TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

MASTER’S THESIS

Quantitative Analysis of Model Transformations

by
Phu hong Nguyen

Supervisor:
Prof. Dr. Mark van den Brand (TU/e)

Tutor:
Ir. Marcel van Amstel (TU/e)

Eindhoven, 2010
Abstract

Model driven software development (MDSD) is gaining more and more attention as a viable alternative to the traditional code-centric software development paradigm. In MDSD, while models play a central role throughout the entire development process, model-to-model transformations (model transformations) is considered as the heart and soul of MDSD. Many model transformation approaches and languages/engines have been developed in the past years, for examples: ATL (Atlas Transformation Language), QVT (Query/View/Transformation), Xtend, Epsilon, Kermeta and Stratego.

Since model transformations play a significant role in MDSD, the development of methods for assessing the quality of model transformations is necessary. Metrics are widely used for assessing the quality of software, and have already been applied in the context of several transformation languages such as ASF+SDF and ATL. However, the quality of transformations defined in Xtend, the transformation language of the openArchitectureWare (oAW) toolkit, and QVT-Operational (QVTo), the transformation language of the Meta Object Facility Query/View/Transformation (MOF QVT) specification, have not been considered yet. This thesis is about metrics-based quality assessment of model transformations defined in Xtend and QVTo which, to our knowledge, are two of the most widely-used and actively developed transformation languages nowadays.

Our approach aims at assessing the internal quality of model transformations by extracting metrics from the model transformations directly. In this work, we have established two sets of metrics that enable quality assessment of model transformations defined in Xtend and QVTo. One set of metrics is specific for Xtend model transformations and another one is specific for QVTo model transformations. Moreover, quality assessment of
model transformations defined in Xtend and QVTo will not only be enabled by means of metrics but also coverage and dependency analysis. In fact, it is infeasible and error-prone to extract all the metrics, coverage and dependency data by hand. Hence, tools have to be developed for automatically extracting these measures from model transformations. There are several ways of implementing such a tool. Within this project, we have used three different ways to develop three different tools. They are discussed in detail in this thesis. Extracted sets of metrics are displayed in report files. Extracted coverage and dependency data can be visualized for analysis using visualization tools. To evaluate the results, five case studies have been performed.
To my parents

In memory of my mother...
Acknowledgements

First, I would like to thank Nuffic, the Netherlands organization for international cooperation in higher education, for awarding me the prestigious Huygens scholarship to do my Master program in this beautiful country. Without the scholarship, it would be impossible for me to finance my study in the Netherlands. I am also glad to be a member of HUTAC (Huygens Talent Cicle), the official Huygens scholarship alumni organization.

My study of computer science at the Eindhoven University of Technology started in 2008. During the 2009 fall-semester, I followed the course Generic Language Technology and the seminar of the Software Engineering and Technology (SET) group. At that time, I started to get interested in Model Driven (Software) Engineering. Then, very quickly I knew that the ideal topic for my graduation project would be in this field. Since model driven software development is the research focus of SET group, I made an appointment with Prof. Mark van den Brand and we discussed about several MDSD related topics for my graduation project. I owe many thanks to Mark van den Brand for giving me the opportunity to work on a very interesting graduation project and for the weekly meetings, insightful discussions and to-the-point advices throughout the project. I would like to thank my tutor Marcel van Amstel for his daily guidance. He is such a great tutor that every Master student would love to work with. I wish him the best of luck in acquiring his doctorate.

I would like to thank Mark van den Brand and Harold Weffers, the director at LaQuSo, for arranging me a nice working place in the LaQuSo. Thank people in the LaQuSo for the pleasant working environment and the interesting discussions during lunch time. Furthermore, I would like to thank Dr. Alexander Serebrenik for guiding me through the SET seminar to get
me well-prepared for my Master thesis. I am thankful to Dr. Tom Verhoeff and Dr. Danny Holten for their willingness to be part of my examination committee. I also definitely want to thank Danny Holten for the visualization tools I used in this project. Besides, I am very grateful to Sergey Boyko, Sven Efftinge and Ed Willink. Without some guidance from them, it would be much harder to reuse the QVTo compiler and Xtend parser in this project.

Finally, I would like to warmly thank my family and friends for their supports and encouragements, especially from my father and younger sister. Regretfully, for my mother these words of gratitude come too late... Last but not least, a special word of thanks goes to Minh-Ngoc, my fiancee, for her understanding and encouragements during the time I had been doing my Master program.
Contents

List of Figures ix
List of Tables xi

1 Introduction 1
   1.1 Motivation .............................................. 1
   1.2 Project goal ............................................ 2
   1.3 Structure of the thesis ................................. 3

2 State of the art 5
   2.1 MDE & model transformations ............................ 5
      2.1.1 Introduction to MDE ............................... 5
      2.1.2 Model transformations ............................. 7
   2.2 Model transformation formalisms ......................... 7
      2.2.1 ATL ................................................. 8
      2.2.2 QVT ............................................... 9
      2.2.3 oAW Xtend ......................................... 9
   2.3 Related work ........................................... 9

3 Xtend model-to-model transformations 11
   3.1 openArchitectureWare ..................................... 11
   3.2 Xtend model transformations ............................. 13
      3.2.1 Structure of Xtend files .......................... 13
      3.2.2 Expression Extensions ............................ 15
      3.2.3 Create Extensions .................................. 16
      3.2.4 Java Extensions .................................... 16
6 Tools

6.1 General requirements ........................................... 54
  6.1.1 Platform .................................................. 54
  6.1.2 Input ..................................................... 54
  6.1.3 Output .................................................... 54
  6.1.4 Extensibility .............................................. 54

6.2 Design decisions ................................................ 54
  6.2.1 Xtend model transformations extractor tools ................. 55
  6.2.2 QVTo model transformations extractor tool .................. 57

6.3 Java-based Xtend model transformations Extractor Tool .......... 58
  6.3.1 Design ................................................... 58
  6.3.2 Implementation .......................................... 65

6.4 Model-based QVTo model transformations Extractor Tool .......... 66
  6.4.1 Design ................................................... 66
  6.4.2 Implementation .......................................... 70

7 Case studies

7.1 Case study book2pub defined in Xtend .......................... 80
  7.1.1 Description ............................................... 80
  7.1.2 Metrics extracted using JXET .............................. 81

7.2 Case study Sync2Async defined in Xtend ......................... 84
  7.2.1 Description ............................................... 84
  7.2.2 Metrics extracted using JXET and XXET .................... 85
  7.2.3 Dependencies Graph ....................................... 92

7.3 Case study book2pub defined in QVTo .......................... 94
  7.3.1 Description ............................................... 94
  7.3.2 Metrics extracted using MQET ............................ 95

7.4 Case study QVTo2Metrics defined in QVTo ....................... 97
  7.4.1 Description ............................................... 97
  7.4.2 Metrics extracted using MQET ............................ 98
  7.4.3 Coverage Graph .......................................... 100
List of Figures

2.1 Model Driven Software Development ......................... 6
2.2 Model-to-model transformations .............................. 8
3.1 Overview of oAW ......................................... 12
3.2 An example of oAW workflow .............................. 13
4.1 QVT .................................................. 20
6.1 Parsing model transformations .............................. 55
6.2 Storing Xtend abstract syntax tree ......................... 56
6.3 Storing QVTo abstract syntax tree ......................... 57
6.4 Architecture of JXET .................................. 59
6.5 Package diagram of JXET ................................ 60
6.6 Data extractor of JXET .................................. 62
6.7 Java-based data model for Xtend ......................... 63
6.8 Java-based extractor for Xtend ............................ 64
6.9 Data model for Xtend metrics ........................... 65
6.10 Architecture of MQET .................................. 67
6.11 QVTOperational.ecore ................................ 69
6.12 QVTo Module ......................................... 69
6.13 ImperativeOCL.ecore ................................ 71
6.14 Metrics.ecore ........................................ 72
6.15 CallGraph.ecore ...................................... 72
6.16 CoverageRelation.ecore ................................ 73
6.17 Metrics M2M extractor .................................. 74
LIST OF FIGURES

7.1 Upload the zip file of the synchronous to asynchronous communication transformation .................................................. S9
7.2 The simple metrics extracted ......................................................... 90
7.3 The aggregate metrics extracted ...................................................... 91
7.4 Search extension(s) ................................................................. 92
7.5 Dependencies graph of the synchronous to asynchronous communication transformation ...................................................... 93
7.6 The coverage graph is visualized by TreeComaparer ....................... 104
A.1 Class diagram of the expressions types used in an extension ........... 115
B.1 Architecture of XXET ............................................................... 119
B.2 Format of the XML file ........................................................... 121
B.3 Upload extension files ............................................................. 122
B.4 Extension files uploaded .......................................................... 123
B.5 Calculate simple metrics .......................................................... 124
B.6 Calculate aggregate metrics ..................................................... 125
B.7 Search extension statements by name ........................................ 127
C.1 The model transformations QVTo to metrics ................................. 129
C.2 The model transformations QVTo to coverage ............................... 130
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Xtend metrics related to quality attributes (1)</td>
<td>45</td>
</tr>
<tr>
<td>5.2</td>
<td>Xtend metrics related to quality attributes (2)</td>
<td>46</td>
</tr>
<tr>
<td>5.3</td>
<td>Xtend metrics related to quality attributes (3)</td>
<td>47</td>
</tr>
<tr>
<td>5.4</td>
<td>QVTo metrics related to quality attributes (1)</td>
<td>49</td>
</tr>
<tr>
<td>5.5</td>
<td>QVTo metrics related to quality attributes (2)</td>
<td>50</td>
</tr>
<tr>
<td>5.6</td>
<td>QVTo metrics related to quality attributes (3)</td>
<td>51</td>
</tr>
<tr>
<td>7.1</td>
<td>An overview of case studies</td>
<td>79</td>
</tr>
<tr>
<td>7.2</td>
<td>Xtend metrics extracted from the book to publication transformation (1)</td>
<td>82</td>
</tr>
<tr>
<td>7.3</td>
<td>Xtend metrics extracted from the book to publication transformation (2)</td>
<td>83</td>
</tr>
<tr>
<td>7.4</td>
<td>Xtend metrics extracted from the book to publication transformation (3)</td>
<td>84</td>
</tr>
<tr>
<td>7.5</td>
<td>Xtend metrics extracted from the synchronous to asynchronous communi-</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>cation transformation (1)</td>
<td></td>
</tr>
<tr>
<td>7.6</td>
<td>Xtend metrics extracted from the synchronous to asynchronous communi-</td>
<td>87</td>
</tr>
<tr>
<td></td>
<td>cation transformation (2)</td>
<td></td>
</tr>
<tr>
<td>7.7</td>
<td>Xtend metrics extracted from the synchronous to asynchronous communi-</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td>cation transformation (3)</td>
<td></td>
</tr>
<tr>
<td>7.8</td>
<td>QVTo metrics extracted from the book to publication transformation (1)</td>
<td>95</td>
</tr>
<tr>
<td>7.9</td>
<td>QVTo metrics extracted from the book to publication transformation (2)</td>
<td>96</td>
</tr>
<tr>
<td>7.10</td>
<td>QVTo metrics extracted from the book to publication transformation (3)</td>
<td>97</td>
</tr>
<tr>
<td>7.11</td>
<td>QVTo metrics extracted from the QVTo to metrics transformation (1)</td>
<td>98</td>
</tr>
<tr>
<td>7.12</td>
<td>QVTo metrics extracted from the QVTo to metrics transformation (2)</td>
<td>99</td>
</tr>
<tr>
<td>7.13</td>
<td>QVTo metrics extracted from the QVTo to metrics transformation (3)</td>
<td>100</td>
</tr>
<tr>
<td>D.1</td>
<td>QVTo metrics extracted from the ucif to cif transformation (1)</td>
<td>153</td>
</tr>
</tbody>
</table>
LIST OF TABLES

D.2 QVTo metrics extracted from the ucif to cif transformation (2) . . . . . 154
D.3 QVTo metrics extracted from the ucif to cif transformation (3) . . . . . 155
Chapter 1

Introduction

1.1 Motivation

Model driven engineering (MDE) has already been considered as a solution to the handling of complex and evolving software systems [1]. On one hand, the amount and complexity of software that has to be produced and maintained is continuously increasing, but the time pressure of producing reliable software is becoming even tougher. On the other hand, while software systems are becoming more and more important, the required level of reliability of software is increasing as well. These two main challenges for software engineering could be solved by the application of MDE. Since very early, the abstraction level at which software engineers write programs has been raising. MDE is a natural continuation of this trend as its abstraction level is even higher. Software is modeled on a higher level of abstraction and then model transformation techniques and software generation techniques are used to transform models into executable code. The execution of model transformation and software generation can be automated. Therefore, the overall productivity of software engineers will increase. Moreover, the use of models offers the possibility to use these models for analysis techniques such as model checking, performance analysis, correctness proofs, etc. This allows a way to deal with part of the reliability problems [2].

Model transformation is not only one of the key concepts of MDE but also considered as the heart and soul of model driven software development [3]. Since MDE is becoming more and more popular, model transformations also become increasingly important. Because of the prominent role of model transformations in today’s and future software
1. INTRODUCTION

engineering, there should be a methodology for developing model transformations with high quality [4]. A first step in this direction is developing methods for assessing the quality of model transformations. For most other types of software artifacts, e.g., source code and models, there already exist approaches for assessing their quality.

With the evolution of MDE, many formalisms for describing model transformations have become available, for example ATL [5], QVT [6], XTend [7], Kermeta [8], Epsilon [9], etc. Software measurement and metrics are key technologies to manage and control software development projects. Metrics have been studied extensively to assess the quality of (object-oriented) software [10, 11, 12]. Some of the metrics defined in earlier studies can be adapted such that they can be used to measure certain aspects of model transformations. Moreover, new metrics that are specific for model transformations can also be introduced. In fact, metrics have been proposed for measuring the quality of model transformations created with ASF+SDF [4] and ATL [13, 14]. By exploring the methods for assessing the quality of model transformations created with various formalisms, a methodology for developing model transformations with high quality might be discovered. That is, once quality problems in model transformations have been identified, it is possible to propose a methodology for improving their quality. This methodology will probably consist of a set of guidelines which, if adhered to, lead to high-quality model transformations [4]. Therefore, it is also necessary to enable quality measurement of model transformations created using other popular formalisms. In this project, we enable quality assessment of model transformations defined in two of the most popular transformation languages nowadays: Xtend and QVT-Operational (QVTo) [6].

1.2 Project goal

The goal of this project is to enable automated quality assessment of model transformations by means of metrics, coverage and dependency. Our approach is aimed at assessing the internal quality of model transformations by extracting metrics from the model transformations directly. This is referred to as direct quality assessment [15]. First and foremost, the specifications of Xtend and QVTo are studied in order to derive two sets of metrics which are specific for measuring Xtend model transformations and QVTo model transformations. For each set of metrics, it is necessary to develop a tool
to automatically extract metrics from the model transformations. An important aspect
in order to develop the tools is to discover how the Xtend parser and the QVTo parser
work. To enable automated extraction of metrics from the model transformations, it
is also vital to discover the structures of abstract syntax tree (AST) of Xtend model
transformations and AST of QVTo model transformations. Not only by means of met-
rics, quality assessment is also enabled by means of model transformations coverage
and dependency analysis. The disadvantage of using metrics for analysis is that they
are only numbers. They do not provide information in detail, i.e., which part of the
metamodels is used in the transformation (coverage), or exactly which transformation
function is unused (dependency). Therefore, the tools also need to extract coverage
and dependency data that is used to produce coverage graph and dependency graph
for doing analysis. We would like to visualize the coverage graph and dependency graph
in such a way that striking values can easily be observed. Finally, the results will be
evaluated via case studies.

Initially, the scope of this Master project was restricted to enabling quantitative
analysis of Xtend model transformations by means of metrics. However, after having
this task done, we decided to work more for enabling quality assessment of QVTo
model transformations as well. In fact, quality assessment of Xtend (QVTo) model
transformations is not only enabled by means of metrics but also by coverage and
dependency analysis. For these reasons, the sub-goals of this project are:

- Establish a metrics suite enabling measurement of Xtend model transformations.
- Establish a metrics suite enabling measurement of QVTo model transformations.
- Develop tools that enable automatic calculation of these two sets of metrics as
  well as automatic extraction of coverage and dependency data.
- Perform case studies to validate the results.

1.3 Structure of the thesis

The remainder of this thesis is structured as follows.

In Chapter 2, MDE and model transformations are briefly introduced. Moreover,
some formalisms for creating model transformations nowadays will be listed. The last
1. INTRODUCTION

Section in this chapter is about related work in quality assessment of model transformations.

Chapter 3 is about Xtend model transformations within the openArchitectureWare (oAW) toolkit for MDSD. First, the overview of oAW is given. Then, Xtend model transformations are described in detail.

Similar to Chapter 3, Chapter 4 is about QVTo model transformations within the Eclipse M2M implementation. First, an overview of QVT, the transformation metamodel is given. Then, the syntax and semantics of QVTo are explained in detail.

Chapter 5 begins with some aspects of quality measurement such as software metrics and validation of measures. Then, quality attributes for enabling quality assessment of model transformations are mentioned. Next, the most important sections of this chapter are dedicated for describing two sets of metrics, one is specific for Xtend and another one is specific for QVTo. The expected relations of metrics to quality attributes are embedded in the proposal of the metrics.

Without a tool, automatically extracting metrics from model transformations is infeasible. Chapter 6 contains all information about how to develop such a tool. There is more than one way to implement such a tool. Within this project, three tools have been developed: Java-based Xtend model transformations extractor, XML-based Xtend model transformations extractor (for extracting metrics from Xtend model transformations) and model-based QVTo model transformations extractor (for extracting metrics from QVTo model transformations). The requirements, design decisions, architectures and implementation of these tools are addressed in this chapter.

The tools have been used to perform case studies on research and industrial model transformations defined in Xtend and QVTo to verify whether these tools accomplish their tasks. The results of these case studies are described in Chapter 7 as well as the analysis that was discussed on the results by the developers of the model transformations.

The last chapter, namely Chapter 8 contains the conclusions and future work on the subject.
Chapter 2

State of the art

2.1 MDE & model transformations

2.1.1 Introduction to MDE

The history of software development shows that the level of abstractions when developing software systems has always been increasing. Software development started with programming code that was directly executable by machines. After that, the next generation are imperative languages like C, which provide some common libraries for system functionality and can be programmed independent from the underlying processor architecture. The current state-of-the-art programming languages, or so-called third-generation programming languages, are object-oriented ones like Java or C#. Following this trend, MDE has been introduced as a promising approach for software development which gaining more and more attention by industry and academia. The key point of MDE is to go to a higher level of abstractions when developing systems. It focuses more on the software design than on its implementation. The main goal of this paradigm is to create models in an efficient and domain-specific way. This is accomplished by using domain specific languages (DSLs) and generating software from these models. The objective is to increase productivity and reduce time-to-market by enabling development and using concepts closer to the problem domain at hand, rather than those offered by programming languages. The benefits of this development process are efficiency, quality, maintainability and solution focus. Therefore, Model-Driven Software Development (MDSD) or Model-Driven Architecture (MDA), as it is called by the Object Management Group (OMG), is the next level of software develop-
2. STATE OF THE ART

The IBM manifesto claims that MDA-based approaches are founded on three ideas: direct representation, automation and standards (19). MDSD applies the principles of MDE in the development of software systems. Figure 2.1 (adapted from (18)) shows how MDSD works. In MDSD, a software system is specified as a set of models. These models are often written by experts in a domain (maybe non-computer scientists), using DSLs. DSLs (20) are defined through metamodels. In this way, a software system can be expressed properly using DSLs by experts in that domain.

However, in many cases, it is impossible to generate the source code of the system.
directly from the initial set of models. The set of models, defined in a DSL, often needs to be transformed to another set of models, defined in another DSL. This process is called model transformation. There could be more than one model transformation needed until a set of models is obtained with enough detail to implement the system. This is one of the main goals for model transformations. Besides, model transformations can also be used to transform the initial set of models to some sets of models defined in other DSLs suitable for, for example, verification or simulation purposes. Therefore, model transformations are mentioned as the heart and soul of MDSD.

After the final set of models is obtained, the code generation step takes place. That is, model-to-text transformations are used to transform the models to the source code of the system. The model-to-model transformations and model-to-text transformations can be automated to generate the complete source code or partly complete source code which can be merged with manually written code.

2.1.2 Model transformations

Model transformations play an important role in the MDE approach. A model transformation is specified by a model transformation specification, which is defined in a model transformation language. The model transformation specification is used by a model transformation engine to automatically generate target models from source models. In this way, model transformation formalisms provide the essential mechanisms for (repeatedly) manipulating and transforming models that finally can be used for various purposes such as code generation, simulation and verification of software systems.

In the model world, writing model transformations is a common task in software development. Software engineers are supported in performing this task by mature tools and techniques such as Integrated Development Environments (IDEs), compilers, and debuggers. Some model transformation languages and toolkits are introduced in the next section.

2.2 Model transformation formalisms

In this section, an overview of model transformation engines/languages is presented. Due to the large number of transformation languages around, in this section only those
2. STATE OF THE ART

transformation languages that, to our knowledge, are most widely-used and actively developed, are focused on. These languages are: QVT (6), the OMG standard language for specifying model transformations, ATL (5) that is probably the most widely-used transformation language today, Kermeta (8), Epsilon (9), and XTend, which is the transformation language of the popular openArchitectureWare (oAW) framework (7). Figure 2.2 shows how model-to-model transformations work. The model transformations are defined in transformation definitions using one of the transformation languages. These transformation definitions need refer to source metamodel(s) and target metamodel(s) to define the transformation rules that indicate how source model(s) conforms to source metamodel(s) should be transformed to target model(s) conforms to target metamodel(s). Transformation engine executes transformation definition to make the model transformation happened.

Figure 2.2: Model-to-model transformations - This figure shows how model-to-model transformations work

2.2.1 ATL

ATL (5) is the most successful hybrid model transformation language to date. It contains a mixture of declarative and imperative constructs. ATL transformations are unidirectional, operating on read-only source models and producing a write-only target model. During the execution of a transformation the source model may be navigated but changes are not allowed. Target model cannot be navigated. ATL is supported
by a set of development tools such as an editor, a compiler, a virtual machine, and a debugger.

2.2.2 QVT

QVT (6, 21), the current OMG standard for model transformation, adopts a hybrid style by providing both declarative and imperative constructs. QVT consists of three different domain-specific languages: Relations, Core and Operational Mappings (QVTo). The former two are declarative (the implementation is left to an interpreter), while the latter is imperative (a direct implementation of transformation instructions). There have been some implementations for QVT, for example, Eclipse M2M (22) and SmartQVT (23) for the QVTo variant and MediniQVT (24) for the QVT-R variant. Currently, QVTo is the best supported variant in terms of tools.

2.2.3 oAW Xtend

Xtend (7) is a functional transformation language used throughout oAW. It can be used in constraint checks, model transformations and generators. With respect to integration, Xtend is integrated with additional model management languages of the openArchitectureWare framework that target other tasks such as code generation (Xpand) and model validation (Check). Xtend is a purely imperative language. Since oAW is one of the best toolkit for MDSD, Xtend also has good support in terms of tool.

2.3 Related work

Model transformation metrics have been addressed in (4) which focused on model transformations created using the ASF+SDF (25) term rewriting system. There, a quality model which defines the meaning of quality attributes in the context of model transformation is proposed. A set of metrics has been proposed for ASF+SDF transformations as predictors for these quality attributes. Even though some of these metrics are specific for ASF+SDF only, the approach may be generalized and applied to other model transformations formalisms as well.

As ATL is probably the most widely-used transformation language today, there have been some studies aimed at defining metrics for ATL model transformations (13, 14). The metrics defined in these papers overlap because they are defined for the same
2. STATE OF THE ART

transformation language. Some of the metrics considered in (13) are used to measure the input and output metamodels. Therefore both direct and indirect quality assessment is performed. Only direct quality assessment is considered in (14). In this work, we also consider direct quality assessment only. Many metrics which are proposed for Xtend and QVTo model transformations (see Sections 5.3 and 5.4) are conceptually equivalent to the metrics for ATL. However, some other metrics are entirely specific for Xtend or QVTo.

In (26, 27), metrics have been introduced for OCL expressions. The proposed metrics are based on the syntactic structure of the expressions (number of referred attributes, number of navigations, etc) and on the constructs used in their definition (as the number of forAll and select iterators). Indeed, the metrics in (26) are helpful to determine, for instance, the understandability of expressions. Moreover, the complexity of OCL expressions is addressed in (27). Since many transformation languages (i.e., ATL or QVTo) heavily rely on OCL expressions, some of the metrics for OCL expressions can be adapted such that they can be used to measure certain aspects of model transformations.
Chapter 3

Xtend model-to-model transformations

3.1 openArchitectureWare

The openArchitectureWare (oAW) platform [28] which has been migrated to Eclipse, so-called oAW 5, is currently one of the best toolkits for MDSD. We do not use any old oAW plugin anymore, but instead use the new Xpand/Xtend/Check Eclipse projects that are fully integrated into the Eclipse Modeling Framework. However, since the term oAW is still very common, we prefer to use it.

oAW is a suite of tools and components assisting with model driven software development built upon a modular MDA/MDD generator framework implemented in Java supporting arbitrary import (model) formats, meta models, and output (code) formats. The oAW system is based on a workflow engine which executes different processing steps, like model instantiation, validation, model-to-model and model-to-text transformations. The workflow engine precisely controls the generator’s workflow, as specified in an XML workflow definition. Figure 3.1 shows an overview of oAW.

With a suitable instantiator, oAW can read any model. Currently, out-of-the-box support is provided for EMF, various UML tools (MagicDraw, Poseidon, Enterprise Architect, Rose, XDE, Innovator, etc), the Eclipse UML2 project, textual models (using JavaCC or antlr parsers as frontends), XML, Visio as well as pure::variants variant configuration models [7].
Model-to-model transformations can be implemented using Xtend, a textual, functional transformation language. Xtend is an OCL like expression language used throughout oAW. It can be used in constraint checks, model transformations and generators.

oAW can generate any kind of output, using a powerful template language called Xpand that supports template polymorphism, template aspects and many other advanced features necessary for building non-trivial code generators such as optional typing of model elements and sorting and filtering of element sets. Templates are written in this language and used by the generator that is linked to the development workflow of the platform.

As we mentioned before, oAW components have been migrated to Eclipse Modeling. Xpand, Xtend and Check merge into the next generation - Xpand generator framework (under Model-to-Text Transformation (M2T)). Xtext is now under the Textual Modeling Framework (TMF). And oAW workflow engine has become Modeling Workflow Engine (MWE).

The Xpand generator framework provides textual languages, that are useful in different contexts in the MDSD process (e.g. checks, extensions, code generation, model transformation). Each language (Check, Xtend, and Xpand) is built up on a common
3.2 Xtend model transformations

This section which is largely based on the documentation of Xpand / Xtend / Check (29), focuses on describing the Xtend transformation language. Xtend is a functional language suited for a number of uses, mainly for model transformation (i.e., creating a new model based on one or more existing models), others are model modification (i.e., adding to, removing from or just modifying an existing model) or metamodel extension (7).

