Université de Montréal

# Survey of Template-Based Code Generation

par
Lechanceux Kavuya Luhunu

Département d’informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l’obtention du grade de Maître ès sciences (M.Sc.)
en computer science

Avril, 2017

## RÉSUMÉ

L’automatisation de la génération des artefacts textuels à partir des modèles est une étape critique dans l’Ingénierie Dirigée par les Modèles (IDM). C’est une transformation de modèles utile pour générer le code source, sérialiser les modèles dans de stockages persistents, générer les rapports ou encore la documentation. Parmi les différents paradigmes de transformation de modèle-au-texte, la génération de code basée sur les *templates* (TBCG) est la plus utilisée en IDM. La TBCG est une technique de génération qui produit du code à partir des spécifications de haut niveau appelées *templates*. Compte tenu de la diversité des outils et des approches, il est nécessaire de classifier et de comparer les techniques de TBCG existantes afin d’apporter un soutien approprié aux développeurs. L’objectif de ce mémoire est de mieux comprendre les caractéristiques des techniques de TBCG, identifier les tendances dans la recherche, et éxaminer l’importance du rôle de l’IDM par rapport à cette approche. J’évalue également l’expressivité, la performance et la mise à l’échelle des outils associés selon une série de modèles. Je propose une étude systématique de cartographie de la littérature qui décrit une intéressante vue d’ensemble de la TBCG et une étude comparitive des outils de la TBCG pour mieux guider les dévloppeurs dans leur choix. Cette étude montre que les outils basés sur les modèles offrent plus d’expressivité tandis que les outils basés sur le code sont les plus performants. Enfin, Xtend2 offre le meilleur compromis entre l’expressivité et la performance.

**Mots clés: Ingénierie dirigée par les modèles, Génération de code, Étude systématique de cartographie, Étude comparative.**

## ABSTRACT

A critical step in model-driven engineering (MDE) is the automatic synthesis of a textual artifact from models. This is a very useful model transformation to generate application code, to serialize the model in persistent storage, generate documentation or reports. Among the various model-to-text transformation paradigms, Template-Based Code Generation (TBCG) is the most popular in MDE. TBCG is a synthesis technique that produces code from high-level specifications, called templates. It is a popular technique in MDE given that they both emphasize abstraction and automation. Given the diversity of tools and approaches, it is necessary to classify and compare existing TBCG techniques to provide appropriate support to developers. The goal of this thesis is to better understand the characteristics of TBCG techniques, identify research trends, and assess the importance of the role of MDE in this code synthesis approach. We also evaluate the expressiveness, performance and scalability of the associated tools based on a range of models that implement critical patterns. To this end, we conduct a systematic mapping study of the literature that paints an interesting overview of TBCG and a comparative study on TBCG tools to better guide developers in their choices. This study shows that model-based tools offer more expressiveness whereas code-based tools performed much faster. Xtend2 offers the best compromise between the expressiveness and the performance.

**Keywords: Model-driven engineering, Code generation, Systematic mapping study, Comparative study.**

**CONTENTS**

**RÉSUMÉ** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **ii**

**ABSTRACT** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **iii**

**CONTENTS** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **iv**

**LIST OF TABLES** . . . . . . . . . . . . . . . . . . . . . . . . . . **viii**

**LIST OF FIGURES** . . . . . . . . . . . . . . . . . . . . . . . . . **ix**

**LIST OF APPENDICES** . . . . . . . . . . . . . . . . . . . . . . . **xi**

**DEDICATION** . . . . . . . . . . . . . . . . . . . . . . . . . . . . **xii**

**ACKNOWLEDGMENTS** . . . . . . . . . . . . . . . . . . . . . . . **xiii**

**CHAPTER 1: INTRODUCTION** . . . . . . . . . . . . . . . . . . . **1**

1.1 Context . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

1.2 Problem Statement and Thesis Proposition . . . . . . . . . . . . 2

1.3 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.4 Outline . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

**CHAPTER 2: BACKGROUND AND STATE OF THE ART** . . . . . . . **4**

2.1 Code Generation . . . . . . . . . . . . . . . . . . . . . . . . . 4

2.2 Code Generation in the Context of MDE . . . . . . . . . . . . . . 5

2.3 Code Generation Techniques . . . . . . . . . . . . . . . . . . . 6

2.4 Template-based Code Generation . . . . . . . . . . . . . . . . . 7

2.5 Literature Reviews on Code Generation . . . . . . . . . . . . . . 8

2.6 Tools performance and metamodel pattern . . . . . . . . . . . . . . . . 11

**CHAPTER 3: SYSTEMATIC MAPPING STUDY OF TBCG . . . . . . . 12**

3.1 Research Methods . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

3.1.1 Objectives . . . . . . . . . . . . . . . . . . . . . . . . . . 13

3.1.2 Selection of Source . . . . . . . . . . . . . . . . . . . . . . 13

3.1.3 Screening Procedure . . . . . . . . . . . . . . . . . . . . . 14

3.1.4 Classification Scheme . . . . . . . . . . . . . . . . . . . . 15

3.2 Paper Selection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18

3.2.1 Paper Collection . . . . . . . . . . . . . . . . . . . . . . . 19

3.2.2 Screening . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

3.2.3 Eligibility during Classification . . . . . . . . . . . . . . . . 20

3.3 Evolution of TBCG . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

3.3.1 General trend . . . . . . . . . . . . . . . . . . . . . . . . . 21

3.3.2 Publications and venues . . . . . . . . . . . . . . . . . . . 22

3.4 Characteristics of Template-Based Code Generation . . . . . . . . . . . 23

3.4.1 Template style . . . . . . . . . . . . . . . . . . . . . . . . 23

3.4.2 Input type . . . . . . . . . . . . . . . . . . . . . . . . . . . 25

3.4.3 Output type . . . . . . . . . . . . . . . . . . . . . . . . . . 27

3.4.4 Application scale . . . . . . . . . . . . . . . . . . . . . . . 28

3.4.5 Context . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29

3.4.6 Orientation . . . . . . . . . . . . . . . . . . . . . . . . . . 30

3.4.7 Application domain . . . . . . . . . . . . . . . . . . . . . . 30

3.5 Relations between Characteristics . . . . . . . . . . . . . . . . . . . . 32

3.5.1 Statistical correlations . . . . . . . . . . . . . . . . . . . . 32

3.5.2 Other interesting relations . . . . . . . . . . . . . . . . . . 33

3.6 Template-based Code Generation Tools . . . . . . . . . . . . . . . . . 36

3.6.1 Popular tools . . . . . . . . . . . . . . . . . . . . . . . . . . 37

3.6.2 Unspecified and other tools . . . . . . . . . . . . . . . . . . . . . 39

3.6.3 Trends of tools used . . . . . . . . . . . . . . . . . . . . . . . . 39

3.6.4 Characteristics of tools . . . . . . . . . . . . . . . . . . . . . . . 40

3.7 MDE and Template-based Code Generation . . . . . . . . . . . . . . . 41

3.8 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43

3.8.1 RQ1: What are the trends in TBCG? . . . . . . . . . . . . . . . 43

3.8.2 RQ2: What are the characteristics of TBCG approaches? . . . . 43

3.8.3 RQ3: To what extent are TBCG tools being used? . . . . . . . . 44

3.8.4 RQ4: What is the place of MDE in TBCG? . . . . . . . . . . . 44

3.9 Threats to validity . . . . . . . . . . . . . . . . . . . . . . . . . . . 45

3.9.1 Construction validity . . . . . . . . . . . . . . . . . . . . . . . 45

3.10 External validity . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47

**CHAPTER 4: COMPARISON OF THE EXPRESSIVENESS AND PERFORMANCE OF TBCG TOOLS . . . . . . . . . . . . . . 48**

4.1 Metamodel patterns for template implementation . . . . . . . . . . . 48

4.1.1 Pattern 1: Navigation . . . . . . . . . . . . . . . . . . . . . . . 49

4.1.2 Pattern 2: Variable dependency . . . . . . . . . . . . . . . . . . . 50

4.1.3 Pattern 3: Polymorphism . . . . . . . . . . . . . . . . . . . . . . 51

4.1.4 Pattern 4: Recursion . . . . . . . . . . . . . . . . . . . . . . . . 52

4.1.5 Combination of patterns . . . . . . . . . . . . . . . . . . . . . . 54

4.2 Tools expressiveness . . . . . . . . . . . . . . . . . . . . . . . . . . 55

4.2.1 Code generation tools . . . . . . . . . . . . . . . . . . . . . . . 55

4.2.2 Pattern-based comparison of tools . . . . . . . . . . . . . . . . . 61

4.3 Tools performance . . . . . . . . . . . . . . . . . . . . . . . . . . . 68

4.3.1 Experiment setup . . . . . . . . . . . . . . . . . . . . . . . . . . 68

4.3.2 Data collection . . . . . . . . . . . . . . . . . . . . . . . . . 70

4.3.3 Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71

4.3.4 Performance without recursion . . . . . . . . . . . . . . . . . 71

4.3.5 Performance with recursion . . . . . . . . . . . . . . . . . . . 72

4.3.6 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . 73

4.3.7 Limitations . . . . . . . . . . . . . . . . . . . . . . . . . . . 74

**CHAPTER 5: CONCLUSION . . . . . . . . . . . . . . . . . . . . . 75**

5.1 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 75

5.1.1 Systematic mapping study of TBCG . . . . . . . . . . . . . . . 75

5.1.2 Comparison of the Expressiveness and Performance of TBCG Tools . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76

5.2 Outlook . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76

**BIBLIOGRAPHY . . . . . . . . . . . . . . . . . . . . . . . . . . . 78**

# LIST OF TABLES

3.I Evolution of paper corpus during the study process . . . . . . . . 18
3.II Most popular venues . . . . . . . . . . . . . . . . . . . . . . . . 24

4.I Classification of TBCG tools. . . . . . . . . . . . . . . . . . . . . 57
4.II Summary of the qualitative evaluation of the tools expressiveness. 67
4.III Characteristics of the input models . . . . . . . . . . . . . . . . . 69

II.I Classification table 1–31 . . . . . . . . . . . . . . . . . . . . . . . lxxxii
II.II Classification table 32–63 . . . . . . . . . . . . . . . . . . . . . . lxxxiii
II.III Classification table 64–95 . . . . . . . . . . . . . . . . . . . . . . lxxxiv
II.IV Classification table 96–127 . . . . . . . . . . . . . . . . . . . . . lxxxv
II.V Classification table 128–159 . . . . . . . . . . . . . . . . . . . . lxxxvi
II.VI Classification table 160–191 . . . . . . . . . . . . . . . . . . . . lxxxvii
II.VII Classification table part 192–223 . . . . . . . . . . . . . . . . . . lxxxviii
II.VIII Classification table part 224–255 . . . . . . . . . . . . . . . . . . lxxxix
II.IX Classification table 256–287 . . . . . . . . . . . . . . . . . . . . xc
II.X Classification table 288–319 . . . . . . . . . . . . . . . . . . . . xci
II.XI Classification table 320–351 . . . . . . . . . . . . . . . . . . . . xcii
II.XII Classification table 352–383 . . . . . . . . . . . . . . . . . . . . xciii
II.XIII Classification table 384–415 . . . . . . . . . . . . . . . . . . . . xciv
II.XIV Classification table part 416–447 . . . . . . . . . . . . . . . . . . xcv
II.XV Classification table 448–481 . . . . . . . . . . . . . . . . . . . . xcvi

## LIST OF FIGURES

2.1 Components of TBCG . . . . . . . . . . . . . . . . . . . . . . . . 7

3.1 The systematic mapping process we followed . . . . . . . . . . . 12
3.2 Evolution of papers in the corpus . . . . . . . . . . . . . . . . . . . 22
3.3 Distribution of template style facet . . . . . . . . . . . . . . . . . . 23
3.4 Template style evolution . . . . . . . . . . . . . . . . . . . . . . . . 26
3.5 Design-time input distribution . . . . . . . . . . . . . . . . . . . . 26
3.6 Design-time input evolution . . . . . . . . . . . . . . . . . . . . . . 27
3.7 Distribution of output type facet . . . . . . . . . . . . . . . . . . . 28
3.8 Distribution of application scale facet. . . . . . . . . . . . . . . . . 29
3.9 Distribution of context facet. . . . . . . . . . . . . . . . . . . . . . 29
3.10 Distribution of application domain facet . . . . . . . . . . . . . . 31
3.11 Relation between template style (vertical) and input/output types (horizontal) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
3.12 Relation between output (vertical) and design-time input (horizontal) types showing the number of papers in each intersection . . . 35
3.13 Tools categories . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
3.14 Popular tools . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
3.15 Evolution of the MDE facet . . . . . . . . . . . . . . . . . . . . . . 41

4.1 Navigation pattern . . . . . . . . . . . . . . . . . . . . . . . . . . . 49
4.2 Variable dependency pattern . . . . . . . . . . . . . . . . . . . . . . 50
4.3 Polymorphism pattern . . . . . . . . . . . . . . . . . . . . . . . . . 51
4.4 Recursion pattern . . . . . . . . . . . . . . . . . . . . . . . . . . . 52
4.5 Invoice metamodel . . . . . . . . . . . . . . . . . . . . . . . . . . . 54

4.6 Tool performance for experiment A in log scale . . . . . . . . . . . 71
4.7 Tool performance for experiment B in log scale . . . . . . . . . . . 73

## LIST OF APPENDICES

**Appendix I:** **Final corpus bibliography of the SMS** . . . . . . . . . . . . **xiv**

**Appendix II:** **Classification table of the SMS** . . . . . . . . . . . . . . . . **lxxxi**

**Appendix III:** **Templates of each tool for the experiments** . . . . . . . . . **xcvii**

I would like to dedicate this thesis to the people who supported me throughout this amazing journey. To my parents, thank you for your unconditional love and devotion. You have agreed to huge sacrifices to make this a successful adventure and I will always be grateful for that. To my brothers and sisters, for your prayers and advices. Your encouragements have made me stronger than you can ever imagine. To my friends who made my daily life more enjoyable.

## ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Prof. Eugene Syriani for the continuous support of my study and research, for his patience, motivation, enthusiasm, and immense knowledge.

I thank my fellow lab-mates in GEODES Group, for the stimulating discussions and for all the fun we have had in the last two years.

Last but not the least, I would like to thank my family: my brothers, my sisters and my parents, for supporting me in every aspects of my life.

# CHAPTER 1

# INTRODUCTION

## 1.1 Context

Model-driven engineering (MDE) has advocated the use of model-to-text transformations as a core component of its paradigm [55]. A common workflow in MDE is to produce a program without the need of programming. Modelers first describe a domain-specific platform-independent model. This model is refined with platform-specific concepts from the target framework of the final application. The platform-specific model is then synthesized to the source code of the program using a dedicated M2T tool [74]. These transformations are used to generate application code, serialize models in persistent storage, document, visualize or explore models.

Code generation has been around since the 1950s, taking its origin in early compilers [73]. Since then, software organizations have been relying on code synthesis techniques in order to reduce development time and increase productivity [52]. Automatically generating code is a generic approach where the same generator can be reused to produce many different artifacts according to the varying inputs it receives. It also provides opportunities to detect errors in the input artifact early on before the generated code is compiled, when the output is source code.

There are many techniques to generate code, such as programmatically, using a meta-object protocol, or aspect-oriented programming. Since the mid-1990s, template-based code generation (TBCG) emerged as an approach requiring less effort for the programmers to develop code generators. Templates favor reuse following the principle of *write once, produce many*. The concept was heavily used in web designer software (such as Dreamweaver) to generate web pages and Computer Aided Software

Engineering (CASE) tools to generate source code from UML diagrams. Many development environments started to include a template mechanism in their framework such as Microsoft Text Template Transformation Toolkit (T4) [65] for .NET and Velocity [6] for Apache.

## 1.2 Problem Statement and Thesis Proposition

The software engineering research community has focused essentially on primary studies proposing new TBCG techniques, tools and applications. Furthermore, TBCG is a popular M2T technique in MDE given that it emphasizes abstraction and automation. This has led to over 70 different tools developed in the past two decades. However, there are no instructions to guide them. As a result, MDE developers are faced with a difficult choice when selecting the most appropriate M2T tool [74].

To overcome these issues, the developers can rely on web forums dedicated to code generation and collect the maximum information about the tools before selecting the best fit. However, there is a high possibility of favoring misleading informations since the reliability of the source cannot be verified. Furthermore, the developers can simply select the most popular tool and carry out their code generation task. However, it is possible that the selected tool is unable to satisfy the developer's expectations.

Therefore, our proposal is to conduct a systematic mapping study of the literature in order to understand the trends, identify the characteristics of TBCG, assess the popularity of existing tools, and determine the influence that MDE has had on TBCG. We are interested in various facets of TBCG, such as characterizing of the templates, of inputs and outputs, along with the evolution of the amount of publications using TBCG over the past 16 years. Based on this systematic literature study, we compare the nine most popular TBCG tools found in the literature. We perform a qualitative evaluation of their expressiveness based on typical metamodel patterns that influence the implementation of the templates. The expressiveness of a tool is the set of language constructs that can be used to complete a particular task natively. This is important since, to the best of our knowledge, there are no available metrics to assess the code generation templates. We also evaluate the performance and scalability of these tools based on a range of models that conform to a metamodel composed by the combination of these patterns.

## 1.3 Contributions

The goal of this thesis is to add valuable knowledge in the field of TBCG by studying both the literature and the tools. The contributions of this thesis are the following:

1. A systematic mapping study of the literature to paint an interesting picture about the trends and uses of TBCG.
2. A comparative study of TBCG tools to evaluate their expressiveness power and test their performance.

## 1.4 Outline

This thesis is organized as follows. In Chapter 2, we introduce the necessary background on TBCG and discuss the related work. In Chapter 3, we describe in details our systematic mapping study of the literature. In Chapter 4, we describe our comparative study on TBCG tools. We present the metamodel patterns we used for this study. We report on the expressiveness of the tools in and their performance. Finally, we conclude in Chapter 5.

# CHAPTER 2

# BACKGROUND AND STATE OF THE ART

In this chapter, we review the notion of code generation and introduce TBCG. We also briefly outline MDE principles to better understand its relationship with TBCG. Finally we discuss related work on systematic mapping studies in general and secondary studies about code generation.

## 2.1 Code Generation

In this paper, we view code generation as in automatic programming [73] rather than compilers. The underlying principle of automatic programming is that a user defines what he expects from the program and the program should be automatically generated by a software without any assistance by the user. This generative approach is different from a compiler approach.

Compilers produce code executable by a computer from a specification conforming to a programming language, whereas automatic programming transforms user specifications into code which often conforms to a programming language. Compilers have a phase called code generation that retrieves an abstract syntax tree produced by a parser and translates it into machine code or bytecode executable by a virtual machine. Compared to code generation as in automatic programming, compilers can be regarded as tasks or services that are incorporated in or post-positioned to code generators [49].

As Balzer [7] states, there are many advantages to code generation. The effort of the user is reduced as he has fewer lines to write: specifications are shorter than the program that implements them. Specifications are easier to write and to understand for a user, given that they are closer to the application and domain concepts. Writing specifications is less error-prone than writing the program directly, since the expert is the one who writes the specification rather than another programmer.

These advantages are in fact the pillar principles of MDE and domain-specific modeling. Floch et al. [35] observed many similarities between MDE and compilers research and principles. Thus, it is not surprising to see that many, though not exclusively, code generation tools came out of the MDE community. The advantages of code generation should be contrasted with some of its limitations. For example, there are issues related to integration of generated code with manually written code and to evolving specifications that require to re-generate the code [80]. Sometimes, relying too much on code generators may produce an overly general solution that may not necessarily be optimal for a specific problem.

## 2.2 Code Generation in the Context of MDE

MDE is a software development approach that uses abstraction to bridge the gap between the problem space and the software implementation [80]. To bridge the gap between the application domain and the solution domain, MDE uses models to describe complex systems at multiple levels of abstraction, as well as automated support for transforming and analyzing models. This separation allows the description of key intellectual assets in a way that is not coupled to specific programming languages or target platforms.

Domain-specific modeling (DSM) [42] is a branch of MDE that allows models to be manipulated at the level of abstraction of the application domain the model is intended for, rather than at the level of computing. In DSM, domain experts can create models that describe some computational need using abstractions and notations that match their own domain of expertise. Thus, end-users who do not possess the skills needed to write computer programs using traditional languages (like Java or C++) can describe their solution in a more familiar language.

In MDE parlance, models represent abstractions of a real system, capturing some of its essential properties. A model conforms to a metamodel, which defines the abstract syntax and static semantics of the modeling language. This language can either be a domain-specific language (DSL) or a general purpose language like UML. Developers manipulate models by means of model transformation. Transformations can have different purposes [61], such as translating, or refining models. One particular kind of model transformation is devoted to code generation with model-to-text transformations [27].

A common workflow in MDE is to produce a program without the need of programming [55]. Modelers first describe the high-level level system in a computation-independent model. This is then evolved into a domain-specific platform-independent model. This model is in turn refined with platform-specific concepts from the target framework of the final application. The platform-specific model is then synthesized to the source code of the program using a dedicated model-to-text transformation tool [74]. Model-to-text transformations are used to implement code, generate documentation, serialize models, or visualize and explore models. We refer to [49] for a history of code generation and an in-depth explanation of its role in MDE.

## 2.3 Code Generation Techniques

As briefly outlined in [49] and in [27], there are many techniques that can be used to generate code. We briefly outline the main ones here.

**Visitor based** approaches consist of programmatically traversing the internal representation of the input, while relying on an API dedicated to manipulate the input and to write the output to a text stream. This is used in [14].

**Meta-programming** is a language extension approach, such as using a meta-object protocol. For example, in OpenJava [82], a Java meta-program creates a Java file, compiles it on the fly, and loads the generated program in its own run-time.

**In-line generation** relies on a preprocessor that generates additional code to the existing one, such as with the C++ standard template library or C macro preprocessor instructions. An example is available in [11].

**Code annotations** are added in-line to existing code and is internally transformed into more expanded code. Examples include JavaDoc and attributes in C#. This approach is used in [26].

**Template based** is described below.

## 2.4 Template-based Code Generation

The literature agrees on a general definition of model-to-text code generation [27] and on templates. Jörges [49] identifies three components in TBCG: the data, the template, and the output. However, there is another component that is not mentioned which is the meta-information the generation logic of the template relies on. Therefore, we conducted this study according to the following notion of TBCG.

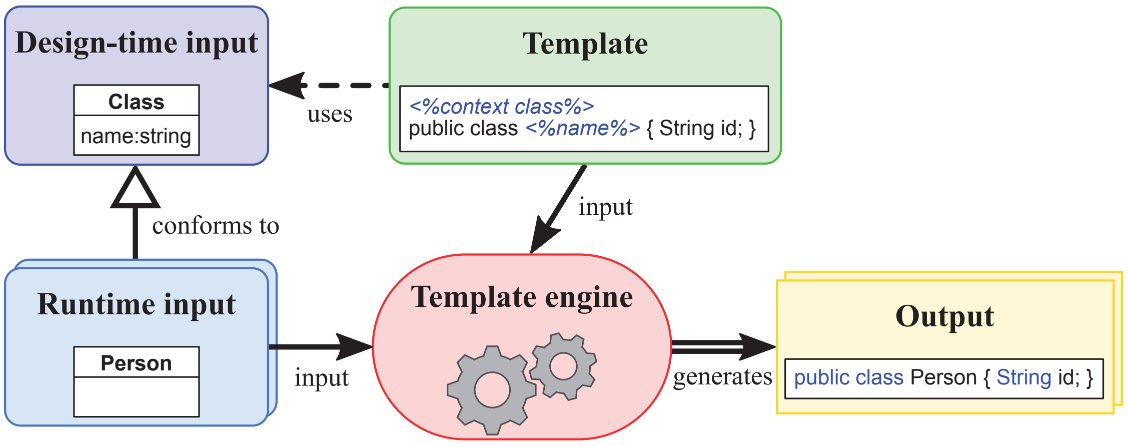

Figure 2.1: Components of TBCG

Figure 2.1 summarizes the main concepts of TBCG. We consider TBCG as a synthesis technique that uses templates in order to produce a textual artifact, such as source code, called the *output*. A template is an abstract and generalized representation of the textual output it describes. It has a *static part*, text fragments that appear in the output "as is". It also has a *dynamic part* embedded with splices of meta-code that encode the generation logic. Templates are executed by the *template engine* to compute the dynamic part and replace meta-codes by static text according to *run-time input*. The *design-time input* defines the meta-information which the run-time input conforms to. The dynamic part of a template relies on the design-time input to query the run-time input by filtering the information retrieved and performing iterative expansions on it. Therefore, TBCG relies on a design-time input that is used to define the template and a run-time input on which the template is applied to produce the output. For example, a TBCG engine that takes as run-time input an XML document relies on an XML schema as design-time input. Definition 1 summarizes our definition of TBCG.

**Definition 1.** A synthesis technique is a TBCG if it specifies a set of templates, assumes a design-time input, requires run-time inputs, and produces textual output.

For example, the work in [51] generates a C# API from Ecore models using Xpand. According to Definition 1, the templates of this TBCG example are Xpand templates, the design-time input is the metamodel of Ecore, the run-time input is an Ecore model, and the output is a C# project file and C# classes.

## 2.5 Literature Reviews on Code Generation

In evidence-based software engineering [54], a systematic literature review is a secondary study that reviews primary studies with the aim of synthesizing evidence related to a specific research question. Several forms of systematic reviews exist depending on the depth of reviewing primary studies and on the specificities of research questions. Unlike conventional systematic literature reviews that attempt to answer a specific question, a systematic mapping studies (SMS) aim at classifying and performing a thematic analysis on a topic [53]. SMS is a secondary study method that has been adapted from other disciplines to software engineering in [15] and later evolved by Petersen et al. in [69]. A SMS is designed to provide a wide overview of a research area, establish if research evidence exists on a specific topic, and provide an indication of the quantity of the evidence specific to the domain.

Over the years, there have been many primary studies on code generation. However, we could not find any secondary study on TBCG explicitly. Still, the following are closely related secondary studies.

Mehmood et al. [64] performed a SMS regarding the use of aspect-oriented modeling for code generation, which is not based on templates. They analyzed 65 papers mainly based on three main categories: the focus area, the type of research, and the type of contribution. The authors concluded that this synthesis technique is still immature. The study shows that no work has been reported to use or evaluate any of the techniques proposed.

Gurunule et al. [44] presented a comparison of aspect orientation and MDE techniques to investigate how they can each be used for code generation. The authors found that further research in these areas can lead to significant advancements in the development of software systems. Unlike Mehmood et al. [64], they did not follow a systematic and repeatable process.

Dominguez et al. [29] performed a systematic literature review of studies that focus on code generation from state machine specifications. The study is based on a set of 53 papers, which have been classified into two groups: pattern-based and not pattern-based. The authors do not take template-based approaches into consideration.

Batot et al. [10] performed a SMS on model transformations solving a concrete problem that have been published in the literature. They analyzed 82 papers based on a classification scheme that is general to any model transformation approach, which includes model-to-text transformations. They conclude that concrete model transformations have been pulling out from the research literature since 2009 and are being considered as development tasks. They also found that 22% of their corpus solve concrete problems using refinement and code synthesis techniques. Finally, they found that research in model transformations is heading for a more stable and grounded validation.

