation from trees to graphs. The tree metamodel has the language constraint that there are no non-trivial cycles in the parent relationship, excluding the root that self-cycled are allowed.

\[ t : \text{Tree and } t \neq t.\text{parent} \implies t \notin t.\text{parent}^+ \]

where \( r^+ \) is the non-reflexive transitive closure of \( r \). Trees may be their own parent if they are the root node of a tree.

The graph metamodel has the constraint that edges must always connect different nodes, excluding the root, where self-cycles are allowed:

\[ e : \text{Edge implies } e.\text{source} \neq e.\text{target} \]
and that edges are uniquely defined by their source and target, together:

\[ e_1 : \text{Edge and } e_2 : \text{Edge and} \]
\[ e_1.\text{source} = e_2.\text{source and } e_1.\text{target} = e_2.\text{target implies } e_1 = e_2 \]

The identity constraint means that tree nodes must have unique names, and likewise for graph nodes. Transformation can formally be specified by two global constraints:

C1 "For each tree node in the source model there is a graph node in the target model with the same name":

\[ t : \text{Tree implies Node} \rightarrow \exists(n | n.\text{name} = t.\text{name}) \]

C2 "For each non-trivial parent relationship in the source model, there is an edge representing the relationship in the target model":

\[ t : \text{Tree and } t.\text{parent} \neq t \text{ and} \]
\[ n : \text{Node and } n.\text{name} = t.\text{name and} \]
\[ n_1 : \text{Node and } n_1.\text{name} = t.\text{parent.name implies} \]
\[ \exists(\text{Edge} | \text{e.source} = n \text{ and e.target} = n_1) \]

The constraints correspond directly to the informal requirements for the transformation. The specification of this transformation is compared in five different formalisms: QVT-Relational [117] (Medini QVT tool is used in this thesis), Kermeta [35], ATL [7], UML-RSDS [69] and Viatra [154].

4.2 Second Case Study : UML Class Diagram to Relational Database Transformation

The second case study concerns the mapping of a data model expressed in UML class diagram notation to the more restricted data modelling language of relational database. The focus of relational database is on data. The class diagram captures static view of the system and models classes and relationship between them in the real world. Despite importance of class diagram in UML, software developers turns to
4.2. SECOND CASE STUDY : UML CLASS DIAGRAM TO RELATIONAL DATABASE TRANSFORMATION

relational database to make their objects persistence. Transformation of class diagram to relational database is not a trivial task, because the notation and concepts of these languages are different. Modelling aspects such as inheritance, association classes, many-many associations and qualified associations need to be removed from the class diagram model and their semantics expressed instead using the language facilities such as *tables*, *primary keys* and *foreign keys* of relational databases.

The source of the transformation is a class diagram, conforming to the metamodel of Figure 4.2 and the target is the relational database, conforming to the metamodel shown in Figure 4.3. The source metamodel consists of *Classes* having a name, which inherited from abstract class *UMLModelElement*. In addition, there is a *Package* class in the metamodel. This class inheritance from class *UMLModelElement* and contains a set of *elements* of type *PackageElement*. The class *PackageElement* has a reference *namespace* pointing to the belonging *Package*. The class *Class* contains a set of attributes of type *Attribute*. Additionally, it has a *general* reference pointing to the superclasses for modelling inheritance relations. There is a class *PrimitiveDataType* which models primitive data types. The class *Attribute* has the reference *owner* pointing to the belonging class. Both *Class* and *PrimitiveDataType* classes inherit from *Classifier* class. The *Classifier* itself inheritance from *PackageElement* and declares the type of attribute.

Relational database metamodel consists of the class having a *name* which inherited from abstract class *RModelElement*. The class *Schema* is also inheritance from *RModelElement*. The *Schema* consists a set of *table* of type *Table*. In addition, the class *Table* has a reference *schema* pointing to the belonging *Schema*. The class *Table* consists a set of *column* of type *Column*. The class *Column* has the reference *owner* pointing to the the belonging *Table*. There is an alternative class *Key*. The class *Table* consists a set of *keys*. Moreover, the class *Column* and *Table* has reference *key* pointing to the belonging key. Additionally, a *Key* may consist of a set of *columns*. The class *Table* and *Column* consists a set of *foreignKey* of type *ForeignKey*. The class *ForeignKey* has the reference *owner* pointing to the belonging *Table*. Additionally, a *ForeignKey* may consist a set of *columns*.

In general *Attributes* are mapped to *Columns*, *Classes* mapped to *Tables* and *Primary keys*, *Associations* are mapped to *Foreign keys*, and lastly *Packages* to *Schemas*. The *Classes* that are marked as persistent in the source model should be transformed
4.2. SECOND CASE STUDY: UML CLASS DIAGRAM TO RELATIONAL DATABASE TRANSFORMATION

Figure 4.2: Class diagram metamodel

into a single Table of the same name in the target model. The Table contains one or more Columns corresponding to the Attributes of Class, and one or more Columns corresponding to Association of Class. Attributes with PrimitiveDataType are transformed to a single Column with the same type as PrimitiveDataType. However, Attributes whose type is a persistent Class need to be transformed into one or more Columns. It is created from the Primary Key of persistent class. In addition, the Foreign Key element is created and referred to the Table from the persistent Class.

The solutions are then used from the published specifications of this problem for QVT-R [117], Kermeta [35], ATL [7], Viatra [154], and our own version defined in UML-RSDS[87].
4.3 Evaluation Criteria

The evaluation criteria for assessment of these case studies is summarised in Table 4.1. These criteria correspond to the list of measured attributes from the third chapter. They are divided into properties of the transformation language including abstraction level, specification size, complexity, development effort, syntactic correctness, extensibility and closeness, and properties of an implementation in a particular tool for that language containing, execution time, maximum capability, fault tolerance and maturity. The aspects should be separated because there may be several different tools for a given language. For instance we use the Medini tool [101] for QVT-Relational, but it is also executable in Eclipse. The classifications of the properties that differ according to their distribution of values for each case study solutions are described below. The rest of the properties are evaluated based on the classifications in Chapter
4.3. EVALUATION CRITERIA

<table>
<thead>
<tr>
<th>Transformation language</th>
<th>Abstraction level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size: LOC</td>
</tr>
<tr>
<td></td>
<td>Complexity: (1) syntactic complexity (number of operators, number of features) (2) structural complexity (i) total number calls (ii) total recursive calls (iii) maximum call depth Modularity: (1) factorisation (2) cohesion versus coupling Extensibility: extension to tree metamodel Interoperability: inclusion in a transformation process Development effort Close to well-known notation</td>
</tr>
<tr>
<td>Transformation tool</td>
<td>Execution time</td>
</tr>
<tr>
<td></td>
<td>Maximum capability provable syntactic correctness Correctness: termination, confluence Effectiveness Fault tolerance Interoperability with Eclipse Maturity</td>
</tr>
</tbody>
</table>

Table 4.1: Evaluation criteria for the solutions of first and second case studies

3.

Table 4.2 classifies size as low, medium or high.

The categorization is based on the size values for the case studies solutions. In this evaluation the lower the lines of code the better is the verbosity and conciseness of the approach and furthermore the more functional is the transformation approach. In the first case study the size of transformations are three, sixteen (two of the approaches), nineteen and twenty. It has been divided such that at least one approach exists in each category of low, medium and high. The first quartile of values is considered low, the second and third medium, and the 4th high. A lower LOC value corresponds to a higher quality. For the size categorization of the first case study the mid point between first quartile and second quartile is 9 and the mid point of the third quartile
4.3. EVALUATION CRITERIA

<table>
<thead>
<tr>
<th>Value range for first case study</th>
<th>Value range for second case study</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..9 lines</td>
<td>0..50 lines</td>
<td>Low</td>
</tr>
<tr>
<td>9..19 lines</td>
<td>51..100 lines</td>
<td>Medium</td>
</tr>
<tr>
<td>20+ lines</td>
<td>101+ lines</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 4.2: Size categories for the first case study

<table>
<thead>
<tr>
<th>Value range for first case study</th>
<th>Value range for second case study</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..25</td>
<td>0..154</td>
<td>Low</td>
</tr>
<tr>
<td>26..34</td>
<td>155..205</td>
<td>Medium</td>
</tr>
<tr>
<td>35+</td>
<td>206+</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 4.3: Complexity categories for the first case study

and forth quartile is 19.5. Similarly for the categorization of size in second case study the approximate quartile of values are considered.

The factors of syntactic and structural complexity are summed to provide an overall complexity measure of a specification. Syntactic complexity is due to the complexity of the expressions used within the specification. It is measured by summing the number of occurrences of operators (such as =, :) , features and entities (such as name, parent) in the transformation definition. Structural complexity of transformation is measured by total number of calls, the number of recursive calls and maximum call depth in the transformation definition. Table 4.3 gives the classification of complexity as low, medium or high for the solutions of the two case studies that are considered in this chapter. The classification is based on the distribution of measured values in the case studies solutions. In this evaluation the lower the complexity the more functional and maintainable is the approach. The complexity values for the case studies are quite close to each other. In the table of categorization they divided in a way that at least one value exists in each category of high, medium and low.

Extensibility considers how much effort is required to adapt the transformation to extended metamodels. For instance if anything is added to the tree metamodel does it reflect easily to the node metamodel in the target model? The extensibility depends on both the language and the case study. If the transformation is modular for a specific case study, it is more convenient to extend it. The higher is the extensibility the more portable is the transformation approach.
4.3. EVALUATION CRITERIA

The evaluation of *embedability* enables us to know how effectively the transformation can be reused within a larger refinement or re-expression transformations. The higher the embeddability the more functional is the transformation approach.

*Development effort* in each transformation approach is measured in person-minutes. The first quartile of values is considered *low*, the second and third *medium*, and the 4th *high*. I chose the mid point value to categories the attributes. For the execution time categorization the mid point of the first quartile and second quartile is 35 and the mid point of the third quartile and forth quartile is around 55. *Low* development effort is considered to be zero to 35 minutes, *medium* between 36 to 55, and *high* over 55 minutes for the first case study. Development effort is not measured for evaluation of the second case study, where the previously implemented version of transformations are used. The lower the development time the more functional is the transformation approach.

The *execution time* of the transformation is measured on a computer system running on standard Windows XP with 3 GB of main memory and dual processor core. The classification categories for execution time is based on the processing time of the model with hundred elements in the first group of test cases. The execution time is *low* for the approach if it executes this test case in less than 23 ms, *medium* if between 24 to 164 ms and, *high* for over 165 ms. The first quartile of values is considered *low*, the second and third *medium*, and the 4th *high*. I chose the mid point value to categories the attributes. For the execution time categorization the mid point of the first quartile and second quartile is 23 and the mid point of the third quartile and forth quartile is around 164.

The lower the execution time of transformation tool the more functional and efficient is the approach.

In addition, test cases with increasing size are given to the transformation approaches to measure the *maximum capability* of the tools. This feature is evaluated for the first case study and models with increasing size, in both horizontal and vertical levels are applied to the tools. The classification of maximum capability in horizontal level is shown in Table 4.4 and in vertical level is presented in Table 4.5. In the horizontal test the number of children of a tree at a particular level increases, while in the vertical test the depth of the tree increments. The higher is the maximum capability the tool the more efficient it is.
4.4. TEST CASES

<table>
<thead>
<tr>
<th>Value range for horizontal capability</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..10,000 elements</td>
<td>Low</td>
</tr>
<tr>
<td>10,001..50,000 elements</td>
<td>Medium</td>
</tr>
<tr>
<td>50,001+ elements</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 4.4: Value range for Maximum Capability in horizontal level for the first case study

<table>
<thead>
<tr>
<th>Value range for vertical capability</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..1,000 elements</td>
<td>Low</td>
</tr>
<tr>
<td>1,001..4,000 elements</td>
<td>Medium</td>
</tr>
<tr>
<td>4,001+ elements</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 4.5: Value range for Maximum Capability in vertical level for the first case study

In this section we specified the evaluation criteria for assessment of first and second case studies. For measures of size, complexity, execution time, maximum capability, development effort, maturity, factorisation and cohesion specific boundary values have chosen which give a high-level classification of measures (as low, medium and high). These boundary values are based on the actual distribution of values in the specific solutions, but can be varied depending on the needs of the evaluator. The classifications for the rest of independent criteria are presented in Chapter 3. In this chapter abstraction level, size, complexity, modularity, extensibility, interoperability, development effort, closeness to the well-know notations such as OCL, programming language or any established standard, execution time, maximum capability, correctness, effectiveness, fault tolerance, interoperability with Eclipse and maturity of each solution will be evaluated and then categorized according to the specified boundary values.

4.4 Test cases

The solutions for the first case study are tested with sixteen test cases of increasing size and complexity. These test cases investigate the influence of increasing the elements size on both horizontal and vertical levels on the execution time of the transformation.

In the first group of test cases, the elements increase horizontally, i.e. increasing
4.4. TEST CASES

Figure 4.4: Test Case 1 for evaluation of execution time in the first case study

on the amount of siblings on the same level. Four test cases are selected in this group with 100, 500, 1,000, 10,000, 50,000 and 100,000 elements in the second level of the tree (Figure 4.4). The second group of test cases increase vertically. Elements with depths of 100, 500, 1,000, 2,000, 4,000 and 6,000 are selected for this group (Figure 4.5). The last set of test cases consider the effect of both horizontal and vertical increases on the execution time of the transformation. The first test case in this group has a depth of two with ten elements in the first level and hundred elements in the second level. Then the second test case has one more level but, 1,000 elements in the third level. Following that, the third test case has two more levels with 10,000 elements in the forth level and 100,000 elements in the fifth level. Finally, the last test case has additional level with 1,000,000 elements in the ending level (Figure 4.6). Table 4.6 shows the number of elements in each test case in the third group. In the following sections the “tree to graph” transformation in different approaches is presented and these testcases will be applied to them.
4.5 QVT-Relational Solution for the First Case Study

The QVT-Relations formalism (QVT-R) is a notation for defining model transformations as sets of rules. Each rule consists of left and right-hand side object models showing generic structures of instances from the source and target metamodels. There is a when clause, specified in OCL, which defines necessary preconditions for the ap-
4.5. QVT-RELATIONAL SOLUTION FOR THE FIRST CASE STUDY

Figure 4.6: Test Case 3 for evaluation of execution time in the first case study

Applications of the rule, and a where clause, defining effects or postconditions that the rule should establish when it is completed. The “tree to graph” transformation can be defined by three QVT rules TreetoNodeMain, TreetoNode and TreetoEdge. The specification is structured as a set of rules, called relations. The notation enforce in the code below means that the rule is enforced by modifying a model (source or target) where necessary. The checkonly notation indicates that the model is checked but not modified by the rule. It is also possible for both the source and target to be enforced, supporting bidirectional transformations, although there are semantic problems with such transformations [141]. A top relation is executed upon all elements in the source model to which it is applicable, in an arbitrary order, other relations are invoked (directly or indirectly) from top relations.
4.5. QVT-RELATIONAL SOLUTION FOR THE FIRST CASE STUDY

<table>
<thead>
<tr>
<th></th>
<th>Testcase 1</th>
<th>Testcase 2</th>
<th>Testcase 3</th>
<th>Testcase 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Level 2</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Level 3</td>
<td>-</td>
<td>1,000</td>
<td>1,000</td>
<td>1,000</td>
</tr>
<tr>
<td>Level 4</td>
<td>-</td>
<td>-</td>
<td>10,000</td>
<td>10,000</td>
</tr>
<tr>
<td>Level 5</td>
<td>-</td>
<td>-</td>
<td>100,000</td>
<td>100,000</td>
</tr>
<tr>
<td>Level 6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1,000,000</td>
</tr>
<tr>
<td>Total</td>
<td>110</td>
<td>1,110</td>
<td>111,110</td>
<td>1,111,110</td>
</tr>
</tbody>
</table>

Table 4.6: Number of elements in each level of test cases in group 3 for evaluation of first case study

The top relation *Tree2NodeMain*, takes the tree elements in the source model and generates corresponding graph elements in the target with the same name. This relation is applicable to all the trees which have no parent relation. This is checked in the *when* clause of the rule.

```java
1 transformation Tree2Graph(source : Tree, target : Node) {
2   top relation Tree2NodeMain {
3     tn : String;
4     checkonly domain source s : Tree::Treeclass { name = tn };
5     enforce domain target t : Node::Nodeclass { name = tn };
6     when { s . parent .oclIsUndefined(); }
7   }
8 }
9}
```

The relation *Tree2Edge* generates *edges* in the target model for all *tree* elements in the source model with the *parent* relation.

```java
1 top relation Tree2Edge {
2   checkonly domain source s : Tree::Treeclass {}; 
3   enforce domain target t : Node::Edgeclass {}; 
4   when { not s . parent .oclIsUndefined(); } 
5   where { Tree2Node(s, t.sourceref); 
6       Tree2Node(s.parent, t.targetref); 
7   } 
8 } 
9 }
```

In addition, another operation *Tree2Node* is called to map the *name* of tree elements to the *name* of graph elements for the child and its *parent*.

```java
1 relation Tree2Node {
2   tn : String;
3   checkonly domain source s : Tree::Treeclass { name = tn };
4   enforce domain target t : Node::Nodeclass { name = tn };
5 }
```
4.5. QVT-RELATIONAL SOLUTION FOR THE FIRST CASE STUDY

<table>
<thead>
<tr>
<th>Transformation language</th>
<th>Abstraction level</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size: LOC</td>
<td>20 lines (High)</td>
</tr>
<tr>
<td>Syntactic complexity</td>
<td>(no. of operators, no. of features)</td>
<td>13 operators, 17 features</td>
</tr>
<tr>
<td>Structural complexity</td>
<td></td>
<td>2 calls, 0 recursive calls, max call depth = 1</td>
</tr>
<tr>
<td>Total complexity</td>
<td></td>
<td>(33) Medium</td>
</tr>
<tr>
<td>Modularity</td>
<td>factorisation: 20%</td>
<td>cohesion: 100%, coupling: 0%</td>
</tr>
<tr>
<td>Modularity total</td>
<td></td>
<td>Medium</td>
</tr>
<tr>
<td>Extensibility:</td>
<td>extension to tree metamodel</td>
<td>Low</td>
</tr>
<tr>
<td>Interoperability:</td>
<td>inclusion in a transformation process</td>
<td>Low</td>
</tr>
<tr>
<td>Development effort</td>
<td>65 min (High)</td>
<td>High</td>
</tr>
<tr>
<td>Close to well-known notation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transformation tool</td>
<td>Execution time</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Maximum capability (Horizontal)</td>
<td>10,000 model elements (Low)</td>
</tr>
<tr>
<td></td>
<td>Maximum capability (Vertical)</td>
<td>1,000 model elements (Low)</td>
</tr>
<tr>
<td></td>
<td>Syntactic correctness</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td>Correctness (Termination)</td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>Correctness (Confluence)</td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>Effectiveness</td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>Fault tolerance</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Eclipse plug-ins</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Maturity</td>
<td>Medium</td>
</tr>
</tbody>
</table>

Table 4.7: Evaluation criteria for QVT-Relational Solution for the First Case Study

4.5.1 Evaluation properties of QVT-Relational

Table 4.7 summarises the evaluated attributes of the transformation language and tools for the QVT-R solution.

Transformation Language Properties

The notation is at a high level of abstraction, close to the form of the constraints which define the requirements, so in principle, specifications can be directly validated. Each QVT-R rule has a mathematical interpretation as a predicate (based on the when and where clauses). The transformation is specified with twenty lines of code which
4.5. QVT-RELATIONAL SOLUTION FOR THE FIRST CASE STUDY

classified as high according to the Table 4.2.

The number of operators such as `oclIsUndefined` are thirteen and the number of features such as `name` are seventeen in the QVT-R solution. There are two explicit calls with maximum depth of one and no recursive calls in the transformation specification. The sum of all these features provides medium complexity for this approach. The minimum syntactic complexity is three for measurement of factorization in the first case study. The expressions with minimum size of three syntactic in this specification are `name = tn` and `s.parent.oclIsUndefined()`. The total number of expressions are five as `name = tn` expression is repeated four times. Factorisation in QVT-R specification for this case study is one over five because of the existence of a single unique expression. Obviously, the factorisation can be improved by factorising the repeated parts of the rule such as `name = tn`. Moreover, the transformation specification is located inside one individual module which lead to a high cohesion in this approach. It has to be mention that we consider coupling and cohesion within the module and not within the rule. It is considered as a future extension for the evaluation task. In total, modularity is medium according to the definition given in Chapter 3.

Extension to a larger metamodel is possible and requires to change all the rules in the specification. However, this is not a trivial task in QVT-R approach. Since the notation is based closely upon UML and OCL, and the use of MOF metamodels, interoperability with other UML-based development tools should be possible. External composition relates to the case when individual transformations are composed while in internal composition the internal rules of a particular transformation are composed. The fact that QVT-R needs a distinguished “top relation” means that it can not clearly possible to compose the relations in this language.

The development effort for generating this case study was high for an non-expert developer. The developer does not need to define trace links as they are automatically created and managed in QVT-R. The result of usability survey showed that respondents considered the specification to have a clear structure, but still requires large effort to be understood.
4.5. QVT-RELATIONAL SOLUTION FOR THE FIRST CASE STUDY

<table>
<thead>
<tr>
<th>Test case</th>
<th>number of elements</th>
<th>Execution time(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>297</td>
</tr>
<tr>
<td>2</td>
<td>500</td>
<td>3611</td>
</tr>
<tr>
<td>3</td>
<td>1,000</td>
<td>26270</td>
</tr>
<tr>
<td>4</td>
<td>10,000</td>
<td>73942</td>
</tr>
<tr>
<td>5</td>
<td>50,000</td>
<td>Run out of Stack</td>
</tr>
<tr>
<td>6</td>
<td>100,000</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.8: Group 1-Test Cases results for QVT-Relational

<table>
<thead>
<tr>
<th>Test case</th>
<th>number of elements</th>
<th>Execution time(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>2469</td>
</tr>
<tr>
<td>2</td>
<td>500</td>
<td>2890</td>
</tr>
<tr>
<td>3</td>
<td>1000</td>
<td>7189</td>
</tr>
<tr>
<td>4</td>
<td>2000</td>
<td>Run out of Stack</td>
</tr>
<tr>
<td>4</td>
<td>4000</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>6000</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.9: Group 2-Test Cases results for QVT-Relational

Transformation Tool Properties

The transformation runs in MediniQVT tool [101]. The transformation execution time is high and around 250 ms for the testcase with hundred elements in the second level of the tree. The results of test cases for evaluation of maximum capability presents in Tables 4.8, 4.9 and 4.10. The maximum capability in horizontal level is 10,000 elements and in vertical level is 1000 elements. The MediniQVT has limited capacity for performing large test cases because it is executed by an interpreter. The “Run out of Stack” error is given in MediniQVT, which is some type of out of memory error.

Syntactic correctness and definedness for QVT-Relations specifications can be analysed by expressing these properties as logical formulae in OCL and then using an analysis tool for OCL [28]. However, there is no check for rule completeness or semantic correctness. Moreover, there is only one possible outcome from the implicit execution order of the rules. In this example the rules TreeToNode(t) and TreeToNode(t.parent) must be applied before the application of TreeToEdge(t) rule is
completed. The reason for this is due to the \emph{where} clause in the \textit{TreetoEdge} rule. However, the elements can be in any order for the application of the \textit{TreetoEdge} rule. The termination of the QVT-R solution is medium as there is no recursive calls in the specification. The order of the application of the rules are not explicitly defined in this approach.

Medini QVT has been available since 2007 [101]. It provides a run-time exception handling including errors in processing result in the termination of the transformation and provides an error message giving the specification lines where the error occurred. QVT-R was introduced by OMG as an international standard [117]. Version 1.0 was released in 2008, and there are twenty published case studies using this approach.

### 4.6 Kermeta Solution for the First Case Study

Kermeta is a Java-like object-oriented programming language, constructed on the type system of the Meta Object Framework (MOF) [113]. Kermeta can be used to define metamodels, as sets of classes, and to transform models of these metamodels. Several different styles of transformations specification are possible with Kermeta. Within each source language metaclass, we could define operations to specify the transformation rules applicable to this type of source model element. Alternatively, we may define a separate transformation metaclass that is independent of the source or target metamodels. The code for transformation of “tree to graph” is listed below. The operation \texttt{createNodes} applies a recursion down the structure of a tree, from the parent to its children, which maps trees into nodes and edges. Such operations can be iterated over all the \textit{Tree} elements in a set by using an iterator operator such as each:

```
def createNodes(tree):
    nodes = []
    for child in tree:
        nodes.append(createNodes(child))
    return nodes
```

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Execution Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6031</td>
</tr>
<tr>
<td>2</td>
<td>7360</td>
</tr>
<tr>
<td>3</td>
<td>Run out of Stack</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.10: Group 3-Test Cases results for QVT-Relational
trees.each { t | createNodes(t, root) }

Other OCL collection operators such as `forAll`, `select` and `collect` also have implementations in Kermeta. The `main` method is an entry point for the transformation in this implementation.

```kotlin
class Tree2Graph {
  operation createNodes(inputParent: TMM1, inputRoot: Root) : NMM2 is do
    var nodeForParent : NMM2 init NMM.new
    nodeForParent.name := inputParent.name
    inputRoot.Nodes.add(nodeForParent)
    inputParent.child.each { t |
      stdio.writeln("Now create Node for child " + t.name)
      var nodeForChild := createNodes(t, inputRoot)
      var newEdge : EdgeMM init EdgeMM.new
      newEdge.source := nodeForChild
      newEdge.target := nodeForParent
      inputRoot.Edges.add(newEdge)
    }
    result := nodeForParent
  end

  operation main() : Void is do
    var root : Root init Root.new
    createNodes(inputModel : TMM1) is loadTMM1()
    var repository : EMFRepository init EMFRepository.new
    var resource : Resource init repository.createResource("platform:/resource/models/NMM2.xmi", "platform:/resource.metamodel/NodeMM2.ecore")
    resource.instances.add(root)
    resource.save()
  end
}
```

### 4.6.1 Evaluation properties for Kermeta

Table 4.11 shows the summary of evaluated attributes for Kermeta solution.

**Transformation Language properties**

As can be seen, the level of abstraction is quite low compared to the other languages considered in this research. The specification explicitly defines model management steps such as the addition of new edges and nodes to the target model which are implicit in the QVT specification. Kermeta allows the definition of pre and post conditions for operations. These can be checked at runtime to detect erroneous processing of models. The size of transformation in Kermeta approach for the first case study is medium (excluding `main()` operation because it initialises the model,
## 4.6. KERMETA SOLUTION FOR THE FIRST CASE STUDY

<table>
<thead>
<tr>
<th>Transformation language</th>
<th>Abstraction level</th>
<th>Size: LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low</td>
<td>16 lines (Medium)</td>
</tr>
<tr>
<td>Syntactic complexity</td>
<td>11 operators, 19 features</td>
<td></td>
</tr>
<tr>
<td>(no. of operators, no. of features)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structural complexity</td>
<td>2 calls, 1 recursive calls, max call depth = 0</td>
<td></td>
</tr>
<tr>
<td>Complexity total</td>
<td>(33)Medium</td>
<td></td>
</tr>
<tr>
<td>Modularity</td>
<td>factorisation: 100%</td>
<td></td>
</tr>
<tr>
<td>Modularity total</td>
<td>cohesion: 100%, coupling: 0%</td>
<td></td>
</tr>
<tr>
<td>Extensibility:</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>extension to tree metamodel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interoperability:</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>inclusion in a transformation process</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Development effort</td>
<td>30 min (Low)</td>
<td></td>
</tr>
<tr>
<td>Close to well-known notation</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Transformation tool</td>
<td>Medium</td>
<td></td>
</tr>
<tr>
<td>Execution time</td>
<td>20,000 model elements (Medium)</td>
<td></td>
</tr>
<tr>
<td>Maximum capability (Horizontal)</td>
<td>1,100 model elements (Medium)</td>
<td></td>
</tr>
<tr>
<td>Maximum capability (Vertical)</td>
<td>Medium</td>
<td></td>
</tr>
<tr>
<td>Syntactic correctness</td>
<td>Low</td>
<td></td>
</tr>
<tr>
<td>Correctness (Termination)</td>
<td>Medium</td>
<td></td>
</tr>
<tr>
<td>Correctness (Confluence)</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Effectiveness</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Fault tolerance</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Eclipse plug-ins</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Maturity</td>
<td>Medium</td>
<td></td>
</tr>
</tbody>
</table>

|                     | Medium           |
|                     | 30 min (Low)     |
|                     | High             |
|                     | High             |
|                     | High             |
|                     | Medium           |

Table 4.11: Evaluation criteria for Kermeta Solution for the First Case Study

locates the source and target language which is done externally in non-imperative languages). The syntactic complexity is measured by counting the number of operators and features in the transformation implementation. There are eleven operations and nineteen features in this implementation. Additionally, there are two calls in the implementation, one in the main function and one in the `createNodes` operation, which is recursive. Recursive calls have a substantial effect on the complexity of specification. The total complexity of this approach is medium. The number of repeated expressions in the transformation is zero and factorisation is 100%. Moreover, the transformation is encapsulated in an individual module which result in high cohe-

1operators such as ::=, each, + and features such as name, child
4.6. KERMETA SOLUTION FOR THE FIRST CASE STUDY

In general, modularity is high and is provided in an object-oriented manner, by classes and packages.

Kermeta provides mechanisms to import and export models as objects. It is possible to extend the metamodel to the larger one by changing the classes in the specification (the metamodel is extended to cover comprehensive transformation). In addition, transformations can be easily combined using programming constructions which support both internal and external compositions.

It is most notable that the development effort is low because of the Object-Oriented nature of Kermeta. The standard notations such as OCL is used in the specification and interoperability with other MOF-based tools is also a possibility. In the usability survey, most of the respondents considered that the approach is more understandable and learnable but does not seem to be attractive to them. Tracing must be implemented explicitly by the developer.

Transformation Tool Properties

The execution time of transformation is medium and around thirty-one ms for model with hundred elements in the first group of the test cases. The result of the test cases are shown in Tables 4.12, 4.6.1 and 4.14. Kermeta is considered as one of the more favourable approaches for model transformation in respect to its similarity to Object-Oriented languages like Java. However, it is essential to pay attention to memory and stack overflow limitations for larger input models. The evaluation shows that an early stack overflow are often experienced due to the use of recursion in the specification. Furthermore, the more calls exist in the solution, the more effort is needed to handle the transformation task. The finding showed that the execution time of the transformation increases dramatically by increasing the elements of the model. Additionally, Kermeta is able to handle more elements in flat trees than vertical trees, but still it encounters the heap overflow issue.

There is no direct support for establishing syntactic correctness; however assertions and invariants could be used to check this for individual models. In addition, there is no support for checking definedness, completeness or semantic correctness. In contrast to QVT-R, the order of rule applications is explicit, and must be defined by the programmer. While this can result in larger and more complex specifications, the
4.7. ATL Solution for the First Case Study

<table>
<thead>
<tr>
<th>Test case</th>
<th>number of elements</th>
<th>Execution time(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>31</td>
</tr>
<tr>
<td>2</td>
<td>500</td>
<td>109</td>
</tr>
<tr>
<td>3</td>
<td>1,000</td>
<td>187</td>
</tr>
<tr>
<td>4</td>
<td>10,000</td>
<td>3297</td>
</tr>
<tr>
<td>5</td>
<td>20,000</td>
<td>7297</td>
</tr>
<tr>
<td>6</td>
<td>50,000</td>
<td>Run out of stack</td>
</tr>
<tr>
<td>7</td>
<td>100,000</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.12: Group 1-Test Cases results for Kermeta

<table>
<thead>
<tr>
<th>Test case</th>
<th>number of elements</th>
<th>Execution time(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>31</td>
</tr>
<tr>
<td>2</td>
<td>500</td>
<td>110</td>
</tr>
<tr>
<td>3</td>
<td>1,000</td>
<td>345</td>
</tr>
<tr>
<td>4</td>
<td>2,000</td>
<td>Run out of stack</td>
</tr>
<tr>
<td>5</td>
<td>4,000</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>6,000</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.13: Group 2-Test Cases results for Kermeta

direct control by the developer over the transformation processing may reduce the possibilities of semantic errors and increase the effectiveness of the tool. In addition, through this method we can impose a specific deterministic processing to achieve confluence. In general, termination of Kermeta is direct with the explicit nature of the algorithm. However, in this solution a recursive call makes the termination and confluence unclear.

As is observed, Kermeta is an effective tool to tackle this kind of transformation problem. The robustness is high for such an approach. Further it has good support for interconnection with Eclipse. Kermeta has been available since 2005, and it has an extensive history of use, with over forty published case studies (e.g., [108, 105]).

4.7 ATL Solution for the First Case Study

The Atlamod Transformation Language (ATL) is a hybrid model transformation language. The specification consists of a set of rules with source and target pattern,
4.7. ATL SOLUTION FOR THE FIRST CASE STUDY

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Execution Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>47</td>
</tr>
<tr>
<td>2</td>
<td>172</td>
</tr>
<tr>
<td>3</td>
<td>Run out of stack</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.14: Group 3-Test Cases results for Kermeta

specified using OCL. A rule application occurs when the source pattern of the rule matches some part of the source model. The corresponding elements that satisfy the target pattern are then created in the target model. A rule may implicitly call another rule, or inherit from another rule, and may also explicitly invoke other rules (similarly to the invocation of relations within the where clause of a QVT-R rule). In ATL the tree to graph transformation can be coded as follows:

```plaintext
module TreeToGraph; -- Module Template
create OUT : NodeMM from IN : TreeMM ;
rule Tree2Node {  
  from t : TreeMM!Tree(t.parent->notEmpty() and not t.parent=t)  
  to out : NodeMM!Node(  
    name <- t.name )  
  edg : NodeMM!Edge(  
    source <- out,  
    target <- t.parent )  
}
rule Tree2NodeSecond {  
  from t : TreeMM!Tree(not t.parent->notEmpty() or t.parent=t)  
  to out : NodeMM!Node(  
    name <- t.name )  
}
```

The `Tree2Node` rule processes tree nodes with `parents` (distinct from the node), and creates both a new graph node for the tree, and an edge connecting it to (the node corresponding to) its parent. The second rule processes tree nodes without `parents`, and creates only a new corresponding graph node. When setting `edg.target`, the first rule implicitly depends upon itself or the second rule to map `t.parent` to a graph node which can be assigned to this reference.

4.7.1 Evaluation properties for ATL

The evaluation of ATL properties for the first case study is presented in Table 4.15.
4.7. ATL SOLUTION FOR THE FIRST CASE STUDY

<table>
<thead>
<tr>
<th>Transformation language</th>
<th>Abstraction level</th>
<th>Medium</th>
<th>Size: LOC</th>
<th>19 (Medium)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>syntactic complexity</td>
<td>18 operators, 18 features</td>
<td>1 calls, 0 recursive calls, max call depth = 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Structural complexity</td>
<td>18 operators, 18 features</td>
<td>1 calls, 0 recursive calls, max call depth = 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Complexity total</td>
<td>(38) High</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Modularity</td>
<td>factorisation: 50%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
|                         | Modularity total | cohesion: 100%, coupling: 0%
|                         | Medium |
|                         | Extensibility: | Medium |
|                         | extension to tree metamodel |
|                         | Interoperability: | High |
|                         | inclusion in a transformation process |
|                         | Development effort | 40 min (Medium) |
|                         | Close to well-known notation | High |
| Transformation tool      | Execution time | Low | 50,000 model elements (Medium) |
|                         | Maximum capability (Horizontal) | Medium | 4000 model elements (Medium) |
|                         | Maximum capability (Vertical) | Medium |
|                         | syntactic correctness | Medium |
|                         | Correctness (Termination) | Low |
|                         | Correctness (Confluence) | Medium |
|                         | Effectiveness | High |
|                         | Fault tolerance | High |
|                         | Eclipse plug-ins | High |
|                         | Maturity | High |

Table 4.15: Evaluation criteria for ATL Solution for the First Case Study

Transformation Language Properties

The rules in ATL are at a relatively high level of abstraction. A logical interpretation of the rules, similar to that of QVT-R rules, can be constructed. The size of transformation is medium with nineteen lines specification.

There are 18 operators such as $\rightarrow$, and, not, !, $<$ and features such as name, parent in the specification. There is only one implicit call in the specification (line 11). However, the complexity of transformation is high for this approach. There are eight expressions with more than a minimum size (three in this case) in this specification.
4.7. ATL SOLUTION FOR THE FIRST CASE STUDY

², and four of them are repeated. Therefore, the factorisation for this approach is four over eight. In addition, the specification is located inside one module. It has to be mention that we consider coupling and cohesion within the module and not within the rule. It is considered as a future extension for the evaluation task. The modularity of ATL is medium and at the level of rules.

It can be identified that extension to the larger metamodel requires extra changes in the model and may need to cover more lines of code. In addition, one ATL transformation can be sequentially composed with another transformation which is defined in separate modules by means of a Java script. It means it is possible to extract the ATL specification in code and then reused it in other transformation tasks externally. Moreover, we can provide internal composition within the rules in ATL.

The development effort for generating this case study was medium and expert assistance was available. The next highly important factor of ATL is to be interoperable with other MOF-based development tools [63]. Usability survey shows that participants considered this approach less understandable, but attractive. Trace links are automatically created and managed and do not need to be defined by the developer. ATL rules are compiled into an executable format, ATL VM [63].

Transformation Tool Properties

The ATL approach generates the target result of input models with hundred elements in an appropriate time. The transformation runs in Eclipse. The maximum capability of the tool is medium. The result of test cases are shown in tables 4.16, 4.17 and 4.18. The ATL tool is capable of handling the input models with 50,000 elements in horizontal level and 4,000 elements in vertical level.

There are no checks for rule completeness or semantic correctness. There is no recursions in the specification of both case studies and the termination of the solution is medium. The Tree2Node rule has linear iterations over their source domains. With ATL there is no direct tool support for syntactic correctness or definedness, although the techniques of [28] could in principle be used to support this. Alongside, OCL tools

²\(t.\text{parent} \rightarrow \text{notEmpty}()\), \(t.\text{parent} = t\), not \(t.\text{parent} = t\), \(t.\text{parent} \rightarrow \text{notEmpty}()\) and not \(t.\text{parent} = t\), not \(t.\text{parent} \rightarrow \text{notEmpty}()\), \(t.\text{parent} \rightarrow \text{notEmpty}()\), \(t.\text{parent} = t\), not \(t.\text{parent} \rightarrow \text{notEmpty}()\) or \(t.\text{parent} = t\)
4.8. UML-RSDS SOLUTION FOR THE FIRST CASE STUDY

<table>
<thead>
<tr>
<th>Test case</th>
<th>number of elements</th>
<th>Execution time(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>500</td>
<td>31</td>
</tr>
<tr>
<td>3</td>
<td>1,000</td>
<td>31</td>
</tr>
<tr>
<td>4</td>
<td>10,000</td>
<td>750</td>
</tr>
<tr>
<td>5</td>
<td>50,000</td>
<td>2672</td>
</tr>
<tr>
<td>6</td>
<td>100,000</td>
<td>Run out of heap</td>
</tr>
</tbody>
</table>

Table 4.16: Group 1-Test Cases results for ATL

<table>
<thead>
<tr>
<th>Test case</th>
<th>number of elements</th>
<th>Execution time(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>500</td>
<td>31</td>
</tr>
<tr>
<td>3</td>
<td>1000</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>2000</td>
<td>63</td>
</tr>
<tr>
<td>4</td>
<td>4000</td>
<td>140</td>
</tr>
<tr>
<td>5</td>
<td>6000</td>
<td>Run out of heap</td>
</tr>
</tbody>
</table>

Table 4.17: Group 2-Test cases results for ATL

could be used to check the syntax and type-correctness of the rule patterns. There are checks for rule consistency: if two rules are both applicable at the same time to the same model, an error is given. This helps to identify some cases of ambiguity in the specification, but does not ensure confluence in all cases.

The ATL tool behaves reasonably in the presence of false assumptions. The effectiveness of transformation is high. In addition ATL generates an appropriate target model from the large scale inputs in an efficient time.

It has been reported that the first version of ATL was released in 2003. Then in the beginning of 2006, it became an Eclipse project. There are several case studies of industries and academics using ATL for the purpose of transformation.

4.8 UML-RSDS Solution for the First Case Study

UML-RSDS is a model-driven development approach which generates executable systems from high-level specifications consisting of class diagrams, constraints and state machines [82]. It can be used to specify model transformations by formalising them
4.8. UML-RSDS SOLUTION FOR THE FIRST CASE STUDY

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Execution Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>62</td>
</tr>
<tr>
<td>2</td>
<td>78</td>
</tr>
<tr>
<td>3</td>
<td>6687</td>
</tr>
<tr>
<td>4</td>
<td>Run out of heap</td>
</tr>
</tbody>
</table>

Table 4.18: Group 3-Test cases results for ATL

as constraints or operations upon metamodels. Code generation can then be used to produce executable implementations of the model transformations. However, direct code generation from such high-level constraints may produce highly inefficient code. Instead, the constraints can be refined by defining explicit transformation rules as operations, specified by precondition and postcondition predicates, which in turn define how particular elements of the source model are mapped to the target model. As in Kermeta, these operations may be placed within the source model metaclasses, or placed in new classes, external, but referring to the source or target metamodels. In this example, the mapping from trees to graph nodes could be expressed by two operations (these are generated from C1 and C2):

\[
\begin{align*}
    mapTreeToNode & \quad \forall t : Tree \cdot \exists n : Node \cdot n.name = t.name \\
    mapTreeToEdge & \quad \forall t : Tree \cdot t \neq t.parent \rightarrow \\
                      & \quad \exists e : Edge \cdot e.source = Node[t.name] \text{ and } e.target = Node[t.parent.name]
\end{align*}
\]

We assume there are primary keys in this example. The notation \( Node[x] \) refers to the node object with primary key (in this case name) value equal to \( x \), it is implemented in the UML-RSDS tools by maintaining a map from the key values to the nodes. In OCL it would be expressed as

\[
Node.allInstances() \rightarrow select(name = x) \rightarrow any()
\]

Likewise, \( Node.name \) abbreviates

\[
Node.allInstances() \rightarrow collect(name)
\]

This is done explicitly in UML-RSDS and there no implicit conversion in this approach.
4.8. UML-RSDS SOLUTION FOR THE FIRST CASE STUDY

At the design level, the transformation is defined as a UML class with a behaviour defined by an activity. This controls the allowed order of application of the rules. In this example this order is to first iterate all the tree-to-node mappings, then the tree-to-edge mappings:

\[
\begin{align*}
\text{for } t : Tree & \text{ do } \text{mapTreeToNode}(t) ; \\
\text{for } t : Tree & \text{ do } \text{mapTreeToEdge}(t)
\end{align*}
\]

4.8.1 Evaluation properties for UML-RSDS

Table 4.19 summarises the evaluated attributes of the transformation language and tools for the UML-RSDS solution.

Transformation Language properties

The level of abstraction of the rule-based specification is high, and is close to the requirements, permitting validation. There are three lines of code for performing this case which is considerably low in comparison to other approaches.

The syntactic complexity is measured by counting total number of operators and features \(^3\) in the specification. There are twelve operators and thirteen features in the specification of this language. There is no call in the specification and in total the complexity of specification for this approach is low. The number of expressions greater than the minimum size are six. The number of repeated expressions are zero and, therefore factorisation is 100%. The transformation specification is small and encapsulated inside one module.

The recommended specification style for model transformations is to avoid dependencies between rules, and to use an explicit control expression (behaviour) to define rule application orders. The solution can be used for the extended tree metamodel and requires changes to each rule. For UML-RSDS development effort was medium and expert assistance was available. Since UML-RSDS is based upon MOF and UML, interoperability with other UML development methods and tools should be possible. For this language no new notation beyond UML and OCL is required (it may use abbreviation of OCL such as \(Node[x]\)), and the transformation execution is explicit.