This transformation language provides the possibility to define rich libraries of independent operations and non-invasive metamodel extensions based on either Java methods or extension expressions which are OCL-like expressions but with Java-like syntax. Those libraries can be referenced from all other textual languages (Check and Xpand), that are based on the expressions framework (29).

![Figure 3.2: An example of oAW workflow](image)

Figure 3.2: An example of oAW workflow - This figure shows an example of oAW workflow

Figure 3.2 shows an example of oAW workflow. The components can be configured arbitrarily so that it is possible to parse multiple models and combine them into an
3. XTEND MODEL-TO-MODEL TRANSFORMATIONS

internal abstract syntax graph. Also, multiple transformers for model-to-model transformations with a validation step after each transformation could be configured, as well as multiple generators for different target artifacts. It can be seen that Xtend plays an important role in the oAW workflow.

3.2.1 Structure of Xtend files

Xtend model transformations are defined in Xtend files. An Xtend file’s extension must be *.ext. Let us take a look at an extend file (29).

```xtend
import my::metamodel;

extension other::ExtensionFile;
/**
 * Documentation
 */
anExpressionExtension(String stringParam) :
    doingStuff(with(stringParam))
;
/**
 * java extensions are just mappings
 */
String aJavaExtension(String param) : JAVA
    my.JavaClass.staticMethod(java.lang.String)
;
```

The example shows the following statements:

- Import statements: used to import name spaces of different types
- Extension import statements: used to import another extend file.
- Expression or Java extensions: model transformations are mainly defined by these extensions.

Moreover, it is also possible to export extensions from another extension file together with local extensions by adding the keyword `reexport` to the end of the respective extension import statement.

```xtend
extension fully::qualified :: ExtensionFileName reexport;
```
3.2 Xtend model transformations

Actually, there are three types of extension: *create extension*, *expression extension*, and *java extension*. The remaining subsections describe these types of extension one by one.

### 3.2.2 Expression Extensions

The syntax of a simple *expression extension* is as follows:

```java
ReturnType extensionName(ParamType1 paramName1, ParamType2...): expression−using−params;
```

Example:

```java
String getterName(NamedElement ele) : 'get'+ele.name.firstUpper();
```

There are two different ways of how to invoke an extension. It can be invoked like a function:

```java
getterName(myNamedElement)
```

The other way to invoke an extension is through the "member syntax":

```java
myNamedElement.getterName()
```

For any invocation in member syntax, the target expression (the member) is mapped to the first parameter. Therefore, both syntactical forms do the same thing.

#### 3.2.2.1 Cached Extensions

If an extension is called without side effects very often, it makes sense to cache the result for each set of parameters, in order improve the performance. That can be achieved by adding the keyword `cached` to the extension:

```java
cached String getterName(NamedElement ele) :
    'get'+ele.name.firstUpper()
;
```

The `getterName` will be computed only once for each `NamedElement`.

#### 3.2.2.2 Private Extensions

By default all extensions are public. That means they are visible from outside the extension file. In order to hide extensions, the keyword `private` has to be added in front of them:
3. XTEND MODEL-TO-MODEL TRANSFORMATIONS

```java
private internalHelper(NamedElement ele) :
    // implementation....
;
```

The `private` keyword can be used with `create extensions` and `java extensions` as well.

### 3.2.3 Create Extensions

`Create extensions`, as a side effect when called, create an instance of the type given after the `create` keyword. To avoid the duplication when creating instances, Xtend provides explicit support. That is, `create extensions` are only executed once per tuple of parameters. So, `create extensions` always are cached extensions.

The syntax of a simple `create extension` is as follows:

```java
create ReturnType extensionName(ParamType1 paramName1, ParamType2...):
    expression−using−params
;
```

### 3.2.4 Java Extensions

It is possible to call a Java method from inside an expression. This can be done by providing a Java extension, for example:

```java
Void myJavaExtension(String param) :
    JAVA my.Type.staticMethod(java.lang.String)
;
```

The signature is the same as for any other extension. The implementation is redirected to a public static method in a Java class. Its syntax is:

```java
JAVA fully.qualified.Type.staticMethod(my.ParamType1, my.ParamType2, ...);
```

### 3.2.5 Arrounds

One of the special supports that oAW (Xtend) provides is aspect-oriented programming (AOP) in Xtend, called `around advices`. The goal of using `around advices` is to change behavior in a non-invasive way (change some things without modifying any code in the packaged extensions). For example, using the workflow engine, it is possible to package
a written generator and deliver it as a kind of black box. If we want to use such a
generator but need to change some things without modifying any code, we can make
use of around advices (29).

The following advice is weaved around every invocation of an extension whose name
starts with 'my::generator::':

```plaintext
around my::generator::*(*) :
    log(' Invoking ' + ctx.name) => ctx.proceed()
;
```

Aspect orientation is basically about weaving code into different points inside the
call graph of a software module. Such points are called join points. In Xtend the join
points are the extension invocations. Which join points the contributed code should be
executed on is specified by something like a 'query' on all available join points. Such a
query is called a point cut.

```plaintext
around [pointcut] :
    expression;
```

A point cut consists of a fully qualified name and a list of parameter declarations.
The extension name part of a point cut must match the fully qualified name of the
definition of the join point. Such expressions are case sensitive. The asterisk character
is used to specify wildcards. Some examples:

```plaintext
my::Extension::definition // extensions with the specified name
org::oaw::* // extensions prefixed with 'org::oaw::'
*Operation* // extensions containing the word 'Operation' in it.
* // all extensions
```
Chapter 4

QVTo model-to-model transformations

4.1 Overview of QVT

Query, View, Transformation (QVT) might be more popular than Xtend as a transformation language used for transforming models. Xtend which is a lightweight model transformation language is often used together with Check and Xpand in model-to-text transformations. QVT (similar to ATL) is entirely dedicated to performing model-to-model transformations. It is defined by the Object Management Group (OMG). The name of the language suggests a three-part structure. The first part is named Query because queries can be applied to a source model, an instance of the source meta-model. Next, the View is a description of what the target model should look like. Finally the Transformation is the part where the results of the queries are projected on the view, and thereby creating the target model.

The specification of QVT depends on the MOF and OCL specifications. The Object Constraint Language (OCL) is a formal language originally used to describe constraints in UML models. Nowadays it can be used with any MOF-based meta-model. OCL can be used for purposes such as specifying pre- and conditions of methods (see Section 4.2.2.2) and as query language (see Section 4.2.5). QVT consists of three different domain-specific languages: Relations, Core and Operational Mappings. The former two are declarative (the implementation is left to an interpreter), while the latter is imperative (a direct implementation of transformation instructions). Each
4. QVTO MODEL-TO-MODEL TRANSFORMATIONS

of them can be combined with black-box operations (see Section 4.5). An overview of the different domain-specific languages is depicted in Figure 4.1. Each language is ultimately transformed to the Core language.

![Diagram of QVT metamodel relationships](image)

Figure 4.1: QVT - Relationships between QVT metamodels

Operational Mappings, often called QVTo, is an imperative language that extends Relations and Core. The syntax looks similar to other imperative languages, such as Java. Unlike the declarative variants, Operational mappings is unidirectional, operating on read-only source models and producing a write-only target model. During the execution of a transformation the source model may be navigated but changes are not allowed. Target model cannot be navigated. Operational mappings are either transformed to the Relations language and then to the Core language, or directly to the Core language.

Currently, Operational Mappings is the best supported variant in terms of tools. Although there exist implementations for the Relations variant (MediniQVT), QVTo is better documented and has a larger user community. At least, it can be seen that there are many practical questions related to QVTo and very few questions related to QVT-R in the Eclipse Community Forums.

There are two implementations for QVTo: Eclipse M2M and SmartQVT. The first is used in this report. The latter could be used for future work, because seemingly it also is a very suitable implementation. Moreover, the QVTo metrics proposed in this report can be used for SmartQVT as well because the concepts are the same even though it would require a different implementation.
In this chapter, QVT-Operational is explained in detail. To present it in a less abstract way, several examples are provided. The structure of QVTo consists of three main packages:

- QVTOperational package: contains general structuring elements and top-level constructions.
- ImperativeOCL package: extension to OCL expressions and type system.
- Standard Library

### 4.2 QVTOperational package

QVTOperational package defines top-level unit elements which are used for general structuring, declarations. It also defines expressions for object creation and modification that will be explained later.

#### 4.2.1 Framework

A QVT Transformation has a certain structure, called the framework. It consists of model-type definitions, a transformation declaration and a main function.

##### 4.2.1.1 Modeltype definitions

A modeltype definition is a reference to a meta-model. It is possible to include entire meta-models in the transformation (in-line definition), or reference to externally defined ones. Obligatory parts of a modeltype definition are a name and a reference. This reference can be location specific, i.e., pointing at a file within the Eclipse workspace. In Eclipse this is done by preceding the location with "platform:/resource/".

```plaintext
//Modeltype definition using the location specific reference
modeltype MM1 uses "platform:/resource/MM1ToMM2/transforms/MM1.ecore";
```

Additionally the reference can be a package namespace URI, Uniform Resource Identifier. In order to achieve this, the meta-models are registered in Eclipse, after which they can be referenced in the following way.

```plaintext
//Modeltype definition using the package namespace URI reference
modeltype MM1 uses "http://mm1/1.0";
```
4. QVTO MODEL-TO-MODEL TRANSFORMATIONS

4.2.1.2 Transformation declaration

A transformation declaration consists of a name, an input and an output metamodel. The latter two refer to the defined modeltypes. It is possible to define a transformation that uses multiple input and output meta-models.

```plaintext
transformation MM1ToMM2(in Source:MM1, out Target: MM2);
```

4.2.1.3 Main function

The actual transformation starts with the main() function. The purpose of this function is to set environment variables and call the first mapping. The names of the meta-models can be used in the main function.

```plaintext
//Source represents the instance of the source meta-model. Source is a set of all //elements. By using rootObjects, only the objects at the highest level are selected . //Additionally, by applying the filter "[Model]" only the Model objects are selected .
main()
{
    Source.rootObjects()[Model]->map toModel();
}
```

4.2.2 Mapping

Mappings form the core of QVT transformations. They ensure that an object from an instance of the source meta-model is transformed into an object in the created instance of the target meta-model. A mapping is invoked by using the keyword `map` or `xmap`. When using `map` and the mapping fails, it returns the null-value. When using `xmap`, an exception is raised instead of returning a null-value.

4.2.2.1 Declaration

A mapping declaration consists of the classname of the object being transformed, a name of the mapping and the classname of the object that will be the result of the mapping.

```plaintext
//Mapping declaration of "toModel()". Because both the input and output class is named //"Model", it has to be prefixed with the modelfype the objects belong to.
mapping MM1::Model::toModel() : MM2::Model { ... }
```
4.2 QVTOperational package

4.2.2.2 Conditions

A mapping declaration can be extended with conditions defined using OCL. To execute a mapping, the source object must conform to these conditions. There are two types of conditions, pre- and post-condition, indicated with the keywords when and where respectively. There is currently no real use for post-conditions, however it is already implemented for future use.

```ocl
//Pre-condition: the Name of the Model object from the MM1 instance should start with an "M"

mapping MM1::Model::toModel() : MM2::Model
//The keyword "self" refers to the input object.

when { self.Name.startsWith("M"); }
{ ... }
```

4.2.2.3 Body

The body of a mapping consists of three sections: init, population and end. The purpose of the init-section is to initialize variables and parameters, and print messages to the terminal. The actual mapping is specified in the population-section. Finally, the end-section is intended to contain additional code that should be executed before leaving the mapping.

```ocl
init { log("Arrived in mapping toModel()."); }
//Population section. This keyword can be omitted.
//Copy the name from the input object to the output object
Name := self.Name;
//A model contains at least one Automaton, each one must be mapped after which it is assigned to the target model.
body += self.body ->map toBody();
end { log("Leaving the mapping toModel()."); }
```

4.2.2.4 Parameters

It is possible to parameterize a mapping. Parameters should have unique names (within the mapping), and have a type: a basic type like Integer, or an object (possibly declared in the metamodel). Parameters are separated by commas.

```ocl
mapping MM1::Model::toModel( Prefix: String, Loc: MM1::Location ) : MM2::Model
```
4. QVTO MODEL-TO-MODEL TRANSFORMATIONS

4.2.3 Resolving

When an object of the source model is transformed to an object in the target model, and later the same source object is referred to by another object, it should not be transformed again. Instead, it should be resolved to the object just created in the target model. The function `resolve` returns a set of target objects. These objects are the result of an earlier mapping from the source object on which the resolving is being applied.

4.2.4 Helpers

A helper is an operation which uses expressions to obtain data from the input object(s), after which it returns an output object. A read-only helper which is not allowed to create or update any objects is called a query, but this separation is not implemented yet in QVTo Eclipse M2M. There is a clear difference between helpers and mappings. Mostly both the input and output objects of a helper belong to the same model, whereas the input and output objects of a mapping often belong to different models, i.e., the source and target models. A helper can be used like a mapping, however this is not the intention of this operation. Like a mapping, a helper can be parameterized. Each parameter can be prefixed with "in" (default), "out" or "inout". When the prefix contains "in", the parameter can be used as input, when it contains "out" as output.

4.2.5 Inheritance

4.2.5.1 Disjuncts

The keyword `disjuncts` is used to indicate that the type of the source object of a "disjunct mapping" is compared to the types of source objects of the indicated mappings after the keyword disjuncts. The first matching mapping performs a transformation on the object. Besides the types of the source objects, the disjunct mapping considers the pre- and post conditions of the indicated mappings. In other words, a disjunct mapping acts like a dispatcher: it checks the type of the input object and passes it on to the first indicated mapping able to transform the object.
4.2.5.2 Inherits and Merges

Mappings can be specialized in other mappings. There exist two options to accomplish this: merges and inherits. An inherited mapping is executed between the init and population section of a mapping, a merged mapping after the end-section of a mapping. The latter could be used to override results of the mapping itself.

4.2.6 Intermediate Classes and Properties

It is possible to define intermediate classes and properties in a transformation. They can be used to make a part of the transformation easier because many classes and properties often need to be manipulated for temporary use but should not be included in the target model. An intermediate class is either an entirely new class or a class which extends an existing class. The class type can be used anywhere in the transformation. An intermediate property is an attribute which is added to an existing class. The type of an intermediate property is often an intermediate class. Intermediate classes and properties do not appear in the target model, they are temporarily. The syntax used to implement these intermediate classes and properties is the same syntax used to declare in-line meta-models (see Section 4.2.1.1).

4.3 ImperativeOCL package

This package contains basic operators, constructions and functions:

- Assignments
- Variables
- Loops (while, foreach)
- Loop interrupt constructs (break, continue)
- Conditional execution workflow
- Convenient shorthand notation
- Mutable collections
4. QVTO MODEL-TO-MODEL TRANSFORMATIONS

The operators used in QVT are similar to the ones used in other programming languages.

Basic constructions to describe the program flow include if/else statements, switch/case statements, and while-loops. It is also possible to use for-loops.

```plaintext
if (condition) then {
  /* Statement 1 */
} else {
  /* Statement 2 */
} endif;
```

It is possible to use conditional expressions in QVT. A conditional expression is a very short way to assign values to an object, depending on a single condition.

```plaintext
var a: String;
var b: Integer;
a := ( if (b=b) then "Logic" else "Miracle" endif; )
```

The switch-statement can replace large if/else statements. Each case represents a unique condition. When a case is true, the next statement is executed. When none of the cases are true, the default statement after "else" is executed.

```plaintext
switch {
  case (condition1) /* Statement 1 */;
  case (condition2) /* Statement 2 */;
  else /* Statement 3 */;
}
```

Finally, the while-statement or while-loop executes a statement while a condition is true. The condition often includes a count variable whose value is altered in the statement. A while-loop can for instance be used to iterate over a set. It is possible to exit the loop before the condition does not hold anymore. This can be accomplished by using the keyword "break". The other keyword often used in loops, "continue", can be used to skip part of the statement and immediately jump to the start of the statement.

```plaintext
while (condition) {
  /* Statement */
}
```

Two important statements used when debugging the transformation are the "log" function and the "assert" expression. The log function displays a message in the terminal, and can be extended with a condition, using the keyword "when". Besides
simple text it is possible to display current values of variables. These values can be transformed to the String type either by using the function repr(), or toString() when it is available. There is a subtle difference between the two: when repr() is for example applied to a floating-point type variable, the represented precision can vary with the toString().

```log
log("message" + variable.repr() + "message" + variable.toString());
```

The `assert` function also displays messages in the terminal, but unlike the informative nature of log, it displays warnings and errors. There are three different assertion levels: "warning", "error" or "fatal". When the level is "fatal", the transformation is terminated at that point. The former two just display the message after which the transformation continues.

```assert
assert level (condition) with log("message");
```

### 4.4 Standard Library

In QVTo, there is also a standard library which provide many built-in functions:

- Element operations: subobjects, deepclone
- Model operations: rootObjects, copy
- String routines: startsWith, indexOf
- Mutable collection routines: List::add, Dict::get
- Transformation execution routines: transform

### 4.5 Black-box implementation

A Blackbox allows complex algorithms to be coded in any supported programming language, such as Java, [6]. This is useful because some algorithms are hard to implement in OCL, or cannot be expressed at all. Because OCL is not a truly programming language, it is not possible to use program logic or use non-query operations. For example, it is not possible to retrieve the current date using OCL.
4. QVTO MODEL-TO-MODEL TRANSFORMATIONS
Chapter 5

Quality attributes and metrics

This chapter focuses on describing quality attributes and metrics for enabling quality assessment of model transformations defined in Xtend and QVTo. We start with discussing some aspects of quality and measurement. Next, quality attributes intended to assess with metrics will be mentioned. Then, a set of metrics specific for measuring Xtend model transformations is presented. Similarly, another set of metrics specific for measuring QVTo model transformations is presented. Finally, every metric in each set of metrics is related to the quality attributes based on our expectation.

5.1 Quality and measurement

This section shows how metrics can be used for software measurement and then validation of measurement is discussed.

5.1.1 Software measurement and metrics

Software measurement and metrics have often been used to manage and to control software development projects. Because of the important role of model transformations in MDSD, it is also necessary to enable measurement of model transformations. We also do this by means of metrics.

Measurement is essential for any engineering activity and for increasing scientific and technical knowledge regarding both the practice of software development and empirical research in software technology [36]. Measurement results can reflect the distinguishing qualities of the measured objects. However, software is difficult to measure
5. QUALITY ATTRIBUTES AND METRICS

because of its immateriality. Nevertheless, it is difficult, however not impossible, to measure software. In fact, there are many metrics proposed by the community to measure software via reflecting the characteristics of the code, the development process and the project team. Some of them are easy to evaluate because they are directly measurable by counting lines of code or number of attributes. These metrics are mostly provided by the source code itself and analyzing its static structure. Others are more complex and cannot be collected that easy. They only can be gathered indirectly via a combination of direct metrics or other indirect metrics (37).

With the development of MDE, model transformation languages have matured to a point where people have started experimenting with model transformation definitions themselves in addition to the language they are written in. In addition to the transformation language properties, the properties of model transformation definitions themselves become important, such as scalability, maintainability and reusability (38). Because using model transformation languages to define model transformations is very similar to writing code for software, metrics can certainly be used to measure the properties of model transformation definitions as well. Thus, there should be metrics specifically defined for measuring model transformations and they should reflect the quality attributes which are also specifically proposed for model transformations.

In conclusion, to assess the software/model transformation quality, an appropriate set of metrics needs to be identified that expresses several quality attributes. IEEE introduced a standard for a software quality metrics methodology (39). Another popular approach for defining metrics is the Goal-Question-Metric methodology of Basili et al. (40). The metrics defined in this chapter are derived based on our expectation of the relationship between the metrics themselves and the quality attributes of model transformations described in Section 5.2. These two sets of metrics are specific for Xtend model transformations and QVTo model transformations, see Sections 5.3 and 5.4.

5.1.2 Validation of measurement

In (37), the authors discuss the role of validation in software measurement. Validation is critical to the success of software measurement (41) and also to quality assessment of model transformations. Kitchenham et al. proposed a framework that can help researchers and practitioners to understand, how to validate a measure, how to assess
5.2 Quality attributes

Model transformations are in many ways similar to other software engineering artifacts. In the model world, writing model transformation specifications is a common task in software development. Software engineers are supported in performing this task by mature tools and techniques in the same way as they are supported now by Integrated Development Environments (IDEs), compilers, and debuggers in their everyday work (5). Model transformations may be used by several software engineers, have to be changed according to changing requirements and should preferably be reused. Therefore, it is necessary to have a quality model, similar to the general software quality model described by Boehm et al (43), but specific for model transformation. This quality model is a hierarchical decomposition of a number of quality attributes (44). These quality attributes, many of which apply to software in general as well, have been identified as relevant for model transformations. In (44), they are presented: Understandability, Modifiability, Reusability, Modularity, Completeness, Consistency and Conciseness. The names of attributes should be self explanatory; however a more detailed description of what they mean in the context of model transformations can be found in (44). In addition to those quality attributes, Performance which was identified as relevant in the quality model presented in (45) is also considered.

5.3 Xtend metrics

This section is about the metrics which are proposed for measuring Xtend model transformations. These metrics are specific for Xtend. Nevertheless, some of them are
5. QUALITY ATTRIBUTES AND METRICS

conceptually similar to metrics defined for ATL [13, 14] or metrics defined for QVTo in Section 5.4. Certainly, many of them are entirely specific for Xtend since Xtend is not heavily based on OCL expressions like ATL or QVTo. The Xtend metrics can be divided into the following categories:

- **Extension metrics:** Xtend model transformations are mainly defined by means of extensions. This category is dedicated for all the metrics related to extensions.

- **Dependency metrics:** There should be a lot of metrics to show the dependency between Xtend files, extensions. They are grouped into this category.

- **Miscellaneous metrics:** Some other metrics which do not fit the discussed categories above are presented in this category.

In the main part of this section below, each of these categories will be addressed by giving the description and motivation for every metric belonging to them. Furthermore, the relations between metrics and quality attributes will be discussed based on our expectations (also based on ASF+SDF and ATL results). An overview of all the metrics can be found in Tables 7.2 and 7.3 in Section 7.1.

### 5.3.1 Extension metrics

Xtend model transformations are defined using extensions. A measure for the size of an Xtend model transformation can be the number of extensions it consists of. Because extensions may be defined in several Xtend files (.ext), the metric number of extensions shows the number of extensions which are defined within all Xtend files of the Xtend model transformation. In Xtend, there are three kinds of extensions: expression extensions, create extensions, and Java extensions. To be more specific, the metrics number of expression extensions, number of create extensions and number of Java extensions are used to measure the amount of a specific type of extensions defined in the model transformation. Moreover, extensions can be defined as private. The metric number of private extensions is used to detect such private extensions.

One of the unique points which only Xtend (not ATL or QVTo) has is the support for caching the result of an extension call. If an extension is called without side effects very often, it is possible to cache the result for each set of parameters, in order improve
the performance. We propose to measure the number of cached extensions and the number of private cached extensions.

A model transformation defined in Xtend can consist of more than one Xtend files. The metric number of extensions per Xtend file can show the distribution of extensions over Xtend files. This metric shows how many number of extensions are defined in each Xtend file. Thus, it can be used to measure the modularity of an Xtend model transformation. Furthermore, we again distinguish for each type of extensions: number of expression extensions per Xtend file, number of create extensions per Xtend file and number of java extensions per Xtend file. Besides, the distribution of cached extensions and private extensions over Xtend files is also considered. So, we have the metrics number of cached extensions per Xtend file, number of private extensions per Xtend file and number of private cached extensions per Xtend file.

Extensions need to be called explicitly. It may be the case there are some extensions present in a model transformation that are never invoked, i.e., extensions which could be obsolete. To detect such unused extensions, the metric number of unused extensions is included. Again, we also have the metrics number of unused expression extensions, number of unused create extensions, number of unused java extensions, number of unused cached extensions and number of unused private extensions.

On a lower level, we define the metrics number of unused expression extensions per Xtend file, number of unused create extensions per Xtend file, number of unused java extensions per Xtend file, number of unused cached extensions per Xtend file and number of unused private extensions per Xtend file.

The disadvantage of using metrics is that they are only numbers. Therefore, they do not provide information in detail, i.e., exactly which extension is unused. This problem is solved by extracting not only metrics but also dependency data. The dependency data contains information about the caller-callee relations between extensions in all Xtend files. The caller-callee graph can be visualized using the tool ExtraVis [16].

In Xtend, it is possible to define extensions with the same name but with different parameters. To measure this kind of extension overloading, we define the metrics number of overloaded extensions, number of overloaded extensions per Xtend file and number of extensions per extension name. Regarding parameters, the metrics number of parameters and number of parameters per extension are proposed. The former shows the total number of parameters of all extensions and the latter shows number
of parameters on a lower level, per extension. For various reasons, it may be the case that some of these parameters are never used. To detect this, the metrics number of unused parameters and number of unused parameters per extension are used.

Local variables can be declared and used within extensions. These variables are often used to provide separation of concerns, i.e., to split the calculation of certain output in logically parts. This should increase the understandability of the extension or the Xtend file. We propose the metrics number of local variables, number of local variables per extension and number of extensions have at least one variable. We also measure the number of unused local variables and number of unused local variables per extension to detect obsolete variable declarations. Moreover, it may be the case that some variables have the same name but are declared with different types. This could influence the understandability of the whole transformation. To detect such variables, the metric number of local variables which have same name but not same type is proposed.

There could be some extensions which have only null in their body. They may be under construction or may be obsolete. We define the metrics number of null extensions and number of null extensions per Xtend file.

In Xtend, statements in the body of an extension are often connected like they are in a chain. Each expression in a chain expression is evaluated in sequence, but only the result of the last expression is returned. To get more insight in the size of a model transformation or the size of an extension, we can measure the metrics length of chain expression and length of chain expression per extension. Furthermore, the size/complexity of an extension can also be reflected by the number of operation calls it has. Operation calls are extension calls as well as method calls of objects (i.e., getter, setter methods). Hence, we define metrics number of operation calls per extension.

Like OCL, the oAW expression sub-language defines several special operations on collections which can be used in Xtend. For example, sometimes, an expression yields a large collection, but we are only interested in a special subset of the collection. Xtend has the select and reject operations to specify a selection out of a specific collection. Moreover, it is also possible the collect operation to specify a collection which is derived from another collection, but which contains objects that are not in the original collection. It is to be expected that model transformations involving more operations on
collections are more complex. Therefore, we propose the metrics number of select expressions, number of collection expressions, number of select expressions per extension and number of collection expressions per extension.

The expression language is statically type checked. However it is possible to explicitly declare type casts. However, such a type cast may be difficult to understand. The metrics number of casts and number of casts per extension are proposed to detect this problem.

Xtend supports condition expressions like if expressions or switch...case expressions. To measure the amount of decision points in an extension, the metric extension cyclomatic complexity is used. This metric is related to McCabe’s cyclomatic complexity (47).