There are other studies that attempted to classify code generation techniques. However, they did not follow a systematic and repeatable process. For example, Czarnecki et al. [27] proposed a feature model providing a terminology to characterize model transformation approaches. They distinguished two categories for model-to-text approaches: those that are visitor-based and those that are template-based; the latter being in line with Definition 1. The authors found that many new approaches to model-to-model transformation have been proposed recently, but relatively little experience is available to assess their effectiveness in practical applications.

Rose et al. [74] extended the feature model of Czarnecki et al. to focus on template-based model-to-text transformation tools. Their classification is centered exclusively on tool-dependent features. Their feature model has been synthesized from a comparison of six M2T tools. They identified four mandatory features and as many optional features divided that characterize a M2T language. Like us, their goal is to help developers when they are faced to choose between different tools. This study is close to the work of Czarnecki in [27] but focuses only on a feature model for M2T. In our work, we have implemented templates for specific metamodel patterns and we discuss about the challenges and limitations and how to remedy to these for each tool.

## 2.6 Tools performance and metamodel pattern

There have also been many studies on model transformation tools performance. However, most of them deal with other types of model transformations and do not include M2T transformations. Still, the following are closely related works.

Bergmann et al. [13] proposed a benchmark evaluation of incremental pattern matching in graph transformations. They carried out various measurements to assess the performance of the incremental pattern matcher of a graph transformation tool and identified the key areas where its performance could be increased. The difference with our study is that we evaluate TBCG tools performance without any benchmark.

The transformation tool contest papers [48, 72, 85] investigated and compared graph transformation tools that participated in various transformation tool contests. The goal of this type of paper is to help the reader gain an overview of the field and its tools. The difference with our study is that we focus on the performance of M2T tools only while in their case they investigate other model transformation tools.

Cho et al. [25] analyzed Domain Specific Modeling Language (DSML) concrete syntax and identified their recurring problems such as the best way to design or evolve a base metamodel if the concrete syntax is more complex. Finally they proposed three metamodel design patterns as initial solutions to solve those problems. The main difference with our study is that we define the patterns from the point of view of the implementation of the template. We focus on the potential impact that they can have in implementing the dynamic part of the template while Cho et al.simply focus on the construction of metamodels using design patterns and threat them as metamodel components.

# CHAPTER 3

# SYSTEMATIC MAPPING STUDY OF TBCG

We propose a systematic mapping study of the literature to paint an interesting picture about the trends and uses of TBCG. In this chapter, we describe the research methods and the paper selection process. We also report the results of our study and the explain their implications. Furthermore, we discuss about the role of MDE in TBCG, the use of the associate tools and the trends in TBCG.

## 3.1 Research Methods

In order to analyze the topic of TBCG, we conducted a SMS following the process defined by Petersen et al. in [69] and summarized in Figure 3.1. The definition of research question is discussed in Section 3.1.1. The search conduction is described in Section 3.1.2. We present the screening of papers in Section 3.1.3. The relevant papers are obtained based on the criteria presented in Section 3.1.3.1 and Section 3.1.3.2. The keywording using abstracts step is described in Section 3.1.4. The last step corresponds to assigning a value to each facet of the classification scheme.

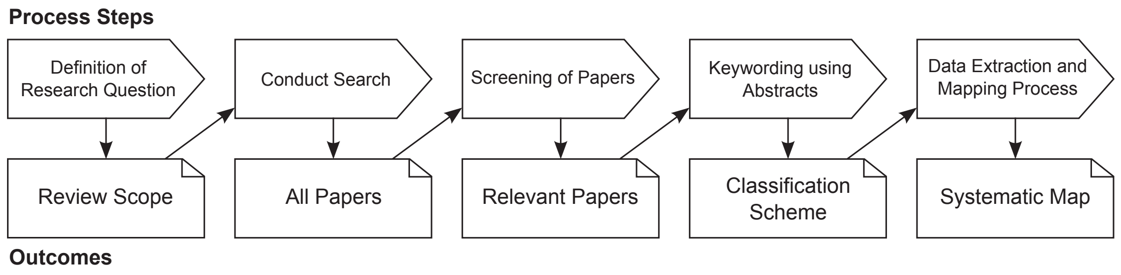

Figure 3.1: The systematic mapping process we followed

### 3.1.1 Objectives

The objective of this study is to obtain an overview of the current research in the area of TBCG and to characterize the different approaches that have been developed. We defined four research questions to set the scope of this study:

1. **What are the trends in template-based code generation?** We are interested to know how this technique has evolved over the years.
2. **What are the characteristics of template-based code generation approaches?** We want to identify major characteristics of this techniques and their tendencies.
3. **To what extent are template-based code generation tools being used?** We are interested in identifying popular tools and their uses.
4. **What is the place of MDE in template-based code generation?** We seek to determine whether and how MDE has influenced TBCG.

### 3.1.2 Selection of Source

We delimited the scope of the search to be regular publications that mention TBCG as at least one of the approaches used for code generation and published between 2000–2016. Therefore, this includes publications where code generation is not the main contribution. For example, Buchmann et al. [19] used TBCG to obtain ATL code while their main focus was implementing a higher-order transformation. Given that not all publications have the term "code generation" in their title, we formulated a query that retrieves publications based on their title, abstract, or full text (when available) mentioning "template" and "code generation", their variations, and synonyms. We used the following query:

```
template* AND "code generat*" OR "code synthesi*"
```

The used query was validated with a sample of 100 pre-selected papers we knew should be included.

### 3.1.3 Screening Procedure

Screening is the most crucial phase in a SMS [69]. We followed a two-stage screening procedure: automatic filtering, then title and abstract screening. In order to avoid the exclusion of papers that should be part of the final corpus, we followed a strict screening procedure. With four reviewers at our disposal, each article is screened by at least two reviewers independently. Two other students and I worked on a sample of 88 papers. When both reviewers of a paper disagree upon the inclusion or exclusion of the paper, a physical discussion is required. If the conflict is still unresolved, an additional senior reviewer is involved in the discussion until a consensus is reached. To determine a fair exclusion process, a senior reviewer reviews a sample of no less than 20% of the excluded papers at the end of the screening phase, to make sure that no potential paper is missed.

#### 3.1.3.1 Inclusion criteria

A paper is included if it explicitly indicates the use of TBCG or if it proposes a TBCG technique. We also include papers if the name of a TBCG tool appears in the title, abstract, or content.

#### 3.1.3.2 Exclusion criteria

Results from the search were first filtered automatically to discard records that were outside the scope of this study: papers not in computer science, not in the software engineering domain, with less than two pages of length (e.g., proceedings preface), not peer-reviewed (e.g., white papers), not written in English, or not published between the years 2000 and 2016. Then, papers were excluded through manual inspection based on the following criteria:

– **No code generation.** There is no code generation technique used.
– **Not template-based code generation.** Code generation is mentioned, but the considered technique is not template-based according to Definition 1.
– **Not a paper.** This exclusion criterion spans papers that were not caught by the automatic filtering. For example, some papers had only the abstract written in English and the content of the paper in another language. Additionally, there were 24 papers where the full text was not accessible online.

For the first two criteria, when the abstract did not give enough details about the code generation approach, a quick look at the full text helped clear any doubts on whether to exclude the paper or not. Reviewers were conservative on that matter.

### 3.1.4 Classification Scheme

There are generally two ways to construct the classification scheme [69, 76]. One approach consists of extracting the classification scheme by analyzing the included papers and determining the important classification properties form the abstract, keywords or content. Alternatively, one can construct a scheme using the general knowledge of the field. In our study, we used a hybrid approach in which we combined our general knowledge with the information extracted from the abstracts during the screening phase. The classification scheme is used to classify all retained papers along different facets that are of interest in order to answer our research questions. It helps analyzing the overall results and gives an overview of the trends and characteristics of TBCG. The facets we classified the corpus with are the following:

**Template style**: We characterize the style of the templates used in code generation approach.

  – **Predefined**: This template style is reserved for approaches where the template used for code generation is defined internally to the tool. However, a subset of the static part of the template is customizable to vary slightly the generated output. This is, for example, the case for common CASE tools where there is a predefined template to synthesize a class diagram into a number of programming languages. Nevertheless, the user can specify what language construct to use for association ends with a many cardinality, such as `Array` or `ArrayList` for Java templates.
  – **Output-based**: This style covers templates that are syntactically based on the actual target output. In contrast with the previous style, output-based templates offer full control on how the code is generated, both on the static and dynamic parts. The generation logic is typically encoded in meta-code as in the example of Figure 2.1.
  – **Rule-based**: In this style, templates focus on computing the dynamic part with the static part being implicit. The template lists declarative production rules that are applied on-demand by the template engine to obtain the final target output. For example, this is used to render the concrete textual syntax from the abstract syntax of a model using a grammar.

**Input type**: This facet consists of the design-time and runtime inputs. We characterize the language of the design-time input that is necessary to develop templates. We also characterize the input given to the generator during the execution of a TBCG. Generally, the run-time input is an instance that conforms to the design-time input.
  – **General purpose**: for generic languages reusable across different domains that are not programming languages, such as UML. Instances of a generic language include the Ecore model of a particular class diagram.
  – **Domain specific**: for languages targeted for a particular domain, such as the

metamodel of a DSL. Instances of a DSL include a Simulink model.

  - **Schema**: for structured data definitions, such as XML schema definition or database schema. Instances of data that follows a well-defined structure include XML.
  - **Programming language**: for well-defined programming languages. The runtime input consists of a source code.

**Output type**: We characterize the artifacts output by the code generator. A paper may be classified in more than one of the following categories.

  - **Source code**: for executable code conforming to a specific programming language.
  - **Structured data**: for code that is not executable, such as HTML.
  - **Natural language**: when plain text is generated.

**Tool**: We capture the tool or language used for TBCG. If a tool is not clearly identified in a paper or the TBCG is programmed directly, we classify the tool as **unspecified**. We consider a tool to be **popular** when it is used in at least 1% of the papers. Otherwise, we classify it in the **other** category.

**MDE**: We determine whether the part of the solution where TBCG is applied in the paper follows MDE techniques and principles. A good indication is if the design-time input is a metamodel.

**Context**: We determine where TBCG falls in the overall transformation process of the approach. We already presented a typical workflow in Section 2.2. Code generation is never the first step unless it is **standalone**. Otherwise, it is either used as an **intermediate** step or it is the **last** step of a transformation process.

**Application scale**: We characterize the scale of the artifact on which the TBCG approach is applied. We distinguish between **large scale** applications, **small scale**, or **no application** when the code generation was not applied on any example.

**Application domain**: We classify the general domain TBCG has been applied on. For example, this includes **Software engineering**, **Embedded systems**, etc.

**Orientation**: We distinguish **industrial** papers, where at least one author is affiliated to industry, from **academic** papers otherwise.

**Publication type**: We distinguish papers published in **conference** proceedings, as **journal** articles, or **other** formats such as workshop proceedings or book collections.

**Venue type**: We classify papers based on the where they have been published. We distinguish between general **software engineering** venues, venues specific to **MDE**, and all **other** venue types.

## 3.2 Paper Selection

Table 3.I summarizes the flow of information through the selection process of this study. This section explains how we obtained the final corpus of papers.

| **Phase** | **Number of papers** |
|---|---:|
| **Collection** | |
| Engineering Village | 4 043 |
| Scopus | 932 |
| SpringerLink | 2 671 |
| *Initial corpus* | *5 131* |
| **Screening** | |
| Excluded during screening | 4 553 |
| *Included* | *578* |
| **Classification** | |
| Excluded during classification | 99 |
| *Final corpus* | *481* |

Table 3.I: Evolution of paper corpus during the study process

### 3.2.1 Paper Collection

The paper collection step was done in two phases: querying and automatic duplicates removal. There are several online databases that index software engineering literature. For this study, we considered three main databases to maximize coverage: ENGINEERING VILLAGE [1], SCOPUS [2], and SPRINGERLINK [3]. The first two cover typical software engineering editors (IEEE XPLORE, ACM DIGITAL LIBRARY, ELSEVIER). However, from past experiences [10], they do not include all of SPRINGER publications. We used the search string from Section 3.1.2 to retrieve all papers from these three databases. We obtained 7 646 candidate papers that satisfy the query and the options of the search stated in Section 3.1.3.2. We then removed automatically all duplicates using EndNote software. This resulted in 5 131 candidate papers for the screening phase.

### 3.2.2 Screening

Based on the exclusion criteria stated in Section 3.1.3.2, each candidate paper was screened by at least two reviewers to decide on its inclusion. To make the screening phase more efficient, we used a home-made tool [76]. After all the reviewers completed screening the papers they were assigned, the tool calculates an inter-rater agreement coefficient. In our case, the Cohen's Kappa coefficient was 0.813. This high value shows that the reviewers were in almost perfect agreement.

Among the initial corpus of candidate papers, 4 556 were excluded, 551 were included and 24 received conflicting ratings.

During the screening, the senior reviewer systematically verifies each set of 100 rejected papers for sanity check. A total of 7 more papers were included back hence the rejected papers were reduced to 4 549. Almost all cases of conflicts were about a dis-

1. `https://www.engineeringvillage.com/`
2. `https://www.scopus.com/`
3. `http://link.springer.com/`

agreement on whether the code generation technique of a paper was using templates or not. These conflicts were resolved in physical meetings and 20 of them were finally included for a total of 578 papers and 4 553 excluded.

Among the excluded papers, 52% were rejected because *no code generation* was used. We were expecting such a high rate because terms such as "templates" are used in many other fields, like biometrics. Also, many of these papers were referring to the C++ standard template library [60], which is not about code generation. We counted 34% papers excluded because they were *not using templates*. Examples of such papers are cited in Section 2.3. Also, more than a quarter of the papers were in the compilers or embedded system domains, where programming the code generation phase is more complex as it requires more computations compared to using a template mechanism. Finally, 5% of the papers were considered as *not a paper*. In fact, this criterion was in place to catch papers that escaped the automatic filtering from the databases.

### 3.2.3 Eligibility during Classification

Once the screening phase over, we thoroughly analyzed the full text of the remaining 578 papers to classify them according to our classification scheme. Doing so allowed us to confirm that the code generation approach was effectively template-based according to Definition 1. We encountered papers that used multiple TBCG tools: they either compared tools or adopted different tools for different tasks. We classified each of these papers as a single publication, but incremented the occurrence corresponding to the tools referred to in the paper. This is the case of [32] where the authors use Velocity and XSLT for code generation. Velocity generates Java and SQL code, while XSLT generates the control code.

We excluded 99 additional papers. During screening, we detected situations where the abstract suggested the implementation of TBCG, whereas the full text proved otherwise. In most of the cases, the meaning of TBCG differed from the description presented in Section 2.4. As shown in [77] the terms template-based and generation are used in the context of networking and distributed systems. We also encountered circumstances where the tool mentioned in the abstract requires the explicit use of another component to be considered as TBCG, such as Simulink TLC, as in [67].

The final corpus considered for this study contains 481 papers and it is available in Appendix II.

## 3.3 Evolution of TBCG

We start with a thorough analysis of the trends in TBCG in order to answer the first research question.

### 3.3.1 General trend

Figure 3.2 reports the number of papers per year, averaging around 28. The general trend indicates that the number of publications with at least one template-based code generation method started increasing in 2002 to reach a first local maximum in 2005 and then remained relatively constant until 2012. This increase coincides with the early stages of MDE and the first edition of the MODELS conference, previous called UML conference. This is a typical trend where a research community gets carried away by the enthusiasm of a new potentially interesting domain, which leads to more publications. However, this does not represent the most prolific period for TBCG. In fact, in 2013 we notice a significant peak with 2.4 times the average numbers of publications observed in the previous years. Figure 3.2 then shows a sudden decrease in 2015.

Resorting to statistical methods, the high coefficient of variability and modified Thompson Tau test indicate that 2013 and 2015 are outliers in the range 2005–2016, where the average is 37 papers per year. The sudden isolated peak in 2013 is the result

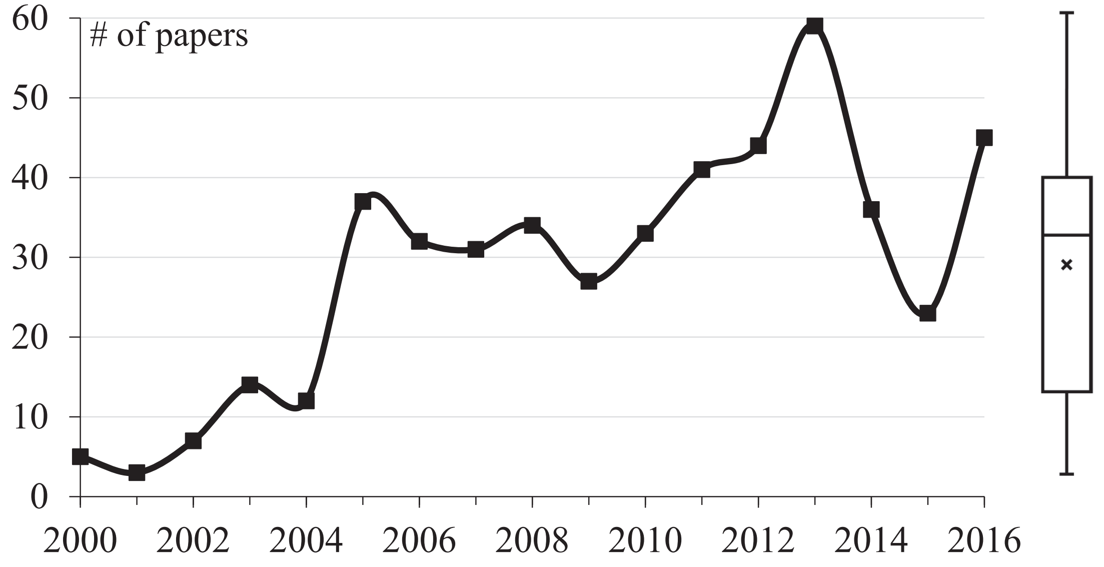

Figure 3.2: Evolution of papers in the corpus

of a special event or popularity of TBCG. The following decrease in the amount of papers published should not be interpreted as a decline in interest in TBCG, but that some event happened around 2013 which boosted publications, and then it went back to the steady rate of publication as previous years. In fact, in 2016 the standard deviation is above the average.

### 3.3.2 Publications and venues

We analyzed the papers based on the type of publication and the venue of their publication. MDE venues account for only 22% of the publications, so are software engineering venues, while the majority (56%) were published in other venues. Table 3.II shows the most popular venues that have at least five papers from the final corpus. These top venues account for just more than a quarter of the total number of publications. Among them, MDE venues account for 60% of the papers. MODELS[4], SOSYM, and ECMFA[5]

4. We grouped the UML conference with MODELS.
5. We grouped the ECMDA-FA conference with ECMFA.

are the three most popular venues with a total of 71 publications between them. This is very significant given that the average is only 1.67 paper per venue with a standard deviation of 2.63. Also, 43% of venues had only one paper using TBCG, which is the case for most of the other venues.

The peak in 2013 was mainly influenced by MDE and software engineering venues. However the drop in 2015 is the result of an accumulation of the small variations among the other venues. Since 2014, MDE venues account for 10–12 papers per year, while only 6–7 in software engineering.

As for the publication type, conference publications have been dominating at 64%. Journal article account for 24% of all papers. Interestingly, we notice a steady increase in journal articles, reaching a maximum of 15 in 2016.

## 3.4 Characteristics of Template-Based Code Generation

We examine the characteristics of TBCG using the classification scheme presented in Section 3.1.4.

### 3.4.1 Template style

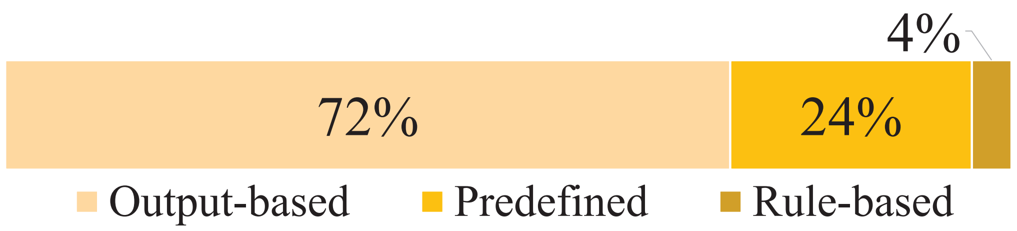

Figure 3.3: Distribution of template style facet

As the stacked bar chart in Figure 3.3 illustrates, the vast majority of the publications follow the **output-based** style. This consists of papers like [28], where Xpand is used to generate workflow code used to automate modeling tools. There, it is the final output

| Venue | Venue & Publication type | | # Papers |
|---|---|---|---|
| Model Driven Engineering Languages and Systems (MODELS) | MDE | Conference | 26 |
| Software and Systems Modeling (SOSYM) | MDE | Journal | 26 |
| European Conference on Modeling Foundations and Applications (ECMFA) | MDE | Conference | 19 |
| Generative and Transformational Techniques in Software Engineering (GTTSE) | Soft. eng. | Conference | 11 |
| Generative Programming: Concepts & Experience (GPCE) | Soft. eng. | Conference | 8 |
| International Conference on Computational Science and Applications (ICCSA) | Other | Conference | 8 |
| Software Language Engineering (SLE) | MDE | Conference | 7 |
| Leveraging Applications of Formal Methods, Verification and Validation (ISOLA) | Other | Conference | 7 |
| Automated Software Engineering (ASE) | Soft. eng. | Journal+Conference | 5+2 |
| International Conference on Web Engineering (ICWE) | Other | Conference | 6 |
| Evaluation of Novel Approaches to Software Engineering (ENASE) | Soft. eng. | Conference | 5 |

Table 3.II: Most popular venues

target text that drives the development of the template. This high score is expected since output-based style is the original template style for TBCG as depicted in Figure 3.4. This style has always been the most popular style since 2000.

The **predefined** style is the second most popular. Most of these papers generate code using a CASE tool, such as [39] that uses Rhapsody to generate code to map UML2 semantics to Java code with respect to association ends. Apart from CASE Tools, we also classified papers like [84] as predefined style since the output code is already fixed as HTML and the programmer uses the tags to change some values based on the model. There is no other action that can be performed to further customize the final code. Each year, around 28% of the papers were using the predefined style, except for a peak of 39% in 2005, given the popularity of CASE tools then. We found 19 publications that used **rule-based** style templates. This includes papers like [45] which generates Java code with Stratego from a DSL. A possible explanation of such a low score is that this is the most difficult template style to implement. It had a maximum of two papers per year throughout the study period.

### 3.4.2 Input type

**General purpose languages** account for almost half of the design-time input of the publications, as depicted in Figure 3.5. UML (class) diagrams, which are used as meta-models for code generation, are the most used for 87% of these papers as in [28] where a class diagram is provided an design-time input to generate workflow. Other general purpose languages that were used are, for example, the architecture analysis and design language (AADL) [18] and feature diagrams [20]. The **schema** category comes second with 21% of the papers. For example, a database schema is used as input at design-time in [59] to generate Java for a system that demonstrates that template can improve software development. Also, an XML schema is used in [41] as design-time input to

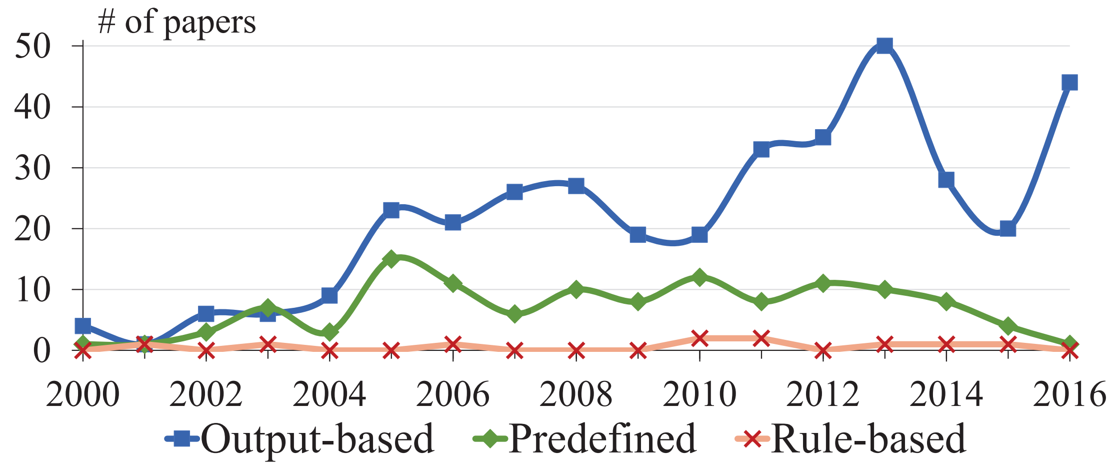

Figure 3.4: Template style evolution

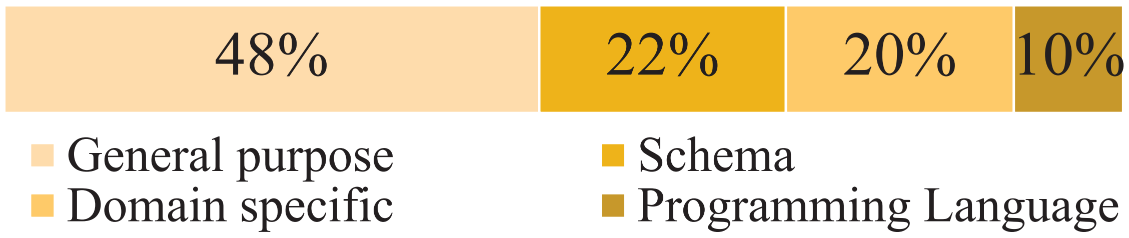

Figure 3.5: Design-time input distribution

produce C programs in order to implement an approach that can efficiently support all the configuration options of an application in embedded systems. **DSLs** are almost at par with schemata. They have been gaining popularity and gradually reducing the gap with general purpose languages. For example in [21], a custom language is given as the design input in order to generate C and C++ to develop a TBCG approach dedicated to real-time systems. The least popular design-time input type is **programming language**. This includes papers like [34] where T4 is used to generate hardware description (VHDL) code for configurable hardware. In this case, the input is another program on which the template depends.