\(^3\)operators such as \(\exists, \Rightarrow, =\) and features such as \(Node, Tree, name\)
4.8. UML-RSDS SOLUTION FOR THE FIRST CASE STUDY

<table>
<thead>
<tr>
<th>Transformation language</th>
<th>Abstraction level</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size: LOC</td>
<td>3 lines (Low)</td>
<td></td>
</tr>
<tr>
<td>Syntactic complexity</td>
<td>12 operators, 13 features</td>
<td></td>
</tr>
<tr>
<td>(no. of operators, no. of features)</td>
<td>0 call</td>
<td></td>
</tr>
<tr>
<td>Structural complexity</td>
<td>0 recursive calls, max call depth = 0</td>
<td></td>
</tr>
<tr>
<td>Complexity total</td>
<td>25 (Low)</td>
<td></td>
</tr>
<tr>
<td>Modularity</td>
<td>factorisation: 100%, cohesion: 100%, coupling: 0%</td>
<td></td>
</tr>
<tr>
<td>Modularity total</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Extensibility:</td>
<td>Medium</td>
<td></td>
</tr>
<tr>
<td>extension to tree metamodel</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Interoperability:</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>inclusion in a transformation process</td>
<td>45 min (Medium)</td>
<td></td>
</tr>
<tr>
<td>Development effort</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Close to well-known notation</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Transformation tool</td>
<td>Execution time</td>
<td>Low</td>
</tr>
<tr>
<td>Maximum capability (Horizontal)</td>
<td>200000 model elements (High)</td>
<td></td>
</tr>
<tr>
<td>Maximum capability (Vertical)</td>
<td>200000 model elements (High)</td>
<td></td>
</tr>
<tr>
<td>Syntactic correctness</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Correctness (Termination)</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Correctness (Confluence)</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Effectiveness</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Fault tolerance</td>
<td>Medium</td>
<td></td>
</tr>
<tr>
<td>Eclipse plug-ins</td>
<td>Low</td>
<td></td>
</tr>
<tr>
<td>Maturity</td>
<td>Low</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.19: Evaluation criteria for UML-RSDS Solution for the First Case Study

Tracing of the rule applications must be performed explicitly by the developer if required. The use of identity attributes is recommended as an alternative means of propagating changes from the source model to the target model, by identifying which target model elements are semantically linked to which source model elements. In the specification of the first case study in UML-RSDS a tree is implicitly linked to the graph node with the same name. This is done independently of which model transformation rules were used to create the target model from the source.

Implementation is by translation to Java and the generated code is close in structure to the specification, which assists in comprehension. The usability survey for UML-RSDS shows that this approach is less understandable, attractive and learnable
4.8. UML-RSDS SOLUTION FOR THE FIRST CASE STUDY

in comparison to other approaches investigated in this study.

Transformation Tool Properties

The transformation is extremely fast and generates the result in less than one ms. UML-RSDS performs the tree to graph transformation task in two iterations. The result of test cases in Tables 4.20, 4.21 and 4.22 show that UML-RSDS has a simple linear solution which has identical execution time for flat and vertical trees. In this solution the only limitation relates to the heap capacity for performing large test cases. The maximum capability of the UML-RSDS tool is a model with 200,000 elements.

Syntactic correctness of individual rules can be proved by using an automated translation from UML-RSDS to the B formal notation [81], and applying proof within B. For example, it could be proven that the mapTreeToEdge ensures the graph metamodel constraint that the source and target of all edges are distinct. The rule mapTreeToEdge is not executed unless the rule mapTreeToNode is executed before hand. It results to have confluence in this transformation. Alongside, the termination for UML-RSDS is high because the bounded loops are used for both rules in this solution. Furthermore, the transformation in UML-RSDS is quite robust. The UML-RSDS tool checks completeness of rules by checking that all data features of an object are set in the operation which creates it. For example, in the operation mapTreeToEdge, an error message would be given if there was no assignment to the target of the new edge. Procedures for checking semantic correctness exist but have not been automated. Likewise, there are rules to determine when unordered iterations are confluent [86], but these are not incorporated into the toolset.

The assumptions of transformation is checked when a model is loaded. The runtime error is not checked in the transformation tool. The UML-RSDS tool presents the worst result in terms of fault tolerance in comparison to other approaches in this thesis. The UML-RSDS has short history of use (three years) with no Eclipse plugins. Only recently it enables transfer data file (xml) of this approach to Eclipse but it is still not complete.
4.9. VIATRA SOLUTION FOR THE FIRST CASE STUDY

<table>
<thead>
<tr>
<th>Test case</th>
<th>number of elements</th>
<th>Execution time(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>500</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1,000</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>10,000</td>
<td>88</td>
</tr>
<tr>
<td>5</td>
<td>50,000</td>
<td>150</td>
</tr>
<tr>
<td>6</td>
<td>100,000</td>
<td>169</td>
</tr>
</tbody>
</table>

Table 4.20: Group 1-Test Cases results for UML-RSDS

<table>
<thead>
<tr>
<th>Test case</th>
<th>number of elements</th>
<th>Execution time(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>500</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1,000</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>2,000</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>4,000</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>6,000</td>
<td>28</td>
</tr>
<tr>
<td>6</td>
<td>50,000</td>
<td>126</td>
</tr>
<tr>
<td>7</td>
<td>100,000</td>
<td>230</td>
</tr>
</tbody>
</table>

Table 4.21: Group 2-Test cases results for UML-RSDS

4.9 Viatra Solution for the First Case Study

Viatra [154] is a graph-based model transformation language. In this approach, the transformation rules are specified by graph matching and graph rewriting. A pattern language is used to select elements from the model, and a rule defines how these and related elements are modified. Unlike the other approaches discussed here, Viatra is not based upon MOF and OCL, but uses a different metamodelling formalism, VPM [154]. A procedural language, ASM, is used to explicitly schedule the application of rules using sequencing, loops, conditionals, etc. The “tree to graph” transformation can be specified in Viatra as a machine consisting of two graph transformation rules, each defined by a pre and postcondition pattern, and a top-level algorithm to control the order of their execution:

\[
\text{machine tree2graph} \\
\{ \text{gtrule mapTreeToNode(in T) =} \\
\]
4.9. VIATRA SOLUTION FOR THE FIRST CASE STUDY

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Execution Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>250</td>
</tr>
<tr>
<td>4</td>
<td>Run out of heap</td>
</tr>
</tbody>
</table>

Table 4.22: Group 3-Test cases results for UML-RSDS

```plaintext
{ precondition pattern isTree(T,Nme) =
  { tree(T); name(T,Nme); } }
postcondition pattern newNode(N,Nme) =
  { node(N); name(N,Nme); } }
grule mapTreeToEdge(in T) =
  { precondition pattern isTreeWithParent(T,N,P,N1) =
    { tree(T); parent(T,P); tree(P); check(T != P);
      name(T,Nme1); node(N); name(N,Nme1);
      name(P,Nme2); node(N1); name(N1,Nme2); }
  postcondition pattern newEdge(N,N1,E) =
    { edge(E); source(E,N); target(E,N1); } }
rule main() =
  seq {
    forall T with apply mapTreeToNode(T) do skip;
    forall T with apply mapTreeToEdge(T) do skip;
  }
```

The specification is similar to that of the UML-RSDS version in structure; however the notation used for rules is related to a logic-programming formalism such as Prolog instead of OCL.
### 4.9. VIATRA SOLUTION FOR THE FIRST CASE STUDY

<table>
<thead>
<tr>
<th>Transformation language</th>
<th>Abstraction level</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size: LOC</td>
<td>16 lines (Medium)</td>
<td></td>
</tr>
<tr>
<td>Syntactic complexity</td>
<td>20 operators, 16 features</td>
<td></td>
</tr>
<tr>
<td>(no. of operators, no. of features)</td>
<td>2 call</td>
<td></td>
</tr>
<tr>
<td>Structural complexity</td>
<td>0 recursive calls, max call depth = 1</td>
<td></td>
</tr>
<tr>
<td>Complexity total</td>
<td>(39) High</td>
<td></td>
</tr>
<tr>
<td>Modularity</td>
<td>factorisation: 100%</td>
<td></td>
</tr>
<tr>
<td>Modularity total</td>
<td>cohesion: 100%, coupling: 0%</td>
<td></td>
</tr>
<tr>
<td>High</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Extensibility:</td>
<td>Low</td>
<td></td>
</tr>
<tr>
<td>extension to tree metamodel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interoperability:</td>
<td>Low</td>
<td></td>
</tr>
<tr>
<td>inclusion in a transformation process</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Development effort</td>
<td>75 min (High)</td>
<td></td>
</tr>
<tr>
<td>Close to well-known notation</td>
<td>Low</td>
<td></td>
</tr>
<tr>
<td>Syntaxic correctness</td>
<td>Medium</td>
<td></td>
</tr>
<tr>
<td>Correctness (Termination)</td>
<td>Medium</td>
<td></td>
</tr>
<tr>
<td>Correctness (Confluence)</td>
<td>Medium</td>
<td></td>
</tr>
<tr>
<td>Effectiveness</td>
<td>Medium</td>
<td></td>
</tr>
<tr>
<td>Fault tolerance</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Eclipse plug-ins</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Maturity</td>
<td>High</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.23: Evaluation criteria for Viatra Solution for the First Case Study

### 4.9.1 Evaluation properties for Viatra

Table 4.23 summarises the evaluated attributes of the transformation language and tools for Viatra solution.

**Transformation Language properties**

The specification is at quite high level of abstraction and is organised in a modular manner consisting of modules with sets of rules and a control algorithm. The generated specification has moderate size in the first case study.

There are twenty operations and sixteen features in this specification with no calling relation between rules except the main function which has two calls for organizing the functionality of the program. The total complexity is high for this approach. The
factorisation is 100% because of unique expression in this specification. In addition, the transformation specification encapsulated inside one module which leads to have high modularity for this approach. The modular specification of the rule makes it possible to extend it to the larger metamodel; however it is difficult to change each particular rule. The user specifies input and output converters to use Viatra with other development methods.

It should be mentioned that Viatra is not based upon MOF and OCL, and uses a different metamodelling formalism. However, the interoperability with other tools is difficult and tracing needs to be managed explicitly by the developer. For Viatra the rules may be executed by interpretation, or compiled to a directly executable form. The usability survey shows that Viatra is considered as both understandable and learnable, but not attractive from the users’ point of view.

**Transformation Tool properties**

There is a partial check for syntactic correctness in this approach. The type checker tools can be applied for the validation of this approach. There is explicit scheduling of different rules, for Viatra, which makes it better than QVT-R and ATL, where there is only implicit scheduling by setting the conditions of the rules. But for each rule, it is not possible to control the order of applications on the individual elements that it is applied to (e.g., the order of the trees in `mapTreeToNode`). Some ASM constructs, such as `forall`, execute their statements in an indeterminate order which may create a problem for confluence.

The error message is shown by the parser for mismatch types and syntax errors. In addition, run-time errors are shown by the interpreter. The original version of this approach is called Viatra and the current generation is Viatra2. Development of the original Viatra began in 2000 and the first public release was in 2002. Later in 2004, Viatra2 was released. Until now there are more than fifty published case studies using the Viatra approach.
4.10. QVT-R SOLUTION FOR THE SECOND CASE STUDY

4.10 QVT-R Solution for the Second Case Study

The transformation of “Class diagram to Relational database” in QVT-R maps persistent Class of source model to the simple Relational database Table in the target [117]. Following that the Primary key is generated and Columns of the Table is identified. Moreover, this transformation considers the mapping of Attributes of the persistent Class to Column of the Table. However, depending on the type of Attribute different transformation may be accrued. An attribute of a primitive datatype maps to a single Column and Attribute of complex datatype maps to a set of columns corresponding to its set of primitive datatype attributes. Additionally, Attributes that are inherited from class hierarchy are mapped to the Columns of the table. Foreign key relationship are presented in the target model corresponding to an association between two persistent classes. The appendix A presents the specification of QVT-R for this case study.

Transformation Language properties

The size of QVT-R transformation for the second case study is 180. This is categorized as high according to the classification of values in Section 4.3. The complexity of a specification can be measured by analysing its call graph. Figure 4.7 presents the call graph of QVT-R for the second case study. Each node represents a rule and edges represent the calls from one rule to another, both implicit and explicit calls. The total numbers of calls in the call graph are ten. There are four recursive calls and six non-recursive calls in the specification. The complexity of expressions within the specification for the QVT-R approach is 163. The total complexity of this approach is medium. The transformation specification is factorized. The specification is divided into two modules and there is only one call within the modules. In total, the modularity is high for this approach. The rest of the attributes have the similar score to the fist case study and are not further discussed here. The development effort for evaluation of second case study is not considered because the published versions of solutions have been investigated.
4.11 KERMETA SOLUTION FOR THE SECOND CASE STUDY

The Kermeta implementation for “Class diagram to Relational database is presented in Appendix A. The operation transform maps all the persistent Class to the Table with the same name [35]. The Table is then added to the list of tables. The values are stored explicitly in Kermeta, but it is stored automatically in declarative languages like QVT-R. The implementation is then mapped the Attribute of Class to a Column of Table. Depending on the type of Attribute different mapping is presented. If the type of Attribute is Primitive a simple Column is generated in the target side. However, if the type of Attribute is Class additional ForeignKey is created. The specification is then focus on mapping of Association to Column. If the destination of Association is a persistent Class a ForeignKey is generated. This key is then added.
4.12. ATL SOLUTION FOR THE SECOND CASE STUDY

to the list of Foreign keys.

Transformation Language properties

The Kermeta approach has imperative style and the size and complexity of transformation increases for the larger case study. The size of Kermeta for the second transformation is 174, which is high according the classification of values in Section 4.3. The number of syntactic complexity are 188 and the number of calls are twenty-two from which nine are recursive calls. Figure 4.8 presents the call graph of Kermeta for the second case study. Each node represents a rule and edges represent the calls from one rule to another, both implicit and explicit calls. The factorization is 85%. The transformation is divided into two modules with total number of three calls within the modules. The percentage of coupling for this approach is 14% and cohesion is 86%. In total, the modularity of this approach is medium. The rest of attributes present the similar result as the first case study.

4.12 ATL Solution for the Second Case Study

The ATL transformation from “Class diagram to Relational Database” is given from [8]. This specification does not consider inheritance and mapping of associations. In ATL specification the relational database metamodel contains a set of columns and has a reference to its keys. There are two references as owner and keyOf for class Column that pointing to the Table it belongs to and of which it is part of the key (in case it is a key). Moreover, Column has a reference to Type. Figure 4.9 presents the relational database metamodel that is used for ATL specification.

The matched rule Class2Table transform a class model to a relational model. In this rule the name of Class is mapped to the name of Table. The col reference defined as a set to contain all Columns that have been created for single-valued attributes, while the key is also described. The reference key contains the pointer to the specified key. Additionally, the instance of Attribute needs to be created as key in this specification and its type reference needs to reference the Type with the name Integer. Following that for each DataType instance a Type instance with the same name is created. For each DataType with a single-valued Attribute instance a
Column instance with the same name has to be created. A Table is generated for each multi-valued Attribute instance of the type DataType. The name of the Table assigns to the name of Attribute of Class by using an underscore and name of the Attribute. In addition, the col reference set has to reference the two described Columns. In this implementation an identifier instance for Column need to be created. The name of the Class of Attribute is mapped to its name with “id” and its type need to reference a Type with the name Integer. Finally, it is essential to generate the Foreign key Column with the same name as the name of Attribute concatenated with “Id”. In addition, its type needs to reference a Type with the name Integer.
4.13. UML-RSDS SOLUTION FOR THE SECOND CASE STUDY

Transformation Language properties

The rule based specification language of ATL provides convenient mechanism to specify the transformation. This specification does not consider inheritance and mapping of associations. The size of this transformation is 86. The number of syntactic complexity is 188. Figure 4.10 shows the call graph of ATL for the second case study. Each node represents a rule and edges represent the calls from one rule to another, both implicit and explicit calls. In total, there are twenty-two number of calls in the specification. The factorization of transformation is 50%, However, it is encapsulated inside one module. The rest of properties are evaluated similar to the first case study.

4.13 UML-RSDS Solution for the Second Case Study

The formal specification of the transformation as a single global relation between the source and target languages can be split into six core constraints:

C1 “For each persistent attribute in the source model there is a unique column in
4.13. UML-RSDS SOLUTION FOR THE SECOND CASE STUDY

Figure 4.10: Calling dependencies of ATL solution for the second case study

C2 “For each persistent class in the source model, there is a unique table representing

\[
\forall a: \text{Attribute} \cdot a.\text{owner}.\text{kind} = \text{‘Persistent’} \implies \exists_1 cl: \text{Column} \cdot cl.\text{rdbId} = a.\text{umlId} \text{ and } cl.\text{name} = a.\text{name} \text{ and } cl.\text{kind} = a.\text{kind} \text{ and } \begin{cases} a.\text{type}.\text{name} = \text{‘INTEGER’} \implies cl.\text{type} = \text{‘NUMBER’} \end{cases} \text{ and } \begin{cases} a.\text{type}.\text{name} = \text{‘BOOLEAN’} \implies cl.\text{type} = \text{‘BOOLEAN’} \end{cases} \text{ and } \begin{cases} a.\text{type}.\text{name} \neq \text{‘INTEGER’} \text{ and } a.\text{type}.\text{name} \neq \text{‘BOOLEAN’} \implies cl.\text{type} = \text{‘VARCHAR’} \end{cases}
\]

the target model, of corresponding type”:

C2 “For each persistent class in the source model, there is a unique table representing
4.13. UML-RSDS SOLUTION FOR THE SECOND CASE STUDY

the class in the target model, with columns for each owned attribute”:

∀ c : Class · c.kind = ‘Persistent’ implies
    ∃ t : Table · t.rdbId = c.umlId and t.name = c.name and
        t.kind = ‘Persistent’ and
        Column[c.attribute.umlId] ⊆ t.column

For a set x, the notation Entity[x] refers to the set of Entity objects with
primary key (in this case rdbId) value in x, it can be implemented in Java by
maintaining a map from the key values to nodes. In OCL it would be expressed
as

Entity.allInstances()→select(x→includes(id))

For a single value y, Entity[y] denotes

Entity.allInstances()→select(id = y)→any()

C3 “For each root class in the source model there is a unique primary key in the
target model”:

∀ c : Class · c.kind = ‘Persistent’ and c.general = { } implies
    ∃ k : Key · k.rdbId = c.umlId + “_Pk” and k.name = c.name + “_Pk” and
        k.owner = Table[c.umlId] and k.kind = ‘PrimaryKey’ and
    ∃ cl : Column · cl.rdbId = c.umlId + “_Id” and
        cl.name = c.name + “_Id” and cl.type = ‘NUMBER’ and
        cl : k.column and cl.kind = ‘PrimaryKey’ and
        cl : k.owner.column

C4 “For each association in the source model, there is a unique foreign key repre-
senting it in the target model”:

\[ \forall a : \text{Association} \text{ and } a\.kind = '\text{Persistent}' \text{ implies } \exists f_k : \text{ForeignKey} \cdot f_k\.rdbId = a\.umlId + "Fk" \text{ and } \\
 f_k\.name = a\.name + "Fk" \text{ and } \\
 f_k\.owner = \text{Table}[a\.source\.umlId] \text{ and } \\
 f_k\.kind = 'association' \text{ and } \\
 f_k\.refersTo = \text{Table}[a\.destination\.umlId] \text{ and } \\
 \exists c_l : \text{Column} \cdot c_l\.rdbId = a\.umlId + "Ref" \text{ and } \\
 c_l\.name = a\.name + "Ref" \text{ and } \\
 c_l : f_k\.column \text{ and } c_l\.kind = 'ForeignKey' \text{ and } \\
 c_l\.type = 'NUMBER' \text{ and } \\
 c_l : f_k\.owner\.column \]

C5 “If \( c \) is a subclass of \( d \), all columns of \( d \)'s table are included in \( c \)'s table”:

\[ \forall c, d : \text{Class} \cdot c\.kind = '\text{Persistent}' \text{ and } d\.general \text{ implies } \\
 \text{Table}[d\.umlId].\text{column} \subseteq \text{Table}[c\.umlId].\text{column} \]

C6 “For each package in the source model, there is a unique schema representing it in the target model”:

\[ \forall p : \text{Package} \cdot \\
 \exists_1 s : \text{Schema} \cdot s\.rdbId = p\.umlId \text{ and } \\
 s\.name = p\.name \text{ and } s\.kind = p\.kind \text{ and } \\
 s\.tables = \text{Table}[p\.elements\.umlId] \]

This specification is superficially similar to the rule-based description of the transformation given in [117], however the key difference is that our specification is at a higher level of abstraction, defining the intended state of the target model at completion of the transformation (independently of how the transformation is implemented).

**Transformation Language properties**

The size of UML-RSDS is 35 for the second case study, which is extremely lower than other approaches. It is because of particular style of specification is this approach. However, the number of syntactic complexity is higher than other approaches. There
is no call in the specification of this approach and the specification is factorized. The rest of properties for UML-RSDS are presented the same result as the first case study.

### 4.14 Viatra Solution for the Second Case Study

The Vitara specification for transformation of “Class diagram to Relational database” is presented in [154]. The main function in this specification calls the rule of transformation in order. The model is then initialize through alternative rule “initModels”. The rule “class2tableR” checks the Class in its pre-condition and then maps them to the Table and PrimaryKey. The rule “attr2columnR” maps the Attribute of class to Column of the Table. However, in this rule a distinction has been made between Association and Attribute through a negative condition. Viatra specification defined alternative rules for mapping of attributes with String type and attributes with Integer type. Finally, Association is mapped to the Column and Foreign Key. The sub-rules including createNewTable, createPrimaryKeyInTable, createNewColumn, createNewColumnType and createNewFKey are also specified in Viatra and called within the high level rules. In Viatra a globally unique identifier called a fully qualified name (FQN) is defined for each model element during transformation.

**Transformation language properties**

The size of graph based specification language of Viatra increases for the second case study. It is 180 and classified as high according to the specified boundaries in Section 4.3. The number of syntactic complexity for this approach is 142. The call graph of specification is presented in Figure 4.11. Each node represents a rule and edges represent the calls from one rule to another, both implicit and explicit calls. The total number of calls are 16 and there is no recursive call in the specification. The Viatra specification is factorized and transformation is divided into two modules. The percentage of cohesion is 42%.
4.15 Comparison of approaches

The relative capabilities of each transformation approach is evaluated according to the framework in Chapter 3. The second “class diagrams to relational database” case study is compared according to the published specification of the transformation approaches. This section presents the comparisons of each approach with respect to the quantitative metrics of size, complexity, execution time, modularity, development effort and correctness. In addition, we summarise the results of the usability survey.

Size

An obvious size metric is the number of lines of code in the specification. When comparing the specification sizes that is measured by the lines of code, it should be
4.15. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th>Approach</th>
<th>LOC in first case study</th>
<th>Summary</th>
<th>LOC in second case study</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>QVT-R</td>
<td>20</td>
<td>High</td>
<td>180</td>
<td>High</td>
</tr>
<tr>
<td>Kermeta</td>
<td>16</td>
<td>Medium</td>
<td>174</td>
<td>High</td>
</tr>
<tr>
<td>ATL</td>
<td>19</td>
<td>Medium</td>
<td>86</td>
<td>Medium</td>
</tr>
<tr>
<td>UML-RSDS</td>
<td>3</td>
<td>Low</td>
<td>35</td>
<td>Low</td>
</tr>
<tr>
<td>Viatra</td>
<td>16</td>
<td>Medium</td>
<td>180</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 4.24: Size metrics of case studies

considered that the Viatra and Kermeta specifications both include model management codes (as in the main method in Kermeta) that should be counted separately from the core transformation rules. Table 4.24 shows the size metrics for the two case studies. It indicates a big difference between total number of lines of code in UML-RSDS than other approaches. This is due to the particular style of specification in this language. Kermeta presents the better result than ATL for the first case study, however the low level paradigm for defining a transformation in Kermeta, results to have a very large size implementation for the second case study. The QVT-R transformation approach is not a suitable transformation in terms of size for both case studies. Moreover, the size of hybrid transformation language of Viatra is inappropriate for the second case study.

According to size categories of table 4.2, ATL, Kermeta and Viatra generate moderate size metrics for the first case study. The size metrics for the second case study is high for Kermeta, Viatra and QVT-R.

Complexity

The complexity of a specification can be measured by analysing its call graph. A call graph is a directed graph that represents calling relationships between subroutines in a program. In the call graph, each node represents functions or rules while edges represent calls between nodes. The size of the call graph affects the complexity of the program. The greater the number of arcs in the call graph, the higher is the dependency between different parts of the program, and so the greater is the complexity. Recursive calls have a substantial effect on complexity. Depending on the input model
we can go through a recursion several times, which makes the transformation difficult to understand and verify. In this thesis we only measure if there is recursion in the specification or not.

Each node represents a rule and edges represent the calls from one rule to another, both implicit and explicit calls.

Table 4.25 compares the complexities of the approaches, based on the total number of calls and depth of calls for the second case study. Depth1 is the maximum depth of call chains not involving recursive loops, and Depth2 is the maximum length of a recursive loop. This shows considerable differences in the styles of specification adopted, with recursion being used substantially in some solutions such as Kermita and QVT, and not used at all in the other solutions.

Another form of complexity is due to the complexity of the expressions used within the specification: the greater the number of expression operators or references to metamodel features\(^4\), the more effort is required to comprehend and work with the specification.

Tables 4.26 and 4.27 compare the complexities of the approach, based on the syntactic complexities, the total number of calls and Depth1 of calls for the case studies. The result of comparison for the first case study shows that the Kermita approach generates moderate specification in terms of complexity; However, its complexity is high for the second case study. The UML-RSDS approach presented the best result.

---

\(^4\)operators such as, =, ;, \(\rightarrow\text{exists}()\)...etc. and features such as name, parent...etc.
in terms of complexity for the first case study, but its high complexity for the second case study is due to the complexity of its expressions in the specification. The rule bases specification language of ATL presented the best result in terms of complexity for the second case study. Both factors of syntactic complexity and structural complexity increase for Viatra and Kermeta in the larger transformation. This is due to the imperative specification style in Kermeta and unusual logic programming paradigm of Viatra.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Syntactic Complexity</th>
<th>Total number of calls</th>
<th>Total number of recursive calls</th>
<th>Depth 1</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>QVT-R</td>
<td>30</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>Medium</td>
</tr>
<tr>
<td>Kermeta</td>
<td>30</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>Medium</td>
</tr>
<tr>
<td>ATL</td>
<td>39</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>High</td>
</tr>
<tr>
<td>UML-RSDS</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Low</td>
</tr>
<tr>
<td>Viatra</td>
<td>36</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 4.26: Complexity comparison for the first case study

<table>
<thead>
<tr>
<th>Approach</th>
<th>Syntactic Complexity</th>
<th>Total number of calls</th>
<th>Total number of recursive calls</th>
<th>Depth 1</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>QVT-R</td>
<td>163</td>
<td>10</td>
<td>4</td>
<td>4</td>
<td>Medium</td>
</tr>
<tr>
<td>Kermeta</td>
<td>188</td>
<td>22</td>
<td>9</td>
<td>4</td>
<td>High</td>
</tr>
<tr>
<td>ATL</td>
<td>146</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>Low</td>
</tr>
<tr>
<td>UML-RSDS</td>
<td>214</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>High</td>
</tr>
<tr>
<td>Viatra</td>
<td>142</td>
<td>16</td>
<td>0</td>
<td>3</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 4.27: Complexity comparison for the second case study

**Execution Time**

The execution time of the approaches for the first case study is measured by comparing their execution time on the input with hundred elements from the first sets
4.15. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th>Approach</th>
<th>Execution time (ms)</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>QVT-R</td>
<td>297</td>
<td>High</td>
</tr>
<tr>
<td>Kermeta</td>
<td>31</td>
<td>Medium</td>
</tr>
<tr>
<td>ATL</td>
<td>15</td>
<td>Low</td>
</tr>
<tr>
<td>UML-RSDS</td>
<td>0</td>
<td>Low</td>
</tr>
</tbody>
</table>

Table 4.28: Execution time comparison for the first case study

of testcases. Table 4.28 compares the execution time of the approaches. The result shows that UML-RSDS performs this task in less than 1 ms. In addition, ATL executes the transformation in short amount of time for the first case study. In contrast, the execution time of transformation for QVT-R is high.

Stress testing is carried out for the first case study to measure the maximum capability of a transformation tool, in terms of maximum size of models in horizontal and vertical level. For the approaches such as ATL and QVT-R that use an interpreter, the amount of time taken to produce the executable transformation will increase more dramatically than the approaches where execution occurs through a machine code such as UML-RSDS. The UML-RSDS tool has a linear execution time in flat tree and vertical tree.

Modularity

Modularity is considered both in terms of how factorised the specification is, and how the proportion of calls within modules (cohesion) relates to the proportion of calls between modules (coupling). In the first case study all of the approach generates specifications with high or medium modularity features.

Table 4.30 shows the encapsulation of related rules inside individual modules for the second case study. ATL and UML-RSDS are considered as a single integrated module but Kermeta, Viatra and QVT-R are decomposed into smaller units. The results of evaluation for modularity factor is summarised in Table 4.29

In UML-RSDS there are repeated occurrence of duplicated expressions such as type.name and c.kind = 'persistent', but their size is less than minimum size \(^5\). The Kermeta solution for this case study is almost factorised. The factorisation of QVT-

\(^5\)seven in the second case study
4.15. COMPARISON OF APPROACHES

R and Viatra are 100% as there is no repeated expression in the transformation specification for this case study.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Factorisation</th>
<th>Coupling</th>
<th>Cohesion</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>QVT-R</td>
<td>100%</td>
<td>10%</td>
<td>90%</td>
<td>High</td>
</tr>
<tr>
<td>Kermeta</td>
<td>85%</td>
<td>14%</td>
<td>86%</td>
<td>Medium</td>
</tr>
<tr>
<td>ATL</td>
<td>50%</td>
<td>0%</td>
<td>100%</td>
<td>Medium</td>
</tr>
<tr>
<td>UML-RSDS</td>
<td>100%</td>
<td>0%</td>
<td>100%</td>
<td>High</td>
</tr>
<tr>
<td>Viatra</td>
<td>100%</td>
<td>57%</td>
<td>43%</td>
<td>Low</td>
</tr>
</tbody>
</table>

Table 4.29: Modularity measures for class to relational database

The notations all support a concept of module, consisting of collections of closely-related rules. The overall ranking of approaches with respect to modularity agrees with the ordering for size, complexity and correctness in UML-RSDS, Kermeta and ATL approaches for the first case study.

**Development Effort**

Development time refers to the time spent on the implementation of the case study on each transformation language. The time for understanding the problem or specification notation is not considered here. Table 4.31 shows the time spent on development of the first case study. The ranking is from 1 (least complexity and development time) to 5 (most). The developer in this case has had no prior experience with any of the new approaches or tools; however it has had some Java, UML and OCL expertise. Due to the use of published specification for the second case study the development effort is not considered for its evaluation.

It can be seen that the development time of Kermeta is lower than other approach even without an expert assistance. This is because the object-oriented programming paradigm in this approach requires the least work to construct and analyse the specification. In contrast, developing a simple “tree to graph” transformation takes a substantial amount of time in QVT-R and Viatra languages. This is because of the particular specification procedure in these languages.
4.15. COMPARISON OF APPROACHES

Table 4.30: Modularisation of Call graphs for UML to Relational database case study

Correctness

We assess the correctness by three separate 5-point measures of syntactic correctness, termination and confluence. Table 4.32 shows the results for each of these, considered as numeric values from 0 (None) to 4 (Comprehensive), and the resulting overall average score.

The key finding that can be extracted from the table is the high ranking of correctness in UML-RSDS. Syntactic correctness of individual rules in UML-RSDS can be proved by using an automated translation from UML-RSDS to the B formal notation,
4.15. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th>Approach</th>
<th>Development time (min)</th>
<th>Ranking</th>
<th>Summary</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>QVT-R</td>
<td>65</td>
<td>4</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Kermeta</td>
<td>30</td>
<td>1</td>
<td>Low</td>
<td></td>
</tr>
<tr>
<td>ATL</td>
<td>40</td>
<td>2</td>
<td>Medium</td>
<td>Expert assistance was available</td>
</tr>
<tr>
<td>UML-RSDS</td>
<td>45</td>
<td>3</td>
<td>Medium</td>
<td>Expert assistance was available</td>
</tr>
<tr>
<td>Viatra</td>
<td>75</td>
<td>5</td>
<td>High</td>
<td>Unfamiliar paradigm</td>
</tr>
</tbody>
</table>

Table 4.31: Development effort comparison for the tree to graph transformation

<table>
<thead>
<tr>
<th>Approach</th>
<th>Syntactic</th>
<th>Termination</th>
<th>Confluence</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>QVT-R</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>Low</td>
</tr>
<tr>
<td>Kermeta</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>Medium</td>
</tr>
<tr>
<td>ATL</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>Low</td>
</tr>
<tr>
<td>UML-RSDS</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>High</td>
</tr>
<tr>
<td>Viatra</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>Medium</td>
</tr>
</tbody>
</table>

Table 4.32: Correctness measures

and applying proof within B. Confluence is provided and alongside, the termination is high because of the bounded loops. In Kermeta the precise control over the ordering of rule applications makes the confluence clear. The termination of the solution in ATL is clear because of the linear iterations of the rules over their source domains. There is explicit scheduling of different rules, for Viatra, which makes its confluence better than QVT-R and ATL.

**Usability factors**

Understandability, learnability and attractiveness of the approaches that restricted here to these aspects as they concern the specification language, rather than the tool were evaluated by a survey of informatics professionals ranging from PhD students to senior academics. In addition to the questions we include a small test case to assess the detailed understanding of the participant: the participant needs to explain where in the code a particular aspect of the transformation is dealt with. This is a factor for understandability. The difference between this score for detailed understanding and the level of initial knowledge is also taken as a learnability factor. Table 4.33 sum-

---

6Generating edge between nodes
4.15. COMPARISON OF APPROACHES

marises the results of the survey for “tree to graph” transformation case study. The answers were on a five-point scale, with value 4 meaning very high (understandability, learnability, attractiveness) and 0 meaning none. For each question the average value of the response across the different respondents is taken.

<table>
<thead>
<tr>
<th>Question</th>
<th>UML-RSDS</th>
<th>QVT</th>
<th>Kermeta</th>
<th>ATL</th>
<th>Viatra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expertise level</td>
<td>0.75</td>
<td>0</td>
<td>0.5</td>
<td>1.25</td>
<td>0</td>
</tr>
<tr>
<td>Relate formal to informal (U)</td>
<td>0.75</td>
<td>2</td>
<td>3.75</td>
<td>1.25</td>
<td>2</td>
</tr>
<tr>
<td>Understanding detail (U)</td>
<td>1.6</td>
<td>1</td>
<td>3.2</td>
<td>2</td>
<td>2.4</td>
</tr>
<tr>
<td>Well structured (A)</td>
<td>1.25</td>
<td>1.5</td>
<td>0.75</td>
<td>1.75</td>
<td>1.25</td>
</tr>
<tr>
<td>Attractive (A)</td>
<td>1.75</td>
<td>2.5</td>
<td>1.75</td>
<td>2.5</td>
<td>0.75</td>
</tr>
<tr>
<td>Understanding effort (L)</td>
<td>0.75</td>
<td>2</td>
<td>3.5</td>
<td>1.25</td>
<td>2</td>
</tr>
<tr>
<td>Gained knowledge (L)</td>
<td>0.85</td>
<td>1</td>
<td>2.7</td>
<td>0.75</td>
<td>2.4</td>
</tr>
</tbody>
</table>

Table 4.33: Summary of usability survey question responses for the first case study

Table 4.34 sums the factors for each usability characteristic and gives the qualitative summary for each approach. The result of survey shows that Kermeta is considered as the most understandable and learnable approach among others. This is because of the familiarity of the users with object oriented programming style. In contrast, the UML-RSDS is ranked as less understandable, learnable and attractive approach for the users’. The overall ranking of approaches with regard to usability is therefore: Kermeta (7.825); Viatra (5.4); QVT (5); ATL (4.75); UML-RSDS (3.475). This ordering agrees with the ordering of development time for the first case study.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Understandability</th>
<th>Attractiveness</th>
<th>Learnability</th>
</tr>
</thead>
<tbody>
<tr>
<td>QVT-R</td>
<td>1.5 (Low)</td>
<td>2 (Medium)</td>
<td>1.5 (Low)</td>
</tr>
<tr>
<td>Kermeta</td>
<td>3.475 (High)</td>
<td>1.25 (Low)</td>
<td>3.1 (High)</td>
</tr>
<tr>
<td>ATL</td>
<td>1.625 (Low)</td>
<td>2.125 (Medium)</td>
<td>1 (Low)</td>
</tr>
<tr>
<td>UML-RSDS</td>
<td>1.175 (Low)</td>
<td>1.5 (Low)</td>
<td>0.8 (Low)</td>
</tr>
<tr>
<td>Viatra</td>
<td>2.2 (Medium)</td>
<td>1 (Low)</td>
<td>2.2 (Medium)</td>
</tr>
</tbody>
</table>

Table 4.34: Usability summaries for the first case study
4.15. COMPARISON OF APPROACHES

4.15.1 Evaluation summaries

For each approach, we can complete a table of its quality characteristics based upon the measured values for its attributes. The different attribute measures could be combined using some scheme of numeric weighting, and likewise for the combination of subcharacteristics which contribute to a quality characteristic. Here, we will use a simple count of the attributed values which fall into the top two categories of the five-point scale from the viewpoint of the subcharacteristics. This allows decision-makers to gain a quick overview of the merits of a particular approach, in the event of a close contest between two approaches. This way the detailed measurement values for the approaches can be compared.

Table 4.35 summarises the characteristic values for QVT-R. The first sets of ticks are used to show the evaluation of the first case study and the second for the evaluation of the second one. Development effort is not considered for the evaluation of the second case as the published specification of this case study is used. This approach scores highly for \textit{Interoperability}, \textit{Reliability}, poorly for \textit{Usability}, \textit{Efficiency}, \textit{Portability} and has mixed result for others characteristics. The QVT-R tool needs to provide a methodology to process the transformation in shorter amount of time and increases its capacity for handling large input models.

Table 4.36 illustrates the summary characteristic values for Kermeta. The Kermeta approach scores highly for \textit{Accuracy}, \textit{Interoperability}, \textit{Reliability}, \textit{Efficiency} and \textit{Portability}, and has mixed result for \textit{Maintainability} and \textit{Usability}. The low level paradigm for defining a transformation in Kermeta, results to have a very large size implementation in substantial case studies.

Table 4.37 shows the summary table for ATL. The ATL approach scores highly for \textit{Interoperability}, \textit{Reliability}, \textit{Efficiency} and \textit{Portability}, and has mixed result for other characteristics. The rule based specification of ATL is a convenient paradigm for specifying a transformation.

Table 4.38 presents the summary characteristic values for UML-RSDS. As can be seen, UML-RSDS approach scores highly for \textit{Accuracy}, \textit{Efficiency}, \textit{Portability} and poorly for \textit{Reliability}, \textit{Usability}, and has mixed result for other characteristics. The UML-RSDS tool needs to improve its strategy for handling errors within models and within processing. Although UML-RSDS is based on the standard UML and OCL.
### 4.15. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Subcharacteristic</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Suitability</td>
<td>Abstraction level ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Size × ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Effectiveness ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Development effort × −</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Execution time × ×</td>
</tr>
<tr>
<td>Accuracy</td>
<td>Correctness × ×</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Completeness ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>Interoperability</td>
<td>Embeddable in transformation process × ×</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Close to well-known notation ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Eclipse plug-ins ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>Functionality</td>
<td>Close to well-known notation ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>compliance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>Maturity</td>
<td>History of use ✓ ✓</td>
</tr>
<tr>
<td></td>
<td>Fault tolerance</td>
<td>Tolerance of false assumptions ✓ ✓</td>
</tr>
<tr>
<td>Usability</td>
<td>Understandability</td>
<td>× −</td>
</tr>
<tr>
<td></td>
<td>Learnability</td>
<td>× −</td>
</tr>
<tr>
<td></td>
<td>Attractiveness</td>
<td>✓ −</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Time behaviour</td>
<td>Execution time × ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum capability × ×</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Changeability</td>
<td>Size × ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modularity ✓ ✓</td>
</tr>
<tr>
<td>Portability</td>
<td>Adaptability</td>
<td>Extensibility × ×</td>
</tr>
</tbody>
</table>

Table 4.35: Quality characteristics for QVT-R. The first sets of ticks are used to show the evaluation of the first case study and the second for the evaluation of the second one.

notations, the style of specification in this language is not attractive for the users’.

Finally, Table 4.40 expresses the summary characteristic values for Viatra. It scores highly for *Accuracy* and *Reliability*, poorly for *Portability* and has mixed result for the other characteristics. The Viatra transformation approach is not based upon MOF and OCL and uses a different metamodelling formalism. It needs a large amount of effort to specify a transformation in a hybrid language of Viatra.

Tables 4.40 and 4.41 give the overall ranking in terms of the ISO/IEC 9126 characteristics. These tables present the number of good characteristics of each approach. If some factors are more investigated for a particular purpose, it will count more
## 4.15. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Subcharacteristic</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Suitability</td>
<td>Abstraction level × ×&lt;br&gt;Size √ ×&lt;br&gt;Complexity √ ×&lt;br&gt;Effectiveness √ √&lt;br&gt;Development effort √ −&lt;br&gt;Execution time √ √</td>
</tr>
<tr>
<td>Accuracy</td>
<td></td>
<td>Correctness √ √&lt;br&gt;Completeness √ √</td>
</tr>
<tr>
<td>Interoperability</td>
<td></td>
<td>Embeddable in transformation process √ √&lt;br&gt;Close to well-known notation √ √&lt;br&gt;Eclipse plug-ins √ √</td>
</tr>
<tr>
<td>Functionality</td>
<td>Maturity</td>
<td>Close to well-known notation √ √</td>
</tr>
<tr>
<td></td>
<td>Fault tolerance</td>
<td>Tolerance of false assumptions √ √</td>
</tr>
<tr>
<td>Usability</td>
<td>Understandability</td>
<td>√ −</td>
</tr>
<tr>
<td></td>
<td>Learnability</td>
<td>√ −</td>
</tr>
<tr>
<td></td>
<td>Attractiveness</td>
<td>× −</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Time behaviour</td>
<td>Execution time √ √&lt;br&gt;Maximum capability √ √</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Changeability</td>
<td>Size √ ×&lt;br&gt;Complexity √ ×&lt;br&gt;Modularity √ √</td>
</tr>
<tr>
<td>Portability</td>
<td>Adaptability</td>
<td>Extensibility √ √</td>
</tr>
</tbody>
</table>

Table 4.36: Quality characteristics for Kermeta. The first sets of ticks are used to show the evaluation of the first case study and the second for the evaluation of the second one.

In the overall ranking table. Then the total score presents the general view about each approach. For measures like complexity, size development time and execution time, a high value means a Low score. For the rest of the attributes the high value means a High score. I also counted the Medium score as a good representative for characteristic of approaches.