5.3.2 Dependency metrics

In Xtend, an Xtend file can import other Xtend files or standard libraries. The following metrics are used to measure the dependency of an Xtend file on other Xtend files or standard libraries. To measure the import dependencies of Xtend files, we define the metrics number of imported standard libraries per Xtend file and number of imported Xtend files per Xtend file. Moreover, the metric number of usages per Xtend file shows how many times an Xtend file is imported by other Xtend files. Since extensions of an imported Xtend file can also be reexported together with local extensions, we propose the metric number of Xtend files reexported per Xtend file.

For more detail on the dependencies between Xtend files, we need more fan-in, fan-out metrics. The metrics number of calls from outside to local extensions per Xtend file (module fan-in) and number of calls from local extensions to outside per Xtend file (module fan-out) are proposed.

On a lower level, we have many extension-level dependencies, such as: extensions depending on extensions, expression extensions depending on create extensions, expression extensions depending on java extensions, create extensions depending on expression extensions, create extensions depending on java extensions, extensions depending on built-in extensions (in standard libraries). The metric number of extension calls per extension is used to measure how many extension calls are made from the body of an extension (extension fan-out). To be more specific about what kind of extension calls,
the metrics \textit{number of create extension calls per extension}, \textit{number of expression extension calls per extension} and \textit{number of java extension calls per extension} are proposed. Furthermore, to see how the dependency of an extension on local extensions in the same module differs from its dependency on other extensions outside (in other modules), we propose the metrics \textit{number of internal extension calls per extension} (internal extension fan-out) and \textit{number of external extension calls per extension} (external extension fan-out).

Similarly to fan-out metrics on extension-level, we define many fan-in metrics on extension-level. The metrics \textit{number of usages per extension}, \textit{number of usages per expression extension}, \textit{number of usages per create extension} and \textit{number of usage per java extension} are proposed together with the metrics \textit{number of internal usages per extension} and \textit{number of external usages per extension}.

Last but not least, the dependency of extensions on built-in extensions is measured by the metric \textit{number of calls to extensions in standard libraries}, \textit{number of calls to extensions in standard libraries per Xtend file} and \textit{number of calls to extensions in standard libraries per extension}. There are some built-in extensions that need to be paid more attention. For example, the built-in extensions in the standard library IO are often used for debugging purposes and showing errors: \texttt{debug()}, \texttt{info()}, \texttt{error()}, etc. The occurrence of these built-in extensions may indicate that the model transformation is still under development. Therefore, we define the metrics \textit{number of calls to IO extensions}, \textit{number of calls to IO extensions per Xtend file} and \textit{number of calls to IO extensions per extension}.

### 5.3.3 Miscellaneous metrics

This section presents some other metrics which are not discussed in previous categories but also very important. The modularity of a model transformation can be reflected by the \textit{number of Xtend files} used to define it. Moreover, the more namespaces imported, the more metatypes can be used in an Xtend file which may imply that model transformation are more complex. Therefore, we define the metrics \textit{number of namespaces per Xtend file} and \textit{number of unused namespaces per Xtend file}. Last but not least, since \texttt{around} advices are specially supported by oAW Xtend but using of \texttt{around} advices could make Xtend model transformations more complex and more difficult to
understand, the metrics \textit{number of arounds} and \textit{number of arounds per Xtend file} are proposed.

5.4 QVTO metrics

This section describes the metrics which have been defined for measuring QVTo model transformations. The metrics described here are specific for QVTo but many of them are similar to the metrics for ATL model transformations \cite{13,14} as well as the metrics for Xtend model transformations presented previously. The QVTo metrics can be divided into following categories:

- **Mapping metrics**: mappings are key elements of a QVTo model transformation. Thus this category is dedicated for all metrics related to mappings.

- **Helper/Query metrics**: metrics related to helpers/queries are presented in this category.

- **Dependency metrics**: There should be a lots of metrics used to show the dependencies between modules, mappings, helpers/queries. They are grouped in this category.

- **Miscellaneous metrics**: Some other metrics which are not fit the discussed categories above are presented in this category.

In the subsections below, each of these categories will be addressed by giving the description and motivation for every metric belonging to it. Furthermore, the relations between metrics and quality attributes will be discussed based on our expectations (also based on ASF+SDF and ATL results). An overview of all the metrics can be found in Tables 7.8, 7.9 and 7.10 in Section 7.3.2.

5.4.1 Mapping metrics

Transformation functions in QVTo are implemented by means of mappings. They map one or more source model elements to one or more target model elements. A measure for the size of a QVTo model transformation is the \textit{number of mappings} defined in all the modules of the model transformation. We propose also the metric \textit{number of abstract mappings} because a mapping can be declared as abstract. An abstract mappings
5. QUALITY ATTRIBUTES AND METRICS

can be inherited by other mappings that makes the transformation more difficult to understand. Moreover, a mapping declaration can be extended with conditions defined using OCL. To execute a mapping, the source object must conform to these conditions. Thus, mappings with condition may make the model transformation more complex. The number of mappings with condition can be one of the metrics to measure the understandability of a model transformation.

A model transformation defined in QVTo can consist of more than one module (in different qvto files). The metric number of mappings per module can show the distribution of mappings over modules. Thus, it can be used to measure the modularity of a model transformation. On a lower level, the size of mappings can be measured by means of number of subobjects per mapping. Subobjects of a mapping are all elements which create the mapping.

It is possible to define mappings with the same name but with different parameters. To measure this kind of mapping overloading we define the metrics number of overloaded mappings and number of mappings per mapping name. The metric number of overloaded mappings shows the amount of mapping names which are overloaded (used by more than one mapping). The metric number of mappings per mapping name shows how many mappings use the same name.

Mappings need to be called explicitly. It may be the case there are some mappings present in a model transformation that are never invoked, i.e., mappings which could be obsolete. To detect such unused mappings, the metric number of unused mappings is included.

In QVTo, there are three types of mapping extension which are more or less similar to the inheritance concepts in object-oriented languages. Firstly, if a mapping inherits other mapping(s), the statement(s) in the ancestor mapping(s) will be executed right before the statements in the body (mapping population section) of the descendant mapping. Secondly, vice versa, if a mapping merges other mapping(s), the statement(s) in the merged mapping(s) will be executed after all the statements in the merging mapping. Thirdly, if a mapping disjuncts other mapping(s), it delegates the call to all the disjuncted mappings and the first disjuncted mapping which is matched will be executed. The use of mapping extension may affect the quality of a model transformation in a similar way as it affects object-oriented software. A mapping deeper in the extension hierarchy may be more error-prone because it inherits/merges a number of
properties from its ancestors. Moreover, in deep hierarchies it is often unclear from which mapping a new mapping should inherit from or merge with. For these reasons, the metrics number of mapping inheritances, number of mapping mergers, number of mapping disjunctions and number of mappings per disjunction are proposed.

A mapping always has at least two mapping parameters self and result as its input and output model elements. The metric number of parameters per mapping counts these two parameters plus any extra parameter. It may be the case that some of these parameters are never used for various reasons. To detect this, we propose to measure the number of unused parameters per mapping. Besides, there are three direction kinds of mapping parameters: in, inout and out. A parameter with in direction means the object passed for read-only access. This is also the default direction if no direction is explicitly declared for a parameter. Next, a parameter with inout direction means the object passed for update, retains its value. Last, a parameter with out direction means the parameter receives new value (not necessarily newly created object). Thus, three metrics are defined to measure three kinds of mapping parameters: number of mapping parameters with in direction per mapping, number of mapping parameters with inout direction per mapping and number of mapping parameters with out direction per mapping.

The body of a mapping may contain operations on collections because collections often need to be manipulated while performing model transformations. The metric number of operations on collections per mapping can be used to measure the understandability of a model transformation because the more collections are manipulated in a mapping, the more complex a mapping could be. It is also possible to define local variables in a mapping. The metric number of variables per mapping is used to measure the use of variables in mappings. To detect obsolete variable definitions, the metric number of unused variables per mapping may be used.

One of the main parts of QVTo transformation language is the ImperativeOCL package which provides us the power of traditional programming language with assignment statements, variables, loops, loop interrupt constructs and conditional execution workflow. Thus, a mapping can contain many decision points. To detect the complexity of a mapping, the metric cyclomatic complexity per mapping which is related to McCabe’s cyclomatic complexity should not be missing in the set of metrics. It measures the amount of decision points in a mapping. In this work, we have only
5. QUALITY ATTRIBUTES AND METRICS

considered decision points are If expressions or Switch...case expressions because they are used most. Thus, one point for future work is to also consider other conditional expressions like loops, loop interrupt constructs, etc.

5.4.2 Helper/Query metrics

Besides mappings, a QVTo model transformation also consists of helpers (queries) which are operations that perform computations on one or more source objects and provides results. Therefore the size of a model transformation is also influenced by the number of helpers it includes. Similar to mapping metrics, the metrics number of helpers per module are proposed to reflect the modularity of a model transformation.

The following metrics which have been defined for mappings can also be proposed for helpers/queries as well. Many helper metrics do not differentiate between helpers and queries because it is not necessary to do so.

- Number of Unused Helpers
- Number of Overloaded Helpers
- Number of Helpers per Helper Name
- Number of Parameters per Helper
- Number of Unused Parameters per Helper
- Number of Helper parameters with \texttt{in} direction per Helper
- Number of Helper parameters with \texttt{inout} direction per Helper
- Number of Helper parameters with \texttt{out} direction per Helper
- Number of Variables per Helper
- Number of Unused Variables per Helper
- Number of Operations on Collections per Helper
- Cyclomatic Complexity per Helper
- Number of Subobjects per Helper
5.4 QVTO metrics

5.4.3 Dependency metrics

In QVTo, similar to Xtend, a module (qvto file) can import other modules. The dependency of modules on other modules can be measured by following metrics. The metrics number of imported modules per module and number of times a module is imported per module are used to measure the import dependencies of modules. To look closer into these dependencies, the metrics number of calls from other modules per module, number of calls from mappings in other modules per module and number of calls from helpers in other modules per module (module fan-in) are proposed together with the metrics number of calls to other modules per module, number of calls to mappings in other modules per module and number of calls to helpers in other modules per module (module fan-out).

Furthermore, the dependencies can be captured on a lower level: mappings depending on mappings, mappings depending on helpers, helpers depending on mappings, helpers depending on helpers, and mappings and helpers depending on built-in functions. Thus, a number of other fan-in, and fan-out metrics are proposed, for example, the metrics number of calls to mappings per mapping (shows how many times a mapping is called), number of calls to mappings from mappings per mapping and number of calls to mappings from helpers per mapping measure the fan-in of mappings. Vice versa, the metrics number of calls from mappings to mappings/helpers per mapping, number of calls from mappings to mappings per mapping and number of calls from mappings to helpers per mapping measure the fan-out of mappings. Similarly, the fan-in, and fan-out metrics of helpers can be defined. An overview of all the fan-in and fan-out metrics can be found in Tables 7.9 and 7.10 in Section 7.3.2.

Last but not least, the dependency of mappings and helpers/queries on built-in functions is measured by the number of calls to built-in functions. Built-in functions are OCL functions and also additional QVTo operations such as resolve expressions. There are some built-in functions that need to be paid more attention. For example, resolve expressions are expressions that inspect trace records to retrieve source or target objects which participated in the previous mapping executions. Resolve expressions are an useful instrument of retrieving trace information. However, it is to be expected that model transformations with a large number of calls to the resolve expressions are harder to understand. There is also the \texttt{log()} function which is used to print information to
the console, that can be used for debugging. The occurrence of this kind of function may indicate that the model transformation is still under development.

5.4.4 Miscellaneous metrics

Some other metrics which are not fit the discussed categories above are presented in this section. The metric number of modules measures the amount of modules that make up a model transformation. There are three types of module in QVTo: Operational Transformation, Library and Blackbox. An Operational Transformation module is used to define the process of converting 1..* source models into 1..* target models. A Library module can not be used to define this process but only mappings and/or helpers which can be imported by other modules. Blackbox module is not considered in this thesis. Thus, the metrics number of Operational Transformation modules and number of Library modules are proposed. In addition, the metric number of Subobjects per module can show how big a module is in term of amount of subobjects it contains. These metrics can provide insight in the size and modularity of a model transformation. It may be the case that there is a module from which mappings or helpers are never called from outside that module. Only one exception is the module that contains the entry operation main(). To detect this, we define the metric number of unused modules, number of unused Operational Transformation modules and number of unused Library modules.

The following metrics provide insight in the context of the model transformation. It is to be expected that model transformations involving more models and metamodels are more complex. Therefore we propose to measure the number of imported metamodels, the number of input models and the number of output models. On a lower level, we propose the metrics number of metamodels per module, the number of input models per Operational Transformation module and the number of output models per Operational Transformation module. For each model parameter there is a model extent which is a container for model objects. A model extent can be referred to with model parameter name. In case the more references to model extents are used, the more complex the model transformation is expected.

Moreover, in QVTo there are some more elements which can be used while defining a model transformation. For example, a constructor is an operation that defines how to create and populate the properties of an instance of a given class. An intermediate
5.5 Relation of metrics to quality attributes

In this section, the relation between the metrics, derived for Xtend (see Section 5.3) and QVTo (see Section 5.4) model transformations, and quality attributes (see Section 5.2) are discussed. First, the metrics for Xtend model transformations are related to quality attributes. Then, the relation between the metrics for QVTo model transformations and the same quality attributes are also considered. The reason for relating metrics to quality attributes is that quality level with respect to a given attribute can be reflected by measure of a metric. Moreover, a metric may be related to more than one quality attribute, and vice versa, a quality attribute may also be affected by one or more metric. These relationships are based on our expectation but an empirical study in the same way as described in (4) can be performed to validate these relationships. This is one point for future work.

5.5.1 Relating Xtend metrics to quality attributes

Tables 5.1, 5.2 and 5.3 summarize the effect of each Xtend metric defined in Section 5.3 on the quality attributes. In these tables, we use the symbols ‘+’ and ‘-’ to denote a ‘positive’ or ‘negative’ effect of a value of a metric on a quality attribute. The higher a value of a metric is, the more a quality attribute is effected. Some examples of how the metrics for Xtend model transformations are related to the quality attributes are discussed as follows.

A measure for the size of an Xtend model transformation is the number of extensions it encompasses. Thus, the number of extensions has a negative effect on the
understandability and modifiability of an Xtend model transformation. The larger an Xtend model transformation is (in terms of the number of extensions), the harder it is to understand or modify. Similarly, as can be seen from its definition, the number of overloaded extensions also has a negative effect on the understandability and modifiability of an Xtend model transformation. However, the number of local variables has a positive effect on the understandability and modifiability because they are often used to logically split the calculation of certain output into separate parts. The extension cyclomatic complexity, which measures the amount of decision points in an extension, has a negative effect on the understandability and modifiability of an Xtend model transformation.

The number of private extensions has a negative effect on the reusability of an Xtend model transformation because private extensions cannot be used from external Xtend files. But the number of usages per Xtend file, which shows the number of times an Xtend file is imported by other Xtend files, has a positive effect on the reusability of an Xtend model transformation.

The number of Xtend files certainly has a positive effect on the modularity of an Xtend model transformation. The number of call to IO extensions has a negative effect on the completeness of a model transformation because they are often used for debugging purposes and showing errors during the development of a model transformation.

One of the metrics, which is related to the consistency of an Xtend model transformation, is the number of variables with same name but different types. This metric has a negative effect on the consistency because it indicates that a model transformation contains conflicting information.

The number of unused extensions, number of unused local variables and number of unused parameters per extension surely have a negative effect on the conciseness of an Xtend model transformation.

As extensions can be cached to improve the performance of Xtend model transformations, the number of cached extensions has a positive effect on the performance of a model transformation.
## 5.5 Relation of metrics to quality attributes

<table>
<thead>
<tr>
<th>Metric</th>
<th>Quality Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Understandability</td>
</tr>
<tr>
<td># Extensions</td>
<td>-</td>
</tr>
<tr>
<td># Private Extensions</td>
<td>-</td>
</tr>
<tr>
<td># Public Extensions</td>
<td>-</td>
</tr>
<tr>
<td># Cached Extensions</td>
<td>-</td>
</tr>
<tr>
<td># Private Cached Extensions</td>
<td>-</td>
</tr>
<tr>
<td># Public Cached Extensions</td>
<td>-</td>
</tr>
<tr>
<td># Non-Cached Extensions</td>
<td>-</td>
</tr>
<tr>
<td># Private Non-Cached Extensions</td>
<td>-</td>
</tr>
<tr>
<td># Public Non-Cached Extensions</td>
<td>-</td>
</tr>
<tr>
<td># Create Extensions</td>
<td>-</td>
</tr>
<tr>
<td># Expression Extensions</td>
<td>-</td>
</tr>
<tr>
<td># Java Extensions</td>
<td>-</td>
</tr>
<tr>
<td># Unused Extensions</td>
<td>-</td>
</tr>
<tr>
<td># Unused Cached Extensions</td>
<td>-</td>
</tr>
<tr>
<td># Unused Non-Cached Extensions</td>
<td>-</td>
</tr>
<tr>
<td># Unused Expression Extensions</td>
<td>-</td>
</tr>
<tr>
<td># Unused Create Extensions</td>
<td>-</td>
</tr>
<tr>
<td># Unused Java Extensions</td>
<td>-</td>
</tr>
<tr>
<td># Extensions with Parameter</td>
<td>-</td>
</tr>
<tr>
<td># Extensions without Parameter</td>
<td>-</td>
</tr>
<tr>
<td># Overloaded Extensions</td>
<td>-</td>
</tr>
<tr>
<td># Extensions with Null body</td>
<td>-</td>
</tr>
<tr>
<td># Calls to Extensions of stdlib</td>
<td>-</td>
</tr>
<tr>
<td># Calls to Extensions of stdlib IO</td>
<td>-</td>
</tr>
<tr>
<td># debug() Calls</td>
<td>-</td>
</tr>
<tr>
<td># info() Calls</td>
<td>-</td>
</tr>
<tr>
<td># error() Calls</td>
<td>-</td>
</tr>
<tr>
<td># syserr() Calls</td>
<td>-</td>
</tr>
<tr>
<td># throwError() Calls</td>
<td>-</td>
</tr>
<tr>
<td># Xtend files</td>
<td>-</td>
</tr>
<tr>
<td># Operation Calls</td>
<td>-</td>
</tr>
<tr>
<td># Local Variables</td>
<td>-</td>
</tr>
<tr>
<td># Unused Local Variables</td>
<td>-</td>
</tr>
<tr>
<td># Variables have same name but different types</td>
<td>-</td>
</tr>
<tr>
<td># Standard Ecore Features</td>
<td>-</td>
</tr>
<tr>
<td># Type Select Expressions</td>
<td>-</td>
</tr>
<tr>
<td># Collection Expressions</td>
<td>-</td>
</tr>
<tr>
<td># Casts</td>
<td>-</td>
</tr>
<tr>
<td># Global Variable Expression</td>
<td>-</td>
</tr>
<tr>
<td># Length of Chain Expressions</td>
<td>-</td>
</tr>
<tr>
<td># Arrounds</td>
<td>-</td>
</tr>
</tbody>
</table>

*Table 5.1: Xtend metrics related to quality attributes (1)*
## 5. QUALITY ATTRIBUTES AND METRICS

<table>
<thead>
<tr>
<th>Metric</th>
<th>Understandability</th>
<th>Modifiability</th>
<th>Reusability</th>
<th>Completeness</th>
<th>Consistency</th>
<th>Concreteness</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td># Extensions per Extend File</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Private Extensions per Extend File</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Public Extensions per Extend File</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Cached Extensions per Extend File</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Private Cached Extensions per Extend File</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Public Cached Extensions per Extend File</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Non-Cached Extensions per Extend File</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Private Non-Cached Extensions per Extend File</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Public Non-Cached Extensions per Extend File</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Expression Extensions per Extend File</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Create Extensions per Extend File</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td># Java Extensions per Extend File</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Unused Extensions per Extend File</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Unused Cached Extensions per Extend File</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Unused Non-Cached Extensions per Extend File</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Unused Expression Extensions per Extend File</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Unused Create Extensions per Extend File</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Unused Java Extensions per Extend File</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Extensions have parameter per Extend File</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Extensions have no parameter per Extend File</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Overloaded Extensions per Extend File</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Parameters per Extension</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Unused Parameters per Extension</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Local Variables per Extension</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Unused Local Variables per Extension</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Null Extensions per Extension File</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Operation Calls per Extension</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># If Expressions per Extension</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Type-Select Expressions per Extension</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Constructor-Call Expressions per Extension</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Boolean-Operations per Extension</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Collection-Expressions per Extension</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Switch-Expressions per Extension</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Cast-Expressions per Extension</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td># Global Variable Expressions per Extension</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Length of Chain Expression per Extension</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Extension metrics</th>
<th>Understandability</th>
<th>Modifiability</th>
<th>Reusability</th>
<th>Completeness</th>
<th>Consistency</th>
<th>Concreteness</th>
<th>Performance</th>
</tr>
</thead>
</table>

### Table 5.2: Xtend metrics related to quality attributes (2)
### 5.5 Relation of metrics to quality attributes

<table>
<thead>
<tr>
<th>Metric</th>
<th>Quality Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Understandability</td>
</tr>
<tr>
<td># Times Imported by other Extension Files per Extension File</td>
<td>+</td>
</tr>
<tr>
<td># Standard Library Extension Calls per Extension</td>
<td>+</td>
</tr>
<tr>
<td># IO Standard Library Extension Calls per Extension File</td>
<td>-</td>
</tr>
<tr>
<td># Extension Calls per Extension</td>
<td>-</td>
</tr>
<tr>
<td># Expression Extension Calls per Extension</td>
<td>-</td>
</tr>
<tr>
<td># Create Extension Calls per Extension</td>
<td>-</td>
</tr>
<tr>
<td># Java Extension Calls per Extension</td>
<td>-</td>
</tr>
<tr>
<td># Times Called per Extension</td>
<td>+</td>
</tr>
<tr>
<td># Times Called per Expression Extension</td>
<td>+</td>
</tr>
<tr>
<td># Times Called per Create Extension</td>
<td>+</td>
</tr>
<tr>
<td># Times Called per Java Extension</td>
<td>+</td>
</tr>
<tr>
<td># Calls to Internal Extensions per Extension</td>
<td>-</td>
</tr>
<tr>
<td># Times called by Internal Extensions per Extension</td>
<td>-</td>
</tr>
<tr>
<td># Calls to External Extensions per Extension</td>
<td>-</td>
</tr>
<tr>
<td># Times called by External Extensions per Extension</td>
<td>-</td>
</tr>
<tr>
<td># Imported Meta-models per Extension File</td>
<td>-</td>
</tr>
<tr>
<td># Arrounds per Extension File</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 5.3: Xtend metrics related to quality attributes (3)

#### 5.5.2 Relating QVTo metrics to quality attributes

Tables 5.4, 5.5 and 5.6 summarize the effect of each QVTo metric defined in Section 5.4 on the quality attributes. Some examples of how the metrics for QVTo model transformations are related to the quality attributes are discussed as follows.

Two of the measures for the size of a QVTo model transformation are the number of mappings and the number of helpers it encompasses. Thus, the number of mappings and the number of helpers have a negative effect on the understandability and modifiability of a QVTo model transformation. Moreover, the appearances of conditions on mappings may make a model transformation more difficult to understand. Therefore, the number of mappings with condition has a negative effect on the understandability. However, the number of intermediate classes and the number of intermediate properties may positively effect the understandability of a QVTo model transformation because they can be used to store temporary results which make the transformation more clear.
5. QUALITY ATTRIBUTES AND METRICS

The number of times a module is imported per module certainly has a positive effect on the reusability of a QVTo model transformation. Similarly, the number of calls from other modules per module also positively effects on the reusability. However, the number of mappings with condition has a negative effect on the reusability because condition will restrict the reuse of mapping. The reusability can also be effected negatively by the number of unused modules.

The modularity of a model transformation can be reflected by the number of modules. This metric has a positive effect on the modularity. The modularity is also positively influenced by some dependency metrics like the number of imported modules per module, the number of times a module is imported per module and the number of call to other modules per module.

The number of calls to log(), assert() has a negative effect on the completeness of a model transformation because they are often used for debugging purposes during the development of a QVTo model transformation. The metric number of variables with same name but different types has a negative effect on the consistency because it indicates that a model transformation contains conflicting information.