Over the years, the general-purpose category has dominated the input type facet,

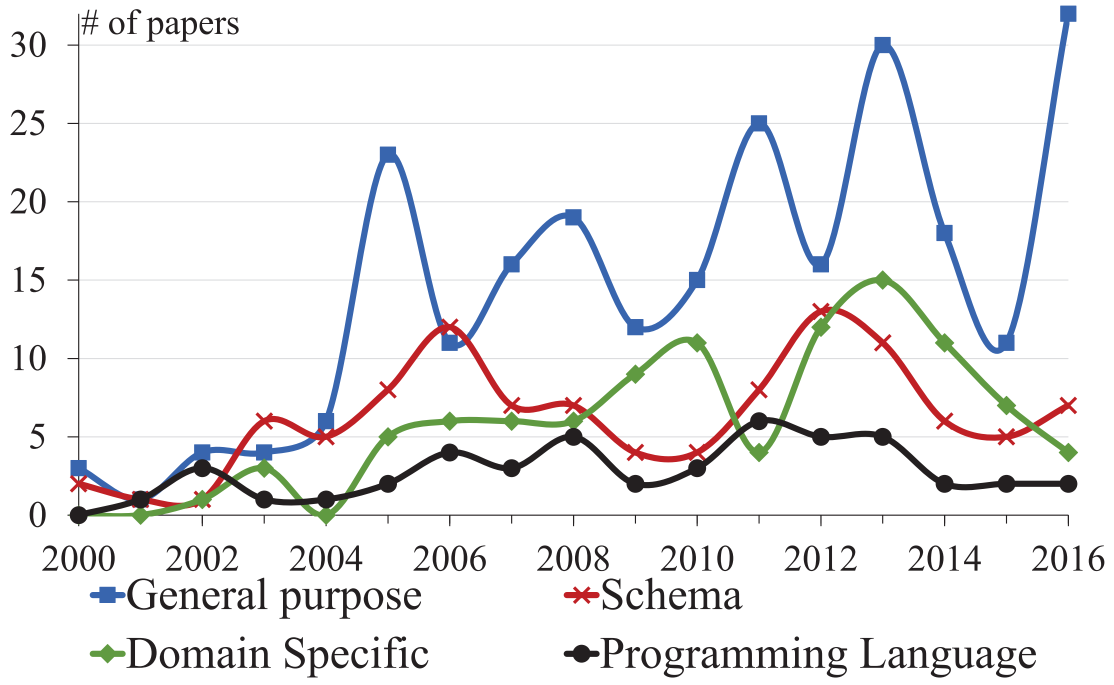

Figure 3.6: Design-time input evolution

as depicted in Figure 3.6. 2003 and 2006 were the only exceptions where schema obtained slightly more publications. We also notice a shift from schema to domain-specific design-time input types. Domain-specific input started increasing in 2009 but never reached the same level as general purpose. Programming language input maintained a constant level, with an average of 1% per year. Interestingly, in 2011, there were more programming languages used than DSLs.

Run-time input follows the same trend as design-time input. This is expected since run-time input is an instance of design-time input.

### 3.4.3 Output type

Figure 3.7 shows the distribution of output type facet. An overwhelming majority of the papers use TBCG to generate **source code**. This includes papers like [24] where Java code is generated an adaptable access control tool for electronic medical records.

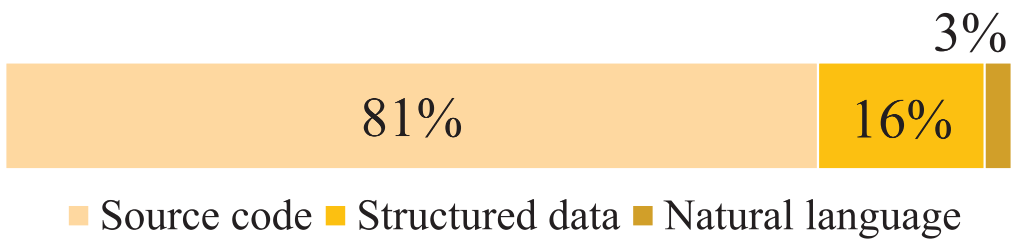

Figure 3.7: Distribution of output type facet

Java and C are the most targeted programming languages with respectively 69% and 19% of the time. Writing a program manually often requires proved abilities especially with system and hardware languages, such as VHDL [17]. This is why 10% of these papers generate low level source codes. Generation of **structured data** includes TBCG of mainly XML and HTML files. For example [36] produces both HTML and XML as parts of the web component to ease regression testing. Interestingly, we were able to find 13 papers that generate **natural language** text (in English). For example in [79], the authors present an automatic technique for identifying code fragments that implement high level abstractions of actions and expressing them as a natural language description. In addition, we found that around 4% of the papers generate combinations of at least two output types. This includes papers such as [86] that generate both C# and HTML from a domain-specific model and [28] that produce Java as well as natural language text for a system that provides workflow and automation tools for modeling.

Structured data and natural language output remained constant over the years, unlike source code which follows the general trend.

### 3.4.4 Application scale

As depicted in Figure 3.8, most papers applied TBCG on **large scale** examples. This result indicates that TBCG is a technique which scales with larger amounts of data. This includes papers like [56] that uses Acceleo to generate hundreds of lines of aspect-

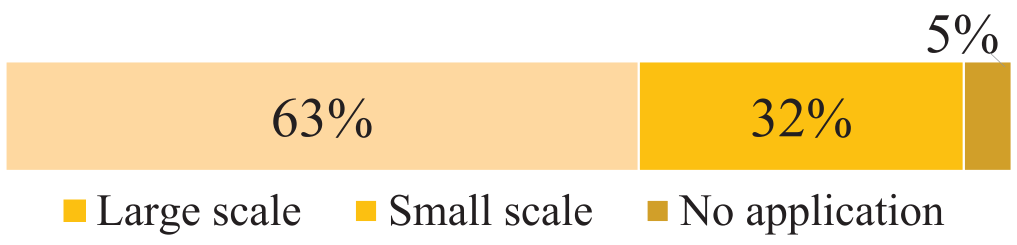

Figure 3.8: Distribution of application scale facet.

oriented programming code. **Small scale** obtains 32% of the papers. This is commonly found in research papers that only need a small and simple example to illustrate their solution. This is the case in [47] in which a small concocted example shows the generation process with the Epsilon Generation Language (EGL) [57]. **No application** was used in 5% of the publications. This includes papers like [31] where authors just mention that code synthesis is performed using a tool named Mako-template. Even though the number of publications without an actual application is very low, this demonstrates that some authors have still not adopted good practice to show an example of the implementation. This is important, especially when the TBCG approach is performed with a newly developed tool.

While large-scale applications follow the general trend of papers, the other two categories remained constant over the years.

### 3.4.5 Context

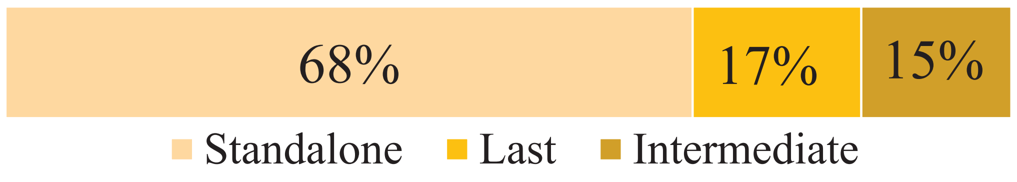

Figure 3.9: Distribution of context facet.

The distribution of context facet is presented in Figure 3.9. TBCG was used most of the time **standalone**, such as in [84]. The other two classes **last** and **intermediate** obtain respectively 18% and 15% of the papers. As an example, TBCG is an intermediate step in [79] where the generated algorithm is given as one of the inputs of an extraction task. TBCG is the last step of a process in [37] that starts with the execution of the various tasks of an integration system and ends with the generation of the final source code. Most papers only focus on the code generation part but this may have been a part of a bigger project.

### 3.4.6 Orientation

A quarter (24%) of the papers in the corpus are authored by a researcher from **industry**. The remaining 76% are written only by **academics**. This is a typical distribution since industrials tend to not publish their work. This result shows that TBCG is used in industry as in [51]. Industry oriented papers have gradually increased since 2003 until they reached a peak in 2013.

### 3.4.7 Application domain

The tree map in Figure 3.10 highlights the fact that TBCG is used in many different areas. **Software engineering** obtains more than half of the papers with 55% of the publications. We have grouped in this category other related areas like ontologies, information systems or software product lines. This is expected given that the goal of TBCG is to synthesize software applications. For example, the work in [12] uses the Rational CASE tool to generate Java programs in order to implement an approach that transforms UML state machine to behavioral code. The next category is **embedded systems** which obtains 13% of papers. Embedded systems often require low level hardware code difficult to write. Some even consider code generation to VHDL as a compilation rather than

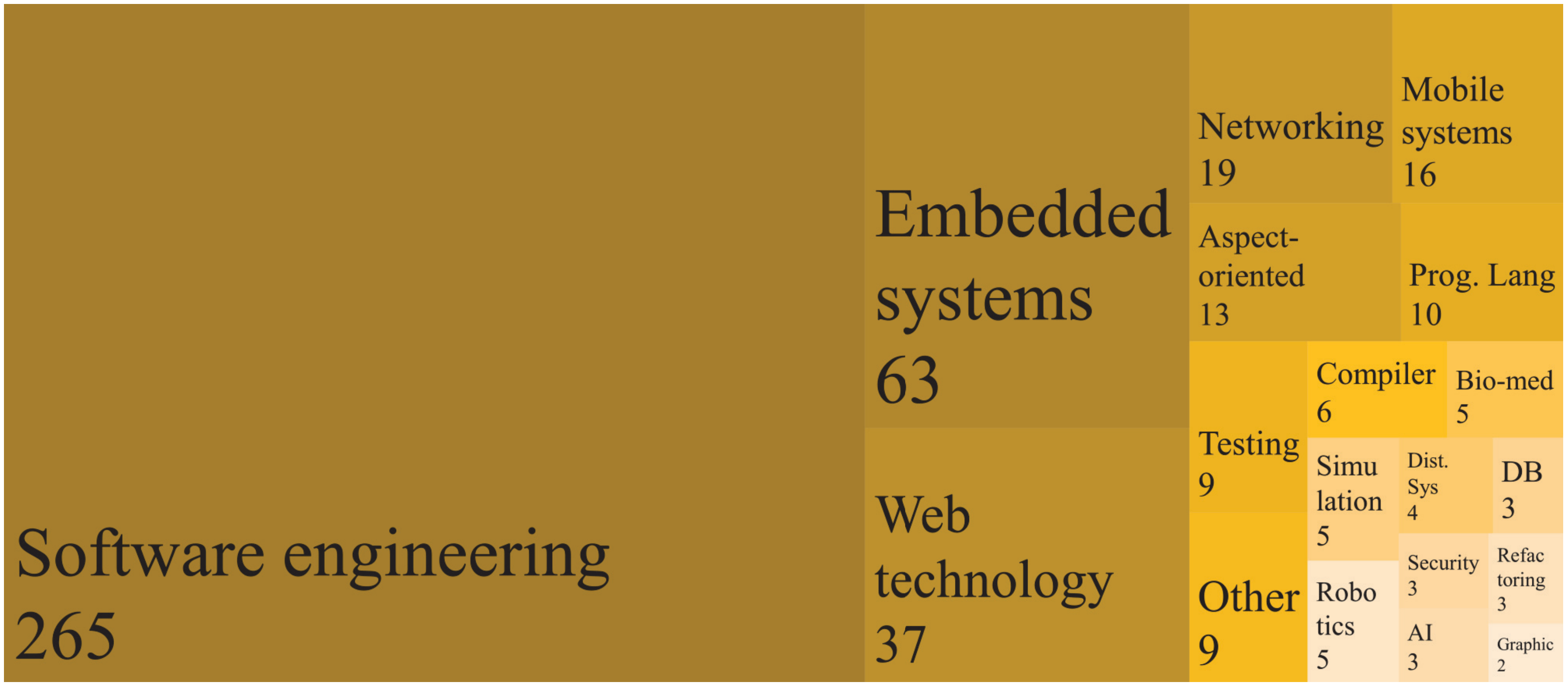

Figure 3.10: Distribution of application domain facet

automatic programming. In this category, we found papers like [30] in which Velocity is used to produce Verilog code to increase the speed of simulation. **Web technology** related application domains account for 8% of the papers. It consists of papers like [75] where the authors worked to enhance the development dynamic web sites. **Networking** obtains 4% of the papers, such as [22] where code is generated for a telephony service network. **Compiler** obtains 1% of the papers, such as [63] where a C code is generated and optimized for an Intel C compiler. It is interesting to note that several papers were applied in domains such as **bio-medicine** [70], **artificial intelligence** [37], and **graphics** [71].

We combined application domains with a single paper into the **other** category. This regroups domains such as agronomy, education, and finance. It is important to mention that the domain discussed in this category corresponds to the domain of application of TBCG employed, which differs from the publication venue.

## 3.5 Relations between Characteristics

To further characterize the trends observed in Section 3.4, we identified significant and interesting relations between the different facets of the classification scheme.

### 3.5.1 Statistical correlations

A Shapiro-Wilk test of each category determined that the none of them are normally distributed. Therefore, we opted for the Spearman two-tailed test of non-parametric correlations with a significance value of 0.05 to identify correlations between the trends of each category. The only significantly strong correlations we found statistically are between the two input types, and between MDE and input type.

With no surprise, the correlation between **run-time and design time input** is the strongest among all, with a correlation coefficient of 0.944 and a *p*-value of less than

0.001. This concurs with the results found in Section 3.4.2. An example is when the design-time input is UML, the run-time input is always a UML diagram as in [70]. Such a strong relationship is also noticeable in [40] with programming languages and source code, as well as in [37] when a schema design is used for structured data. As a result, all run-time input categories are correlated to the same categories as for design-time input. We will therefore treat these two facets together as *input type*.

There is a strong correlation of coefficient of 0.738 and a p-value of less than 0.001 between **input type and MDE**. As expected, more than 90% of the papers using general purpose and domain specific inputs are follow the MDE approach.

### 3.5.2 Other interesting relations

We also found weak but statistically significant correlations between the remaining facets. We discuss the result here.

#### 3.5.2.1 Template style

Figure 3.11 shows the relationship between template style, design-time input, and output types. We found that for the predefined templates, there are twice as many papers that use schema input than domain specific. However, for output-based, domain specific inputs are used slightly more often. We also notice that general purpose input is never used with rule-based templates. The output type follows the same general distribution regardless of the template style.

We found no rule-based style approach that has validated the TBCG component in their paper. User studies and formal validations were only performed on approaches using output-based templates.

All rule-based style approaches have included a sample application. Meanwhile, the proportion of small scale was twice more important for predefined templates (51%) then

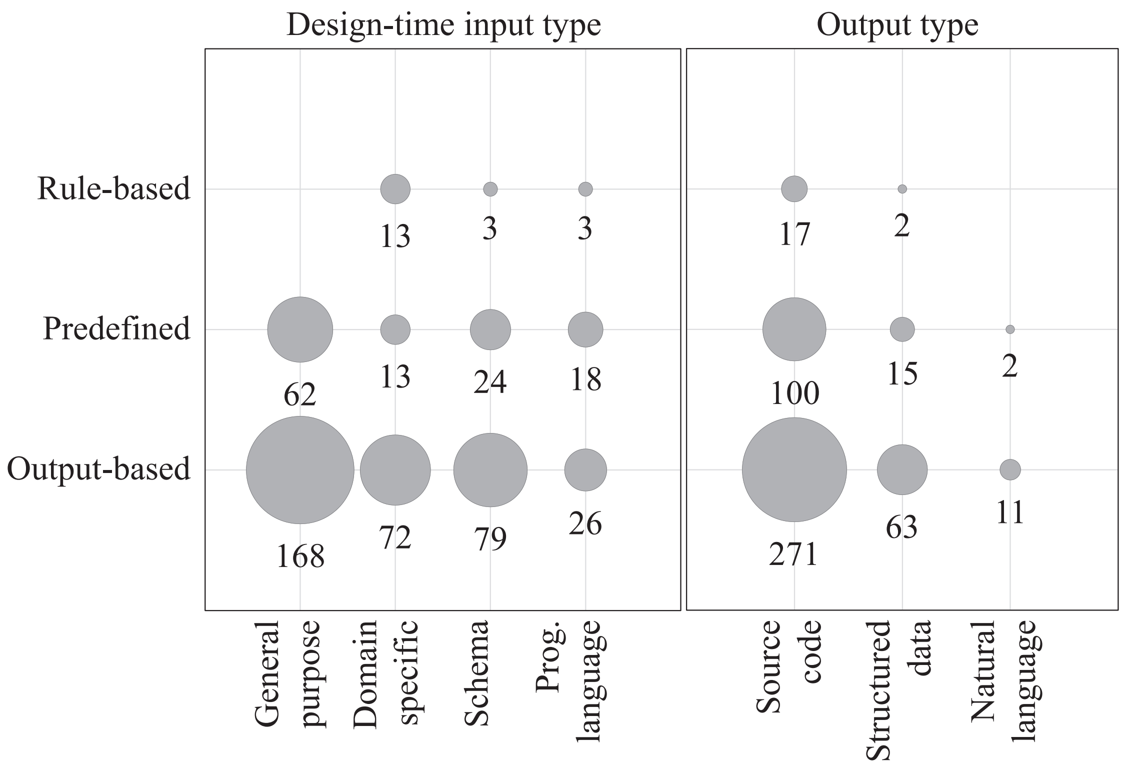

Figure 3.11: Relation between template style (vertical) and input/output types (horizontal)

for output-based (27%).

We found that popular tools were used twice more often on output-based templates (58%) than on predefined templates (23%). Rule-based templates never employed a tool that satisfied our popularity threshold, but used other tools such as Stratego.

We found that all papers using a rule-based style template do not follow an MDE approach. On the contrary, 70% of the output-based style papers and 56% of the predefined ones follow an MDE approach.

We noted that regardless of the template style, TBCG is used in an intermediate step or at the last step equally often.

Finally, we found that for each template style, the number of papers authored by an industry researcher fluctuated between 22–30%.

### 3.5.2.2 Input type

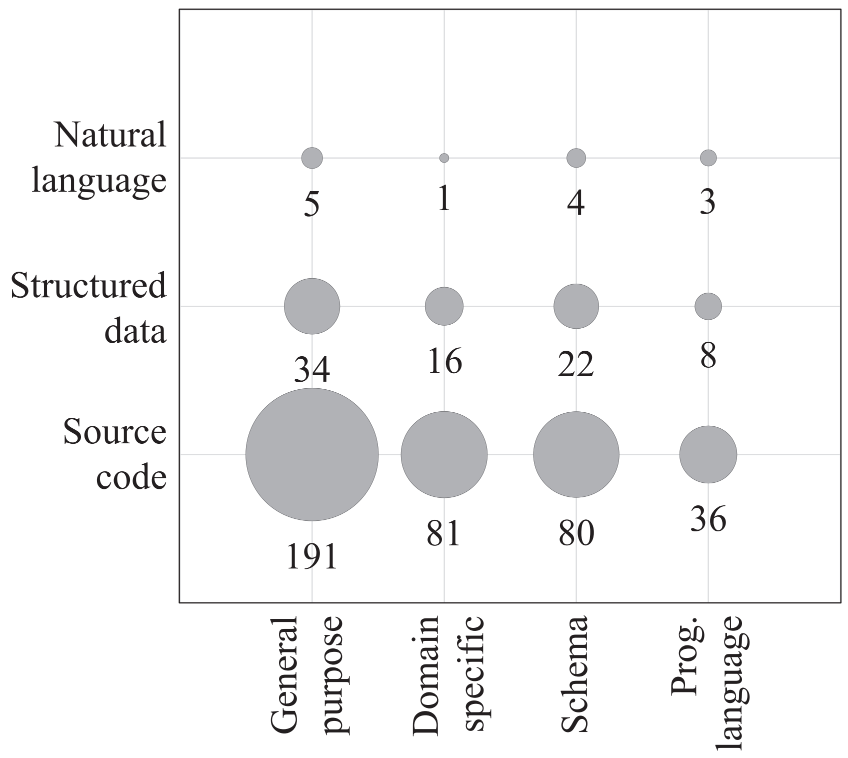

Figure 3.12: Relation between output (vertical) and design-time input (horizontal) types showing the number of papers in each intersection

The bubble chart in Figure 3.12 illustrates the tendencies between input and output types. It is clear that source code is the dominant generated artifact regardless of the input type. Source code is more often generated from general purpose and domain specific inputs than from schema and programming languages. Also, the largest portion of structured data is generated from a schema input. Finally, the most generated natural language text is when source code is provided as input.

Moving on to input type and application scale, we found that small scales are used 40% of the time when the input is a programming language. The number of papers with no sample application is very low (5%) regardless of the template style. Finally, 74%

of papers using large scale applications use a domain specific input, which is slightly higher than those using a general purpose input with 71%.

Next, when we compared input type to validation, we found that no paper using a DSL or a programming language used any formal method of validation. 22% of the papers using a DSL as input used a benchmark to validate their approach, which is higher than the 19% of the papers using general purpose languages. Also, we found that 77% of the papers using a general purpose language as input did not validate their approach.

#### 3.5.2.3 Output type

As we compared output type to orientation, we found that industrials generate slightly more source code than academics: 89% vs. 80%. However, academics generate more structured data and natural language than industrials: 18% vs. 6% and 3% vs. 1% respectively.

#### 3.5.2.4 Application scale

We found that 65% of the papers without application are from the academy. Between application scale and tools, we found that 74% of the papers that make use of a popular tool used large scale application to illustrate their approach. Also, 62% of the papers using unpopular tools[6] use large scale applications. Small scale is likely to be used in unpopular tools rather than popular tools.

## 3.6 Template-based Code Generation Tools

Figure 3.13 shows that half of the papers used a popular TBCG tool, whereas the other half used less popular tools (the other category), did not mention any TBCG tool,

6. Refers to the union of other and unspecified categories of the tool facet.

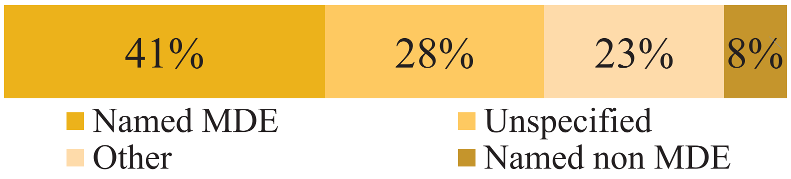

Figure 3.13: Tools categories

or implemented the code generation directly for the purpose of the paper. We also see that more than half of the popular tools do not follow MDE approaches.

### 3.6.1 Popular tools

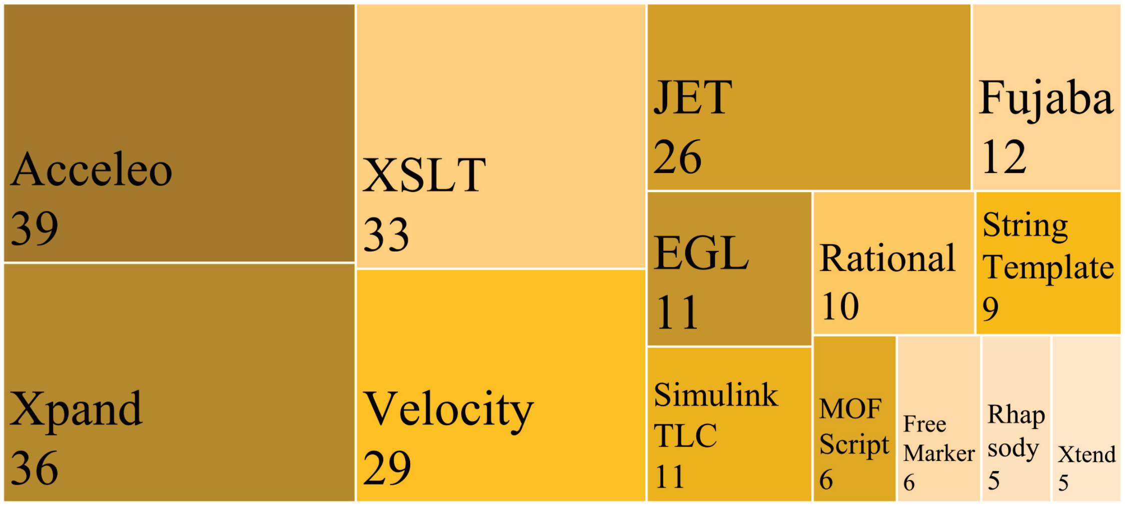

Figure 3.14: Popular tools

Figure 3.14 shows the distribution of popular tools used in at least 1% of the papers, i.e., five papers. Acceleo and Xpand are the most popular with respectively 16% and 15%of the papers using them. Their popularity is probably due to their simple syntax and ease of use [43] and the fact that they are MDE tools [51]. They both have an OCL-like language for the dynamic part and rely on a metamodel specified in Ecore as design-time input.

EGL also has a structure similar to the other model-based tools. It is natively integrated with languages from the Epsilon family, thus relies on the Epsilon Object Language for its dynamic part. MOFScript is another popular model-based tool that only differs in syntax from the others. Xtend2 is the least used popular model-based tool. It is both an advanced form of Xpand and a simplified syntactical version of Java.

XSLT is the third most popular tool used. It is suitable for XML documents only. Some use it for models represented in their XMI format, as it is the case in [3]. XSLT follows the template and filtering strategy. It matches each tag of the input document and applies the corresponding template.

JET [58] and Velocity [30] are used as often as each other on top of being quite similar. The main difference is that JET uses an underlying programming language (Java) for the dynamic part. In JET, templates are used to help developers generate a Java class that implements the code generation.

StringTemplate [4] has its own template structure. It can be embedded into a Java code where strings to be output are defined using templates. Note that all the tools mentioned above use an output-based template style.

The most popular CASE tools for TBCG are Fujaba [23], Rational [16], and Rhapsody [8]. One of the features they offer is to generate different target languages from individual UML elements. All CASE tools (even counting the other category) have been used in a total of 39 papers, which puts them at par with Xpand. CASE tools are mostly popular for design activities; code generation is only one of their many features. CASE tools have a predefined template style.

Simulink TLC is the only rule-based tool among the most popular ones. As a rule-based approach, it has a different structure compared to the above mentioned tools. Its main difference is that the developer writes the directives to be followed by Simulink in order to render the final C code from S-functions.

We notice that the most popular tools are evenly distributed between model-based tools (Acceleo, Xpand) and code-based tools (JET, XSLT). Surprisingly, XSLT, which has been around the longest, is less popular than Xpand. This is undoubtedly explained by the advantages that MDE has to offer [7, 49].

### 3.6.2 Unspecified and other tools

As depicted in Figure 3.13, 27% of the papers did not specify the tool that was used, as in [38] where the authors introduce the concept of a meta-framework to resolve issues involved in extending the life of applications. Furthermore, 24% of the papers used less popular tools, present in less than five papers, such as T4 [34] and Cheetah [63], a python powered template mainly used for web developing. Like JET, Cheetah templates generate Python classes, while T4 is integrated with .NET technology. Some CASE tools were also in this category, such as AndroMDA [66]. Other examples of less popular tools are Groovy template [36], Meta-Aspect-J [5], and Jinja2 [46]. The fact that new or less popular tools are still abundantly used suggests that research in TBCG is still active with new tools being developed or evolved.

### 3.6.3 Trends of tools used

Each one of these tools had a different evolution over the years. Unspecified tools were prevailing before 2004 and then kept a constant rate of usage until a drop since 2014. We notice a similar trend for CASE tools that were the most popular in 2005 before decreasing until 2009. They only appear in at most three papers per year after 2010. The use of the most popular tool, Xpand, gradually increased since 2005 to reach the peak in 2013 before decreasing. The other category maintained an increasing trend until 2014. Yet, a few other popular tools appeared later on. For example, EGL started appearing in 2008 and had its peak in 2013. Acceleo appeared a year later and was the most popular TBCG tool in 2013–2014. Finally, MOFScript had no more than a paper

per year since 2005. StringTemplate and T4 were used scarcely since 2006 and 2009 respectively.

### 3.6.4 Characteristics of tools

We have also analyzed each popular tool with respect to the characteristics presented in Section 3.4. As mentioned earlier, most of the popular tools implement output-based template technique except the CASE tools which are designed following the predefined style.

Tools such as Acceleo, Xpand, EGL, MOFScript and 97% of the CASE tools papers are only used based on an MDE approach, given that they were created by this community. Nevertheless, there are tools that were never used with MDE principles, like T4. Such tools can handle a program code or a schema as metamodel but have no internal support for modeling languages. Moreover, the programmer has to write his own stream reader to parse the input, but they allow for a broader range of artifacts as inputs that do not have to be modeled explicitly. A few code-based tools provide internal support for model-based approaches. For instance, Velocity, XSLT, and StringTemplate can handle both UML and programmed metamodel as design-time input.