This table shows that Kermeta and ATL can be considered as a good candidates for carrying out these two types of transformations on the basis of their overall rankings. The rankings are specific to these types of refinement and re-expression problems. Developers effort was lowest for implementing the first case study in Ker-
### 4.15. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Subcharacteristic</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Suitability</td>
<td>Abstraction level √√</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Size √√</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity × √</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Effectiveness √ √</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Development effort √ −</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Execution time √ √</td>
</tr>
<tr>
<td>Accuracy</td>
<td>Correctness × ×</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Completeness √ √</td>
<td></td>
</tr>
<tr>
<td>Interoperability</td>
<td>Embeddable in transformation process √ √</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Close to well-known notation √ √</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Eclipse plug-ins √ √</td>
<td></td>
</tr>
<tr>
<td>Functionality</td>
<td>Close to well-known notation √ √</td>
<td></td>
</tr>
<tr>
<td>compliance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>Maturity</td>
<td>History of use √ √</td>
</tr>
<tr>
<td>Fault tolerance</td>
<td>Tolerance of false assumptions √ √</td>
<td></td>
</tr>
<tr>
<td>Usability</td>
<td>Understandability × −</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Learnability × −</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Attractiveness √ −</td>
<td></td>
</tr>
<tr>
<td>Efficiency</td>
<td>Time behaviour</td>
<td>Execution time √ √</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum capability √ √</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Changeability</td>
<td>Size √ √</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity × √</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modularity √ √</td>
</tr>
<tr>
<td>Portability</td>
<td>Adaptability</td>
<td>Extensibility √ √</td>
</tr>
</tbody>
</table>

Table 4.37: Quality characteristics for ATL. The first sets of ticks are used to show the evaluation of the first case study and the second for the evaluation of the second one.

meta. This is due to the familiarity of the object-oriented programming paradigm and requires the least work to construct and analyse the specification. Clearly, explicit control over the rule execution in Kermeta seems preferable for usability, correctness and analysis. However, the imperative style of Kermeta causes a complex specification for the larger case studies. ATL has an appropriate rule based paradigm for specifying a transformation. This approach presents a less complex result for “class to relational database” case study than the other investigated approaches. In addition, the use of a standard notation such as OCL in the specification supports the interoperability of the ATL tool. The UML-RSDS approach presents strong result in
### 4.15. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Subcharacteristic</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Suitability</td>
<td>Abstraction level ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Size ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity ✓ ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Effectiveness ✓ ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Development effort ✓ –</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Execution time ✓ ✓</td>
</tr>
<tr>
<td>Accuracy</td>
<td>Correctness ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Completeness ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>Interoperability</td>
<td>Embeddable in transformation process ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Close to well-known notation ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Eclipse plug-ins × ×</td>
<td></td>
</tr>
<tr>
<td>Functionality</td>
<td>Compliance</td>
<td>Close to well-known notation ✓ ✓</td>
</tr>
<tr>
<td>Reliability</td>
<td>Maturity History of use × ×</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fault tolerance Tolerance of false assumptions × ×</td>
<td></td>
</tr>
<tr>
<td>Usability</td>
<td>Understandability ¥ –</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Learnability ¥ –</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Attractiveness ¥ –</td>
<td></td>
</tr>
<tr>
<td>Efficiency</td>
<td>Time behaviour Execution time ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Maximum capability ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>Maintainability</td>
<td>Changeability Size ✓ ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Complexity ✓ ×</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Modularity ✓ ✓</td>
<td></td>
</tr>
<tr>
<td>Portability</td>
<td>Adaptability Extensibility ✓ ✓</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.38: Quality characteristics for UML-RSDS. The first sets of ticks are used to show the evaluation of the first case study and the second for the evaluation of the second one.

terms of Functionality. However due to its particular style of the specification and low level result in Reliability, this tool is less appropriate than ATL and Kermeta. QVT-R and Viatra are placed lower than other approaches in the overall ranking table. Even developing a simple “tree to graph” transformation takes a substantial time in these languages.

In the tree to graph re-expression transformation it is essential to keep trace of transformation to identify which elements of source model are related to the element in the target model via specific rules. The solutions have addressed these problems in different ways. Trace links are automatically created and managed in QVT-R and
4.15. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Subcharacteristic</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Suitability</td>
<td>Abstraction level √ √</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Size √ ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity × ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Effectiveness √ √</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Development effort × –</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Execution time – –</td>
</tr>
<tr>
<td>Accuracy</td>
<td>Correctness</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td>Completeness</td>
<td>√</td>
</tr>
<tr>
<td>Interoperability</td>
<td>Embeddable in transformation process × ×</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Close to well-known notation × ×</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Eclipse plug-ins</td>
<td>√</td>
</tr>
<tr>
<td>Reliability</td>
<td>Maturity</td>
<td>History of use √ √</td>
</tr>
<tr>
<td></td>
<td>Fault tolerance</td>
<td>Tolerance of false assumptions √ √</td>
</tr>
<tr>
<td>Usability</td>
<td>Understandability</td>
<td>√ –</td>
</tr>
<tr>
<td></td>
<td>Learnability</td>
<td>√ –</td>
</tr>
<tr>
<td></td>
<td>Attractiveness</td>
<td>× –</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Time behaviour</td>
<td>Execution time – –</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum capability – –</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Changeability</td>
<td>Size √ ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity × ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modularity √ ×</td>
</tr>
<tr>
<td>Portability</td>
<td>Adaptability</td>
<td>Extensibility × ×</td>
</tr>
</tbody>
</table>

Table 4.39: Quality characteristics for Viatra. The first sets of ticks are used to show the evaluation of the first case study and the second for the evaluation of the second one.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Funct. (12)</th>
<th>Rel. (2)</th>
<th>Usab. (3)</th>
<th>Effic. (2)</th>
<th>Maint. (3)</th>
<th>Port. (1)</th>
<th>Total (23)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kermeta</td>
<td>11</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td>ATL</td>
<td>10</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>UML-RSDS</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>QVT-R</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>Viatra</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 4.40: Overall ranking of approaches for the first case study

ATL and do not need to be defined by developer. However, the trace in Kermeta and Viatra are implemented explicitly by the developers. UML-RSDS relies on the
use of primary keys to link newly-created superordinate elements of the target model to their existing components elements. An efficient means to lookup of elements by primary key is built-in to the generated Java code of transformations.

4.16 Conclusion

In this chapter, five transformation approaches are evaluated on two samples of refinement and re-expression problems using the evaluation framework in Chapter 3. The significant differences between the model transformation approaches concern their specification paradigm and level of abstraction. ATL is a hybrid transformation language, QVT-R is highly declarative, Viatra and UML-RSDS are hybrid, while Kermeta is implementation-oriented. The results of overall rankings in terms of the ISO/IEC 9126 characteristics suggest that Kermeta and ATL could be considered as a good candidates for carrying out these two types of transformations. The explicit programming paradigm of Kermeta is the easiest for novice model transformation developers to apply, particularly on simple re-expression and refinement transformations. ATL has an appropriate rule based paradigm for specifying a transformation. The result of evaluation showed that ATL generated an effective solution to the large case studies in terms of size and complexity. The UML-RSDS approach also presented appropriate result for both case studies, but the particular specification style in this language and the lack of interoperability with Eclipse make it less usable than Kermeta and ATL. It took a substantial amount of time to implement the simple “tree to graph” transformation in QVT-R language. In addition, the unusual logic-programming paradigm of Viatra increases developer’s effort. Viatra provides specific import and export facilities, while ATL, QVT-R and Kermeta can be ex-
ecuted within the Eclipse tool, supporting, in principle, interoperation with other Eclipse-hosted tools.

UML-RSDS provides syntactic correctness by using automated translation to the B formal notation. The termination for UML-RSDS is high because the loops are bounded in each individual rule. Kermeta establishes indirect syntactic correctness by defining assertions and invariants. Other semantic checks, such as termination and confluence would also be beneficial for developers. In Kermeta the order of rule applications is explicit and termination is direct for iterative algorithms. ATL provides a check that no two rules are both enabled simultaneously on the same model; however this does not ensure confluence in all cases. There is explicit scheduling of different rules for Viatra, which makes its confluence better than QVT-R and ATL.

Compiled approaches like UML-RSDS have low execution time and high capacity through machine code execution. Rule-based specification can be reconciled with the need to demonstrate correctness, through limiting the complexity of the calling relation between rules such as avoiding recursion, implicit rule invocation and excessive call depth. These restrictions also enhance the general analysability, modularity and extensibility of a specification.
Model migration is a challenging topic in Model Driven Engineering (MDE) in which, instance models are updated to re-establish conformance in response to metamodel evolution [129]. To compare the suitability of transformation tools for performing migration task, UML-RSDS in addition to eight other transformation tools are applied to the model migration case study. The case study was presented at Transformation Tools Contest (TTC) workshop in 2010 [124]. The case study includes migration of UML activity diagrams from version 1.4 to UML 2.2. In this chapter we apply the UML-RSDS transformation tool to tackle this problem and then compare with the Epsilon Flock [126] and the GrGen.NET [24] graph transformation tool from submitted solutions in the workshop. This chapter is based on a paper that has been published in Transformation Tool Contest Workshop in 2010 [84].

First, Section 5.1 presents the transformation problem for the case study. Next, Section 5.2 defines the criteria for evaluation of the case study. Following that, test case is shown in Section 5.3. Sections 5.4, 5.5 and 5.6, describe the individual solutions to the migration problem. Moreover, Section 5.7 compares the different solutions on the relative values of their characteristics. A conclusion in Section 5.8, explores the findings of this comparison and concludes the chapter.
5.1 Third Case Study: Migration of the UML Activity Diagrams from version 1.4 to UML 2.2

The task described by [124] involves the transformation of models of the UML 1.4 activity diagram language [112] into models of the UML 2.2 [114]. In UML 1.4 the language of activity diagrams is a variant of the state machine language. However in UML 2.2 the activity diagram is reformulate as kind of Petrinet, such as places and transitions. The structure of these two languages are quite similar, so the transformation can be specified in a direct manner based on the structure of the source language. The metamodel of UML 1.4 activity diagram and UML 2.2 are shown in Figures 5.1 and 5.2. Moreover, Table 5.1 gives the informal correspondence of the source and target languages. It can be seen that the transformation is mainly a direct mapping from source language metaclasses and features to corresponding target

---

Figure 5.1: UML 1.4 Activity Graphs based on [112]
5.1. THIRD CASE STUDY: MIGRATION OF THE UML ACTIVITY DIAGRAMS FROM VERSION 1.4 TO UML 2.2

Figure 5.2: UML 2.2 Activity Diagrams based on [114]

language metaclasses and features. For instance, ActivityGraph in UML 1.4 need to be transformed into Activity in UML 2.2. This transformation consists of mapping transitions of ActivityGraph to edge of Activity, partitions of ActivityGraph to group of Activity and finally top.subvertex of ActivityGraph to node of Activity. The only parts requiring significant logic in the transformation are (i) the mapping of different kinds of pseudostates to different kinds of activity nodes, and (ii) the mapping of transitions to control or object flows, depending on what state vertices they connect, and of transitions with triggers to control nodes which receive the trigger event or signal.

The transformation can be formalised by a set of constraints, which define how the source and target models are related. For example, the correspondence of FinalStates and ActivityFinalNodes could be defined by a constraint C1:

\[ f \text{ : FinalState} \Rightarrow \exists n : \text{ActivityFinalNode} \cdot n\text{.name} = f\text{.name} \]
### Table 5.1: Mapping from UML 1.4 activity diagrams to UML 2.2

<table>
<thead>
<tr>
<th>UML 1.4 metaclass/feature</th>
<th>UML 2.2 metaclass/feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>StateVertex</td>
<td>ActivityNode</td>
</tr>
<tr>
<td>FinalState</td>
<td>ActivityFinalNode</td>
</tr>
<tr>
<td>ActionState</td>
<td>OpaqueAction</td>
</tr>
<tr>
<td>ObjectFlowState</td>
<td>DataStoreNode</td>
</tr>
<tr>
<td>CompositeState</td>
<td>StructuredActivityNode</td>
</tr>
<tr>
<td>subvertex of CompositeState</td>
<td>node of StructuredActivityNode</td>
</tr>
<tr>
<td>Pseudostate (kind = initial)</td>
<td>InitialNode</td>
</tr>
<tr>
<td>Pseudostate (kind = join)</td>
<td>JoinNode</td>
</tr>
<tr>
<td>Pseudostate (kind = fork)</td>
<td>ForkNode</td>
</tr>
<tr>
<td>Pseudostate (kind = junction and one incoming and ≥ 2 outgoing transitions)</td>
<td>DecisionNode</td>
</tr>
<tr>
<td>Pseudostate (kind = junction and one outgoing and ≥ 2 incoming transitions)</td>
<td>MergeNode</td>
</tr>
<tr>
<td>SimpleState, outgoing transition has trigger</td>
<td>AcceptEventAction</td>
</tr>
<tr>
<td>Transition (one end is an ObjectFlowState)</td>
<td>ObjectFlow</td>
</tr>
<tr>
<td>Transition (neither end is an ObjectFlowState)</td>
<td>ControlFlow</td>
</tr>
<tr>
<td>source, target of Transition guard.expression of Transition</td>
<td>source, target of ActivityEdge guard (in OpaqueExpression) of ActivityEdge</td>
</tr>
<tr>
<td>Partition</td>
<td>ActivityPartition</td>
</tr>
<tr>
<td>contents of Partition</td>
<td>containedNode of ActivityPartition</td>
</tr>
<tr>
<td>ActivityGraph</td>
<td>Activity</td>
</tr>
<tr>
<td>transitions of ActivityGraph</td>
<td>edge of Activity</td>
</tr>
<tr>
<td>partition of ActivityGraph</td>
<td>group of Activity</td>
</tr>
<tr>
<td>top.subvertex of ActivityGraph</td>
<td>node of Activity</td>
</tr>
</tbody>
</table>

Likewise for the other source language metaclasses and features.

### 5.2 Evaluation Criteria

The evaluation criteria for migration case study is listed in Table 5.2. These criteria are divided into properties of the transformation language and properties of an implementation in a particular tool for that language.

Based on the actual distribution of values, specific boundary have been chosen

---

1. abstraction level, specification size, complexity, development effort, syntactic correctness, extensibility, close to well-known notation, understandability, modularity
2. effectiveness, execution time, termination, confluence, Eclipse plug-ins, maximum capability, fault tolerance, maturity
5.2. EVALUATION CRITERIA

<table>
<thead>
<tr>
<th>Transformation language</th>
<th>Abstraction level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size: LOC</td>
</tr>
<tr>
<td></td>
<td>Complexity:</td>
</tr>
<tr>
<td></td>
<td>(1) syntactic</td>
</tr>
<tr>
<td></td>
<td>complexity</td>
</tr>
<tr>
<td></td>
<td>(number of</td>
</tr>
<tr>
<td></td>
<td>operators,</td>
</tr>
<tr>
<td></td>
<td>number of</td>
</tr>
<tr>
<td></td>
<td>features)</td>
</tr>
<tr>
<td></td>
<td>(2) structural</td>
</tr>
<tr>
<td></td>
<td>complexity</td>
</tr>
<tr>
<td></td>
<td>(i) total number</td>
</tr>
<tr>
<td></td>
<td>calls</td>
</tr>
<tr>
<td></td>
<td>(ii) total</td>
</tr>
<tr>
<td></td>
<td>recursive calls</td>
</tr>
<tr>
<td></td>
<td>(iii) max call</td>
</tr>
<tr>
<td></td>
<td>depth</td>
</tr>
<tr>
<td></td>
<td>Modularity:</td>
</tr>
<tr>
<td></td>
<td>(1) factorisation</td>
</tr>
<tr>
<td></td>
<td>(2) cohesion</td>
</tr>
<tr>
<td></td>
<td>versus coupling</td>
</tr>
</tbody>
</table>

| Extensibility:          |
| extension to tree       |
| metamodel               |

| Interoperability:       |
| inclusion in a          |
| transformation process  |

| Development effort      |
| close to well-known     |
| notation               |

| Transformation tool     |
| Execution time          |
| provable syntactic      |
| correctness             |
| Correctness:            |
| termination, confluence |
| Effectiveness           |
| Fault tolerance         |
| Interoperability with   |
| Eclipse                |
| Maturity                |

Table 5.2: Evaluation criteria for solutions of third case study

to give a high level classification of measure for size, complexity, execution time, maximum capability, development effort, maturity, factorisation and cohesion. The rest of the properties are evaluated based on the classifications of Chapter 3.

In migration cases the transformation size is an important factor because, large part of the model remains unchanged during the transformation and it is appropriate to have automated mapping for unaffected elements. Size is classified as low, medium and high based on the size category of Table 5.3. Approaches which generate transformation with less than twenty-five lines of code categorized as low while, the ones with the specification size between twenty-six to fifty are considered as medium. In the table of categorization they divided in a way that at least one value exists in each category of high, medium and low. The high category refers to the language with more than fifty lines of code.
5.2. EVALUATION CRITERIA

<table>
<thead>
<tr>
<th>Value range</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..25 lines</td>
<td>Low</td>
</tr>
<tr>
<td>26..50 lines</td>
<td>Medium</td>
</tr>
<tr>
<td>51+ lines</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 5.3: Size categories for migration transformation

<table>
<thead>
<tr>
<th>Value range</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..100</td>
<td>Low</td>
</tr>
<tr>
<td>101..200</td>
<td>Medium</td>
</tr>
<tr>
<td>201+</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 5.4: Complexity categories for migration transformation

Table 5.4, shows the classification of complexity. The transformation approach is less complex if the total number of syntactic complexity and structure complexity is less than 100, medium between 101-200 and high for more than 200. In the table of categorization they divided in a way that at least one value exists in each category of high, medium and low.

In the description of TTC10 case study, three extensions have been defined. Extensibility in this case study considers to what extent the transformation tools can provide solutions to the extensions specified in Table 5.5. The first extension includes alternative migration semantics for ObjectFlowState. In the case study description, ObjectFlowState should be migrated to instances of ObjectNode. In addition, instances of Transition that have an ObjectFlowState as their source or target should be migrated to instances of ObjectFlow. However, the extension stated that instances of ObjectFlowState (and any connected Transitions) should be migrated to instances of ObjectFlow. The second extension is to explore the feasibility of migrating the concrete syntax of the activity diagram to the concrete syntax in their chosen UML 2 tool. Finally, the last extension expresses how to migrate a UML 1.4 model represented in XMI 1.x to a UML 2.1 model represented in XMI 2.x. The complete description of these three extensions can be found in Sections 3.1, 3.2 and 3.3 of the case study description [124]. The extensibility of transformation approach is high if it generates the three specified extensions. It is medium if it produce them partially and low if there is no implementation for the extensions.

The development time of transformation is low if the developer spent less than 2
5.3 Test Case

Figure 5.3 shows an activity diagram for filling orders (page 3-158 of [112]). This is an input model for testing the transformation approaches on this case study. The diagrams is partitioned into three swimlanes, representing different organisational units vertically. Activities are represented with rounded rectangles and Transitions with directed arrows. Fork and Join nodes are specified using a solid black rectangle. Decision nodes are represented with a diamond. Guards on transitions are specified using square brackets.

5.4 UML-RSDS

In this (2010) version of UML-RSDS the operations were placed in source metaclasses. This avoids the need to define additional transformation classes. However,
it reduces flexibility because it binds the form of the transformation closely to the particular source metamodel. In subsequent versions of UML-RSDS transformations are specified separately (as use cases).

In the metamodel migration case study, the operations for mapping states to activity nodes can be placed in each StateVertex subclass. For example the transformation of FinalState to ActivityFinalNode presents as the following operation in FinalState:

\[\text{toActivity}()\]

\[\text{post:}\]

\[\forall f : \text{FinalState} \cdot \exists n : \text{ActivityFinalNode} \cdot n.\text{name} = f.\text{name}\]

The notation \(\exists c : C \cdot P\) is a simplified syntax for the OCL formula

\[C.\text{allInstances}() \rightarrow \text{exists}(c \mid P)\]

### 5.4.1 Specification of migration task in UML-RSDS

Because of the similarity in structure between the source and target metamodels, the transformation can be specified by rules placed in the source language metaclasses,
one per metaclass. The rules then define how that entity should be mapped into the target model. This makes the transformation easy to understand and modify.

**Actions, FinalState and ObjectFlowState**

The mapping described in [124] maps ActionStates to OpaqueActions; however this loses information about the action performed in the state (the entry action of the state), so it may be preferable to map ActionState to Action; instead. UML 1.4 actions are effectively a subset of UML 2.2 actions, so this mapping would be valid.

For FinalState, ActionState and ObjectFlowState, we have the following constraints.

For FinalState:
\[ \forall f : \text{FinalState} \cdot \exists n : \text{ActivityFinalNode} \cdot n.\text{name} = f.\text{name} \]

For ActionState:
\[ \forall ac : \text{ActionState} \cdot \exists n : \text{OpaqueAction} \cdot n.\text{name} = ac.\text{name} \]

For ObjectFlowState:
\[ \forall o : \text{ObjectFlowState} \cdot \exists n : \text{DataStoreNode} \cdot n.\text{name} = o.\text{name} \& n.\text{type} = o.\text{type} \]

It is assumed that UML 1.4 types can be mapped without change into UML 2.2 types.

**Pseudostates**

We defined the following constraints For Pseudostate:

- \( \forall p : \text{Pseudostate} \cdot p.\text{kind} = \text{initial} \)
  \[ \Rightarrow \exists n : \text{InitialNode} \cdot n.\text{name} = p.\text{name} \& \]
- \( \forall p : \text{Pseudostate} \cdot p.\text{kind} = \text{join} \)
  \[ \Rightarrow \exists n : \text{JoinNode} \cdot n.\text{name} = p.\text{name} \& \]
- \( \forall p : \text{Pseudostate} \cdot p.\text{kind} = \text{fork} \)
  \[ \Rightarrow \exists n : \text{ForkNode} \cdot n.\text{name} = p.\text{name} \& \]
- \( \forall p : \text{Pseudostate} \cdot p.\text{kind} = \text{junction} \& p.\text{incoming.size} = 1 \)
  \[ \Rightarrow \exists n : \text{DecisionNode} \cdot n.\text{name} = p.\text{name} \& \]
- \( \forall p : \text{Pseudostate} \cdot p.\text{kind} = \text{junction} \& p.\text{incoming.size} > 1 \)
  \[ \Rightarrow \exists n : \text{MergeNode} \cdot n.\text{name} = p.\text{name} \)
A junction state is mapped to a decision node if it has one incoming transition, otherwise to a merge node.

**Simple and Composite states**

A simple state is assumed to be used in an activity diagram in order to wait for an event to occur. These states are mapped to `AcceptEventAction` instances:

\[
\forall s: SimpleState.\text{outgoing.size} = 1 \land s.\text{outgoing.trigger.size} = 1 \Rightarrow \\
\exists n: AcceptEventAction.\ n.\text{name} = s.\text{name} \land \\
\quad \quad \quad n.\text{trigger} = s.\text{outgoing.trigger}
\]

For `SimpleState`. For `CompositeState` we have the following operation:

```plaintext
toActivity() 
subvertex.toActivity()
```

In this case the composite state is mapped by mapping each of its contained states. We could also specify the creation of a `StructuredActivityNode` with the same name as the composite state, and with `node` set equal to the activity nodes mapped from `subvertex`.

**Guard and Transition**

Guards are mapped to opaque expressions:

\[
\forall g: Guard.\exists oe: OpaqueExpression.\\
\quad oe.\text{name} = g.\text{name} \land \\
\quad oe.\text{language} = g.\text{expression}.\text{language} \land \\
\quad oe.\text{body} = g.\text{expression}.\text{body}
\]

For `Guard`. Transitions are mapped to particular activity edges:

\[
(\forall t: Transition.\ t.\text{source} : ObjectFlowState \lor t.\text{target} : ObjectFlowState \Rightarrow \\
\exists f: ObjectFlow.\ f.\text{name} = t.\text{name} \land \\
\text{since action states cannot have triggers on their outgoing transitions, page 3-159 of [112]}
\]
A transition is mapped to an object flow if it has source or target in \textit{ObjectFlowState}, otherwise to a control flow. The not equal syntax of / is used in UML-RSDS. The expression \( t \text{.source} : \text{ObjectFlowState} \) means the source of transition is \textit{ObjectFlowState}, while the expression \( t \text{.source} / : \text{ObjectFlowState} \) means that the source of transition is not \textit{ObjectFlowState}.

The notation \textit{ActivityNode}[x] for a single object \( x \) refers to the activity node object with primary key (in this case name) value equal to \( x \), it is implemented in the Java code generated by the UML-RSDS tools by maintaining a map from the key values to nodes. In OCL it would be expressed as

\[
\text{ActivityNode.allInstances()} \rightarrow \text{select(name = x)} \rightarrow \text{any()}
\]

\section*{Partitions}

The partitions of the source model are mapped to activity partitions of the target model, with corresponding contents:

\[
\forall \ pa : \text{Partitions} \cdot \exists \ ap : \text{ActivityPartition} \cdot \ ap\text{.name} = \ pa\text{.name} & \\
\quad \ ap\text{.containedNode} \ = \ \text{ActivityNode[pa\text{.contents.name}]} & \\
\quad \ ap\text{.containedEdge} \ = \ \text{ActivityEdge[pa\text{.contents.name}]} 
\]

For \textit{Partition}.

\section*{Activities}

Finally, activity graphs are mapped to activities:

\[
\begin{align*}
\forall \ t : \text{Transition} \cdot \ t\text{.source} \!=\! \text{ObjectFlowState} \text{ or } t\text{.target} \!=\! \text{ObjectFlowState} \Rightarrow \\
\exists \ f : \text{ControlFlow} \cdot \ f\text{.name} = t\text{.name} & \\
\quad \ f\text{.source} \ = \ \text{ActivityNode[t\text{.source.name}]} & \\
\quad \ f\text{.target} \ = \ \text{ActivityNode[t\text{.target.name}]} & \\
\quad \ f\text{.guard} \ = \ \text{OpaqueExpression[t\text{.guard.name}]} 
\end{align*}
\]
∀ \( ac : ActivityGraph \) \( \exists \ a : Activity \cdot a.name = ac.name \&\)
\( a.node = ActivityNode[ac.top.subvertex.name] \&\)
\( a.group = ActivityPartition[ac.partition.name] \&\)
\( a.edge = ActivityEdge[ac.transitions.name] \)

**Overall algorithm**

The recommended specification style for model transformations (Specifying transformation rules within the metamodel) is to avoid dependencies between rules. In the older version of UML-RSDS, which has been used to tackle this case study the operations are added to the source metamodel avoiding the need to define additional transformation. The main part of transformation are defined within the metamodels. However, the overall algorithm is written as an activity of transformation metaclass. During the PhD period the UML-RSDS approach has been improved significantly. The current version of this approach specified the transformation by using a single use case and avoids the needs to specify the overall algorithm in an individual metaclass.

The overall algorithm is specified as the activity of the transformation metaclass \( SMtoActivity \). This is use as an explicit control expression to define rule application orders.

\[
\text{init()} ; \\
\text{CompositeState.toActivity()} ; \\
\text{Guard.toActivity()} ; \\
\text{Transition.toActivity()} ; \\
\text{Partition.toActivity()} ; \\
\text{ActivityGraph.toActivity()} \\
\]

Figure 5.4 shows a screen shot of the UML-RSDS system with the transformation source and target metamodels. In general, the mapping of subordinate parts of an element must be performed before the mapping of the element itself.

### 5.4.2 Evaluation properties for UML-RSDS

Table 5.6 summarises the evaluated attributes of the transformation language and tool for the UML-RSDS solution.
5.4. UML-RSDS

Transformation Language properties

The level of abstraction of the rule-based specification is high, and is close to the requirements. The recommended specification style for model transformations is to avoid dependencies between rules, and to use an explicit control expression to define rule application orders. This style relies on the use of primary keys for model elements, or other means of matching already-processed elements of the source model to their corresponding target model elements (e.g., by use of traces), hence newly-created superordinate elements of the target model are linked to their existing component elements. An efficient means for lookup of elements by primary key is built-in to the generated Java code of transformations. The transformation specification provides mapping from the elements of the source model to the target model, whether they are effected by migration task or not. This leads to a transformation with thirty-six lines of code.

There is no call in transformation specification. In addition, the total numbers of features are 140 and numbers of operators are 120. It is due to the explicit copying strategy in UML-RSDS, that results in high complexity. The factorization is 87% and
5.4. UML-RSDS

<table>
<thead>
<tr>
<th>Transformation tool</th>
<th>Abstraction level</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size: LOC</td>
<td>38 lines (Medium)</td>
<td></td>
</tr>
<tr>
<td>Syntactic complexity</td>
<td>120 operators, 140 features</td>
<td></td>
</tr>
<tr>
<td>Structural complexity</td>
<td>0 call</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 recursive calls, max call depth = 0</td>
<td></td>
</tr>
<tr>
<td>Complexity total</td>
<td>260 (High)</td>
<td></td>
</tr>
<tr>
<td>Modularity</td>
<td>factorisation: 87%</td>
<td></td>
</tr>
<tr>
<td>Modularity total</td>
<td>cohesion:100% , coupling: 0%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Medium</td>
<td></td>
</tr>
<tr>
<td>Extensibility</td>
<td>Low</td>
<td></td>
</tr>
<tr>
<td>Interoperability</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Development effort</td>
<td>6 hours (High)</td>
<td></td>
</tr>
<tr>
<td>Close to well-known notation</td>
<td>High</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transformation tool</th>
<th>Execution time</th>
<th>60 ms (Low)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntactic correctness</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Correctness (Termination)</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Correctness (Confluence)</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Effectiveness</td>
<td>Medium</td>
<td></td>
</tr>
<tr>
<td>Fault tolerance</td>
<td>Medium</td>
<td></td>
</tr>
<tr>
<td>Eclipse plug-ins</td>
<td>Low</td>
<td></td>
</tr>
<tr>
<td>Maturity</td>
<td>Low</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.6: Evaluation criteria of UML-RSDS solution for migration case study

The transformation is located inside one module. The total modularity is high as both factorisation and cohesion are high. Extension to the core tasks are not provided by UML-RSDS, but it is possible to do that while it needs more lines of code.

The development effort for implementing this task in UML-RSDS was high and around six hours. The method should be usable for software engineers familiar with UML and OCL, since no additional notation is used. However, the workshop participants considered UML-RSDS as the less understandable approach among others. UML-RSDS models can be stored in text or XMI formats. The generated Java code can be used independently of the UML-RSDS toolset (The input model for UML-RSDS in the migration case is shown in Appendix B).
Transformation Tool properties

The execution time of transformation is 60 ms for the model of activity diagram of filling orders specified in Section 5.3. The UML-RSDS tools check completeness of rules by checking that all data features of an object are set in any operation which creates it. For example, in the operation `toActivity()` on transitions, which creates an ActivityEdge, an error message would be given if there was no assignment to the target of the new edge. Procedures for checking semantic correctness exist but have not been automated. The syntactic correctness is provided by translation to the B formal method and proof tool [81][92]. Termination is provided for UML-RSDS solution, since all the processing is done by bounded loops. For each of the unordered iterations used in the transformation, it can be shown that the order of applications within the iteration does not affect the result. This results to have high confluence. UML-RSDS generates the correct target model for migration case study.

Fault tolerance for UML-RSDS is low, because the tool only gives Java error message and does not identify where the error is happened in terms of the specification. Furthermore, the maturity of tool is low as has been publicly available for less than three years. In addition, the tool has no interface to Eclipse; however it is considered as future extension. This case study is the first migration task that has transformed within this tool. Tracing of rule applications must be performed explicitly by the developer if required. The use of identity attributes is recommended as an alternative means of propagating changes from the source model to the target model. This is done by identifying which target model elements are semantically linked to which source model elements, independently of which model transformation rules were used to create the target model from the source. Implementation is by translation to Java. The generated code is close in structure to the specification, which assists in comprehension.

5.5 Epsilon Flock

Epsilon Flock is a model to model transformation language developed for model migration strategies. It uses novel conservative copying algorithm for relating source and target model elements. In this tool the user only needs to specify the parts of the
5.5. EPSILON FLOCK

model that do not conform to the extended metamodel, while the unaffected model elements are copied automatically. Flock has a rule-based specification that mixes declarative and imperative styles. The migration strategies are organised into sets of modules which further comprise with number of rules.

5.5.1 Specification of migration task in Epsilon Flock

The model migration case of Transformation Tool Contest has been solved by L.Rose and his colleagues at York University [126]. The solution to the migration problem is shown below:

Actions, Transitions and Final State

The Flock specification maps ActionState to OpaqueAction through a single migration rule. Each rule specifies an original type (ActionState) and an evolved type (Opaque-Action). Rules can also specify an optional guard (a boolean expression express with the when keyword) to limit applicability of the rules. Flock uses the Epsilon Object Language (EOL) [74] to specify the body of migration rules. The code below presents the transformation of ActionState to OpaqueAction in Flock.

```
1 migrate ActionState to OpaqueAction
```

Similarly, the code below shows that the instance of FinalState and Transitions are transformed to ActivityFinalNode and ControlFlow in the target model, respectively.

```
1 migrate FinalState to ActivityFinalNode
2 migrate Transition to ControlFlow
```

Pseudostates

Pseudostates are the elements in the source model that no longer exist in the target model. Listing below shows the code for mapping Pseudostates to corresponding Nodes. The variables original and migrated are used in the body of the rule to access the original and migrated model elements.

```
1 migrate Pseudostate to InitialNode when: original.kind = Original! PseudostateKind@initial
2 migrate Pseudostate to DecisionNode when: original.kind = Original!
3 PseudostateKind@junction
4 migrate Pseudostate to ForkNode when: original.kind = Original!
5 PseudostateKind@fork
6 migrate Pseudostate to JoinNode when: original.kind = Original!
7 PseudostateKind@join
```
It contains four migration rules, which migrate \textit{Pseudostates} to some subtype of \textit{Node} (InitialNode, DecisionNode, etc) based on the value of the original model.

\textbf{Activities}

In UML 2.2, activities no longer inherit from state machines. As such, some of the features defined by activity have been renamed. Specially \textit{transitions} have become \textit{edges} and \textit{partitions} have become \textit{groups}. Furthermore, the \textit{states} of an \textit{Activity} are now contained to \textit{nodes}, rather than in the \textit{subvertex} feature of a composite state accessed via the \textit{top} feature of \textit{Activity}. The code below shows the mapping of these elements. Flock defines the built-in \textit{equivalent} operation to find the equivalent migrated model elements.

\begin{verbatim}
1 migrate ActivityGraph to Activity {
2    migrated.edge = original.transitions.equivalent();
3    migrated.group = original.partition.equivalent();
4    migrated.node = original.top.subvertex.equivalent();
5 }
\end{verbatim}

In UML 1.4, the guard feature of Transition references a \textit{Guard}, which in turn references an \textit{Expression} via its \textit{expression} feature. In UML 2.2, the \textit{Guard} feature of \textit{Transition} references an \textit{OpaqueExpression} directly. The listing below captures this in Flock.

\begin{verbatim}
1 migrate Guard to OpaqueExpression {
2    migrated.body.addAll(original.expression.body);
3 }
\end{verbatim}

\textbf{Partitions}

In UML 1.4 activity diagrams, \textit{Partition} specifies a single containment reference for its \textit{contents}. In UML 2.2 activity diagrams, \textit{partition} specify two containment features for their \textit{contents}, \textit{edges} and \textit{nodes}. The code below shows the transformation of \textit{Partition} to \textit{ActivityPartition} in Flock.

\begin{verbatim}
1 migrate Partition to ActivityPartition {
2    migrated.edges = original.contents.collect(e:Transition | e.equivalent());
3    migrated.nodes = original.contents.collect(n:StateVertex | n.equivalent());
4 }
\end{verbatim}

\textbf{ObjectFlows}

In UML 1.4 activity diagrams, object flows are specified using \textit{ObjectFlowState}, a subtype of \textit{StateVertex}. In UML 2.2 activity diagrams, object flows are modelled using
5.5. EPSILON FLOCK

<table>
<thead>
<tr>
<th>Transformation language</th>
<th>Abstraction level</th>
<th>Size: LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Medium</td>
<td>23 lines (Low)</td>
</tr>
</tbody>
</table>

| syntactic complexity    | 23 operators, 50 features |
| Structural complexity   | 0 calls, 0 recursive calls, max call depth = 0 |
| Complexity total        | 73 (Low) |
| Modularity              | factorisation: 100%, cohesion: 100%, coupling: 0% |
| Modularity total        | High |
| Extensibility to the core task | High |
| Interoperability: inclusion in a transformation process | High |
| Development effort      | 2 hours (Low) |
| Close to well-known notation | Medium |
| Execution time          | Medium |
| syntactic correctness   | Low |
| Correctness (Termination) | Low |
| Correctness (Confluence) | Medium |
| Effectiveness           | High |
| Fault tolerance         | High |
| Eclipse plug-ins        | High |
| Maturity                | Low |

Table 5.7: Evaluation criteria of Flock solution for migration case study

A subtype of ObjectNode. In UML 2.2, flows that connect to and from ObjectNodes must be represented with ObjectFlows rather than ControlFlows. The code below shows this part of transformation.

```java
migrate ObjectFlowState to ActivityParameterNode
migrate Transition to ObjectFlow when original.source.isTypeOf(
    ObjectFlowState) or original.target.isTypeOf(ObjectFlowState)
```

5.5.2 Evaluation properties for Flock

Table 5.7 summarises the evaluated attributes of the transformation language and tools for the Flock solution.
5.5. EPSILON FLOCK

Transformation Language Properties

Flock is a declarative, textual and external language. It updates models in-place and uses conservative copy terms for model migration cases. In conservative copy, model elements that conform to both the source and target metamodel are automatically copied from the source to the migrated model. The transformation size is low, with total number of twenty-three lines of code avoiding the need to specify the unaffected elements. The conservative copying strategy generates less complex transformation in comparison to other investigated approach. Flock has twenty-three operators such as `add`, `=`, and fifty features such as `kind`, `body` in its specification. Moreover, there is no call in the transformation specification and the structural complexity is zero. The factorized transformation is bounded inside one module. The extension to the core tasks are provided comprehensively in Flock [126]. The approach uses OCL notation for the specification and the level of closeness to the well-known notation is Medium. According to the questionnaire used in the workshop, Flock was considered the most understandable approach. The development team reported four hours for implementing this case study.

Transformation Tool Properties

The execution time of transformation for the specified input model is 140 ms. The order of execution of rules is not clear because of recursive descent execution. However, in this transformation different order of execution generates the same result. Flock generates the correct target model but there is no facility to prove its syntactic correctness. Moreover, termination of transformation is not clear because conservative copy automatically ignores model elements that do not conform to the metamodel. Importantly, Flock developed particularly for performing migration task and it generates the correct result in an appropriate time.

The robustness is high for this approach as it shows where the error appeared in the specification. The maturity of tool is low and was available since 2009. The positive point of this tool is to have an interface with Eclipse.
5.6 GrGen.NET

The Graph Rewrite Generator (GrGen.Net) tool is a general purpose graph rewrite system enabling direct formulation of complex transformation problems. GrGen.NET compiles declarative specifications of graph metamodels, patterns, and rewrite rules into .NET modules. In addition, it supports high level rule application control language and sequential iterative control over the rules.

5.6.1 Specification of migration task in GrGen.NET

The GrGen.NET solution was developed by S. Buchwald and his colleagues from University of Karlsruhe [24]. The problem was solved by iterated application of match rule for edge or node of UML 1.4 and rewrites the corresponding to UML 2.2 target type. GrGen has a strategy for separating the transformation for node type and edge type.

**Action State**

Rules has two parts in GrGen specification; pattern part to specify the graph pattern for matching, and a nested rewrite part to specify the changing. The transformation from ActionState to OpaqueAction is presented below:

```
1 rule transform ActionState {
2   state : minuml ActionState ;
3   modify {
4     opaque : uml OpaqueAction<state >;
5     eval { opaque. name = state. name; }
6   }
7 }
```

Line 2, encodes the pattern part of a node as an ActionState. It then retypes to a node opaque of type OpaqueAction in lines 3 to 4. Furthermore, the name attribute of the original node is assigned to the name attribute of the new retyped node in the attribute evaluation eval in line 5.

**Pseudostates**

For rewriting Pseudostates to different nodes in UML 2.2 local information is included in the specification. In order to specify four possible target types for the source type an alternative construct is used.
5.6. GRGEN.NET

The if-clause checks the kind value of the source node to be reflected as the correct type in the target model.

Transitions

For rewriting the Transition nodes alternative statement is used. It distinguish the connected nodes of ObjectFlowState (rewritten to Pin). Lines 4 to 16 identifies the transformation to ControlFlow, preventing the navigation patterns that is matched on the object flow situations.

```plaintext
1 rule transformTransition {
2   transition : Transition ;
3
4   alternative {
5     controlFlow {
6       negative {
7         transition < :StateVertex incoming : uml:Pin ;
8       }
9       negative {
10      transition < :StateVertex outgoing : uml:Pin ;
11     }
12     modify {
13       cf : uml:ControlFlow<transition >;
14       eval { cf.name = transition.name; }
15     }
16     }
17     incomingObjectFlow {
18      transition < :StateVertex incoming : uml:Pin ;
19      modify {
20       of : uml:ObjectFlow<transition >;
21       eval { of.name = transition.name; }
22     }
23     }
24     outgoingObjectFlow { / similar to case above / }
```
The remaining part of transformation are similar to the rules that have explained here. The complete solution can be found in [25].

### 5.6.2 Evaluation properties for GrGen

Table 5.8 summarises the evaluated attributes of the transformation language and tools for the GrGen.NET solution.

<table>
<thead>
<tr>
<th>Transformation language</th>
<th>Abstraction level</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size: LOC</td>
<td>219 lines (High)</td>
</tr>
<tr>
<td>syntactic complexity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(no. of operators, no. of features)</td>
<td>29 operators, 114 features</td>
<td></td>
</tr>
<tr>
<td>Structural complexity</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Complexity total</td>
<td>143 (Medium)</td>
</tr>
<tr>
<td>Modularity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modularity total</td>
<td>factorisation: 73%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cohesion: 100%, coupling: 0%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Medium</td>
</tr>
<tr>
<td>Extensibility to the core task:</td>
<td>Medium</td>
<td></td>
</tr>
<tr>
<td>Interoperability:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>inclusion in a transformation process</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Development effort</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Close to well-known notation</td>
<td>4 hours (Medium)</td>
<td></td>
</tr>
<tr>
<td>Transformation tool</td>
<td>Execution time</td>
<td>140 ms</td>
</tr>
<tr>
<td></td>
<td>syntactic correctness</td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>Correctness (Termination)</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Correctness (Confluence)</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td>Effectiveness</td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>Fault tolerance</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Eclipse plug-ins</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Maturity</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 5.8: Evaluation criteria of GrGen solution for migration case study
Transformation Language Properties

GrGen specifies transformation in the high level of abstraction, except some imperative code fragments for complex transformations. Handling nodes and edges separately generates a simple solution to the transformation task, but result in more lines of code (219). There are twenty-nine operators and 114 features in the specification. The number of expressions greater than size seven in this specification are fifteen and the number of repeated expressions are four. This means that, the factorization is eleven over fifteen and the percentage of it is 73%. In addition, the specification can be encapsulated inside one module with no calls in the specification. Extension to the core tasks are provided in the GrGen solution (Section 4 of [24]). The first extension was solved through an additional declarative rule, while the second one was not tackled directly due to the lack of a concrete syntax. The last extension; however was not generated in the GrGen solution. OCL notations are used in the specification. The level of understandability for this approach is around average, based on the workshop participants point of view. The development effort for this case study was about four hours.

Transformation Tool Properties

The GrGen transformation tool generated the target in 140 ms. It provided the correct target model for the migration case study, but it is not possible to prove the syntactic correctness. The termination is clear because of the linear solution of this approach. The explicit scheduling of different rules in GrGen solution provides the clear order of executions, though it is not possible to control the order in which rules are applied to the individual elements (this is only possible in languages like Kermeta, which has precise control over application of rules to individual elements.). An error message is given in GrGen if the rules are erroneous according to the specification. It includes the location of the error in the rule specification file. The GrGen solution is considered as an effective approach for model migration apart from substantial size of specification. GrGen.NET has been publicly available since 2003 and about fifty transformation case studies have been implemented using it. In addition, complete interoperability with Eclipse is provided in GrGen.
5.7. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th>Approach</th>
<th>LOC</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML-RSDS</td>
<td>38</td>
<td>Medium</td>
</tr>
<tr>
<td>FLOCK</td>
<td>23</td>
<td>Low</td>
</tr>
<tr>
<td>GrGen</td>
<td>219</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 5.9: Size comparison for migration case study

5.7 Comparison of approaches

In [123] the solutions are evaluated according to correctness, conciseness, understandability, appropriateness, tool maturity and extensibility. The result of the evaluation indicated Flock as the winner for the model migration case study in the Transformation Tool Contest 2010. We compare UML-RSDS, Epsilon Flock and GrGen according to the evaluation framework in Chapter 3.