The number of unused mappings/helpers, number of unused variables and number of unused parameters per mapping/helper surely have a negative effect on the conciseness of a QVTo model transformation. The performance of a QVTo model transformation is positively influenced by the number of calls to Resolve expressions because in this case, result of a mapping is reused and this mapping is not executed again.
### 5.5 Relation of metrics to quality attributes

<table>
<thead>
<tr>
<th>Metric</th>
<th>Understandability</th>
<th>Modifiability</th>
<th>Reusability</th>
<th>Modularity</th>
<th>Completeness</th>
<th>Consistency</th>
<th>Conciseness</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping metrics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Mappings</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Mappings with Condition</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Overloaded Mappings</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Unused Mappings</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Abstract Mappings</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Mapping Inheritances</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Mapping Mergers</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Mapping Disjunctions</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Helper</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Helpers</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Unused Helpers</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Overloaded Helpers</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Helper Dep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Calls to log()</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Calls to assert()</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Calls to Resolve Expressions</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Miscellaneous metrics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Modules</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Library Modules</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Operational Transformation Modules</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Unused Modules</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Unused Library Modules</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Unused Operational Transformation Modules</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Helper Dep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Input Models</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Output Models</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Imported Metamodels</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Intermediate Classes</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Intermediate Properties</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Variables with same name but different types</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.4: QVTo metrics related to quality attributes (1)
## 5. QUALITY ATTRIBUTES AND METRICS

<table>
<thead>
<tr>
<th>Metric</th>
<th>Quality Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Understandability</td>
</tr>
<tr>
<td># Mappings per Module</td>
<td>-</td>
</tr>
<tr>
<td># Subobjects per Mapping</td>
<td>-</td>
</tr>
<tr>
<td># Parameters per Mapping</td>
<td>-</td>
</tr>
<tr>
<td># Unused Parameters per Mapping</td>
<td>-</td>
</tr>
<tr>
<td># Parameters with direction in per Mapping</td>
<td>-</td>
</tr>
<tr>
<td># Parameters with direction inout per Mapping</td>
<td>-</td>
</tr>
<tr>
<td># Parameters with direction out per Mapping</td>
<td>-</td>
</tr>
<tr>
<td># Mappings per Mapping Name (Overloadings)</td>
<td>-</td>
</tr>
<tr>
<td># Mappings per Disjunction</td>
<td>-</td>
</tr>
<tr>
<td># Variables per Mapping</td>
<td>+</td>
</tr>
<tr>
<td># Unused Variables per Mapping</td>
<td>-</td>
</tr>
<tr>
<td># Operations on Collections per Mapping</td>
<td>-</td>
</tr>
<tr>
<td>Cyclomatic Complexity per Mapping</td>
<td>-</td>
</tr>
<tr>
<td># Helpers per Module</td>
<td>-</td>
</tr>
<tr>
<td># Subobjects per Helper</td>
<td>-</td>
</tr>
<tr>
<td># Helpers per Helper Name</td>
<td>-</td>
</tr>
<tr>
<td># Parameters per Helper</td>
<td>-</td>
</tr>
<tr>
<td># Unused Parameters per Helper</td>
<td>-</td>
</tr>
<tr>
<td># Parameters with direction in per Helper</td>
<td>-</td>
</tr>
<tr>
<td># Parameters with direction inout per Helper</td>
<td>-</td>
</tr>
<tr>
<td># Parameters with direction out per Helper</td>
<td>-</td>
</tr>
<tr>
<td># Variables per Helper</td>
<td>+</td>
</tr>
<tr>
<td># Unused Variables per Helper</td>
<td>-</td>
</tr>
<tr>
<td># Operations on Collections per Helper</td>
<td>-</td>
</tr>
<tr>
<td>Cyclomatic Complexity per Helper</td>
<td>-</td>
</tr>
</tbody>
</table>

*Table 5.5: QVTo metrics related to quality attributes (2)*
### 5.5 Relation of metrics to quality attributes

<table>
<thead>
<tr>
<th>Metric</th>
<th>Quality Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Understandability</td>
</tr>
<tr>
<td># Imported Modules per Module</td>
<td>-</td>
</tr>
<tr>
<td># Times a Module is Imported per Module</td>
<td>-</td>
</tr>
<tr>
<td># Calls from Helpers in other Modules per Module</td>
<td>-</td>
</tr>
<tr>
<td># Calls from Other Modules per Module</td>
<td>-</td>
</tr>
<tr>
<td># Calls to Mappings in Other Modules per Module</td>
<td>-</td>
</tr>
<tr>
<td># Calls to Helpers in Other Modules per Module</td>
<td>-</td>
</tr>
<tr>
<td># Calls to Other Modules per Module</td>
<td>-</td>
</tr>
<tr>
<td># Calls from Mappings to Mappings per Mapping</td>
<td>-</td>
</tr>
<tr>
<td># Calls from Mappings to Helpers per Mapping</td>
<td>-</td>
</tr>
<tr>
<td># Calls from Mappings to Mappings/Helpers per Mapping</td>
<td>-</td>
</tr>
<tr>
<td># Calls to Mappings from Mappings per Mapping</td>
<td>-</td>
</tr>
<tr>
<td># Calls to Mappings from Helpers per Mapping</td>
<td>-</td>
</tr>
<tr>
<td># Calls to Mappings per Mapping</td>
<td>-</td>
</tr>
<tr>
<td># Calls from Helpers to Helpers per Helper</td>
<td>-</td>
</tr>
<tr>
<td># Calls from Helpers to Mappings per Helper</td>
<td>-</td>
</tr>
<tr>
<td># Calls from Helpers to Mappings/Helpers per Helper</td>
<td>-</td>
</tr>
<tr>
<td># Calls to Helpers from Mappings per Helper</td>
<td>-</td>
</tr>
<tr>
<td># Calls to Mappings from Helpers per Helper</td>
<td>-</td>
</tr>
<tr>
<td># Calls to Helpers per Helper</td>
<td>-</td>
</tr>
<tr>
<td># Calls to log() per Module</td>
<td>-</td>
</tr>
<tr>
<td># Calls from Mappings to log() per Mapping</td>
<td>-</td>
</tr>
<tr>
<td># Calls from Helpers to log() per Helper</td>
<td>-</td>
</tr>
<tr>
<td># Calls to assert() per Module</td>
<td>-</td>
</tr>
<tr>
<td># Calls from Mappings to assert() per Mapping</td>
<td>-</td>
</tr>
<tr>
<td># Calls from Helpers to assert() per Helper</td>
<td>-</td>
</tr>
<tr>
<td># Calls to Resolve Expressions per Module</td>
<td>-</td>
</tr>
<tr>
<td># Calls from Mappings to Resolve Expressions per Mapping</td>
<td>-</td>
</tr>
<tr>
<td># Calls from Helpers to Resolve Expressions per Helper</td>
<td>-</td>
</tr>
<tr>
<td># Input Models per Operational Transformation</td>
<td>-</td>
</tr>
<tr>
<td># Output Models per Operational Transformation</td>
<td>-</td>
</tr>
<tr>
<td># Metamodels per Module</td>
<td>-</td>
</tr>
<tr>
<td># Subobjects per Module</td>
<td>-</td>
</tr>
<tr>
<td># Intermediate Classes per Operational Transformation</td>
<td>+</td>
</tr>
<tr>
<td># Intermediate Properties per Operational Transformation</td>
<td>+</td>
</tr>
</tbody>
</table>

Table 5.6: QVTo metrics related to quality attributes (3)
5. QUALITY ATTRIBUTES AND METRICS
Chapter 6

Tools

In the previous chapter, two sets of metrics have been proposed, one for measuring model transformations defined in Xtend and another one for measuring model transformations defined in QVTo. Because model transformations are often too complex, doing quantitative analysis by hand is infeasible and very error-prone. Therefore, it is desirable to perform an automated analysis. The purpose of implementing a tool is to enable automated extraction of the metrics as well as coverage and dependency data which can be visualized. The disadvantage of using metrics for analysis is that they are only numbers. They do not provide information in detail, i.e., which part of the metamodels is used or exactly which transformation function is unused. Therefore, the coverage and dependency information also need to be extracted and visualized to solve this problem.

This chapter describes the three tools that have been developed for this purpose. These tools are used for calculating the metrics as well as extracting coverage and dependency data although they are developed in different ways. The reasons for these different ways are explained in the section of design decisions. The remaining parts of this chapter are as follows. The first section contains the general requirements for developing such a tool. In the second section, the design decisions are discussed. The tools will be described in detail in Sections 6.3, 6.4 and Appendix B.

Not all details of the tools are given in this chapter. Appendices A, B and C can be referred to for more information.
6. TOOLS

6.1 General requirements

The general requirements for implementing a tool are given below.

6.1.1 Platform

The tool should work on the Microsoft Windows XP/7 platform, it would be nice if it could also run on other platforms but this is not necessary. It should run responsively on a normal computer, i.e., equivalent at least to a 2 GHz Pentium 4 processor with 512 MB internal memory.

6.1.2 Input

The tool should be able to use definition files of model transformations (*.ext for Xtend and *.qvto for QVTo) as input. Moreover, metamodel file(s) could also be used as part of the input.

6.1.3 Output

The output of the tool certainly are the metrics calculated as well as the coverage and dependency data extracted from the input model transformations. The dependency data should be able to be used by the tool ExtraVis (46, 49) to visualize dependency graph and the coverage data should be able to be visualized by the tool TreeComparer (49).

6.1.4 Extensibility

It is undoubted that there could be more metrics than the ones presented in this thesis that may be needed later on. Therefore the tool needs to be designed in such a way that extra metrics can be added and their calculation can be implemented easily.

6.2 Design decisions

As mentioned previously, our approach is aimed at assessing the internal quality of model transformations by extracting metrics from the model transformations directly. Moreover, quantitative analysis of model transformations need to be supported by using a tool as it is infeasible to do it by hand. In order to develop such a tool, it is
6.2 Design decisions

vital to reuse the Xtend (QVTo) parser/compiler to parse the files containing Xtend (QVTo) model transformations, see Figure 6.1. The figure shows model transformations (defined in any transformation language like ATL, QVTo or Xtend) are often parsed to obtain abstract syntax trees (AST). One of the main challenges is mastering the structure of the AST obtained after parsing model transformations. Our tool has to be developed based on the AST because the AST contains all information of model transformations which makes the metrics calculation programmable. In fact, there is more than one way in which elements of the AST can be obtained. Therefore it is possible to have more than one way for developing the tool. On the other hand, the structure of AST is specific for each transformation language. Hence, design decisions of quality assessor tool will be considered for each transformation language separately as following sections.

![Figure 6.1: Parsing model transformations](image)

**Figure 6.1: Parsing model transformations** - This figure shows the Xtend (QVTo) parser can be reused to obtain the AST of model transformations

6.2.1 Xtend model transformations extractor tools

In case of Xtend model transformations, the extension file(s) *.ext are parsed by the Xtend parser to get the corresponding AST(s) of model transformations. First and foremost, it is important to know that the AST of Xtend model transformations is represented in the form of in-memory Java objects (POJO’s). Without knowing the structure of the AST and how to access its elements, the extraction of metrics, dependency and coverage can not be programmable. Second, it is infeasible to extract the metrics directly from those in-memory Java objects of the AST. The extraction can be done more efficiently if elements of the AST can be converted into a well-structured data model which is suitable for this purpose. This section discusses three data models which may be used to store all elements of an Xtend model transformation extracted from the AST.
6. TOOLS

A relational database system could be one option for storing the elements. However, the calculation of metrics would be very complicated, especially for dependency metrics. Many data tables would have to be queried together to calculate one metric. Therefore, the performance would be low. This way is not chosen.

Another option is to store the elements also using in-memory Java objects whose structure is suitable for extracting the metrics. As the Xtend parser is written in Java and the AST is in form of in-memory Java objects, it is not too difficult to extract the elements from AST into the data model of Java objects. Some part of the Xtend AST can be seen in Appendix A. We did choose this way to develop a tool. We have built a tool that uses the data model of Java objects to calculate all the Xtend metrics. However, we found that code we had to write is quite complicated to calculate metrics, especially dependency metrics. So even though this way is doable but it will not be easy to upgrade the tool in the future. The implementation of this tool is presented in detail in Section 6.3 and also in Appendix A.

It would be nice to come up with another data model which can make the metrics calculation easier. In fact, we can easily serialize elements of the AST to an XML file with a pre-defined format. Then, we can use Xpath and Xquery to query the data and calculate the metrics quite efficiently. We have developed another tool by following this way, see Appendix B.

![Figure 6.2: Storing Xtend abstract syntax tree](image)

To sum up, two data models have been chosen to implement the tools for calculating Xtend metrics. Figure 6.2 displays a data extractor is developed for extracting elements...
of the Xtend AST(s) to store in these two data models. By using the data model of in-memory Java objects, we implemented algorithms in Java to calculate the metrics in the Java based Xtend M2M extractor tool (JXET), see Section 6.3. For the XML based Xtend M2M extractor tool (XXET, see Appendix B), we used the data model of XML where Xpath and Xquery can be used to calculate the metrics. By using two different tools, we can also compare the results extracted from them.

6.2.2 QVTo model transformations extractor tool

In case of QVTo model transformations, qvto file(s) *.qvto will be parsed by the QVTo parser to get the corresponding AST(s) of model transformations. Similar to Xtend, the AST of QVTo model transformations is also in form of in-memory Java objects (POJO’s) but these are different from those of Xtend since the abstract syntax of the language is different. However, we can see that the two ways of implementing quality assessor tool applied for Xtend can similarly be applied for QVTo as well.

Moreover, an AST of QVTo model transformation can be serialized to a QVTo model which conforms to the metamodels QVTOperational.ecore (for QVTOperational package) and ImperativeOCL.ecore (for ImperativeOCL package). Figure 6.3 shows a
QVTo AST can not only be stored in two data models introduced in Figure 6.2 but it also can be serialized to a QVTo model by using a QVTo serializer. Thus, for measuring QVTo model transformations we can have one more way to develop a measurement tool. That is, QVTo model transformations themselves (or any other EMF based M2M tool, i.e., ATL), can be used to transform a QVTo model of the AST to a model of Metrics which conforms to a predefined Metrics metamodel. In other words, we can use QVTo model transformations to extract the metrics from the QVTo model. We have developed the Model-based QVTo M2M extractor tool (MQET) for measuring QVTo model transformations by using this way, see Section 6.4.

6.3 Java-based Xtend model transformations Extractor Tool

In the previous section, we have chosen in-memory Java objects, whose structure is suitable for extracting the metrics, as a way to store elements of the Xtend AST. Based on that, we can implement algorithms in Java, namely Java based Xtend M2M extractor tool (JXET), to calculate the Xtend metrics. This section shows the design and implementation of JXET.

6.3.1 Design

The architecture of JXET can be seen in Figure 6.4. Here, the Xtend parser is reused to parse Extension file(s) to obtain corresponding Xtend AST(s). Then, a data extractor uses the Xtend AST(s) as its input to extract elements from the Xtend AST(s) to in-memory Java objects (Xtend model), whose structure is suitable for extracting the Xtend metrics. The class diagrams of the data extractor and the Xtend model are presented in Section 6.3.1.2.

Next, the Xtend model is used as input for an extractor. The output of this extractor consists of a set of (in-memory) Xtend metrics, a temporary file containing dependency data and a temporary file containing coverage data. The set of in-memory Xtend metrics is used by a report maker to generate a report file containing all calculated Xtend metrics. The dependency temporary file is used by an ”ExtraVis input generator” to generate an input file which can be used by the ExtraVis tool.
Figure 6.4: Architecture of JXET - This figure shows the architecture of JXET
coverage temporary file is used by another generator to obtain an input file which is visible by using TreeComparer [19].

### 6.3.1.1 Packages

In Figure 6.5, the package diagram for the tool is depicted. The arrows between the packages represent an "uses" relationship. In the rest of this subsection, a short description of the function of the elements in each of the packages is given.

**Figure 6.5: Package diagram of JXET** - This figure shows the Package diagram of JXET

- Package my.xtend.analyzer: this package extracts data from a configuration file by using the package my.xtend.config and makes a call to the package my.xtend.report
to generate the report of metrics. The configuration file contains name of the Xtend model transformation and other information (see Appendix A).

- **Data extractor:** the data extractor is implemented in the package `my.xtend.data.extraction`. This package is used by the package `my.xtend.report` to extract all the elements of the Xtend model transformation to the Xtend model.

- **Xtend model:** the Xtend model is implemented in the package `my.xtend.data` which stores elements of the Xtend AST in a well defined data model suitable for extracting metrics. It uses the package `data.expression` for extracting expression elements from the Xtend AST.

- **Extractor:** the ”center” extractor (see Figure [6.4]) is implemented in the package `metric.calculation` which manipulates data from the Xtend model to calculate metrics, dependency and coverage. The package `my.xtend.util` contains functions for calculating minimum, maximum, average, medium and standard deviation of an array of integer numbers which are used for deriving aggregate metrics.

- **Xtend metrics:** The package `my.xtend.metric` contains metric definitions which are used by the package `my.xtend.metric.data`. All the metrics extracted by the extractor are stored in the package `my.xtend.metric.data`. It is then used by the package `my.xtend.report` for generating the metrics reports.

- **EXTRAVISInputGenerator:** this package is used by the package `metric.calculation` for exporting the dependency data to the input file of ExtraVis.

- **CoverageVisualizationGenerator:** this package is used by the package `metric.calculation` for exporting the coverage data to the input file of TreeComparer.

### 6.3.1.2 Class diagrams

Since the class diagram of the whole tool does not fit in this page, this section displays class diagrams of packages one by one. Some classes are grouped together, this means they belong to the same package. The classes are described in detail in Appendix A.

The class diagram of the data extractor is depicted in Figure [6.6]. As an Xtend model transformation can consist of more than one module (extension file), the `Xtend-PrjExtractor` may use one or more `ExtFileExtractor` to correspondingly extract Xtend
elements in each of modules, see Figure 6.6. Each *ExtFileExtractor* itself can consist of
Extension Statement Extractor(s) for handling the Extension Statement(s) (Expression
Extension Statements, Create Extension Statements and Java Extension Statements)
in the module.

![Class Diagram](image)

**Figure 6.6: Data extractor of JXET** - This figure shows the class diagram of Data extractor of JXET

The output of the data extractor described above is the Java-based data model
for storing Xtend model transformation elements. It can be seen from Figure 6.6
that the *XtendPrjExtractor* extracts data of Xtend model transformation elements and
store them in the *XtendPrjData*. Figure 6.7 shows the class diagram of the Java-based data model for storing Xtend model transformation elements. An *XtendPrjData*
may contain one or more *ExtFileData* which in turn may consist of many Extension
Statement Data objects (corresponding to Expression Extension Statements, Create
Extension Statements and Java Extension Statements). The *ExpressionExtStmData*
and the *CreateExtStmData* contain a set of *ExpressionDatas* which handle expressions
in body of an extension statement. However, the *JavaExtStmData* is used to store data
of a Java Extension Statement that contains a call to a Java method. Thus, its body
is not explored.

Based on the extracted data model, the "center" extractor (see Figure 6.4) is used
to extract Xtend metrics and generate a temporary file of dependency as well as a
6.3 Java-based Xtend model transformations Extractor Tool

Figure 6.7: Java-based data model for Xtend - This figure shows the class diagram of the Java-based data model for Xtend

temporary file of coverage. Figure 6.8 shows the class diagram of this extractor. The XtendPrjMetricCalculator manipulates the data model stored in the XtendPrjData object. The method calculateXtendPrjMetricData() is used to extract all the Xtend metrics from the data model. To extract all the Xtend metrics, for each extension file, the XtendPrjMetricCalculator uses an ExtFileMetricCalculator which extracts the Xtend metrics at the extension file level. On a lower lever, each ExtFileMetricCalculator uses ExtStmMetricCalculators (CreateExtStmMetricCalculator, ExpressionExtStmMetricCalculator and JavaExtStmMetricCalculator) for calculating metrics at the extension statement level. Finally, all the metrics are summarized from lower levels to higher levels which produces the final set of metrics for the entire model transformation.

The XtendPrjMetricCalculator also has the method exportDependenciesData() to generate the temporary file containing dependency data and the method exportCoverageData() to generate the temporary file containing coverage data.

The set of metrics is stored in the model of Xtend metrics, see Figure 6.9. The final set of metrics for the entire model transformation is stored in the XtendPrjMetricData. Actually, there are two sets of metrics, one for storing simple metrics and one for storing aggregate metrics. These metrics are then used by the report maker to produce the
Figure 6.8: Java-based extractor for Xtend - This figure shows the class diagram of the Java-based extractor for Xtend
report of metrics. The class diagrams of the report maker, the ExtraVis input generator and the generator for coverage file can be found in Appendix A.

![Class diagram for Xtend metrics](image)

**Figure 6.9: Data model for Xtend metrics** - This figure shows the class diagram of the data model for Xtend metrics

### 6.3.1.3 Graphical user interface

Graphical user interface (GUI) is not an important aspect of this tool at this moment. However, it would be nice if the tool could be implemented as an Eclipse plugin and has a GUI. This is a point for future work.

### 6.3.2 Implementation

The tool was created using the Java programming language. Since extensibility is an important requirement, a few words are spent on the extensibility of the tool.

As discussed in Section 6.1.4 more metrics than the ones presented in this thesis may be needed later on. The package my.xtend.metric contains metric definitions which are very easy to extend. In this package, the metric names are defined in the
6. TOOLS

form of (Java) enumeration which can be easily extended. Moreover, the framework for calculating metrics is clear, that allows the implementation of metrics calculation (package my.xtend.metric.calculation) from lower levels to higher levels. Thus, a new simple metric or a new aggregate metric can easily be defined and the calculation of it can be implemented by following this framework.

6.4 Model-based QVTo model transformations Extractor Tool

As discussed in the design decisions (Section 6.2.2), the two ways used to develop the tools for Xtend can also be applied for QVTo. However, an AST of QVTo model transformation can also be serialized to a QVTo model that conforms to the metamodels QVTOperational.ecore (for QVTOperational package) and ImperativeOCL.ecore (for ImperativeOCL package). This means we can use QVTo model transformations themselves or any other EMF based M2M tool (i.e., QVTo or ATL) to extract the metrics from QVTo model(s) to a Metrics model which conforms to a predefined Metrics metamodel (14). This section describes how we can use this way to develop a tool for QVTo, named Model-based QVTo-model-transformations Extractor Tool (MQET), which enables automatic collection of the QVTo metrics presented in Section 5.4. Moreover, the tool can also extract coverage and dependency data which can be used by visualization tools to visualize the coverage and dependency graphs.

6.4.1 Design

6.4.1.1 Architecture

The overview architecture of MQET is shown in figure 6.10. First, the input for MQET certainly is QVTo file(s). Model transformations defined in QVTo can be modularized which means that a transformation module may import other transformation modules and library modules. Moreover, in order to compile QVTo file(s) successfully, the corresponding metamodel(s) is implicitly required. Thus, the input for MQET should be all the QVTo file(s) and metamodel file(s) which create the model transformation.

Next, the QVTo compiler of the QVTo engine is reused to compile all the input file(s). This process yields QVTo abstract syntax tree(s) (AST) corresponding to the
6.4 Model-based QVTo model transformations Extractor Tool

Figure 6.10: Architecture of MQET - This figure shows the architecture of the Model-based QVTo-model-transformations Extractor Tool
input file(s). Similar to the Xtend ASTs, these QVTo AST(s) are in the form of in-memory Java objects. One of the advantages of the QVTo engine (also of ATL engine) is its support for serializing the QVTo AST(s) to QVTo model(s). So, instead of extracting metrics from the QVTo AST(s) in the form of in-memory Java objects, it is more convenient to serialize those Java objects to QVTo model(s) first, then extract QVTo metrics from these QVTo model(s). The models are the input for the metrics extractor, the dependency extractor and the coverage extractor. These extractors are themselves model transformations implemented in QVTo.

The metrics M2M extractor consists of several modules which have one operational mapping that maps the QVTo transformation module, and a number of helpers and queries, each for calculating the value of one of the QVTo metrics. The output of the metrics extractor is a model that contains the metrics data. This model is used as input for the Metrics M2T generator which was introduced in the metrics extraction tool for ATL (14). This generator is a model-to-text transformation implemented in Xpand (50). The output of the Metrics M2T generator is a comma separated value file that can be read by a spreadsheet application.

Similarly, the dependency M2M extractor and coverage M2M extractor take the same QVTo model(s) as inputs but the final outputs of these branches are the dependency data file and the coverage data file which are able to be used by some visualization tools.

6.4.1.2 Metamodels

All the models mentioned above have to conform to their metamodels. To implement the extractors, it is necessary to understand these metamodels. In this section, the metamodels will be addressed one by one.

First, a QVTo model conforms to the metamodels QVTOperational.ecore (see figure 6.11) and ImperativeOCL.ecore (see figure 6.13). The QVTOperational package contains general structuring elements and top-level constructions. In figure 6.11 it can be seen that a QVTo model can be an instance of an OperationalTransformation or a Library which extends Module (see figure 6.12). All other elements of a QVTo model transformation such as mappings, helpers, etc can be accessed from that top node.
6.4 Model-based QVTo model transformations Extractor Tool

Figure 6.11: QVTOperational.ecore - This figure shows the QVTOperational metamodel

Figure 6.12: QVTo Module - This figure shows the elements of a Module
6. TOOLS

The figure 6.13 shows all elements of ImperativeOCL package which is an extension to OCL expressions and type system. In fact, these two metamodels also depend on other metamodels (ecore files) such as OCL.ecore, Ecore.ecore and OCLEcore.ecore.

Second, a metrics model conforms to the metamodel Metrics.ecore (see figure 6.14). In chapter 5, we presented two types of metrics: metrics that are measured over the entire transformation and metrics that are measured on a smaller scale, i.e., per module, per mapping, per helper/query. The former type of metric has a single value for the entire transformation. In figure 6.14, we can see that it has type SimpleMetric. The latter type of metric has multiple values for the entire transformation, that is, one for every element that is measured (module, mapping, or helper/query). To assess the transformation as a whole we do not present all of these values. Instead, minimum, maximum, average, median and standard deviations are given for these metrics. This kind of metric has type AggregatedIntegerMetric or AggregatedRealMetric.

A dependency model and a coverage model correspondingly conform to the metamodels CallGraph.ecore (see figure 6.15) and CoverageRelation.ecore (see figure 6.16).

6.4.2 Implementation

This section shows how the extractors and the generators are implemented.

6.4.2.1 Metrics M2M extractor

The metrics M2M extractor consists of a QVTo transformation module, several QVTo library modules and a QVTo blackbox module. The main QVTo transformation module, namely QVTo2Metrics.qvto, has a transformation which takes a QVTo model as its input and a Metrics model as its output. In order to extract QVTo metrics, the module QVTo2Metrics.qvto has to import several QVTo library modules: MappingMetrics.qvto, HelperMetrics.qvto, DependencyMetrics.qvto and MiscellaneousMetrics.qvto. These QVTo library modules contain a number of helpers and queries, each for calculating the value of one of the QVTo metrics. Furthermore, they need to use helpers defined in the module Utilities.qvto for calculating minimum, maximum, average, median, and standard deviation. The module Utilities.qvto itself imports a blackbox (m2m.qvt.oml.ExampleJavaLib) to use some java method(s), i.e., Math.sqrt(). Overview of the relation of the modules can be seen in figure 6.17.
6.4 Model-based QVTo model transformations Extractor Tool

platform/resource/org.eclipse.m2m.qvt.oml.ecore.imperativeocl/model/ImperativeOCL.ecore

- ImperativeOCL
  - AltExp -> ImperativeExpression
  - AssertExp -> ImperativeExpression
  - AssignExp -> ImperativeExpression
  - BlockExp -> ImperativeExpression
  - BreakExp -> ImperativeExpression
  - CatchExp -> ImperativeExpression
  - ComputeExp -> ImperativeExpression
  - ContinueExp -> ImperativeExpression
  - DictLiteralExp -> LiteralExp
  - DictLiteralPart -> EModelElement
  - DictionaryType -> CollectionType
  - ForExp -> ImperativeLoopExp
  - ImperativeExpression -> OCLExpression
  - ImperativeIteratesExp -> ImperativeLoopExp
  - ImperativeLoopExp -> LoopExp, ImperativeExpression
  - InstantiationExp -> ImperativeExpression
  - ListLiteralExp -> LiteralExp
  - ListType -> CollectionType
  - LogExp -> OperationCallExp, ImperativeExpression
  - OrderedTupleLiteralExp -> LiteralExp
  - OrderedTupleLiteralPart -> EModelElement
  - OrderedTupleType -> EClass
  - RaiseExp -> ImperativeExpression
  - ReturnExp -> ImperativeExpression
  - SeverityKind
  - SwitchExp -> ImperativeExpression
  - TryExp -> ImperativeExpression
  - Typedef -> EClass
  - UnlinkExp -> ImperativeExpression
  - UnpackExp -> ImperativeExpression
  - VariableInitExp -> ImperativeExpression
  - WhileExp -> ImperativeExpression

Figure 6.13: ImperativeOCL.ecore - This figure shows the ImperativeOCL metamodel
6. TOOLS

Figure 6.14: Metrics.ecore - Metrics metamodel

Figure 6.15: CallGraph.ecore - Dependencies Relation metamodel
Figure 6.16: CoverageRelation.ecore - Coverage Relation metamodel
6. TOOLS

Figure 6.17: Metrics M2M extractor - This figure shows the relation of the modules in the Metrics M2M extractor.