A surprising result we found is that EGL is the only MDE tool that has its papers mostly published in MDE venues like SOSYM, MODELS, and ECMFA. All the other tools are mostly published in other venues like ICSSA, whereas software engineering venues, like ASE or ICSE, and MDE venues account for 26–33% of the papers for each of the rest of the MDE tools.

CASE tools, MOFScript, Velocity, and Simulink TLC mostly generate program code. The latter is always used in the domain of embedded systems. Papers that use StringTemplate do not include any validation process, so is Velocity in 93% of the papers using it. XSLT has been only used to generate structured data as anticipated.

Other tools are the most used TBCG in the industry. This is because the tool is often internal to the company [ID:208]. Among the most popular tools, Xpand is the most in the industry.

## 3.7 MDE and Template-based Code Generation

Overall, 64% of the publications followed MDE techniques and principles. For example in [83], the authors propose a simulation environment with an architecture that aims at integrating tools for modeling, simulation, analysis, and collaboration. As expected, most of the publications using output-based and predefined techniques are classified as model-based papers. The remaining 36% of the publications did not use MDE. This includes all papers that use a rule-based template style as reported in Section 3.5. For example, the authors in [20] developed a system that handles the implementation of dependable applications and offers a better certification process for the fault-tolerance mechanisms.

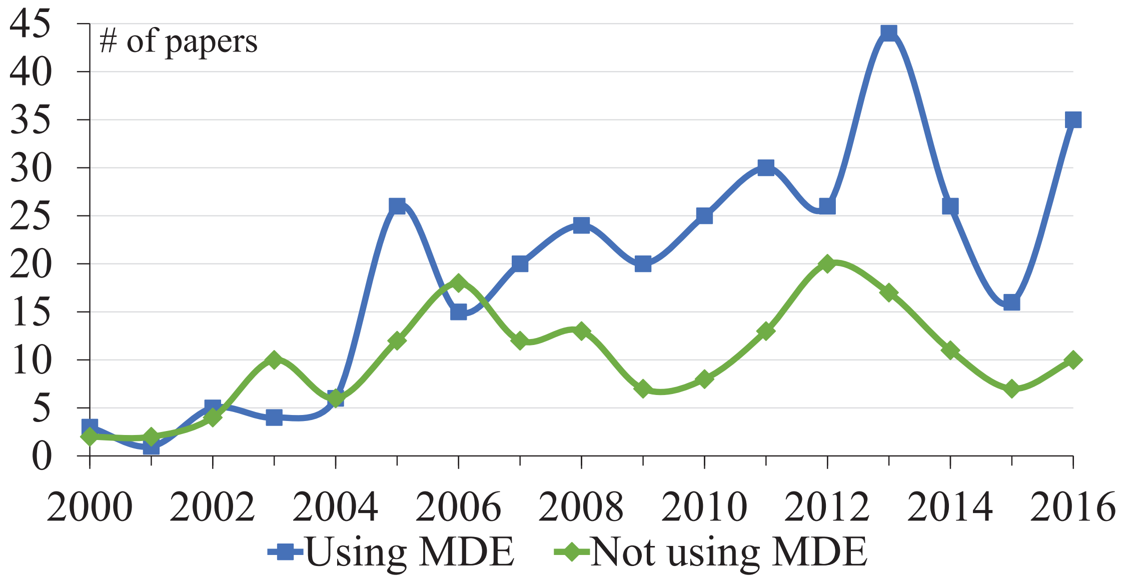

Figure 3.15: Evolution of the MDE facet

As Figure 3.15 shows, the evolution of the MDE category reveals that model-based approach started overpassing code-based techniques in 2005, except for 2006. It increased to reach a peak in 2013 and then started decreasing as the general trend of the corpus. Overall, model-based techniques for TBCG have been dominating other techniques in the past 12 years.

We also analyzed the classification of only MDE papers with respect to the characteristics presented in Section 3.1. We only focus here on facets with different results compared to the general trend of papers. We found that only half of the total number of papers using unspecified and other tools are model-based papers. We only found one paper that uses a programming language as design-time input with MDE [33]. This analysis also shows that the year 2005 clearly marked the shift from schema to domain-specific design-time inputs, as witnessed in Section 3.4.2. Thus after general purpose, which obtains 69% of the publications, domain specific accounts for a better score of 26%, while schema obtains only 4%. With respect to the run-time category, the use of domain-specific models increased to reach a peak in 2013. As expected, no program code is used for MDE papers, because MDE typically does not consider them as models, unless a metamodel of the programming language is used.

Interestingly, MDE venues are only the second most popular after other venues for MDE approaches. Finally, MDE journal papers maintained a linear increase over the years, while MDE conference papers had a heterogeneous evolution similar to the general trend of papers.

## 3.8 Discussion

### 3.8.1 RQ1: What are the trends in TBCG?

The statistical results from this significantly large sample of papers clearly suggest that TBCG has received sufficient attention from the research community. The community has maintained a production rate in-line with the last 11 years average, especially with a constant rate of appearance in journal articles. The only exceptions were a significant boost in 2013 and a dip in 2015. The lack of retention of papers appearing in non MDE may indicate that TBCG is now applied in development projects rather than being a critical research problem to solve. Also, conference papers as well as venues outside MDE and software engineering had a significant impact on the evolution of TBCG. Given that TBCG seems to have reached a steady publication rate since 2005, we can expect contributions from the research community to continue in that trend.

### 3.8.2 RQ2: What are the characteristics of TBCG approaches?

Our classification scheme constitutes the main source to answer this question. The results clearly indicate the preferences the research community has regarding TBCG. Output-based templates have always been the most popular style from the beginning. Nevertheless, there have been some attempts to propose other template styles, like the rule-based style, but they did not catch on. Because of its simplicity to use, the predefined style is probably still popular in practice, but it is less mentioned in research papers. TBCG has been used to synthesize a variety of application code or documents. As expected, the study shows that high-level language inputs have prevailed over any other type. Specifically for MDE approaches to TBCG, the input to transform is moving from general purpose to domain-specific models. Academic researchers have contributed most, as expected with a literature review, but we found that industry is actively and continuously using TBCG as well. The study also shows that the community is moving from large-scale applications to smaller-sized examples in research papers. This concurs with the level of maturity of this synthesis approach. The study confirms that the community uses TBCG to generate mainly source code. This trend is set to continue since the automation of computerized tasks is continuing to gain ground in all fields. Finally, TBCG has been implemented in many domains, software engineering and embedded systems being the most popular, but also unexpectedly in unrelated domains, such as bio-medicine and finance.

### 3.8.3 RQ3: To what extent are TBCG tools being used?

In this study, we discovered a total of 77 different tools for TBCG. Many studies implemented code generation with a custom-made tool that was never or seldom reused. This indicates that the development of new tools is still very active. MDE tools are the most popular. Since the research community has favored output-based template style, this has particularly influenced the tools implementation. This template style allows for more fine-grained customization of the synthesis logic which seems to be what users have favored. This particular aspect is also influencing the expansion of TBCG into industry. Well-known tools like Acceleo, Xpand and Velocity are moving from being simple research material to effective development resources in industry. Finally, the study shows that there has been a shift from CASE tools to output-based tools since 2005.

### 3.8.4 RQ4: What is the place of MDE in TBCG?

All this analysis clearly concludes that the advent of MDE has been driving TBCG research. In fact, MDE has led to increase the average number of publications by a factor of four. There are many advantages to code generation, such as reduced development effort, easier to write and understand domain/application concepts and less error-prone [7]. These are, in fact, the pillar principles of MDE and domain-specific modeling [52]. Thus, it is not surprising to see that many, though not exclusively, code generation tools came out from the MDE community. As TBCG became a commonplace in general, the research in this area is now mostly conducted by the MDE community. Furthermore, MDE has brought very popular tools that have encountered a great success, and they are also contributing to the expansion of TBCG across industry. It is important to mention that the MDE community publishes in specific venues like MODELS, SOSYM, or ECMFA unlike other research communities where the venues are very diversified. This resulted in three MDE venues at the top of the ranking.

## 3.9 Threats to validity

The results presented in this systematic mapping study have depended on many factors that could potentially limit the study.

### 3.9.1 Construction validity

Threats to construction validity deals with the problems related to the design of the research method and especially to identifying relevant primary studies.

In a strict sense, our findings are valid only for our sample that we collected from 2000–2016. This leads to determine whether the primary studies used in our survey are a good representation of the whole population. From Figure 3.2, we can observe that our sample can be attributed as a representative sample of the whole population. In particular, the average number of identified primary studies per year is 28 with a standard deviation of 15.76. A more systematic selection process would have been difficult to be exhaustive about TBCG. We selected three of the major online databases. These databases are complementary and we are confident that they index a maximum number of relevant publications. We chose to obtain the best possible coverage at the cost of duplications.

Another potential limitation is the search query. It is difficult to encode a query that is restrictive enough to discard unrelated publications but at the same time retrieves all the relevant ones. In order to obtain a satisfactory balance, we included synonyms and captured possible declinations. Our search query could suggest a restriction of the type of output. However, the size of the final corpus we classified is about ten times larger than other SMS related to code generation (see Section 2.5). We are therefore confident that the final corpus is a representative subset of all relevant publications on TBCG.

Finally, given that we obtained a sufficiently large final corpus for typical SMS, we did not perform snowballing which may have resulted in collecting additional papers omitted by the search engines.

#### 3.9.1.1 Internal validity

A potential limitation is related to data extraction. It is difficult to extract data from relevant publications, especially when the quality of the paper is low, when code generation is not the primary contribution of the paper, or when critical information for the classification is not directly available in the paper. For example in [62], the authors only mention the name of the tool used to generate the code. In order to mitigate this threat, we had to resort to searching for additional information about the tool: reading other publications that use the tool, traversing the website of the tool, installing the tool, or discussing with the tools experts.

Another possible threat is the screening of papers based on inclusion and exclusion criteria that we defined before the study was conducted. During this process, we examined only the title, the abstract. Therefore, there is a probability that we excluded relevant publications such as [22], that do not include any TBCG terms. In order to mitigate this threat, whenever we were unsure whether a publication should be excluded or not we conservatively opted to include it. However, during classification when reading the whole content of the paper, we may still have excluded it.

## 3.10 External validity

External threats to validity cope with problems that might arise during conclusion generalization. The results we obtained are based on TBCG only. Even though our classification scheme includes facets like validation, orientation, application domain, that are not related to the area, we followed a topic based classification. The core characteristics of our study are strictly related to this particular code synthesis technique. We have defined characteristics like template style and the two levels of inputs that we believe are exclusive to TBCG. Therefore, the results cannot be generalized to other code generation techniques mentioned in Section 2.3.

#### 3.10.0.2 Conclusion validity

Threats to conclusion validity (or reliability) deal with problems that might arise when deriving conclusions and whether the SMS can be repeated. Our study is based on a large number of primary studies. This helps us mitigate the potential threats related to the conclusions of our study. A missing paper or a wrongly classified paper would have a very low impact on the statistics compared to a smaller number of primary studies. In addition, as a senior reviewer did a sanity check on the rejected papers, we are confident that we did not miss a significant number of papers. Hence, the chances for wrong conclusions are small. Replication of this study can be achieved as we provided all the details of our research method in Section 3.1. Also, our study follows the methodology described in [69].

# CHAPTER 4

# COMPARISON OF THE EXPRESSIVENESS AND PERFORMANCE OF TBCG TOOLS

In this chapter, we compare the expressiveness power and performance of the nine most popular tools spanning the different technological approaches. We also evaluate the expressiveness based on common metamodel patterns and evaluate the performance on a range of models that conform to a metamodel composed by the combination of these patterns.

## 4.1 Metamodel patterns for template implementation

The degree of complexity of a template lies in the implementation of its dynamic part. Templates are defined based on the design-time input: the schema of the input or metamodel for model-based tools. To evaluate the expressiveness of TBCG tools, we identify *patterns* which are common structures found in metamodels that drive the implementation of the dynamic part of the template. We present each pattern in its simplest generalized form of occurrence. Note that their actual use in a metamodel may rely on one of its variants. Patterns can be combined arbitrarily to make up more complex structures. They are described in UML class diagram notation augmented with an annotation to identify variants.

To identify these patterns, we analyzed a plethora of metamodels that were used for a template-based model-to-text transformation. We investigated metamodel repositories, such as the metamodel zoo from AtlanMod [2] and ReMoDD [ReM], as well as known metamodel design patterns [25]. Additionally, we analyzed the systematic mapping study described in Chapter 3. We also based ourselves on industrial experiences for generating code for web applications and reporting from large metamodels of legacy code comprising nearly 1 000 elements each [78].

The following list of patterns is meant to be minimal, not complete. All template code snippets in this section are based on a common running example of invoice production, for which the metamodel is depicted in Figure 4.5.

### 4.1.1 Pattern 1: Navigation

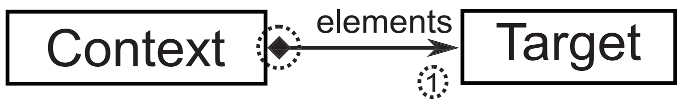

Figure 4.1: Navigation pattern

The simplest metamodel pattern is a navigable relation between two classes as depicted in Figure 4.1. From a template implementation point of view, this basic pattern is often used to access the data of a target class related to the class of the current `Context` class. For example, Listing 4.1 shows an XSLT template accessing the attributes of the class `Metadata` (lines 3–5) from the `Invoice` class (line 2).

```
1 <xsl:template match="Model">
2 <xsl:for-each select="invoice">
3 Transaction date: <xsl:value-of select="meta/@date"/>
4 Vendor name: <xsl:value-of select="meta/@cashier"/>
5 </xsl:for-each>
6 </xsl:template>
```

Listing 4.1: XSLT template to access metadata of an invoice

The different variants of this pattern are circled with dotted lines. One variant depends on the cardinality of the association end: either one or a list of target objects is accessible from the `Context` class. This influences the syntax of the access to the `Target` class. Some TBCG tools treat associations (references) and compositions (containments) differently. Hence another variant is based on the type of the relation. Composing this pattern with itself allows to access multiple target classes from the `Context` class.

### 4.1.2 Pattern 2: Variable dependency

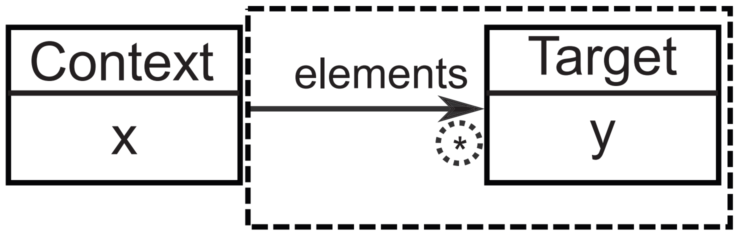

Figure 4.2: Variable dependency pattern

Considering the pattern in Figure 4.2, the implementation of a template often requires the developer to output a value that depends on variables present in other contexts. Here, the expressiveness of the dedicated language for the dynamic part would have an impact on the way the template is written. It is often needed to rely on external components such as other programming languages when the dedicated language for the dynamic part is limited. Listing 4.2 shows how calculations are implemented in Velocity. First, the global variables are initialized (lines 1–2). Then we calculate the `subtotal` value by adding the `price` value for each of the `PricedItem` class instances (line 7). The printed `total` price is calculated based on the `taxRate` (lines 9–10).

```
1  #set ($total = 0)
2  #set ($d = "$")
3  #foreach ($item in $invoice.items)
4  #set($total = $total + $item.price)
5  Name: $item.name
6  Type: $item.type
7  Price: $item.price $d #end
8  Tax: $invoice.taxRate %
9  #set ($total = $total * (1 + $invoices.taxRate / 100))
10 Total: $total $d
```

Listing 4.2: Velocity template to compute the total of the invoice

There are three variants of this pattern. One variant when the value depends on an attribute present in the same `Context` class. Another variant is when parts of the computation require data from one. The last variant is when these parts require more related classes.

The main difference between the first two patterns is that Pattern 4.1.2 is about the computation of values that depend on variables, whereas Pattern 4.1.1 is simply used to output the data of a class that is not in the current `Context` class.

### 4.1.3 Pattern 3: Polymorphism

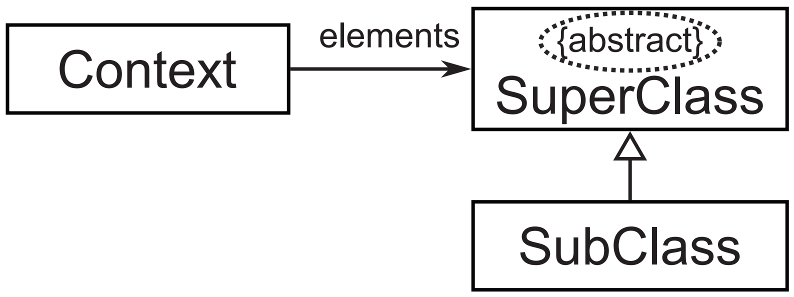

Figure 4.3: Polymorphism pattern

The presence of an inheritance relation between two classes in the metamodel is an attractive opportunity to reuse parts of a template and hence avoid code duplication. Given the pattern in Figure 4.3, the developer can implement the template for an output based on the super class and only implement what varies for the subclass. For example, Listing 4.3 shows an Acceleo template that implements polymorphism. The `generateItems` template is called (line 3) from the `Invoice` context (line 2). The template engine then executes the appropriate template based on the argument type (lines 7–14).

```
1 [template public generateInvoice(invoice : Invoice, model : Model)]
2 [for (item : Item | invoice.items)]
```

```
[generateItems(item)/]
[/for]
[/template]
[template public generateItems(elements : Item)]
Name:[elements.name/]
Type:[elements.type/]
[/template]
[template public generateItems(elements : PricedItem)]
Name:[elements.name/]
Type:[elements.type/]
Price:[elements.price/]$
[/template]
```

Listing 4.3: Acceleo template for polymorphism

A variant of this pattern is when the super class is abstract: some tools do not allow to define a template in the context of an abstract type.

### 4.1.4 Pattern 4: Recursion

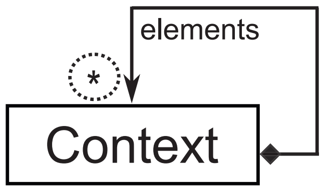

Figure 4.4: Recursion pattern

This pattern consists of a recursive relation of a class. From the usability point of view, the developer can reapply the template on objects of the same type in a transparent way. It requires the template definition block to be encapsulated within the `Context` class. The source domain describes the way in which a language allows the specification of the elements of the source model(s) on which a template will be executed [74]. It will have an impact on how the developer writes the template.

Listing 4.4 shows an example of recursion in Xtend2. The `getCategoryLevel` definition block is called for the first time (line 2) to find the appropriate level of the category of the invoice. Lines 11–13 depth-first searches recursively the corresponding category among the descendants of the category reference.

```
 1 def generateInvoice(Invoice invoice, Model model){
 2 '''
 3 «getCategoryLevel(model.rootCat, invoice.category, 0, sum, invoice.taxRate)»
 4 '''
 5 }
 6 def getCategory(Category catRef, Category cat, int index, long total, int taxRate){
 7 '''
 8 «IF cat.name.equals(catRef.name)»
 9 «IF index > 30»
10 Discount: 15%
11 Tax: «taxRate»%
12 Total: «total * (1 + taxRate / 100)»$
13 «ELSE»
14 Discount: 10%
15 Tax: «taxRate»%
16 Total: «total * (1 + taxRate / 100)»$
17 «ENDIF»
18 «ENDIF»
19 «FOR subCat : catRef.subs»
20 «generateCategory(subCat, cat, index+1, total, taxRate)»
21 «ENDFOR»
22 '''
23 }
```

Listing 4.4: Xtend2 code for recursion

The only variant of this pattern is the cardinality of the association end which influences the traversal strategy (depth-first vs. breadth-first).

### 4.1.5 Combination of patterns

All the metamodels we analyzed (c.f. Section 4.1) exhibit combinations of instances of these four patterns.. For example, the polymorphism pattern can be combined with the navigation pattern to form a metamodel containing a super class that contains a subclass that is linked to another class through a navigable relation. To compare the different tools on the same example, we created a minimal class diagram that combines all four patterns, as depicted in Figure 4.5. This will be the design-time input we will use to specify the code generation templates.

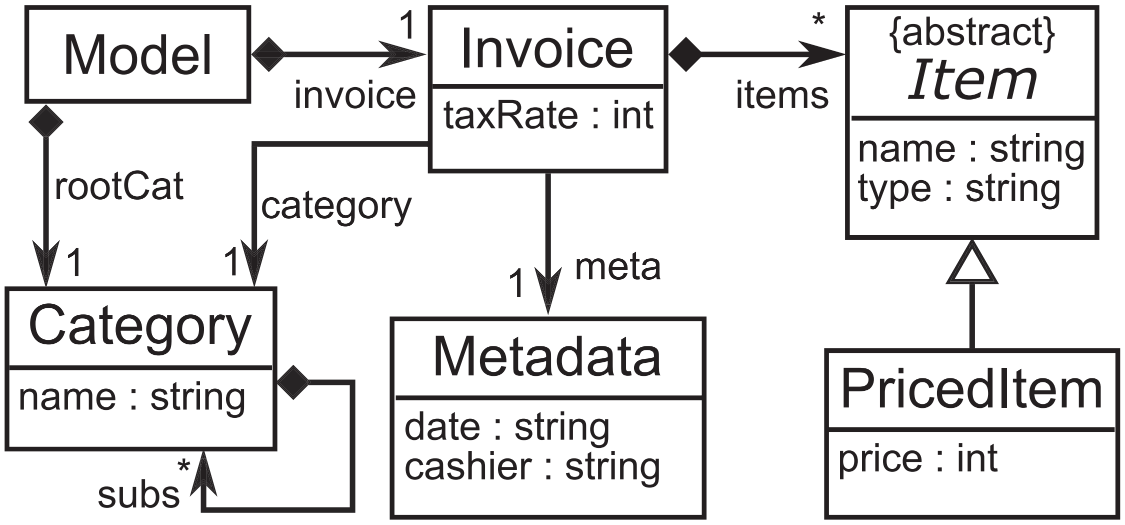

Figure 4.5: Invoice metamodel

This metamodel describes the production of invoices in a given store. The invoice shows the purchased items and their prices which are printed within the context of the `Invoice` class instance. The total price is calculated by adding the value of the attribute `price` of each `PricedItem` class instance. In addition, the total price depends on the `taxRate` and the discount rate. The discount rate is calculated depending on the depth level of the `Category` class instance whereas the `taxRate` value is fixed. The total price is printed along with all the other computed values in the output. In this running example, all templates produce an output similar to the sample shown in Listing 4.5 rendering an invoice with three priced items and a second level discount category.

```
Store: FooBar
Cashier: Alice
Transaction date: Sun Aug 13 00:00:00 EDT 2017
********************************
Name: Shoes
Type: Clothing and Shoes
Price: 100$
Name: Laptop
Type: Electronics
Price: 500$
Name: Pizza
Type: Food
Price: 20$
********************************
Category: Student
Discount: 10%
Tax: 15%
Total: 641.17$
```

Listing 4.5: Sample output

## 4.2 Tools expressiveness

In this paper, we compare the nine most popular template-based code generation tools for which the template follows the style of the output. The comparison of the tools is based on a qualitative evaluation of how the developer must implement the template corresponding to each pattern.

### 4.2.1 Code generation tools

We classify the tools into two groups: those that are model-based and those that are code-based. Table 4.I synthesizes the main characteristics of each tool. The dedicated language indicates the language used to express the dynamic part of the template. The input type is the type of design-time input accepted by the template. The source domain is defined in Section 4.1.3. The execution mode of the template refers to the different ways to execute the template file to produce the output. The last two columns of the table influence the implementation strategy of templates: whether they are based on block definition per type and whether custom functions internally defined within a template can be invoked from templates.

#### 4.2.1.1 Model-based tools

Model-based tools rely on a metamodel as design-time input and on current mainstream MDE technologies. For example, all those listed rely on the Eclipse Modeling Framework [81], where models are specified in Ecore. The source domain of the templates is therefore a metamodel element (an EClass). The template of all model-based tools is executed through an interpreter that creates a file, writes in it the output, and evaluates the dynamic part of the template.

**4.2.1.1.1 Acceleo** is a pragmatic implementation of the Object Management Group (OMG) MOF Model to Text Language (MTL) standard [68]. An Acceleo file consists of a set of typed definition blocks that are sections of the template where the developer specifies how each element type visited in the model shall be rendered, as depicted in Listing 4.3. In a sense, this follows a similar strategy to how ATL transformation rules are specified (one per type defined in the metamodel) [50]. Dynamic parts are expressed in the MOFM2T language, an extension of OCL with imperative expressions. Acceleo does not support function definition, but allows for the definition of encapsulated OCL queries.

| **Tools** | **Dedicated language** | **Input type** | **Source domain** | **Execution mode** | **Typed block def.** | **Func.** |
|---|---|---|---|---|---|---|
| Acceleo | MOFM2T extended OCL | Ecore model | Metamodel element | Interpreted | Yes | No |
| Xpand | Limited OCL | Ecore model | Metamodel element | Interpreted | Yes | No |
| EGL | EOL | Ecore model | Metamodel element | Interpreted | No | Yes |
| Xtend2 | Abstraction of Java | Ecore model | Metamodel element | Interpreted or instant gen. | No | Yes |
| JET | Java | Java object | Root of the data structure | Run Java output | No | No |
| Velocity | Java-based scripting language | Java object | Any Java variable | Interpreted | No | No |
| T4 | C# | C# object | Any C# variable | Instant gen. or Run C# output | No | Yes |
| StringTemplate | Limited scripting language | Java object | Any Java variable | Interpreted | No | No |
| XSLT | XSLT+XPath expressions | XML document | XML element | Interpreted | Yes | Yes |

Table 4.I: Classification of TBCG tools.

**4.2.1.1.2 Xpand** also relies on the element type to organize the typed definition blocks. The dynamic language is less powerful than in Acceleo, as it relies on a limited subset of OCL, also lacking support for queries and modules. Listing 4.12 shows an example of the syntax of Xpand. Although functions cannot be defined within Xpand templates, it is possible to call function defined in Xtend extensions or to custom Java code.

**4.2.1.1.3 Xtend2** is a complete programming language. It relies on a DSL that is an abstraction of Java extended with lambda expressions and templates that are ultimately compiled to Java code. For TBCG purposes, Xpand dynamic blocks can be integrated, as illustrated in Listing 4.4. Instead of typed definition blocks, the template is organized in functions. A part from the interpreted mode of execution, an Xtend2 template can also generate the output instantly as it is saved when editing in a domain-specific modeling environment (run-time Eclipse instance).

**4.2.1.1.4 EGL** belongs to the family of Epsilon languages and thus relies on Epsilon Object Language (EOL) for the dynamic part of the templates. Unlike the other model-based tools, an EGL template is organized exactly how the output will appear, the dynamic part is specified in macros written in EOL. Nevertheless, functions can be defined and invoked within the template.

### 4.2.1.2 Code based tools

Unlike model-based tools, the structure of the code-based tools consists of static and dynamic parts that are mixed together without the concept of template definition block (except for XSLT). Furthermore, the dynamic part is expressed in the underlying programming language, e.g., manipulating the Java objects for each source domain.