Size

The obvious size metric is the number of lines of code in the specification. As Table 5.9 shows there is a big difference between the number of lines of code in GrGen solution than UML-RSDS and Flock. It is most noticeable that the large size of transformation in GrGen is related to the strategy of separately processing the node and edge elements. It makes the transformation more modular and usable; however increases the size and complexity of specification. Table 5.10 presents the complexity comparison of the solutions for the migration case study. The observation shows that, the conservative copying strategy in Flock reduces the specification size and complexity by avoiding the need for specifying elements that are unaffected by metamodel evolution. However, in UML-RSDS it is essential to specify the effected and non-effected elements. The ranking of the solutions on the basis of size is: Flock; UML-RSDS; GrGen.

Complexity

There is no call in the specification of transformations, therefore, the complexity is measured by comparing the number of syntactic complexity in each approach. Table 5.10 summarizes the result of complexity measures of the approaches. It can be
5.7. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th>Approach</th>
<th>Number of operators</th>
<th>Number of features</th>
<th>Total</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML-RSDS</td>
<td>120</td>
<td>140</td>
<td>260</td>
<td>High</td>
</tr>
<tr>
<td>FLOCK</td>
<td>23</td>
<td>50</td>
<td>73</td>
<td>Low</td>
</tr>
<tr>
<td>GrGen</td>
<td>29</td>
<td>114</td>
<td>143</td>
<td>Medium</td>
</tr>
</tbody>
</table>

Table 5.10: Complexity comparison for migration case study

<table>
<thead>
<tr>
<th>Approach</th>
<th>Factorisation</th>
<th>Coupling</th>
<th>Cohesion</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML-RSDS</td>
<td>87%</td>
<td>0%</td>
<td>100%</td>
<td>Medium</td>
</tr>
<tr>
<td>FLOCK</td>
<td>100%</td>
<td>0%</td>
<td>100%</td>
<td>High</td>
</tr>
<tr>
<td>GrGen</td>
<td>73%</td>
<td>0%</td>
<td>100%</td>
<td>Medium</td>
</tr>
</tbody>
</table>

Table 5.11: Modularity measures for migration case study

observed that, the number of operators, features and total complexity of the UML-RSDS approach is extremely greater than the Epsilon flock. The reason for this is that the UML-RSDS approach has explicit specification, while Flock uses implicit copying strategies. The GrGen transformation approach presents the moderate specification in terms of complexity for model migration task. The ranking of the solutions on the basis of complexity is: Flock; GrGen; UML-RSDS.

Modularity

Modularity of approaches is investigated according to the cohesion and factorisation. It can be observed that there is no calls in the specification of the approaches in this chapter and all of them are encapsulated inside one module. This results to have 100% cohesion in each solution. Table 5.11 indicates the percentage of factorisation and modularity ranking for case studies. The strong modularity percentage is reported for all the approaches.

Development effort

The development effort of a transformation refers to the time spent on development of migration case study and does not include the time spent for understanding the problem (developers are experts in all cases). Table 5.12 shows the development effort for the approaches. The evaluation showed that the task developer spent less
time for development of migration task in the Flock tool in comparison to the UML-RSDS and GrGen approach. This occurs as the copying strategy in Flock reduces the development time and effort of writing more lines of code.

Correctness

Correctness was assessed by three separate 5-point measures of syntactic correctness, termination and confluence. Table 5.13 shows the correctness measure for model migration case study. The correctness of the UML-RSDS approach is high in comparison to other approaches. Specially it was mentioned that, the syntactic correctness is checked by translation to the B formal method. Moreover, the termination is clear by using bounded loops in the specification. Alongside these, the confluence is guaranteed as different order of execution of the rules generate the same result. The termination of the Flock solution is not clear because of the non-determinate order of the recursive descent strategy Also confluence is not promised in the GrGen approach. The ordering with regard to correctness is: UML-RSDS; GrGen; Flock.

Maturity

The UML-RSDS approach is relatively immature, having been publicly available for less than two years. Around ten substantial case studies have been implemented using the UML-RSDS tool. The weak point of this tool is that there is no interface to Eclipse; instead model input and output is by means of text files. The first release of
5.7. COMPARISON OF APPROACHES

Flock was in September 2009. It has Eclipse plug-ins and is applied to six examples of migration problems. The question that were asked on the Epsilon Flock forum indicates that there are a few people using Flock to solve their migration problems. GrGen.NET has been publicly available since 2003 and about fifty transformation case studies have been implemented using it.

Usability

The workshop participants were asked to fill out evaluation sheet for each solution. One of the evaluation factors was understandability (the other usability features of framework is not checked for this approach). It means, how easy it is to read and understand the transformation specification. Three options defined for measuring understandability; i) no idea how it works (-3) ii) some idea how it works (0) iii) fully understand how it works (3). In total, twelve marking forms were submitted to the workshop. The ranking of approaches according to average score for each solution is shown in Table 3 in [123]. The result is therefore: Flock (2.7); GrGen (1.4) and UML-RSDS (-0.5). I have to mention that this score is for the older version of UML-RSDS (the specifications were not fully OCL). However, the new version scores higher in the recent evaluation of TTC workshop. This ordering agrees with the ordering of evaluation for complexity, modularity and development time. It can be seen that, although UML-RSDS is based on the standard UML and OCL notations the specification paradigm in this approach is not understandable for expert workshop participants.

5.7.1 Evaluation summaries

The above section compared transformation languages according to specified criteria for migration task. The observation showed a significant difference between the lines of code in the Flock (23 lines) and UML-RSDS (38 lines), than GrGen (219 lines) in this case study. Separation of nodes and edges elements in GrGen increases the size and complexity of specification. Having less lines of code in the Flock than other investigated approaches is because copying strategy prevents from repeating of unchanged elements. The finding indicated that the explicit specification style in UML-RSDS resulted in to have more complex transformation in terms of structure
than the Flock migration tool. There is no call in the specification of solutions for performing this case study. Flock’s specification is factorised. In general, the approaches investigated here provide the solution with medium and high modularity.

Flock provided the three extensions specified in the case study description, while GrGen generated the first extension completely. In fact, it is possible to extend UML-RSDS for the three extensions, but it requires more effort and lines of code in the specification. The development time for the UML-RSDS approach reported longer than other approaches as it was a first migration case performed with this tool, and required explicit specification for mapping individual elements. Flock and GrGen use OCL notation for specifying the constraints and UML-RSDS is based on UML and OCL and no additional notation is used in its specification.

The syntactic correctness is provided in UML-RSDS by translation to the B formal method and proof tool. In addition, termination is provided for this solution, since all the processing is done by bounded loops. Confluence of UML-RSDS is high as the order of applications within the iteration does not affect the result. However, in Flock termination of transformation is not clear because of the recursive descent execution strategy of the approach. In addition, syntactic correctness is not visible because conservative copy automatically ignores model elements that do not conform to the metamodel. Despite this the termination is clear in GrGen due to the linear solution of this approach. Explicit order of rules in GrGen prevents a fault from arising in the case of conflict. However, rule conflicts does not ensure confluence.

In the Flock and GrGen approaches the error message is shown in addition to showing the location where the problem has arisen. While, in the UML-RSDS approach there is only a Java error without identifying the location of error. It has been reported that GrGen has been publicly available since 2003 and about fifty transformation case studies have been implemented using it, while UML-RSDS and Flock have been available recently.

In this section separate tables are used to show the summary characteristic values of each approach. The evaluation showed that UML-RSDS scores highly for Accuracy and Efficiency, but poorly for Reliability, Usability and Portability, according to Table 5.14. An automatic generation of unaffected elements is considered as a possible extension for the UML-RSDS tool. This will reduce the complexity and development effort of the transformation in this approach and therefore improves its functionality.
### 5.7. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Subcharacteristic</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Suitability</td>
<td>Abstraction level ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Size ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Effectiveness ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Development effort ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Execution time ✓</td>
</tr>
<tr>
<td>Accuracy</td>
<td></td>
<td>Correctness ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Completeness ✓</td>
</tr>
<tr>
<td>Interoperability</td>
<td></td>
<td>Embeddable in transformation process ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Close to well-known notation ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Eclipse plug-ins ×</td>
</tr>
<tr>
<td>Reliability</td>
<td>Maturity</td>
<td>History of use ×</td>
</tr>
<tr>
<td></td>
<td>Fault tolerance</td>
<td>Tolerance of false assumptions ×</td>
</tr>
<tr>
<td>Usability</td>
<td>Understandability</td>
<td>×</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Time behaviour</td>
<td>Execution time ✓</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Changeability</td>
<td>Size ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modularity ✓</td>
</tr>
<tr>
<td>Portability</td>
<td>Adaptability</td>
<td>Extensibility ×</td>
</tr>
</tbody>
</table>

Table 5.14: Quality characteristics of UML-RSDS for the migration case study

The next highly important evaluation result has been reported for Flock in Table 5.15. It scores highly for most of the criteria investigated in this comparison, including Suitability, Interoperability, Usability, Efficiency, Maintainability and Portability.

Table 5.16 indicates the high scores of GrGen for Accuracy, Interoperability, Reliability, Efficiency and Portability. It scores poorly for Usability.

We present the overall rankings of the approaches in terms of the ISO/IEC 9126 characteristics in Table 5.17. These tables present the number of good characteristics of each approach. If some factors are more investigated for a particular purpose, it will count more in the overall ranking table. Then the total score presents the general view about each approach. For measures like complexity, size development time and execution time, a high value means a Low score. For the rest of the attributes the high
5.8. Conclusion

Model migration is considered as a specialisation of model to model transformation. This chapter compared the general purpose transformation language of UML-RSDS, the graph transformation tool of GrGen and the Flock migration tool on the migration case study that was presented at Transformation Tool Contest 2010 workshop. The way in which activity diagrams are modelled in the UML differs between versions 1.4

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Subcharacteristic</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Suitability</td>
<td>Abstraction level ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Size ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Effectiveness ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Development effort ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Execution time ✓</td>
</tr>
<tr>
<td></td>
<td>Accuracy</td>
<td>Correctness ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Completeness ✓</td>
</tr>
<tr>
<td>Interoperability</td>
<td>Embeddable in transformation process ✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Close to well-known notation ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Eclipse plug-ins ✓</td>
</tr>
<tr>
<td>Functionality</td>
<td>Maturity</td>
<td>History of use ×</td>
</tr>
<tr>
<td>Compliance</td>
<td>Fault tolerance</td>
<td>Tolerance of false assumptions ✓</td>
</tr>
<tr>
<td>Usability</td>
<td>Understandability</td>
<td>✓</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Time behaviour</td>
<td>Execution time ✓</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Changeability</td>
<td>Size ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modularity ✓</td>
</tr>
<tr>
<td>Portability</td>
<td>Adaptability</td>
<td>Extensibility ✓</td>
</tr>
</tbody>
</table>

Table 5.15: Quality characteristics of Epsilon Flock for the migration case study

value means a High score. I also counted the Medium score as a good representative for characteristic of approaches.

The GrGen and Flock transformation approaches generates quite a similar score, but Flock is less complex and more understandable from the users point of view.
5.8. CONCLUSION

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Subcharacteristic</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Suitability</td>
<td>Abstraction level ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Size ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Effectiveness ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Development effort ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Execution time ✓</td>
</tr>
<tr>
<td>Accuracy</td>
<td></td>
<td>Correctness ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Completeness ✓</td>
</tr>
<tr>
<td>Interoperability</td>
<td></td>
<td>Embeddable in transformation process ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Close to well-known notation ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Eclipse plug-ins ✓</td>
</tr>
<tr>
<td>Functionality</td>
<td>compliance</td>
<td>Close to well-known notation ✓</td>
</tr>
<tr>
<td>Reliability</td>
<td>Maturity</td>
<td>History of use ✓</td>
</tr>
<tr>
<td></td>
<td>Fault tolerance</td>
<td>Tolerance of false assumptions ✓</td>
</tr>
<tr>
<td>Usability</td>
<td>Understandability</td>
<td>×</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Time behaviour</td>
<td>Execution time ✓</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Changeability</td>
<td>Size ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modularity ✓</td>
</tr>
<tr>
<td>Portability</td>
<td>Adaptability</td>
<td>Extensibility ✓</td>
</tr>
</tbody>
</table>

Table 5.16: Quality characteristics of GrGen.NET for the migration case study

<table>
<thead>
<tr>
<th>Approach</th>
<th>Funct. (11)</th>
<th>Rel. (2)</th>
<th>Usab. (3)</th>
<th>Effic. (2)</th>
<th>Maint. (2)</th>
<th>Port. (1)</th>
<th>Total (21)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flock</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>GrGen</td>
<td>11</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>UML-RSDS</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 5.17: Overall ranking of approaches for the migration case study

to 2.2 of the specification. In the former, activities are defined as a special case of state machines, while in the latter it is defined with more general semantic base. The case study maps between these two versions of UML.

Epsilon Flock is an appropriate tool for performing migration tasks due to the automatic copying strategy of model elements that are unaffected by metamodel adaptations. It results to spend less time on development of the case study. In addition, the transformation specification has an appropriate size, and is less complex in comparison to other investigated approaches. The Flock language is expressive
5.8. CONCLUSION

enough to specify the migration task and the transformation correctly migrates the test case included in the case resources. However, the implicit copying restricts Flock from providing a comprehensive correctness. This occurs as the conservative copy automatically ignores model elements that do not conform to the metamodel. GrGen is a general purpose graph rewrite system which performs migration task by separating the process of edges and nodes. This results to have a transformation with excessive lines of code and complexity. The GrGen and Flock transformation approaches generates quite a similar results, but Flock is more understandable from the users point of view. The abstraction level of UML-RSDS approach is quite high. The tool specify the model migration transformation case study in a direct manner by specifying affected and non-affected elements. This leads to a bigger size and more complex transformations than Flock approach. Further improvement in UML-RSDS approach should consider an automatic way of copying unaffected elements. The tool has the advantage of using standard UML and OCL notations to specify transformations, which then reduce the cost of learning new notations. Specially, for UML-RSDS it can be shown that for each of the unordered iterations used in the transformation, the order of applications within the iteration does not affect the result of the iteration. In fact, model migration with Flock is likely to be less productive than UML-RSDS and GrGen in terms of execution time. The finding showed that it takes longer to execute transformations in Flock, because migration strategies are interpreted, while UML-RSDS and GrGen are compiled.
Case Study 4: Comparative Evaluation of Transformation Approaches on the Quality Improvement case study

In this chapter a comparison of five established model transformation languages from different language categories (1) the GrGen graph transformation language [60], (2) the Kermeta imperative language [35], (3) the QVT-Relational declarative language [117], (4) the ATL hybrid approach [62] and (5) the UML-RSDS general purpose MDE tool [94] on a complex refactoring case study is presented. The evaluation is based on the framework specified in Chapter 3. Such a transformation iteratively rewrites a model in-place to improve some quality characteristics. It operates on a single model, and is endogenous, with the same source and target language. This chapter is based on the paper that has been submitted to Science of Computer Programming journal in 2012. Additionally, the case study has been accepted as a challenging task in Transformation Tool Contest in 2013 [90].

In this chapter first a description to the problem domain is provided. Second, the evaluation criteria for the case study is described. Following that, different test cases with increasing size and complexity for their application to the quality improvement case study is presented. Next, the solutions to the case study is given. Different solutions are then compared on the relative values of their characteristics to find the most appropriate ones. Lastly, we finish with a conclusion.
6.1 Forth Case study : Class Diagram Restructuring Transformation

This case study is selected as being typical of the category of refactoring/restructuring transformations. It involves the creation, deletion and relocation of elements within a single model, and requiring fixpoint iterations of prioritised rules, and fine-grained control over the rule execution to achieve optimal results. Evaluation of transformation approaches on this case study, therefore should give results indicative of the suitability of the approaches for this general category of problems.

The aim of the case study transformation is to remove from a UML class diagram all cases where there are two or more sibling or root classes which all own a common-named and typed attribute, and to rationalise and amalgamate all such apparent clone copies of attributes. It is used as one of a general collection of transformations which aim to improve the quality of a specification or design level class diagram. Figure 6.1 shows the metamodel for the source and target language of this transformation. It can be assumed that:

Class name uniqueness: No two classes (instances of Entity) have the same name.

Type name uniqueness: No two types have the same name.

Property name uniqueness in classes: The owned attributes (properties) of each class have distinct names within the class, and do not have common names with the attributes of any superclass.
6.1. FORTH CASE STUDY: CLASS DIAGRAM RESTRUCTURING TRANSFORMATION

Figure 6.2: Rule 1

Single inheritance: There is no multiple or redundant inheritance.

These pre-conditions must also be preserved by the transformation.

The informal transformation steps are the following:

(1) Pull up common attributes of all direct subclasses: If the set of all direct subclasses \( g = c . \text{specialisation}.\text{specific} \) of a class \( c \) has two or more elements, and all classes in \( g \) have an owned attribute with the same name \( n \) and type \( t \), add an attribute of this name and type to \( c \), and remove the copies from each element of \( g \) (Figure 6.2).

(2) Create subclass for duplicated attributes: If a class \( c \) has two or more direct subclasses \( g = c . \text{specialisation}.\text{specific} \), and there is a subset \( g_1 \) of \( g \), of size at least 2, all the elements of \( g_1 \) have an owned attribute with the same name \( n \) and type \( t \), but there are elements of \( g - g_1 \) without such an attribute, introduce a new class \( c_1 \) as a subclass of \( c \). \( c_1 \) should also be set as a direct superclass of all those classes in \( g \) which own a copy of the cloned attribute. Add an attribute of name \( n \) and type \( t \) to \( c_1 \) and remove the copies from each of its direct subclasses (Figure 6.3).

(3) Create root class for duplicated attributes: If there are two or more root classes all of which have an owned attribute with the same name \( n \) and type \( t \), create a new root class \( c \). Make \( c \) the direct superclass of all root classes with such an attribute, and add an attribute of name \( n \) and type \( t \) to \( c \) and remove the copies from each of the direct subclasses (Figure 6.4).
6.2 EVALUATION CRITERIA

It is a requirement of the transformation to minimise the number of new classes introduced, to avoid introducing superfluous classes into the model. This means that rule 1 “Pull up attributes” should be prioritised over rules 2 “Create subclass” or 3 “Create root class”.

For this case study there is no need to consider the derivation of an inverse transformation, since there is no unique inverse, and the reverse process has no utility.

6.2 Evaluation criteria

Table 6.1 summarises the evaluation criteria for model transformation approaches to be assessed for the case study solutions. Properties of transformation language
### 6.2. EVALUATION CRITERIA

<table>
<thead>
<tr>
<th>Transformation language</th>
<th>Abstraction level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size: LOC</td>
</tr>
<tr>
<td></td>
<td>Complexity: (1) syntactic complexity (number of operators, number of features)</td>
</tr>
<tr>
<td></td>
<td>(2) structural complexity</td>
</tr>
<tr>
<td></td>
<td>(i) total number calls</td>
</tr>
<tr>
<td></td>
<td>(ii) total recursive calls</td>
</tr>
<tr>
<td></td>
<td>(iii) maximum call depth</td>
</tr>
<tr>
<td></td>
<td>Modularity: (1) factorisation</td>
</tr>
<tr>
<td></td>
<td>(2) cohesion versus coupling</td>
</tr>
<tr>
<td></td>
<td>Extensibility:</td>
</tr>
<tr>
<td></td>
<td>extension to UML 2 class diagrams</td>
</tr>
<tr>
<td></td>
<td>Interoperability:</td>
</tr>
<tr>
<td></td>
<td>inclusion in a transformation process</td>
</tr>
<tr>
<td></td>
<td>Correctness:</td>
</tr>
<tr>
<td></td>
<td>provable syntactic correctness</td>
</tr>
<tr>
<td></td>
<td>Development effort</td>
</tr>
<tr>
<td></td>
<td>Close to well-known notation</td>
</tr>
<tr>
<td></td>
<td>Maturity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transformation tool</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Maximum capability</td>
</tr>
<tr>
<td></td>
<td>Correctness: termination, confluence</td>
</tr>
<tr>
<td></td>
<td>Effectiveness</td>
</tr>
<tr>
<td></td>
<td>Fault tolerance</td>
</tr>
<tr>
<td></td>
<td>Interoperability with Eclipse</td>
</tr>
<tr>
<td></td>
<td>Maturity</td>
</tr>
</tbody>
</table>

Table 6.1: Evaluation criteria for solutions

and properties of an implementation in a particular tool for that language is divided in this evaluation. The list of classifications of the properties that depend on the distribution of solution’s values for the quality improvement case study is given below. The rest of the properties are evaluated based on the classifications described in Chapter 3.

Table 6.2 shows the size classification for the evaluation of this case study. The

---

1 Abstraction level, specification size, complexity, development effort, syntactic correctness, extensibility, close to well-known notation, understandability, modularity

2 Effectiveness, execution time, termination, confluence, Eclipse plug-ins, maximum capability, fault tolerance, maturity
6.2. EVALUATION CRITERIA

<table>
<thead>
<tr>
<th>Value range</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..50 lines</td>
<td>Low</td>
</tr>
<tr>
<td>51..375 lines</td>
<td>Medium</td>
</tr>
<tr>
<td>376+ lines</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 6.2: Size categories

<table>
<thead>
<tr>
<th>Value range</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..120</td>
<td>Low</td>
</tr>
<tr>
<td>121..700</td>
<td>Medium</td>
</tr>
<tr>
<td>701+</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 6.3: Complexity categories

Ranges were chosen based on the distribution of size values for the case study solutions (24, 81, 83, 102, 653): the first quartile of values is considered low, the second and third medium, and the 4th high. A lower LOC value corresponds to a higher quality. I chose the mid point value to categories the attributes. For the size categorization the mid point of the first quartile and second quartile is 51.5 and the mid point of the third quartile and forth quartile is around 375.

Table 6.3 gives the classification of complexity as low, medium or high, based on the distribution of measured values (102, 132, 152, 190, 1214) in the case study solutions. The greater the number of expression operators\(^3\) or references to metamodel entities or features\(^4\) the more effort is required to comprehend and work with the specification. Similarly to the size categorization, the first quartile of values is considered low, the second and third medium, and the 4th high. A lower complexity value corresponds to a higher quality. For the complexity categorization the mid point of the first quartile and second quartile is approximately 120 and the mid point of the third quartile and forth quartile is around 701.

Extensibility considers how much effort is required to adapt the transformation to extended metamodels, specifically to the UML 2 class diagram metamodel in Figure 6.5.

Interoperability consists of Embedability: how effectively the transformation can be reused within a larger quality improvement process, consisting of transformations

\(^3\) such as =, :, \(\exists\)
\(^4\) such as Generalization, ownedAttribute, specific
to (1) remove redundant inheritance (Figure 6.6), (2) remove multiple inheritance (Figure 6.7): choose one superclass and replace the inheritance by a many-one association to the superclass. This is repeated until there is no multiple inheritance, (3) replace concrete superclasses by an abstract class and a new concrete subclass of this class (Figure 6.8). If a concrete class A is a superclass, make it abstract and add a new subclass AConcrete to represent the actual objects of A.

Development effort is measured in person-minutes. low effort is considered to be 0 to 110 minutes, medium 111 to 360, and high over 360 minutes. The first quartile of values is considered low, the second and third medium, and the 4th high. The lower the development time the more functional is the transformation approach.

The execution time was measured on standard Windows XP and Windows 7 laptop and PC hardware. In addition, all the executable solutions have been tested on the common SHARE platform [132]. A virtual machine with 1 GB of main memory and one processor core assigned to it. The classification categories for execution time are based on the execution time for the model of 100 copies of test case 2. It is low if the test case executes under 600ms; medium if between 600ms to 500s and high over
6.2. EVALUATION CRITERIA

Figure 6.6: Remove redundant inheritance

Figure 6.7: Remove multiple inheritance

500s.

Maximum capability is classified as low, medium, high according to Table 6.4, this is normalised based on the actual values for the solutions (1000, 5000, 5000, 100000, 100000).

<table>
<thead>
<tr>
<th>Value range</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..3000 elements</td>
<td>Low</td>
</tr>
<tr>
<td>3001..50000 elements</td>
<td>Medium</td>
</tr>
<tr>
<td>50001+ elements</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 6.4: Maximum capability categories

The effectiveness measure used is the proportion of clone copies of attributes
6.3. TEST CASES

Figure 6.8: Replace concrete superclasses by an abstract class and a new concrete subclass of this class

which are removed by the transformation. That is, if there are \( n \) copies of attributes which could, in principle, be removed by the rules of Section 6.1, and the implemented transformation removes \( m \leq n \) copies, the effectiveness is \( m/n \). Effectiveness measures of 75% or more are considered as high, measures of 50% to 74% are considered as medium and measures below 50% are low.

The section presented the evaluation criteria for assessment of quality improvement case study. We specified the particular boundary values for the criteria that depend on the actual distribution of the values in each solution. The rest of properties will be evaluate according to the list of criteria in Chapter 3. We compare the solutions of the approaches on the class diagram restructuring case according to abstraction level, size, complexity, modularity, extensibility, interoperability, development effort, closeness to the well-know notations such as OCL, execution time, maximum capability, correctness, effectiveness, fault tolerance, interoperability with Eclipse and maturity.

6.3 Test cases

The solutions are tested on five test cases of increasing size and complexity. These test cases represent both typical scenarios which could be expected to arise in class diagram modelling (test cases 1 to 4), and pathological examples designed to check the behaviour of the transformation in extreme cases (test case 5 and the duplications
6.3. TEST CASES

The first test case is a simple test for alternative applications of rule 2. Figure 6.9 shows the starting model. Applying the rule to classes B, C, D is the preferred choice because it creates fewer new classes than an application to A and B. The resulting model should have a superclass of B, C and D containing an attribute with name “b” and type T2 (Figure 6.10).

A larger test case, involving applications of rules 1 “Pull up attributes” and 3 “Create root class”, is shown in Figure 6.11. This illustrates the principle that rule 1 should be applied from the bottom of the class hierarchy upwards, to increase the opportunities to remove multiple copies of an attribute in one step.

The ideal result of applying the transformation to this test is shown in Figure
6.3. TEST CASES

6.12, but other results are possible in which the factoring of duplicated attributes is incomplete. For example, the sequence of rule applications:

\begin{align*}
&\text{b from } E, F \text{ to } D \quad \text{(rule 1)} \\
&\text{b from } D, G \text{ to } DG \quad \text{(rule 3)} \\
&\text{b from } B, C \text{ to } A \quad \text{(rule 1)} \\
&\text{b from } A, DG \text{ to } ADG \quad \text{(rule 3)}
\end{align*}

fails to amalgamate the clone copies of attribute \(a\), and has 80\% effectiveness.

The third test case has 20 classes and 20 properties, with 13 clone copies, and with a maximum inheritance depth of 4.
6.4 UML-RSDS SOLUTION FOR CLASS DIAGRAM RESTRUCTURING

CASE STUDY

The fourth test case has 105 classes arranged in an inheritance hierarchy of depth 3. Classes 1 to 5 are root classes, classes 6 to 30 inherit from these in groups of 5 (e.g., classes 6 to 10 inherit class 1), and classes 31 to 105 inherit the second level classes in groups of 3 (e.g., classes 31, 32 and 33 inherit from class 6). There are 12 essentially different attributes, each with 10 clones.

Test case 5 has 500 classes, each of which is a root class, and there are ten attributes in each class, with the attributes of each class being a copy of those in each other class (i.e., 5000 attributes, with 4990 clone copies).

In addition we carried out ‘stress testing’ to measure the maximum capability of a transformation tool, in terms of the maximum size of models which a transformation tool is capable of processing. These tests were models formed from duplicated copies of test case 2, of sizes up to 10,000 copies (40,000 classes, 40,000 attributes, 20,000 generalisations).

6.4 UML-RSDS solution for class diagram restructuring case study

UML-RSDS is a hybrid specification language, which defines system data by UML class diagrams, together with OCL constraints, and defines behaviour by operations of classes and by use cases which operate on the class diagram. Each use case can be declaratively specified by a set $Asm$ of preconditions, and a sequence $Cons$ of post-conditions. It is a general-purpose specification and design language for model-driven development, with a software tool which produces executable Java implementations of UML-RSDS models. Transformations can be specified in UML-RSDS in terms of the source and target metamodels they operate upon (represented as class diagrams) and the transformation functionality (represented as constraints of use cases, operations, activities or other UML behavioural elements).

The case study transformation specification in UML-RSDS consists of the class diagram of Figure 6.1, and a single use case which represents the transformation. The use case has precondition constraints $Asm$ expressing the pre-conditions of the transformation, and three postcondition constraints $Cons$, corresponding to the three informal rules:
(C1):

\[
\begin{align*}
& a : \text{specialization}\_\text{specific}\_\text{ownedAttribute} \& \\
& \text{specialization}.\text{size} > 1 \& \\
& \text{specialization}.\text{specific} \rightarrow \forall ( \text{ownedAttribute} \rightarrow \exists ( b \mid b.\text{name} = a.\text{name} \& b.\text{type} = a.\text{type}) ) \Rightarrow \\
& \quad a : \text{ownedAttribute} \& \\
& \quad \text{specialization}.\text{specific}.\text{ownedAttribute} \rightarrow \exists ( \text{name} = a.\text{name}) \rightarrow \text{isDeleted}()
\end{align*}
\]

This specifies that an instance \((self)\) of \(Entity\), and instance \(a\) of \(Property\) match the constraint LHS if: (i) \(a\) is in the set of attributes of all direct subclasses of \(self\), (ii) there is more than one direct subclass of \(self\), and (iii) every direct subclass of \(self\) has an attribute with the same name and type as \(a\).

The conclusion specifies that (i) the property \(a\) is moved up to the superclass \(self\), (ii) all attributes with name \(a.\text{name}\) are deleted from all direct subclasses of \(self\).

\(s \rightarrow \text{isDeleted}()\) is a built-in operator of UML-RSDS, which deletes the object or set of objects \(s\) from their model, removing them from all entity types and association ends.

Rule \((C1)\) is applied first, to the class \(A\) with subclasses \(B\) and \(C\), when this transformation is executed on the second test case (Figure 6.11), moving the copy of attribute \(b\) from class \(B\) up to class \(A\), and deleting the copy of \(b\) in \(C\).

(C2):

\[
\begin{align*}
& a : \text{specialization}\_\text{specific}\_\text{ownedAttribute} \& \\
& v = \text{specialization} \rightarrow \exists ( \text{specific}\_\text{ownedAttribute} \rightarrow \exists ( b \mid b.\text{name} = a.\text{name} \& b.\text{type} = a.\text{type}) ) \& \\
& v.\text{size} > 1 \Rightarrow \\
& \quad \text{Entity} \rightarrow \exists ( e \mid e.\text{name} = \text{name} + \text{”}2\text{”} + a.\text{name} \& \\
& \quad \quad a : e.\text{ownedAttribute} \& \\
& \quad \quad e.\text{specialization} = v \& \\
& \quad \quad \text{Generalization} \rightarrow \exists ( g \mid g : \text{specialization} \& g.\text{specific} = e ) ) \& \\
& \quad v.\text{specific}\_\text{ownedAttribute} \rightarrow \exists ( \text{name} = a.\text{name}) \rightarrow \text{isDeleted}()
\end{align*}
\]

The assumption specifies that an instance \((self)\) of \(Entity\), and instance \(a\) of \(Property\) match the constraint LHS if: (i) \(a\) is in the set of attributes of all direct subclasses of \(self\), (ii) the set \(v\) of all specializations of \(self\) whose class contains a clone attribute of \(a\) has size greater than 1. The \(let\) variable is used is \(v\) to factoring
6.4. UML-RSDS SOLUTION FOR CLASS DIAGRAM RESTRUCTURING
CASE STUDY

out repeated occurrences of expressions.

The conclusion specifies that: (i) a new entity $e$ is created, and the property $a$ is
moved up to $e$, (ii) the specializations of $e$ are $v$, (iii) $e$ is made a subclass of $self$, and (iv) all the clone copies of $a$ in $v$ are deleted.

This rule is applied first, to class $S$ with $v$ referring to subclasses $A$ and $B$, and
operating on attribute $a$, when this transformation is executed on the first test case (Figure 6.9). The rule creates a new subclass $e$ of $S$ and moves the copy of attribute $a$ from class $A$ up to class $e$, and deletes the copy of $a$ in $B$. The rule is then applied to the subclasses $C$ and $D$ of $S$ to promote attribute $b$ to another new subclass $e2$ of $S$.

$$(C3):$$
\begin{align*}
a & : ownedAttribute & \\
generalization.size & = 0 & \\
v & = Entity \rightarrow select(generalization.size = 0 & \\
  ownedAttribute \rightarrow exists(b | b.name = a.name & b.type = a.type)) & \\
v.size & > 1 \Rightarrow \\
Entity & \rightarrow exists(e | e.name = name + \_3 + a.name & \\
a & : e.ownedAttribute & \\
v.ownedAttribute & \rightarrow select(name = a.name) \rightarrow isDeleted() & \\
v & \rightarrow forAll(c | Generalization \rightarrow exists(g | g : e.specialization & g.specific = c)))
\end{align*}

This constraint matches against a class $self$ and an attribute $a$, if: (i) $self$ is a
root class, (ii) $a$ is an attribute of $self$ and has at least one clone in the attributes of
other root classes. $v$ is the set of all root classes with a copy of $a$. The let variable is
used is $v$ to factoring out repeated occurrences of expressions.

The effect of the constraint is then: (i) to create a new (root) entity $e$, (ii) to
move $a$ into the attributes of $e$, (iii) to delete all clones of $a$ from $v$, (iv) to make the
elements of $v$ direct subclasses of $e$.

On test case 2, this rule is applied to classes $A$ and $D$ to create a new superclass
$AD$, and to move attribute $a$ up to this class.

The design and implementation of these constraints is automatically synthesised
by UML-RSDS, following the process described in [88]. This design carries out a
fixpoint iteration of constraint 1, then of constraint 2, then of constraint 3, all in a
composite fixpoint iteration:

\[(\text{stat}(C1)\ast; \text{stat}(C2)\ast; \text{stat}(C3)\ast)\ast\]

where \(\text{stat}(Cn)\) implements \(Cn\).

In the second constraint, a \textit{let} definition has been used (the variable \(v\)) in order to avoid repeated computation of the expression defining \(v\). Likewise in \(C3\).

The inverse directions of associations are automatically set when one direction is set: in \(C2\) the update \(e\.\text{specialization} = v\) automatically removes the elements of \(v\) from \(self\.\text{specialization}\). Likewise in \(C2\) and \(C3\) the assignment \(a : e\.\text{ownedAttribute}\) removes \(a\) from \(self\.\text{ownedAttribute}\). \(g\) is added to \(e\.\text{generalization}\) when \(e\.\text{specific} = g\) is set.

### 6.4.1 Evaluation properties for UML-RSDS

Table 6.5 summarises the evaluated attributes of the transformation language and tools for this solution.

**Transformation language properties**

The language is capable of a high level of abstraction, and a primarily declarative specification can be given for this case study. In total, there are twenty-four lines of code in the specification.

There are no invocations of operations within the specification (no rule calls alternative rule), so the structural complexity is zero. There are seventy-eight operators and fifty-four features in the specification. The total complexity of transformation is medium. However, the specification fails to be modular because there are repeated occurrences of duplicated expressions. For example, the test

\[\text{ownedAttribute} \rightarrow \text{exists}(b \mid b\.\text{name} = a\.\text{name} \& b\.\text{type} = a\.\text{type})\]

occurs three times, once in each constraint antecedent, and should be factored out into a query operation \(\text{hasAttribute}(n : \text{String}, t : \text{Type}) : \text{Boolean of Entity}\). The UML-RSDS language provides the capability for such factoring by the use of operations invoked from constraints. There are sixty expressions of complexity seven or more in
6.4. UML-RSDS SOLUTION FOR CLASS DIAGRAM RESTRUCTURING

CASE STUDY

<table>
<thead>
<tr>
<th>Transformation language</th>
<th>Abstraction level</th>
<th>Syntactic complexity (no. of operators, no. of features)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>High</td>
<td>C1: 15 operators, 17 features</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C2: 29 operators, 20 features</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C3: 34 operators, 17 features</td>
</tr>
<tr>
<td></td>
<td>Size: LOC</td>
<td>24 lines (Low)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Structural complexity</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Complexity total</td>
<td>132 (Medium)</td>
</tr>
<tr>
<td></td>
<td>Modularity (Interoperable to Eclipse)</td>
<td>factorisation: 68%, cohesion: 100%, coupling: 0</td>
</tr>
<tr>
<td></td>
<td>Modularity total</td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>Extensibility:</td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>extension to UML 2 class diagrams</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interoperability:</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>inclusion in a transformation process</td>
<td>Low 120 min (Low)</td>
</tr>
<tr>
<td></td>
<td>Development effort</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Close to well-known notation</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td>Transformation tool</td>
<td>Execution time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum capability</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Syntactic correctness</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Correctness (Termination)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Correctness (Confluence)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Effectiveness</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fault tolerance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Eclipse plug-ins</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maturity (History of use)</td>
</tr>
</tbody>
</table>

Table 6.5: Evaluation criteria for UML-RSDS solution

the rules, and nineteen non-unique subexpressions of complexity seven or more, so the percentage of such unique subexpressions is 68%. Alongside, cohesion is 100%, as there are no calls within the specification.

The solution can be used for extended class diagram metamodels. The transformation can be included in the sequence (1), (2), (3) of quality improvement transformations defined in Section 6.2, some of its assumptions are established by transformations (1) and (2), so it must sequentially follow these transformations, it also does not interfere with their effect, so this sequential composition is valid. Likewise, transformation (3) should sequentially follow the transformation: (3) does not invalidate the transformation because the new classes introduced by (3) are all empty (they have only inherited features), so no new cases of duplicated attributes are intro-
6.4. UML-RSDS SOLUTION FOR CLASS DIAGRAM RESTRUCTURING
CASE STUDY

duced. The transformations can be composed internally in UML-RSDS by means of
use case inclusion. They could also be composed externally by successive invocation
of the Java programs produced for each transformation. Development effort for each
constraint was about thirty to fifty minutes, not including test case construction.

Although the UML-RSDS specification language is closely based upon UML no-
tations, the language and tool only have a relatively short history of application to
transformation problems (three years). In the usability survey, three of five respon-
dents considered that the specification was well-structured and that low or medium
effort was required to understand the specification. However only one respondent
correctly identified the part of the specification responsible for moving duplicated
attributes.

Transformation tool properties

UML-RSDS has comprehensive support for the proof of syntactic correctness, using a
translation to the B formal method and proof tool [81][92]. The transformation does
not ensure that new classes have unique names. There has to be some techniques
to force the name to be unique. For instance, adding underscore to make sure it
will be unique. It can be shown that the transformation does not introduce multiple
inheritance: in $(C_2)$ the original direct subclasses $v$ specific of $self$ are set instead to
be direct subclasses of the new class $e$, and this is set to be a direct subclass of $self$.
In $(C_3)$ the new class $e$ is a root class with direct subclasses the set $v$ of original root
classes. Attributes have distinct names within classes in the target model, because
for $(C_1)$ there cannot already be an attribute of $self$ with name $a$.name. For $(C_2)$
and $(C_3)$ the attribute is added to a new empty class. Termination can be shown
by using a variant function defined as the total number of attributes in the model.
However the rules are not confluent: different choices in the ordering of applications of
the rules to elements will result in different models. It is not possible to declaratively
specify fine-grained control over the order of processing elements by a rule $(Cons
constraint)$. In $(C_2)$ for example, we need to iterate through the subclass attributes
$a$ of a class in descending order of the size of the set of subclasses that contain a copy
of $a$. Only by using a more explicit style similar to that of the Kermeta solution,
could such control be enforced.
On the first test case the two occurrences of a are merged first; instead of the occurrences of b, so that two new classes are created. The second test case results in the correct model, as shown in Figure 6.12. Alternative input formats of the model could produce non-optimal results however. On the third test case the transformation takes 10 ms, and removes ten of eleven clone copies in ten steps, introducing four new classes. The effectiveness is 10/11 = 91%. On the fourth test case the transformation takes 50 ms and applies constraint (C1) thirty-nine times, and constraint (C2) sixteen times. It removes ninety-two clone copies, so has effectiveness 100%, but is non-optimal (2 additional classes are introduced). On the fifth test case the transformation takes 97 s and eliminates all the cloned attribute copies, with only one new class being introduced, an optimal result. Table 6.6 shows the execution times (on the SHARE platform) and effectiveness measures for the test cases considered.

<table>
<thead>
<tr>
<th>Test case</th>
<th>Number of classes</th>
<th>Number of attributes</th>
<th>Execution time (ms)</th>
<th>Effectiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>100% (non-optimal)</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>100% (optimal)</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>20</td>
<td>10</td>
<td>91% (non-optimal)</td>
</tr>
<tr>
<td>4</td>
<td>105</td>
<td>132</td>
<td>50</td>
<td>100% (non-optimal)</td>
</tr>
<tr>
<td>5</td>
<td>500</td>
<td>5000</td>
<td>96568</td>
<td>100% (optimal)</td>
</tr>
</tbody>
</table>

Table 6.6: Test case results for UML-RSDS

The Java implementation generated by the UML-RSDS tools shows an above-linear growth of execution time when constructing large sets of elements, as in rule 3 in the final test case. Table 6.7 shows the results of the maximum capability tests on the SHARE platform. The primary cause in the time complexity explosion are the two applications of rule 3, both involving the construction and search of sets of entities of size \( \text{dim} \), the number of duplications of test case 2. The UML-RSDS tool can express assumptions of a transformation, and check these when a model is loaded. It has no capability for handling runtime errors, these simply cause Java exceptions.

UML-RSDS is relatively immature, having been publicly available for two years. About ten case studies have been published. There is no interface to Eclipse; instead model input and output is by means of text files. There is a facility to export model data as XMI format files for import to Eclipse.
6.5. GRGEN.NET SOLUTION FOR CLASS DIAGRAM RESTRUCTURING
CASE STUDY

<table>
<thead>
<tr>
<th>Test case</th>
<th>Number of classes</th>
<th>Number of attributes</th>
<th>Execution time</th>
<th>Effectiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>2*100</td>
<td>400</td>
<td>400</td>
<td>90ms</td>
<td>100%</td>
</tr>
<tr>
<td>2*200</td>
<td>800</td>
<td>800</td>
<td>330ms</td>
<td>100%</td>
</tr>
<tr>
<td>2*500</td>
<td>2000</td>
<td>2000</td>
<td>2363ms</td>
<td>100%</td>
</tr>
<tr>
<td>2*1000</td>
<td>4000</td>
<td>4000</td>
<td>13s</td>
<td>100%</td>
</tr>
<tr>
<td>2*5000</td>
<td>20000</td>
<td>20000</td>
<td>156s</td>
<td>100%</td>
</tr>
<tr>
<td>2*10000</td>
<td>40000</td>
<td>40000</td>
<td>1137s</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 6.7: Maximum capability test results

6.5 GrGen.NET solution for class diagram restructuring case study

This section describes the GrGen.NET solution to the case study. The solution is developed by Dr Pieter Van Gorp from Eindhoven University of technology. The GrGen has a feature to prioritise rules for the purpose of reducing the number of new classes generated during the refactoring transformation. The formalization of this feature in GrGen syntax is as follows:

```plaintext
1 xgrs { rule1 || rule2 || rule3 }*
```

The above script provides the sequence of ordering the rules in the GrGen specification. The second rule executes if rule1 fails and the third rule executes if the second rule fails. The * operator iterates while at least one of the three rules matches.