The module QVTo2Metrics.qvto has an operational mapping which takes a transformation module (a QVTo model) as its input. The output of this mapping is a Metrics model.

```plaintext
import MappingMetrics;
import HelperMetrics;
import MiscellaneousMetrics;
import DependencyMetrics;

modeltype QVTO uses "http://www.eclipse.org/QVT/1.0.0/Operational";
modeltype METRICS uses "http:///Metrics.ecore";

transformation QVTo2Metrics(in qvtoModel:QVTO, out metricModel:METRICS);

main() {
    qvtoModel.objects()[QVTO::expressions::OperationalTransformation] -> map qvto2metrics();
}

mapping QVTO::expressions::OperationalTransformation::qvto2metrics() : METRICS::Metrics {
    TrafoName := self.name;
    allModules := OrderedSet { self };
    getAllModules(self, result);

    // Miscellaneous Metrics
    SimpleMetrics += NumModules(allModules);
    ...
```
6.4 Model-based QVTo model transformations Extractor Tool

So, name of the transformation module is mapped to the TrafoName of the output Metrics model. More important, in order to assess the entire QVTo model transformation, the main module and all the imported modules of the QVTo model transformation have to be collected completely before calculating QVTo metrics. This can be done by using the helper getAllModules() and the intermediate property allModules as follows:

### intermediate property
```markdown
METRICS::Metrics::allModules: OrderedSet(QVTO::expressions::Module)
```

```java
helper getAllModules(trans : QVTO::expressions::Module, m : METRICS::Metrics) {
    if (trans.moduleImport.importedModule <> null) then{
        trans.moduleImport.importedModule->forEach(mdl){
            m.allModules += mdl.oclAsType(QVTO::expressions::Module);

            // recursively get imported modules
            getAllModules(mdl.oclAsType(QVTO::expressions::Module), m);
        }
    }
    endif;
    return;
}
```

The helper getAllModules() is called recursively to extract all the imported modules and keep them in the intermediate property allModules. Based on the set of modules, helpers and queries in other modules (MappingMetrics.qvto, HelperMetrics.qvto, DependencyMetrics.qvto and MiscellaneousMetrics.qvto) are called to calculate the QVTo metrics. For example, the query NumModules in the module MiscellaneousMetrics.qvto is used to calculate the SimpleMetric "Number of Modules".

```java
library MiscellaneousMetrics;

import Utilities;

modeltype QVTO uses "http://www.eclipse.org/QVT/1.0.0/Operational";
modeltype METRICS uses "http:///Metrics.ecore";

query NumModules(transModules : OrderedSet(QVTO::expressions::Module)) : SimpleMetric{
    var name := "Number of Modules";
    var value := transModules->size();
    return getNewSimpleMetric(name, value);
}
```

...
Every query which is used for calculating a metric always calls one of the helpers in the module Utilities.qvto to instantiate a new metric.

```qvt
library Utilities;

import m2m.qvt.oom.ExampleJavaLib;

modeltype METRICS uses "http:///Metrics.ecore";

helper getNewSimpleMetric(name : String, value : Integer): SimpleMetric{
    var m : SimpleMetric := object SimpleMetric{};
    m.Metric := name;
    m.Value := value;
    return m;
}

helper getNewAggregatedIntegerMetric(name : String, intseq : Sequence(Integer)):
    AggregatedIntegerMetric{
    var m : AggregatedIntegerMetric;
    m := object AggregatedIntegerMetric{};
    m.Metric := name;
    if (intseq.size() > 0) then {
        m.Minimum := getIntSeqMinimum(intseq);
        m.Maximum := getIntSeqMaximum(intseq);
        m.Average := getIntSeqAverage(intseq);
        m.Median := getIntSeqMedian(intseq);
        m.StandardDeviation := getIntSeqStdDev(intseq);
    }endif;
    return m;
}
...
```

The full QVTo code for calculating QVTo metrics can be seen in Appendix C.

6.4.2.2 Metrics M2T generator

As we know, the output of the Metrics M2M extractor is a Metrics model. This model is used as input for the Metrics M2T generator. This generator is a model-to-text transformation implemented in Xpand. The output of the Metrics M2T generator is a comma separated value file that can be read by a spreadsheet application. So
6.4 Model-based QVTo model transformations Extractor Tool

the idea is to transform a Metrics model to a report file which is easier to read. The implementation of this Metrics M2T generator can be found in Appendix C.
Chapter 7

Case studies

The purpose of performing case studies is to assess whether the tools described in Sections 6.3 and Appendix B fulfill their tasks. That is, the metrics, coverage and dependency data extracted from (Xtend, QVTo) model transformations of several case studies by using these tools are to be analyzed and validated. The inputs of these case studies are definition files of (Xtend, QVTo) model transformations taken from research partners and self written model transformations.

There are five case studies have been performed. An overview of case studies is shown in Table 7.1. In the first two case studies, the metrics and dependency data extracted from two model transformations defined in Xtend are analyzed. For the other three case studies, we analyze the metrics and coverage data extracted from three model transformations defined in QVTo.

<table>
<thead>
<tr>
<th>#</th>
<th>Transformation</th>
<th>Supplier</th>
<th>Language</th>
<th># File(s)</th>
<th>To be analyzed</th>
<th>Tool(s) used</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>book2pub</td>
<td>Self written</td>
<td>Xtend</td>
<td>1</td>
<td>Metrics</td>
<td>JXET</td>
</tr>
<tr>
<td>2</td>
<td>Sync2Async</td>
<td>Research partner</td>
<td>Xtend</td>
<td>4</td>
<td>Metrics, Dependencies</td>
<td>JXET, XXET</td>
</tr>
<tr>
<td>4</td>
<td>book2pub</td>
<td>Self written</td>
<td>QVTo</td>
<td>1</td>
<td>Metrics</td>
<td>MQET</td>
</tr>
<tr>
<td>3</td>
<td>QVTo2Metrics</td>
<td>Self written</td>
<td>QVTo</td>
<td>6</td>
<td>Metrics, Coverage</td>
<td>MQET</td>
</tr>
<tr>
<td>5</td>
<td>ucf2cif</td>
<td>Research partner</td>
<td>QVTo</td>
<td>5</td>
<td>Metrics</td>
<td>MQET</td>
</tr>
</tbody>
</table>

Table 7.1: Case studies - An overview of case studies.

The remainder of this chapter describes the results of the case studies as well as the analysis on the results.
7. CASE STUDIES

7.1 Case study book2pub defined in Xtend

In this case study, we choose to analyze a simple Xtend model transformation with the help of the Java-based Xtend Extractor Tool (JXET, see Section 6.3).

7.1.1 Description

The book to publication transformation has been given in ATL zoo but here it is solved using Xtend, see Listing 7.1. Its purpose is to transform a book model that adheres to the book metamodel into a publication model that adheres to the publication metamodel. The transformation rules are follows. The title of a publication should be the title of a book. The authors attribute of a publication should be set to the concatenation of the authors of each of the chapters separated by the word and and there should be no duplicates. The number of pages of the publication should be the sum of the number of all of the pages of the chapters in the book.

Listing 7.1: The book2pub transformation defined in Xtend

```java
import Publication;
import Book;

create PublicationMM transform( BookMM bookMM ):
    setPubs( bookMM.books.tranfo() );

create Publication tranfo( Book book ):
    setTitle ( book.title ) ->
    setNbPages( book.chapters.collect(e | e.nbPages).getSum() ) ->
    setAuthors( book.chapters.collect( e | e.author).toList().getSumStr() )
;

int getSum(List[int] in) :
    if in.size == 0 then 0 else in.first() + in.withoutFirst().getSum()
;

String getSumStr(List[String] in) :
    if in.size == 0 then ""
    else if in.withoutFirst().getSumStr() == "" then
        in.first()
    else
        in.first() + " and " + in.withoutFirst().getSumStr()
;
```

80
7.1 Case study book2pub defined in Xtend

```java
String getAuthors(List<String> in) {
    if (in.size() == 0) return "";
    else return in.first() + " and " + in.withoutFirst().getSumStr();
}
```

7.1.2 Metrics extracted using JXET

In this section, we present the metric values that were automatically extracted from the book2pub transformation by JXET (see Section 6.3). Table 7.2 contains the simple metrics and Table 7.3 contains the aggregate metrics. The first column of the tables states the category a metric belong to. The second column contains name of the metric itself. The remainder of the column(s) contains the value(s) for a metric.

The metrics values give some insight into the transformation. It is clear that the size of the transformation is very small with only 5 extensions (2 create extensions, 3 expression extensions and 0 Java extension) defined and all of them are public (by default, since no `private` keyword used). Because create extension are always cached, 2 cached extensions are counted. Also, 1 extension which, is unused, is of type expression extension and non-cached. All 5 extensions are using parameter(s). In Section 5.3.2, we suggested that the occurrence of built-in extensions of standard library IO such as debug(), info(), syserr() could indicate that the model transformation is still under development. From the metrics extracted, we can see no call is made to these built-in extensions. This implies that the model transformation is mature. Moreover, some other metrics values such as 30 operation calls, 2 collection expressions and 2 chain expressions also indicate the small size of the transformation.

The values of aggregate metrics could show us more detail of the transformation. For example, within 30 operation calls, each of 5 extensions use an average of 6 operation calls, minimum is 2 and maximum is 10. It is also interesting to analyze value of the metric Extension Cyclomatic Complexity. The cyclomatic complexity of an extension is number of "paths" in which the "body" of an extension is executed. Initially, the cyclomatic complexity of an extension is 1, and base on the number of if...else and switch...case expressions in the body of an extension, its cyclomatic complexity will be increased. The highest cyclomatic complexity of an extension in the transformation is 81.
5. If the cyclomatic complexity of an extension goes beyond a threshold, for example 15, that extension could be too complex and should be rewritten in a simpler way.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td># Extensions</td>
<td>5</td>
</tr>
<tr>
<td># Private Extensions</td>
<td>0</td>
</tr>
<tr>
<td># Public Extensions</td>
<td>5</td>
</tr>
<tr>
<td># Cached Extensions</td>
<td>2</td>
</tr>
<tr>
<td># Private Cached Extensions</td>
<td>0</td>
</tr>
<tr>
<td># Public Cached Extensions</td>
<td>2</td>
</tr>
<tr>
<td># Non-Cached Extensions</td>
<td>3</td>
</tr>
<tr>
<td># Private Non-Cached Extensions</td>
<td>0</td>
</tr>
<tr>
<td># Public Non-Cached Extensions</td>
<td>3</td>
</tr>
<tr>
<td># Create Extensions</td>
<td>2</td>
</tr>
<tr>
<td># Expression Extensions</td>
<td>3</td>
</tr>
<tr>
<td># Java Extensions</td>
<td>0</td>
</tr>
<tr>
<td># Unused Extensions</td>
<td>1</td>
</tr>
<tr>
<td># Unused Cached Extensions</td>
<td>0</td>
</tr>
<tr>
<td># Unused Non-Cached Extensions</td>
<td>1</td>
</tr>
<tr>
<td># Unused Expression Extensions</td>
<td>1</td>
</tr>
<tr>
<td># Unused Create Extensions</td>
<td>0</td>
</tr>
<tr>
<td># Unused Java Extensions</td>
<td>0</td>
</tr>
<tr>
<td># Extensions with Parameter</td>
<td>5</td>
</tr>
<tr>
<td># Extensions without Parameter</td>
<td>0</td>
</tr>
<tr>
<td># Overloaded Extensions</td>
<td>0</td>
</tr>
<tr>
<td># Extensions with Null body</td>
<td>0</td>
</tr>
<tr>
<td># Calls to Extensions of stdlib</td>
<td>0</td>
</tr>
<tr>
<td># Calls to Extensions of stdlib IO</td>
<td>0</td>
</tr>
<tr>
<td># debug() Calls</td>
<td>0</td>
</tr>
<tr>
<td># info() Calls</td>
<td>0</td>
</tr>
<tr>
<td># error() Calls</td>
<td>0</td>
</tr>
<tr>
<td># syserr() Calls</td>
<td>0</td>
</tr>
<tr>
<td># throwError() Calls</td>
<td>0</td>
</tr>
<tr>
<td># Xtend files</td>
<td>0</td>
</tr>
<tr>
<td># Operation Calls</td>
<td>30</td>
</tr>
<tr>
<td># Local Variables</td>
<td>0</td>
</tr>
<tr>
<td># Unused Local Variables</td>
<td>0</td>
</tr>
<tr>
<td># Variables have same name but different types</td>
<td>0</td>
</tr>
<tr>
<td># Standard Ecore Features</td>
<td>0</td>
</tr>
<tr>
<td># Type Select Expressions</td>
<td>0</td>
</tr>
<tr>
<td># Collection Expressions</td>
<td>2</td>
</tr>
<tr>
<td># Casts</td>
<td>0</td>
</tr>
<tr>
<td># Global Variable Expression</td>
<td>0</td>
</tr>
<tr>
<td># Length of Chain Expressions</td>
<td>2</td>
</tr>
<tr>
<td># Arounds</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 7.2: Xtend metrics extracted from the book to publication transformation (1)
7.1 Case study book2pub defined in Xtend

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># Extensions per Extend File</td>
<td>5</td>
<td>5</td>
<td>5.0</td>
<td>5.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Private Extensions per Extend File</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Public Extensions per Extend File</td>
<td>5</td>
<td>5</td>
<td>5.0</td>
<td>5.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Cached Extensions per Extend File</td>
<td>2</td>
<td>2</td>
<td>2.0</td>
<td>2.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Private Cached Extensions per Extend File</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Public Cached Extensions per Extend File</td>
<td>2</td>
<td>2</td>
<td>2.0</td>
<td>2.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Non-Cached Extensions per Extend File</td>
<td>3</td>
<td>3</td>
<td>3.0</td>
<td>3.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Private Non-Cached Extensions per Extend File</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Public Non-Cached Extensions per Extend File</td>
<td>3</td>
<td>3</td>
<td>3.0</td>
<td>3.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Expression Extensions per Extend File</td>
<td>3</td>
<td>3</td>
<td>3.0</td>
<td>3.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Create Extensions per Extend File</td>
<td>2</td>
<td>2</td>
<td>2.0</td>
<td>2.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Java Extensions per Extend File</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Unused Extensions per Extend File</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Unused Cached Extensions per Extend File</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Unused Non-Cached Extensions per Extend File</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Unused Expression Extensions per Extend File</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Unused Create Extensions per Extend File</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Unused Java Extensions per Extend File</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Extensions have parameter per Extend File</td>
<td>5</td>
<td>5</td>
<td>5.0</td>
<td>5.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Extensions have no parameter per Extend File</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Overloaded Extensions per Extend File</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Parameters per Extension</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Unused Parameters per Extension</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Local Variables per Extension</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Unused Local Variables per Extension</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Null Extensions per Extension File</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Operation Calls per Extension</td>
<td>2</td>
<td>10</td>
<td>6.0</td>
<td>6.0</td>
<td>2.92</td>
</tr>
<tr>
<td># If Expressions per Extension</td>
<td>0</td>
<td>2</td>
<td>0.8</td>
<td>1.0</td>
<td>0.84</td>
</tr>
<tr>
<td># Type-Select Expressions per Extension</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Constructor-Call Expressions per Extension</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Boolean-Operations per Extension</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Collection-Expressions per Extension</td>
<td>0</td>
<td>2</td>
<td>0.4</td>
<td>0.0</td>
<td>0.89</td>
</tr>
<tr>
<td># Switch-Expressions per Extension</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Cast-Expressions per Extension</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Global Variable Expressions per Extension</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Length of Chain Expression per Extension</td>
<td>0</td>
<td>2</td>
<td>0.4</td>
<td>0.0</td>
<td>0.89</td>
</tr>
<tr>
<td>Extension Cyclomatic Complexity</td>
<td>1</td>
<td>5</td>
<td>2.6</td>
<td>3.0</td>
<td>1.67</td>
</tr>
</tbody>
</table>

Table 7.3: Xtend metrics extracted from the book to publication transformation (2)
### 7. CASE STUDIES

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># Times Imported by other Extension Files per Extension File</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Standard Library Extension Calls per Extension</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># IO Standard Library Extension Calls per Extension File</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Extension Calls per Extension</td>
<td>1</td>
<td>2</td>
<td>1.4</td>
<td>1.0</td>
<td>0.55</td>
</tr>
<tr>
<td># Expression Extension Calls per Extension</td>
<td>0</td>
<td>2</td>
<td>1.2</td>
<td>1.0</td>
<td>0.84</td>
</tr>
<tr>
<td># Java Extension Calls per Extension</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Times Called per Extension</td>
<td>0</td>
<td>3</td>
<td>1.2</td>
<td>1.0</td>
<td>1.3</td>
</tr>
<tr>
<td># Times Called per Expression Extension</td>
<td>0</td>
<td>3</td>
<td>1.0</td>
<td>0.0</td>
<td>1.41</td>
</tr>
<tr>
<td># Times Called per Create Extension</td>
<td>0</td>
<td>1</td>
<td>0.2</td>
<td>0.0</td>
<td>0.45</td>
</tr>
<tr>
<td># Times Called per Java Extension</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls to Internal Extensions per Extension</td>
<td>1</td>
<td>2</td>
<td>1.4</td>
<td>1.0</td>
<td>0.55</td>
</tr>
<tr>
<td># Times called by Internal Extensions per Extension</td>
<td>0</td>
<td>3</td>
<td>1.2</td>
<td>1.0</td>
<td>1.3</td>
</tr>
<tr>
<td># Calls to External Extensions per Extension</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Times called by External Extensions per Extension</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Imported Meta-models per Extension File</td>
<td>2</td>
<td>2</td>
<td>2.0</td>
<td>2.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Arrounds per Extension File</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 7.4: Xtend metrics extracted from the book to publication transformation (3)

### 7.2 Case study Sync2Async defined in Xtend

#### 7.2.1 Description

SLCO (Simple Language of Communicating Objects) \(^{(52)}\) is a domain-specific language that provides constructs for specifying the structure and behavior of systems consisting of objects that operate in parallel and communicate with each other.

In the SLCO language, channels are used to communicate between objects. These channels can either be synchronous or asynchronous. In case communication is synchronous, both sender and receiver need to be available before a signal can be sent. In this way, sender and receiver synchronize on communication. In case communication is asynchronous, a sender can send a signal and proceed with its execution even though the receiver is not yet ready to receive the signal.

Suppose that a system expressed in SLCO needs to be migrated from a synchronous to an asynchronous environment. In this case it is no longer possible to communicate using synchronous channels. However, the system needs to exhibit the same behavior. Therefore, the system needs to be modified. This should be done by model transformations that replace all synchronous channels in an SLCO model by asynchronous
channels. Furthermore, since the system needs to exhibit the same behavior, the transformations should adapt an SLCO model in such a way that synchronization over these newly created asynchronous channels is achieved.

The model transformation defined in Xtend for doing this task can be found in Appendix D.

7.2.2 Metrics extracted using JXET and XXET

Since we have developed not only one but two tools (JXET, see Section 6.3 and XXET, see Appendix D) for measuring Xtend model transformations, we present the results using these tools to extract metrics from the same Xtend model transformation presented in this case study.

7.2.2.1 Metrics extracted using JXET

This section shows the results of using JXET to extract metrics from the synchronous to asynchronous communication transformation. These results will be compared with the results of using another tool, XXET, which extracts metrics from the same synchronous to asynchronous communication transformation.

In the Table 7.5, the metric number of Xtend files shows that the model transformation contains 4 Xtend files. This indicates that the model transformation is modularized. The total number of 34 extensions certainly reflects the size of the transformation. All the extensions using at least one parameter. Moreover, there are 13 over 34 extensions are overloaded which suggest that the transformation is not easy to understand. The number of calls to IO extensions is only 2 over 34 extensions. This implies that the transformation is quite stable because there are not many calls to debugging functions.
### Table 7.5: Xtend metrics extracted from the synchronous to asynchronous communication transformation (1)

In the Table 7.6, it can be seen that the smallest Xtend file contains 3 extensions while the largest Xtend file contains 18 extensions. Thus, the modules are not very

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td># Extensions</td>
<td>34</td>
</tr>
<tr>
<td># Private Extensions</td>
<td>17</td>
</tr>
<tr>
<td># Public Extensions</td>
<td>17</td>
</tr>
<tr>
<td># Cached Extensions</td>
<td>6</td>
</tr>
<tr>
<td># Private Cached Extensions</td>
<td>2</td>
</tr>
<tr>
<td># Public Cached Extensions</td>
<td>4</td>
</tr>
<tr>
<td># Non-Cached Extensions</td>
<td>28</td>
</tr>
<tr>
<td># Private Non-Cached Extensions</td>
<td>15</td>
</tr>
<tr>
<td># Public Non-Cached Extensions</td>
<td>13</td>
</tr>
<tr>
<td># Create Extensions</td>
<td>1</td>
</tr>
<tr>
<td># Expression Extensions</td>
<td>33</td>
</tr>
<tr>
<td># Java Extensions</td>
<td>0</td>
</tr>
<tr>
<td># Unused Extensions</td>
<td>3</td>
</tr>
<tr>
<td># Unused Cached Extensions</td>
<td>1</td>
</tr>
<tr>
<td># Unused Non-Cached Extensions</td>
<td>2</td>
</tr>
<tr>
<td># Unused Expression Extensions</td>
<td>3</td>
</tr>
<tr>
<td># Unused Create Extensions</td>
<td>0</td>
</tr>
<tr>
<td># Unused Java Extensions</td>
<td>0</td>
</tr>
<tr>
<td># Extensions with Parameter</td>
<td>34</td>
</tr>
<tr>
<td># Extensions without Parameter</td>
<td>0</td>
</tr>
<tr>
<td># Overloaded Extensions</td>
<td>13</td>
</tr>
<tr>
<td># Extensions with Null body</td>
<td>0</td>
</tr>
<tr>
<td># Calls to Extensions of stdlib</td>
<td>4</td>
</tr>
<tr>
<td># Calls to Extensions of stdlib IO</td>
<td>2</td>
</tr>
<tr>
<td># debug() Calls</td>
<td>0</td>
</tr>
<tr>
<td># info() Calls</td>
<td>0</td>
</tr>
<tr>
<td># error() Calls</td>
<td>0</td>
</tr>
<tr>
<td># syserr() Calls</td>
<td>2</td>
</tr>
<tr>
<td># throwError() Calls</td>
<td>0</td>
</tr>
<tr>
<td># Xtend files</td>
<td>4</td>
</tr>
<tr>
<td># Operation Calls</td>
<td>157</td>
</tr>
<tr>
<td># Local Variables</td>
<td>9</td>
</tr>
<tr>
<td># Unused Local Variables</td>
<td>0</td>
</tr>
<tr>
<td># Variables have same name but different types</td>
<td>2</td>
</tr>
<tr>
<td># Standard Ecore Features</td>
<td>6</td>
</tr>
<tr>
<td># Type Select Expressions</td>
<td>0</td>
</tr>
<tr>
<td># Collection Expressions</td>
<td>15</td>
</tr>
<tr>
<td># Casts</td>
<td>14</td>
</tr>
<tr>
<td># Global Variable Expression</td>
<td>0</td>
</tr>
<tr>
<td># Length of Chain Expressions</td>
<td>37</td>
</tr>
<tr>
<td># Arrounds</td>
<td>0</td>
</tr>
</tbody>
</table>
balanced. The average number of parameters per extension is around 2, which is normal. However, there exists extension(s) with 5 parameters which could be complicated. Similarly, the biggest cyclomatic complexity of an extension is 12 which is high. This also indicates that there exist extension(s) which have many decision points.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># Extensions per Extend File</td>
<td>3</td>
<td>18</td>
<td>8.5</td>
<td>6.5</td>
<td>6.56</td>
</tr>
<tr>
<td># Private Extensions per Extend File</td>
<td>0</td>
<td>17</td>
<td>4.25</td>
<td>0.0</td>
<td>8.5</td>
</tr>
<tr>
<td># Public Extensions per Extend File</td>
<td>1</td>
<td>7</td>
<td>4.25</td>
<td>4.5</td>
<td>2.75</td>
</tr>
<tr>
<td># Cached Extensions per Extend File</td>
<td>0</td>
<td>2</td>
<td>1.5</td>
<td>2.0</td>
<td>1.0</td>
</tr>
<tr>
<td># Private Cached Extensions per Extend File</td>
<td>0</td>
<td>2</td>
<td>0.5</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td># Public Cached Extensions per Extend File</td>
<td>0</td>
<td>2</td>
<td>1.0</td>
<td>1.0</td>
<td>1.15</td>
</tr>
<tr>
<td># Non-Cached Extensions per Extend File</td>
<td>1</td>
<td>16</td>
<td>7.0</td>
<td>5.5</td>
<td>6.38</td>
</tr>
<tr>
<td># Private Non-Cached Extensions per Extend File</td>
<td>0</td>
<td>15</td>
<td>3.75</td>
<td>0.0</td>
<td>7.5</td>
</tr>
<tr>
<td># Public Non-Cached Extensions per Extend File</td>
<td>1</td>
<td>6</td>
<td>3.25</td>
<td>3.0</td>
<td>2.63</td>
</tr>
<tr>
<td># Expression Extensions per Extend File</td>
<td>3</td>
<td>17</td>
<td>8.25</td>
<td>6.5</td>
<td>6.08</td>
</tr>
<tr>
<td># Create Extensions per Extend File</td>
<td>0</td>
<td>1</td>
<td>0.25</td>
<td>0.0</td>
<td>0.5</td>
</tr>
<tr>
<td># Java Extensions per Extend File</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Unused Extensions per Extend File</td>
<td>0</td>
<td>1</td>
<td>0.75</td>
<td>1.0</td>
<td>0.5</td>
</tr>
<tr>
<td># Unused Cached Extensions per Extend File</td>
<td>0</td>
<td>1</td>
<td>0.25</td>
<td>0.0</td>
<td>0.5</td>
</tr>
<tr>
<td># Unused Non-Cached Extensions per Extend File</td>
<td>0</td>
<td>1</td>
<td>0.5</td>
<td>0.5</td>
<td>0.58</td>
</tr>
<tr>
<td># Unused Expression Extensions per Extend File</td>
<td>0</td>
<td>1</td>
<td>0.75</td>
<td>1.0</td>
<td>0.5</td>
</tr>
<tr>
<td># Unused Create Extensions per Extend File</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Unused Java Extensions per Extend File</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Extensions have parameter per Extend File</td>
<td>3</td>
<td>18</td>
<td>8.5</td>
<td>6.5</td>
<td>6.56</td>
</tr>
<tr>
<td># Extensions have no parameter per Extend File</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Overloaded Extensions per Extend File</td>
<td>0</td>
<td>7</td>
<td>3.25</td>
<td>3.0</td>
<td>2.99</td>
</tr>
<tr>
<td># Parameters per Extension</td>
<td>1</td>
<td>5</td>
<td>2.15</td>
<td>2.0</td>
<td>0.78</td>
</tr>
<tr>
<td># Unused Parameters per Extension</td>
<td>0</td>
<td>1</td>
<td>0.06</td>
<td>0.0</td>
<td>0.24</td>
</tr>
<tr>
<td># Local Variables per Extension</td>
<td>0</td>
<td>3</td>
<td>0.26</td>
<td>0.0</td>
<td>0.75</td>
</tr>
<tr>
<td># Unused Local Variables per Extension</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Null Extensions per Extension</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Operation Calls per Extension</td>
<td>1</td>
<td>24</td>
<td>4.62</td>
<td>2.0</td>
<td>5.47</td>
</tr>
<tr>
<td># If Expressions per Extension</td>
<td>0</td>
<td>3</td>
<td>0.24</td>
<td>0.0</td>
<td>0.65</td>
</tr>
<tr>
<td># Type-Select Expressions per Extension</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Constructor-Call Expressions per Extension</td>
<td>0</td>
<td>2</td>
<td>0.12</td>
<td>0.0</td>
<td>0.41</td>
</tr>
<tr>
<td># Boolean-Operations per Extension</td>
<td>0</td>
<td>5</td>
<td>0.68</td>
<td>0.0</td>
<td>1.17</td>
</tr>
<tr>
<td># Collection-Expressions per Extension</td>
<td>0</td>
<td>4</td>
<td>0.44</td>
<td>0.0</td>
<td>0.96</td>
</tr>
<tr>
<td># Switch-Expressions per Extension</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Cast-Expressions per Extension</td>
<td>0</td>
<td>2</td>
<td>0.41</td>
<td>0.0</td>
<td>0.66</td>
</tr>
<tr>
<td># Global Variable Expressions per Extension</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Length of Chain Expression per Extension</td>
<td>0</td>
<td>9</td>
<td>1.09</td>
<td>0.0</td>
<td>1.82</td>
</tr>
<tr>
<td>Extension Cyclomatic Complexity</td>
<td>1</td>
<td>12</td>
<td>2.12</td>
<td>1.0</td>
<td>2.2</td>
</tr>
</tbody>
</table>