**4.2.1.2.1 Velocity** is a Java-based template engine. It permits anyone to use a simple yet powerful template language to reference objects defined in Java code [6]. The dedicated language for the dynamic part consists of a scripting language that supports Java expressions. It requires Java objects as the input and unlike model-based tools the reference to these Java objects need not be an element type contained in the metamodel, i.e., the template can handle any Java variable as the source domain. Velocity templates do not support functions calls internal to the template, but to external Java code. It is interpreted in order to generate the output.

**4.2.1.2.2 JET** templates are used to generate the Java class responsible for printing the desired output. This code is automatically generated when the template is saved. To generate the final output, the developer needs to run this Java code. Dynamic parts are written in Java code, as they appear almost as is in the generated code. JET requires a single Java objects as the input that contains all the required input data as the source domain. Listing 4.6 shows an example of a JET template that computes the total of the invoice where the input is the `argument` variable. Variables are first initialized (lines 1–4) before starting the loop (lines 5–11) that iterates over each instance of the `Item` class to get the attribute values (lines 8–10) and calculate the subtotal of the invoice (line 11). The total price is calculated based on the `taxRate` and the subtotal (line 13). Note that attributes are accessed using *getter* functions because the design-time input used in Listing 4.6 is the Java code generated from the Ecore model of an invoice.

```
1 <% Model model = (Model) argument; %>
2 <% int subtotal = 0; %>
3 <% Invoice invoice = model.getInvoice(); %>
4 <% EList<Item> items = invoice.getItems();
5 for(Iterator<Item> iterator = items.iterator(); iterator.hasNext();) {
6 Item item = iterator.next(); %>
7 Name: <%= item.getName() %>
8 Type: <%= item.getType() %>
Price: <%= ((PricedItem)item).getPrice() %>
<% subtotal += ((PricedItem)item).getPrice(); %>
<%}%>
Tax: <%= invoice.getTaxRate() %>%
Total: <%= subtotal * (1 + invoice.getTaxRate() / 100) %>$
```

Listing 4.6: JET template to compute the total of the invoice

**4.2.1.2.3 T4** templates consists of the static output text where the dynamic part is written as a C# program. The input required is a C# object and like the previous tools, the reference to these objects can be of any type. T4 has two modes of execution. Like in Xtend2, it can generate the output instantly as the developer is editing the template . Also, like in JET, it automatically generates C# code that shall be executed by the developer to produce the output. Listing 4.7 shows an example of a T4 template that computes the total of the invoice. Here, the input is a C# reference to the invoice model serialized in XMI. Functions can be defined and invoked in the template as it is done in a standard C# class.

```
<# foreach (XElement item in invoice.Elements("items")){
string name = item.Attribute(XName.Get("name")).Value;
string type = item.Attribute(XName.Get("type")).Value;
string price = item.Attribute(XName.Get("price")).Value;
subtotal += long.Parse(price); #>
Name: <#= name #>
Type: <#= type #>
Price: <#= price #>$
<# } #>
Tax: <# invoice.Attribute("taxRate") #>%
Total: <#= total = subtotal * (1 + long.Parse(invoice.Attribute("taxRate")) / 100) #>$
```

Listing 4.7: Excerpt of T4 template

**4.2.1.2.4 StringTemplate** is dedicated to performing string replacements on a textual input. Several programming languages (e.g., Java, Python, JavaScript) have implemented it. The dedicated language for the dynamic part is a very limited scripting language that is programming language neutral. It mainly consists of the dot operator as shown in Listing 4.8. StringTemplate requires a variable as input and, in our case, the a Java object, since we relied on its Java implementation. To execute a template, the developer needs to execute the enclosing Java program and the interpreter makes the appropriate string replacement in the template.

**4.2.1.2.5 XSLT** is a powerful template-based engine to transform XML documents. Being described in XML itself, it consists of a set of blocks that specify how each tag element is rendered, such as the one in Listing 4.1. The dedicated language for the dynamic part consists of large choice of powerful XSLT and XPath expressions. The XSLT engine interprets the template file to generate the output by applying the template corresponding to the XML element to match (e.g., `Model` on line 1).

### 4.2.2 Pattern-based comparison of tools

We compare the expressiveness of these TBCG tools based on the patterns introduced in Section 4.1.

#### 4.2.2.1 Navigation pattern

All tools successfully implement this trivial pattern, mostly through the use of the dot operator, as shown in Listing 4.8. In XSLT, this is accomplished with the `xsl:value-of` expression, as depicted in Listing 4.1.

```
Transaction date: <invoice.meta.date>
Vendor name: <invoice.meta.vendorName>
```

Listing 4.8: StringTemplate code to access data from the target class

However, XSLT requires a different strategy when in the presence of an association relation (not a composition). Listing 4.9 shows how the name of the `Category` associated to an `Invoice` is accessed. Due to the way Ecore is serialized in XMI, the developer needs to manually compare (line 5–9) the `id` contained in the `Invoice` class (line 3) to those of the target classes (line 4). It is only when the `id` matches that the corresponding `name` is printed.

```
1  <xsl:template match="Model">
2  <xsl:for-each select="invoice">
3  <xsl:variable name="invoice_cat_id" select="@category"/>
4  <xsl:for-each select="../rootCat">
5  <xsl:choose>
6  <xsl:when test="attribute::id = $invoice_cat_id">
7  Category: <xsl:value-of select="attribute::name"/>
8  </xsl:when> </xsl:choose>
9  </xsl:for-each>
10 </xsl:template>
```

Listing 4.9: XSLT code to access data through an association

### 4.2.2.2 Variable dependency pattern

We implemented this template to output and calculate the total of the invoice. As depicted in Listing 4.10, this is possible in Acceleo thanks to the powerful OCL-like dedicated language that provides built-in mathematical functions, such as *sum()* for the collection type. XSLT provides even more built-in mathematical functions to the developer for the dynamic part.

```
[template public generateInvoice(invoice : Invoice, model : Model)]
[for (item : Item | invoice.items)]
[generateItems(item)/][/for]
Tax: [invoice.taxRate/]
Total: [invoice.items.oclAsType(PricedItem).price->sum()*(1 invoice.taxRate/100)/] $
[/template]
```

Listing 4.10: Acceleo code to compute the total of invoice

To implement this pattern, EGL, JET, Velocity, T4, and Xtend2 rely on the use of global variables and statement blocks, as depicted in Listings 4.2, 4.6, and 4.7. A statement block is a statement in the dynamic part of the template that is not printed, e.g., for-loop, if-statement or variable assignment. Note that in Xtend2, the result of the statement blocks is also printed in the output.

```
/*** In Xtend ***/
import InvoiceMM;
Void addPrice(PricedItem pi):
JAVA template.Utils.getSubTotal(InvoiceMM.PricedItem);

/*** In Java ***/
package template;
import InvoiceMM.PricedItem;
public class Utils {
public static int subtotal = 0;
public static void getSubTotal(PricedItem item) {
subtotal += item.getPrice();
} }
```

Listing 4.11: Xtend and Java extension to calculate the invoice subtotal

It was not possible to implement this pattern with StringTemplate and Xpand *natively*. Even though Xpand provides an OCL-like language for the dynamic part, it does not provide any built-in mathematical function like Acceleo. We had to extend the template with a Java program to handle the calculations. Listing 4.12 shows the invocation of the Xtend method `addPrice()` (line 13), which in turn invokes the appropriate Java method `getSubTotal()` presented in Listing 4.11.

StringTemplate does not allow for assignments, hence such calculations cannot be achieved. Note that it is possible to perform a simple numerical operation such as “a+b”. However, without assignment, it is not possible to cumulate the subtotal over every `PricedItem`. All functions available are dedicated to string manipulation. Therefore, we had to first perform a model-to-model transformation to reduce the complexity of the template and perform the calculations in the Java input. Then, we only pass the computed values to the template to correctly output the invoice.

#### 4.2.2.3 Polymorphism pattern

We noted that the main difference between tools is highlighted in the variant when an abstract superclass is present. In Acceleo, Xpand, and Xtend2, it is mandatory to write a template block for the super class even though its content is not printed in the output. Listing 4.12 shows that the definition of the template block for `Item` is required, even though only the one for `PricedItem` will be executed.

```
«DEFINE invoice FOR Invoice»
«EXPAND item FOREACH items»
«ENDDEFINE»
«DEFINE item FOR Item»
//This is for the super abstract class: content is not printed
Nom: «name»
Type: «type»
«ENDDEFINE»
«DEFINE item FOR PricedItem»
Name: «name»
Type: «type»
Price: «price»
«addPrice()»
«ENDDEFINE»
```

Listing 4.12: Xpand code for polymorphism

In contrast, in EGL the content of the superclass template definition block is output, along with the content of the one for the subclass. Listing 4.13 shows that the developer can write the common behavior inside the abstract template definition block and the content specific to the subclasses in a different blocks, thus favoring reuse.

```
[% for (invoice in Invoice) { %]
[% for (item in Item) { %]
Name: [%= item.name %]
Type: [%= item.type %]
[% } %]
[% for (item in PricedItem) { %]
Price: [%= item.price %]
[% } %]
[% } %]
```

Listing 4.13: EGL template for polymorphism

In JET, Velocity, T4 and StringTemplate, no template code can be defined for abstract classes. Thus, the developer must replicate the common template code for all possible subclasses, unless it is encapsulated in a function. In XSLT, the developer is also restricted to use the data of the subclasses, given that tags correspond to concrete elements in the XMI document.

#### 4.2.2.4 Recursion pattern

We implemented this pattern to obtain the depth level of the invoice `Category` from the hierarchy of categories present in the model. We were only able to implement it in EGL, Acceleo, Xtend2, and T4 thanks to the use of function or typed definition block. Listing 4.14 shows an example of how recursion can be implemented in EGL using the function feature.

```
[%for (modl in Model) { %]
[%for (invoice in Invoice) { %]
[% invoice.category.getCategory(modl.rootCat, invoice.category, 0, total, invoice.
    taxRate); %]
[%}%]
[%}%]
[% operation Category getCategory(catRef:Category, cat:Category, index:Integer, total:
    Integer, taxRate:Integer):Integer{%]
[% if (cat.name = catRef.name) { %]
[% if (index > 30) { %]
Discount: 15%
[% total = total * (1 - 15 / 100); %]
Tax: [%=taxRate%]%
[% total = total * (1 + taxRate / 100); %]
Total: [%= total %]$
[% } %]
[% else { %]
Discount: 10%
[% total = total * (1 - 10 / 100); %]
Tax: [%=taxRate%]%
[% total = total * (1 + taxRate / 100); %]
Total: [%= total %]$
[% } %]
[% } %]
[% for (subCat in catRef.subs) { %]
[% subCat.getCategory(subCat, cat, index+1); %]
[% } %]
[% } %]
```

Listing 4.14: EGL code for recursion

Listing 4.15 shows an example of recursion using the typed definition block feature in Acceleo.

```
[template public generateInvoice(invoice:Invoice, model:Model)]
[for (cat : Category | invoice.category)]
[getCategory(model.rootCat,invoice.category,0,invoice.items.oclAsType(Item).price ->
    sum(),invoice.taxRate)/]
[/for]
[/template]
[template public getCategory(catRef:Category, cat:Category,index:Integer,total:Integer,
    taxRate:Integer)]
[if (cat.name = catRef.name)]
[if (index > 30)]
Discount: 15%
Tax:[taxRate/]%
Total:[(total * (1 - 15 / 100)) * (1 + taxRate / 100)/]$
[else]
Discount: 10%
Tax:[taxRate/]%
Total: [(total * (1 - 10 / 100)) * (1 + taxRate / 100)/]$
[/if]
[/if]
```

```
[for (subCat : Category | cat.subs)]
[getCategory(subCat, cat, index+1, total,taxRate)/]
[/for]
[/template]
```

Listing 4.15: Acceleo code for recursion

Listing 4.4 shows an example of recursion using the typed definition block in Xtend2. As for T4, the dedicated language allows to call C# functions defined in the template and thus implement recursion.

Although Xpand supports typed definition blocks, they only take a single argument which is a type of element in the input metamodel. Thus it is not possible to implement this pattern.

XSLT does not implement this pattern either. Although it supports the definition of functions, there is no trace between the argument that is passed to the function and the variable passed in the initial invocation. Therefore, the developer can modify the value inside the function, but cannot manipulate the input data directly from the function.

It is not possible to implement this pattern in JET, StringTemplate, and Velocity due to the absence of typed definition block or function.

| **Tool** | **Pattern 1** | **Pattern 2** | **Pattern 3** | **Pattern 4** |
|---|---|---|---|---|
| Acceleo | Complete | Complete | Complete | Complete |
| Xpand | Complete | Incomplete | Complete | Incomplete |
| EGL | Complete | Complete | Complete | Complete |
| Xtend2 | Complete | Complete | Complete | Complete |
| JET | Complete | Complete | Complete | Incomplete |
| Velocity | Complete | Complete | Complete | Incomplete |
| T4 | Complete | Complete | Complete | Complete |
| StringTemplate | Complete | Incomplete | Complete | Incomplete |
| XSLT | Complete | Complete | Complete | Incomplete |

Table 4.II: Summary of the qualitative evaluation of the tools expressiveness.

Table 4.II summarizes the qualitative evaluation of the tools expressiveness, showing whether a tool successfully implemented the pattern or not.

## 4.3 Tools performance

We conducted an experiment to compare the performance of TBCG tools.

### 4.3.1 Experiment setup

For this experiment, we consider set of five scales of model sizes conforming to the metamodel in Figure 4.5. For each scale we generated 10 input models from the metamodel using the tool in [9]. Table 4.III shows the characteristics of the input models averages per scale. The model size corresponds to the number of class instances in each model. The second column indicates the depth level of the invoice category, effectively counting the number of recursive calls: the value returned by, for example, Listing 4.14. The following columns indicate the number of instances of each pattern in the input model. The navigation pattern appears in the relations `Model` to `Invoice`, `Invoice` to `Metadata` and `Invoice` to `Category`. The number of variable dependency pattern instances is the number of `PricedItem` objects plus one for accessing the invoice `taxRate`. The number of occurrences of the polymorphism pattern counts the number of *PricedItem* objects. Finally, the number of occurrences of the recursion pattern is the number of `Category` objects less the root category.

Since the models are synthesized in Ecore, we had to convert them into a format suitable for code-based tools. We therefore generated the models to Java objects using Acceleo for Velocity, JET, and StringTemplate. We used the XMI version of Ecore as runtime input for XSLT. We also opted for this solution for T4, instead of generating C# objects directly. Otherwise this would have required to translate the Java classes of the metamodel automatically generated by EMF into C#. Nevertheless, the runtime

| **Model size** | **Category index** | **Pattern 1 instances** | **Pattern 2 instances** | **Pattern 3 instances** | **Pattern 4 instances** |
|---:|---:|---:|---:|---:|---:|
| 10 | 2 | 3 | 7 | 6 | 1 |
| 100 | 1 | 3 | 92 | 91 | 6 |
| 1 000 | 18 | 3 | 975 | 974 | 24 |
| 10 000 | 49 | 3 | 9 919 | 9 918 | 70 |
| 100 000 | 77 | 3 | 98 977 | 98 976 | 86 |

Table 4.III: Characteristics of the input models

input given to T4 remains C# objects encapsulating the XMI document. Note that we executed Xtend2 in its interpreted mode and T4 to output C# code that is then executed.

We automated the execution process for each tool with a Java program that loads the required input models and runs the template engines. We launched the performance tests on a 64-bit Windows PC equipped with 16GB of RAM and an Intel Dual Core I7-2600 CPU that clocks at 3.40GHz.

### 4.3.2 Data collection

We implemented the template of each TBCG tool such that the output is identical across the tools for the same input model, as in the sample in Listing 4.5. We ran each experiment 10 times. We collected the execution times of the template engine excluding the time to generate the output file since writing to disk is machine dependent. We discarded outlier values for the warm-up rounds and reported the averages, given that the standard deviation was negligible.

As discussed in Section 4.2.2.4, it is not possible to implement the recursion pattern in Xpand, XSLT, JET, Velocity, and StringTemplate, unless we resort to implement the recursion outside the template in a Java program, but that would defeat the purpose of the comparison. Nevertheless for Pattern 2, we relied on such extensions for Xpand only to perform the calculation of the total as in Listing 4.11. Note that looping over each `PricedItem` is still done in the template as in Listing 4.12. To ensure fairness among tools, we performed two distinct experiments. In experiment A, the discount rate is 15 if the invoice category is the root category, otherwise it is 10. In experiment B, we determine the discount rate based on the depth level of the invoice category. The discount rate is 15 if the depth level of the invoice category is more than 30, otherwise it is 10. Therefore only Acceleo, EGL, Xtend2, and T4 are considered in B, whereas all tools are considered in A.

### 4.3.3 Results

### 4.3.4 Performance without recursion

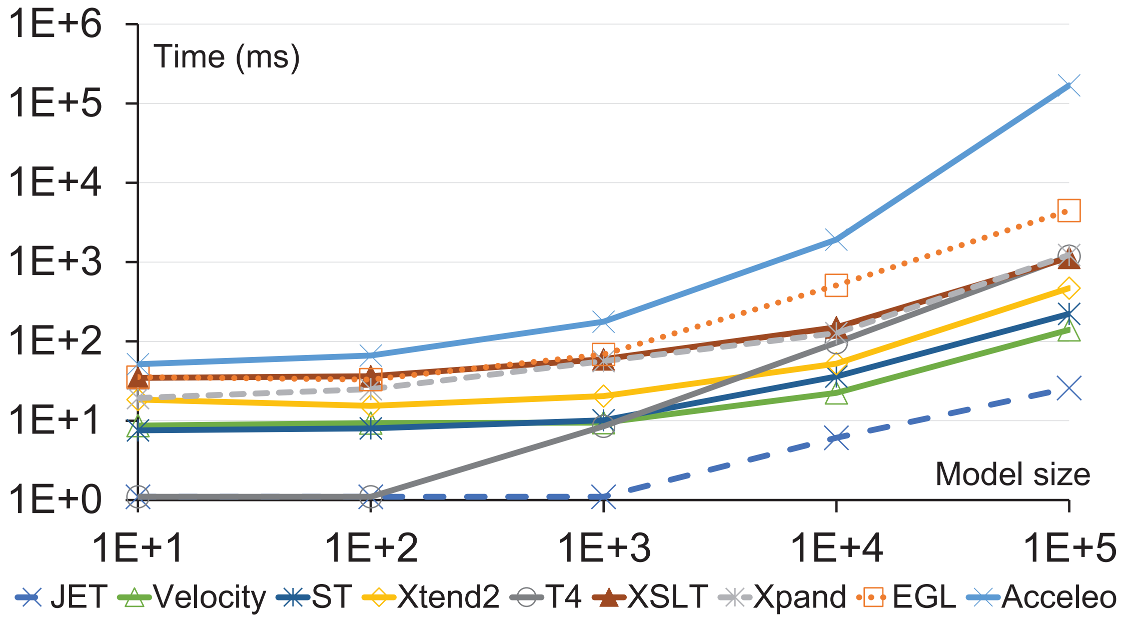

Figure 4.6: Tool performance for experiment A in log scale

As Figure 4.6 shows, the execution time increases with the size of the model for all tools. In the following discussion, the numbers refer to the cumulative time to run experiment A on all models for each tool.

Overall, JET is the fastest tool, completing the whole experiment in just 33ms. This is expected since JET generates instantly the corresponding Java class from the template as the developer is writing the template. Therefore, the execution time here corresponds to executing the generated Java code that produces the output. Excluding the special case of JET, the template engine of Velocity is the fastest with a total of 190ms. It is followed by StringTemplate that completes the experiment in 284ms. It reaches a high performance because templates in StringTemplate are very simple, the essential part of the computation is performed in the enclosing Java program.

T4 is as efficient as JET for smaller models with fewer than 1 000 objects, as they are both executed in the same way. However, for larger models, it becomes slower than the other two tools. Xtend2 performs better overall in only 577ms, which is faster than T4, making it the fastest model-based tool. Sharing the same underlying architecture, Xpand comes next with 1.4s. It is followed very closely by XSLT.

The slowest tools in this experiment are EGL followed by Acceleo, taking respectively 5s and 171s to complete the test. Note that for smaller models with at most 1 000 objects, EGL performs almost as fast as XSLT, but Acceleo is twice as fast. However, their time increases exponentially for larger models.

Velocity templates execution scales remarkably well by only a factor of 15 for models with $10^5$ elements compared to smaller models with $10^3$ elements. It is followed by JET, Xtend2, StringTemplate, and XSLT with around a factor of 25. For the remaining tools, the size of the model has a significant effect on their performance which is worsened by a factor of 45–108. T4 and Acceleo have the worst scale factor with 140 and 955 respectively.

### 4.3.5 Performance with recursion

The results for experiment B, reported in Figure 4.7, show a similar trend for the four tools concerned. The recursion pattern did not influence significantly the performance of Acceleo and T4 (slower by 1%). However, Xtend2 performed 10% slower than for experiment A. This is expected since, for Pattern 4, we only verify the equality of the name of the invoice category with the root category for experiment A, while that number is determined by the category level for experiment B (second column of Table 4.III). EGL performed 10% faster than for experiment A. This is the dedicated language EOL supports the caching feature. The function is only executed once for each distinct Integer and subsequent calls on the same target return the cached result [57].

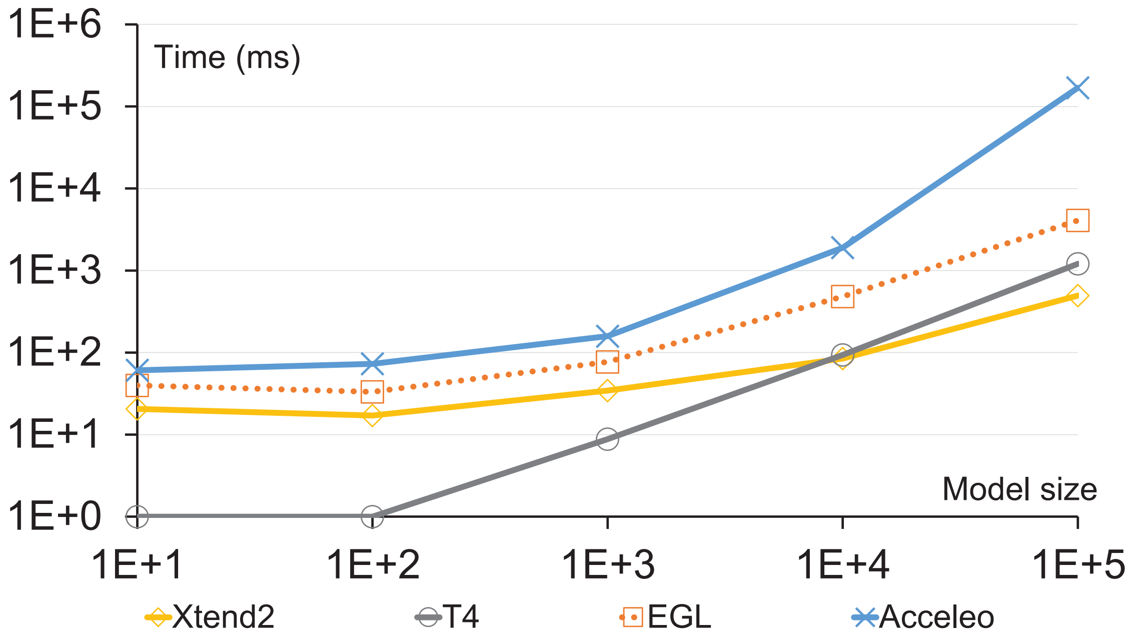

Figure 4.7: Tool performance for experiment B in log scale

### 4.3.6 Discussion

From the experiments we conducted, we observe that the fastest tools are code-based, especially for bigger models with more than 1 000 objects. Their dynamic language is based on the host programming language (Java/C#) which explains their impressive performance. JET is the best tool performance-wise. It generates the desired output almost instantly irrespective of the input model. In our experiments, T4 is the only exception since, for larger models, the XML navigation becomes an overhead compared to having encoded model manipulations directly in C#. Xtend2 is the fastest model-based tool and is even faster than XSLT. It performed on average 16 times slower compared to JET.

In contrast, model-based tools are best suited for complex input data manipulations and provide adequate support for non-trivial patterns, like recursion, within the template. In particular, Xtend2 appears to be the most capable tool since it contains the necessary features to successfully implement all the metamodel patterns in times similar to code-based tools. Furthermore, a useful attribute of model-based tools is the level of abstraction of the dynamic part: a modeling paradigm for model-based tools and a programming paradigm for code-based tools. Consequently, the logic is spread across different artifacts for code-based templates, which hampers their cohesion.

### 4.3.7 Limitations

The results presented in this comparative study have depended on many factors that could potentially limit the study.

The input model of some of the code-based tools given as Java objects could have favored their final results. As noted in Section 4.3.1, the fact that the template engines executions and the input models are coded in Java could benefit the code-based tools over the Model-based tools.

Although the presented patterns help highlight both the limitations and the performance of the tools, they are not exhaustive. Hence there might be other template implementations that are not considered in this study.

The absence of an expert for each tool can also limit the findings of our study. In fact, we are not experts in each of these tools. Therefore, there are potentially other optimization processes that we were not aware of when implementing the different templates. However, we believe that the templates are written from the perspective of the general level of competence of a regular software engineer.

# CHAPTER 5

# CONCLUSION

We conclude by summarizing the contributions of this thesis and outline future work. The work presented in this thesis makes several contributions to the field of TBCG.

## 5.1 Summary

A survey on template-based code generation has been missing in the literature. In this thesis, I present a literature survey on this topic and compare existing tools to provide the necessary guidelines and insights to help developers select the most appropriate approach and tool for their problem.

### 5.1.1 Systematic mapping study of TBCG

We conducted a SMS on the topic of TBCG, which has been missing in the literature. The objectives of this study are to better understand the characteristics of TBCG techniques and associated tools, identify research trends, and assess the importance of the role that MDE plays. We have systematically scanned the published, peer-reviewed literature and studied an extensive set of 481 papers published during the period 2000–2016. The analysis of this corpus is organized into facets of a novel classification scheme, which is of great value to modeling and software engineering researchers who are interested in painting an overview of the literature on TBCG. Our study shows that the community has been diversely using TBCG over the past 16 years and that research and development is still very active. TBCG has greatly benefited from MDE in 2005 and 2013 which mark the two peaks of the evolution of this area, tripling the average number of publications. In addition, TBCG has favored a template style that is output-based and high level modeling languages as input. TBCG is mainly used to generate source code and has been applied in a variety of domains. The community has been favoring the use of custom tools for code generation over popular ones. Most research using TBCG follows an MDE approach. Furthermore, both MDE and non-MDE tools are becoming effective development resources in industry.

### 5.1.2 Comparison of the Expressiveness and Performance of TBCG Tools

We presented the outcome of a comparative study on TBCG tools to evaluate their expressiveness power and test their performance. The goal of this paper is to implement M2T templates for specific metamodel patterns, investigate the limitations of TBCG tools with respect to an input these patterns and determine the most capable among the selected tools. The study is carried out based on nine most popular output-based TBCG tools over the last two decades years.

Our study shows that model-based tools are the most capable tools since most of them successfully implemented all the metamodel patterns. However, code-based tools performed much faster than model-based tools. JET is the best tool performance-wise. It generated the desired output almost instantly irrespective of the input model and failed to implement just the recursive pattern. Xtend2 offers the best compromise between the expressiveness and the performance. It succeeded to implement all the metamodel patterns in a reasonable time. Finally, we found that the recursion pattern is achieved by T4 and almost all the model-based tools, but it does not influence the overall performance of the tools.