The specification of first rule in GrGen syntax is as follows:

```plaintext
1 rule rule1 {
2   c : Class;
3      :SuperOf(c, g1); :SuperOf(c, g2);
4      g1 : Class :ownedAttribute-> a1 : Property :type-> t : Type;
5      g2 : Class :ownedAttribute-> a2 : Property;
6      :SameAttribute(a1, a2);
7      negative {
8          g3 : Class;
9            :SuperOf(c, g3);
10         g1;
11        negative {
12            g3 :ownedAttribute-> a3 : Property;
13            :SameAttribute(a1, a3);
14        }
15    }
16    modify {
17        c :ownedAttribute-> a4 : Property :type-> t;
18        eval {
19            a4 :name= a1 :name;
20        }
21        exec(RemoveAttributeFromSubclasses(c, a4) -> [createInverseEdges]);
22    }
```
6.5. GRGEN.NET SOLUTION FOR CLASS DIAGRAM RESTRUCTURING
CASE STUDY

Lines 2 to 6 encode a pattern of a class with two subclasses with the same attribute. This corresponds to the "two or more elements" from the informal rule description. This rule takes the class (line 2) with more than one subclasses (lines 3) by including the helper pattern SuperOf twice. It then bounds the properties of subclasses (lines 4, 5) and checks if they have the same name though the helper pattern SameAttribute. A class is considered as node, having an edge of type ownedAttribute. Alongside, Property is assigned to node with an edge of type Type in line 4. Following that in line 5 the similar specification is repeated for the other subclass, without considering the Type. This is because the Type is checked through the SameAttribute. The GrGen specification has limitation for specifying the universal qualifier; however it specifies them through the negations of the existing constraints. For instance, in the GrGen specification the description of "all classes in g have ..." is changed to "there is no class in g that does not have ...". GrGen codes this expression by checking that there should not be a class without the bounded attribute (line 7-15). Similarly, to check the properties the pattern sameAttribute is called. The rest of the code is responsible to add the common attribute to the superclass (lines 16-23). It creates new attributes for the superclass (line 17) and assigns its name to name of the common attribute in the subclasses (line 19). Finally, it removes the common attribute from the subclasses (line 21).

The definition of helper patterns ("SameAttribute" and "SuperOf") that have been used in the specification of the first rule is as follows:

```plaintext
pattern SameAttribute(a1: Property, a2: Property) {
  independent {
    if { a1 . name == a2 . name; }
    a1 . type < Type > t: Type < Type > a2;
  }
}
```

The helper pattern SameAttribute checks the identification of two attributes by comparing the name (line 3) and type (line 4) of them.

```plaintext
pattern SuperOf(c1: Class, c2: Class) {
  c1 . specialisation --> : Generalisation --> : specific --> c2;
}
```

The SuperOf helper pattern identifies the existence of path between a superclass $c1$ and a subclass $c2$ with an edge of type specialisation, a node of type Generalization and an edge of type specific. In addition RemoveAttributeFromSubclasses and
6.5. GRGEN.NET SOLUTION FOR CLASS DIAGRAM RESTRUCTURING
CASE STUDY

createInverseEdges are other helpers that rule1 relies upon. The RemoveAttributeFromSubclasses is defined as:

```java
1 rule RemoveAttributeFromSubclasses(c:Class, attr:Property) {
2     iterated {
3         c -->:specialisation --> Generalization -->:specific --> g:Class -->:ownedAttribute --> a:Property;
4         SameAttribute(a, attr);
5     modify {
6         delete(a);
7     }
8     }
9 }
```

The iterated block is used in RemoveAttributeFromSubclasses rule to apply the rewrite constructs as long as matches are found. The above code bounds the attribute of subclass and compares it with the attribute which has already bounded in the first rule (the common attributes in all the subclasses). Following that, it removes that attribute from the list of existing attributes in the superclass.

The inverse links are specified explicitly through createInverseEdges. The full listings of this solution can be found on the SHARE site for this chapter [132]. In addition the complete implementation is provided in Appendix C of the thesis.

6.5.1 Evaluation properties for GrGen

Table 6.8 summarises the evaluated attributes of the GrGen solution. Although the GrGen solution is larger in size and structural complexity than the UML-RSDS solution, it exhibits lower overall complexity. In addition, the approach has higher maturity than UML-RSDS.

Transformation language properties

GrGen specifies transformations at a high level of abstraction, while some imperative code fragments are required for complex specifications. The transformation specified in terms of graph pattern in this language, and consequently its syntax is related to the visual presentation of graph nodes and edges. There are 102 lines of code in the specification.

In total, there are twenty-three calls in the code with a maximum depth of two. However, there is no recursive call in the specification. The sum of operators are fifty-two and features are forty-nine. Figure 6.13 shows part of the call graph of this solution.
### 6.5. GRGEN.NET SOLUTION FOR CLASS DIAGRAM RESTRUCTURING

#### CASE STUDY

<table>
<thead>
<tr>
<th>Transformation language</th>
<th>Abstraction level</th>
<th>Size: LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntactic complexity</td>
<td>High</td>
<td>102 lines (Medium)</td>
</tr>
<tr>
<td>(no. of operators, no. of features)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structural complexity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complexity total</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modularity</td>
<td>factorisation 65%</td>
<td>cohesion = 100%, coupling = 0%</td>
</tr>
<tr>
<td>Modularity total</td>
<td>Medium</td>
<td></td>
</tr>
<tr>
<td>Extensibility:</td>
<td>extension to UML 2 class diagrams</td>
<td>Medium</td>
</tr>
<tr>
<td>Interoperability:</td>
<td>inclusion in a transformation process</td>
<td>High</td>
</tr>
<tr>
<td>Development effort</td>
<td>100 min (Low)</td>
<td>Medium</td>
</tr>
<tr>
<td>Close to well-known notation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transformation tool</td>
<td>Execution time</td>
<td>Medium</td>
</tr>
<tr>
<td>Maximum capability</td>
<td>50,000 model elements (High)</td>
<td></td>
</tr>
<tr>
<td>Syntactic correctness</td>
<td>Medium</td>
<td></td>
</tr>
<tr>
<td>Correctness (Termination)</td>
<td>Medium</td>
<td></td>
</tr>
<tr>
<td>Correctness (Confluence)</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Effectiveness</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Fault tolerance</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Eclipse plug-ins</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Maturity</td>
<td>Medium</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.8: Evaluation criteria for GrGen solution

It is possible to include a transformation in a larger process by means of a `xgrs` script sequentially composing the rules of the transformations. Development effort for generating this case study was low and around hundred minute. In addition, OCL notations are used in the specification. In the usability survey four of five participants considered the specification clearly structured, and with a low or medium effort to understand. All respondents correctly identified the code used in the learnability test (no participant had prior knowledge of GrGen).
6.5. GRGEN.NET SOLUTION FOR CLASS DIAGRAM RESTRUCTURING
CASE STUDY

Figure 6.13: Call graph of GrGen solution

<table>
<thead>
<tr>
<th>Test case</th>
<th>Number of classes</th>
<th>Number of attributes</th>
<th>Execution time (ms)</th>
<th>Effectiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
<td>80</td>
<td>100% (non-optimal)</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>7</td>
<td>100</td>
<td>100% (optimal)</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>20</td>
<td>90</td>
<td>100% (optimal)</td>
</tr>
<tr>
<td>4</td>
<td>105</td>
<td>132</td>
<td>100</td>
<td>100% (optimal)</td>
</tr>
<tr>
<td>5</td>
<td>500</td>
<td>5000</td>
<td>472580</td>
<td>100% (optimal)</td>
</tr>
</tbody>
</table>

Table 6.9: Test case results for GrGen

Transformation tool properties

The GrGen.Net generates C# programs to implement the transformation. It provides efficient implementation with a high maximum capability. The modification and extension is at the level of specification rather than implementation. Table 6.9 gives the test results for GrGen, evaluated on the SHARE environment. Transformation generates effective result for the specified test cases. The maximum capacity results are shown in Table 6.10.

It is not possible to prove syntactic correctness using static verification support in GrGen. Nonetheless, it is simple to write GrGen assertions for the pre-condition constraints, and then to evaluate these in the target model to verify correctness. Inverse links to associations are set explicitly, unlike the implicit updating of inverse links used in UML-RSDS. The visual debugger in GrGen provides a convenient way for validation of a transformation. Termination of the solution is provided because each rule removes one property instance. However, there is no proof technique to show this. There is only the possibility of application of one single rule at a time.
Table 6.10: Maximum capability test results (GrGen)

<table>
<thead>
<tr>
<th>Test case</th>
<th>Number of classes</th>
<th>Number of attributes</th>
<th>Execution time</th>
<th>Effectiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>2*100</td>
<td>400</td>
<td>400</td>
<td>1192ms</td>
<td>100%</td>
</tr>
<tr>
<td>2*200</td>
<td>800</td>
<td>800</td>
<td>8332ms</td>
<td>100%</td>
</tr>
<tr>
<td>2*1000</td>
<td>4000</td>
<td>4000</td>
<td>123s</td>
<td>100%</td>
</tr>
<tr>
<td>2*10000</td>
<td>40000</td>
<td>40000</td>
<td>464s</td>
<td>100%</td>
</tr>
</tbody>
</table>

Assumptions of a transformation can be checked in the source model. The error message is given in GrGen if the rules are erroneous according to the specification. It includes the location of the error in the rule specification file.

GrGen.NET has been publicly available since 2003 and about fifty transformation case studies have been implemented using it. In addition, interoperability with Eclipse is provided.

### 6.6 Kermeta solution for class diagram restructuring case study

This section describes the Kermeta solution to the case study. The solution is developed by Suresh Pilly from National Institute for Research in Computer Science and Control. The implementation of Kermeta for performing this case study follows different procedure from UML-RSDS and GrGen.

In this solution the process of transformation is initialised by leaf nodes. The leaf nodes are classes that are not specialised by any other classes. In the implementation below the set of leaf nodes (leafNodes) are selected to be processed by the rules.

```kotlin
1 var leafNodes : Set<String> init Set<String>.new
2 root.Elem.select{ e | e.isInstanceOf(Class) }.each{ e |
3     var cl : Class
4     cl != e
5     if cl.specialisation.size() == 0 then
6         leafNodes.add(cl.name)
7     end
8 }
```

The code segment below, processes the parents for the sets of leaf nodes which has been selected in the previous step and then mark that node as being processed. It implies that all the nodes below such nodes have also been processed.

```kotlin
1 var procNodes : Set<String> init Set<String>.new
```
6.6. KERMETA SOLUTION FOR CLASS DIAGRAM RESTRUCTURING
CASE STUDY

The implementation of first rule in Kermeta is shown below. In order to find the common attribute in the classes a set of properties is generated (line 5). It is initialized with the ownedAttributes of one of the classes from the list that is marked as processed in previous step. Following that, the set is compared with the properties of next unprocessed class. The result of comparison is a set that contains the common properties of two compared classes. Continuously, the final set contains the common attributes between all the classes. The termination condition is to have an empty base set which indicates no common properties (line 16). Having found the sets of common properties in the next step they are removed from subclasses and added to the superclass (lines 13,14).

```
1 var propSet : Set<Set<Property>> init Set<Set<Property>>.new
2 c.specialisation.each{ sc |
3 var sc : Class
4 var prop : Set<Property> init Set<Property>.new
5 sc.ownedAttribute.each{ p | prop.add(p) }
6 propSet.add(prop)
8 }
9 var iset : Set<Property>
10 iset := Process(propSet)
11 if iset.size() > 0 then
12 stdio.writeln("Common Properties")
13 addPropertiesSuper(c,iset)
14 remPropertiesChildren(c,iset)
15 else
16 stdio.writeln("No Common Properties")
17 end
```

The full listings of this solution can be found in the SHARE site [132] and appendix.
6.6. KERMETA SOLUTION FOR CLASS DIAGRAM RESTRUCTURING

CASE STUDY

6.6.1 Evaluation properties for Kermeta

Table 6.11 summarises the evaluated attributes of the transformation language and tools for the Kermeta solution. Although the complexity and size of the Kermeta solu-

<table>
<thead>
<tr>
<th>Transformation language</th>
<th>Abstraction level</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size: LOC</td>
<td>653 lines (High)</td>
</tr>
<tr>
<td>Syntactic complexity</td>
<td>260</td>
<td></td>
</tr>
<tr>
<td>Structural complexity</td>
<td>54 calls,</td>
<td></td>
</tr>
</tbody>
</table>
|                         | 0 recursive calls,
|                         | max call depth = 4
| Modularity              | factorisation: 53%|
|                         | coupling = 44%, cohesion = 56% |
| Modularity total        | Low |
| Extensibility:          | Medium |
| extension to UML 2 class diagrams |
| Interoperability:       | High |
| inclusion in a transformation process |
| Development effort      | 440 min (High) |
| Close to well-known notation | High |
| Transformation tool     | Execution time   | Medium |
|                         | Maximum capability| 10,000 model elements (Medium) |
|                         | Syntactic correctness | Medium |
|                         | Correctness (Termination) | Medium |
|                         | Correctness (Confluence) | Medium |
|                         | Effectiveness     | High |
|                         | Fault tolerance   | High |
|                         | Eclipse plug-ins  | Comprehensive |
|                         | Maturity          | High |

Table 6.11: Evaluation criteria for Kermeta solution

Although the complexity and size of the Kermeta solution is significantly higher than the UML-RSDS and GrGen solutions, the language does provide a more precise control over rule applications.

Transformation language properties

Kermeta is at a lower level of abstraction than the other languages considered in this paper. This has negative consequences for the size (653 lines of code) and complex-
ity of the solution, and also leads to a higher development effort. However greater power is available to the specifier, and more detailed control over the transformation processing. This results in higher effectiveness than the declarative solutions, and imposes a specific deterministic processing to achieve confluence.

The number of operations and features are 260 in this implementation. In total, there are fifty-four calls in the transformation and the maximum depth is four. The negative point of this solution is to have recursive calls in the specification. It can be seen that $ProcessRule_1$ and $ProcessRule_2$ are recursively defined. Moreover, the factorization is 53% and implementation is encapsulated inside eight modules. The modularization is provided by expert developer. The percentage of cohesion is 56% and coupling is 44% based on the specified encapsulation. Figure 6.14 shows the calling dependencies between the classes and operations of this solution. The dashed...
6.6. KERMETA SOLUTION FOR CLASS DIAGRAM RESTRUCTURING

CASE STUDY

Table 6.12: Test case results for Kermeta

<table>
<thead>
<tr>
<th>Test case</th>
<th>Number of classes</th>
<th>Number of attributes</th>
<th>Execution time (ms)</th>
<th>Effectiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>100% (optimal)</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>7</td>
<td>10</td>
<td>100% (optimal)</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>20</td>
<td>10</td>
<td>100% (optimal)</td>
</tr>
<tr>
<td>4</td>
<td>105</td>
<td>132</td>
<td>20</td>
<td>100% (optimal)</td>
</tr>
<tr>
<td>5</td>
<td>500</td>
<td>5000</td>
<td>out of memory</td>
<td>-</td>
</tr>
</tbody>
</table>

lines indicate internal dependencies (cohesion) of modules, while solid lines indicate external dependencies (coupling). 56% of the calls are internal to modules. There are seventeen expressions of size seven or more, of which nine are non-unique, giving a factorisation value of 53%.

Extension to a larger subset of UML 2 would be facilitated by this factorisation; however a substantial amount of new transformation code would be required. The transformation can be effectively embedded in a larger quality improvement process. This is due to the flexible imperative nature of the tool. Development effort for generating this case study was high and around 440 minutes. The OCL notations are used for the specification. Furthermore, interoperability with other MOF-based tools is possible.

In the usability survey, three of five respondents considered that there was no clear structure to the specification, four of five respondents found that a large or very large effort was required to understand it. Three of five respondents correctly identified the code segment for the learnability test.

Transformation tool properties

Table 6.12 shows the results of the four main test cases using Kermeta on the SHARE environment. It generates an effective result for the selected test cases. In addition, the results of the maximum capability tests which could be executed for Kermeta on the SHARE environment is given in table 6.13. The construction of large sets of elements in the applications of rule 3 “create root class” appears to be the main source of execution costs and memory resource usage. Compared to the other solutions, more memory is consumed in the processing of sets of elements due to the overheads involved in enforcing a precise iteration order over the class hierarchy of
a model. Application of rule 3 was generally the most expensive component of the

table.

<table>
<thead>
<tr>
<th>Test case</th>
<th>Number of classes</th>
<th>Number of attributes</th>
<th>Execution time</th>
<th>Effectiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>2*100</td>
<td>400</td>
<td>400</td>
<td>13s</td>
<td>100% (optimal)</td>
</tr>
<tr>
<td>2*200</td>
<td>800</td>
<td>800</td>
<td>100s</td>
<td>100% (optimal)</td>
</tr>
<tr>
<td>2*500</td>
<td>2000</td>
<td>2000</td>
<td>1500s</td>
<td>100% (optimal)</td>
</tr>
<tr>
<td>2*1000</td>
<td>4000</td>
<td>4000</td>
<td>12600s</td>
<td>100% (optimal)</td>
</tr>
</tbody>
</table>

Table 6.13: Maximum capability test results

execution. Compared to UML-RSDS the execution time on the test cases does not vary substantially influenced by the test case size.

The explicit nature of the algorithms for the Kermeta solution means that it is relatively direct to establish termination of this solution: the elements involved in each application of a rule are assembled using bounded loops, and the top-level control over the application of the rules is also carried out by bounded loops. However, the use of recursion for the top-level control of the application of the rules makes proof of termination less clear than for UML-RSDS. The fact that updates to the model are carried out in many places in the code also hinders proof of syntactic correctness. The precise control over the ordering of rule applications makes the confluence clear; however there is no proof for this.

Kermeta provides an assertion capability, so that assumptions can be checked at runtime. The robustness is high for this approach and it shows the error and the exact line that it happened. Kermeta has good support for interconnection with Eclipse. It has been available since 2005, and it has an extensive history of use, with over forty published case studies (e.g., [108, 105]).

6.7 ATL solution for class diagram restructuring case study

This section describes the ATL solution to the case study. The solution is implemented in ATL 3.2 and runs in the latest version of Eclipse Indigo 3.7. The solution is developed by Dr Javier Troya Castilla from Malaga University. Ideally, this case study should be defined in the refining mode of ATL. The current refining mode
expresses the transformation by means of matched rules and not lazy rules or
imperative sections. Furthermore, it is not trivial to modify or delete specific elements
in this mode. For instance, the existing refining mode redefines source elements and
presents them in the target model. However, it is not clear how to add new elements
in the target model without modifying the source elements. In order to show the
limitation of ATL refining mode I provide two examples by using the metamodel of
the case study in this chapter. In the first example the aim is to create a new class
from every existing class in our model and extend its name. The following matched
rule is written:

```
1 rule NewClass{
2 from
3 c : ClassDiagram ! Class
to
4 c2 : ClassDiagram ! Class(
5 name <- c.name + ' extended name'
6 )
7 )
8 }
```

This rule does not add a new class, but redefines the one in the left hand side of
the rule. The current version of refining mode does not provide a mechanism to add
a new class in the target side.

Furthermore, deletion of elements is limited in refining mode. The keyword drop
is used to delete the elements, however it use is very restricted. The element can be
deleted in a rule if deleting the element is happened in all the matched rule, and not
one individual element. Alternative example is provided to show this limitation. In
the example below the aim is to delet all the classes of the model. The following
matched rule is written:

```
1 rule DeleteClass{
2 from
3 c : ClassDiagram ! Class
to
4 drop
5 )
6 }
```

This rule deletes all the classes and its incoming and outgoing references. However
this is not possible to add or modify some elements in a rule and then delete the
elements in the source pattern of the rule. Moreover, it is not possible for the rule to
match more than one element and then delete one of these elements.

In general the ATL’s concrete and abstract syntax still needs many modifications
and improvements in order to provide a powerful refining mode. In [144], problems
and conflicts that can arise by using refining mode is explained. Therefore, due to the
6.7. ATL SOLUTION FOR CLASS DIAGRAM RESTRUCTURING CASE STUDY

limitations that exist in the refining mode, the solution to this case study is instead given in the default mode.

The ATL solution for this case study provides separate rules to handle properties of subclasses and superclasses. The part of this solution is explained below. The CopyPropertiesNonSubclasses matched rule, copy all the properties of classes that are not a subclass.

```plaintext
1 -- This rule copies properties (all of them) from classes which are not subclasses.
2 rule CopyPropertiesNonSubclasses{
3     from p : ClassDiagram!Property,
4         c : ClassDiagram!Class(c.OwnedAttribute -> includes(p) and
5             c.generalisation ->size()=0)
6     to pOut : ClassDiagram!Property{
7         name <- p.name,
8         type <- p.type
9     }
10 }
```

This rule checks that the class has the property (line 5) and its belonging class is not a subclass (line 6). It then maps the properties (line 8) with the same name (line 9) and type (line 10) in the target model. The next rule (CopyPropertiesSubclasses) maps the properties of classes which are not shared between subclasses. The corresponding implementation is presented below.

```plaintext
1 -- This rule copies those properties belonging to subclasses which are not repeated in all
2 -- the subclasses of a class.
3 rule CopyPropertiesSubclasses{
4     from p : ClassDiagram!Property,
5         c : ClassDiagram!Class
6         cSup : ClassDiagram!Class (c.OwnedAttribute -> includes(p) and c.generalisation ->
7             exists(g | g.general = cSup) and
8             cSup.specialisation -> exists (g | not g.specific.OwnedAttribute->exists
9                 (pr | pr.name = p.name)))
10     to pOut : ClassDiagram!Property{
11         name <- p.name,
12         type <- p.type
13     }
14 }
```

This rule takes the property, its belonging class and its superclass (lines 5, 6, 7). Following that it checks whether the property is shared with a sibling of the subclass (line 8). Finally it maps those properties that are not shared with all siblings (lines 10-13).

The CopyClasses rule is defined to match all the classes. The restriction is defined for the properties of a class which are shared between all of its subclasses. The imperative code is added to this section due to the limitation of the declarative part.
6.7. ATL SOLUTION FOR CLASS DIAGRAM RESTRUCTURING CASE STUDY

1 This rule copies all the classes. Besides, it adds to those new classes those attributes
2 created by rules CopyPropertiesSubclasses and CopyPropertiesNonSubclasses, when it
3 corresponds.
4
5 rule CopyClasses{
6  from
7    c : ClassDiagram ! Class
8    to
9    cOut : ClassDiagram ! Class
10       name <- c.name,
11       generalisation <- c.generalisation,
12       specialisation <- c.specialisation
13  }
14
15 do{
16   if (c.generalisation -> size() = 0){
17     cOut.OwnedAttribute <- c.OwnedAttribute;
18   }
19
20   else{
21     thisModule.cSup <- c.generalisation->first().general; -- We suppose
22                      there is not multiple inheritance
23     for (p in c.OwnedAttribute){
24       if (thisModule.cSup.specialisation -> exists (g | not g.specifc
25                          .OwnedAttribute->exists( pr | pr.name = p.name))){
26         cOut.OwnedAttribute <- cOut.OwnedAttribute -> append(p);
27       }
28     }
29   }
30
31   if (c.specialisation -> size() >0){
32     for (p in c.specialisation->first().specific.OwnedAttribute){
33       if (c.specialisation -> forAll(g | g.specifc.OwnedAttribute ->
34                          exists (pr | pr.name = p.name and pr.type = p.type))){
35         -- A call to lazy rule CreateProperty is made
36         cOut.OwnedAttribute <- cOut.OwnedAttribute -> append(
37                     thisModule.CreateProperty(p));
38       }
39     }
40   }
41
42   if (c is not a subclass, all the attributes are copied by rule CopyPropertiesNonSubClasses,
43   so we add them here
44   }
45
46   if (c is a subclass, those attributes not repeated in its siblings are copied by
47   rule CopyPropertiesSubclasses, so we add them here
48   }
49
50   if (c is a superclass, we create a new attribute in it if all its subclasses
51   contain an attribute with the same name and type
52   }
53
54   }
55
56 }
57
58 }
59
60 The first part of the code maps the superclasses and their belonging attributes
61 (lines 15-16). The internal trace is then checked by ATL engine for the properties
62 created by the rule CopyPropertiesNonSubClasses. In the next part, subclasses and
63 their attributes that are not repeated in the sibling classes are mapped (lines 20-26).
64 Finally, the subclasses that share the common attribute between all the sibling classes
65 are checked (lines 29-36). In this case, a new property is created by using alternative
66 lazy rule CreateProperty. The lazy rule creates a new property with the same name
67 and type. The full listings of this solution can be found on the SHARE site [132] and
68 the appendix C of this thesis. ATL does not provide the complete solution to the
69 case study due to the existing limitation.
### 6.7.1 Evaluation properties of ATL

The properties of the ATL default mode solution for the first rule are shown in table 6.14. In order to have appropriate comparison for the complete transformation in ATL, the result of measurement for size, complexity and development time for the first rule is scaled up by a factor of 3. Scaling up the values is not ideal, but provides the rough version of solution.

<table>
<thead>
<tr>
<th>Transformation language</th>
<th>Abstraction level</th>
<th>Medium 81 (Medium) 243 (Medium)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size: LOC</td>
<td>81 (Medium)</td>
</tr>
<tr>
<td></td>
<td>Estimated size (completed solution)</td>
<td>243 (Medium)</td>
</tr>
<tr>
<td>syntactic complexity</td>
<td></td>
<td>144</td>
</tr>
<tr>
<td>Structural complexity</td>
<td></td>
<td>6 calls, 0 recursive calls, max call depth = 2</td>
</tr>
<tr>
<td>Complexity total for rule 1</td>
<td></td>
<td>152 (Medium)</td>
</tr>
<tr>
<td>Estimated complexity (completed)</td>
<td></td>
<td>456 (Medium)</td>
</tr>
<tr>
<td>Modularity</td>
<td></td>
<td>factorisation: 72%, cohesion: 100%, coupling 0%</td>
</tr>
<tr>
<td>Modularity total</td>
<td></td>
<td>High</td>
</tr>
<tr>
<td>Extensibility:</td>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>extension to UML 2 class diagrams</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interoperability:</td>
<td></td>
<td>High</td>
</tr>
<tr>
<td>inclusion in a transformation process</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Development effort</td>
<td>150 min (Medium)</td>
<td></td>
</tr>
<tr>
<td>Estimated development time (completed)</td>
<td>450 min (High)</td>
<td></td>
</tr>
<tr>
<td>Close to well-known notation</td>
<td></td>
<td>High</td>
</tr>
<tr>
<td>Transformation tool</td>
<td></td>
<td>High</td>
</tr>
<tr>
<td>Execution time</td>
<td>5,000 model elements (Low)</td>
<td></td>
</tr>
<tr>
<td>Maximum capability</td>
<td>Medium</td>
<td></td>
</tr>
<tr>
<td>Syntactic correctness</td>
<td>Medium</td>
<td></td>
</tr>
<tr>
<td>Correctness (Termination)</td>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>Correctness (Confluence)</td>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>Effectiveness</td>
<td></td>
<td>High</td>
</tr>
<tr>
<td>Fault tolerance</td>
<td></td>
<td>High</td>
</tr>
<tr>
<td>Eclipse plug-ins</td>
<td></td>
<td>Comprehensive</td>
</tr>
<tr>
<td>Maturity</td>
<td></td>
<td>High</td>
</tr>
</tbody>
</table>

Table 6.14: Evaluation criteria for ATL solution
Transformation language properties

The declarative parts of the ATL language are at a high level of abstraction; however this type of problem requires the use of imperative features, as in the solution for the first rule presented here. The specification is relatively similar to the informal description of the transformation, though a significant difference is the separation of the copying of unchanged parts of the model (which is implicit in the rule descriptions) from the explicit modifications needed (promotion of the duplicate copies of an attribute). The latter is performed in the CopyClasses rule. Size of the incomplete transformation is eighty-one in ATL. If the size measure scaled up by a factor of 3 then the estimated size for complete solution would be 243 (81*3). The need for explicit copying rules leads to a high syntactic complexity, as do the imperative code blocks in CopyClasses. Structural complexity is eight since there are six invocations of rules by other rules within the specification (five implicit calls and one explicit), and a maximum call depth of two. Figure 6.15 shows the call graph of the first rule in ATL. The complexity of the complete solution in this approach is estimated as 456. There are thirty-two expressions of complexity seven or above, of which nine are duplicated, so the percentage of unique expressions of complexity seven or above is 72%. The specification is encapsulated inside one module.

Extension to full UML class diagrams could be carried out in principle, with considerable additional complexity to the rules, and with additional rules. The

---

5At least one copy rule for every entity type in the metamodel – there are over thirty such entities
6.7. ATL SOLUTION FOR CLASS DIAGRAM RESTRUCTURING CASE STUDY

<table>
<thead>
<tr>
<th>Test case</th>
<th>Number of classes</th>
<th>Number of attributes</th>
<th>Execution time (ms)</th>
<th>Effectiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7</td>
<td>7</td>
<td>84</td>
<td>40% (non-optimal)</td>
</tr>
</tbody>
</table>

Table 6.15: Test case results for ATL

<table>
<thead>
<tr>
<th>Test case</th>
<th>Number of classes</th>
<th>Number of attributes</th>
<th>Execution time</th>
<th>Effectiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>2*100</td>
<td>400</td>
<td>400</td>
<td>1142s</td>
<td>40%</td>
</tr>
<tr>
<td>2*200</td>
<td>800</td>
<td>800</td>
<td>10888s</td>
<td>40%</td>
</tr>
<tr>
<td>2*500</td>
<td>2000</td>
<td>2000</td>
<td>150999s</td>
<td>40%</td>
</tr>
<tr>
<td>2*1000</td>
<td>4000</td>
<td>4000</td>
<td>173292s</td>
<td>40%</td>
</tr>
<tr>
<td>2*5000</td>
<td>20000</td>
<td>20000</td>
<td>147978s</td>
<td>40%</td>
</tr>
</tbody>
</table>

Table 6.16: Maximum capability test results

Copy rules for generalisations and properties cannot be reused for rule 2 and 3 of the problem. This is because different copying criteria are necessary for these rules, and hence this leads to a multiplicity of variants of copying rules. One ATL transformation can be sequentially composed with another (the transformations being defined in separate modules) by means of a Java script.

The development time of this case study with ATL was around 150 minutes. It is predicted that complete solution to the “class diagram restructuring” case study in ATL will take around 450 minutes. The use of standard OCL notation within rules assists in the interoperability and comprehensibility of ATL specifications.

**Transformation tool properties**

Table 6.15 shows the test case results for ATL on the SHARE environment. Only test case 2 was attempted. All possible applications of the first rule were performed successfully. The maximum capability tests were carried out as shown in Table 6.16. Similarly the applications of the first rule were performed in each case. While the ATL tools were able to execute all of the capacity test cases, up to size 50000 elements, only the first is (with an execution time of nineteen minutes) completed in a reasonable time.

Proof of syntactic correctness is also complicated by the copying strategy, e.g., types.
it is not clear if CopyClasses can introduce duplicated attributes of a class or not. Furthermore, there is no formal tool support for verification. Some initial work on ATL verification using SMT provers has been carried out, for simple declarative ATL transformations in default mode [27]. The ATL approach used for the first rule would suffer potentially from the same problems of lack of confluence and failure to achieve 100% effectiveness. There is no control over the order of applications of rules in ATL specification. However, termination of the solution is clear because of the linear iterations of the rules over their source domains.

ATL provides runtime checking for rule conflicts, and gives error messages indicating the specification lines responsible, if there is a runtime error. ATL is a relatively mature language, it was first available in 2003, and has over hundred published case studies [8]. In addition the complete interoperability with Eclipse is provided.

6.8 QVT-Relational solution for class diagram restructuring case study

This section describes the application of the QVT-R language and Medini QVT tools to the class diagram restructuring case study. It is complex to perform this task in QVT-R because of the lack of default copy rules. Such rules can be defined explicitly, but this is not trivial for cases where large parts of a model are copied.

Rule 1

This rule is implemented by identifying those Generalization objects such that the linked classes of the object satisfy the first rule’s conditions. In order to generate the list of such Generalizations, we declare a top relation rule called TakeGeneralization. This rule applies to every instance of Generalization. The when clause of this rule declares that the general end (superclass) has more than one subclass, where sub-classes have attributes with the same name and type. In the where clause of this rule another relation CheckRule1 is called which has the Generalization instance as its argument.

```plaintext
1 transformation QualityImprovement (source : KCL, target : KCL)
2 {
3   top relation TakeGeneralization {
4     checkonly domain source sourcegen : KCL::Generalization {
5       general end (superclass) has more than one subclass, where sub-
6       classes have attributes with the same name and type. In the where clause of this rule another relation CheckRule1 is called which has the Generalization instance as its argument.
```
6.8. QVT-RELATIONAL SOLUTION FOR CLASS DIAGRAM
RESTRUCTURING CASE STUDY

It is required to prioritise Rule 1 over Rule 2. This can be ensured by using CheckRule1, which specifies that no sibling class exists without a copy of the property.

It means that all of the subclasses have an attribute with the same name and type.

If the conditions of the first rule are satisfied, the corresponding Generalization is copied. The CopyGeneralisation rule is a non-top relation and only matches its arguments in the source and target model [47].

The next step is to assign a superclass to the selected Generalization. The TakeSuperClass relation takes all the classes from the input model and in the when clause of the relation selects the classes that have a specialisation element which has already been copied by the CopyGeneralisation rule. Furthermore, this rule performs the main task of moving the property from the subclasses to the superclass. This happens by accessing the property of the subclass in the checkonly domain of the relation and then adding it into the ownedAttributes of the superclass. In order to copy the name of the superclass the relation CopyClass is called in the where clause of this rule.
6.8. QVT-RELATIONAL SOLUTION FOR CLASS DIAGRAM
RESTRUCTURING CASE STUDY

Rule 2

Similarly, in the second rule it is required to copy the set of Generalization which are of interest to us. However, the copying only takes place if the condition of the second rule is true. TakeGeneralizationForRule2 collects all the relevant Generalization objects, in the when clause while checking the condition of the second rule. It checks whether the superclass has more than two subclasses and then checks if more than one subclass has a property with the same name and type. Satisfaction of these conditions for some Generalization is not sufficient for the applicability of the second Rule. In addition, it is required to check that there are some subclasses that all have properties with a different name or type to the identified properties. This condition is checked in CheckRule2:

In the when clause of CheckRule2 rule we check if there exists at least one class which has different properties to the properties of other classes. Therefore, satisfaction of this condition enables the copying of the Generalization for the purpose of the second rule. In addition, coping the Generalization of this rule allows us to generate the subclass and to assign it to the generalizations relation.
6.8. QVT-RELATIONAL SOLUTION FOR CLASS DIAGRAM
RESTRUCTURING CASE STUDY

Similar to the first rule we define a non-top marker relation of \textit{CopyGeneralisationForRule2} in order to classify \textit{Generalisation} objects:

\begin{verbatim}
relation CopyGeneralisationForRule2 {
  check only domain source sourcegen \in KCL::Generalisation {};
  enforce domain target targetgen \in KCL::Generalisation {};
}
\end{verbatim}

The \textit{TakesuperClassForRule2} assigns a superclass to the \textit{Generalisation} objects. This rule then checks if the \textit{specialization} of the \textit{class} has been copied in the list of \textit{Generalisation} of interest for the second rule. If the \textit{class} has any \textit{specialisation} which satisfies this condition then the \textit{property} with the same \textit{name} and \textit{type} is moved to this \textit{class}. Then the \textit{class} is assigned as a superclass to the other end of the \textit{specialisation}. The process of transformation up to this point is similar to the process of the first rule; however in the second rule we rename the superclass to \textit{NEW class}. This will happen through the \textit{Generatenewclass} rule.

\begin{verbatim}
relation Generatenewclass {
  n : String;
  check only domain source s \in KCL::Class {
    name = n,
    specialization = spsourceuper \in KCL::Generalisation {
      specific = spe \in KCL::Class {
        ownedAttribute = prop \in KCL::Property {}}
    }
  }
  enforce domain target t :KCL::Class {
    specialization = sptargetsuper \in KCL::Generalisation {},
    ownedAttribute = prop \in KCL::Property {}};
  where \{ CopyGeneralisationForRule2(spsourceuper, sptargetsuper); \}
}
\end{verbatim}

In the \textit{where} clause of \textit{Generatenewclass} rule, the \textit{GenerateSuperSuperClass} rule is called which is responsible for assigning the original superclass to the new class:

\begin{verbatim}
relation GenerateSuperSuperClass {
  n : String;
  check only domain source s \in KCL::Class {
    name = n;
  }
  enforce domain target t :KCL::Class {
    name = n + 'NEW';
  }
  where \{ GenerateSuperSuperClass(s, t); \}
}
\end{verbatim}

In this rule the original superclass is renamed as \textit{NEW} and we then generate a new class for the main superclass. This strategy has the advantage of preserving the \textit{Generalisation} relations.
6.8. QVT-RELATIONAL SOLUTION FOR CLASS DIAGRAM
RESTRUCTURING CASE STUDY

6.8.1 Evaluation properties of QVT-Relational

Table 6.17 summarises the evaluated attributes of the transformation language and tools for the QVT-R solution. The QVT-R solution contains the first rule and partially the second rule. The measure of size, complexity and development effort is scaled up by a factor of 1.5 in this solution to have clear understanding of the completed solution.

Transformation language properties

The QVT-R language is at a high level of abstraction. However, the nature of the problem such as copying, creating, deletion and modification of model elements, prevents a simple declarative specification of the transformation in QVT-R. It should be noted that none of the languages in this survey have achieved such a description, even in the case of UML-RSDS the constraints have a strongly procedural flavour and cannot be interpreted as pure logical postconditions of the transformation. The transformation is of moderate size and complexity, and has been modularised by factorisation of conceptually distinct predicates, and by sequential decomposition, into a selection phase followed by a restructuring phase. There are eleven calls in the specification and maximum call depth is two. If the size and complexity of QVT-R were scaled up by 1.5, then a complete solution would have estimated size of 125 lines and complexity of 284. These results does not alter their classification on the three-point scale. Figure 6.16 shows the call graph of this solution. There are forty-six expressions of size seven or more, of which twenty-eight are unique, giving a factorisation of 61%. If the transformation is modularised into two modules according to Figure 6.16, then there is one inter-module call (from CheckRule2 to CopyClass) and ten intra-module calls, ie., the cohesion is 91%.

QVT is a standard language in model transformation the notation is based closely
upon UML and OCL. Because of the limitation that exists in performing refinement tasks in QVT-R it seems unlikely that the approach could be easily extended to larger class diagram metamodels. The fact that QVT-R needs a distinguished “top relation” means that it can not clearly possible to compose the relations in this language. Development effort for generating the first rule and part of second rule in the QVT-R approach was medium, and needed research into specialised patterns for defining copy transformations in QVT-R. If the effort for QVT-R is scaled up, then it has a value of 420, which is high according to the classification of development effort.

Table 6.17: Evaluation criteria for QVT-R solution
6.8. QVT-RELATIONAL SOLUTION FOR CLASS DIAGRAM
RESTRUCTURING CASE STUDY

Figure 6.16: Call graph of QVT-R solution

<table>
<thead>
<tr>
<th>Test case</th>
<th>Number of classes</th>
<th>Number of attributes</th>
<th>Execution time (ms)</th>
<th>Effectiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7</td>
<td>7</td>
<td>109</td>
<td>40% (non-optimal)</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>20</td>
<td>159</td>
<td>55% (non-optimal)</td>
</tr>
<tr>
<td>4</td>
<td>105</td>
<td>132</td>
<td>297</td>
<td>61% (non-optimal)</td>
</tr>
<tr>
<td>5</td>
<td>500</td>
<td>5000</td>
<td>Out of memory</td>
<td>–</td>
</tr>
</tbody>
</table>

Table 6.18: Test case results for QVT-R

In the usability survey three respondents considered the structure to be clear, but all considered the understandability effort to be large or very large.

6.8.2 Transformation tool properties

Table 6.18 shows the test case’s results for QVT-R on the SHARE environment. For test case 1 there should be one application of the second rule. However, in the implementation all a and b properties are moved to the same new superclass. In test case 2 the implementation performs this testcase correctly for the first rule (not the third rule) and the execution time is 109 ms. Test case 3 takes 159 ms but is not completely processed. For test case 4 the implementation performs all the application of the first rule, which can be done in the first iteration and it performs the second rule partially. The time taken is 297 ms. The maximum capability tests were carried out as shown in Table 6.19. Processing of larger models was not possible and an out of memory error was produced. The correctness of the transformation cannot be established because of some remaining problems:
6.8. QVT-RELATIONAL SOLUTION FOR CLASS DIAGRAM RESTRUCTURING CASE STUDY

<table>
<thead>
<tr>
<th>Test case</th>
<th>Number of classes</th>
<th>Number of attributes</th>
<th>Execution time</th>
<th>Effectiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>2*100</td>
<td>400</td>
<td>400</td>
<td>1610ms</td>
<td>40%</td>
</tr>
<tr>
<td>2*200</td>
<td>800</td>
<td>800</td>
<td>2359ms</td>
<td>40%</td>
</tr>
<tr>
<td>2*500</td>
<td>2000</td>
<td>2000</td>
<td>38s</td>
<td>40%</td>
</tr>
</tbody>
</table>

Table 6.19: Maximum capability test results

- The transformation removes copies of a property which has clones from subclasses but also it adds all copies into the superclass (it does not keep one copy and destroy the other copies). Therefore, we need additionally to remove duplicate copies from the superclass.

- It is not possible to prioritize the rules in QVT even though the conditions for the first rule and the second rule have been made disjoint, both transformations could potentially apply at different locations in a model at the same time.

- The transformation is not complete. The QVT relational language does not support any technique for the third rule as there is no connection in the model between different root classes.

- Further copy rules are needed to map parts of the source model which are not matched by the restructuring rules. The scope of what is changing is not clear in this approach.

- It is difficult to compose the transformation, because of the idea of top-level versus non-top-level rules.

As with the other declarative and hybrid language solutions, the confluence of the QVT-R solution cannot be established. Termination should follow since no recursion is used; however the two-phased copying approach means that there is no simple variant which is reduced by every rule application.

The Medini tool [101] provides run-time exception handling: errors in processing result in termination of the transformation and an error message giving the specification lines where the error occurred. QVT-R is defined by an international standard [117]. Version 1.0 was released in 2008, and there are twenty published case studies. Medini QVT has been available since 2007.
6.9. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th>Approach</th>
<th>Total number of lines of code</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML-RSDS</td>
<td>24</td>
<td>Low</td>
</tr>
<tr>
<td>GrGen</td>
<td>102</td>
<td>Medium</td>
</tr>
<tr>
<td>Kermeta</td>
<td>653</td>
<td>High</td>
</tr>
<tr>
<td>ATL</td>
<td>81 (completed solution = 243)</td>
<td>Medium</td>
</tr>
<tr>
<td>QVT-R</td>
<td>83 (completed solution = 124)</td>
<td>Medium</td>
</tr>
</tbody>
</table>

Table 6.20: Size comparison for quality improvement case study

6.9 Comparison of approaches

In this section comparison of the relative features of each transformation approach is given. The comparison is based on the quantitative metrics of size, complexity, modularity, development effort and correctness. We also summarise the results of the usability survey, and discuss what correlations exist between different measures for the solutions. The correlations between metrics including size, complexity, development effort, modularity is calculated. Furthermore, the correlation of usability character and these metrics are also presented. The correlation is only provided for this case study as other ones are quite small cases. Finally, the ranking of the approaches according to the ISO 9126 quality characteristics is given.

In order to analyse the correlation between different measures for the solutions, the Pearson correlation coefficient is used [135]. It calculates the linear relationship between +1 and -1 variables. A positive relationship between the metrics and quality attributes implies a positive correlation, while a negative relation indicates a negative correlation. The closer the coefficient is to either -1 or 1, the stronger the correlation between the variables is. The Pearson correlation coefficient is implemented in UML-RSDS [94]. In this thesis the correlation of metrics and quality attributes of the quality improvement case study is investigated. The other case studies are quite small and will not produce appropriate result in terms of correlation.