Table 7.6: Xtend metrics extracted from the synchronous to asynchronous communication transformation (2)
In the Table 7.7, the most imported Xtend file is imported by 3 other Xtend files. So, this Xtend file must contain helper functions which can highly be reused. From the values for the metric number of expression extension calls per extension, we can conclude that there are a lot of calls to a number of expression extensions. This can be explained easily since there is only 1 create extension which defines the model transformation while there are 33 expression extensions which are used as helper functions for the transformation.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Min.</th>
<th>Max.</th>
<th>Avg.</th>
<th>Med.</th>
<th>StdDev</th>
</tr>
</thead>
<tbody>
<tr>
<td># Times Imported by other Extension Files per Extension File</td>
<td>0</td>
<td>3</td>
<td>1.25</td>
<td>1.0</td>
<td>1.26</td>
</tr>
<tr>
<td># Standard Library Extension Calls per Extension</td>
<td>0</td>
<td>2</td>
<td>0.12</td>
<td>0.0</td>
<td>0.41</td>
</tr>
<tr>
<td># IO Standard Library Extension Calls per Extension File</td>
<td>0</td>
<td>2</td>
<td>0.5</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td># Extension Calls per Extension</td>
<td>0</td>
<td>12</td>
<td>1.74</td>
<td>1.0</td>
<td>2.22</td>
</tr>
<tr>
<td># Expression Extension Calls per Extension</td>
<td>0</td>
<td>12</td>
<td>1.68</td>
<td>1.0</td>
<td>2.23</td>
</tr>
<tr>
<td># Create Extension Calls per Extension</td>
<td>0</td>
<td>1</td>
<td>0.06</td>
<td>0.0</td>
<td>0.24</td>
</tr>
<tr>
<td># Java Extension Calls per Extension</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Times Called per Extension</td>
<td>0</td>
<td>3</td>
<td>1.29</td>
<td>1.0</td>
<td>0.72</td>
</tr>
<tr>
<td># Times Called per Expression Extension</td>
<td>0</td>
<td>3</td>
<td>1.0</td>
<td>1.0</td>
<td>0.74</td>
</tr>
<tr>
<td># Times Called per Create Extension</td>
<td>0</td>
<td>2</td>
<td>0.06</td>
<td>0.0</td>
<td>0.34</td>
</tr>
<tr>
<td># Times Called per Java Extension</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls to Internal Extensions per Extension</td>
<td>0</td>
<td>4</td>
<td>1.18</td>
<td>1.0</td>
<td>1.29</td>
</tr>
<tr>
<td># Times called by Internal Extensions per Extension</td>
<td>0</td>
<td>3</td>
<td>1.06</td>
<td>1.0</td>
<td>0.74</td>
</tr>
<tr>
<td># Calls to External Extensions per Extension</td>
<td>0</td>
<td>12</td>
<td>0.56</td>
<td>0.0</td>
<td>2.09</td>
</tr>
<tr>
<td># Times called by External Extensions per Extension</td>
<td>0</td>
<td>2</td>
<td>0.24</td>
<td>0.0</td>
<td>0.55</td>
</tr>
<tr>
<td># Imported Meta-models per Extension File</td>
<td>1</td>
<td>2</td>
<td>1.25</td>
<td>1.0</td>
<td>0.5</td>
</tr>
<tr>
<td># Arrounds per Extension File</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 7.7: Xtend metrics extracted from the synchronous to asynchronous communication transformation (3)

7.2.2.2 Metrics extracted using XXET

This subsection shows the metrics extracted from the same synchronous to asynchronous communication transformation but using the tool XXET, see Appendix B. XXET is a web-based application can be accessed via a web browser. The idea is to allow internet users to upload their Xtend model transformation specifications and get the metrics calculated for their transformation specifications. First, we upload a zip file containing the Xtend files of the synchronous to asynchronous communication transformation. XXET gets the zip file uploaded and all the Xtend files are extracted from the zip file. These Xtend files are listed as can be seen in Figure 7.1 Actually,
XXET has already reused the Xtend parser to parse the uploaded Xtend files to obtain their ASTs and these ASTs have already been serialized to an XML file with a predefined format. Then, the XML file is loaded by the native XML database Exist. Therefore, by the time we see the list of Xtend files, the Exist database has the data already for calculation.

Figure 7.1: Upload the zip file of the synchronous to asynchronous communication transformation - This figure shows the zip file containing Xtend files of the synchronous to asynchronous communication transformation has been uploaded. All four Xtend files have been received and listed.

Next, after uploading the Xtend files successfully, it is possible to calculate the simple metrics as well as the aggregate metrics. Figure 7.2 and 7.3 show the metrics extracted from the same synchronous to asynchronous communication transformation mentioned earlier. It can be seen that the values of the metrics extracted by both tools are the same.
7. CASE STUDIES

Figure 7.2: The simple metrics extracted - This figure shows the simple metrics extracted from the synchronous to asynchronous communication transformation.
7.2 Case study Sync2Async defined in Xtend

![Quantitative Analysis](image)

**Figure 7.3:** The aggregate metrics extracted - This figure shows the aggregate metrics extracted from the synchronous to asynchronous communication transformation.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Min</th>
<th>Max</th>
<th>Avg</th>
<th>Med</th>
<th>StdDev</th>
</tr>
</thead>
<tbody>
<tr>
<td># Extensions per Extend File</td>
<td>3</td>
<td>10</td>
<td>8.5</td>
<td>8.5</td>
<td>6.55743</td>
</tr>
<tr>
<td># Expression Extensions per Extend File</td>
<td>3</td>
<td>17</td>
<td>8.25</td>
<td>6.5</td>
<td>6.07590</td>
</tr>
<tr>
<td># Create Extensions per Extend File</td>
<td>0</td>
<td>1</td>
<td>0.25</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Java Extensions per Extend File</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Unused Extensions per Extend File</td>
<td>0</td>
<td>1</td>
<td>0.75</td>
<td>1.0</td>
<td>0.5</td>
</tr>
<tr>
<td># Unused Cached Extensions per Extend File</td>
<td>0</td>
<td>1</td>
<td>0.25</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Unused Non-Cached Extensions per Extend File</td>
<td>0</td>
<td>1</td>
<td>0.5</td>
<td>0.5</td>
<td>0.57735</td>
</tr>
<tr>
<td># Unused Expression Extensions per Extend File</td>
<td>0</td>
<td>1</td>
<td>0.75</td>
<td>1.0</td>
<td>0.5</td>
</tr>
<tr>
<td># Unused Create Extensions per Extend File</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Unused Java Extensions per Extend File</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td># Extensions have parameter per Extend File</td>
<td>3</td>
<td>10</td>
<td>8.5</td>
<td>6.5</td>
<td>6.5743</td>
</tr>
<tr>
<td># Overloaded Extensions per Extend File</td>
<td>0</td>
<td>7</td>
<td>3.25</td>
<td>3.0</td>
<td>2.98607</td>
</tr>
<tr>
<td># Parameters per Extension</td>
<td>1</td>
<td>5</td>
<td>2.147050823529417</td>
<td>2.0</td>
<td>0.78383</td>
</tr>
<tr>
<td># Unused Parameters per Extension</td>
<td>0</td>
<td>1</td>
<td>0.05823529411784705</td>
<td>0.0</td>
<td>0.23883</td>
</tr>
</tbody>
</table>

Moreover, to provide more information about the transformation, extensions can be searched by name as shown in Figure 7.4.

**Figure 7.4: Search extension(s)** - This figure shows the extensions which contain “add” in their names

### 7.2.3 Dependencies Graph

The dependency graph of the synchronous to asynchronous communication transformation is depicted in Figure 7.5. It can be seen that there are 4 Xtend files and all the extensions belonging to each Xtend file are displayed. Moreover, the caller-callee relation between the extensions can be seen clearly.
7.2 Case study Sync2Async defined in Xtend

Figure 7.5: Dependencies graph of the synchronous to asynchronous communication transformation - This figure shows the dependency graph of the synchronous to asynchronous communication transformation
7. CASE STUDIES

7.3 Case study book2pub defined in QVTo

7.3.1 Description

This is the similar book to publication transformation to the one described in Section 7.1, but here it is solved using a model transformation defined in QVTo. The QVTo model transformation specification of book to publication transformation is shown in Listing 7.2.

Listing 7.2: The book2pub transformation defined in QVTo

```plaintext
modeltype BOOKS uses 'http:///BOOKS.ecore';
modeltype PUB uses 'http:///PUB.ecore';

transformation book2pub(in bookModel:BOOKS, out pubModel:PUB);

main() {
    bookModel.objects()[Book] -> map book2pub();
}

mapping BOOKS::Book::book2pub() : PUB::Publication {
    title := self.title;
    nbPages := self.chapters->nbPages->sum();
    authors := self.chapters.author
}
```

7.3.2 Metrics extracted using MQET

It can be seen from the metrics that the size of this model transformation defined in QVTo is very small, with only 1 mapping in 1 operational transformation module. Within this mapping, 3 operations on collections are found which are 2 operations on "chapters" and 1 operation on "nbPages". The cyclomatic complexity of the mapping is 1. That means, there is only 1 "execution path" in the body of the mapping and it is true since no conditional expression is defined. The values of number of metamodels per module, number of input models per Operational Transformation and number of output models per Operational Transformation are 2, 1 and 1 which are very "normal" in a "normal" transformation which transforms a model conforms to a metamodel to another model conforms to another metamodel.
### 7.3 Case study book2pub defined in QVTo

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td># Mappings</td>
<td>1</td>
</tr>
<tr>
<td># Mappings with Condition</td>
<td>0</td>
</tr>
<tr>
<td># Overloaded Mappings</td>
<td>0</td>
</tr>
<tr>
<td># Unused Mappings</td>
<td>0</td>
</tr>
<tr>
<td># Abstract Mappings</td>
<td>0</td>
</tr>
<tr>
<td># Mapping Inheritances</td>
<td>0</td>
</tr>
<tr>
<td># Mapping Mergers</td>
<td>0</td>
</tr>
<tr>
<td># Mapping Disjunctions</td>
<td>0</td>
</tr>
<tr>
<td># Helpers</td>
<td>0</td>
</tr>
<tr>
<td># Unused Helpers</td>
<td>0</td>
</tr>
<tr>
<td># Overloaded Helpers</td>
<td>0</td>
</tr>
<tr>
<td># Calls to log()</td>
<td>0</td>
</tr>
<tr>
<td># Calls to assert()</td>
<td>0</td>
</tr>
<tr>
<td># Calls to Resolve Expressions</td>
<td>0</td>
</tr>
<tr>
<td># Modules</td>
<td>1</td>
</tr>
<tr>
<td># Library Modules</td>
<td>0</td>
</tr>
<tr>
<td># Operational Transformation Modules</td>
<td>1</td>
</tr>
<tr>
<td># Unused Modules</td>
<td>0</td>
</tr>
<tr>
<td># Unused Library Modules</td>
<td>0</td>
</tr>
<tr>
<td># Unused Operational Transformation Modules</td>
<td>0</td>
</tr>
<tr>
<td># Input Models</td>
<td>1</td>
</tr>
<tr>
<td># Output Models</td>
<td>1</td>
</tr>
<tr>
<td># Imported Metamodels</td>
<td>2</td>
</tr>
<tr>
<td># Intermediate Classes</td>
<td>0</td>
</tr>
<tr>
<td># Intermediate Properties</td>
<td>0</td>
</tr>
<tr>
<td># Variables with same name but different types</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table 7.8**: QVTo metrics extracted from the book to publication transformation (1)
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># Mappings per Module</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td># Subobjects per Mapping</td>
<td>61</td>
<td>61</td>
<td>61.0</td>
<td>61</td>
<td>0.0</td>
</tr>
<tr>
<td># Parameters per Mapping</td>
<td>2</td>
<td>2</td>
<td>2.0</td>
<td>2</td>
<td>0.0</td>
</tr>
<tr>
<td># Unused Parameters per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Parameters with direction in per Mapping</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td># Parameters with direction inout per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Parameters with direction out per Mapping</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td># Mappings per Mapping Name (Overloadings)</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td># Mappings per Disjunction</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Variables per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Unused Variables per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Operations on Collections per Mapping</td>
<td>3</td>
<td>3</td>
<td>3.0</td>
<td>3</td>
<td>0.0</td>
</tr>
<tr>
<td># Mappings per Module</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td># Helpers per Module</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Subobjects per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Helpers per Helper Name</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Parameters per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Unused Parameters per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Parameters with direction in per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Parameters with direction inout per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Parameters with direction out per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Variables per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Unused Variables per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Operations on Collections per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>Cyclomatic Complexity per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 7.9: QVTo metrics extracted from the book to publication transformation (2)
7.4 Case study QVTo2Metrics defined in QVTo

7.4.1 Description

In this case study, we present the metrics we extracted from the metrics extractor (QVTo2Metrics) described in Section 6.4.2.1 using that same tool. Part of the QVTo specification of this transformation can be found in Appendix D.

Table 7.10: QVTo metrics extracted from the book to publication transformation (3)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># Imported Modules per Module</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Times a Module is Imported per Module</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Mappings in other Modules per Module</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Helpers in other Modules per Module</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls to Mappings in Other Modules per Module</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls to Helpers in Other Modules per Module</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls to Other Modules per Module</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Mappings to Mappings per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Mappings to Helpers per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Mappings to Mappings/Helpers per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls to Mappings from Mappings per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls to Mappings from Helpers per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Mappings to Mappings per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls to Helpers to Helpers per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Helpers to Mappings per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Helpers to Mappings/Helpers per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls to Mappings from Mappings per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Helpers from Mappings per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls to Helpers from Helpers per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls to Helpers from Mappings/Helpers per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls to Helpers per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls to log() per Module</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Mappings to log() per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Helpers to log() per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls to assert() per Module</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Mappings to assert() per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Helpers to assert() per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls to Resolve Expressions per Module</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Mappings to Resolve Expressions per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Helpers to Resolve Expressions per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Input Models per Operational Transformation</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td># Output Models per Operational Transformation</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td># Metamodels per Module</td>
<td>2</td>
<td>2</td>
<td>2.0</td>
<td>2</td>
<td>0.0</td>
</tr>
<tr>
<td># Subobjects per Module</td>
<td>102</td>
<td>102</td>
<td>102.0</td>
<td>102</td>
<td>0.0</td>
</tr>
<tr>
<td># Intermediate Classes per Operational Transformation</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Intermediate Properties per Operational Transformation</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
</tbody>
</table>
7. CASE STUDIES

7.4.2 Metrics extracted using MQET

The extracted metrics are presented in Tables 7.11 and 7.12. The metric values give some insight into the transformation. There are only 1 mappings but 100 helpers. This is easy to understand since the mapping uses other helpers for calculating the metrics. 7 unused helper here consists of 4 helpers in the Utilities.qvto module which are used to handle Real numbers but they are irrelevant. Moreover, the other 3 helpers are only used for testing purpose but they are already obsolete. All of them should be removed. The number of calls to log() is 9, which means the transformation is still under development. Actually the transformation is still under development because the calculations of some more metrics are planned to be implemented.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td># Mappings</td>
<td>1</td>
</tr>
<tr>
<td># Mappings with Condition</td>
<td>0</td>
</tr>
<tr>
<td># Overloaded Mappings</td>
<td>0</td>
</tr>
<tr>
<td># Unused Mappings</td>
<td>0</td>
</tr>
<tr>
<td># Abstract Mappings</td>
<td>0</td>
</tr>
<tr>
<td># Mapping Inheritances</td>
<td>0</td>
</tr>
<tr>
<td># Mapping Mergers</td>
<td>0</td>
</tr>
<tr>
<td># Mapping Disjunctions</td>
<td>0</td>
</tr>
<tr>
<td># Helpers</td>
<td>102</td>
</tr>
<tr>
<td># Unused Helpers</td>
<td>6</td>
</tr>
<tr>
<td># Overloaded Helpers</td>
<td>0</td>
</tr>
<tr>
<td># Calls to log()</td>
<td>14</td>
</tr>
<tr>
<td># Calls to assert()</td>
<td>0</td>
</tr>
<tr>
<td># Calls to Resolve Expressions</td>
<td>0</td>
</tr>
<tr>
<td># Modules</td>
<td>6</td>
</tr>
<tr>
<td># Library Modules</td>
<td>5</td>
</tr>
<tr>
<td># Operational Transformation Modules</td>
<td>1</td>
</tr>
<tr>
<td># Unused Modules</td>
<td>0</td>
</tr>
<tr>
<td># Unused Library Modules</td>
<td>0</td>
</tr>
<tr>
<td># Unused Operational Transformation Modules</td>
<td>0</td>
</tr>
<tr>
<td># Input Models</td>
<td>1</td>
</tr>
<tr>
<td># Output Models</td>
<td>1</td>
</tr>
<tr>
<td># Imported Metamodels</td>
<td>4</td>
</tr>
<tr>
<td># Intermediate Classes</td>
<td>0</td>
</tr>
<tr>
<td># Intermediate Properties</td>
<td>0</td>
</tr>
<tr>
<td># Variables with same name but different types</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 7.11: QVTo metrics extracted from the QVTo to metrics transformation (1)
### 7.4 Case study QVTo2Metrics defined in QVTo

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># Mappings per Module</td>
<td>0</td>
<td>1</td>
<td>0.17</td>
<td>0</td>
<td>0.41</td>
</tr>
<tr>
<td># Subobjects per Mapping</td>
<td>1241</td>
<td>1241</td>
<td>1241.0</td>
<td>1241</td>
<td>0.0</td>
</tr>
<tr>
<td># Parameters per Mapping</td>
<td>2</td>
<td>2</td>
<td>2.0</td>
<td>2</td>
<td>0.0</td>
</tr>
<tr>
<td># Unused Parameters per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Parameters with direction in per Mapping</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td># Parameters with direction inout per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Parameters with direction out per Mapping</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td># Mappings per Mapping Name (Overloadings)</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td># Mappings per Disjunction</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Variables per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Unused Variables per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Operations on Collections per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>Cyclomatic Complexity per Mapping</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td># Helpers per Module</td>
<td>1</td>
<td>32</td>
<td>17.0</td>
<td>16</td>
<td>10.43</td>
</tr>
<tr>
<td># Subobjects per Helper</td>
<td>28</td>
<td>264</td>
<td>78.44</td>
<td>58</td>
<td>50.91</td>
</tr>
<tr>
<td># Helpers per Helper Name</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td># Parameters per Helper</td>
<td>2</td>
<td>3</td>
<td>2.02</td>
<td>2</td>
<td>0.14</td>
</tr>
<tr>
<td># Unused Parameters per Helper</td>
<td>0</td>
<td>1</td>
<td>0.46</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td># Parameters with direction in per Helper</td>
<td>1</td>
<td>2</td>
<td>1.03</td>
<td>1</td>
<td>0.17</td>
</tr>
<tr>
<td># Parameters with direction inout per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Parameters with direction out per Helper</td>
<td>0</td>
<td>1</td>
<td>0.99</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td># Variables per Helper</td>
<td>0</td>
<td>3</td>
<td>1.81</td>
<td>2</td>
<td>0.82</td>
</tr>
<tr>
<td># Unused Variables per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Operations on Collections per Helper</td>
<td>0</td>
<td>13</td>
<td>3.02</td>
<td>2</td>
<td>2.75</td>
</tr>
<tr>
<td>Cyclomatic Complexity per Helper</td>
<td>1</td>
<td>5</td>
<td>1.42</td>
<td>1</td>
<td>0.96</td>
</tr>
</tbody>
</table>

**Table 7.12:** QVTo metrics extracted from the QVTo to metrics transformation (2)
7. CASE STUDIES

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># Imported Modules per Module</td>
<td>0</td>
<td>4</td>
<td>1.33</td>
<td>1</td>
<td>1.37</td>
</tr>
<tr>
<td># Times a Module is Imported per Module</td>
<td>0</td>
<td>4</td>
<td>1.33</td>
<td>1</td>
<td>1.37</td>
</tr>
<tr>
<td># Calls from Mappings in other Modules per Module</td>
<td>0</td>
<td>32</td>
<td>14.33</td>
<td>15</td>
<td>12.5</td>
</tr>
<tr>
<td># Calls from Helpers in other Modules per Module</td>
<td>0</td>
<td>90</td>
<td>15.0</td>
<td>0</td>
<td>36.74</td>
</tr>
<tr>
<td># Calls from Other Modules per Module</td>
<td>0</td>
<td>90</td>
<td>29.33</td>
<td>18</td>
<td>31.47</td>
</tr>
<tr>
<td># Calls to Mappings in Other Modules per Module</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls to Helpers in Other Modules per Module</td>
<td>0</td>
<td>86</td>
<td>29.33</td>
<td>18</td>
<td>29.66</td>
</tr>
<tr>
<td># Calls to Other Modules per Module</td>
<td>0</td>
<td>86</td>
<td>29.33</td>
<td>18</td>
<td>29.66</td>
</tr>
<tr>
<td># Calls from Mappings to Mappings per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Mappings to Helpers per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Helpers to Mappings per Mapping</td>
<td>0</td>
<td>5</td>
<td>0.96</td>
<td>1</td>
<td>0.51</td>
</tr>
<tr>
<td># Calls from Helpers to Helpers per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Helpers to Mappings/Helpers per Mapping</td>
<td>87</td>
<td>87</td>
<td>87.0</td>
<td>87</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Mappings to Mappings/Helpers per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Mappings to Mappings per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Mappings to Helpers per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Helpers to Mappings per Helper</td>
<td>0</td>
<td>1</td>
<td>0.85</td>
<td>1</td>
<td>0.36</td>
</tr>
<tr>
<td># Calls from Helpers to Helpers per Helper</td>
<td>0</td>
<td>61</td>
<td>0.96</td>
<td>0</td>
<td>6.62</td>
</tr>
<tr>
<td># Calls to Helpers per Helper</td>
<td>0</td>
<td>61</td>
<td>1.81</td>
<td>1</td>
<td>6.5</td>
</tr>
<tr>
<td># Calls to log() per Module</td>
<td>0</td>
<td>5</td>
<td>2.33</td>
<td>1</td>
<td>2.34</td>
</tr>
<tr>
<td># Calls to log() per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls to log() per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls to assert() per Module</td>
<td>0</td>
<td>3</td>
<td>0.14</td>
<td>0</td>
<td>0.51</td>
</tr>
<tr>
<td># Calls to assert() per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls to Resolve Expressions per Module</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls to Resolve Expressions per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Mappings to Resolve Expressions per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Mappings to Resolve Expressions per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Input Models per Operational Transformation</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td># Output Models per Operational Transformation</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td># Metamodels per Module</td>
<td>1</td>
<td>4</td>
<td>2.83</td>
<td>3</td>
<td>1.17</td>
</tr>
<tr>
<td># Subobjects per Module</td>
<td>687</td>
<td>2700</td>
<td>1592.67</td>
<td>1386</td>
<td>658.97</td>
</tr>
<tr>
<td># Intermediate Classes per Operational Transformation</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Intermediate Properties per Operational Transformation</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Table 7.13:** QVTo metrics extracted from the QVTo to metrics transformation (3)

7.4.3 Coverage Graph

The model transformation QVTo to metrics is using at least three metamodels because the QVTo model conforms to the QVTOperational.ecore as well as ImperativeOCL.ecore and the Metrics model conforms to the Metrics.ecore. Since the current implementation of model transformation from QVTo model to Coverage Relation model only works for the QVTo model transformations with only one input and one output.
metamodel, it is not yet feasible to analysis the coverage graph for the model transformation QVTo to metrics. Therefore, this section will analyze the coverage graph of another QVTo model transformation, see Listing 7.3.