## 5.2 Outlook

This survey is an added knowledge to the software engineering research community and practitioners. It guides them into making an informed choice of the appropriate

TBCG tool based on their requirements. Furthermore, it particularly benefits the researchers by presenting an overview of TBCG and by highlighting the areas that require more attention, hence guiding their future works.

As future work, we would like to revise the query to include not only "code" as the main output, but all the other possible artifacts such as documents. We plan to identify more patterns based on larger sets of metamodels and templates. We believe that this will also provide a feedback on the use of our metamodel patterns. Finally, we are investigating further criteria to compare TBCG tools, such as the elaboration of specific metrics to further enhance the guidelines.

# BIBLIOGRAPHY

[ReM] Remodd The Repository for Model-Driven Development. `http://www.cs.colostate.edu/remodd/v1/content/repository-model-driven-development-remodd-overview`. Accessed: 2017-03-15.

[2] (2015). The Metamodel Zoos. `http://web.emn.fr/x-info/atlanmod/index.php?title=Zoos`. Accessed: 2017-03-15.

[3] Adamko, A. (2005). Modeling data-oriented web applications using uml. In *EUROCON 2005 - The International Conference on Computer as a Tool*, volume 1, pages 752–755. IEEE.

[4] Anjorin, A., Saller, K., Rose, S., and Schürr, A. (2013). A framework for bidirectional model-to-platform transformations. In *5th International Conference on Software Language Engineering, SLE 2012, Revised Selected Papers*, volume 7745 of *LNCS*, pages 124–143. Springer Berlin Heidelberg.

[5] Antkiewicz, M. and Czarnecki, K. (2006). Framework-specific modeling languages with round-trip engineering. In *Model Driven Engineering Languages and Systems*, volume 4199 of *LNCS*, pages 692–706. Springer Berlin Heidelberg.

[6] Apache Software Foundation (2016). The Apache Velocity Projet. `http://velocity.apache.org/`. Accessed: 2017-04-06.

[7] Balzer, R. (1985). A 15 Year Perspective on Automatic Programming. *Transactions on Software Engineering*, 11(11):1257–1268.

[8] Basu, A. S., Lajolo, M., and Prevostini, M. (2005). A methodology for bridging the

gap between uml and codesign. In *UML for SOC Design*, pages 119–146. Springer US.

[9] Batot, E. and Sahraoui, H. (2016). A Generic Framework for Model-Set Selection for the Unification of Testing and Learning MDE Tasks. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, pages 374–384.

[10] Batot, E., Sahraoui, H., Syriani, E., Molins, P., and Sboui, W. (2016). Systematic mapping study of model transformations for concrete problems. In *International Conference on Model-Driven Engineering and Software Development*, pages 176–183.

[11] Beckmann, O., Houghton, A., Mellor, M., and Kelly, P. H. (2004). Runtime code generation in C++ as a foundation for domain-specific optimisation. In *Domain-Specific Program Generation*, volume 3016 of *LNCS*, pages 291–306. Springer Berlin Heidelberg.

[12] Behrens, T. and Richards, S. (2000). Statelator-behavioral code generation as an instance of a model transformation. In *International Conference on Advanced Information Systems Engineering*, volume 1789 of *LNCS*, pages 401–416. Springer Berlin Heidelberg.

[13] Bergmann, G., Horváth, Á., Ráth, I., and Varró, D. (2008). *A Benchmark Evaluation of Incremental Pattern Matching in Graph Transformation*, pages 396–410. Springer Berlin Heidelberg.

[14] Bonta, E. and Bernardo, M. (2009). Padl2java: A java code generator for process algebraic architectural descriptions. In *European Conference on Software Architecture*, pages 161–170. IEEE.

[15] Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M., and Khalil, M. (2007). Lessons from applying the systematic literature review process within the software engineering domain. *Journal of systems and software*, 80(4):571–583.

[16] Brown, A. W., Conallen, J., and Tropeano, D. (2005). Introduction: Models, modeling, and model-driven architecture (MDA). In *International Conference on Model-Driven Software Development*, pages 1–16. Springer Berlin Heidelberg.

[17] Brox, M., Sánchez-Solano, S., del Toro, E., Brox, P., and Moreno-Velo, F. J. (2013). CAD tools for hardware implementation of embedded fuzzy systems on FPGAs. *IEEE Transactions on Industrial Informatics*, 9(3):1635–1644.

[18] Brun, M., Delatour, J., and Trinquet, Y. (2008). Code generation from AADL to a real-time operating system: An experimentation feedback on the use of model transformation. In *Engineering of Complex Computer Systems*, pages 257–262. IEEE.

[19] Buchmann, T. and Schwägerl, F. (2013). Using meta-code generation to realize higher-order model transformations. In *International Joint conference on Software Technologies*, pages 536–541.

[20] Buckl, C., Knoll, A., and Schrott, G. (2005). Development of dependable real-time systems with Zerberus. In *11th IEEE Pacific Rim International Symposium on Dependable Computing*, pages 404–408.

[21] Buckl, C., Regensburger, M., Knoll, A., and Schrott, G. (2007). Models for automatic generation of safety-critical real-time systems. In *Availability, Reliability and Security*, pages 580–587. IEEE.

[22] Buezas, N., Guerra, E., de Lara, J., Martín, J., Monforte, M., Mori, F., Ogallar, E., Pérez, O., and Cuadrado, J. S. (2013). Umbra designer: Graphical modelling for

telephony services. In *European Conference on Modelling Foundations and Applications*, volume 7949 of *LNCS*, pages 179–191. Springer Berlin Heidelberg.

[23] Burmester, S., Giese, H., and Schäfer, W. (2005). Model-driven architecture for hard real-time systems: From platform independent models to code. In *European Conference on Model Driven Architecture-Foundations and Applications*, volume 3748 of *LNCS*, pages 25–40. Springer Berlin Heidelberg.

[24] Chen, K., Chang, Y.-C., and Wang, D.-W. (2010). Aspect-oriented design and implementation of adaptable access control for electronic medical records. *International Journal of Medical Informatics*, 79(3):181–203.

[25] Cho, H. and Gray, J. (2011). Design Patterns for Metamodels. In *Proceedings of the Compilation of the Co-located Workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, & VMIL'11*, SPLASH '11 Workshops, pages 25–32. ACM.

[26] Córdoba, I. and de Lara, J. (2015). A modelling language for the effective design of Java annotations. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, pages 2087–2092. ACM.

[27] Czarnecki, K. and Helsen, S. (2006). Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–645.

[28] Dahman, W. and Grabowski, J. (2010). Uml-based specification and generation of executable web services. In *System Analysis and Modeling*, volume 6598 of *LNCS*, pages 91–107. Springer Berlin Heidelberg.

[29] Domíguez, E., Pérez, B., Rubio, A. L., and Zapata, M. A. (2012). A systematic review of code generation proposals from state machine specifications. *Information and Software Technology*, 54(10):1045–1066.

[30] Durand, S. H. and Bonato, V. (2012). A tool to support Bluespec SystemVerilog coding based on UML diagrams. In *Annual Conference on IEEE Industrial Electronics Society*, pages 4670–4675. IEEE.

[31] Ecker, W., Velten, M., Zafari, L., and Goyal, A. (2014). The metamodeling approach to system level synthesis. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 1–2. IEEE.

[32] Fang, M., Ying, J., and Wu, M. (2006). A template engineering based framework for automated software development. In *10th International Conference on Computer Supported Cooperative Work in Design*, pages 1–6. IEEE.

[33] Fertalj, K., Kalpic, D., and Mornar, V. (2002). Source code generator based on a proprietary specification language. In *Hawaii International Conference on System Sciences*, volume 9. IEEE.

[34] Fischer, T., Kollner, C., Hardle, M., and Muller-Glaser, K. D. (2014). Product line development for modular FPGA-based embedded systems. In *Symposium on Rapid System Prototyping*, pages 9–15. IEEE.

[35] Floch, A., Yuki, T., Guy, C., Derrien, S., Combemale, B., Rajopadhye, S., and France, R. B. (2011). Model-Driven Engineering and Optimizing Compilers: A Bridge Too Far? In *Model Driven Engineering Languages and Systems*, volume 6981 of *LNCS*, pages 608–622. Springer Berlin Heidelberg.

[36] Fraternali, P. and Tisi, M. (2009). A higher order generative framework for weaving traceability links into a code generator for web application testing. In *International Conference on Web Engineering*, volume 5648 of *LNCS*, pages 340–354. Springer Berlin Heidelberg.

[37] Fu, J., Bastani, F. B., and Yen, I.-L. (2006). Automated AI planning and code pattern based code synthesis. In *International Conference on Tools with Artificial Intelligence*, pages 540–546. IEEE.

[38] Furusawa, T. (2010). Attempting to increase longevity of applications based on new SaaS/cloud technology. *Fujitsu Scientific and Technical Journal*, 46:223–228.

[39] Gessenharter, D. (2008). Mapping the UML2 semantics of associations to a java code generation model. In *International Conference on Model Driven Engineering Languages and Systems*, volume 5301 of *LNCS*, pages 813–827. Springer Berlin Heidelberg.

[40] Ghodrat, M. A., Givargis, T., and Nicolau, A. (2008). Control flow optimization in loops using interval analysis. In *International conference on Compilers, architectures and synthesis for embedded systems*, pages 157–166. ACM.

[41] Gopinath, V. S., Sprinkle, J., and Lysecky, R. (2011). Modeling of data adaptable reconfigurable embedded systems. In *International Conference and Workshops on Engineering of Computer Based Systems*, pages 276–283. IEEE.

[42] Gray, J., Tolvanen, J.-P., Kelly, S., Gokhale, A., Neema, S., and Sprinkle, J. (2007). *Handbook of Dynamic System Modeling*, book section Domain-Specific Modeling, pages 7–20. CRC Press.

[43] Guduvan, A.-R., Waeselynck, H., Wiels, V., Durrieu, G., Fusero, Y., and Schieber, M. (2013). A meta-model for tests of avionics embedded systems. In *nternational Conference on Model-Driven Engineering and Software Development*, pages 5–13.

[44] Gurunule, D. and Nashipudimath, M. (2015). A review: Analysis of aspect orientation and model driven engineering for code generation. *Procedia Computer Science*, 45:852–861.

[45] Hemel, Z., Kats, L. C., Groenewegen, D. M., and Visser, E. (2010). Code generation by model transformation: a case study in transformation modularity. *Software & Systems Modeling*, 9(3):375–402.

[46] Hinkel, G., Denninger, O., Krach, S., and Groenda, H. (2016). Experiences with model-driven engineering in neurorobotics. In *12th European Conference on Modelling Foundations and Applications, ECMFA 2016*, pages 217–228. Springer International Publishing.

[47] Hoisl, B., Sobernig, S., and Strembeck, M. (2013). Higher-order rewriting of model-to-text templates for integrating domain-specific modeling languages. In *International Conference on Model-Driven Engineering and Software Development*, pages 49–61.

[48] Jakumeit, E. et al. (2014). A Survey and Comparison of Transformation Tools Based on the Transformation Tool Contest. *Science of Computer Programming*, 85, Part A:41–99.

[49] Jörges, S. (2013). *Construction and Evolution of Code Generators*, volume 7747, chapter 2 The State of the Art in Code Generation, pages 11–38. Springer Berlin Heidelberg.

[50] Jouault, F., Allilaire, F., Bézivin, J., and Kurtev, I. (2008). ATL: A Model Transformation Tool. *Science of Computer Programming*, 72(1-2):31–39.

[51] Jugel, U. and Preußner, A. (2011). A case study on API generation. In *System Analysis and Modeling: About Models*, volume 6598 of *LNCS*, pages 156–172. Springer Berlin Heidelberg.

[52] Kelly, S. and Tolvanen, J.-P. (2008). *Domain-Specific Modeling: Enabling Full Code Generation*. John Wiley & Sons.

[53] Kitchenham, B. A., Budgen, D., and Brereton, O. P. (2011). Using mapping studies as the basis for further research - a participant-observer case study. *Information and Software Technology*, 53(6):638–651.

[54] Kitchenham, B. A., Dyba, T., and Jorgensen, M. (2004). Evidence-Based Software Engineering. In *International Conference on Software Engineering*, pages 273–281, Washington, DC, USA. IEEE Computer Society.

[55] Kleppe, A. G., Warmer, J., and Bast, W. (2003). *MDA Explained. The Model Driven Architecture: Practice And Promise*. Addison-Wesley.

[56] Kokar, M., Baclawski, K., and Gao, H. (2006). Category theory-based synthesis of a higher-level fusion algorithm: an example. In *International Conference on Information Fusion*, pages 1–8.

[57] Kolovos, D., Rose, L., Paige, R. F., and Garcıa Domınguez, A. (2010). *The Epsilon Book*. Eclipse.

[58] Kövi, A. and Varró, D. (2007). An eclipse-based framework for ais service configurations. In *4th International Service Availability Symposium, ISAS*, volume 4526 of *LNCS*, pages 110–126. Springer Berlin Heidelberg.

[59] Li, J., Xiao, H., and Yi, D. (2012). Designing universal template for database application system based on abstract factory. In *2012 International Conference on Computer Science and Information Processing*, CSIP, pages 1167–1170.

[60] Liu, Q. (2006). C++ techniques for high performance financial modelling. *WIT Transactions on Modelling and Simulation*, 43:1–8.

[61] Lúcio, L., Amrani, M., Dingel, J., Lambers, L., Salay, R., Selim, G. M., Syriani,

E., and Wimmer, M. (2014). Model Transformation Intents and Their Properties. *Software & Systems Modeling*, 15(3):685–705.

[62] Ma, M., Meissner, M., and Hedrich, L. (2012). A case study: Automatic topology synthesis for analog circuit from an asdex specification. In *Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design*, pages 9–12. IEEE.

[63] Manley, R. and Gregg, D. (2010). A program generator for intel aes-ni instructions. In *Progress in Cryptology - INDOCRYPT 2010: 11th International Conference on Cryptology*, volume 6498 of *LNCS*, pages 311–327. Springer Berlin Heidelberg.

[64] Mehmood, A. and Jawawi, D. N. (2013). Aspect-oriented model-driven code generation: A systematic mapping study. *Information and Software Technology*, 55(2):395–411.

[65] Microsoft (2016). Microsoft text template transformation toolkit. `https://msdn.microsoft.com/en-us/library/bb126445.aspx`. Accessed: 2016-07-05.

[66] Muller, P.-A., Studer, P., Fondement, F., and Bézivin, J. (2005). Platform independent web application modeling and development with Netsilon. *Software & Systems Modeling*, 4(4):424–442.

[67] O'Halloran, C. (2013). Automated verification of code automatically generated from simulink®. *Automated Software Engineering*, 20(2):237–264.

[68] OMG (2008). MOF Model to Text Language, v1.0.

[69] Petersen, K., Feldt, R., Mujtaba, S., and Mattsson, M. (2008). Systematic Mapping Studies in Software Engineering. In *Proceedings of the 12th International Confer-*

*ence on Evaluation and Assessment in Software Engineering*, volume 17 of *EASE'08*, pages 68–77. British Computer Society.

[70] Phillips, J., Chilukuri, R., Fragoso, G., Warzel, D., and Covitz, P. A. (2006). The caCORE software development kit: Streamlining construction of interoperable biomedical information services. *BMC medical informatics and decision making*, 6(2):1–16.

[71] Possatto, M. A. and Lucrédio, D. (2015). Automatically propagating changes from reference implementations to code generation templates. *Information and Software Technology*, 67:65–78.

[72] Rensink, Aand Van Gorp, P. (2010). Graph Transformation Tool Contest 2008. *International Journal on Software Tools for Technology Transfer*, 12(3):171–181.

[73] Rich, C. and Waters, R. C. (1988). Automatic programming: myths and prospects. *Computer*, 21(8):40–51.

[74] Rose, L. M., Matragkas, N., Kolovos, D. S., and Paige, R. F. (2012). A Feature Model for Model-to-Text Transformation Languages. In *ICSE Workshop on Modeling in Software Engineering*, pages 57–63. IEEE Press.

[75] Schattkowsky, T. and Lohmann, M. (2002). Rapid development of modular dynamic web sites using uml. In *International Conference on the Unified Modeling Language*, volume 2460 of *LNCS*, pages 336–350. Springer Berlin Heidelberg.

[76] Seriai, A., Benomar, O., Cerat, B., and Sahraoui, H. (2014). Validation of software visualization tools: A systematic mapping study. In *IEEE Working Conference on Software Visualization*, VISSOFT, pages 60–69.

[77] Singh, A., Schaeffer, J., and Green, M. (1991). A template-based approach to

the generation of distributed applications using a network of workstations. *IEEE Transactions on Parallel and Distributed Systems*, 2(1):52–67.

[78] Sousa, V., Syriani, E., and Paquin, M. (2017). Feedback on How MDE Tools are Used Prior to Academic Collaboration. In *32nd ACM SIGAPP Symposium On Applied Computing*.

[79] Sridhara, G., Pollock, L., and Vijay-Shanker, K. (2011). Automatically detecting and describing high level actions within methods. In *International Conference on Software Engineering*, pages 101–110. ACM.

[80] Stahl, T., Voelter, M., and Czarnecki, K. (2006). *Model-Driven Software Development – Technology, Engineering, Management*. John Wiley & Sons.

[81] Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2008). *EMF: Eclipse Modeling Framework*. Eclipse Series. Addison-Wesley Professional, 2nd edition edition.

[82] Tatsubori, M., Chiba, S., Killijian, M.-O., and Itano, K. (2000). OpenJava: A Class-Based Macro System for Java. In *Reflection and Software Engineering*, volume 1826 of *LNCS*, pages 117–133. Springer.

[83] Touraille, L., Traoré, M. K., and Hill, D. R. (2011). A model-driven software environment for modeling, simulation and analysis of complex systems. In *Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, pages 229–237. Society for Computer Simulation International.

[84] Valderas, P., Pelechano, V., and Pastor, O. (2006). Towards an end-user development approach for web engineering methods. In *International Conference on Advanced Information Systems Engineering*, volume 4001, pages 528–543. Springer Berlin Heidelberg.

[85] Varró, D., Asztalos, M., Bisztray, D., Boronat, A., Dang, D., Geiß, R., Greenyer, J., Van Gorp, P., Kniemeyer, O., Narayanan, A., Rencis, E., and Weinell, E. (2008). *Transformation of UML Models to CSP: A Case Study for Graph Transformation Tools*, pages 540–565. Springer Berlin Heidelberg.

[86] Vokáč, M. and Glattetre, J. M. (2005). Using a domain-specific language and custom tools to model a multi-tier service-oriented applicationexperiences and challenges. In *International Conference on Model Driven Engineering Languages and Systems*, volume 3713, pages 492–506. Springer.

# Appendix I

## Final corpus bibliography of the SMS

# BIBLIOGRAPHY

[1] Ab Rahim, L. and Whittle, J. (2010). Verifying semantic conformance of state machine-to-java code generators. In *Model Driven Engineering Languages and Systems*, pages 166–180.

[2] Ablonskis, L. (2010). An approach to generating program code in quickly evolving environments. *Information Systems Development*, pages 259–267.

[3] Ablonskis, L. and Nemuraite, L. (2010). Discovery of model implementation patterns in source code. *Information Technology and Control*, 39(1):68–76.

[4] Abrahão, S., Iborra, E., and Vanderdonckt, J. (2008). Usability evaluation of user interfaces generated with a model-driven architecture tool. *Maturing Usability*, pages 3–32.

[5] Aceto, G., Tarsitano, G., Jaekel, F.-W., and Benguria, G. (2012). Towards mda best practice: An innovative interpreter for smes. *Enterprise Interoperability V*, pages 237–246.

[6] Achilleos, A., Georgalas, N., and Yang, K. (2007). An open source domain-specific tools framework to support model driven development of oss. *Model Driven Architecture- Foundations and Applications*, pages 1–16.

[7] Achilleos, A., Kapitsaki, G. M., and Papadopoulos, G. A. (2012). A model-driven framework for developing web service oriented applications. *Current Trends in Web Engineering*, pages 181–195.

[8] Acquaviva, A., Bombieri, N., Fummi, F., and Vinco, S. (2013). Semi-automatic generation of device drivers for rapid embedded platform development. *Computer-*

*Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 32(9):1293–1306.

[9] Adamko, A. (2005). Modeling data-oriented web applications using uml. In *Computer as a Tool, 2005. EUROCON 2005.The International Conference on*, volume 1, pages 752–755.

[10] Adamkó, A. (2006). Uml-based modeling of data-oriented web applications. *Journal of Universal Computer Science*, 12(9):1104–1117.

[11] Akehurst, D., Howells, G., and Mcdonald-Maier, K. (2007). Implementing associations: Uml 2.0 to java 5. *Software & Systems Modeling*, 6(1):3–35.

[12] Alloush, I., Chiprianov, V., Kermarrec, Y., and Rouvrais, S. (2012). Linking telecom service high-level abstract models to simulators based on model transformations: The ims case study. *Information and Communication Technologies*, pages 100–111.

[13] Alloush, I., Kermarrec, Y., and Rouvrais, S. (2013a). A generalized model transformation approach to link design models to network simulators: Ns-3 case study. In *International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, pages 337–44.

[14] Alloush, I., Kermarrec, Y., and Rouvrais, S. (2013b). A transversal alignment between measurements and enterprise architecture for early verification of telecom service design. *Advances in Communication Networking*, pages 245–256.

[15] Alonso, D., Vicente-Chicote, C., Sánchez, P., Alvarez, B., and Losilla, F. (2007). Automatic ada code generation using a model-driven engineering approach. *Reliable Software Technologies Ada Europe 2007*, pages 168–179.

[16] Altiparmak, H., Tokgoz, B., Balcicek, O., Ozkaya, A., and Arslan, A. (2013). Source code generation for large scale applications. In *Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE), 2013 International Conference on*, pages 404–410.

[17] AmÃlio, N., Glodt, C., Pinto, F., and Kelsen, P. (2011). Platform-variant applications from platform-independent models via templates. *Electronic Notes in Theoretical Computer Science*, 279(3):3–25.

[18] Amelunxen, C., Königs, A., Rötschke, T., and Schürr, A. (2006). Moflon: A standard-compliant metamodeling framework with graph transformations. *Model Driven Architecture–Foundations and Applications*, pages 361–375.

[19] Ammar, M., Baklouti, M., Pelcat, M., Desnos, K., and Abid, M. (2016). *Automatic Generation of S-LAM Descriptions from UML/MARTE for the DSE of Massively Parallel Embedded Systems*, pages 195–211. Springer International Publishing.

[20] André, E., Benmoussa, M. M., and Choppy, C. (2014). Translating uml state machines to coloured petri nets using acceleo: A report. *arXiv preprint arXiv:1405.1112*.

[21] Andre, P., Ardourel, G., and Sunye, G. (2004). The bosco project: a jmi-compliant template-based code generator. In *IASSE*, pages 157–62.

[22] Anjorin, A., Saller, K., Rose, S., and Schürr, A. (2013). A framework for bidirectional model-to-platform transformations. *Software Language Engineering*, pages 124–143.

[23] Antebi, O., Neubrand, M., and Puder, A. (2012). Cross-compiling android applications to windows phone 7. *Mobile Computing, Applications, and Services*, pages 283–302.

[24] Antkiewicz, M. and Czarnecki, K. (2006). Framework-specific modeling languages with round-trip engineering. In *Model Driven Engineering Languages and Systems*, pages 692–706.

[25] Antkiewicz, M. and Czarnecki, K. (2008). Design space of heterogeneous synchronization. *Generative and Transformational Techniques in Software Engineering II*, pages 3–46.

[26] Arkın, E., Tekinerdogan, B., and İmre, K. M. (2013). Model-driven approach for supporting the mapping of parallel algorithms to parallel computing platforms. In *Model Driven Engineering Languages and Systems*, pages 757–773.

[27] Arnoldus, J., Bijpost, J., and Van Den Brand, M. (2007). Repleo: A syntax-safe template engine. In *Proceedings of the 6th International Conference on Generative Programming and Component Engineering*, pages 25–32.

[28] Aßmann, U., Bartho, A., Bürger, C., Cech, S., Demuth, B., Heidenreich, F., Johannes, J., Karol, S., Polowinski, J., Reimann, J., Schroeter, J., Seifert, M., Thiele, M., Wende, C., and Wilke, C. (2014). Dropsbox: the dresden open software toolbox. *Software & Systems Modeling*, 13(1):133–169.

[29] Axelsen, H. B. (2011). Clean translation of an imperative reversible programming language. *Compiler Construction*, pages 144–163.

[30] Azevedo, L., Fernandes, C. T., and Guerra, E. M. (2013). Architectural model for generating user interfaces based on class metadata. In *Computational Science and Its Applications - ICCSA 2013*, volume 7973 LNCS, pages 230–245.

[31] Backes, M., Busenius, A., and Hriţcu, C. (2012). On the development and formalization of an extensible code generator for real life security protocols. *NASA Formal Methods*, pages 371–387.

[32] Badreddin, O., Forward, A., and Lethbridge, T. C. (2014a). Exploring a model-oriented and executable syntax for uml attributes. *Software Engineering Research, Management and Applications*, pages 33–53.

[33] Badreddin, O., Forward, A., and Lethbridge, T. C. (2014b). Improving code generation for associations: Enforcing multiplicity constraints and ensuring referential integrity. *Software Engineering Research, Management and Applications*, pages 129–149.

[34] BaekGyu, K., Phan, L. T. X., Sokolsky, O., and Insup, L. (2013). Platform-dependent code generation for embedded real-time software. In *2013 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES), 29 Sept.-4 Oct. 2013*, 2013 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES), pages 1–10. IEEE.

[35] Bagheri, H. and Sullivan, K. (2012). Pol: Specification-driven synthesis of architectural code frameworks for platform-based applications. In *Proceedings of the 11th International Conference on Generative Programming and Component Engineering*, GPCE '12, pages 93–102. ACM.

[36] Baker, P. and Jervis, C. (2007). Testing uml2.0 models using ttcn-3 and the uml2.0 testing profile. *SDL 2007: Design for Dependable Systems*, pages 86–100.

[37] Balderas-Contreras, T., Cumplido, R., and Rodriguez, G. (2014). Synthesizing vhdl from activity models in uml 2. *International Journal of Circuit Theory and Applications*, 42(5):542–550.

[38] Balogh, A., Bergmann, G., Csertán, G., Gönczy, L., Horváth, Á., Majzik, I., Pataricza, A., Polgár, B., Ráth, I., and Varró, D. e. a. (2010). Workflow-driven tool in-

tegration using model transformations. *Graph Transformations and Model-Driven Engineering*, pages 224–248.

[39] Balogh, A. and Varro, D. (2006). Advanced model transformation language constructs in the viatra2 framework. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, volume 2, pages 1280–1287.

[40] Banerjee, A. and Gupta, S. K. S. (2014). Model based code generation for medical cyber physical systems. In *Proceedings of the 1st Workshop on Mobile Medical Applications*, MMA '14, pages 22–27. ACM.

[41] Barbier, G., Cucchi, V., and Hill, D. R. (2013). Contribution of model-driven engineering to crop modeling. *Computational Science and Its Applications ICCSA 2013*, pages 253–263.

[42] Basir, N., Denney, E., and Fischer, B. (2010). Deriving safety cases for hierarchical structure in model-based development. *Computer Safety, Reliability, and Security*, pages 68–81.