Size

A simple size metric is the number of lines of code in a transformation specification. Table 6.20 shows the size measures for the solutions. The declarative approaches have generally lower size metrics than the hybrid or imperative approaches. As in the
following sections, it should be noted that the ATL and QVT solutions only covered one and two of the three rules of the problem, respectively. If these solutions were scaled up by a factor of 3 and 1.5, respectively, then a complete ATL solution would have an estimated size of 243 lines, and a complete QVT-R solution 124 lines. This does not change the classification of these solutions as medium size. Total number of lines of code in UML-RSDS is relatively lower than other approaches, it is because of the particular specification paradigm in this approach. Due to the imperative style of specification in Kermeta, the solution have substantial lines of code. The ranking of the complete solutions on the basis of size is: UML-RSDS; GrGen; QVT-R; ATL; Kermeta.

Complexity

Complexity of a transformation is influenced by the calling relationship between subparts of its specification. In order to understand a rule, it is also necessary for a reader or analyser of the specification to understand also the rules that it calls. On the other hand, recursive calls have an additional effect on complexity, since the reader must understand how the recursion progresses and terminates. It is therefore necessary that the total number of calls, recursive as well as non-recursive calls in a transformation should be counted. Depth of call chains also influences the complexity. The deeper the call chain, the harder it is to understand the transformation. This form of structural complexity of a specification or program can be measured by analysing its call graph. A call graph is a directed graph that represents calling relationships between subroutines in a program. In the call graph each node represents functions or rules, and edges represent calls between the nodes. The size of the call graph affects the complexity of the program. The greater the number of arcs in the call graph, the higher is the dependency between different parts of the program, and so the greater is the complexity.

Another form of complexity is due to the complexity of the expressions used within the specification: the greater the number of expression operators \(^6\) or references to metamodel features \(^7\), the more effort is required to comprehend and work with the specification. Table 6.21 compares the complexities of the approaches, based on the

---

\(^6\)such as =, :, →exists()

\(^7\)such as ownedAttribute, specific
syntactic complexities, the total number of calls, total number of recursive calls and depth of calls on the quality improvement case study. Depth1 is the maximum depth of call chains not involving recursive loops. Scaling up the complexities of the ATL and QVT-R solutions produces the figures of 456 and 285 respectively, which does not alter their classification on the three-point scale. Kermeta has significantly higher complexity than the more declarative approaches. It has eight recursive calls which have a substantial negative effect on the complexity. ATL is however in second place, due to the complex processing needed for this problem. UML-RSDS, whilst very concise in terms of lines of code, has in compensation a high density of complex expressions within its constraints, and is in 4rd place. The ranking of the complete solutions on the basis of complexity is: GrGen; UML-RSDS; QVT-R; ATL; Kermeta.

The Pearson correlation coefficient of size with complexity (using all five solutions) is 0.99.

Modularity

Modularity is considered both in terms of how factorised the specification is and how the proportion of calls within modules (cohesion) relates to the proportion of calls between modules (coupling). Table 6.22 compares the modularity of the specifications.

Only the Kermeta and QVT specifications were decomposed into modules on the basis of the transformation rules. Modules consisted of both the main rule processing for each of the required rules, together with auxiliary rules/operations for this processing. Modularity of Kermeta is classified as low in comparison to other approach. This relates to high density of calls among modules in this approach and therefore the low percentage of cohesion. All the notations are capable of increasing the factorisa-
6.9. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th>Approach</th>
<th>Factorisation</th>
<th>Coupling</th>
<th>Cohesion</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML-RSDS</td>
<td>68%</td>
<td>0</td>
<td>100%</td>
<td>Medium</td>
</tr>
<tr>
<td>GrGen</td>
<td>65%</td>
<td>0</td>
<td>100%</td>
<td>Medium</td>
</tr>
<tr>
<td>Kermeta</td>
<td>53%</td>
<td>44%</td>
<td>56%</td>
<td>Low</td>
</tr>
<tr>
<td>ATL</td>
<td>72%</td>
<td>0</td>
<td>100%</td>
<td>High</td>
</tr>
<tr>
<td>QVT-R</td>
<td>61%</td>
<td>9%</td>
<td>91%</td>
<td>Medium</td>
</tr>
</tbody>
</table>

Table 6.22: Modularity measures for quality improvement case study

<table>
<thead>
<tr>
<th></th>
<th>Rule 1 (min)</th>
<th>Rule 2 (min)</th>
<th>Rule 3 (min)</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML-RSDS</td>
<td>40</td>
<td>30</td>
<td>50</td>
<td>Medium</td>
</tr>
<tr>
<td>GrGen</td>
<td>40</td>
<td>30</td>
<td>30</td>
<td>Low</td>
</tr>
<tr>
<td>Kermeta</td>
<td>120</td>
<td>160</td>
<td>160</td>
<td>High</td>
</tr>
<tr>
<td>ATL</td>
<td>150</td>
<td></td>
<td></td>
<td>Medium</td>
</tr>
<tr>
<td>QVT-R</td>
<td>120</td>
<td>160</td>
<td></td>
<td>Medium</td>
</tr>
</tbody>
</table>

Table 6.23: Development effort for quality improvement case study

Unsurprisingly, the larger and more complex the specification, the more effort was required to develop it, with Kermeta. If the effort for ATL and QVT-R are scaled up, they have values of 450 and 420 respectively, placing them close to Kermeta in terms of effort. The ranking of the complete solutions on the basis of development effort is: GrGen; UML-RSDS; QVT-R; Kermeta; ATL.

The correlation of size with development effort, using all the solutions, is 0.58.

Development effort

The development effort of a transformation refers to the time spent on expressing each individual rule of the requirements into the specific transformation language. However, the time for understanding the problem or the specification notation is not considered here, nor is the time taken in testing. Table 6.23 shows the development effort time for each rule in the five different transformation languages.

Unsurprisingly, the larger and more complex the specification, the more effort was required to develop it, with Kermeta. If the effort for ATL and QVT-R are scaled up, they have values of 450 and 420 respectively, placing them close to Kermeta in terms of effort. The ranking of the complete solutions on the basis of development effort is: GrGen; UML-RSDS; QVT-R; Kermeta; ATL.

The correlation of size with development effort, using all the solutions, is 0.58.
6.9. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th>Approach</th>
<th>Syntactic</th>
<th>Termination</th>
<th>Confluence</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML-RSDS</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>Medium</td>
</tr>
<tr>
<td>GrGen</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>Medium</td>
</tr>
<tr>
<td>Kermeta</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>High</td>
</tr>
<tr>
<td>ATL</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>Medium</td>
</tr>
<tr>
<td>QVT-R</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>Low</td>
</tr>
</tbody>
</table>

Table 6.24: Correctness measures for quality improvement case study

The correlation of complexity with development effort, using all the solutions, is 0.67.

Correctness

Correctness was assessed by three separate 5-point measures of syntactic correctness \(^8\), termination \(^9\) and confluence \(^10\). Table 6.24 shows the results for each of these, considered as numeric values from 0 (None) to 4 (Comprehensive), and the resulting overall average score.

UML-RSDS supports proof of syntactic correctness by direct induction over transformation steps that pre-condition is preserved. Inverse links of associations are set automatically. Termination is proved by showing that Property.size is strictly decreased by each transformation step. However, the lack of detailed control over the transformation execution means that confluence cannot be proved (and indeed it fails simply re-ordering the elements in the input model file can produce non-isomorphic result models). For GrGen the results are similar, except that syntactic correctness is less clear because the specification notation is further removed from pure logic, with operation calls and sequencing in the rule definitions. Unlike UML-RSDS, there is no tool support for correctness proof. Again, confluence fails. Kermeta is able to enforce an optimal clone-removal strategy and achieve confluence in principle. Establishing syntactic correctness and termination is obstructed by the complex control flow and use of recursion. For ATL confluence fails, and the QVT-R solution has minor errors also in syntactic correctness. The ordering of the solutions with regard to correctness is: Kermeta; UML-RSDS; GrGen; ATL; QVT-R.

---

\(^8\) Ability to prove that the target models constructed satisfy the pre-condition constraints
\(^9\) Ability to prove termination
\(^10\) Ability to prove confluence or determinacy of the transformation
6.9. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th>Question</th>
<th>UML-RSDS</th>
<th>GrGen</th>
<th>Kermeta</th>
<th>ATL</th>
<th>QVT-R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expertise level (E)</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.1</td>
<td>0.8</td>
</tr>
<tr>
<td>Relate formal to informal (U1)</td>
<td>1.2</td>
<td>2.2</td>
<td>0.6</td>
<td>1.8</td>
<td>0.4</td>
</tr>
<tr>
<td>Understanding detail (U2)</td>
<td>1.6</td>
<td>2.4</td>
<td>3.2</td>
<td>2.6</td>
<td>1.0</td>
</tr>
<tr>
<td>Well structured (A1)</td>
<td>1.6</td>
<td>2.8</td>
<td>1.2</td>
<td>2.8</td>
<td>1.2</td>
</tr>
<tr>
<td>Attractive (A2)</td>
<td>2.8</td>
<td>1.2</td>
<td>0.6</td>
<td>2.0</td>
<td>2.4</td>
</tr>
<tr>
<td>Understanding effort (L1)</td>
<td>1.4</td>
<td>1.8</td>
<td>0.8</td>
<td>1.8</td>
<td>0.4</td>
</tr>
<tr>
<td>Gained knowledge (L2=U2 - E)</td>
<td>1.2</td>
<td>2.0</td>
<td>2.8</td>
<td>1.6</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Table 6.25: Summary of usability survey question responses for quality improvement case study

<table>
<thead>
<tr>
<th>Approach</th>
<th>Understandability</th>
<th>Attractiveness</th>
<th>Learnability</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML-RSDS</td>
<td>1.4 (Low)</td>
<td>2.2 (Medium)</td>
<td>1.3 (Low)</td>
</tr>
<tr>
<td>GrGen</td>
<td>2.3 (Medium)</td>
<td>2 (Medium)</td>
<td>1.9 (Medium)</td>
</tr>
<tr>
<td>Kermeta</td>
<td>1.9 (Medium)</td>
<td>0.9 (Low)</td>
<td>1.8 (Medium)</td>
</tr>
<tr>
<td>ATL</td>
<td>2.2 (Medium)</td>
<td>2.4 (Medium)</td>
<td>1.7 (Medium)</td>
</tr>
<tr>
<td>QVT-R</td>
<td>0.7 (Low)</td>
<td>1.8 (Medium)</td>
<td>0.3 (None)</td>
</tr>
</tbody>
</table>

Table 6.26: Usability summaries for quality improvement case study

**Usability factors**

Similar to the first case study, the usability factor is analysed by a survey of informatics professionals ranging from PhD students to senior academics. Table 6.25 summarises the results of the survey and Table 6.26 sums the factors for each usability characteristic and gives the qualitative summary for each approach. The overall ranking of approaches with regard to usability is therefore: ATL (6.3); GrGen (6.2); UML-RSDS (4.9); Kermeta (4.6); QVT-R (2.8).

This ordering is inversely correlated with the values for complexity and size and development effort, and positively correlated with modularity, as could be expected by analogy with similar results for programs [31].

The Pearson correlation coefficient of usability with complexity for Kermeta,
6.9. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th></th>
<th>Size</th>
<th>Complexity</th>
<th>Dev.Effort</th>
<th>Usability</th>
<th>Modularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>0.99</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complexity</td>
<td></td>
<td>0.58</td>
<td></td>
<td>-0.55</td>
<td>-0.88</td>
</tr>
<tr>
<td>Dev.Effort</td>
<td>0.58</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Usability</td>
<td>-0.55</td>
<td>-0.66</td>
<td></td>
<td>-0.68</td>
<td>-0.72</td>
</tr>
<tr>
<td>Modularity</td>
<td>-0.88</td>
<td>-0.90</td>
<td>-0.72</td>
<td>0.69</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.27: Correlation results of measures

UML-RSDS and GrGen is -0.663, indicating a moderately strong linear dependency.

The correlation coefficient of usability with size for Kermeta, UML-RSDS and GrGen is -0.55.

The correlation coefficient of usability with modularity is 0.69 for Kermeta, UML-RSDS, GrGen and ATL, indicating a positive linear correlation.

Correlation results

Table 6.27 summarises the correlation results we have obtained between the different measures of the solutions. These generally indicate that the expected relationships hold between these measures, and give some evidence of the validity and consistency of the selected measures.

6.9.1 Evaluation summaries

For each approach, we can complete a table of its quality characteristics based upon the measured values for its attributes. Different attribute measures could be combined using some scheme of numeric weighting, and likewise for the combination of subcharacteristics which contribute to a quality characteristic. Here we will use a simple count of the attribute values which fall into the top two categories of the five-point scale from the viewpoint of the subcharacteristic. This allows decision-makers to gain a quick overview of the merits of a particular approach, in the event of a close contest between two approaches, the detailed measurement values for the approaches can be compared.

Table 6.28 (Table 217) shows the summary characteristic values for UML-RSDS. As can be seen, this approach scores highly for Suitability, Accuracy, Efficiency and Maintainability but poorly for Reliability, Usability.
### 6.9. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Subcharacteristic</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Suitability</td>
<td>Abstraction level ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Size ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Effectiveness ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Development effort ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Execution time ✓</td>
</tr>
<tr>
<td>Accuracy</td>
<td></td>
<td>Correctness ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Completeness ✓</td>
</tr>
<tr>
<td>Interoperability</td>
<td></td>
<td>Embeddable in transformation process ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Close to well-known notation ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Eclipse plug-ins ×</td>
</tr>
<tr>
<td>Functionality</td>
<td>Compliance</td>
<td>Close to well-known notation ✓</td>
</tr>
<tr>
<td>Reliability</td>
<td>Maturity</td>
<td>History of use ×</td>
</tr>
<tr>
<td></td>
<td>Fault tolerance</td>
<td>Tolerance of false assumptions ×</td>
</tr>
<tr>
<td>Usability</td>
<td>Understandability</td>
<td>×</td>
</tr>
<tr>
<td></td>
<td>Learnability</td>
<td>×</td>
</tr>
<tr>
<td></td>
<td>Attractiveness</td>
<td>✓</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Time behaviour</td>
<td>Execution time ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum capability ✓</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Changeability</td>
<td>Size ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modularity ✓</td>
</tr>
<tr>
<td>Portability</td>
<td>Adaptability</td>
<td>Extensibility ✓</td>
</tr>
</tbody>
</table>

Table 6.28: Quality characteristics of UML-RSDS for quality improvement case study

Table 6.29 presents the summary characteristic values for GrGen. This approach scores highly for Functionality, Maintainability, Efficiency, Reliability, Portability and has mixed result for Usability.

Table 6.30 illustrates the summary characteristic values for Kermeta. This approach scores highly for Accuracy, Interoperability, Efficiency, Reliability and Portability, poorly for Maintainability and have mixed result for Usability and Suitability.

Table 6.31 gives the summary of ATL. The scaled up version of size, complexity and development effort is considered in this table. This approach scores highly for Interoperability, Reliability and Maintainability, and scores poorly for Accuracy, Efficiency and Portability. It has mixed result for the other characteristics.
### 6.9. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Subcharacteristic</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Suitability</td>
<td>Abstraction level ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Size ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Effectiveness ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Development effort ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Execution time ✓</td>
</tr>
<tr>
<td>Accuracy</td>
<td></td>
<td>Correctness ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Completeness ✓</td>
</tr>
<tr>
<td>Interoperability</td>
<td></td>
<td>Embeddable in transformation process ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Close to well-known notation ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Eclipse plug-ins ✓</td>
</tr>
<tr>
<td>Functionality</td>
<td>compliance</td>
<td>Close to well-known notation ✓</td>
</tr>
<tr>
<td>Reliability</td>
<td>Maturity</td>
<td>History of use ✓</td>
</tr>
<tr>
<td></td>
<td>Fault tolerance</td>
<td>Tolerance of false assumptions ✓</td>
</tr>
<tr>
<td>Usability</td>
<td>Understandability</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Learnability</td>
<td>×</td>
</tr>
<tr>
<td></td>
<td>Attractiveness</td>
<td>✓</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Time behaviour</td>
<td>Execution time ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum capability ✓</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Changeability</td>
<td>Size ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modularity ✓</td>
</tr>
<tr>
<td>Portability</td>
<td>Adaptability</td>
<td>Extensibility ✓</td>
</tr>
</tbody>
</table>

Table 6.29: Quality characteristics of GrGen for quality improvement case study

Table 6.32 shows the summary characteristic values for QVT-R. The size, complexity and modularity is scaled up by a factor of 1.5 to have an estimation of complete solution. In addition, it is estimated that the execution time and maximum capability for 100% effectiveness is not appropriate in this tool. This approach scores highly for Interoperability, Reliability and poorly for Accuracy, Efficiency and Portability, with mixed result for Usability and Suitability.

In conclusion, we could consider GrGen, UML-RSDS and Kermeta as good candidates to select for carrying out this transformation, on the basis of their overall rankings in terms of the ISO 9126 characteristics in Table 6.33. These tables present the number of good characteristics of each approach. If some factors are more in-
6.9. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Subcharacteristic</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Suitability</td>
<td>Abstraction level ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Size ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Effectiveness √</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Development effort ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Execution time √</td>
</tr>
<tr>
<td>Accuracy</td>
<td></td>
<td>Correctness √</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Completeness √</td>
</tr>
<tr>
<td>Interoperability</td>
<td>Embeddable in transformation process √</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Close to well-known notation √</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Eclipse plug-ins √</td>
<td></td>
</tr>
<tr>
<td>Functionality</td>
<td>compliance</td>
<td>Close to well-known notation √</td>
</tr>
<tr>
<td>Reliability</td>
<td>Maturity</td>
<td>History of use √</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fault tolerance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tolerance of false assumptions √</td>
</tr>
<tr>
<td>Usability</td>
<td>Understandability</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td>Learnability</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td>Attractiveness</td>
<td>×</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Time behaviour</td>
<td>Execution time √</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum capability √</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Changeability</td>
<td>Size ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modularity ×</td>
</tr>
<tr>
<td>Portability</td>
<td>Adaptability</td>
<td>Extensibility √</td>
</tr>
</tbody>
</table>

Table 6.30: Quality characteristics of Kermeta for quality improvement case study

investigated for a particular purpose, it will count more in the overall ranking table. Then the total score presents the general view about each approach. For measures like complexity, size development time and execution time, a high value means a Low score. For the rest of the attributes the high value means a High score. I also counted the Medium score as a good representative for characteristic of approaches.

It should be noted that this ranking of approaches is specific to this type of refactoring problem: in particular, GrGen and UML-RSDS are favoured over ATL and QVT-R because of their stronger support for update-in-place transformations. The inefficiency of model copying solutions for such problems is evident in the execution results for ATL and QVT-R compared to the other solutions. Different categories of
6.9. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Subcharacteristic</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Suitability</td>
<td>Abstraction level ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Size ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Effectiveness ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Development effort ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Execution time ✓</td>
</tr>
<tr>
<td>Accuracy</td>
<td></td>
<td>Correctness ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Completeness ×</td>
</tr>
<tr>
<td>Interoperability</td>
<td></td>
<td>Embeddable in transformation process ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Close to well-known notation ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Eclipse plug-ins ✓</td>
</tr>
<tr>
<td>Functionality</td>
<td>compliance</td>
<td>Close to well-known notation ✓</td>
</tr>
<tr>
<td>Reliability</td>
<td>Maturity</td>
<td>History of use ✓</td>
</tr>
<tr>
<td></td>
<td>Fault tolerance</td>
<td>Tolerance of false assumptions ✓</td>
</tr>
<tr>
<td>Usability</td>
<td>Understandability</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Learnability</td>
<td>×</td>
</tr>
<tr>
<td></td>
<td>Attractiveness</td>
<td>✓</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Time behaviour</td>
<td>Execution time ×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum capability ×</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Changeability</td>
<td>Size ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modularity ✓</td>
</tr>
<tr>
<td>Portability</td>
<td>Adaptability</td>
<td>Extensibility ×</td>
</tr>
</tbody>
</table>

Table 6.31: Quality characteristics of ATL for quality improvement case study transformation problem, such as model migrations, would produce different rankings.

The fundamental problems of this case study are:

- **The need to match multiple elements, including sets of arbitrary size.**

- **The need to control the order of application of rules.**

- **The need to support update-in-place transformation.** The transformation is input-destructive and it would be very inefficient to implement by means of copying models.

- **The need to provide optimisations.** The complexity of expressions used in matching and updates would produce inefficient implementations without the use of
### 6.9. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Subcharacteristic</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Suitability</td>
<td>Abstraction level ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Size ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Effectiveness ✗</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Development effort ✗</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Execution time ✗</td>
</tr>
<tr>
<td>Accuracy</td>
<td></td>
<td>Correctness ✗</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Completeness ✗</td>
</tr>
<tr>
<td>Interoperability</td>
<td></td>
<td>Embeddable in transformation process ✗</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Close to well-known notation ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Eclipse plug-ins ✓</td>
</tr>
<tr>
<td>Functionality</td>
<td>compliance</td>
<td>Close to well-known notation ✓</td>
</tr>
<tr>
<td>Reliability</td>
<td>Maturity</td>
<td>History of use ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tolerance of false assumptions ✓</td>
</tr>
<tr>
<td>Usability</td>
<td>Understandability</td>
<td>✗</td>
</tr>
<tr>
<td></td>
<td>Learnability</td>
<td>✗</td>
</tr>
<tr>
<td></td>
<td>Attractiveness</td>
<td>✓</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Time behaviour</td>
<td>Execution time ✗</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum capability ✗</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Changeability</td>
<td>Size ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complexity ✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modularity ✓</td>
</tr>
<tr>
<td>Portability</td>
<td>Adaptability</td>
<td>Extensibility ✗</td>
</tr>
</tbody>
</table>

Table 6.32: Quality characteristics of QVT-R for quality improvement case study optimisation.

- The need to provide fixpoint iteration of rules.

The solutions have addressed these problems in different ways, depending on the facilities available in the different languages:

**Multiple element matching** In the UML-RSDS solution this is addressed by using multiple quantifiers iterating over quantified variables which serve as pivot elements for the constraint, and deriving the sets of interest (sets of sibling classes with cloned attributes) from these multiple pivot elements.
6.9. COMPARISON OF APPROACHES

<table>
<thead>
<tr>
<th>Approach</th>
<th>Func. (12)</th>
<th>Rel. (2)</th>
<th>Usab. (3)</th>
<th>Effic. (2)</th>
<th>Maint. (3)</th>
<th>Port. (1)</th>
<th>Total (23)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GrGen.NET</td>
<td>12</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>UML-RSDS</td>
<td>11</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>Kermeta</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>ATL</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>QVT-R</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 6.33: Overall ranking of approaches for the quality improvement case study

In GrGen, the sets of elements are specified implicitly by means of graph patterns, positive and negative. Modifications involving such sets are performed by procedural iteration of updates to individual elements.

In the Kermeta solution, sets of elements are constructed by iterations over the sets of all classes or attributes in the model, these sets are then passed to operations which update the model based on the sets.

In the ATL solution multiple elements are gathered into sets by means of the using construct, the sets are derived from the input elements of the rule.

In the QVT-R solution, the sets of classes containing copies of a specific attribute need to be precomputed by a sequentially proceeding transformation. The attributes which have copies to be removed are identified by the ‘marking’ pattern of [47]. These steps considerably complicate the transformation specification and are a consequence of the element-by-element pattern matching and processing in QVT rules.

A more powerful solution than these would be to provide direct quantification/matching over sets of elements as pivots for transformation rules, as proposed for the CGT language in [50].

Control of rule ordering In UML-RSDS rule applications are ordered based on rule priority: the first rule takes precedence over the second rule, and the second rule over the third rule. The priority is syntactically defined by rule ordering.

In GrGen an explicit xgrs statement defines the processing order of the rules.

In Kermeta the ordering of the rules is defined by an explicit algorithm invoking the rules in the required order.
6.9. COMPARISON OF APPROACHES

In ATL the application condition of rule 2 is defined to exclude the application condition of rule 1, so that rule 2 is only applied if rule 1 is not applicable.

In QVT-R the relative ordering of the top-level relation executions can be controlled by the use of when and where clauses, a rule that should occur after another should contain a when clause condition that can only be established by its predecessor.

A fine level of control, e.g., to iterate over classes from the leaves of the inheritance hierarchy upwards, is required to give an optimal solution to the problem. This is only achieved in the Kermeta solution by explicitly processing the classes from leaf classes up to their superclasses. For the declarative languages, a new transformation language construct to specify particular orders of matchings in a model could be useful.

Update-in-place execution This is directly supported by GrGen, Kermeta and UML-RSDS. In QVT-R this is not directly supported, instead the input model must be copied to the output model, with any required changes being applied. In-place transformations are supported in ATL with the use of the refining mode; however this execution mode is still immature and only matched rules are allowed.

Optimisation facilities In UML-RSDS execution of the constraints is optimised by factoring out repeated occurrences of expressions using let variables \(^{11}\), and by omitting negative application condition tests.

In GrGen, helper operations are used to factor out repeated expression evaluations.

In Kermeta, repeated evaluation of expressions can be avoided by calling an operation to evaluate the expression once, and storing its result to use in multiple places.

In ATL, using definitions (local variables of rules) as well as helpers are used to avoid duplicated expressions.

\(^{11}\) \(v\) in C2 and C3
In QVT-R, subrelations can be defined to factor out some repeated calculations, and the precomputation of sets of elements used repeatedly can be used to optimise computations.

**Fixpoint iteration** In UML-RSDS this is the default implementation of constraints that both read and write some entity or feature.

In GrGen and Kermeta it needs to be explicitly programmed as an iteration which halts when no further elements matching the application conditions of any rule exist in the model. In QVT-R it is the standard implementation of top-level rules, and it is the standard implementation of ATL matched rules.

### 6.10 Conclusions

The case study has clearly identified the advantages and disadvantages of different transformation approaches for restructuring transformation problems. The sample problem was simplified by using a small metamodel, compared to the UML class diagrams. It nonetheless presents the same issues as the full restructuring problem. Specific deficiencies of particular approaches, such as the lack of appropriate language support in QVT-R and ATL have been identified. General deficiencies of transformation approaches have also been presented, such as the need for fine-grain control over the order in which elements of a model are selected for matching against a rule, and intrinsic language support for matching sets of elements at a time.

There is a dramatic difference between the size of specification in UML-RSDS than other approaches. It is because of the particular specification paradigm in this language. Kermeta has significantly higher complexity than the more declarative approaches. It is due to the imperative style of specification in this language. The development effort was less for generating GrGen solution and, it scored higher in the usability survey. The maximum capability of UML-RSDS and GrGen is higher than other approaches. UML-RSDS has a comprehensive support for the proof of syntactic correctness, using a translation to the B formal method and proof tool; however the lack of detailed control over the transformation execution means that confluence cannot be proved. Only Kermeta is able to enforce an optimal clone-removal strategy and achieve confluence in principle.
Evaluation of this case study showed that UML-RSDS, GrGen and Kermeta are suitable candidates for performing this quality improvement case study. QVT-R does not seem suitable for this type of transformation problem, because of the difficulty of handling sets of elements, and the lack of support for input-destructive update-in-place transformations. ATL is not currently suitable because of the limitations of the refining mode. In terms of size, complexity and development time, UML-RSDS and GrGen have clear advantages over Kermeta, and in terms of maturity, GrGen and Kermeta are preferable to UML-RSDS. If guaranteed optimal solutions were required, then Kermeta would be the most appropriate approach to use.
Conclusion

We have investigated the evaluation of model transformation approaches on different case studies, using a systematic framework. Measuring model transformation in Model Driven Engineering is a critical subject of study, which helps us to understand, identify, improve and select appropriate specification tools for a particular transformation task. This thesis established an evaluation framework and procedure which can compare and select transformation approaches for a particular problem, using quantitative measures. The framework is based on the ISO/IEC 9126 quality model framework and its use for software evaluation. The ISO/IEC 9126 paradigm defines quality models based on general characteristics of software, which are further refined into subcharacteristics. In this thesis relevant characteristics and subcharacteristics for the evaluation of the model transformation have been selected and further decomposed into measurable attributes. The Goal Question Metrics paradigm is used to identify metrics for the measurements of the attributes of the transformation specification language and transformation implementation tool of each approach. The case studies used in this thesis have clearly identified the advantages and disadvantages of different transformation approaches for refinement, re-expression, migration and restructuring transformation problems.

The remainder of this chapter summarises the contributions of the thesis. First, a brief overview of the thesis is given, looking at each of the chapters in turn. Next, the review contributions of the thesis are detailed. Following that, the summary contributions are given. Then, potential extensions to the thesis research are presented. Finally, concluding remarks are provided.
7.1 Overview of Thesis

Chapter 2 presents Model Driven Engineering and its related concepts. It provides some discussion on the notation of the model and its different classifications. The main contribution of Chapter 2 is the explanation of model transformation as a central element in MDE. Alongside this, a comprehensive taxonomy of model transformations from different viewpoints are provided. Furthermore, the chapter presents a brief introduction to common-used transformation languages in the community, using a typical example of a transformation from "a class diagram to a relational database". Through this, the difficulty of selecting an appropriate transformation language is pointed out. Finally, Chapter 2 ends with a discussion on the advantages of the ISO/IEC 9126 quality model as a fundamental framework for evaluation of the transformation approaches.

Chapter 3 highlights the importance of measurement in model transformation and proposes a methodology for evaluating the transformation approaches. The procedure of measurement is shown by selecting the relevant characteristics and subcharacteristics to model transformation from the ISO/IEC 9126 software quality model. Alongside this, the use of Goal Question Metrics paradigm for identifying appropriate metrics for the measurement of attributes in this framework is explained.

Chapter 4 details a comparison of five different transformation specification approaches on the well-known "tree to graph" case study. In addition, a refinement transformation from "a class diagram to a relational database" is investigated. Chapter 4 analyses the effectiveness of the approaches for performing these tasks, with regards to the evaluation methodology in Chapter 3.

Chapter 5 indicates the applicability of the UML-RSDS transformation tool to the migration case study for transformation of "UML activity diagrams from version 1.4 to 2.2". Following this, expert evaluation is generated by comparing UML-RSDS to the Epsilon Flock and GrGen implementation, using the comparison methodology in Chapter 3.

Chapter 6 compares five established model transformation languages from different language categories upon a transformation problem that is typical of a quality-improvement restructuring transformation. The problems with the specification of these categories of model transformation are then highlighted.
7.2 Review of Contributions

We introduce a systematic procedural framework for measurement of model transformation approaches in Chapter 3 based on the ISO/IEC 9126 quality model. The framework is applicable for measurement of transformation languages with different structures and styles. By systematically comparing and evaluating the selected transformation approaches on the case studies, clear guidelines can be provided for the appropriateness of different types of transformation approaches for the specific transformation problem. This section summarizes the result of each transformation language for performing case studies in Chapters 4, 5 and 6.

7.2.1 Refinement and Re-expression transformations in Chapter 4

In Chapter 4 we explore the effectiveness of five transformation languages on the “tree to graph” and “class to relational database” transformations. The “tree to graph” transformation relates tree objects in the source model to node objects in the target model with the same name, and defines that there is an edge object in the target model for each non-trivial relationship from a tree node to its parent. The second case study concerns the mapping of a data model expressed in UML class diagram notation to the more restricted data modelling language of relational database.

The evaluation shows that the ATL transformation language generates a moderate size specification at high level of abstraction for both case studies. The rule based specification in ATL is a convenient paradigm for specifying a transformation. In addition, it provides a less complex specification for the larger case study of “class to relational database” transformation in comparison to other approaches. The ATL approach enables to process the large input models in an appropriate time. The developer has implemented the first case study in ATL language under the supervision of an expert in a reasonable amount of time. Importantly, there is a methodology to check the consistency of the rules in ATL approach. However, there is no direct tool to support the syntactic correctness. ATL has a long history of use and is interoperable with other environments and tools. Moreover, the result of the usability survey showed that ATL is considered attractive to the users. Based on the evaluation
framework in this thesis it has been noted that ATL scores highly for *Interoperability, Reliability, Efficiency* and *Portability*, and has mixed result for other characteristics for performing refinement and re-expression case studies in Chapter 4.

The UML-RSDS transformation approach generates a transformation by formalizing operations upon metamodels. The tool produces a factorised specification with few lines of code for both case studies. In addition, the interoperability is high because the standard UML and OCL notations are used for the specification. The transformation tool enables one to handle large input models and produces the results in a short amount of time. The first case study is developed in an appropriate time with the assistance of expert in UML-RSDS approach. In addition, this tool provides syntax check, although it lacks semantic checks. The termination of UML-RSDS is clear with the use of bounded loops in the specification. Moreover, different orders of execution of rules generate the same results in both case studies. Through this the confluence is guaranteed. UML-RSDS has recently become available in public, and there is no interoperability with Eclipse. The UML-RSDS approach scores highly for *Accuracy, Efficiency, Portability* and poorly for *Reliability, Usability*, and has mixed result for other characteristics. The significant problem in UML-RSDS is the lack of *Reliability* and *Usability*. This tool needs to improve its strategy for handling errors within models and within processing. Although UML-RSDS is based on the standard UML and OCL notations, the style of specification in this language is not attractive for the users’.

The QVT-R produces a large size transformation at a high level of abstraction for the two case studies. The complexity of transformation specification is medium for both case studies. The QVT-R approach produces a factorised specification in the second case study, while the factorisation is only 20% in the first one. The transformation is divided into two modules for the second case study, but still the total modularity is moderate, as there is only one call between two modules. It takes a considerable amount of time to implement the first case study in QVT-R without an experts assistance. The attractiveness of the approach is reflected through the usability survey; however the notations are less understandable and learnable in comparison to other approaches. The specification is executable in the MediniQVT tool, where it is also has an interoperability with Eclipse. The execution time of QVT-R transformation tool in high in comparison to other approaches, and its maximum capacity is
7.2. REVIEW OF CONTRIBUTIONS

also limited. All in all, the QVT-Relational approach generates a transformation that scores highly for Interoperability, Reliability, poorly for Usability, Efficiency, Portability, with mixed result for others characteristics. The QVT-R tool needs to provide a methodology to process the transformation in shorter amount of time and increase its capacity for handling large input models.

Kermeta is the other language whose quality is investigated in Chapter 4. It provides a transformation in a low level of abstraction. The imperative nature of the code causes the size and complexity of the second case study to be large. In addition, the specification of transformation is nearly factorized in both cases and modularity is provided in an object-oriented way by using classes and packages. Expert assistance is not necessary, we saw that even an inexperienced developer was able to develop the transformation in a short period of time. It must be taken into account that there is no direct support for establishing syntactic correctness; however assertions and invariants could be used to check this for individual models. A plus side to Kermeta is that the order of rule applications is explicit, and must be defined by the developer. This in turn improves the consistency for this language. The understandability and learnability of Kermeta are relatively high from the users perspective; however it is not perceived as attractive. Kermeta has a long history of use and scores highly for the interoperability with Eclipse. In summary, Kermeta scores highly for Accuracy, Interoperability, Reliability, Efficiency and Portability, and has mixed result for Maintainability and Usability for performing re-expression and refinement case studies.

Viatra is another transformation language which is investigated in the comparison study of Chapter 4. This language produces a transformation at a high level of abstraction. A moderate specification size is generated for the first case study and a high one for the second. A negative feature of the Viatra approach is that the specification is indeed complex. In both case studies the transformations are factorized; however it is only in the second case study where the rule is encapsulated inside the two modules. This results in a number of calls between two modules and reduces the cohesion of the specification. The semantic correctness is checked partially in the Viatra approach. Importantly, there is an explicit scheduling of different rules. However, it is not possible to control the order of application on the individual elements. In Viatra the maturity of the tool is classified as medium and the interoperability
7.2. REVIEW OF CONTRIBUTIONS

Table 7.1: Overall ranking of approaches for the first case study in terms of the ISO 9126 characteristics

<table>
<thead>
<tr>
<th>Approach</th>
<th>Funct. (12)</th>
<th>Rel. (2)</th>
<th>Usab. (3)</th>
<th>Effic. (2)</th>
<th>Maint. (3)</th>
<th>Port. (1)</th>
<th>Total (23)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kermeta</td>
<td>11</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td>ATL</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>UML-RSDS</td>
<td>11</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>QVT-R</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>Viatra</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>-</td>
<td>2</td>
<td>0</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 7.1 and 7.2 shows the overall ranking of the approaches for the two case studies in Chapter 4. These tables present the number of good characteristics of each approach. If some factors are more investigated for a particular purpose, it will count more in the overall ranking table. Then the total score presents the general view about each approach. For measures like complexity, size development time and execution time, a high value means a Low score. For the rest of the attributes the high value means a High score. I also counted the Medium score as a good representative for characteristic of approaches. The results of comparison on refinement and re-expression type of transformation problems, suggest that the explicit programming paradigm of Kermeta and the hybrid approach of ATL are the most suited for novice model transformation developer’s to apply, at least on simple transformations. Developers effort was lowest for implementing the first case study in Kermeta. This is due to the familiarity of the object-oriented programming paradigm and requires the least work to construct and analyse the specification. Clearly, explicit control over the rule execution in Kermeta seems preferable for usability, correctness and analysis. However, the imperative style of Kermeta causes a complex specification for the larger case studies. ATL has an appropriate rule based paradigm for specifying a transformation. This approach presents a less complex result for "class to relational
7.2. REVIEW OF CONTRIBUTIONS

<table>
<thead>
<tr>
<th>Approach</th>
<th>Funct. (11)</th>
<th>Rel. (2)</th>
<th>Effic. (2)</th>
<th>Maint. (3)</th>
<th>Port. (1)</th>
<th>Total (19)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATL</td>
<td>10</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>Kermeta</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>UML-RSDS</td>
<td>9</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>QVT-R</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>Viatra</td>
<td>5</td>
<td>2</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 7.2: Overall ranking of approaches for the second case study in terms of the ISO 9126 characteristics

database” case study than the other investigated approaches. In addition, the use of a standard notation such as OCL in its specification supports the interoperability of the tool. The UML-RSDS approach presents strong result in terms of Functionality. However due to the particular style of specification and low ranking result for the Reliability characteristic, it is less appropriate than ATL and Kermeta. The QVT-R and Viatra approaches are placed lower than other approaches in the overall ranking table. Even developing a simple “tree to graph” transformation takes a substantial amount of time in these languages.

7.2.2 Model Migration transformation in Chapter 5

The next significant contribution of this thesis is the evaluation of three approaches on the migration transformation of “UML activity diagrams from version 1.4 to 2.2”. In UML 1.4 the language of activity diagrams is a variant of the state machine language. However in UML 2.2 the activity diagram is reformulate as kind of Petrinet, such as places and transitions.

This case study is the first migration task to be tackled with the UML-RSDS tool. It specifies the mapping by sets of rules placed in the source model. The specification depends on the use of primary keys for the model elements. The transformation approach use explicit copying methodology to map elements from the source to the target. Because of this the development time for generating migration task is high and consequently, it generates a complex transformation with many lines of code. The findings shows that the approach is quite modular and is encapsulated inside a single module. The UML-RSDS tool compiles the transformation rather than interprets it. Therefore, the result is produced in shorter space of time. Moreover, in the
specification of UML-RSDS for migration case study the order of the application of the rules does not affect the result. Therefore, the confluence is preserved. The syntactic correctness is provided by translation to the B formal method. Alongside, the termination is provided because of bounded loops in the specification. It has been reported that the transformation tool recognises faults, but it can not identify where the errors occur. Overall, the UML-RSDS approach scores highly for Accuracy and Efficiency, but poorly for Reliability, Usability and Portability in performing migration task in Chapter 5. Further improvement in UML-RSDS approach should consider an automatic way of copying unaffected elements within the transformation procedure.

Flock is a model migration tool and generates a transformation at a high level of abstraction. The tool uses the conservative copying strategy for model migration tasks. This strategy copies all unaffected elements from the source to the target automatically. This therefore results in a transformation with fewer lines of code than the approaches that use explicit copying strategy. Moreover, the calls are implicit in Flock which reduces the complexity of the transformation specification. Additionally, modularisation is high and transformation is factorized in the Flock solution. Flock shows a comprehensive solution to the extended tasks. In addition, the standard notations such as OCL are used in the specification of this approach, that increase interoperability. The order of execution of the rules is not clear in Flock; however different orders generate the same results for this solution. Furthermore, the termination of the Flock solution is not clear because of the non-determinate order of the recursive descent strategy. Flock has been available since 2009 and only a few cases have been solved using it. All in all, Flock scores highly for most of the evaluation criteria in this thesis, including Suitability, Interoperability, Usability, Efficiency, Maintainability and Portability in generating migration task in Chapter 5.

The evaluation of the GrGen solution on the migration case study has also been investigated. The approach generates a declarative specification by using a graph metamodel and rewrites rules. It handles nodes and edges separately, which results in more lines of code and complexity within the specification. The partial extension to the tasks are provided in the GrGen solution. Moreover, the OCL notations are used in the specification. The development time for the migration task in GrGen was reported around four hours. There is no direct tool to check the syntactic correctness
7.2. REVIEW OF CONTRIBUTIONS

<table>
<thead>
<tr>
<th>Approach</th>
<th>Funct. (11)</th>
<th>Rel. (2)</th>
<th>Usab. (3)</th>
<th>Effic. (2)</th>
<th>Maint. (2)</th>
<th>Port. (1)</th>
<th>Total (21)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flock</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>GrGen</td>
<td>11</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>UML-RSDS</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 7.3: Overall ranking of approaches for the migration case study in terms of the ISO 9126 characteristics

in the GrGen approach. However, the linear solution provides clear termination. The explicit order of rules prevents faults from arising in the case of conflicts, but it does not ensure confluence. One of the strengths of the language is that it has a high tolerance for fault occurrences and error messages are shown in the exact place that the problem raised. Importantly, the GrGen approach has a long history of use. In summary, GrGen scores highly for Accuracy, Interoperability, Reliability, Efficiency and Portability. It scores poorly for Usability in producing migration task in Chapter 5.

Table 7.3 shows the overall ranking in terms of the ISO/IEC 9126 characteristics for the Flock, GrGen and UML-RSDS approaches in generating migration case study. The finding shows that model migration tools are significantly more understandable than the general purpose transformation tools for performing migration tasks. The Epsilon Flock is an appropriate candidate for performing migration tasks and generates a less complex specification than the other approaches investigated here. The conservative copying strategy in Flock allows one to have a concise model migration program, while in UML-RSDS many of the transformation rules involve explicit copying of the values that are not affected by the metamodel evolution. The explicit copying in UML-RSDS increases the complexity of specification and, therefore reduces its usability. The separation of edges and nodes in the GrGen solution increases the modularity; however the transformation has excessive size and complexity. The GrGen and Flock transformation approaches generates quite a similar score for the migration task, but the specification in Flock is less complex and more understandable from the users point of view.
7.2.3 Quality Improvement transformation in Chapter 6

The next significant evaluation result from this thesis is given through the “class diagram restructuring” case study. Such a quality improvement transformation iteratively rewrites a model in-place to improve some quality characteristics.

We first investigated the UML-RSDS approach. This language specifies the system data through a UML class diagram together with OCL constraints. The solution to the quality improvement case study is implemented by using a class diagram and a single use case. The use case has sets of assumptions as pre-conditions and sets of constraints as post-conditions. In the UML-RSDS specification, the operations are not invoked, which implies a 0% structural complexity. In addition, the specification is not factorized because of the repeated occurrence of duplicated expressions. This coupled with 100% cohesion in the specification result in a moderate modularity. The transformation specification can be used for the extended class diagram metamodels. It can be composed internally by means of the use case inclusion. Moreover, the development effort for this approach is evaluated as low. The UML-RSDS tool provides syntactic correctness by translation to the B formal method and proof tool. The rules are not confluent, and different orders of execution of the rules generate different results in this solution. The UML-RSDS tool generates effective results for all the test cases. In addition, the result of stress testing with different iterations of the testcase 2 shows the high capability that the tool has for handling large input models. The usability survey shows that the majority of participants considered the specification as well-structured. The evaluation shows that this tool has a short history of being applied to transformation problems and in fact is relatively immature. In addition, there is no Eclipse interface for this tool, instead input and output is by means of text files. From this evaluation we can conclude that, this approach is rated highly for Suitability, Accuracy, Efficiency and Maintainability, but poorly for Reliability and Usability for performing quality improvement case study in Chapter 6.