Listing 7.3: The MMATransformation transformation defined in QVTo

```java
import m2m.qvto.MyJavaLib;

modeltype MMA uses "http://mma/1.0";
modeltype MMB uses "http://mmb/1.0";

transformation MMATransformation(in Source: MMA, out Target: MMB);

main() {
    Source.rootObjects()[Project] -> map ProjectToModel();
}

mapping Project::ProjectToModel() : Model {
    Name := self.Title;
    automata := self.bodies -> map toAutomaton();
}

mapping MMA::Automaton::toAutomaton() : MMB::Automaton
when { self.Name.startsWith("A"); }
{
    Name := self.Name;
    modes := self.locations -> map toMode();
    transitions := self.locations.outgoingEdges -> map toTransition();
    getSqrt(9);
    log("Name of the Automaton: " + Name);
}

mapping Location::toMode() : Mode
disjuncts Rectangle::fromRectangle, Ellipse::fromEllipse,
Triangle::fromTriangle1, Triangle::fromTriangle2, Triangle::fromTriangle0 {
    log("This log is never reached, disjunct mappings don’t use their body.")
}

mapping Rectangle::fromRectangle() : Mode
inherits Location::Updates {
    Shape := "Rectangle";
    Dimension := self.Dimension;
    log("Re");
}
```
mapping Ellipse::fromEllipse() : Mode
inherits Location::Updates {
    Shape := "Ellipse";
    Dimension := self.Radius;
    log("Ef");
}

mapping Triangle::fromTriangle0() : Mode
disjuncts Triangle::fromTriangle1, Triangle:: fromTriangle2{
    log("This log is never reached, disjunct mappings don’t use their body.")
}

mapping Triangle::fromTriangle1() : Mode
inherits Location::Updates
when { 1 = 2 }
{
    /* Since the condition doesn’t hold (1 is obviously not equal to 2), 
    this mapping won’t be executed for a Triangle object. The one below
    will, because there are no conditions, and the input object is a
    Triangle.
    */
    log("getSqrt(9) = ",getSqrt(9));
    log("Tr1");
}

mapping Triangle::fromTriangle2() : Mode
inherits Location::Updates, Location::Updates2 {
    Shape := "Triangle";
    Dimension := self.Side;
    log("Tr2");
}

intermediate class IntermediateMod {
    Urgent: Boolean;
    vars: OrderedSet(Modification);
}

intermediate property Mode::intermediateMods : IntermediateMod;

abstract mapping Location::Updates() : Mode {
    Name := self.Name;
    InitialState := self.IsInitial;
    Modifications := ModificationsQuery(self.updates);
    -- intermediateMods := self.updates.map toIntermediateMod();
7.4 Case study QVTo2Metrics defined in QVTo

```plaintext
mapping Location::Updates2() : Mode {
    Name := self.Name;
    InitialState := self.IsInitial;
    Modifications := ModificationsQuery(self.updates);
    -- intermediateMods := self.updates.map toIntermediateMod();
}

mapping Update::toIntermediateMod() : IntermediateMod {
    Urgent := self.Urgent;
    vars := self.variables -> VarToMod();
}

mapping Variable::VarToMod() : Modification {
    VarName := self.Name;
    VarType := self.Type;
}

-- Use "Modifications := ModificationsQuery(self.updates);" to call this Query

query ModificationsQuery( Updates: Update ) : OrderedSet(Modification) {
    var Modifications: OrderedSet(Modification);
    Updates.variables -> forEach(Var) {
        Modifications += object Modification { VarName := Var.Name; VarType := Var.Type; };
    }; return Modifications;
}

mapping Edge::toTransition() : Transition {
    log("getSqrt(9) = ", getSqrt(9));
    Event := self.Action;
    sourceMode := self.container().oclAsType(Location).resolveone(Mode);
    targetMode := self.targetLocation.resolve(Mode) -> at(1);
}
```

The coverage graph of the QVTo model transformation above is visualized by TreeComaparer and can be seen in Figure 7.6. This figure shows which classes are used by which transformation functions.

In order to see how fast the tool MQET can extract metrics from a large QVTo model transformation, the QVTo transformation from ucif to cif are used. The fifth case study ucif2cif is presented in Appendix D.2.
Figure 7.6: The coverage graph is visualized by TreeComaparer - This figure shows the use-used by relations between classes of the metamodels and transformation functions of the transformation
Chapter 8

Conclusions & future work

8.1 Conclusions

We have addressed the necessity for enabling measurement of model transformations created using different formalisms, in order to analyze the quality of model transformations. In this work, we defined one set of metrics for measuring model transformations created using Xtend and another one for measuring model transformations created using QVTo. The former set of metrics can automatically be collected from Xtend model transformation specifications by the tools JXET and XXET we developed. The latter can automatically be collected from QVTo model transformation specifications by our tool MQET.

The metrics defined in this project are specific for Xtend or QVTo but many of them are conceptually equivalent to each other as well as to the metrics for ATL (13, 14) or the metrics for ASF+SDF (4). However, it can be observed from the sets of metrics that the QVTo metrics are closer to the ATL metrics. This observation makes sense since it is concluded that QVTo has the highest potential to be interoperable with ATL (54). The conceptual equivalence of many metrics defined for different transformation languages implies that some metrics to assess the quality attributes just need to be adapted to the specifics of the transformation formalism. Even though, several metrics are entirely specific for a specific transformation formalism and are not conceptually equivalent to metrics for other formalisms. For example, the metrics number of cached extensions, number of arounds are very specific for Xtend and the metrics number of intermediate properties, number of intermediate classes are very specific for QVTo.
8. CONCLUSIONS & FUTURE WORK

Not only by means of metrics, the (Xtend and QVTo) model transformations can also be analyzed by the visualization of coverage and dependency. The tools we have developed are used to extract metrics as well as coverage and dependency data. This data is used as inputs for the visualization tools such as ExtraVis and TreeComparer to visualize the coverage graph and dependency graph.

Developing tools for measuring model transformations is one of the main goals of this project. During the project, more than one way to develop such a tool have been discovered. But the whole idea is to reuse the parser/compiler of the transformation formalism for parsing the transformation specification in order to obtain the corresponding AST. After mastering the structure of the AST, we can make the extraction of metrics, coverage and dependency programmable. The implementation can certainly be done using the framework of JXET. And if serializing the AST to XML file is convenient, we can use the similar way of implementing XXET. Moreover, in case the transformation formalism has the support for serializing the AST to a model of transformation specification, it is even more convenient by following the way of implementing MQET. Note that the toolkit of one transformation formalism may provide better support than the toolkit of another one. For example, it is quite easy to obtain the model of transformation specification in ATL by running a built-in script but in case of QVTo, we had to find a way to manually reuse the QVTo compiler to obtain the AST and then manually serialize the AST to obtain the model of QVTo transformation specification before we can extract metrics, coverage and dependency from this model.

8.2 Future work

As the scope of this Master project was initially restricted to enabling the measurement of Xtend model transformations by means of metrics, it is very challenging to also do the same task for QVTo model transformations and by means of coverage and dependency as well. So far, we have implemented the extraction of metrics for both Xtend and QVTo model transformations. For Xtend model transformations, the dependency graph can also be visualized but not yet for the coverage graph. For QVTo model transformations, the coverage graph can be visualized but not yet for the dependency graph. Due to time constraint, some missing implementations are deferred to future work.
Metrics need to be related to quality attributes to understand which metrics are relevant for assessing the different attributes of the quality model of transformations. In this work, we just relate metrics to quality attributes based on our expectation but it is not enough. Therefore, another point for future work is to conduct an empirical study in the same way as described in (1). On one hand, metrics should be extracted from a heterogeneous collection of (Xtend or QVTo) model transformations using one of the tools described in Chapter 6. On the other hand, experts of (Xtend or QVTo) transformation language should assess the quality of the same collection of model transformations. Thereafter the correlations between the metrics data acquired from the tool and the quality evaluation acquired from the experts should be analyzed. In this way, it is possible to derive which metrics are relevant for assessing the various attributes of model transformation quality.

By enabling measurement of model transformations defined in different transformation languages, addressing quality problems in model transformations is getting closer to its goal. After that, there should be a methodology proposed for improving quality of model transformations. This methodology will probably consist of a set of guidelines which, if adhered to, lead to high-quality model transformations (1).
References


REFERENCES


REFERENCES


[22] The Eclipse Foundation. **Model To Model (M2M).** http://www.eclipse.org/m2m/.


REFERENCES


[34] S. Nolte. QVT - Operational Mappings, Modeling with the Query Views Transformation. Springer-Verlag, 2009. 20


REFERENCES


[44] M. F. van Amstel, C. F. J. Lange, and M. G. J. van den Brand. Metrics for Analyzing the Quality of Model Transformations. 12th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2008), pages 41-51, Paphos, Cyprus, July 2008. 31


REFERENCES

on Principles of Software Evolution (IWPSE-EVOL’10), pages 48-57, Antwerp, Belgium, September 2010. 84

[53] **The CIF 2 tooling project. CIF: The Compositional Interchange Format for Hybrid Systems.** http://www.se.wtb.tue.nl/sewiki/cif/start, last access October 2010. 103, 152


[56] W3C. **XML Path Language (XPath), Version 1.0.** http://www.w3.org/TR/xpath/ (last access October 2010). 119

[57] W3C. **XQuery.** http://www.w3.org/standards/xml/query (last access October 2010). 119


Appendix A

Implementation details of JXET

A.1 Data model

This section describes the class diagram of expression types which are used in the body of an extension. To extract data from an Xtend AST, it is necessary to understand expression types which are used in the body of an extension in advanced. In this way, all elements of an extension can be extracted.

Figure A.1: Class diagram of the expressions types used in an extension - This figure shows expression types used in an extension
Appendix B

XML-based Xtend Extractor Tool

In the Section 6.3 a tool for measuring Xtend model transformations, namely JXET, has been described. However, it can be seen that the implementation for calculating metrics, i.e., dependency metrics is quite complicated. So even though this way is doable but it may be not easy to maintain the tool in the future. This section presents another tool for measuring Xtend model transformations, namely XML-based Xtend Extractor Tool (XXET). XXET is developed in a different way which could make the implementation for calculating metrics, especially dependency metrics easier. The idea for developing XXET is based on the fact that elements from the Xtend AST can easily be extracted and serialized to an XML file with a predefined format. Thus, we can use XPath and XQuery to query the data in the XML file and calculate the metrics quite efficiently.

B.1 Requirements

Except general requirements mentioned in Section 6.1, some more specific requirements are required for XXET. Actually XXET should be developed as a web application which allows user to upload extension files of an Xtend model transformation and then shows the set of metrics extracted from this Xtend model transformation. Some main features of XXET are described as follows.
B. XML-BASED XTEND EXTRACTOR TOOL

B.1.1 Upload extension files

In order to extract metrics from an Xtend model transformation, we have to parse the extension files in which the Xtend model transformation is defined. This functionality allows user to upload a zip file which contains extension files of an Xtend model transformation. The extension files are then unzipped and parsed by the Xtend parser which is reused.

B.1.2 Calculate simple metrics

This functionality allows user to choose a specific simple-metric to calculate. User can choose to calculate a simple metric one by one, or to calculate all simple metrics at once.

B.1.3 Calculate aggregate metrics

This functionality allows user to choose an aggregate metric to calculate. Aggregate metrics are metrics which contain average, minimum, maximum, median and standard deviation values. User can choose to calculate an aggregate metric one by one, or to calculate all aggregate metrics at once.

B.1.4 Search extension statements by name

As extension statements are the main elements of Xtend model transformations and there are many extension statements per extension files, it would be nice to search for information of extension statements. This functionality allows user to search for extension statements by name. User can choose to search for all extension statements or for a specific kind of extension statements (create extension statements, expression extension statements or java extension statements). Moreover, user can choose to search for extension statements whose names are fully matched with the name provided or just contain the keyword provided.

B.2 Design

Figure B.1 shows the architecture of XXET. Via a web browser, a zip file which contains extension files of an Xtend model transformation can be uploaded to the system. The
extension files uploaded are parsed by the Xtend parser to obtain the Xtend ASTs. The
data extractor here is almost similar to the data extractor of JXET (Figure 6.4). The
only difference is the output of the data extractor of XXET is an XML file (instead of
in-memory Java objects in JXET) which contains elements extracted from the Xtend
AST.

Figure B.1: Architecture of XXET - This figure shows the architecture of XXET

The XML file is imported by the native XML database eXist (55). eXist enables
us to use XPath (56) and XQuery (57) to query the data of elements in the XML file
easily. However, eXist only plays a role as a back-end in the three-tier architecture of
the web application developed by using JavaBeans, JSP/Servlet and Java Server Faces (58). This system runs on the HTTP server Apache Tomcat 7.0 (59).

All the requests from a browser to the system are processed by the server. Then
the server makes HTTP requests to execute XQueries in eXist to get the results in
XML format. The results will be parsed by the corresponding Java objects to extract
the data to Java bean objects. Java Server Faces is used to display the results of Java
beans on the web browser.
B. XML-BASED XTEND EXTRACTOR TOOL

Figure B.2 shows part of the DTD file which contains format of the XML file. From the DTD file, we can see that the root element of the XML file is `xtendPrj` with some attributes, i.e., `prjName` and `prjDir` which contain name of the Xtend M2M project and name of the folder containing the extension files. The root element may contain many elements `extendFile`. Each element `extendFile` have an attribute `extFileName` and may contain many elements, such as `importedNamespace`, `importedExtFiles`, `extensions`. Element `extensions` may contain many many elements `extension`, and so on. The full DTD file can be seen in the Appendix.

Not all details of XXET are given in this section, for example architecture design of the server. Appendices can be referred to for more information.

B.3 Implementation

This section shows the implementation of some main features of XXET: upload extension files, calculate simple metrics, calculate aggregate metrics and search extension statements by name. Some screen shots are shown for a better demonstration.

B.3.1 Upload extension files

Figure B.3 shows the page for uploading extension files. Actually, a zip file which contains extension files can be uploaded from this page. The zip file is processed by a Java bean which extracts the zip file to get the extension files and then reuses the Xtend parser to obtain the Xtend ASTs. These in-memory Xtend ASTs are extracted and serialized to the XML file with the pre-defined format described before. Figure B.4 shows the names of extension files are extracted and parsed. After this step, the calculation of metrics is enabled.

B.3.2 Calculate simple metrics

This section shows how simple-metrics are calculated. The screen shot is shown in Figure B.5. Follows are some examples of XQuery and XPath used to calculate the simple metrics.

- Number of Extension Files: This metric counts the number of extension files within the Xtend project.
B.3 Implementation

Figure B.2: Format of the XML file - This figure shows part of the DTD file which contains format of the XML file.
Figure B.3: **Upload extension files** - This figure shows the page for uploading extension files.
Figure B.4: Extension files uploaded - This figure shows extension files uploaded
Figure B.5: Calculate simple metrics - This figure shows the list of simple metrics calculated
B.3 Implementation

```
count(for $f in doc("xtendM2M/Bi2Uni.xml")//extendFile return $f)
```

- Number of Extension Statements without Parameter: This metric counts the number of extension statements which have no parameter.

```
count(
  for $ext in doc("xtendM2M/Bi2Uni.xml")//extensions/*
  where (count($ext/parameters/*)=0)
  return $ext
)
```

B.3.3 Calculate aggregate metrics

This section shows how aggregate-metrics are calculated. The screen shot is shown in Figure B.6. Follows are some examples of XQuery and XPath used to calculate the aggregate metrics.

![QuantitativeAnalysis](image)

**Figure B.6: Calculate aggregate metrics** - This figure shows the list of aggregate metrics calculated
B. XML-BASED XTEND EXTRACTOR TOOL

- Number of Extension Statements per Extension File: This metric counts the number of extension statements per extension file.

```xml
<result>
  for $extFile in doc("xtendM2M/Bi2Uni.xml")/extendFile
  return element { "extCountPerExtFile" } {
    count(for $ext in $extFile//extension
          return $ext)
  }
</result>
```

- Number of Expression Extension Statements per Extension File: This metric counts the number of expression extension statements per extension file.

```xml
<results>
  for $extFile in doc("xtendM2M/Bi2Uni.xml")/extendFile
  return element { "result" } {
    count(for $ext in $extFile//extension
          where $ext/@extType="expression"
          return $ext)
  }
</results>
```

B.3.4 Search extension statements by name

This section shows how extension statements can be searched by name. The screen shot is shown in Figure 1. Some examples of Xquery and Xpath used to search extension statements by name are follows. It is also possible to order the results by name of extension statement.

- Search extension statements by fully matched name: Providing a name to search, and choose the option "fully matched".

```xml
declare function local : orderSelection($ext,$order) {
  if ($order = "name")
    then $ext/extName
```
Figure B.7: Search extension statements by name - This figure shows the page for searching extension statements by name
B. XML-BASED XTEND EXTRACTOR TOOL

```
else
    $ext/returnType
};

<element { "result " }
  { for $ext in doc("xtendM2M/Bi2Uni.xml")//extension
    let $name := request:get-parameter("name", ())
    let $order := "name"
    where $ext/extName/text() = $name
    order by local:orderSelection($ext, $order)
    return
      <extension>
        { $ext/@extType }
        { $ext/@isPrivate }
        { $ext/@isCached }
        { <extName>
          { $ext/extName/text() }
        }
        { <numberOfParams>
          { count( for $p in $ext//parameters/parameterName return $p ) }
        }
      </extension>
  }

• Search expression extension statements by partially matched name: Providing
  the name to search, and the option "fully matched" is not checked.
```

```
<element { "result " }
  { for $ext in doc("xtendM2M/Bi2Uni.xml")//extension
    let $name := request:get-parameter("name", ())
    where $ext/@extType="expression" and contains( $ext/extName/text() , $name )
    return
      <extension>
        { $ext/@extType }
        { $ext/@isPrivate }
        { $ext/@isCached }
        { <extName>
          { $ext/extName/text() }
        }
        { <numberOfParams>
          { count( for $p in $ext//parameters/parameterName return $p ) }
        }
      </extension>
  }
```
Appendix C

Implementation details of MQET

C.1 Architectures

This section shows details of the architectures of the metrics extractor and coverage extractor, see Figures C.1 and C.2.

![Figure C.1: The model transformations QVTo to metrics - This figure shows the architecture of the model transformation QVTo to metrics](image)

C.2 Source code of the model transformation QVTo to metrics

This section shows some code of the model transformation QVTo to metrics. The main module is QVTo2Metrics.qvto, see Listing C.1. The library module of helper functions for calculating the metrics are shown at the Listing C.2. The other 4 modules can be found in the Deliverables, see Appendix E.
C. IMPLEMENTATION DETAILS OF MQET

Figure C.2: The model transformations QVTo to coverage - This figure shows the architecture of the model transformations QVTo to coverage

Listing C.1: QVTo2Metrics.qvto

```plaintext
import MappingMetrics;
import HelperMetrics;
import MiscellaneousMetrics;
import DependencyMetrics;

modeltype QVTO uses "http://www.eclipse.org/QVT/1.0.0/Operational";
modeltype METRICS uses "http://Metrics.ecore";

transformation QVTo2Metrics(in qvtoModel:QVTO, out metricModel:METRICS);

main() {
    qvtoModel.objects(!QVTO::expressions::OperationalTransformation) => map qvto2metrics();
}

mapping QVTO::expressions::OperationalTransformation::qvto2metrics() : METRICS::Metrics {
    TrafoName := self.name;

    allModules := OrderedSet { self };  
    getAllModules(self, result);

    //Miscellaneous Metrics
    SimpleMetrics += NumModules(allModules);
    SimpleMetrics += NumLibraryModules(allModules);
    SimpleMetrics += NumOperationalTransformationModules(allModules);

    SimpleMetrics += NumUnusedModules(allModules);
    SimpleMetrics += NumUnusedLibraryModules(allModules);
    SimpleMetrics += NumUnusedOperationalTransformationModules(allModules);
}
```
C.2 Source code of the model transformation QVTo to metrics

```java
SimpleMetrics += NumInputModels(allModules);
SimpleMetrics += NumOutputModels(allModules);
AggregatedIntegerMetrics += NumInputModelsPerTransformationModule(allModules);
AggregatedIntegerMetrics += NumOutputModelsPerTransformationModule(allModules);

SimpleMetrics += NumMetaModels(allModules);
AggregatedIntegerMetrics += NumMetaModelsPerModule(allModules);

AggregatedIntegerMetrics += NumSubobjectsPerModule(allModules);

SimpleMetrics += NumIntermediateClasses(allModules);
SimpleMetrics += NumIntermediateProperties(allModules);
AggregatedIntegerMetrics += NumIntermediateClassesPerTransformationModule(allModules);
AggregatedIntegerMetrics += NumIntermediatePropertiesPerTransformationModule(allModules);

SimpleMetrics += NumVariablesWithSameNameNotType(allModules);

// Mapping Metrics
SimpleMetrics += NumMappings(allModules);
SimpleMetrics += NumMappingsWithCondition(allModules);

AggregatedIntegerMetrics += NumMappingsPerModule(allModules);
AggregatedIntegerMetrics += NumSubobjectsPerMapping(allModules);
AggregatedIntegerMetrics += NumParamsPerMapping(allModules);
AggregatedIntegerMetrics += NumUnusedParamsPerMapping(allModules);
AggregatedIntegerMetrics += NumInParamsPerMapping(allModules);
AggregatedIntegerMetrics += NumInOutParamsPerMapping(allModules);
AggregatedIntegerMetrics += NumOutParamsPerMapping(allModules);

SimpleMetrics += NumOverloadedMappings(allModules);
AggregatedIntegerMetrics += NumMappingsPerMappingName(allModules);

SimpleMetrics += NumUnusedMappings(allModules);
SimpleMetrics += NumAbstractMappings(allModules);

AggregatedIntegerMetrics += NumMappingsPerDisjunction(allModules);
SimpleMetrics += NumMappingInheritances(allModules);
SimpleMetrics += NumMappingMergers(allModules);
```

131
C. IMPLEMENTATION DETAILS OF MQET

SimpleMetrics += NumMappingDisjunctions(allModules);
AggregatedIntegerMetrics += NumVariablesPerMapping(allModules);
AggregatedIntegerMetrics += NumUnusedVariablesPerMapping(allModules);
AggregatedIntegerMetrics += NumOpsOnCollectionsPerMapping(allModules);
AggregatedIntegerMetrics += CyclomaticComplexityPerMapping(allModules);

---

//Helper & Query Metrics
SimpleMetrics += NumHelpers(allModules);
//SimpleMetrics += NumQueries(allModules);

AggregatedIntegerMetrics += NumHelpersPerModule(allModules);
//AggregatedIntegerMetrics += NumQueriesPerModule(allModules);
AggregatedIntegerMetrics += NumSubobjectsPerHelper(allModules);
SimpleMetrics += NumUnusedHelpers(allModules);
SimpleMetrics += NumOverloadedHelpers(allModules);
AggregatedIntegerMetrics += NumHelpersPerHelperName(allModules);

AggregatedIntegerMetrics += NumParamsPerHelper(allModules);
AggregatedIntegerMetrics += NumUnusedParamsPerHelper(allModules);
AggregatedIntegerMetrics += NumInParamsPerHelper(allModules);
AggregatedIntegerMetrics += NumInOutParamsPerHelper(allModules);
AggregatedIntegerMetrics += NumOutParamsPerHelper(allModules);
AggregatedIntegerMetrics += NumVariablesPerHelper(allModules);
AggregatedIntegerMetrics += NumUnusedVariablesPerHelper(allModules);
AggregatedIntegerMetrics += NumOpsOnCollectionsPerHelper(allModules);
AggregatedIntegerMetrics += CyclomaticComplexityPerHelper(allModules);

---

//Dependency Metrics
AggregatedIntegerMetrics += NumImportedModulePerModule(allModules);
AggregatedIntegerMetrics += NumTimesModuleImported(allModules);
AggregatedIntegerMetrics += NumCallsFromMappingsInOtherModulesPerModule(allModules);
AggregatedIntegerMetrics += NumCallsFromHelpersInOtherModulesPerModule(allModules);
AggregatedIntegerMetrics += NumCallsFromOtherModulesPerModule(allModules);//Module fan-in
AggregatedIntegerMetrics += NumCallsToMappingsInOtherModulesPerModule(allModules);
AggregatedIntegerMetrics += NumCallsToHelpersInOtherModulesPerModule(allModules);
AggregatedIntegerMetrics += NumCallsToOtherModulesPerModule(allModules);//Module fan–out

AggregatedIntegerMetrics += NumCallsFromMappingsToMappingsPerMapping(allModules);
AggregatedIntegerMetrics += NumCallsFromMappingsToHelpersPerMapping(allModules);
AggregatedIntegerMetrics += NumCallsFromMappingsToMappingsOrHelpersPerMapping(allModules);//Mapping fan–out

AggregatedIntegerMetrics += NumCallsToMappingsFromMappingsPerMapping(allModules);
AggregatedIntegerMetrics += NumCallsToMappingsFromHelpersPerMapping(allModules);
AggregatedIntegerMetrics += NumCallsToMappingsPerMapping(allModules);//Mapping fan–in

AggregatedIntegerMetrics += NumCallsFromHelpersToHelpersPerHelper(allModules);
AggregatedIntegerMetrics += NumCallsFromHelpersToMappingsPerHelper(allModules);
AggregatedIntegerMetrics += NumCallsFromHelpersToMappingsOrHelpersPerHelper(allModules);//Helper fan–out

AggregatedIntegerMetrics += NumCallsToHelpersFromMappingsPerHelper(allModules);
AggregatedIntegerMetrics += NumCallsToHelpersFromHelpersPerHelper(allModules);
AggregatedIntegerMetrics += NumCallsToHelpersPerHelper(allModules);//Helper fan–in

SimpleMetrics += NumCallsToLog(allModules);
AggregatedIntegerMetrics += NumCallsToLogPerModule(allModules);
AggregatedIntegerMetrics += NumCallsToLogPerMapping(allModules);
AggregatedIntegerMetrics += NumCallsToLogPerHelper(allModules);

SimpleMetrics += NumCallsToAssert(allModules);
AggregatedIntegerMetrics += NumCallsToAssertPerModule(allModules);
AggregatedIntegerMetrics += NumCallsToAssertPerMapping(allModules);
AggregatedIntegerMetrics += NumCallsToAssertPerHelper(allModules);

SimpleMetrics += NumCallsToResolve(allModules);
AggregatedIntegerMetrics += NumCallsToResolvePerModule(allModules);
AggregatedIntegerMetrics += NumCallsToResolvePerMapping(allModules);
AggregatedIntegerMetrics += NumCallsToResolvePerHelper(allModules);

}
C. IMPLEMENTATION DETAILS OF MQET

Listing C.2: Utilities.qvt

library Utilities;

import m2m.qvt.MyJavaLib;

modeltype METRICS uses "http:///Metrics.ecore";

helper getNewSimpleMetric(name : String, value : Integer): SimpleMetric{
    var m : SimpleMetric := object SimpleMetric{};
    m.Metric := name;
    m.Value := value;
    return m;
}

helper getNewAggregatedIntegerMetric(name : String, intseq : Sequence(Integer)) : AggregatedIntegerMetric{
    var m : AggregatedIntegerMetric = object AggregatedIntegerMetric{};
    m.Metric := name;

    if (intseq->size() > 0) then {
        m.Minimum := getIntSeqMinimum(intseq);
        m.Maximum := getIntSeqMaximum(intseq);
        m.Average := getIntSeqAverage(intseq);
        m.Median := getIntSeqMedian(intseq);
        m.StandardDeviation := getIntSeqStdDev(intseq);
    } else {
        log("Error: invoked getIntSeqMinimum() on an empty integer sequence");
        return m;
    }

}

helper getIntSeqMinimum(intseq : Sequence(Integer)) : Integer{
    let sortedintseq : Sequence(Integer) = intseq->sortedBy(i|i) in if sortedintseq->notEmpty() then {
        m.Minimum := getIntSeqMinimum(intseq);
        m.Maximum := getIntSeqMaximum(intseq);
        m.Average := getIntSeqAverage(intseq);
        m.Median := getIntSeqMedian(intseq);
        m.StandardDeviation := getIntSeqStdDev(intseq);
    } else {
        log("Error: invoked getIntSeqMinimum() on an empty integer sequence");
    }