[43] Basu, A. S., Lajolo, M., and Prevostini, M. (2005). A methodology for bridging the gap between uml and codesign. *UML for SOC Design*, pages 119–146.

[44] Behrens, T. and Richards, S. (2000). Statelator - behavioral code generation as an instance of a model transformation. In Wangler, B. and Bergman, L., editors, *12th International Conference on Advanced Information Systems Engineering, CAiSE 2000*, volume 1789, pages 401–416. Springer Verlag.

[45] Beneken, T. S. (2005). Evolution and maintenance of mda applications. *Model-Driven Software Development*, pages 269–286.

[46] Bennet, A. F., Chua, B. S., Pflaum, B. L., Erwig, M., Fu, Z., Loft, R. D., and Muccino, J. C. (2008). The inverse ocean modeling system. part i: Implementation. *Journal of Atmospheric and Oceanic Technology*, 25(9):1608–1622.

[47] Benouda, H., Azizi, M., Esbai, R., and Moussaoui, M. (2016). *MDA Approach to Automate Code Generation for Mobile Applications*, pages 241–250. Springer Singapore.

[48] Bezati, E., Yviquel, H., Raulet, M., and Mattavelli, M. (2011). A unified hardware/-software co-synthesis solution for signal processing systems. In *2011 Conference on Design and Architectures for Signal and Image Processing, DASIP 2011, November 2, 2011 - November 4, 2011*, Conference on Design and Architectures for Signal and Image Processing, DASIP, pages 186–191. IEEE Computer Society.

[49] Biehl, M., El-Khoury, J., Loiret, F., and Törngren, M. (2014). On the modeling and generation of service-oriented tool chains. *Software & Systems Modeling*, 13(2):461–480.

[50] Binder, W. and Hulaas, J. (2006). Flexible and efficient measurement of dynamic bytecode metrics. In *5th International Conference on Generative Programming and Component Engineering, GPCE'06. Co-located with the 21st International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006, October 22, 2006 - October 26, 2006*, Proceedings of the 5th International Conference on Generative Programming and Component Engineering, GPCE'06, pages 171–180. Association for Computing Machinery.

[51] Binnig, C. and Schmidt, A. (2002). Development of a uiml renderer for different target languages: Experiences and design decisions. *Computer-Aided Design of User Interfaces III*, pages 267–274.

[52] Birken, K. (2014). Building code generators for dsls using a partial evaluator for the xtend language. In Margaria, T. and Steffen, B., editors, *6th International Symposium Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change, ISoLA, Proceedings, Part I*, volume 8802 of *LNCS*, pages 407–424. Springer Berlin Heidelberg.

[53] Bocciarelli, P., D'Ambrogio, A., Falcone, A., Garro, A., and Giglio, A. (2016). *A Model-Driven Approach to Enable the Distributed Simulation of Complex Systems*, pages 171–183. Springer International Publishing.

[54] Bochao, L. and Ohori, A. (2009). A flattening strategy for sml module compilation and its implementation. *Computer Software*, 26(3):136–154.

[55] Bonichon, R., Déharbe, D., Lecomte, T., and Medeiros, V. (2015). *LLVM-Based Code Generation for B*, pages 1–16. Springer International Publishing.

[56] Bork, M., Geiger, L., Schneider, C., and Zundorf, A. (2008). Towards roundtrip engineering - a template-based reverse engineering approach. In *Model Driven Architecture - Foundations and Applications*, volume 5095 LNCS, pages 33–47.

[57] Bouaziz, R., Kallel, S., and Coulette, B. (2014). An approach for security patterns application in component based models. *Computational Science and Its Applications - ICCSA 2014*, pages 283–296.

[58] Bredenfeld, A., Ihler, E., and Vogel, O. (2000). Genvis-model-based generation of data visualizers. In *Technology of Object-Oriented Languages, 2000. TOOLS 33. Proceedings. 33rd International Conference on*, pages 396–406.

[59] Brown, A. W. (2004). Model driven architecture: Principles and practice. *Software and Systems Modeling*, 3(4).

[60] Brown, A. W., Conallen, J., and Tropeano, D. (2005). Introduction: Models, modeling, and model-driven architecture (mda). *Model-Driven Software Development*, pages 1–16.

[61] Brox, M., Sanchez-Solano, S., Del Toro, E., Brox, P., and Moreno-Velo, F. J. (2013). Cad tools for hardware implementation of embedded fuzzy systems on fpgas. *IEEE Transactions on Industrial Informatics*, 9(3):1635–1644.

[62] Brun, M., Delatour, J., and Trinquet, Y. (2008). Code generation from aadl to a real-time operating system: An experimentation feedback on the use of model transformation. In *Engineering of Complex Computer Systems, 2008. ICECCS 2008. 13th IEEE International Conference on*, pages 257–262. IEEE.

[63] Bucher, R. and Balemi, S. (2006). Rapid controller prototyping with matlab/simulink and linux. *Control Engineering Practice*, 14(2):185–192.

[64] Buchmann, T. and Schwägerl, F. (2013). Using meta-code generation to realize higher-order model transformations. In *8th International Conference on Software Engineering and Applications*, pages 536–541.

[65] Buckl, C., Knoll, A., and Schrott, G. (2005a). Development of dependable real-time systems with zerberus. In *Dependable Computing, 2005. Proceedings. 11th Pacific Rim International Symposium on*, pages 404–408.

[66] Buckl, C., Knoll, A., and Schrott, G. (2005b). The zerberus language: Describing the functional model of dependable real-time systems. *Dependable Computing*, pages 101–120.

[67] Buckl, C., Knoll, A., and Schrott, G. (2006). Template-based development of fault-tolerant embedded software. In *International Conference on Software Engineering Advances*, pages 65–65. IEEE.

[68] Buckl, C., Knoll, A., and Schrott, G. (2007a). Model-based development of fault-tolerant embedded software. In *Leveraging Applications of Formal Methods, Verification and Validation, 2006. ISoLA 2006. Second International Symposium on*, pages 103–110.

[69] Buckl, C., Regensburger, M., Knoll, A., and Schrott, G. (2007b). Models for automatic generation of safety-critical real-time systems. In *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on*, pages 580–587. IEEE.

[70] Buckl, C., Sommer, S., Scholz, A., Knoll, A., and Kemper, A. (2008). Generating a tailored middleware for wireless sensor network applications. In *Sensor Networks, Ubiquitous and Trustworthy Computing, 2008. SUTC '08. IEEE International Conference on*, pages 162–169.

[71] Buezas, N., Guerra, E., De Lara, J., Martin, J., Monforte, M., Mori, F., Ogallar, E., Perez, O., and Sanchez Cuadrado, J. (2013). Umbra designer: Graphical modelling for telephony services. In *9th European Conference on Modelling Foundations and Applications, ECMFA 2013, July 1, 2013 - July 1, 2013*, volume 7949 LNCS of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 179–191. Springer Verlag.

[72] Bures, T., Malohlava, M., and Hnetynka, P. (2008). Using dsl for automatic generation of software connectors. In *Composition-Based Software Systems, 2008. ICCBSS 2008. Seventh International Conference on*, pages 138–147.

[73] Bürger, C., Karol, S., Wende, C., and Aßmann, U. (2011). Reference attribute grammars for metamodel semantics. *Software Language Engineering*, pages 22–41.

[74] Burmester, S., Giese, H., and Schafer, W. (2005a). Model-driven architecture for

hard real-time systems: from platform independent models to code. In *Model Driven Architecture-Foundations and Applications. First European Conference, ECMDA-FA 2005. Proceedings, 7-10 Nov. 2005*, pages 25–40. Springer-Verlag.

[75] Burmester, S., Giese, H., and Tichy, M. (2005b). Model-driven development of reconfigurable mechatronic systems with mechatronic uml. In *European MDA Workshops: Foundations and Applications, MDAFA 2004. Revised Selected Papers, June 10, 2004 - June 11, 2004*, volume 3599 of *Lecture Notes in Computer Science*, pages 47–61. Springer Verlag.

[76] Butt, S. A., Sayyah, P., and Lavagno, L. (2011). Model-based hardware/software synthesis for wireless sensor network applications. In *Saudi International Electronics, Communications and Photonics Conference 2011, SIECPC*, pages 269–286. IEEE Computer Society.

[77] Campos, R. S., Campos, F. O., Gomes, J. M., de Barros Barbosa, C., Lobosco, M., and Dos Santos, R. W. (2013). Comparing high performance techniques for the automatic generation of efficient solvers of cardiac cell models. *Computing*, 95(1):639–660.

[78] Cao, Z., Dong, Y., and Wang, S. (2011). Compiler backend generation for application specific instruction set processors. *Programming Languages and Systems*, pages 121–136.

[79] Cardoso, T., Barros, E., Prado, B., and Aziz, A. (2012). Communication software synthesis from uml-esl models. In *2012 25th Symposium on Integrated Circuits and Systems Design, SBCCI*, pages 1–6. IEEE Computer Society.

[80] Carton, A., Driver, C., Jackson, A., and Clarke, S. (2009). Model-driven theme/uml. *Transactions on Aspect-Oriented Software Development VI*, pages 238–266.

[81] Cechticky, V., Chevalley, P., Pasetti, A., and Schaufelberger, W. (2003). A generative approach to framework instantiation. *Generative Programming and Component Engineering*, pages 267–286.

[82] Cechticky, V., Egli, M., Pasetti, A., Rohlik, O., and Vardanega, T. (2006). A uml2 profile for reusable and verifiable software components for real-time applications. *Reuse of Off-the-Shelf Components*, pages 312–325.

[83] Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., and Matera, M. (2003). Chapter 14 - tools for model-based development of web applications. In *Designing Data-Intensive Web Applications*, The Morgan Kaufmann Series in Data Management Systems, pages 499 – 517. Morgan Kaufmann.

[84] Challenger, M., Kardas, G., and Tekinerdogan, B. (2016). A systematic approach to evaluating domain-specific modeling language environments for multi-agent systems. *Software Quality Journal*, pages 1–41.

[85] Chared, Z. and Tyszberowicz, S. (2013). Projective template-based code generation. In *International Conference on Advanced Information Systems Engineering*, volume 998, pages 81–87.

[86] Chen, C. Y., Shun, Y. C., Cheng, C. C., Liao, P. S., and Fang, Z. C. (2007). Matlab-based rapid controller development platform for control applications. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 221(11):1461–1473.

[87] Chen, K., Chang, Y.-C., and Wang, D.-W. (2010). Aspect-oriented design and implementation of adaptable access control for electronic medical records. *International Journal of Medical Informatics*, 79(3):181–203.

[88] Chen, K. and Lin, C. W. (2006). An aspect-oriented approach to declarative access control for web applications. In *8th Asia-Pacific Web Conference, APWeb 2006: Frontiers of WWW Research and Development*, volume 3841 LNCS, pages 176–188. Springer Berlin Heidelberg, Harbin.

[89] Chen, W., Liu, S., Wei, J., and Wang, L. (2011). Automatic construction of deployment descriptors for web applications. In *e-Business Engineering (ICEBE), 2011 IEEE 8th International Conference on*, pages 257–264.

[90] Chen, X., Liu, P., Qiu, X., Huang, H., and Duan, H. (2012). A novel approach to automatic code generation for automatic train protection. *International Journal of Innovative Computing, Information &amp; Control*, 8(9):6329–6344.

[91] Cheng, C.-H., Geisinger, M., Ruess, H., Buckl, C., and Knoll, A. (2012). Mgsyn: Automatic synthesis for industrial automation. In *24th International Conference on Computer Aided Verification, CAV 2012, July 7, 2012 - July 13, 2012*, volume 7358 LNCS, pages 658–664. Springer Verlag.

[92] Chevillat, C., Carrington, D., Strooper, P., Süß, J. G., and Wildman, L. (2008). Model-based generation of interlocking controller software from control tables. *Model Driven Architecture - Foundations and Applications*, pages 349–360.

[93] Chivers, H. and Paige, R. F. (2005). Xround: Bidirectional transformations and unifications via a reversible template language. *Model Driven Architecture - Foundations and Applications*, pages 205–219.

[94] Choi, S.-H. (2004). Component reconfiguration tool for software product lines with xml technology. *Web Information Systems - WISE 2004*, pages 572–583.

[95] Cicchetti, A., Di Ruscio, D., Iovino, L., and Pierantonio, A. (2013). Managing the

evolution of data-intensive web applications by model-driven techniques. *Software & Systems Modeling*, (1):53–83.

[96] Cirilo, E., Kulesza, U., Coelho, R., de Lucena, C. J., and von Staa, A. (2008). Integrating component and product lines technologies. *High Confidence Software Reuse in Large Systems*, pages 130–141.

[97] Clifton-Everest, R., McDonell, T. L., Chakravarty, M. M., and Keller, G. (2014). Embedding foreign code. *Practical Aspects of Declarative Languages*, pages 136–151.

[98] Clowes, D., Kolovos, D., Holmes, C., Rose, L., Paige, R., Johnson, J., Dawson, R., and Probets, S. (2010). A reflective approach to model-driven web engineering. *Modelling Foundations and Applications*, pages 62–73.

[99] Corre, Y., Diguet, J. P., Lagadec, L., Heller, D., and Blouin, D. (2013). Fast template-based heterogeneous mpsoc synthesis on fpga. In *Reconfigurable Computing: Architectures, Tools and Applications. 9th International Symposium (ARC 2013)*, pages 154–66. Springer Verlag.

[100] Cosentino, V., Tisi, M., and Izquierdo, J. L. C. (2015). A model-driven approach to generate external dsls from object-oriented apis. *SOFSEM 2015: Theory and Practice of Computer Science*, pages 423–435.

[101] Costa, D., Nóbrega, L., and Nunes, N. J. (2007). An mda approach for generating web interfaces with uml concurtasktrees and canonical abstract prototypes. *Task Models and Diagrams for Users Interface Design*, pages 137–152.

[102] Cuadrado, J. S., Guerra, E., and De Lara, J. (2014). A component model for model transformations. *IEEE Transactions on Software Engineering*, 40(11):1042–1060.

[103] Cuesta, A. G., Granja, J. C., and O'Connor, R. V. (2009). A model driven architecture approach to web development. *Software and Data Technologies*, pages 101–113.

[104] Cuesta, P., Gómez, A., and González, J. C. (2008). Agent oriented software engineering. *Issues in Multi-Agent Systems*, pages 1–31.

[105] Dagand, P.-E., Baumann, A., and Roscoe, T. (2009). Filet-o-fish: Practical and dependable domain-specific languages for os development. In *5th Workshop on Programming Languages and Operating Systems, PLOS*, pages 35–39. Association for Computing Machinery.

[106] Dahman, W. and Grabowski, J. (2011). Uml-based specification and generation of executable web services. *System Analysis and Modeling: About Models*, pages 91–107.

[107] Danilchenko, Y. and Fox, R. (2012). Automated code generation using case-based reasoning, routine design and template-based programming. In *MAICS*, pages 119–125.

[108] Danvy, O., Grobauer, B., and Rhiger, M. (2002). A unifying approach to goal-directed evaluation. *New Generation Computing*, 20(1):53–73.

[109] de la Fuente, D., Barba, J., Caba, J., Peñil, P., López, J. C., and Sánchez, P. (2016). *Building a Dynamically Reconfigurable System Through a High-Level Development Flow*, pages 51–73. Springer International Publishing.

[110] de Lara, J. and Guerra, E. (2013). From types to type requirements: Genericity for model-driven engineering. *Software and Systems Modeling*, 12(3):453–474.

[111] de Lara, J., Guerra, E., and Cuadrado, J. S. (2015). Model-driven engineering with domain-specific meta-modelling languages. *Software & Systems Modeling*, 14(1):429–459.

[112] Dekeyser, J.-L., Boulet, P., Marquet, P., and Meftali, S. (2005). Model driven engineering for soc co-design. In *3rd International IEEE Northeast Workshop on Circuits and Systems Conference, NEWCAS 2005, June 19, 2005 - June 22, 2005*, volume 2005, pages 21–25. Institute of Electrical and Electronics Engineers Computer Society.

[113] Demakov, A. V. (2007). Object-oriented description of graph data structures. *Programming and Computer Software*, 33(5):261–271.

[114] Denney, E. and Fischer, B. (2005). Certifiable program generation. In *Generative Programming and Component Engineering: 4th International Conference*, pages 17–28.

[115] Di Natale, M., Guo, L., Zeng, H., and Sangiovanni-Vincentelli, A. (2010). Synthesis of multitask implementations of simulink models with minimum delays. *IEEE Transactions on Industrial Informatics*, 6(4):637–651.

[116] Di Natale, M., Perillo, D., Chirico, F., Sindico, A., and Sangiovanni-Vincentelli, A. (2016). A model-based approach for the synthesis of software to firmware adapters for use with automatically generated components. *Software & Systems Modeling*, pages 1–23.

[117] Di Rocco, J., Di Ruscio, D., Iovino, L., and Pierantonio, A. (2014). Dealing with the coupled evolution of metamodels and model-to-text transformations. In *Model Driven Engineering Languages and Systems*, volume 1331, pages 22–31.

[118] Diethelm, I., Geiger, L., and Zundorf, A. (2005). Applying story driven modeling to the paderborn shuttle system case study. In *International Workshop on Scenarios: Models, Transformations and Tools*, volume 3466 of *Lecture Notes in Computer Science*, pages 109–133. Springer Verlag.

[119] do Nascimento, F. A. M., Oliveira, M. F., and Wagner, F. R. (2012). A model-driven engineering framework for embedded systems design. *Innovations in Systems and Software Engineering*, 8(1):19–33.

[120] Do Nascimento, F. A. M., Oliveira, M. F. S., and Wagner, F. R. (2007). Modes: Embedded systems design methodology and tools based on mde. In *4th International Workshop on Model-Based Methodologies for Pervasive and Embedded Software, MOMPES*, pages 67–76. Inst. of Elec. and Elec. Eng. Computer Society.

[121] Dolgert, A., Gibbons, L., and Kuznetsov, V. (2008). Rapid web development using ajax and python. *Journal of Physics: Conference Series*, 119(4):042011.

[122] Doroshenko, A., Zhereb, K., and Yatsenko, O. (2013). Developing and optimizing parallel programs with algebra-algorithmic and term rewriting tools. In *9th International Conference on Information and Communication Technologies in Education, Research, and Industrial Applications, ICTERI*, volume 412 CCIS of *Communications in Computer and Information Science*, pages 70–92. Springer Verlag.

[123] Drasutis, S., Pilkauskas, V., and Rubliauskas, D. (2005). Transformation for designing distributed internet information systems under model driven architecture. *Information Technology and Control*, 34(2):102–8.

[124] Driver, C., Reilly, S., Linehan, É, a., Cahill, V., and Clarke, Siobhá, n. (2011). Managing embedded systems complexity with aspect-oriented model-driven engineering. *ACM Trans. Embed. Comput. Syst.*, 10(2):21:1–21:26.

[125] Duma, R., Dobra, P., Trusca, M., Petreus, D., and Moga, D. (2011). Towards a rapid control prototyping toolbox for the stellaris lm3s8000 microcontrollers. In *18th IFAC World Congress, August 28, 2011 - September 2, 2011*, volume 18, pages 1965–1970. IFAC Secretariat.

[126] Duma, R., Petreus, D., Sita, V. I., Dobra, P., and Rusu, A. (2010). Rapid control prototyping toolbox for renesas m32c87 microcontroller. In *17th IEEE International Conference on Automation, Quality and Testing, Robotics, AQTR 2010, May 28, 2010 - May 30, 2010*, volume 1, pages 135–140. IEEE Computer Society.

[127] Durand, S. and Bonato, V. (2012). A tool to support bluespec systemverilog coding based on uml diagrams. In *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, pages 4670–4675.

[128] Ecker, W. and Schreiner, J. (2016). Introducing model-of-things (mot) and model-of-design (mod) for simpler and more efficient hardware generators. In *2016 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–6.

[129] Ecker, W., Velten, M., Zafari, L., and Goyal, A. (2014). The metamodeling approach to system level synthesis. In *17th Design, Automation and Test in Europe, DATE*, pages 1–2. Institute of Electrical and Electronics Engineers Inc.

[130] Edmunds, A. (2014). Templates for event-b code generation. In *Abstract State Machines, Alloy, B, TLA, VDM, and Z: 4th International Conference*, pages 284–289.

[131] Edwards, G., Brun, Y., and Medvidovic, N. (2012). Automated analysis and code generation for domain-specific models. In *Joint 10th Working IEEE/IFIP Conference on Software Architecture, WICSA 2012 and 6th European Conference on Software, ECSA 2012*, pages 161–170. IEEE Computer Society.

[132] Eessaar, E. (2008). On pattern-based database design and implementation. In *Software Engineering Research, Management and Applications, 2008. SERA '08. Sixth International Conference on*, pages 235–242.

[133] El Kaed, C., Denneulin, Y., Ottogalli, F.-G., and Mora, L. F. M. (2010). Combining ontology alignment with model driven engineering techniques for home devices interoperability. *Software Technologies for Embedded and Ubiquitous Systems*, pages 71–82.

[134] El-Khoury, J., Ekelin, C., and Ekholm, C. (2016). *Supporting the Linked Data Approach to Maintain Coherence Across Rich EMF Models*, pages 36–47. Springer International Publishing.

[135] Eltoweissy, S. A. B. G. a. G. H. (2007). Model-based evolution of collaborative agent-based systems. *Journal of the Brazilian Computer Society*, 13(4):17–38.

[136] Erdweg, S., Van Der Storm, T., Völter, M., Boersma, M., Bosman, R., Cook, W. R., Gerritsen, A., Hulshout, A., Kelly, S., and Loh, A. e. a. (2013). The state of the art in language workbenches. *Software Language Engineering*, pages 197–217.

[137] Ertl, M. A. and Gregg, D. (2002). Building an interpreter with vmgen. *Compiler Construction*, pages 5–8.

[138] Estevez, E., Marcos, M., Sarachaga, I., Lopez, F., Burgos, A., Perez, F., and Orive, D. (2009). Model based documentation of automation applications. In *Industrial Informatics, 2009. INDIN 2009. 7th IEEE International Conference on*, pages 768–774.

[139] Estévez, E., Sánchez-García, A., Gámez-García, J., Gómez-Ortega, J., and Satorres-Martínez, S. (2016). A novel model-driven approach to support develop-

ment cycle of robotic systems. *The International Journal of Advanced Manufacturing Technology*, 82(1):737–751.

[140] Estublier, J., Ionita, A. D., and Nguyen, T. (2011). Code generation for a bi-dimensional composition mechanism. *Software Engineering Techniques*, pages 171–185.

[141] Etien, A., Muller, A., Legrand, T., and Paige, R. F. (2015). Localized model transformations for building large-scale transformations. *Software & Systems Modeling*, 14(3):1189–1213.

[142] Fang, M., Ying, J., and Wu, M. (2006). A template engineering based framework for automated software development. In *Computer Supported Cooperative Work in Design, 2006. CSCWD '06. 10th International Conference on*, pages 1–6.

[143] Fernández, M. R., Alonso, I. G., and Casanova, E. Z. (2016). Improving the interoperability in the digital home through the automatic generation of software adapters from a sysml model. *Journal of Intelligent & Robotic Systems*, pages 1–11.

[144] Ferreira, J., Silva, A., and Delgado, J. (2007). Ir-case tool [information retrieval]. In *Proceedings of the IASTED International Conference on Software Engineering as part of the 25th IASTED International Multi-Conference on Applied Informatics*, pages 171–176. Acta Press.

[145] Fertalj, K., Kalpic, D., and Mornar, V. (2002). Source code generator based on a proprietary specification language. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 3696–704.

[146] Fischer, B. and Visser, E. (2004). Retrofitting the autobayes program synthesis system with concrete syntax. In *International Seminar on Domain-Specific Program Generation*, volume 3016, pages 239–253. Springer Verlag.

[147] Fischer, T., Kollner, C., Hardle, M., and Muller-Glaser, K. (2014). Product line development for modular fpga-based embedded systems. In *2014 25nd IEEE International Symposium on Rapid System Prototyping*, pages 9–15.

[148] Fleurey, F., Baudry, B., Muller, P.-A., and Le Traon, Y. (2009). Qualifying input test data for model transformations. *Software & Systems Modeling*, 8(2):185–203.

[149] Fleurey, F., Breton, E., Baudry, B., Nicolas, A., and Jézéquel, J.-M. (2007). Model-driven engineering for software migration in a large industrial context. In *Model Driven Engineering Languages and Systems*, pages 482–497.

[150] Fleury, M., Downton, A., and Clark, A. (2000). Analysis prediction template toolkit (aptt) for object-based computation. *IEE Proceedings: Software*, 147(2):37–47.

[151] Floch, A., Yuki, T., Guy, C., Derrien, S., Combemale, B., Rajopadhye, S., and France, R. B. (2011). Model-driven engineering and optimizing compilers: A bridge too far? In *Model Driven Engineering Languages and Systems*, pages 608–622.

[152] Ford, R. and Riley, G. (2012). The bespoke framework generator. *Earth System Modelling - Volume 3*, pages 55–67.

[153] Fraternali, P. and Tisi, M. (2009). A higher order generative framework for weaving traceability links into a code generator for web application testing. *9th International Conference on Web Engineering, ICWE 2009*, pages 340–354.

[154] Fraternali, P. and Tisi, M. (2010). Multi-level tests for model driven web applications. *Web Engineering*, pages 158–172.

[155] Freeman, G., Batory, D., and Lavender, G. (2008). Lifting transformational mod-

els of product lines: A case study. In *1st International Conference on Model Transformations, ICMT*, pages 16–30. Springer Verlag.

[156] Fu, J., Bastani, F. B., and Yen, I. L. (2006). Automated ai planning and code pattern based code synthesis. In *18th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2006, October 13, 2006 - October 15, 2006*, pages 540–544. IEEE Computer Society.

[157] Fu, J., Bastani, F. B., and Yen, I. L. (2008). Model-driven prototyping based requirements elicitation. In *14th Monterey Workshop 2007, September 10, 2007 - September 13, 2007*, volume 5320 LNCS, pages 43–61. Springer Verlag.

[158] Furusawa, T. (2010). Attempting to increase longevity of applications based on new saas/cloud technology. *Fujitsu Scientific and Technical Journal*, 46(2):223–8.

[159] Gamatie, A., Le Beux, S., Piel, E., Atitallah, R. B., Etien, A., Marquet, P., and Dekeyser, J.-L. (2011). A model-driven design framework for massively parallel embedded systems. *Transactions on Embedded Computing Systems*, 10(4):39:1–39:36.

[160] García, E., Valero, S., and Giret, A. (2016). *ROMAS-Magentix2*, pages 153–171. Springer International Publishing.

[161] García, J., Azanza, M., Irastorza, A., and Díaz, O. (2014). Testing mofscript transformations with handymof. *Theory and Practice of Model Transformations*, pages 42–56.

[162] Garg, R. M. and Dahiya, D. (2011). An aspect oriented component based model driven development. *Software Engineering and Computer Systems*, pages 502–517.

[163] Gaustad, K., Shippert, T., Ermold, B., Beus, S., Daily, J., Borsholm, A., and

Fox, K. (2014). A scientific data processing framework for time series netcdf data. *Environmental Modelling & Software*, 60:241–249.

[164] GÃűbe, F., Timmermanns, T., Ney, O., and Kowalewski, S. (2016). Synthesis tool for automation controller supervision. In *2016 13th International Workshop on Discrete Event Systems (WODES)*, pages 424–431.