The second language which is applied to this case study is GrGen. It generates the transformation in terms of graph patterns. Its syntax relates to visual presentation of graph nodes and edges. One of the most important features of the GrGen approach is that it is able to prioritise rules. This means the second rule is only executed under the condition that the first rule fails. Another feature of GrGen is
its ability to extend the transformation by using additional rules or extending the existing ones. The specification factorization is 62%. In addition, the specification is located inside a single module. GrGen.NET produces C# program to implement the transformation. This feature provides this tool with the ability to handle large test cases and generates the results in an appropriate time. Moreover, the validation of the transformation is by use of a visual debugger. The usability survey indicates that participants consider this tool to be structured, while medium effort is needed to understand it. All respondents correctly identified the code that is used in the learnability test. There are fifty transformation case studies which have been implemented using GrGen. Alongside, the maturity of the tool is high and it has been publically available since 2003. In summary, this approach scores highly for Functionality, Maintainability, Efficiency, Reliability, Portability and has mixed result for Usability in order to tackle the quality improvement case study in Chapter 6.

The imperative language of Kermeta produces a transformation with a large size and complexity for this case study. However, the imperative nature of the tool provides more flexibility to control the transformation process. The transformation specification is not factorized in Kermeta and cohesion is low in comparison to other investigated approaches. The Extension to a larger case study is a possibility, but it requires huge amounts of coding. The Kermeta language uses the OCL notations in its specification. This approach generates an effective result to the case studies and responds positively to the stress testing. The approach provides an assertion capability to check the assumptions at runtime. The use of recursion for the top-level control of the application of the rules makes proof of the termination less clear than UML-RSDS. The survey acknowledged that the Kermeta approach is attractive. We also note that, it takes longer than expected to develop this task using Kermeta. Yet the strengths of Kermata are still clear as the maturity considers as high with an extensive history of use. Kermeta has a good support for interconnection with Eclipse and so overall, Kermeta scores highly for Accuracy, Interoperability, Efficiency, Reliability and Portability, poorly for Maintainability and have mixed result for Usability and Suitability for solving quality improvement problem in Chapter 6.

ATL develops a transformation at a high level of abstraction. Unfortunately, the unchanged part of the model needs to be specified explicitly, which results in a high syntactic complexity. It is not possible to implement this task in refining mode.
due to the existence limitation. However, the partial implementation of this case study in the normal mode is provided. There are invocation of rules by other rules within the specification. On a positive note, the factorization of the specification is 72% and the transformation encapsulates inside a single module. It results in a high cohesion and therefore a high modularity. This language provides us with an option of extending to a larger model, although considerable additional complexity to the rules will be required. The use of a standard notation such as OCL in the specification supports the interoperability and comprehensibility of the tool. However, it must be taken into account that the confluence fails as the different order of execution of the rules generate different results. Yet the termination remains clear due to the linear iteration over the source domains. ATL is very commonly used in both the academic and industrial fields and its availability dates back to 2003 with over hundred tasks tackled. This approach scores highly for Interoperability, Reliability and Maintainability, and scores poorly for Accuracy, Efficiency and Portability. It has mixed result for the other characteristics.

The QVT-R attempts to establish a number of relations between source and target models by creating or modifying elements in the target models through the notations of UML and OCL. The complexity of performing the task in QVT-R is due to the explicit copying, creating, deletion and modification of model elements. In addition, the transformation is not completely implemented and only the first rule and part of the second rule is implemented in this approach. The number of the syntactic complexities is reasonable. The transformation is modularized into two modules. Developing this task requires a substantial amount of effort, with a high development time. Given the current state, clearly extending the metamodel will be even more difficult. The confluence of the solution is not established due to the declarative nature of the tool. In the usability survey all participants are needed to put in a large effort to understand the transformation, but most of them are found it well-structured. The Medini tool provides a run-time execution handling for QVT-R approach. Moreover, the tolerance for false assumptions is high and error messages show while specifying the lines where the error has occurred on. However, this tool has limited capacity for performing large test cases. In 2008, QVT is introduced as the standard language for model transformation by the Object Management Group. Until now there are twenty published case studies with QVT-R approach. This approach scores highly
7.2. REVIEW OF CONTRIBUTIONS

<table>
<thead>
<tr>
<th>Approach</th>
<th>Funct. (12)</th>
<th>Rel. (2)</th>
<th>Usab. (3)</th>
<th>Effic. (2)</th>
<th>Maint. (3)</th>
<th>Port. (1)</th>
<th>Total (23)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GrGen.NET</td>
<td>12</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>UML-RSDS</td>
<td>11</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>Kermeta</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>ATL</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>QVT-R</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 7.4: Overall ranking of approaches for the quality improvement case study in terms of the ISO 9126 characteristics

for Interoperability, Reliability and poorly for Accuracy, Efficiency and Portability, with mixed result for Usability and Suitability.

GrGen, UML-RSDS and Kermeta considers as good candidates for carrying out quality improvement transformations (Table 7.4). In terms of size, complexity and development time, UML-RSDS and GrGen have clear advantages over Kermeta, while in terms of maturity, GrGen and Kermeta are preferable to UML-RSDS. Although the complexity and size of the Kermeta solution is significantly higher than the UML-RSDS and GrGen solutions, the language does provide a more precise control over rule applications. The QVT-R approach does not seem suitable for this type of transformation problem, because of the difficulty of handling sets of elements, and the lack of support for update-in-place transformations. Similarly, the ATL transformation approach is not currently suitable because of the limitations of the refining mode.

The correlation results between different measures of the solutions give some evidence for the validity and consistency of the selected measures (It is only evaluated for the last case study as the others are quite small cases.). The Pearson correlation coefficient of usability with complexity for Kermeta, UML-RSDS and GrGen is -0.663, indicating a moderately strong linear dependency. The correlation coefficient of usability with size for Kermeta, UML-RSDS and GrGen is -0.55. The correlation coefficient of usability with modularity is 0.69 for Kermeta, UML-RSDS, GrGen and ATL, indicating a positive linear correlation. It means the more modular the specification is, the more attractive and therefore understandable it becomes.

In addition, we considered the correlation of five transformation approaches, with the scaled version of QVT-R and ATL by 1.5 and 3, respectively (to have estimation of completed solution). The correlation of size against complexity is 0.99 and against development time is 0.58. Alongside, the correlation between complexity and devel-
7.2. REVIEW OF CONTRIBUTIONS

Development effort is reported as 0.67. These shows that three measure of size, complexity and development time agree with each other. Modularity is negatively correlated with development effort at -0.72 when all the solutions are considered.

7.2.4 Summary of Contributions

Table 7.5 shows a summary of the attributes values of the transformation approaches that are applied to tackle each case study. In the first, second and fourth case studies, the ticks correspond to the UML-RSDS (U), Kermeta (K), ATL (A), QVT-R (Q), Viatra (V)/GrGen (G) approaches. For the third case the ticks used indicate UML-RSDS (U), Flock (F) and GrGen (G), respectively.

The result of evaluation in this thesis shows that Kermeta is a reasonable candidate for performing re-expression, refinement and quality improvement case studies. The explicit control over the execution of the rules in Kermeta is desirable for the users. In addition, the object-oriented programming paradigm in this approach requires the least work to construct and analyse the specification. However, Kermeta approach is not the best method for performing large case studies in terms of size and complexity. The finding shows that, the rule-based specification language of ATL is also a good candidate for performing refinement and re-expression transformations. ATL presents an effective solution to the large case studies with appropriate lines of code and complexity in the specification. Kermeta presents a less complex transformation for the “tree to graph” case study than ATL. It needs a large amount of effort to specify a simple “tree to graph” transformation in a procedural language of Viatra and relational language of QVT-R.

ATL is not suitable for implementing update-in-place transformation. It is due to the limitations of refining mode in the ATL approach. Moreover, using the normal mode of ATL does not solve the problem and complicates the task. Similar to the ATL approach, QVT-R is inefficient to tackle the update-in-place problem, due to the limitation of the declarative style in this language.

The rule-based transformation languages of UML-RSDS and GrGen provide strong support for different types of transformation. They present the best solutions for update-in-place transformation, and spend less time on its development. The response time is faster in the UML-RSDS and GrGen approaches than the other tools.
### 7.2. REVIEW OF CONTRIBUTIONS

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Case Study 1</th>
<th>Case Study 2</th>
<th>Case Study 3</th>
<th>Case Study 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstraction level</td>
<td>✓ × ✓ ✓ ✓</td>
<td>✓ × ✓ ✓ ✓</td>
<td>✓ ✓</td>
<td>✓ × ✓ ✓ ✓</td>
</tr>
<tr>
<td>Size</td>
<td>✓ ✓ ✓ × × ✓</td>
<td>✓ ✓ ✓ × × ✓</td>
<td>✓ ✓ ×</td>
<td>✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Complexity</td>
<td>✓ ✓ × ✓ × ✓</td>
<td>× × ✓ ✓ ×</td>
<td>× ✓ ✓</td>
<td>✓ ✓ ✓ × × ✓</td>
</tr>
<tr>
<td>Effectiveness</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Development effort</td>
<td>✓ ✓ × × ×</td>
<td>– – – – –</td>
<td>× ✓ ✓</td>
<td>✓ × × × × ✓</td>
</tr>
<tr>
<td>Execution time</td>
<td>✓ ✓ ✓ × –</td>
<td>✓ ✓ ✓ × –</td>
<td>✓ ✓ ✓</td>
<td>✓ ✓ × × × ✓</td>
</tr>
<tr>
<td>Correctness</td>
<td>✓ ✓ × × ✓</td>
<td>✓ ✓ × × ✓</td>
<td>✓ × ✓</td>
<td>✓ ✓ ✓ × × ✓</td>
</tr>
<tr>
<td>Completeness</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓</td>
<td>✓ ✓ × × × ✓</td>
</tr>
<tr>
<td>Embeddable in transformation process</td>
<td>✓ ✓ ✓ ✓ ×</td>
<td>✓ ✓ ✓ ✓ ×</td>
<td>✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Close to well-known notation</td>
<td>✓ ✓ ✓ ✓ ×</td>
<td>✓ ✓ ✓ ✓ ×</td>
<td>✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Eclipse plug-ins</td>
<td>× ✓ ✓ ✓ ✓ ✓</td>
<td>× ✓ ✓ ✓ ✓ ✓</td>
<td>× ✓ ✓</td>
<td>× ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>History of use</td>
<td>× ✓ ✓ ✓ ✓ ✓</td>
<td>× ✓ ✓ ✓ ✓ ✓</td>
<td>× ✓ ✓</td>
<td>× ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Tolerance of false assumptions</td>
<td>× ✓ ✓ ✓ ✓ ✓</td>
<td>× ✓ ✓ ✓ ✓ ✓</td>
<td>× ✓ ✓</td>
<td>× ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Maximum capability</td>
<td>✓ ✓ ✓ × –</td>
<td>✓ ✓ ✓ × –</td>
<td>– – –</td>
<td>✓ ✓ ✓ × × ✓</td>
</tr>
<tr>
<td>Modularity</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
<td>✓ ✓ ✓</td>
<td>✓ ✓ ✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Extensibility</td>
<td>✓ ✓ ✓ × ×</td>
<td>✓ ✓ ✓ × ×</td>
<td>× ✓ ✓</td>
<td>✓ ✓ ✓ ✓ × ✓</td>
</tr>
</tbody>
</table>

Table 7.5: Summary of the attributes values of the transformation approaches for each case study. In the first ("tree to graph transformation"), second ("class to relational database transformation") and fourth case studies ("quality improvement transformation"), the ticks correspond to the UML-RSDS (U), Kermeta (K), ATL (A), QVT-R (Q), Viatra (V) (first and second case studies)/GrGen (G) (fourth case study) approaches. For the third case ("UML 1.4 to UML 2.2 transformation") the ticks used indicate UML-RSDS (U), Flock (F) and GrGen (G).
investigated here. In addition they effectively handle large test cases. The score of functionality for UML-RSDS is close to GrGen, however the particular style of specification in the UML-RSDS language makes it less understandable than GrGen.

The result of the evaluation in the migration case study presents Flock as the best solution. It is due to the conservative copying strategy in Flock that allows it to have a concise solution. However, UML-RSDS explicitly specify the migration task, which leads to a complex transformation with more lines of code. Similarly, separate processing of edges and nodes in the GrGen approach provide a transformation with excessive lines of code and complexity.

Figure 7.1 presents the summary of results of the best solution for each case study, according to the evaluation framework in this research. I use “H” to denote high, “M” to denote medium and “L” to denote low to represent the results of each quality attribute used within this framework. The inner ring reflects the result of the Kermeta solution for the transformation of “tree to graph” in Chapter 4. If we then work our way out from the inner ring, this is then followed by the result of ATL for the “class to relational database” transformation in Chapter 4. The next ring then presents the Flock evaluation result for the migration of “UML 1.4 to UML 2.2” case study in Chapter 5. Finally, the forth ring from the center shows the result of GrGen for the quality improvement case study in Chapter 6. This figure represents the overview of how good the best approach for each problem domain is. For instance it can be seen that correctness never scores high for any of the approaches.

7.3 Future Work

The previous section has detailed a set of major contributions regarding this thesis. There is great potential for the application of model transformation in Model Driven Engineering community. This section will now detail a set of specific future work that could potentially be undertaken to extend this research further.
7.3. FUTURE WORK

Figure 7.1: Results of best solution for each case study. “H” denotes high, “M” denotes medium and “L” denotes low. The inner ring reflects the result of the Kermeta solution for the transformation of “tree to graph” in Chapter 4. This is then followed by the result of ATL for the “class to relational database” transformation in Chapter 4. The next ring then presents the Flock evaluation result for the migration of “UML 1.4 to UML 2.2” case study in Chapter 5. The forth ring from the center shows the result of GrGen for the quality improvement case study in Chapter 6.

7.3.1 Optimising model transformations using design patterns

Design patterns for model transformation is an important area which provides solution for a number of model transformation specifications and design problems [16] [32]
7.3. FUTURE WORK

[37]. In addition, it improves the quality of the model transformation specification and design, particularly (i) to enable the factoring of complex transformations into modular sub-transformations; (ii) to simplify individual mapping rules of a transformation; (iii) to improve the efficiency of a transformation by removing redundant and duplicated evaluations, and thus simplifying the complex model navigations. In [91], we have identified how metrics of model transformation complexity can be used to guide the choice and application of design patterns to improve the quality and efficiency of model transformation specifications.

Two distinct main categories of model transformation design patterns can be identified:

**Modularisation/decomposition patterns** – concerned with restructuring the transformation specification or design to increase its modularity, and to enable the decomposition of a complex transformation into simpler subtransformations, composed eg., sequentially or in parallel.

**Optimisation patterns** – concerned with increasing the efficiency of transformation execution, by removing redundant or repeated expression evaluations, reducing data storage requirements, etc.

The missing from this research is development of the tool to automatically identifies the appropriate design pattern for particular specification. Future work will consider corporation of this idea into the tool to analysis different metrics and automatically identifies appropriate design patterns for model transformation.

7.3.2 Applying the evaluation framework to other categories of model transformation

The framework in this thesis is applied to four categories of model transformation. However, the framework can be used to evaluate different transformation approaches on other types of transformation problems such as model comparison and model merging. Consequently, the comparison will show an appropriate tool for solving these types of transformation problems. For instance, model comparison compares two models and identifies the differences between them. The resulting model then contains the differences between the two source models. The critical task for these
type of transformation problems is to compare models. Moreover, model merging generates output from integration of source models. The selected tool for this kind of transformation needs to recognize the equivalent elements of the source models and remove the duplicate in the merged model. It is also essential for the tool to provide a flexible merging algorithm to enable further extension of metamodels [96]. One of the potential tools which can be applied to this kind of problem domain, is Epsilon Merging Language (EML) [71]. It has a rule-based language for merging models of diverse metamodels and technologies.

7.3.3 Further extension of evaluation framework

This thesis has introduced an evaluation framework based upon the standard ISO/IEC 9126-1 software quality model. This framework can obviously be expanded to hold a wider range of quality characteristics that provides a more comprehensive measurement of transformation approaches. We realize that some of the properties such as extensibility, adaptability, interoperability require a deeper study and hence this can be undertaken in the future.

Furthermore, the ISO group has collected feedback on the existing standards which lead them to develop a new structure and content for the next generation for these quality standards [142]. The ISO/IEC 9126 defines characteristics, subcharacteristics for measurement of the quality, however it fails to provide guidance for the evaluation process. This is introduced through another series of the standard aligned to the ISO/IEC 14598 [55]. The new version of the standard provides both guidance and requirements for the evaluation process by introducing a mathematical definition for practical measurements of qualities. Any further future work should therefore look at the latest version of the standard ISO quality models and consider more use cases and statistical analysis techniques for the evaluation of different features.

7.3.4 Integration into transML

TransML defines a process for systematic development of model transformations, from requirements, to specifications and designs and implementations [52]. The process is independent of particular transformation languages. Missing from transML are techniques to measure the quality characteristics of transformation specifications and
7.4. CONCLUDING REMARKS

...designs, such as their complexity or modularity. Such measurements could be used to identify potential problems with parts of a transformation (such as excessively complex rules, or highly inter-dependent rules), to identify potential specification errors, and to identify improvements which can be used to resolve these problems.

The potential future work is to integrate the evaluation framework in this thesis with transML by providing metrics that can be applied to the specification and high-level design languages of transML, to evaluate the size, complexity, cohesion and modularity of transformation descriptions in these languages. This will give the capability of language-independent evaluation of transformations.

7.4 Concluding Remarks

This thesis has established a systematic evaluation framework for comparing model transformation approaches, based upon the ISO/IEC 9126 quality characteristics for software systems. Having established the systematic procedural framework for measuring the different features of model transformation, we are then able to apply it for the evaluation of different transformation languages in order to distinguish their benefits. Having done so, the evaluation of transformation languages on the refinement and re-expression case studies are investigated followed by the comparison of the approaches on the migration case study. Finally, the thesis investigated the suitability of transformation approaches on the quality-improvement case study. The evaluation results show clearly the distinctions between the capabilities and suitabilities of different approaches for addressing each form of transformation problem.


[18] B. Bordbar and K. Anastasakis. Uml2alloy: A tool for lightweight modelling of
discrete event systems. In *IADIS International Conference Applied Computing*,

C. Quer. Iso/iec 9126 in practice: what do we need to know? In *Software


[22] L. Briand, S. Morasca, and V. Basili. An operational process for goal-driven
1125, 2002.


case for the transformation tool contest 2010. In P. V. Gorp, S. Mazanek, and

case for the transformation tool contest 2010. http://is.ieis.tue.nl/
staff/pvgorp/share/?page=Con-figureNewSession&vdi=XP-TUe_TTC10_
GrGen.NET_v2.vdi/, 2010.

[26] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. Grose. *Eclipse Mod-

by transformation models. In *Proceedings of the 8th International Workshop
on Model-Driven Engineering, Verification and Validation*, MoDeVVa, pages
9:1–9:8, New York, NY, USA, 2011. ACM.


Appendices
Transformation from Class diagram to relational database

This appendix presents the published transformation of “class diagram to relational database”. The case study has been compared in a fourth chapter.

A.1 Kermeta

The
1 @mainClass "class2RDBMS :: Main"
2 @mainOperation "main"
3
4 package class2RDBMS;
5 class Class2RDBMS
6 {
7     /** The trace of the transformation */
8     reference class2table : Trace<Class, Table>
9     /** Set of keys of the output model */
10    reference fkeys : Collection<FKey>
11
12    operation transform(inputModel : ClassModel) : RDBMSModel is do
13        // Initialize the trace
14        class2table := Trace<Class, Table>.new
15        class2table.create
16        fkeys := Set<FKey>.new
17        result := RDBMSModel.new
18
19        // Create tables
20        getAllClasses(inputModel).select{ c | c.isPersistent }.each{ c |
21            stdio.writeln("Create Table " + c.name)
22            var table : Table init Table.new
23            table.name := c.name
24            class2table.storeTrace(c, table)
25            result.table.add(table)
26        }
27        // Create columns
28        getAllClasses(inputModel).select{ c | c.isPersistent }.each{ c |
29            stdio.writeln("Create Columns for table " + c.name)
A.1. KERMETA

```
createColumns(class2table::getTargetElem(c), c, "")

// Create fkeys
fkeys.each { k | k.createFKeyColumns }

// Create attributes
getAllAttributes(c).each { att |
  createColumnsForAttribute(table, att, prefix)
}

// Create associations
getAllAssociation(c).each { asso |
  createColumnsForAssociation(table, asso, prefix)
}

operation createColumnsForAttribute(table : Table, att : Attribute, prefix : String) is do
  // The type is primitive : create a simple column
  if PrimitiveDataType::isInstance(att.type) then
    var c : Column init Column.new
    c.name := prefix + att.name
    c.type := att.type.name
    table.cols.add(c)
    if att.isPrimary then table.pkey.add(c) end
  else
    var type : Class
    if isPersistentClass(type) then
      // Create a FKey
      var fk : FKey init FKey.new
      fk.prefix := prefix + att.name
      table.fkeys.add(fk)
      fk.references := class2table::getTargetElem(getPersistentClass(type))
    else
      // Recursively include all attributes and asso of the non persistent table
      createColumns(table, type, prefix + att.name)
    end
  end
end

operation createColumnsForAssociation(table : Table, asso : Association, prefix : String) is do
  // The type is persistent
  if isPersistentClass(asso.dest) then
    var fk : FKey init FKey.new
    fk.prefix := prefix + asso.name
    table.fkeys.add(fk)
    fk.references := class2table::getTargetElem(getPersistentClass(asso.dest))
  else
    // Recursively include all attributes and asso of the non persistent table
    createColumns(table, asso.dest, prefix + asso.name)
  end
end
```
A.1. KERMETA

```java
/** *
 * return true if the class or one of its super class is persistent
 */
operation isPersistentClass(cls : Class) : Boolean is do
result := getPersistentClass(cls) != void
end

/** Return the top persistent class for the class given as parameter */
operation getPersistentClass(cls : Class) : Class is do
if cls.isPersistent then
result := cls
else if cls.parent != void then
result := getPersistentClass(cls.parent)
end
end

operation getAllClasses(cmodel : ClassModel) : Class[0..*] is do
result := OrderedSet<Class>.new
cmodel.classifier.each{ c |
    // select only classes (not primitive types)
    if c.getMetaClass == Class then
        var cls : Class cls ?= c // cast the Classifier to Class
        result.add(cls)
    }
end

/** Get the set of attributes of a persistent class */
operation getAllAttributes(cls : Class) : Attribute[0..*] is do
// get inherited attributes
if cls.parent != void then
    result := OrderedSet<Attribute>.new
else
    result := getAllAttributes(cls.parent)
end
// add class attributes (override existing attributes)
cls.attrs.each{ att |
    // check for overriding attributes
    var other_att : Attribute init result.detect{ a | a.name == att.name }
    if other_att != void then
        result.remove(other_att)
    }
result.add(att)
}
// Add attributes from the subclasses
result.addAll(getSubClassesAttributes(cls))
end

operation getSubClassesAttributes(cls : Class) : Attribute[0..*] is do
var model : ClassModel model ?= cls.container
result := OrderedSet<Attribute>.new
// get all the sub-classes
getAllClasses(model).select{ c | c.parent == cls }.each{ c |
    result.addAll(c.attrs)
    result.addAll(getSubClassesAttributes(c))
}
end

/** *
 * Get the set of association of a persistent class
 */
```
A.1. KERMETA

```
operation getAllAssociation(cls : Class) : Association[0..+] is do
  //stdio.writeln("Model = " + cls.container.toString)
  get inherited attributes
  if cls.parent == void then
    result := OrderedSet<Association>.new
  else
    result := getAllAssociation(cls.parent)
  end

  // add class attributes
  model.association.select{asso | asso.src == cls}.each{asso |}
  // check for overriding attributes
  var other_att : Association init result.detect{a | a.name == asso.name}
  if other_att != void then
    result.remove(other_att)
  end
  result.add(asso)

  // Add association defined in subclasses
  result.addAll(getSubClassesAssociation(cls))
end

operation getSubClassesAssociation(cls : Class) : Association[0..+] is do
  var model : ClassModel model ?= cls.container
  result := OrderedSet<Association>.new
  // get all the sub-classes
  getAllClasses(model).select{c | c.parent == cls}.each{c |}
  result.addAll(model.association.select{asso | asso.src == c})
  result.addAll(getSubClassesAssociation(c))
end

class Main
{
  operation main() : Void is do
    var inputModel : ClassModel init loadClassModel
    var transfo : Class2RDBMS init Class2RDBMS.new
    var outputModel : RDBMSModel init transfo.transform(inputModel)

    var repository : EMFRepository init EMFRepository.new
    repository.registerEcoreFile("platform:/resource/fr.irisa.triskell.kermeta.samples/class2RDBMS.metamodels/RDBMS.ecore")
    var root : Resource init repository.createResource("platform:/resource/fr.irisa.triskell.kermeta.samples/class2RDBMS/models/out.xmi")
```
A.2. ATL

// Define a Root ...
resource.instances.add(outputModel)
resource.save()
rescue (e : kermeta::exceptions::ResourceLoadException)
metamodelRegistration(e)
end

operation loadClassModel() : ClassModel is do
var repository : EMFRepository
init EMFRepository.new
repository.registerEcoreFile("platform:/resource/fr.iris.a Triaskell.kermeta.
samples/class2RDBMS/metamodels/ClassMM.ecore")
var resource : Resource
init resource.initializeResource("platform:/resource/fr.
iris.a Triaskell.kermeta.samples/class2RDBMS/metamodels/ClassMM.ecore",
"platform:/resource/mtip/class2RDBMS/metamodels/ClassMM.ecore")
resource.load()

// result ?= resource.instances.one
from var it : Iterator<Object>
init resource.instances.iterator
until it.isOff
loop
var next : Object
init it.next
if (ClassModel.isInstance(next))
result ?= next
end
end

operation metamodelRegistration(e : kermeta::exceptions::ResourceLoadException) is do
stdiowriteln("Problem when loading the model. ENSURE YOU HAVE REGISTERED THE
METAMODELS!
"
+"You can find additional information about this sample on Kermeta web
site : http://kermeta.org/examples")
raise e
end

A.2  ATL

module Class2Relational;
create OUT : Relational from IN : Class;
uses strings;

-- if there is a configuration problem with the library string,
delete the two firstToLower() operations and you no longer need the library "strings"
-- the transformation will execute
-- inheritance not supported yet
-- issue : choose an object--id Type (Integer , String?).
-- We choose Integer here, assuming this type is defined in the source model.
A.2. ATL

17 -- global variable
18 -- context
19 helper def: objectIdType : Relational!Type =
20 Class!DataType.allInstances()->select(e | e.name = 'Integer')->first();
21
22 rule Class2Table {
23     from c : Class!Class
to out : Relational!Table {
    name <- c.name,
    -- Columns are generated from Attributes in another rule not explicitly called here!
    col <- Sequence {key}->union(e.attr->select(e | not e.multiValued)),
    key <- Set {key}
},
    key : Relational!Column {
        name <- 'objectId',
        type <- thisModule.objectIdType
}
}
23
24 rule DataType2Type {
25     from dt : Class!DataType
to out : Relational!Type {
    name <- dt.name
}
}
24
25 rule DataTypeAttribute2Column {
26     from a : Class!Attribute {
        a.type.oclIsKindOf(Class!DataType) and not a.multiValued
    }
to out : Relational!Column {
    name <- a.name,
    type <- a.type
},
    explicit use of implicit tracking links (first expected syntax, then present actual syntax)
    owner <- [Class2Type.key]a.owner
    owner <- thisModule.resolveTemp(a.owner, 'key')
}
27
28 rule MultiValuedDataTypeAttribute2Column {
29     from a : Class!Attribute {
        a.type.oclIsKindOf(Class!DataType) and a.multiValued
    }
to out : Relational!Table {
    name <- a.owner.name + ' '+ a.name,
    col <- Sequence {id, value}
},
    id : Relational!Column {
        name <- a.owner.name.firstToLower() + 'Id',
        type <- thisModule.objectIdType
},
    value : Relational!Column {
        name <- a.name,
        type <- a.type
}
}
30
31 rule ClassAttribute2Column {
A.3. QVT-R

83 from a : Class\(!\)Attribute {
84 a . type .oclIsKindOf(Class\(!\)Class) and not a .multiValued
85 }
86 to
87 forKey : Relational\(!\)Column {
88 name <= a .name + 'Id',
89 type <= thisModule .objectIdType
90 }}
91
92 rule MultiValuedClassAttribute2Column {
93 from
94 a : Class\(!\)Attribute {
95 a .type .oclIsKindOf(Class\(!\)Class) and a .multiValued
96 }
97 to
98 t : Relational\(!\)Table {
99 name <= a .owner .name + '_' + a .name,
100 col <= Sequence {id , forKey}
101 ),
102 id : Relational\(!\)Column {
103 name <= a .owner .name .firstToLower() + 'Id',
104 type <= thisModule .objectIdType
105 }
106 forKey : Relational\(!\)Column {
107 name <= a .name + 'Id',
108 type <= thisModule .objectIdType
109 }
110
111 }
112 }

A.3 QVT-R

1 transformation umlToRdbms(uml:SimpleUML, rdbms:SimpleRDBMS)
2 {
3 key Table (name, schema);
4 key Column (name, owner); // owner:Table opposite column:Column
5 key Key (name, owner); // key of class Key;
6 // owner:Table opposite key:Key
7 top relation PackageToSchema // map each package to a schema
8 {
9 pn: String;
10 checkonly domain uml p:Package {name=pn};
11 enforce domain rdbms s:Schema {name=pn};
12 }
13 top relation ClassToTable // map each persistent class to a table
14 {
15 cn , prefix: String;
16 checkonly domain uml c:Class {namespace=p:Package {}},
17 kind='Persistent', name=cn ,
18 enforce domain rdbms t:Table {schemas=s:Schema {}}, name=cn ,
19 columns=1:Column {name=cn +" importantly", columnname=column},
20 key=k:Key {name=cn +" is", column=column};
21 when {
22 PackageToSchema(p, s);
23 }
24 where {
25 prefix = "";
26 AttributeToColumn(c, t, prefix);  
27 }
28 }
29 relation AttributeToColumn
30 {
31 checkonly domain uml c:Class {};
32277
enforce domain rdbms t:Table {};
primitive domain prefix:String;
where {
  PrimitiveAttributeToColumn(c, t, prefix);
  ComplexAttributeToColumn(c, t, prefix);
  SuperAttributeToColumn(c, t, prefix);
}

relation PrimitiveAttributeToColumn
{
an , pn , cn , sqtype : String ;
checkonly domain uml c: Class { attribute=a: Attribute {name=an ,
type=PrimitiveDataType {name=pn}}};
enforce domain rdbms t:Table {column=c1:Column {name=cn ,
type=sqtype}};
primitive domain prefix:String;
where {
  cn = if (prefix = '') then an else prefix+''+an endif ;
sqtype = PrimitiveTypeToSqlType(pn);
}

relation ComplexAttributeToColumn
{
an , newPrefix : String ;
checkonly domain uml c: Class { attribute=a: Attribute {name=an ,
type=t: Class {}}};
enforce domain rdbms t:Table {};
primitive domain prefix:String;
where {
  newPrefix = prefix+''+an ;
  AttributeToColumn(t, t, newPrefix);
}

relation SuperAttributeToColumn
{
checkonly domain uml c: Class {general=sc: Class {}};
enforce domain rdbms t:Table {};
primitive domain prefix:String;
where {
  AttributeToColumn(sc , t , prefix);
}

// map each association between persistent classes to a foreign key

top relation AssocToFKey
{
srcTbl , destTbl : Table ;
pKey : Key ;
an , scn , dcn , fkn , fcn : String ;
checkonly domain uml a: Association {namespace=p:Package {},
namespace=an ,
source=sc: Class {kind='Persistent ' ,name =scn },
destination=dc : Class {kind='Persistent ' ,name =dcn} }

enforce domain rdbms fk:ForeignKey {schema=s:Schema {},
nname=fkn ,
owner=srcTbl ,
column=fc : Column {name=fcn ,type='NUMBER' ,owner=srcTbl },
refersTo=pKey
}
when {
  PackageToSchema(p, s);
  ClassToTable(sc , srcTbl );
  ClassToTable(dc , destTbl );
pKey = destTbl.key ;
}
where {
  fkn=scn+'''+an+'''+dcn;
}
99  fcn=fkn+'1id';
100 }
101 }

102 function PrimitiveTypeToSqlType(primitiveType : String) : String
103 {
104 if (primitiveType=='INTEGER')
105 then 'NUMBER'
106 else if (primitiveType=='BOOLEAN')
107 then 'BOOLEAN'
108 else 'VARCHAR'
109 endif;
110 }
111 }
112 }

A.4 Viatra

1 namespace uml2reldb.transformations;
2 import uml2.metamodel;
3 import reldb.metamodel;
4 import uml2reldb.metamodel;
5 import datatypes;
6 machine uml2reldb_xform
7 {
8 as function models / 1 ;
9 rule main (in UMLStr , in RefStr , in DBStr ) = seq
10 {
11 call initModels ( UMLStr , RefStr , DBStr );
12 forall C below models ("uml") with apply class2tableR (C) do skip ;
13 forall C below models ("uml") , A below models ("uml") with
14 apply attr2columnR (C, A) do skip ;
15 forall A below models ("uml") with apply attrOf StringTypeR (A) do skip ;
16 forall A below models ("uml") with apply attrOfIntTypeR (A) do skip ;
17 forall A below models ("uml") with apply assoc2tableR (A) do skip ;
18 }
19 rule initModels ( in UMLStr , in RefStr , in DBStr ) =
20 let UMLModel = undef in
21 let DBModel = undef in
22 let RefModel = undef in
23 seq
24 call modelManagement . lookupAndCreate ("uml2" , UMLStr ,
25 "LOOKUP_NOCLEAN" , UMLModel );
26 update models ("uml") = UMLModel ;
27 call modelManagement . lookupAndCreate ("uml2reldb.models" , RefStr ,
28 "CLEANORCREATE" , RefModel );
29 update models ("ref") = RefModel ;
30 call modelManagement . lookupAndCreate ("reldb.models" , DBStr ,
31 "CLEANORCREATE" , DBModel );
32 update models ("db") = DBModel ;
33 }
34 g rule class2tableR ( in Cls ) = 
35 precondition pattern lhs (Cls , ClsNM ) =
36 Class (Cls) below models ("uml ");
37 NamedElement . name (N1 , Cls , ClsNM );
38 String (ClsNM ) below models ("uml");
39 }
40 action 
41 let T = undef in
42 let R = undef in
43 let RS = undef in
44 let RT = undef in seq 
45 call createsNewTable ( value (ClsNM ) , T);
46 call createPrimaryKeywordTable (T);
47 new ( class2table (R) in models ("ref"));
48 new (class2table . srcRef (RS , R , Cls ));
49 new (class2table . trgRef (RT , R , T ));
50 print (" Class "+ fqn(Cls )+" -> Table "+ fqn(T )+"\n");
51 }
52 }
53 }
54 grule attr2columnR (in Cls , in Attr ) = {
55 precondition pattern lhs (Cls , Attr , NMAttr , Tab) = {
56 Class (Cls) below models (" uml ");
57 StructuredClassifier . ownedAttribute (F , Cls , Attr );
58 Property (Attr ) below models (" uml");
59 NamedElement . name (N1 , Attr , NMAttr );
60 String (NMAttr) below models (" uml");
61 class2table . srcRef (RS , R , Cls );
62 class2table (R) below models (" ref");
63 class2table . trgRef (RT , R , Tab );
64 Table (Tab) below models ("db");
65 neg pattern assocProperty (Attr ) = {
66 Property (Attr ) below models (" uml");
67 Association . memberEnd (ME ,Asc , Attr );
68 Association (Asc ) below models (" uml");
69 }}
70 action {
71 let Col = undef in
72 let R = undef in
73 let RT = undef in seq {  
74 let RT = undef in seq {
75 call createNewColumn ( value (NMAttr), Tab , Col );
76 new (attr2column (R) in models (" ref"));
77 new (attr2column . srcRef (RS , R , Attr ));
78 new (attr2column . trgRef (RT , R , Col ));
79 print (" Attribute ":+ fqn(Attr )+" -> Column ":+ fqn(Col )+"\n");
80 }
81 }
82 }
83 grule attrOfStringTypeR (in Attr ) = {
84 precondition pattern lhs (Attr , Col ) = {
85 Property (Attr ) below models (" uml");
86 TypedElement . type (TP , Attr , DTType );
87 PrimitiveType (DTType) below models (" uml");
88 attr2column . srcRef (RS , R , Attr );
89 attr2column (R) below models (" ref");
90 attr2column . trgRef (RT , R , Col );
91 Column (Col) below models ("db");
92 check (name (DTType) == " String")
93 }  
94 action {
95 call createNewColumnType (Col , " VARCHAR (50) ");
96 }
97 }
98 grule attrOfIntTypeR (in Attr ) = {
99 precondition pattern lhs (Attr , Col ) = {
100 Property (Attr ) below models (" uml");
101 TypedElement . type (TP , Attr , DTType );
102 PrimitiveType (DTType) below models (" uml");
103 attr2column . srcRef (RS , R , Attr );
104 attr2column (R) below models (" ref");
105 attr2column . trgRef (RT , R , Col );
106 Column (Col) below models ("db");
107 check (name (DTType) == " int")
108 }
109 action {
110 call createNewColumnType (Col , " INTEGER ");
111 }
112 }
113 grule assoc2tableR (in Assoc ) = {
114 precondition pattern lhs (Assoc , AE1 , AE2 , SrcC , TrgC , SrcTab , TrgTab ) = {

280
A.4. VIATRA

115 Association (Assoc) below models ("uml");
116 Association.memberEnd(C1, Assoc, AE1);
117 Property(AE1) below models ("uml");
118 TypedElement.type(T1, AE1, SrcC);
119 Class(SrcC) below models ("uml");
120 class2table.srcRef(RS1, R1, SrcC);
121 class2table trgRef(RT1, R1, TrgTab);
122 Table(SrcTab) below models ("db");
123 Association.memberEnd(C2, Assoc, AE2);
124 Property(AE2) below models ("uml");
125 TypedElement.type(T2, AE2, TrgC);
126 Class(TrgC) below models ("uml");
127 class2table.srcRef(RS2, R2, TrgC);
128 class2table trgRef(RT2, R2, TrgTab);
129 Table(TrgTab) below models ("db");
130 }
131 action {
132 let T = undefined;
133 let myIdCol = undefined;
134 let SrcCol = undefined;
135 let PK1 = undefined;
136 let FKSrc = undefined;
137 let TrgCol = undefined;
138 let FKTrg = undefined;
139 let R = undefined;
140 let RS = undefined;
141 let RT = undefined;
142 let RT = undefined in seq{
143 call createNewTable (name(AE1) + "+ name(SrcC) + "+ name(AE2) + "+
144 name(TrgC), T);
145 call createNewColumn ("id", T, MyIdCol);
146 call createNewColumnType (MyIdCol, "INTEGER");
147 new(Table.pkey(PK1, T, MyIdCol));
148 call createNewColumn (name(SrcC) + " id", T, SrcCol);
149 call createNewColumnType (SrcCol, "INTEGER");
150 call createNewFKey(T, SrcCol, SrcTab, FKSrc);
151 call createNewColumn (name(TrgC) + " id", T, TrgCol);
152 call createNewColumnType (TrgCol, "INTEGER");
153 call createNewFKey(T, TrgCol, TrgTab, FKTrg);
154 new(assoc2table(R) in models ("ref"));
155 new(assoc2table.srcRef(RS, R, Assoc));
156 new(assoc2table.trgRef(RT, R, T));
157 }
158 }
159 }
160 }
161 rule createNewTable (in TabName, out T) =
162 let N1 = " undefined in
163 let Str = " undefined in seq{
164 new(Table(T) in models ("db"));
165 new(datatypes.String(Str) in models ("db");
166 setValue(Str, TabName);
167 new(Table.tblName(N1, T, Str));
168 }
169 }
170 rule createPrimaryKeyInTable (in T) =
171 let PKCol = undefined in seq{
172 print(fqn(T) + "\n");
173 call createNewColumn ("id", T, PKCol);
174 call createNewColumnType (PKCol, "INTEGER");
175 new(Table.pkey(PK1, T, PKCol));
176 }
177 rule createNewColumn (in ColName, in Tab, out Col) = seq{
178 new(Column(Col) in models ("db"));
179 new(datatypes.String(StrNM) in models ("db");
180 setValue(StrNM, ColName);
181 new(Column.colName(N3, Col, StrNM));
A.5 UML-RSDS

C1 “For each persistent attribute in the source model there is a unique column in the target model, of corresponding type”:

$$\forall a : \text{Attribute} \cdot a.\text{owner}.\text{kind} = \text{'Persistent'} \implies \exists_1 c_1 : \text{Column} \cdot c_1.\text{rdbId} = a.\text{umlId} \land c_1.\text{name} = a.\text{name} \land c_1.\text{kind} = a.\text{kind} \land (a.\text{type}.\text{name} = \text{'INTEGER'} \implies c_1.\text{type} = \text{'NUMBER'}) \land (a.\text{type}.\text{name} = \text{'BOOLEAN'} \implies c_1.\text{type} = \text{'BOOLEAN'}) \land (a.\text{type}.\text{name} \neq \text{'INTEGER'} \land a.\text{type}.\text{name} \neq \text{'BOOLEAN'} \implies c_1.\text{type} = \text{'VARCHAR'})$$

C2 “For each persistent class in the source model, there is a unique table representing the class in the target model, with columns for each owned attribute”:

$$\forall c : \text{Class} \cdot c.\text{kind} = \text{'Persistent'} \implies \exists_1 t : \text{Table} \cdot t.\text{rdbId} = c.\text{umlId} \land t.\text{name} = c.\text{name} \land t.\text{kind} = \text{'Persistent'} \land \text{Column}[c.\text{attribute}.\text{umlId}] \subseteq t.\text{column}$$

For a set $x$, the notation $\text{Entity}[x]$ refers to the set of $\text{Entity}$ objects with primary key (in this case $\text{rdbId}$) value in $x$, it can be implemented in Java by maintaining a map from the key values to nodes. In OCL it would be expressed as

$$\text{Entity}.\text{allInstances}() \rightarrow \text{select}(x \rightarrow \text{includes}(\text{id}))$$

For a single value $y$, $\text{Entity}[y]$ denotes
Entity.allInstances()→select(id = y)→any()

C3 “For each root class in the source model there is a unique primary key in the target model”:

∀ c : Class · c.kind = ‘Persistent’ and c.general = {} implies
∃ 1 k : Key · k.rdbId = c.umlId + “_Pk” and k.name = c.name + “_Pk” and
k.owner = Table[c.umlId] and k.kind = ‘PrimaryKey’ and
∃ 1 cl : Column · cl.rdbId = c.umlId + “_Id” and
cl.name = c.name + “_Id” and cl.type = ‘NUMBER’ and
cl : k.column and cl.kind = ‘PrimaryKey’ and
cl : k.owner.column

C4 “For each association in the source model, there is a unique foreign key representing it in the target model”:

∀ a : Association and a.kind = ‘Persistent’ implies
∃ 1 fk : ForeignKey · fk.rdbId = a.umlId + “_Fk” and
fk.name = a.name + “_Fk” and
fk.owner = Table[a.source.umlId] and
fk.kind = ‘association’ and
fk.refersTo = Table[a.destination.umlId] and
∃ cl : Column · cl.rdbId = a.umlId + “_Ref” and
cl.name = a.name + “_Ref” and
cl : fk.column and cl.kind = ‘ForeignKey’ and
cl.type = ‘NUMBER’ and
cl : fk.owner.column

C5 “If c is a subclass of d, all columns of d’s table are included in c’s table”:

∀ c, d : Class · c.kind = ‘Persistent’ and d : c.general implies
Table[d.umlId].column ⊆ Table[c.umlId].column

C6 “For each package in the source model, there is a unique schema representing it
There are also dual constraints, expressing that the only elements of the target model are those derived from the source model by constraints $C_1$ to $C_6$. Dual constraint $C_7$ expresses that the only columns of a table are those defined by $C_2$, $C_3$, $C_4$ and $C_5$.