}
C.2 Source code of the model transformation QVTo to metrics

```plaintext
endif;

return null;
}

helper getIntSeqMaximum(intseq: Sequence(Integer)): Integer{
  let sortedintseq: Sequence(Integer) = intseq->sortedBy(i|i)
  in if sortedintseq->notEmpty()
    then return sortedintseq->last().oclAsType(Integer)
    else {log("Error: invoked getIntSeqMaximum() on an empty integer sequence");}
  endif;
  return null;
}

global return null;

helper getRealSeqMaximum(realseq: Sequence(Real)): Real {
  let sortedrealseq: Sequence(Real) = realseq->sortedBy(i|i)
  in if sortedrealseq->notEmpty()
    then return sortedrealseq->last().oclAsType(Real)
    else {log("Error: invoked getRealSeqMaximum() on an empty integer sequence");}
  endif;
  return null;
}

helper getIntSeqMedian(intseq: Sequence(Integer)): Integer {
  let sortedintseq: Sequence(Integer) = intseq->sortedBy(i|i)
  in if (sortedintseq->notEmpty()) then {
    var len: Integer := sortedintseq->size();
    if ( len.mod(2) = 0 )
      then {return sortedintseq->at(sortedintseq->size().div(2))}
      else {return sortedintseq->at((sortedintseq->size().div(2)) + 1) }
    endif;
  } else {log("Error: invoked getIntSeqMedian() on an empty integer sequence");}
  endif;

return null;
}

global return null;

helper getRealSeqMedian(realseq: Sequence(Real)): Real {
  let sortedrealseq: Sequence(Real) = realseq->sortedBy(i|i)
  in if sortedrealseq->notEmpty() then if sortedrealseq->size().mod(2) = 0
    then return sortedrealseq->at(sortedrealseq->size().div(2))
    endif;
  else {log("Error: invoked getRealSeqMedian() on an empty integer sequence");}
  endif;

return null;
}
```

135
C. IMPLEMENTATION DETAILS OF MQET

```java
else return sortedrealseq->at((sortedrealseq->size().div(2)) + 1)
endif
else log("Error: invoked getRealSeqMedian() on an empty real sequence")
endif;

return null;
}

helper getIntSeqAverage(intseq: Sequence<Integer>): Real {
    if intseq->notEmpty() then return getRound(intseq->sum() / intseq->size())
    else return 0
    endif;

return 0;
}

helper getRealSeqAverage(realseq: Sequence<Real>): Real{
    if realseq->notEmpty() then return getRound(realseq->sum() / realseq->size())
    else return 0
    endif;

return 0;
}

helper getIntSeqStdDev(intseq: Sequence<Integer>): Real {
    let avg: Real = getIntSeqAverage(intseq)
    in if intseq->size() > 1
    then return getSqrt((intseq->collect(i | (i - avg) * (i - avg))->sum() / (intseq->size() - 1)))
    else return 0
    endif;

return 0;
}

helper getRealSeqStdDev(realseq: Sequence<Real>): Real {
    let avg: Real = getRealSeqAverage(realseq)
    in if realseq->size() > 1
    then return getSqrt((realseq->collect(i | (i - avg) * (i - avg))->sum() / (realseq->size() - 1)))
    else return 0
    endif;

return 0;
}
```
C.2 Source code of the model transformation QVTo to metrics

```java
helper getStringOccurrenceCount(names: Sequence(String)):
Sequence(Integer) =
let uniquenames: Sequence(String) = names–>asSet()  -- Remove duplicates
                           –>asSequence()  -- Restore ordering (needed?)
in uniquenames
   ->collect(i | names
               ->count(i)
   );
```
Appendix D

Details of case studies

D.1 Xtend code of synchronous to asynchronous transformation

Listing D.1: Sync2Async.ext

```xtend
import slco;
import channels;

extension org::eclipse::xtend::util::stdlib::io;
extension Libraries::Equality;
extension Libraries::ModifyModel;

// ############################################################################
// Replaces an synchronous channel by an asynchronous channel and adds
// acknowledgements.
// ############################################################################
// Copy the classes that need to be modified and update all references to these
// classes.
// ############################################################################

slco::Model modifyModelS2AS(
```
D. DETAILS OF CASE STUDIES

```plaintext
slco :: Model model, channels::ChannelList channels
) :
    let cl = model.channels.select(
        c1
        | channels.channels.exists(
            c2
            | c1.channelEqualsChannel(c2)
        )
    ) :
        model.modifyModel(cl, "")
    -> cl.modifyClasses(cl)
    -> cl.modifyChannel()
    -> model.setName(
        model.name + ", as"
    )
    -> model
;

// ############################################################################
// Update Classes to add acknowledgement signals.
// ############################################################################
private Void modifyClasses(
    slco :: Channel channel, List[ slco :: Channel] channels
) :
    channel.object1.class.modifyClass(channels)
    -> channel.object2.class.modifyClass(channels)
;

private Void modifyChannel(slco::Channel channel) :
    channel.setIsSynchronous(false)
    -> channel.setIsBidirectional(true)
;

private Void modifyClass(slco::Class class, List[ slco :: Channel] channels) :
    channels.port1.modifyClass(class)
    -> channels.port2.modifyClass(class)
```
D.1 Xtend code of synchronous to asynchronous transformation

```xtend
private cached Void modifyClass(slco::Port port, slco::Class class) :
   class.stateMachines.modifyStateMachine(port);

private slco::StateMachine modifyStateMachine(
   slco::StateMachine stateMachine, slco::Port port
) :
   stateMachine.addIntermediateStates(port) -> stateMachine.addAcknowledges(port) -> stateMachine;

private Void addIntermediateStates(
   slco::StateMachine stateMachine, slco::Port port
) :
   stateMachine.transitions.addAll(
      stateMachine.transitions.createIntermediateStates(stateMachine, port)
   );

private Boolean sendsSignalToSynchronousPort(
   slco::Statement statement, slco::Port port
) :
   (statement.metaType == slco::SendSignalStatement &&
   port.portEqualsPort(
      ((slco::SendSignalStatement) statement).port
   ));

private Boolean receivesSignalFromSynchronousPort(

```

141
D. DETAILS OF CASE STUDIES

slco :: Trigger trigger , slco :: Port port

)

(  
  trigger . metaType == slco::SignalReception
  &&
  port . portEqualsPort(  
    (slco :: SignalReception) trigger . port
  )
  )

);

private create List[ slco :: Transition] createIntermediateStates(  
  slco :: Transition transition , slco :: StateMachine stateMachine ,  
  slco :: Port port
  ) :
  if (  
    transition . effect . exists(  
      s
      |  
      s . sendsSignalToSynchronousPort(port)
    )
  ) then (  
    let s = transition . effect . selectFirst (  
      s
      |  
      s . sendsSignalToSynchronousPort(port)
    ) :
    let nt = new slco::Transition :
    let ns = new slco::State :
      ns . setName(  
        transition . source . name + "," + stateMachine.vertices.indexOf(  
          transition . target
        ) . toString()
      )  
      nt . setName(  
        ns . name + "To" + transition . target . name
      )  
      nt . setSource(ns)  
      nt . setTarget(transition . target)  
      transition . setTarget(ns)  
      transition . setName(  
        transition . source . name + ",To" + ns . name
      )  
    )
D.1 Xtend code of synchronous to asynchronous transformation

```xtend
−> stateMachine.vertices.add(ns)
−> nt.effect.addAll(
    transition.effect.select(
        ds |
        transition.effect.indexOf(ds) > transition.effect.indexOf(s)
    )
)
−> this.add(nt)
−> this.addAll(
    nt.createIntermediateStates(stateMachine, port)
)
);

private Void addAcknowledges(
    slco::StateMachine stateMachine, slco::Port port
) :
    stateMachine.addAcknowledgeForSend(port)
−> stateMachine.addAcknowledgeForReceive(port)
;

private Boolean sendsAcknowledgeSignal(slco::Statement statement) :
{
    statement.metaType == slco::SendSignalStatement
    &&
    (slco::SendSignalStatement) statement.signalName.startsWith("Acknowledge")
}
;

private Boolean receivesAcknowledgeSignal(slco::Trigger trigger) :
{
    trigger.metaType == slco::SignalReception
    &&
    ((slco::SignalReception) trigger).signalName.startsWith("Acknowledge")
}
;```
D. DETAILS OF CASE STUDIES

```java
private Void addAcknowledgeForSend(
    slco::StateMachine stateMachine, slco::Port port
) :
    stateMachine.transitions.addAll(
        stateMachine.transitions.select(
            t |
            t.effect.exists(
                s |
                ( s.sendsSignalToSynchronousPort(port) 
                    &&
                    !s.sendsAcknowledgeSignal()
                )
            ).addAcknowledgeForSend(stateMachine, port)
        )
    );
```

```java
private Void addAcknowledgeForSend(
    slco::Transition transition, slco::StateMachine stateMachine, 
    slco::Port port
) :
    let sendStatements =
        transition.effect.select(
            s |
            ( s.sendsSignalToSynchronousPort(port) 
                &&
                !s.sendsAcknowledgeSignal()
            )
        )
    :
    let statement = ((slco::SendSignalStatement) sendStatements.first()) :
    let trigger = new slco::SignalReception :
        if (sendStatements.size == 1) then ( 
            if (transition.target.outgoing.size == 1) then ( 
                trigger.setPort(statement.port)
            )
            trigger.setSignalName( 
```
D.1 Xtend code of synchronous to asynchronous transformation

```xtend
"Acknowledge." + statement.signalName
-> transition.target.outgoing.first().setTrigger(trigger)
) else
syserr("More than one outgoing Transition encountered."
)
) else
syserr("More than one SendSignalStatement encountered."
)
);

// -----------------------------------------------
private Void addAcknowledgeForReceive(
  slco::StateMachine stateMachine, slco::Port port
) :
  stateMachine.transitions.select(t |
    (t.trigger.receivesSignalFromSynchronousPort(port)
    && !t.trigger.receivesAcknowledgeSignal() )
  ).addAcknowledgeForReceive();

// -----------------------------------------------
private Void addAcknowledgeForReceive(slco::Transition transition) :
  ((slco::SignalReception) transition.trigger).addAcknowledgeForReceive(transition
);

// -----------------------------------------------
private Void addAcknowledgeForReceive(
  slco::SignalReception signalReception, slco::Transition transition
);```

145
D. DETAILS OF CASE STUDIES

let s = new slco::SendSignalStatement:
    s.setPort(signalReception.port)
    $> s.setSignalName(
        "$ Acknowledge_" + signalReception.signalName
    )
  $> transition.setEffect(
    transition.effect.reverse().add(s).toList().reverse().toList()
  )
;

Listing D.2: Equality.ext

import slco;

// ####################################################################################
// Equality functions
// ####################################################################################

Boolean portEqualsPort(slco::Port port1, slco::Port port2):
  port1.name == port2.name
  &&
  ((slco::Class) port1.eContainer).name == ((slco::Class) port2.eContainer).name
;

Boolean channelEqualsChannel(slco::Channel channel1, slco::Channel channel2):
  channel1.name == channel2.name
  &&
  channel1.port1.portEqualsPort(channel2.port1)
  &&
  channel1.port2.portEqualsPort(channel2.port2)
  &&
  channel1.object1.objectEqualsObject(channel2.object1)
  &&
  channel1.object2.objectEqualsObject(channel2.object2)
;

Boolean objectEqualsObject(slco::Object object1, slco::Object object2):
  object1.name == object2.name
D.1 Xtend code of synchronous to asynchronous transformation

```xtend
&&
object1.class.name == object2.class.name
;

// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
Boolean classEqualsClass(slco::Class class1, slco::Class class2) :
    class1.name == class2.name
;

// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
Boolean stateMachineEqualsStateMachine(
    slco::StateMachine stateMachine1, slco::StateMachine stateMachine2 ) :
    stateMachine1.name == stateMachine2.name
    &&
    ((slco::Class) stateMachine1.eContainer).classEqualsClass(
        ((slco::Class) stateMachine2.eContainer)
    )
;

// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
Boolean transitionEqualsTransition(
    slco::Transition transition1, slco::Transition transition2 ) :
    transition1.name == transition2.name
    &&
    ((slco::StateMachine) transition1.eContainer).stateMachineEqualsStateMachine(
        ((slco::StateMachine) transition2.eContainer)
    )
;
```

Listing D.3: ModifyModel.ext

```xtend
import slco;

extension org::eclipse::xtend::util::stdlib::io;
extension org::eclipse::xtend::util::stdlib::cloning;
extension Libraries::Clone;
extension Libraries::Equality;

// ###################################################################################################################
// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
```
D. DETAILS OF CASE STUDIES

// Modification of models
// ----------------------------------
// ----------------------------------
slo::Model modifyModel(
    slo::Model model, List[slco:: Channel] channels, String suffix
) :
    channels.addClasses(model, suffix)
    -> channels.updateChannel(suffix)
    -> model.channels.without(channels).updateChannel(channels, suffix)
    -> model
;

Void addClasses(slo:: Channel channel, slo:: Model model, String suffix) :
    channel.port1.addClasses(
        channel.object1, channel.port2, model, suffix
    )
    -> channel.port2.addClasses(
        channel.object2, channel.port1, model, suffix
    )
;

Void addClasses(
    slo:: Port port1, slo:: Object object, slo:: Port port2, slo:: Model model,
    String suffix
) :
    if (
        model.channels.exists(
            c
                |
            c.port1 == port1 && c.port2 != port2
                ||
            c.port2 == port1 && c.port1 != port2
        )
    ) then (
        model.classes.add(
            object.class.copyClass(port1, suffix )
        )
        -> object.updateObject(port1, suffix)
    )
D.1 Xtend code of synchronous to asynchronous transformation

```xtend
) else {
    model.classes.add(
        object.class.copyClass(null, suffix)
    )
    object.updateObject(null, suffix)
}

// cached slco:: Class copyClass(slco:: Class class, slco:: Port port, String suffix) : let c = ((slco:: Class) class.cachedClone(port)) :
    if (port == null) then {
        c.setName(c.name + suffix)
    } else {
        c.setName(c.name + "_" + port.name + suffix)
    }
    c

// Void updateChannel(slco::Channel channel, String suffix) :
    channel.setPort1(
        channel.port1.getClonedPort(
            channel.object1.class
        )
    )
    channel.setPort2(
        channel.port2.getClonedPort(
            channel.object2.class
        )
    )
    channel.setName(channel.name + suffix)

// cached slco:: Object updateObject(
    slco:: Object object, slco:: Port port, String suffix
) :
    object.setClass(
        object.class.copyClass(port, suffix)
    )
    object.setName(object.name + suffix)
```
D. DETAILS OF CASE STUDIES

```java
Void updateChannel(
    slco::Channel channel, List<slco::Channel> channels, String suffix
) :
    if ( channels.exists(
        c |
        ( c.object1.objectEqualsObject(channel.object1) ||
        c.object2.objectEqualsObject(channel.object1) )
    )
    &
    ( channels.exists(
        c |
        ( c.object1.objectEqualsObject(channel.object2) ||
        c.object2.objectEqualsObject(channel.object2) )
    )
) then {
    channel.setPort1(
        channel.port1.getClonedPort(
            channel.object1.class
        )
    )
    -> channel.setPort2(
        channel.port2.getClonedPort(
            channel.object2.class
        )
    )
    -> channel.setName(channel.name + suffix)
} else if ( 
    channels.exists(
        c
    )
```
D.1 Xtend code of synchronous to asynchronous transformation

```xtend
| ( c. object1.objectEqualsObject(channel.object1) || c. object2.objectEqualsObject(channel.object1) )
| )
| ) then ( channel.setPort1(
| channel.port1.getClonedPort(
| channel.object1.class
| )
| ) -> channel.setName(channel.name + suffix)
| ) else if ( channels.exists(
| c | ( c. object1.objectEqualsObject(channel.object2) || c. object2.objectEqualsObject(channel.object2) )
| )
| ) then ( channel.setPort2(
| channel.port2.getClonedPort(
| channel.object2.class
| )
| ) -> channel.setName(channel.name + suffix)
| ) else ( channel )
| )

Listing D.4: Clone.ext

import slco;
extension org::eclipse::xtend::util::stdlib::io;
extension org::eclipse::xtend::util::stdlib::cloning;
extension Libraries::Equality;
```

---

151
D. DETAILS OF CASE STUDIES

D.2 Case study ucif2cif

The size of the QVTo transformation from ucif to cif are very large. This can be seen from the metrics extracted by the tool MQET. This tool had to take about more than 10 seconds to extract the metrics from the QVTo transformation from ucif to cif. The metrics extracted are shown in Tables D.1 and D.2. The source code of the QVTo transformation from ucif to cif can be found in the Deliverables, see Appendix E.
### D.2 Case study ucif2cif

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td># Mappings</td>
<td>252</td>
</tr>
<tr>
<td># Mappings with Condition</td>
<td>0</td>
</tr>
<tr>
<td># Overloaded Mappings</td>
<td>113</td>
</tr>
<tr>
<td># Unused Mappings</td>
<td>1</td>
</tr>
<tr>
<td># Abstract Mappings</td>
<td>0</td>
</tr>
<tr>
<td># Mapping Inheritances</td>
<td>0</td>
</tr>
<tr>
<td># Mapping Mergers</td>
<td>0</td>
</tr>
<tr>
<td># Mapping Disjunctions</td>
<td>18</td>
</tr>
<tr>
<td># Helpers</td>
<td>13</td>
</tr>
<tr>
<td># Unused Helpers</td>
<td>0</td>
</tr>
<tr>
<td># Overloaded Helpers</td>
<td>0</td>
</tr>
<tr>
<td># Calls to log()</td>
<td>41</td>
</tr>
<tr>
<td># Calls to assert()</td>
<td>39</td>
</tr>
<tr>
<td># Calls to Resolve Expressions</td>
<td>28</td>
</tr>
<tr>
<td># Modules</td>
<td>5</td>
</tr>
<tr>
<td># Library Modules</td>
<td>1</td>
</tr>
<tr>
<td># Operational Transformation Modules</td>
<td>4</td>
</tr>
<tr>
<td># Unused Modules</td>
<td>0</td>
</tr>
<tr>
<td># Unused Library Modules</td>
<td>0</td>
</tr>
<tr>
<td># Unused Operational Transformation Modules</td>
<td>0</td>
</tr>
<tr>
<td># Input Models</td>
<td>4</td>
</tr>
<tr>
<td># Output Models</td>
<td>4</td>
</tr>
<tr>
<td># Imported Metamodels</td>
<td>6</td>
</tr>
<tr>
<td># Intermediate Classes</td>
<td>0</td>
</tr>
<tr>
<td># Intermediate Properties</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td># Variables with same name but different types</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table D.1:** QVT metrics extracted from the ucif to cif transformation (1)
### D. DETAILS OF CASE STUDIES

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mapping metrics</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Mappings per Module</td>
<td>0</td>
<td>119</td>
<td>50.4</td>
<td>26</td>
<td>58.28</td>
</tr>
<tr>
<td># Subobjects per Mapping</td>
<td>9</td>
<td>2745</td>
<td>73.89</td>
<td>39</td>
<td>190.24</td>
</tr>
<tr>
<td># Parameters per Mapping</td>
<td>2</td>
<td>3</td>
<td>2.42</td>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td># Unused Parameters per Mapping</td>
<td>0</td>
<td>1</td>
<td>0.12</td>
<td>0</td>
<td>0.32</td>
</tr>
<tr>
<td># Parameters with direction in per Mapping</td>
<td>1</td>
<td>2</td>
<td>1.42</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td># Parameters with direction inout per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Parameters with direction out per Mapping</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td># Mappings per Mapping Name (Overloadings)</td>
<td>1</td>
<td>3</td>
<td>2.08</td>
<td>2</td>
<td>0.46</td>
</tr>
<tr>
<td># Variations per Mapping</td>
<td>2</td>
<td>15</td>
<td>4.44</td>
<td>3</td>
<td>3.57</td>
</tr>
<tr>
<td># Variables per Mapping</td>
<td>0</td>
<td>8</td>
<td>0.09</td>
<td>0</td>
<td>0.67</td>
</tr>
<tr>
<td># Unused Variables per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Operations on Collections per Mapping</td>
<td>0</td>
<td>11</td>
<td>0.79</td>
<td>0</td>
<td>1.35</td>
</tr>
<tr>
<td>Cyclomatic Complexity per Mapping</td>
<td>1</td>
<td>36</td>
<td>1.84</td>
<td>1</td>
<td>3.25</td>
</tr>
<tr>
<td><strong>Helper metrics</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Helpers per Module</td>
<td>0</td>
<td>10</td>
<td>2.6</td>
<td>1</td>
<td>4.22</td>
</tr>
<tr>
<td># Subobjects per Helper</td>
<td>33</td>
<td>1015</td>
<td>322.85</td>
<td>109</td>
<td>361.93</td>
</tr>
<tr>
<td># Helpers per Helper Name</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td># Parameters per Helper</td>
<td>1</td>
<td>3</td>
<td>2.08</td>
<td>2</td>
<td>0.49</td>
</tr>
<tr>
<td># Unused Parameters per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Parameters with direction in per Helper</td>
<td>1</td>
<td>2</td>
<td>1.15</td>
<td>1</td>
<td>0.38</td>
</tr>
<tr>
<td># Parameters with direction inout per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Parameters with direction out per Helper</td>
<td>0</td>
<td>1</td>
<td>0.92</td>
<td>1</td>
<td>0.28</td>
</tr>
<tr>
<td># Variables per Helper</td>
<td>0</td>
<td>9</td>
<td>0.69</td>
<td>0</td>
<td>2.5</td>
</tr>
<tr>
<td># Unused Variables per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Operations on Collections per Helper</td>
<td>0</td>
<td>2</td>
<td>0.15</td>
<td>0</td>
<td>0.55</td>
</tr>
<tr>
<td>Cyclomatic Complexity per Helper</td>
<td>2</td>
<td>61</td>
<td>18.31</td>
<td>7</td>
<td>21.43</td>
</tr>
</tbody>
</table>

Table D.2: QVTo metrics extracted from the ucif to cif transformation (2)
### Dependency metrics

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># Imported Modules per Module</td>
<td>0</td>
<td>2</td>
<td>0.8</td>
<td>0</td>
<td>1.1</td>
</tr>
<tr>
<td># Times a Module is Imported per Module</td>
<td>0</td>
<td>1</td>
<td>0.8</td>
<td>0</td>
<td>0.45</td>
</tr>
<tr>
<td># Calls from Mappings in other Modules per Module</td>
<td>0</td>
<td>25</td>
<td>9.8</td>
<td>1</td>
<td>12.99</td>
</tr>
<tr>
<td># Calls from Helpers in other Modules per Module</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Other Modules per Module</td>
<td>0</td>
<td>25</td>
<td>10.8</td>
<td>3</td>
<td>13.01</td>
</tr>
<tr>
<td># Calls to Mappings in Other Modules per Module</td>
<td>0</td>
<td>24</td>
<td>4.8</td>
<td>0</td>
<td>10.73</td>
</tr>
<tr>
<td># Calls to Helpers in Other Modules per Module</td>
<td>0</td>
<td>1</td>
<td>0.2</td>
<td>0</td>
<td>0.45</td>
</tr>
<tr>
<td># Calls to Other Modules per Module</td>
<td>0</td>
<td>25</td>
<td>5.0</td>
<td>0</td>
<td>11.18</td>
</tr>
<tr>
<td># Calls from Mappings to Mappings per Mapping</td>
<td>0</td>
<td>19</td>
<td>1.93</td>
<td>2</td>
<td>1.94</td>
</tr>
<tr>
<td># Calls from Mappings to Helpers per Mapping</td>
<td>0</td>
<td>15</td>
<td>0.13</td>
<td>0</td>
<td>1.03</td>
</tr>
<tr>
<td># Calls from Mappings to Mappings/Helpers per Mapping</td>
<td>0</td>
<td>19</td>
<td>2.06</td>
<td>2</td>
<td>2.17</td>
</tr>
<tr>
<td># Calls to Mappings from Mappings per Mapping</td>
<td>0</td>
<td>60</td>
<td>2.03</td>
<td>1</td>
<td>6.43</td>
</tr>
<tr>
<td># Calls to Mappings from Helpers per Mapping</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls to Mappings per Mapping</td>
<td>0</td>
<td>60</td>
<td>2.06</td>
<td>1</td>
<td>6.43</td>
</tr>
<tr>
<td># Calls from Helpers to Helpers per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Calls from Helpers to Mappings per Helper</td>
<td>0</td>
<td>16</td>
<td>1.23</td>
<td>0</td>
<td>4.44</td>
</tr>
<tr>
<td># Calls from Helpers to Mappings/Helpers per Helper</td>
<td>0</td>
<td>16</td>
<td>1.23</td>
<td>0</td>
<td>4.44</td>
</tr>
<tr>
<td># Calls to Helpers per Helper</td>
<td>1</td>
<td>32</td>
<td>3.92</td>
<td>1</td>
<td>8.5</td>
</tr>
<tr>
<td># Calls to log() per Module</td>
<td>0</td>
<td>22</td>
<td>8.2</td>
<td>3</td>
<td>9.44</td>
</tr>
<tr>
<td># Calls from Mappings to log() per Mapping</td>
<td>0</td>
<td>2</td>
<td>0.1</td>
<td>0</td>
<td>0.33</td>
</tr>
<tr>
<td># Calls from Helpers to log() per Helper</td>
<td>0</td>
<td>3</td>
<td>1.0</td>
<td>1</td>
<td>0.71</td>
</tr>
<tr>
<td># Calls to assert() per Module</td>
<td>0</td>
<td>22</td>
<td>7.8</td>
<td>2</td>
<td>9.76</td>
</tr>
<tr>
<td># Calls from Mappings to assert() per Mapping</td>
<td>0</td>
<td>2</td>
<td>0.1</td>
<td>0</td>
<td>0.33</td>
</tr>
<tr>
<td># Calls from Helpers to assert() per Helper</td>
<td>0</td>
<td>1</td>
<td>0.85</td>
<td>1</td>
<td>0.38</td>
</tr>
<tr>
<td># Calls to Resolve Expressions per Module</td>
<td>0</td>
<td>28</td>
<td>5.6</td>
<td>0</td>
<td>12.52</td>
</tr>
<tr>
<td># Calls from Mappings to Resolve Expressions per Mapping</td>
<td>0</td>
<td>7</td>
<td>0.11</td>
<td>0</td>
<td>0.57</td>
</tr>
<tr>
<td># Calls from Helpers to Resolve Expressions per Helper</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

### Misc. metrics

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># Input Models per Operational Transformation</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td># Output Models per Operational Transformation</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td># Metamodels per Module</td>
<td>2</td>
<td>4</td>
<td>3.2</td>
<td>4</td>
<td>1.1</td>
</tr>
<tr>
<td># Subobjects per Module</td>
<td>95</td>
<td>9610</td>
<td>5323.8</td>
<td>7470</td>
<td>4840.27</td>
</tr>
<tr>
<td># Intermediate Classes per Operational Transformation</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td># Intermediate Properties per Operational Transformation</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Table D.3:** QVTo metrics extracted from the ucif to cif transformation (3)
D. DETAILS OF CASE STUDIES
Appendix E

Deliverables

All the files created during the course of the graduation project, can be found in the FALCON repository of the SET group, in a folder called ”MSc Assignment Phu”. In this folder, I have put the exports of the Eclipse Workspaces, that contained all the files of JXET, XXET and MQET.