Constraint $C_8$ is the dual of $C_2$, there are also dual constraints $C_9$ of $C_4$ and $C_10$ of $C_6$. 
Input model for UML-RSDS in the Migration case study

The migration case study has been investigated in Chapter 5. This section presents the input model for UML-RSDS for this case study. The creation of a particular source model in UML-RSDS is defined by an operation of the transformation metaclass.

\[\text{init}()\]
\[\text{post:}\]
\[\text{ag} : \text{ActivityGraph} \land \text{ag.name} = \text{"ag"} \land \]
\[\text{fs} : \text{FinalState} \land \text{fs.name} = \text{"fs"} \land \]
\[\text{is} : \text{Pseudostate} \land \text{is.name} = \text{"is"} \land \text{is.kind} = \text{initial} \land \]
\[\text{as1} : \text{ActionState} \land \text{as1.name} = \text{"Request service"} \land \]
\[\text{as2} : \text{ActionState} \land \text{as2.name} = \text{"Pay"} \land \]
\[\text{as3} : \text{ActionState} \land \text{as3.name} = \text{"Collect order"} \land \]
\[\text{as4} : \text{ActionState} \land \text{as4.name} = \text{"Take order"} \land \]
\[\text{as5} : \text{ActionState} \land \text{as5.name} = \text{"Deliver order"} \land \]
\[\text{as6} : \text{SimpleState} \land \text{as6.name} = \text{"Restock"} \land \]
\[\text{as7} : \text{ActionState} \land \text{as7.name} = \text{"Fill order"} \land \]
\[\text{t} : \text{Type} \land \]
\[\text{of1} : \text{ObjectFlowState} \land \text{of1.name} = \text{"Placed Order"} \land \text{of1.type} = \text{t} \land \]
\[\text{of2} : \text{ObjectFlowState} \land \text{of2.name} = \text{"Entered Order"} \land \text{of2.type} = \text{t} \land \]
\[\text{of3} : \text{ObjectFlowState} \land \text{of3.name} = \text{"Filled Order"} \land \text{of3.type} = \text{t} \land \]
\[\text{of4} : \text{ObjectFlowState} \land \text{of4.name} = \text{"Delivered Order"} \land \text{of4.type} = \text{t} \land \]
\[\text{ds1} : \text{Pseudostate} \land \text{ds1.name} = \text{"ds1"} \land \text{ds1.kind} = \text{junction} \land \]
ds2 : Pseudostate & ds2.name = "ds2" & ds2.kind = junction &
dc1 : Pseudostate & dc1.name = "dc1" & dc1.kind = fork &
dc2 : Pseudostate & dc2.name = "dc2" & dc2.kind = join &
be1 : BooleanExpression & be1.language = "text" & be1.body = "in stock" &
g1 : Guard & g1.name = "g1" & g1.expression = be1 &
be2 : BooleanExpression & be2.language = "text" & be2.body = "not in stock" &
g2 : Guard & g2.name = "g2" & g2.expression = be2 &
t0 : Transition & t0.name = "t0" & t0.source = is & t0.target = as1 &
t1 : Transition & t1.name = "t1" & t1.source = as1 & t1.target = dc1 &
t2 : Transition & t2.name = "t2" & t2.source = dc1 & t2.target = as2 &
t3 : Transition & t3.name = "t3" & t3.source = as2 & t3.target = dc2 &
t4 : Transition & t4.name = "t4" & t4.source = dc1 & t4.target = of1 &
t5 : Transition & t5.name = "t5" & t5.source = of1 & t5.target = as4 &
t6 : Transition & t6.name = "t6" & t6.source = as4 & t6.target = of2 &
t7 : Transition & t7.name = "t7" & t7.source = of2 & t7.target = ds1 &
t8 : Transition & t8.name = "t8" & t8.source = ds1 & t8.target = ds2 &
t8.guard = { g1 } &
t9 : Transition & t9.name = "t9" & t9.source = ds1 & t9.target = as6 &
t9.guard = { g2 } &
tr : Event & tr.name = "receive stock" &
t10 : Transition & t10.name = "t10" & t10.source = as6 & t10.target = ds2 &
t10.trigger = { tr } &
t11 : Transition & t11.name = "t11" & t11.source = ds2 & t11.target = as7 &
t12 : Transition & t12.name = "t12" & t12.source = as7 & t12.target = of3 &
t13 : Transition & t13.name = "t13" & t13.source = of3 & t13.target = dc2 &
t14 : Transition & t14.name = "t14" & t14.source = dc2 & t14.target = as5 &
t15 : Transition & t15.name = "t15" & t15.source = as5 & t15.target = of4 &
t16 : Transition & t16.name = "t16" & t16.source = of4 & t16.target = as3 &
t17 : Transition & t17.name = "t17" & t17.source = as3 & t17.target = fs &
cs : CompositeState & cs.name = "cs" &
cs.subvertex = { is, fs, as1, as2, as3, as4, as5, as6, as7, of1, of2, of3, of4, ds1, ds2, dc1, dc2 } &
ag.top = cs &
ag.transitions = { t0, t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17 } &
p1 : Partition & p1.name = "p1" &
p1.contents = { as1, t1, dc1, t4, t2, as2, t16, as3 } &
\begin{verbatim}
p2 : Partition & p2.name = "p2" & p2.contents = { t5, as4, t6, t13, dc2, t14, as5, t15 } &
p3 : Partition & p3.name = "p3" & p3.contents = { t7, ds1, t8, t9, as6, as7, ds2, t10, t11, t12 } &
ag.partition = { p1, p2, p3 }
\end{verbatim}
This appendix presents the solutions to the “class diagram Restructuring” case study in Chapter 6.

C.1 UML-RSDS

\[(C1): \]
\[
a : specialization\_specific.ownedAttribute \&
\]
\[
specialization\_size > 1 \&
\]
\[
specialization\_specific\rightarrow\forall(ownedAttribute\rightarrow\exists(b \mid b.name = a.name \& b.type = a.type)) \Rightarrow
\]
\[
a : ownedAttribute \&
\]
\[
specialization\_specific.ownedAttribute\rightarrow\exists(name = a.name)\rightarrow\text{isDeleted()}
\]

on Entity.
(C2):
a : specialization.specific.ownedAttribute &

\[ v \rightarrow \text{select}(\text{specific.ownedAttribute} \rightarrow \exists (b \mid b.\text{name} = a.\text{name} \& b.\text{type} = a.\text{type})) \& v.\text{size} > 1 \Rightarrow \]

Entity→exists(e \mid e.\text{name} = \text{name} + "2" + a.\text{name} & a : e.\text{ownedAttribute} & e.\text{specialization} = v \& Generalization→exists(g \mid g : specialization \& g.specific = e)) \& v.specific.ownedAttribute→select(name = a.\text{name})→\text{isDeleted()}

on Entity.

(C3):
a : ownedAttribute &

generalization.size = 0 &

\[ v = \text{Entity} \rightarrow \text{select}(\text{generalization.size} = 0 \& \text{ownedAttribute} \rightarrow \exists (b \mid b.\text{name} = a.\text{name} \& b.\text{type} = a.\text{type})) \& v.\text{size} > 1 \Rightarrow \]

Entity→exists(e \mid e.\text{name} = \text{name} + "3" + a.\text{name} & a : e.\text{ownedAttribute} & v.\text{ownedAttribute}→select(name = a.\text{name})→\text{isDeleted()} \& v→\text{forall}(c \mid \text{Generalization} \rightarrow \exists (g \mid g : e.\text{specialization} \& g.specific = c)))

on Entity.

C.2 GrGen

1 rule rule1 {
  2 c : Class;
  3 \$SuperOf(c, g1) ; \$SuperOf(c, g2);
  4 g1 : Class → ownedAttribute→ a1 : Property → type→ t : Type;
  5 g2 : Class → ownedAttribute→ a2 : Property;
  6 : SameAttribute(a1, a2);
  7 negative {  
  8 g3 : Class;
  9 \$SuperOf(c, g3);
  10 g1;  
  11 negative {  
  12 g3 → ownedAttribute→ a3 : Property;
  13 : SameAttribute(a1, a3);
  14  }  
  15 }  
  16 modify {  
  17 c → ownedAttribute→ a4 : Property → type→ t;  
  18 eval {
C.2. GRGEN

```plaintext
19    a4 .name = a1 .name;
20 }
21 exec (RemoveAttributeFromSubclasses(c, a4) => [createInverseEdges]);
22 }
23 }

1 rule rule2 {
2  c .Class;
3 } SuperOf(c, g1) . SuperOf(c, g2) . SuperOf(c, g3);
4 g1 .Class .ownedAttribute => a1 : Property . type => t : Type;
5 g2 .Class .ownedAttribute => a2 : Property;
6 } SameAttribute(a1, a2);
7 g3 .Class;
8 } negative {
9    g3 .ownedAttribute => a3 : Property;
10 } SameAttribute(a1, a3);
11 } modify {
12    c .specialisation => Generalization . specific => c1 .Class;
13    c1 .ownedAttribute => a4 : Property . type => t;
14 } eval {
15    a4 .name = a1 .name;
16 }
17 exec (AddIntermediateClassAndRemoveAttributeFromSubclasses(c, a4, c1) => [createInverseEdges]);
18 }
19 }
20 }

1 rule rule3 {
2  g1 .IsRoot(g1) . IsRoot(g2);
3 g1 .Class .ownedAttribute => a1 : Property . type => t : Type;
4 g2 .Class .ownedAttribute => a2 : Property;
5 } SameAttribute(a1, a2);
6 } modify {
7    c .specialisation => Generalization . specific => c1 .Class;
8 } eval {
9    a3 .name = a1 .name;
10 }
11 exec (AttributeToRootClassesAndMakeSubclass(a3, c) => [createInverseEdges]);
12 }
13 }

1 pattern SameAttribute(a1 : Property, a2 : Property) {
2 independent {
3    if { a1 .name == a2 .name; }
4    a1 .type => t : Type <> a2 .type;
5    6 }
7 pattern SuperOf(c1 : Class, c2 : Class) {
8    c1 .specialisation => Generalization . specific => c2;
9 }
10 }
11 rule RemoveAttributeFromSubclasses(c : Class, attr : Property) {
12    iterated {
13    c .specialisation => Generalization . specific => g .Class . ownedAttribute => a : Property;
14 } SameAttribute(a, attr);
15 } modify {
16    delete(a);
17 }
18 }
19 }
20 }

1 rule AddIntermediateClassAndRemoveAttributeFromSubclasses(super : Class, attr : Property, intermediate : Class) {
2 iterated {

C.2. GRGEN

1 rule AttributeToRootClassesAndMakeSubclass(a: Property, super: Class) {
  iterated {
    IsRoot(root) ; root : Class ;
    root : ownedAttribute -> a : Property ;
    super ; // super <> root
    : SameAttribute(a, attr) ;
    modify {
      super : specialisation -> gOld : Generalization : specific -> g : Class : ownedAttribute -> a : Property ;
      intermediate ; // <> super <> g
      : SameAttribute(a, attr) ;
      modify {
        delete(gOld) ;
        intermediate : specialisation -> gNew : Generalization : specific -> g ;
        delete(a) ;
      }
    }
  }
}

1 rule createInverseEdges {
  alternative {
    specialisation {
      src : Node : specialisation -> trg : Node ;
      negative {
        trg : general -> src ;
      }
      modify {
        trg : general -> src ;
      }
    }
    general {
      src : Node : general -> trg : Node ;
      negative {
        trg : specialisation -> src ;
      }
      modify {
        trg : specialisation -> src ;
      }
    }
    generalization {
      src : Node : generalization -> trg : Node ;
      negative {
        trg : specific -> src ;
      }
      modify {
        trg : specific -> src ;
      }
    }
    specific {
      src : Node : specific -> trg : Node ;
      negative {
        trg : generalization -> src ;
      }
      modify {
        trg : generalization -> src ;
      }
    }
    modify {}
C.3. KERMETA

C.3 Kermeta

ClassMgr

```java
1 class Container {
2   attribute ref : String
3   attribute data : Set<String>
4   attribute proc : Boolean
5   operation create(r : String) : Void is do
6     ref := r
7     data := Set<String>.new
8     proc := false
9   end
10  operation add(c : String) : Void is do
11     data.add(c)
12   end
13 }
14 class ClassMgr
15 {
16   attribute cont : Set<Container>
17   operation newSet(ref : String) : Void is do
18     if cont == void then
19       cont := Set<Container>.new
20     end
21     var c : Container init Container.new
22     c.create(ref)
23     cont.add(c)
24   end
25   operation addClass(ref : String, cl : String) : Void is do
26     cont.each(co |  
27       if co.ref == ref then
28         co.add(cl)
29     end
30   )
31   end
32   operation getLargestSet() : Container is do
33     //stdio.writeln("ClsMgr::getLargestSet")
34     var curr : Container
35     if cont != void then
36       cont.each(co |  
37         if co.proc == false and curr == void then
38          curr := co
39        else
40          if curr != void then
41            if co.proc == false and co.data.size() > curr.data.size() then
42              curr := co
43            end
44            //stdio.writeln("Data Size: " + curr.data.size().toString())
45          end
46        end
47     )
48   end
49   if curr == void then
50     curr := Container.new
51     curr.data := Set<String>.new
52   end
53   curr.proc := true
54   result := curr
55 } end
```

292
C.3. KERMETA

ModelAspects

1 aspect class model {  
2   operation getClassFromString(name : String) : Class is do  
3     //stdio.writeln("getClassFromString")  
4     result := void  
5     self.Element.each(c |  
6        //stdio.writeln("checking element : " + c.name)  
7        if c instanceof Class and c.name == name then  
8           //stdio.writeln("Found Class: " + name)  
9           var cl : Class  
10          cl ?= c  
11          result := cl  
12      end  
13   end  
14 }  
15  
16 aspect class Property {  
17   method equals(compared : Object) : Boolean is do  
18      if void == compared then  
19         result := (self.name == void)  
20      else  
21         var o : Property  
22         o ?= compared  
23         result := false  
24         if o.name.equals(self.name) then  
25            if o.type != void and o.type.name != void then  
26               if o.type.name.equals(self.type.name) then  
27                  result := true  
28               end  
29            end  
30         else  
31            //stdio.writeln("Property missing type info: " + o.name)  
32         end  
33      end  
34   end  
35 }  
36  
37 aspect class Class {  
38   attribute done1 : Integer  
39   attribute done2 : Integer  
40   attribute done3 : Integer  
41   operation checkCommon(o : Class) : Boolean is do  
42      result := false  
43      self.ownedAttribute.each(p |  
44         o.ownedAttribute.each(op |  
45            if p.equals(op) then  
46               result := true  
47         end  
48      end  
49   end  
50 }  
51  
52 aspect class Property {  
53   operation getRefProperty(rp : String) : Property is do  
54      var prop : Property  
55      self.ownedAttribute.each(p |  
56         if p.name.equals(rp) then  
57            prop := p  
58      end  
59   end  
60   result := prop  
61 }  
62  
63 aspect class Property {  
64   operation removeProperty(rp : String) : Void is do  
65      var prop : Property  
66      self.ownedAttribute.each(p |  
67         if p.name.equals(rp) then  
68            result := prop  
69      end  
70 }
C.3. KERMETA

```kem
class PContainer {
  attribute ref : String
  attribute data : Set<String>
  attribute proc : Boolean
  operation create(r : Property) : Void is do
    ref := r.name
    data := Set<String>.new
    proc := false
  end
}
```

PropertyMgr

```kem
1 /* $Id$ */
2 + Creation : February 12, 2012
3 + Licence : EPL
4 + Copyright:
5 + Authors:
6 + suresh
7 */
8 package KCL;
9
10 require kermeta
11 using kermeta::standard
12
13 class PContainer {[
14   attribute ref : String
15   attribute data : Set<String>
16   attribute proc : Boolean
17   operation create(r : Property) : Void is do
18     ref := r.name
19     data := Set<String>.new
20     proc := false
21   end
22 ]
```
C.3. KERMETA

operation add(c : String) : Void is do
  data.add(c)
end

class PropertyMgr
{
  attribute cont : Set<PContainer>
  operation newSet(ref : Property) : Void is do
    if cont == void then
      cont := Set<PContainer>.new
    end
    var c : PContainer init PContainer.new
    c.create(ref)
    cont.add(c)
  end
  operation addClass(ref : Property, cl : Class) : Void is do
    cont.each{co|
      if co.ref == ref.name then
        co.add(cl.name)
      end
    }
  end
  operation MarkLargest() : Void is do
    getLargestSet().proc := true
  end
  operation getCommonProperties() : Set<String> is do
    var props : Set<String> init Set<String>.new
    var pcont : PContainer
    pcont := getLargestSet()
    if pcont.data.size() > 0 then
      props.add(pcont.ref)
      cont.each{pc|
        if not pcont.ref.equals(pc.ref) then
          if pcont.data.containsAll(pc.data) then
            props.add(pc.ref)
          end
        end
      }
    end
  end
  operation getLargestSet() : PContainer is do
    var curr : PContainer
    cont.each{co|
      if co.proc == false and curr == void then
        curr := co
      else
        if curr != void then
          if co.proc == false and co.data.size() > curr.data.size() then
            curr := co
          end
        end
      end
    }
    if curr == void then
      curr := PContainer.new
      curr.data := Set<String>.new
    end
    result := curr
  end
}

ResManager

class ResManager
operation LoadModel(modelName : String) : model is do
    result := void
    var repository : ResourceSet init ResourceSet.new
    var resource : Resource init repository.createResource(modelName, "platform:/resource/QIKermeta/metamodel/KCL.ecore")
    do
        resource.load(void)
        if resource.getContents().size() > 0 then
            result := resource.getContents().first
        end
    rescue(e : ResourceLoadException)
        // stdio.writeln("Resource Load Exception")
    end
end

operation Save(modelName : String, m : model) : Void is do
    var repository : ResourceSet init ResourceSet.new
    var resource : Resource init repository.createResource(modelName, "platform:/resource/QIKermeta/metamodel/KCL.ecore")
    resource.getContents().add(m)
    resource.save(void)
end

class Rule1
    attribute root : KCL::model
    operation addPropertiesSuper(c : Class, prop : Set<Property>) : Void is do
        prop.each[p]
        c.ownedAttribute.add(p)
    end

    operation remPropertiesChildren(c : Class, prop : Set<Property>) : Void is do
        c.specialisation.each[s]
        var sc : Class
        sc := s.specific
        prop.each[p]
        sc.removeProperty(p.name)
    end

    operation ProcessClasses(nodes : Set<String>) : Set<Property> is do
        var propSet : Set<Set<Property>> > init Set<Set<Property>> >.new
        nodes.each[s]
        var sc : Class
        sc := root.getClassFromString(s)
        var prop : Set<Property> init Set<Property>.new
        sc.ownedAttribute.each[p| prop.add(p) ]
        propSet.add(prop)
    end

    operation ProcessSuper(c : Class) : Void is do
        var propSet : Set<Set<Property>> > init Set<Set<Property>> >.new
        c.specialisation.each[s]
        var sc : Class
        sc := s.specific
        var prop : Set<Property> init Set<Property>.new
        sc.ownedAttribute.each[p| prop.add(p) ]
        propSet.add(prop)
C.3. KERMETA

41 var iset : Set<Property>
42 iset := Process(propSet)
43 if iset.size() > 0 then
44 // stdio.writeln("Common Properties")
45 addPropertiesSuper(c, iset)
46 remPropertiesChildren(c, iset)
47 else
48 // stdio.writeln("No Common Properties")
49 end
50 end
51
52 operation Intersect(s1 : Set<Property>, s2 : Set<Property>) : Set<Property> is do
53 result := Set<Property>.new
54 var elem : Property
55 from var it : Iterator<Property> init s1.iterator
56 until it.isOff
57 loop
58 elem := it.next
59 if s2.contains(elem) then
60 result.add(elem)
61 end
62 end
63 end
64 operation Process(propSet : Set<Set<Property>>) : Set<Property> is do
65 var iset : Set<Property> init Set<Property>.new
66 var start : Boolean init true
67 propSet.each{pset |
68 if start then
69 iset := pset
70 start := false
71 end
72 iset := Intersect(pset, iset)
73 }
74 result := iset
75 end
76 }

Rule2
1 class Rule2
2 {
3 attribute root : KCL::model
4
5 operation addProperty(c : Class, p : String, newClass : Class) : Void is do
6 root.Eelement.select(e | e.isInstanceOf(Class)).each{rc |
7 var rcl : Class
8 rcl := rc
9 if rcl.name == c.name then
10 var prop : Property
11 prop := rcl.getProperty(p)
12 if prop != void then
13 newClass.ownedAttribute.add(prop)
14 end
15 end
16 }
17
18 operation removeProperty(c : Class, p : String) : Void is do
19 root.Eelement.select(e | e.isInstanceOf(Class)).each{rc |
20 var rcl : Class
21 rcl := rc
22 if rcl.name == c.name then
23 rcl.removeProperty(p)
24 end
25 end
26
27 operation createGeneral(c : Class, nc : Class, sc : Class) : Void is do
28 root.Eelement.select(e | e.isInstanceOf(Class)).each{rc |
29 var rcl : Class
30 }
C.3. KERMETA

```plaintext
C.3. KERMETA

29 rcl := rc
30 if rcl name == c.name then
292 var spec : Generalization init Generalization.new
303 spec.general := nc
304 spec.specific := rcl
305 rcl.generalization.add(spec)
306 root.ERelation.add(spec)
307
308 var oldGen : Generalization
309 root.ERelation.each[r | ]
307 if r.general == sc and r.specific == rcl then
310 oldGen := r
311 end
312
313 if oldGen != void then
314 // stdio.writeln("remove general")
315 rcl.generalization.remove(oldGen)
316 sc.specialisation.remove(oldGen)
317
318 root.ERelation.remove(oldGen)
319 end
320 end
321 end
322 }
323 end
324}
325 operation processPropertySet(propMgr : PropertyMgr, cont : Container, origSuper : Class)
326 : Boolean is do
327 var name : String init ""
328 var pcont : Set<String>
329 var ps : PContainer
330 pcont := propMgr.getLargestSet()
331 pcont := propMgr.getCommonProperties()
332 if pcont.size() > 0 then
333 pcont.each[p | ] name.append(p)
334 // stdio.writeln("ClassName" + name + ":")
335 var newClass : Class init Class.new
336 newClass.name := name
337 pcont.each[p | ]
338 pcont.data.each[c | ]
339 var cl : Class
340 cl := root.getClassFromString(c)
341 removeProperty(cl,p)
342 }
343 var ref : Class
344 ref := root.getClassFromString(cont.ref)
345 addProperty(ref,p,newClass)
346 removeProperty(ref,p)
347 }
348 pcont.data.each[c | ]
349 var cl : Class
350 cl := root.getClassFromString(c)
351 createGeneral(cl,newClass,origSuper)
352 }
353 var ref : Class
354 ref := root.getClassFromString(cont.ref)
355 createGeneral(ref,newClass,origSuper)
356 root.EElement.add(newClass)
357 if origSuper != void then
358 var spec : Generalization init Generalization.new
359 spec.general := origSuper
360 spec.specific := newClass
361 root.ERelation.add(spec)
362 newClass.generalization.add(spec)
363 end
364 result := true
365 else
366 result := false
```

298
C.3. KERMETA

operation processProperties(cont : Container, origSuper : Class) : Void is do
  // stdio.writeln("processing properties")
  var propMgr : PropertyMgr init PropertyMgr.new
  var ref : Class
  ref := root.getClassFromString(cont.ref)
  ref.ownedAttribute.each(p |
    propMgr.newSet(p)
    cont.data.each(c |
      var cl : Class
      cl := root.getClassFromString(c)
      if cl.checkProperty(p) then
        propMgr.addClass(p, cl)
      end)
  )
  processPropertySet(propMgr, cont, origSuper)
end

operation ProcessGeneral(cl : Class) : Void is do
  // stdio.writeln("Processing General: " + cl.name)
  var children : Set<String>
  children := cl.getChildren()
  var clsMgr : ClassMgr init ClassMgr.new
  children.each(c1 |
    clsMgr.newSet(c1)
    children.each(c2 |
      var cl1 : Class
      var cl2 : Class
      cl1 := root.getClassFromString(c1)
      cl2 := root.getClassFromString(c2)
      if cl1.name != cl2.name then
        // stdio.writeln("Comparing " + cl1.name + ": " + cl2.name)
        if cl1.checkCommon(cl2) then
          // stdio.writeln("Common found")
          clsMgr.addClass(c1, c2)
        end
      end)
  )
  // stdio.writeln("Processing Containers")
  var cont : Container
  cont := clsMgr.getLargestSet()
  if cont.data.size() > 0 then
    processProperties(cont, cl)
  end
end

Rule3
1 class Rule3
2 {
  attribute root : KCL::model
  operation addProperty(c : Class, p : String, newClass : Class) : Void is do
    root.EElement.select(e | e.isInstanceOf(Class)).each{rc |
      var rcl : Class
      rcl ?= rc
      if rcl.name == c.name then
        var prop : Property
        prop := rcl.getRefProperty(p)
        if prop != void then
          newClass.ownedAttribute.add(prop)
        end
    end
}
C.3. KERMETA

operation removeProperty(c : Class, p : String) : Void is do
  root.EElement.select(e | e.isInstanceOf(Class)).each{rc |
    var rcl : Class
    rcl := rc
    if rcl.name == c.name then
      rcl.removeProperty(p)
    end
  }
end

operation createGeneral(c : Class, nc : Class) : Void is do
  root.EElement.select(e | e.isInstanceOf(Class)).each{rc |
    var rcl : Class
    rcl := rc
    if rcl.name == c.name then
      var spec : Generalization init Generalization.new
      spec.general := nc
      spec.specific := rcl
      rcl.generalization.add(spec)
      root.ERelation.add(spec)
    end
  }
end

operation processPropertySet(propMgr : PropertyMgr, cont : Container) : Boolean is do
  var name : String init ""
  var pcont : Set<String> init Set<String>.new
  var pcont : PContainer
  pcont := propMgr.getLargestSet()
  pcont := propMgr.getCommonProperties()
  if pcont.size() > 0 then
    pcont.each{p | name.append(p)}
    //stdio.writeln("ClassName" + name + "<")
    var newClass : Class init Class.new
    newClass.name := name
    pcont.each{p |
      pcont.data.each{c|
        var cl : Class
        cl := root.getClassFromString(c)
        removeProperty(cl, p)
      }
    }
    var ref : Class
    ref := root.getClassFromString(cont.ref)
    addProperty(ref, p, newClass)
    removeProperty(ref, p)
  }
  pcont.data.each{c |
    var cl : Class
    cl := root.getClassFromString(c)
    createGeneral(cl, newClass)
  }
  var ref : Class
  ref := root.getClassFromString(cont.ref)
  createGeneral(ref, newClass)
  root.EElement.add(newClass)
  result := true
else
  result := false
end
end

operation processProperties(cont : Container) : Void is do
  //stdio.writeln("processing properties")
  var propMgr : PropertyMgr init PropertyMgr.new
  var ref : Class
  ref := root.getClassFromString(cont.ref)
  ref.ownedAttribute.each{p |
    propMgr.newSet(p)
  }
end


C.3. KERMETA

82          cont.data.each[c]
83          var cl : Class
84          cl := root.getClassFrom(String(c))
85          if cl.checkProperty(p) then
86              propMgr.addClass(p, cl)
87          end
88      }
89
90      processPropertySet(propMgr, cont)
91    }
92
93    operation ProcessClass(nodes : Set<String>) : Void is do
94        //stdio.writeln("Processing Nodes: ")
95        var clsMgr : ClassMgr init ClassMgr.new
96        nodes.each[s1]
97        var cl : Class
98        cl := root.getClassFrom(String(s1))
99        clsMgr.newSet(cl.name)
100      nodes.each[s2]
101      var c1 : Class
102          c1 := root.getClassFrom(String(s1))
103          clsMgr.newSet(c1.name)
104      nodes.each[s2]
105      var c2 : Class
106          c2 := root.getClassFrom(String(s2))
107          if c1.name != c2.name then
108              //stdio.writeln("Comparing " + c1.name + ": " + c2.name)
109                  if c1.checkCommon(c2) then
110                          //stdio.writeln("Common found")
111                          clsMgr.addClass(c1.name, c2.name)
112              end
113          end
114      end
115  }
116
117      //stdio.writeln("Processing Containers")
118      var cont : Container
119      cont := clsMgr.getLargestSet()
120      if cont.data.size() > 0 then
121          processProperties(cont)
122      end
123  }
124

RuleManager

class RuleManager
{| attribute root : KCL::model
1
2      operation Save(fileName : String) : Void is do
3          var rs : ResManager init ResManager.new
4          rs.Save(fileName, root)
5      end
6
7      operation Init(fileName : String) : Void is do
8          var rs : ResManager init ResManager.new
9          root := rs.LoadModel(fileName)
10      end
11
12    operation ProcessRule1(nodes : Set<String>) : Void is do
13          //stdio.writeln("Processing Rule 1")
14          //nodes.each[cs] stdio.writeln("Class Name: " + cs) }
15          var procNodes : Set<String> init Set<String>.new
16          var rl : Rule1 init Rule1.new
17          rl.root := root
18          nodes.each[cs]
19              var c : Class
20              c := rl.root.getClassFrom(String(cs))
21              c.specialisation.each[p]
22      //stdio.writeln("Parent Nodes: " + p.specfic.name)
23          procNodes.each[pc]
if p.specific.name == pc and p.specific.done1 == void then
    fd := false
end

if fd then
    //stdio.writeln("Adding to procNodes: "+p.specific.name)
    procNodes.add(p.specific.name)
end

nodes.each(cs|
    var c : Class
    c := r1.root.getClassFromString(cs)
    c.generalization.each(g|
        var cl : Class
        cl := g.general
        r1.ProcessSuper(cl)
        cl.done1 := 1
    )
    //stdio.writeln("size of general: "+c.generalization.size().toString())
    if c.generalization.size() == 0 then
        // Reached a root node so we need to stop processing
        c.done1 := 1
    end
)

root := r1.root
//stdio.writeln("procNodes.each(e|stdio.writeln(e})")
ProcessRule1(procNodes)
end
//stdio.writeln("End of Processing Parent Nodes")

operation ProcessRule2(nodes : Set<String>) : Void is do
    //stdio.writeln("Processing Rule 2")
    var procNodes : Set<String> init Set<String>.new
    nodes.each(cs|
        var c : Class
        c := root.getClassFromString(cs)
        c.specialisation.each(p|
            var fd : Boolean init true
            procNodes.each(pc|
                if p.specific.name == pc then
                    fd := false
                end
            )
            if fd then
                procNodes.add(p.specific.name)
            end
        )
    )

    var r2 : Rule2 init Rule2.new
    r2.root := root
    nodes.each(cs|
        var c : Class
        c := r2.root.getClassFromString(cs)
        if c.done2 == void then
            c.generalization.each(g|
                var cl : Class
                cl := g.general
            )
            // Prior to processing, remove children from the main queue for this class
            //cl.specialisation.each(s|
C.3. KERMETA

```plaintext
C.3. KERMETA

88 // stdio.writeln("Mark as done "+ s.specific
89 var sc : Class
90 sc := s.specific
91 sc.done2 := 1
92 }
93 r2.ProcessGeneral(cl)
94 }
95 root := r2.root
96 if procNodes.size()>0 then
97 ProcessRule2(procNodes)
98 end
99 //stdio.writeln("End of Processing Parent Nodes")
100 end
101 operation ProcessRuleNodes3(nodes : Set<String>) : Void is do
102 //stdio.writeln("Processing Rule 3")
103 // Check for Rule 1 First and Apply
104 var r1 : Rule1 init Rule1.new
105 r1.root := root
106 if r1.ProcessClasses(nodes).size()>0 then
107 // First add a new class
108 var name : String init ""
109 nodes.each{p| name.append(p)}
110 //stdio.writeln("ClassName"+name+"<")
111 var newClass : Class init Class.new
112 newClass.name := name
113 nodes.each{cs|
114 var c : Class
115 c := root.getClassFrom(String(cs))
116 var spec : Generalization init Generalization.new
117 spec.general := newClass
118 spec.specific := c
119 root.ERelation.add(spec)
120 c.generalization.add(spec)
121 }
122 r1.ProcessSuper(newClass)
123 root.EElement.add(newClass)
124 root := r1.root
125 var r3 : Rule3 init Rule3.new
126 r3.root := root
127 r3.ProcessClass(nodes)
128 root := r3.root
129 //stdio.writeln("End of Processing Rule 3")
130 end
131 operation Process() : Void is do
132 //stdio.writeln("Processing")
133 var leafNodes : Set<String> init Set<String>.new
134 /
135 root.EElement.each{|r|
136 var cl : NamedElement
137 cl := r
138 stdio.writeln("Class name: "+cl.name+" of type "+cl.getMetaClass().name)
139 }
140 */
141 root.EElement.select{e| e.isInstanceOf(Class)}.each{|c|}
142 var cl : Class
143 cl := c
144 //stdio.writeln("Class name: "+cl.name)
```
C.3. KERMETA

```kotlin
// stdio.writeLn("Specialization Size " + cl.specialisation.size().toString())
if cl.specialisation.size() == 0 then
  // stdio.writeLn(cl.name)
  leafNodes.add(cl.name)
end
}

ProcessRule1(leafNodes)
ProcessRule2(leafNodes)
ProcessRule3()

operation ProcessRule3(): Void is do
  // stdio.writeLn("Processing3")
  var topNodes: Set<String> init Set<String>.new
  root.Eelement.select{|e| e.isInstanceOf(Class)}.each{|c|
    var cl : Class
    cl ?= c
    // stdio.writeLn("Specialization Size " + cl.specialisation.size().toString())
    if cl.generalisation.size() == 0 then
      // stdio.writeLn(cl.name)
      topNodes.add(cl.name)
    end
  }
  if topNodes.size() > 1 then
    // More than one root node in the structure
    ProcessRuleNodes3(topNodes)
  end
end

operation main(): Void is do
  // stdio.writeLn("In main function")
  // tt.start()
  Init("platform:/resource/QIKermeta/model/TestCase2model500.xmi")
  Process()
  Save("platform:/resource/QIKermeta/model/TestCase2model50output.xmi")
  /*
   t1 := tt.getElapsedTime()
   var et : Time init Time.new()
   
   stdio.writeLn({et.getCurrentDateTimeAsString("hh:mm:ss:ms")})
   Save("platform:/resource/QIKermeta/model/TestCase1modeloutput.xmi")
   //stdio.writeLn(t1.getElapsedTime().toString())
   tt.start()
   Init("platform:/resource/QIKermeta/model/TestCase3model.xmi")
   Process()
   t2 := tt.getElapsedTime()
   Save("platform:/resource/QIKermeta/model/TestCase3modeloutput.xmi")
   tt.start()
   Init("platform:/resource/QIKermeta/model/TestCase4model.xmi")
   Process()
   t3 := tt.getElapsedTime()
   Save("platform:/resource/QIKermeta/model/TestCase4modeloutput.xmi")
   tt.start()
   Init("platform:/resource/QIKermeta/model/TestCase4model.xmi")
   Process()
   t4 := tt.getElapsedTime()
   Save("platform:/resource/QIKermeta/model/TestCase4modeloutput.xmi")
   tt.start()
   Init("platform:/resource/QIKermeta/model/TestCase5model.xmi")
   Process()
   t5 := tt.getElapsedTime()
   Save("platform:/resource/QIKermeta/model/TestCase5modeloutput.xmi")
```
C.4. ATL

```plaintext
// stdio . writeln ( dt . getElapsedTime () . toString () )
*/
stdio . writeln ( "Time for Test Case 1: " + t1 . toString ( ) )
stdio . writeln ( "Time for Test Case 2: " + t2 . toString ( ) )
stdio . writeln ( "Time for Test Case 3: " + t3 . toString ( ) )
stdio . writeln ( "Time for Test Case 4: " + t4 . toString ( ) )
stdio . writeln ( "Time for Test Case 5: " + t5 . toString ( ) )
*/
}
```

C.4 ATL

```plaintext
module Rationalisation;
create OUT : ClassDiagram from IN : ClassDiagram;
helper def : cSup : ClassDiagram ! Class = OclUndefined;

6-- This rule copies those properties belonging to subclasses which are not repeated in all
7-- the subclasses of a class.
8rule CopyPropertiesSubclasses{
9  from
10       p : ClassDiagram ! Property,
11       c : ClassDiagram ! Class,
12       cSup : ClassDiagram ! Class(c . ownedAttribute -> includes(p) and c . generalization ->
13            exists(g | g . general = cSup) and
14            cSup . specialisation -> exists (g | not g . specific . ownedAttribute -> exists
15              (pr | pr . name = p . name)))
16  to
17       pOut : ClassDiagram ! Property {
18            name <- p . name,
19            type <- p . type
20      }
21  }
22
23-- This rule copies properties (all of them) from classes which are not subclasses.
24rule CopyPropertiesNonSubclasses{
25  from
26       p : ClassDiagram ! Property,
27       c : ClassDiagram ! Class(c . ownedAttribute -> includes(p) and c . generalization ->
28            size ()=0)
29  to
30       pOut : ClassDiagram ! Property {
31            name <- p . name,
32            type <- p . type
33      }
34  }
35
36-- This rule copies all the classes. Besides, it adds to those new classes those attributes
37-- created by rules CopyPropertiesSubclasses and CopyPropertiesNonSubclasses, when it
38-- corresponds.
39rule CopyClasses{
40  from
41       c : ClassDiagram ! Class
42  to
43       cOut : ClassDiagram ! Class {
44            name <- c . name,
45            generalization <- c . generalization ,
46            specialisation <- c . specialisation
47      }
48  }
```

305
45  --- If \( c \) is not a subclass, all the attributes are copied by rule
46  CopyPropertiesNonSubClasses,
47  so we make the new class contain them
48  if (\( c \).generalization -> size() = 0)
49  cOut.ownedAttribute = c.ownedAttribute;
50  } else {
51  --- If \( c \) is a subclass, those attributes not repeated in its siblings are copied
52  --- by rule CopyPropertiesSubClasses, so we add them here
53  thisModule.cSup = c.generalization->first().general; --- We suppose
54  there is not multiple inheritance
55  for (\( p \) in c.ownedAttribute)
56  if (thisModule.cSup.specialisation -> exists (g | not g.specific
57  .ownedAttribute->exists( pr | pr.name = p.name)))
58  cOut.ownedAttribute = cOut.ownedAttribute -> append(p);
59  }
60  } furtherm More, if \( c \) is a superclass, we create a new attribute in it if all its
61  subclasses
62  --- contain an attribute with the same name and type
63  if (\( c \).specialisation -> size() >0)
64  for (\( p \) in c.specialisation->first().specific.ownedAttribute)
65  if (\( c \).specialisation -> forAll(g | g.specific.ownedAttribute ->
66  exists (pr | pr.name = p.name and pr.type = p.type)))
67  --- A call to lazy rule CreateProperty is made
68  cOut.ownedAttribute = cOut.ownedAttribute -> append( 
69  thisModule.CreateProperty(p));
70  }
71  }
72  --- lazy rule that copies the property received as argument
73  lazy rule CreateProperty{
74  from
75  p : ClassDiagram!Property
76  to
77  pOut : ClassDiagram!Property {
78   name <- p.name,
79   type <- p.type
80  }
81  }
82  }
83  --- This rule copies all the generalization classes
84  rule CopyGeneralization{
85  from
86  g : ClassDiagram!Generalization
87  to
88  gOut : ClassDiagram!Generalization {
89   specific <- g.specific,
90   general <- g.general
91  }
92  }
93  }
94  --- This rule copies all the Type classes
95  rule CopyType{
96  from
97  t : ClassDiagram!Type
98  to
99  tOut : ClassDiagram!Type {
100   name <- t.name
101  }
102  }
103  }
C.5. QVT-R

1. This transformation is not complete. The QVT is applied to rule 1 and partially rule 2 and not rule 3.

--- Rule 1 ---

transformation QualityImprovement (source:KCL, target:KCL) {
  top relation TakeGeneralization {
    checkonly domain source sourcegen : KCL::Generalization {
      specific = sub : KCL::Class {
        ownedAttribute = prop : KCL::Property ( ) ;
      } ;
      enforce domain target targetgen : KCL::Generalization ( ) ;
      when (sourcegen.general.specialisation.size() > 1 and sourcegen.general.specialisation->exists(c | c.specific.name <> sub.name and c.specific.ownedAttribute->exists(p | p.name = prop.name and p.type.name <> prop.type.name) ) ) ;
    } ;
  } ;
  where { CheckRule1 (sourcegen, targetgen) ;}
}

--- Rule 2 ---

transformation QualityImprovement (source:KCL, target:KCL) {
  top relation TakeGeneralisationForRule2 {
    checkonly domain source sourcegen : KCL::Generalization {
      specific = sub : KCL::Class {
        ownedAttribute = prop : KCL::Property ( ) ;
      } ;
      enforce domain target targetgen : KCL::Generalization ( ) ;
      when (sourcegen.general.specialisation.size() > 2 and sourcegen.general.specialisation->exists(c | c.specific.name <> sub.name and c.specific.ownedAttribute->exists(p | p.name = prop.name and p.type.name <> prop.type.name) ) ) ;
    } ;
  } ;
  where { CheckRule2 (sourcegen, targetgen) ;}
}
C.5. QVT-R

57 checkonly domain source sourcegen : KCL::Generalization {
58 specific = subsource : KCL::Class {
59 ownedAttribute = prop : KCL::Property {}};
60 enforce domain target targetgen : KCL::Generalization {
61 specific = subtarget : KCL::Class {}};
62 when sourcegen.general.specialisation->exists(c | c.specific.name <> subsource.name and c.
63 specific.ownedAttribute->exists(p | p.name <> prop.name and p.type.name<> prop.type.name )):
64 where { CopyGeneralisationForRule2(sourcegen , targetgen) ;
65 CopyClass( subsource , subtarget );}
66}
67 relation CopyGeneralisationForRule2{
68 checkonly domain source sourcegen : KCL::Generalization {;
69 enforce domain target targetgen : KCL::Generalization {;
70 top relation TakesuperClassForRule2{
71 n : String;
72 checkonly domain source s : KCL::Class {
73 name = n ,
74 specialisation = spsourcesuper : KCL::Generalization {
75 specific = spe : KCL::Class{
76 ownedAttribute = prop : KCL::Property {}} };
77 enforce domain target t : KCL::Class {
78 specialisation = sptargetsuper : KCL::Generalization {,
79 ownedAttribute = prop : KCL::Property {}};
80 when { CopyGeneralisationForRule2(spsourcesuper , sptargetsuper ) ;}
81 where { GenerateNewclass(s , t) ;}
82 }]
83 relation GenerateNewclass{
84 n : String;
85 checkonly domain source s : KCL::Class {
86 name = n ;
87 enforce domain target t : KCL::Class {name = n + 'NEW'};
88 where { GenerateSuperSuperclass(s , t) ;}
89 ]
90 }
91}
92 relation GenerateSuperSuperclass{
93 n : String;
94 checkonly domain source s : KCL::Class {
95 name = n ;
96 };
97 enforce domain target t : KCL::Class {
98 generalisation = gen : KCL::Generalization{
99 general = s2 : KCL::Class{
100 name = n })}.;
101 }