[165] Ge, G. and Whitehead, E. J. (2008). Rhizome: a feature modeling and generation platform. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, pages 375–378. IEEE.

[166] Gedik, B. and Andrade, H. (2012). A model-based framework for building extensible, high performance stream processing middleware and programming language for ibm infosphere streams. *Software: Practice and Experience*, 42(11):1363–1391.

[167] Geiger, L. and Zündorf, A. (2006). Developing tools with fujaba xprom. *Generative and Transformational Techniques in Software Engineering*, pages 344–356.

[168] Geiger, N., George, T., Hahn, M., Jubeh, R., and Zündorf, A. (2010). Using actions charts for reactive web application modeling. *Current Trends in Web Engineering*, pages 49–60.

[169] Geng, P., Ouyang, M., Li, J., and Xu, L. (2012). Embedded c code generation platform for electric vehicle controller. In *2012 International Conference on Electrical Insulating Materials and Electrical Engineering, EIMEE 2012*, volume 546-547, pages 778–783. Trans Tech Publications.

[170] Geng, P., Ouyang, M., Li, J., and Xu, L. (2013). Automated code generation for development of electric vehicle controller. *Proceedings of the FISITA 2012 World Automotive Congress*, pages 459–468.

[171] George, B., Bohner, S. A., and He, N. (2006). Towards a model level debugger for the cougaar model driven architecture system. *Innovative Concepts for Autonomic and Agent-Based Systems*, pages 86–97.

[172] Gessenharter, D. (2008). *Mapping the UML2 semantics of associations to a Java code generation model*, pages 813–827. Springer.

[173] Ghodrat, M. A., Givargis, T., and Nicolau, A. (2008). Control flow optimization in loops using interval analysis. In *Embedded Systems Week 2008 - 2008 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, CASES'08, October 19, 2008 - October 24, 2008*, pages 157–166. Association for Computing Machinery.

[174] Giese, H. (2005). Towards the model-driven development of self-optimizing mechatronic systems. In *Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme - Workshop on Model-Based Development of Embedded Systems, MBEES 2005, January 10, 2005 - January 14, 2005*, pages 11–22. TU Clausthal.

[175] Girschick, M. (2008). Integrating template based code generation into graphical model transformation. In *Modellierung*, pages 27–40.

[176] Gopinath, V., Sprinkle, J., and Lysecky, R. (2011). Modeling of data adaptable reconfigurable embedded systems. In *Engineering of Computer Based Systems (ECBS), 2011 18th IEEE International Conference and Workshops on*, pages 276–283.

[177] Gračanin, D., Singh, H. L., Bohner, S. A., and Hinchey, M. G. (2005). Model-driven architecture for agent-based systems. *Formal Approaches to Agent-Based Systems*, pages 249–261.

[178] Graichen, C. and D'Amato, F. (2011). Adding code generation to develop a simulation platform. In *Systems, Applications and Technology Conference (LISAT), 2011 IEEE Long Island*, pages 1–6.

[179] Graw, G. and Herrmann, P. (2004). Generation and enactment of controllers for business architectures using mda. *Software Architecture*, pages 148–166.

[180] Greifenberg, T., Hölldobler, K., Kolassa, C., Look, M., Nazari, P. M. S., Müller, K., Perez, A. N., Plotnikov, D., Reiss, D., and Roth, A. e. a. (2015). Integration of handwritten and generated object-oriented code. *Model-Driven Engineering and Software Development*, pages 112–132.

[181] Greifenberg, T., Müller, K., Roth, A., Rumpe, B., Schulze, C., and Wortmann, A. (2016). Modeling variability in template-based code generators for product line engineering. *arXiv preprint arXiv:1606.02903*.

[182] Groce, A. and Pinto, J. (2015). A little language for testing. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9058:204–218.

[183] Groher, I. and Voelter, M. (2009). Aspect-oriented model-driven software product line engineering. *Transactions on Aspect-Oriented Software Development VI*, pages 111–152.

[184] Grow, M., Kim, D., and Kim, Y. (2004). Template-based automatic data flow code generation for mediaprocessors. *Microprocessors and Microsystems*, 28(2):77–84.

[185] Grunske, L., Geiger, L., Zündorf, A., Van Eetvelde, N., Van Gorp, P., and Varro, D. (2005). Using graph transformation for practical model-driven software engineering. *Model-Driven Software Development*, pages 91–117.

[186] Guana, V. and Stroulia, E. (2014). Chaintracker, a model-transformation trace analysis tool for code-generation environments. *Theory and Practice of Model Transformations*, pages 146–153.

[187] Guduvan, A.-R., Waeselynck, H., Wiels, V., Durrieu, G., Fusero, Y., and Schieber, M. (2013). A meta-model for tests of avionics embedded systems. In *International Conference on Model-Driven Engineering and Software Development*, pages 5–13.

[188] Guerra, E., de Lara, J., Wimmer, M., Kappel, G., Kusel, A., Retschitzegger, W., Schönböck, J., and Schwinger, W. (2013). Automated verification of model transformations based on visual contracts. *Automated Software Engineering*, 20(1):5–46.

[189] Guillou, G. and Babau, J.-P. (2016). *IMOCA: A Model-Based Code Generator for the Development of Multi-platform Marine Embedded Systems*, pages 67–77. Springer International Publishing.

[190] Guo, L. and Roychoudhury, A. (2008). Debugging statecharts via model-code traceability. *Leveraging Applications of Formal Methods, Verification and Validation*, pages 292–306.

[191] Han, D., Xu, S., Chen, L., and Huang, L. (2011). Pads: A pattern-driven stencil compiler-based tool for reuse of optimizations on gpgpus. In *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, pages 308–315.

[192] Hartmann, H., Keren, M., Matsinger, A., Rubin, J., Trew, T., and Yatzkar-Haham, T. (2010). Using mda for integration of heterogeneous components in software supply chains. *Software Product Lines: Going Beyond*, pages 2313–2330.

[193] Haschemi, S. and Weißleder, S. (2010). A generic approach to run mutation analysis. *Testing - Practice and Research Techniques*, pages 155–164.

[194] Hastings, S., Oster, S., Langella, S., Ervin, D., Kurc, T., and Saltz, J. (2007). Introduce: An open source toolkit for rapid development of strongly typed grid services. *Journal of Grid Computing*, 5(4):407–427.

[195] Hatzisymeon, G., Houssos, N., Andreadis, D., and Samoladas, V. (2005). An architecture for implementing application interoperation with heterogeneous systems. *Distributed Applications and Interoperable Systems*, pages 194–205.

[196] Haubold, T., Beier, G., Golubski, W., and Herbig, N. (2010). The genesez approach to model-driven software development. *Advanced Techniques in Computing Sciences and Software Engineering*, pages 395–400.

[197] Hauck, M., Kuperberg, M., Huber, N., and Reussner, R. (2014). Deriving performance-relevant infrastructure properties through model-based experiments with ginpex. *Software & Systems Modeling*, 13(4):1345–1365.

[198] Haustein, S. and Pleumann, J. (2005). A model-driven runtime environment for web applications. *Software & Systems Modeling*, 4(4):443–458.

[199] Hebig, R., Giese, H., Stallmann, F., and Seibel, A. (2013). On the complex nature of mde evolution. pages 436–453.

[200] Heidenreich, F., Johannes, J., Karol, S., Seifert, M., and Wende, C. (2013). Model-based language engineering with emftext. *Generative and Transformational Techniques in Software Engineering IV*, pages 322–345.

[201] Heidenreich, F., Johannes, J., Seifert, M., and Wende, C. (2010). Closing the gap between modelling and java. *Software Language Engineering*, pages 374–383.

[202] Helman, T. and Fertalj, K. (2004). Application generator based on parameterized

templates. In *Information Technology Interfaces, 2004. 26th International Conference on*, volume Vol.1, pages 151–157.

[203] Hemel, Z., Kats, L. C., Groenewegen, D. M., and Visser, E. (2010). Code generation by model transformation: a case study in transformation modularity. *Software & Systems Modeling*, 9(3):375–402.

[204] Hemel, Z. and Visser, E. (2010). Pil: A platform independent language for retargetable dsls. *Software Language Engineering*, pages 224–243.

[205] Hendrikx, K., Olivie, H., and Duval, E. (2003). Generative development of object-oriented frameworks. *Technology of Object-Oriented Languages, Systems and Architectures*, pages 31–43.

[206] Heradio, R., Cerrada, J. A., Lopez, J. C., and Coz, J. R. (2009). Code generation with the exemplar flexibilization language. *Electronic Notes in Theoretical Computer Science*, 238(2):25–34.

[207] Hermans, F., Pinzger, M., and Van Deursen, A. (2009). Domain-specific languages in practice: A user study on the success factors. In *Model Driven Engineering Languages and Systems*, pages 423–437.

[208] Hill, J. H., Tambe, S., and Gokhale, A. (2007). Model-driven engineering for development-time qos validation of component-based software systems. In *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, ECBS*, pages 307–316. Institute of Electrical and Electronics Engineers Inc.

[209] Hills, M., Klint, P., and Vinju, J. J. (2013). Meta-language support for type-safe access to external resources. *Software Language Engineering*, pages 372–391.

[210] Hinkel, G., Denninger, O., Krach, S., and Groenda, H. (2016). *Experiences with Model-Driven Engineering in Neurorobotics*, pages 217–228. Springer International Publishing, Cham.

[211] Hohenstein, U. and Elsner, C. (2015). A case study on model-driven development and aspect-oriented programming: Benefits and liabilities. *Software Technologies*, pages 269–290.

[212] Hoisl, B., Sobernig, S., and Strembeck, M. (2013). Higher-order rewriting of model-to-text templates for integrating domain-specific modeling languages. In *1st International Conference on Model-Driven Engineering and Software Development, MODELSWARD*, pages 49–61. INSTICC Press.

[213] Huang, S. S., Zook, D., and Smaragdakis, Y. (2005). Statically safe program generation with safegen. *Generative Programming and Component Engineering*, pages 309–326.

[214] Huang, S. S., Zook, D., and Smaragdakis, Y. (2008). Domain-specific languages and program generation with meta-aspectj. *ACM Transactions on Software Engineering and Methodology*, 18(2):6.

[215] Huiqing, L. and Thompson, S. (2012). Automated api migration in a user-extensible refactoring tool for erlang programs. In *2012 27th IEEE/ACM International Conference on Automated Software Engineering (ASE), 3-7 Sept. 2012*, pages 294–297. IEEE.

[216] Huong, P. V. and Binh, N. N. (2012). An approach to design embedded systems by multi-objective optimization. In *Advanced Technologies for Communications (ATC), 2012 International Conference on*, pages 165–169.

[217] Huy, T. D., Binh, N. T., and Ngoc, N. S. (2016). *Modeling Multidimensional Data Cubes Based on MDA (Model-Driven Architecture)*, pages 85–97. Springer International Publishing.

[218] Iliasov, A. (2003). Templates-based portable just-in-time compiler. *SIGPLAN Notices*, 38(8):37–43.

[219] Inostroza, P. and Van Der Storm, T. (2014). The ttc 2014 fixml case: Rascal solution. In *TCC@ STAF*, volume 1305, pages 47–51.

[220] Jaber, M., Falcone, Y., Dak-Al-Bab, K., Abou-Jaoudeh, J., and El-Katerji, M. (2016). A high-level modeling language for the efficient design, implementation, and testing of android applications. *International Journal on Software Tools for Technology Transfer*, pages 1–18.

[221] Jain, V., Kumar, A., and Panda, P. R. (2011). A sysml profile for development and early validation of tlm 2.0 models. *Modelling Foundations and Applications*, pages 299–311.

[222] Janin, L. and Edwards, D. (2007). Csp transactors for asynchronous transaction level modeling and ip reuse. In *International Conference on Computational Science and its Applications, ICCSA 2007*, volume 4707 LNCS, pages 154–168. Springer Berlin Heidelberg.

[223] Jannach, D. and Kreutler, G. (2004). A knowledge-based framework for the rapid development of conversational recommenders. *Web Information Systems - WISE 2004*, pages 390–402.

[224] Janota, M., Fairmichael, F., Holub, V., Grigore, R., Charles, J., Cochran, D., and Kiniry, J. R. (2009). Clops: A dsl for command line options. *Domain-Specific Languages*, pages 187–210.

[225] Jaouadi, I., Ben Djemaa, R., and Ben-Abdallah, H. (2016). A model-driven development approach for context-aware systems. *Software & Systems Modeling*, pages 1–27.

[226] Jayasinghe, D., Swint, G., Malkowski, S., Li, J., Wang, Q., Park, J., and Pu, C. (2012). Expertus: A generator approach to automate performance testing in iaas clouds. In *2012 IEEE 5th International Conference on Cloud Computing, CLOUD*, pages 115–122. IEEE Computer Society.

[227] Jhang, J., Chung, J.-Y., and Chang, C. K. (2004). Towards increasing web application productivity. In *Applied Computing 2004 - Proceedings of the 2004 ACM Symposium on Applied Computing, March 14, 2004 - March 17, 2004*, volume 2, pages 1677–1681.

[228] Jia, L., He, X., and Dong, Y. (2012). Designing universal template for database application system based on abstract factory. In *2012 International Conference on Computer Science and Information Processing (CSIP)*, pages 1167–70. IEEE.

[229] Jia, X. and Jones, C. (2013). Cross-platform application development using axiom as an agile model-driven approach. *Software and Data Technologies*, pages 36–51.

[230] Jia, X., Steele, A., Qin, L., Liu, H., and Jones, C. (2007). Executable visual software modeling—the zoom approach. *Software Quality Journal*, 15(1):27–51.

[231] Jiang, L.-Y., Wang, R.-C., and Wang, H.-Y. (2006). Code generation framework for grid development. *Journal of China Universities of Posts and Telecommunications*, 13(2):39–42.

[232] Johns, M. (2013). Preparedjs: Secure script-templates for javascript. *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 102–121.

[233] Jones, C. and Jia, X. (2015). Using a domain specific language for lightweight model-driven development. *Evaluation of Novel Approaches to Software Engineering*, pages 46–62.

[234] Jorges, S. (2013). Construction and evolution of code generators: A model-driven and service-oriented approach. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7747:1–265.

[235] Jörges, S., Lamprecht, A.-L., Margaria, T., Naujokat, S., and Steffen, B. (2016). *Synthesis from a Practical Perspective*, pages 282–302. Springer International Publishing.

[236] Jörges, S., Lamprecht, A.-L., Margaria, T., Schaefer, I., and Steffen, B. (2012). A constraint-based variability modeling framework. *International Journal on Software Tools for Technology Transfer*, 14(5):511–530.

[237] Jörges, S., Margaria, T., and Steffen, B. (2008). Genesys: service-oriented construction of property conform code generators. *Innovations in Systems and Software Engineering*, 4(4):361–384.

[238] Jörges, S. and Steffen, B. (2014). Back-to-back testing of model-based code generators. *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, pages 425–444.

[239] Jörges, S., Steffen, B., and Margaria, T. (2011). Building code generators with genesys: A tutorial introduction. *Generative and Transformational Techniques in Software Engineering III*, pages 364–385.

[240] Jugel, U. (2010). Generating smart wrapper libraries for arbitrary apis. *Software Language Engineering*, pages 354–373.

[241] Jugel, U. and Preucner, A. (2011). A case study on api generation. In *System Analysis and Modeling: About Models*, volume 6598 LNCS, pages 156–172.

[242] Kallel, S., Charfi, A., Mezini, M., and Jmaiel, M. (2007). Combining formal methods and aspects for specifying and enforcing architectural invariants. *Coordination Models and Languages*, pages 211–230.

[243] Kallel, S., Kacem, M. H., and Jmaiel, M. (2012). Modeling and enforcing invariants of dynamic software architectures. *Software & Systems Modeling*, 11(1):127–149.

[244] Kallel, S., Loulou, M., Rekik, M., and Kacem, A. H. (2013). Mda-based approach for implementing secure mobile agent systems. *Agent-Oriented Software Engineering XIII*, pages 56–72.

[245] Kalnina, E. and Kalnins, A. (2009). Dsl tool development with transformations and static mappings. *Models in Software Engineering*, pages 356–370.

[246] Kardas, G., Ekinci, E. E., Afsar, B., Dikenelli, O., and Topaloglu, N. Y. (2009). Modeling tools for platform specific design of multi-agent systems. *Multiagent System Technologies*, pages 202–207.

[247] Kerer, C. and Kirda, E. (2001). Layout, content and logic separation in web engineering. *Web Engineering*, pages 135–147.

[248] Kilgo, P., Syriani, E., and Anderson, M. (2012). A visual modeling language for rdis and ros nodes using atom3. *Simulation, Modeling, and Programming for Autonomous Robots*, pages 125–136.

[249] Kim, S., Kim, R. Y. C., and Park, Y. B. (2016). Software vulnerability detection

methodology combined with static and dynamic analysis. *Wireless Personal Communications*, 89(3):777–793.

[250] Kim, W. Y., Son, H. S., Kim, J. S., and Kim, R. Y. C. (2011a). Adapting model transformation approach for android smartphone application. In *3rd International Conference on Advanced Communication and Networking, ACN*, volume 199 CCIS, pages 421–429. Springer Verlag.

[251] Kim, W. Y., Son, H. S., and Kim, R. Y. C. (2011b). Design of code template for automatic code generation of heterogeneous smartphone application. In *Advanced Communication and Networking*, volume 199 CCIS, pages 292–297.

[252] Kindler, E. (2009). Model-based software engineering and process-aware information systems. *Transactions on Petri Nets and Other Models of Concurrency II*, pages 27–45.

[253] Klarl, A., Cichella, L., and Hennicker, R. (2015). From helena ensemble specifications to executable code. *Formal Aspects of Component Software*, pages 183–190.

[254] Klint, P., Van Der Storm, T., and Vinju, J. (2011). Easy meta-programming with rascal. *Generative and Transformational Techniques in Software Engineering III*, pages 222–289.

[255] Kokar, M., Baclawski, K., and Gao, H. (2006). Category theory-based synthesis of a higher-level fusion algorithm: an example. In *2006 9th International Conference on Information Fusion*, pages 1–8.

[256] Kolassa, C., Look, M., Müller, K., Roth, A., Reiß, D., and Rumpe, B. (2016). Tunit-unit testing for template-based code generators. *arXiv preprint arXiv:1606.04682*.

[257] Kolovos, D. S., García-Domínguez, A., Rose, L. M., and Paige, R. F. (2015). Eugenia: towards disciplined and automated development of gmf-based graphical model editors. *Software & Systems Modeling*, pages 1–27.

[258] Kolovos, D. S., Paige, R. F., and Polack, F. A. (2005). An agile and extensible code generation framework. *Extreme Programming and Agile Processes in Software Engineering*, pages 226–229.

[259] Kovesdan, G., Asztalos, M., and Lengyel, L. (2014). Polymorphic templates a design pattern for implementing agile model-to-text transformations. In *XM 2014–Extreme Modeling Workshop*, volume 1239, pages 32–41.

[260] Kövi, A. and Varró, D. (2007). An eclipse-based framework for ais service configurations. *Service Availability*, pages 110–126.

[261] Kovse, J. and Gebauer, C. (2004). Vs-gen: A case study of a product line for versioning systems. In *3rd International Conference on Generative Programming and Component Engineering, GPCE 2004, October 24, 2004 - October 28, 2004*, volume 3286, pages 396–415. Springer Verlag.

[262] Krahn, H. and Rumpe, B. (2006). *Techniques for lightweight generator refactoring*, pages 437–446. Springer.

[263] Křikava, F., Collet, P., and France, R. B. (2014). Sigma: Scala internal domain-specific languages for model manipulations. In *Model Driven Engineering Languages and Systems*, pages 569–585.

[264] Kristensen, L. M. and Veiset, V. (2016). *Transforming CPN Models into Code for TinyOS: A Case Study of the RPL Protocol*, pages 135–154. Springer International Publishing.

[265] Kudlur, M. and Mahlke, S. (2008). Orchestrating the execution of stream programs on multicore platforms. In *2008 ACM SIGPLAN Conference on Programming Language Design and Implementation 2008, PLDI'08, June 7, 2007 - June 13, 2007*, pages 114–124. Association for Computing Machinery.

[266] Kulesza, R., Meira, S. R. L., Ferreira, T. P., Lívio, A., Filho, G. L. S., Neto, M. C. M., and Santos, C. A. S. (2011). A model-driven development approach to integration of web services and interactive applications: A case study in a digital tv platform. In *17th Brazilian Symposium on Multimedia and the Web, WebMedia 2011, co-located with 25th Brazilian Symposium on Database, SBBD 2011*, pages 34–41. Universidade Federal de Santa Catarina.

[267] Kulesza, U., Alves, V., Garcia, A., Neto, A. C., Cirilo, E., De Lucena, C. J., and Borba, P. (2007). Mapping features to aspects: A model-based generative approach. *Early Aspects: Current Challenges and Future Directions*, pages 155–174.

[268] Kulesza, U., Garcia, A., Lucena, C., and Alencar, P. (2005). A generative approach for multi-agent system development. *Software Engineering for Multi-Agent Systems III*.

[269] Kulkarni, V. (2013). Model driven software development. *Modelling Foundations and Applications*, pages 220–235.

[270] Kulkarni, V., Barat, S., and Ramteerthkar, U. (2011). Early experience with agile methodology in a model-driven approach. In *Model Driven Engineering Languages and Systems*, pages 578–590.

[271] Kundu, D., Samanta, D., and Mall, R. (2013). Automatic code generation from unified modelling language sequence diagrams. *IET Software*, 7(1):12–28.

[272] Lachgar, M. and Abdali, A. (2016). Modeling and generating native code for cross-platform mobile applications using dsl. *Intelligent Automation & Soft Computing*, 0:1–14.

[273] Lamancha, B. P., Reales, P., Polo, M., and Caivano, D. (2011a). *Model-Driven Test Code Generation*, pages 155–168. Springer.

[274] Lamancha, B. P., Usaola, M. P., and Velthius, M. P. (2011b). A model based testing approach for model-driven development and software product lines. *Evaluation of Novel Approaches to Software Engineering*, pages 193–208.

[275] Lau, K.-K. and Ukis, V. (2006). Automatic control flow generation from software architectures. *Software Composition*, pages 323–338.

[276] Lee, J., Park, J., Yoo, G., and Lee, E. (2010). Goal-based automated code generation in self-adaptive system. *Journal of Computer Science and Technology*, 25(6):1118–1129.

[277] Lee, K., Lee, W., Kim, J., and Chong, K. (2006). A technique for code generation of usn applications based on nano-qplus. *Computational Science - ICCS 2006*, pages 902–909.

[278] Lee, W., Kim, J., and Kang, J. (2012). A framework for automated construction of node software using low-level attributes in usn application development. *International Journal of Software Engineering and Knowledge Engineering*, 22(5):675–693.

[279] Leonard, E. I. and Heitmeyer, C. L. (2003). Program synthesis from formal requirements specifications using apts. *Higher-Order and Symbolic Computation*, 16(1 - 2):63–92.

[280] Leone, S., de Spindler, A., and McLeod, D. (2013). Model-driven composition of information systems from shared components and connectors. *On the Move to Meaningful Internet Systems: OTM 2013 Conferences*, pages 204–221.

[281] Lethbridge, T. C., Abdelzad, V., Husseini Orabi, M., Husseini Orabi, A., and Adesina, O. (2016). *Merging Modeling and Programming Using Umple*, pages 187–197. Springer International Publishing.

[282] Li, D., Li, F., Huang, X., Lai, Y., and Zheng, S. (2010). A model based integration framework for computer numerical control system development. *Robot. Comput.-Integr. Manuf.*, 26(4):333–343.

[283] Li, H. and Thompson, S. (2012). *A Domain-Specific Language for Scripting Refactorings in Erlang*, pages 501–515. Springer Berlin Heidelberg.

[284] Li, S., Malik, J., Liu, S., and Hemani, A. (2013). A code generation method for system-level synthesis on asic, fpga and manycore cgra. In *1st International Workshop on Many-Core Embedded Systems, MES 2013, in Conjunction with the 40th Annual IEEE/ACM International Symposium on Computer Architecture, ISCA 2013, June 24, 2013 - June 24, 2013*, pages 25–32. Association for Computing Machinery.

[285] Li, W., Lee, Y.-H., Tsai, W.-T., Xu, J., Son, Y.-S., Park, J.-H., and Moon, K.-D. (2012). Service-oriented smart home applications: composition, code generation, deployment, and execution. *Service Oriented Computing and Applications*, 6(1):65–79.

[286] Liao, K., Kisuule, P., Ehrlinger, J., and Dai, J. (2009). Autogeneration of database applications from xml metadata for web-based data entry. In *Computer and Information Science, 2009. ICIS 2009. Eighth IEEE/ACIS International Conference on*, pages 718–723.

[287] Liem, I. and Nugroho, Y. (2008). An application generator framelet. In *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2008. SNPD '08. Ninth ACIS International Conference on*, pages 794–9.

[288] Lihatsky, I., Doroshenko, A., and Zhereb, K. (2013). A template-based method to create efficient and customizable object-relational transformation components. *Information Systems: Methods, Models, and Applications*, pages 178–184.

[289] Linaje, M., Preciado, J. C., Morales-Chaparro, R., Rodríguez-Echeverría, R., and Sánchez-Figueroa, F. (2009). Automatic generation of rias using rux-tool and webratio. *Web Engineering*, pages 501–504.

[290] Linehan, E. and Clarke, Siobhá, n. (2012). An aspect-oriented, model-driven approach to functional hardware verification. *J. Syst. Archit.*, 58(5):195–208.

[291] Lohmann, M., Sauer, S., and Schattkowsky, T. (2003). Progum-web: Tool support for model-based development of web applications. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 2863, pages 101–105. Springer Berlin Heidelberg.

[292] Lohmann, M. and Schattkowsky, T. (2002). Rapid development of modular dynamic web sites using uml. *?UML? 2002—The Unified Modeling Language*, pages 336–350.

[293] López, S. A., Bonino, D., and Corno, F. (2014). Template-based ontology population for smart environments configuration. *Service-Oriented Computing - ICSOC 2013 Workshops*, pages 271–278.

[294] Lora, M., Martinelli, F., and Fummi, F. (2015). Hardware synthesis from software-oriented uml descriptions. In *15th International Microprocessor Test and*

*Verification Workshop, MTV 2014, December 15, 2014 - December 16, 2014*, volume 2015-April of *Proceedings - International Workshop on Microprocessor Test and Verification*, pages 33–38. Institute of Electrical and Electronics Engineers Inc.

[295] Lussenburg, V., Van Der Storm, T., Vinju, J., and Warmer, J. (2010). Mod4j: A qualitative case study of model-driven software development. In *Model Driven Engineering Languages and Systems*, pages 346–360.

[296] Ma, K., Yang, B., and Wang, H. (2010). A formalizing hybrid model transformation approach for collaborative system. In *Computer Supported Cooperative Work in Design (CSCWD), 2010 14th International Conference on*, pages 71–76.

[297] Ma, M., Meissner, M., and Hedrich, L. (2012). A case study: Automatic topology synthesis for analog circuit from an asdex specification. In *Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD), 2012 International Conference on*, pages 9–12.

[298] Magdalenic, I., Radosevic, D., and Kermek, D. (2011). Implementation model of source code generator. *Journal of Communication Software &amp; Systems*, 7(2):71–9.

[299] Mahapatra, S. (2011). Enabling modular design platforms using variants in model-based design. In *AIAA Modeling and Simulation Technologies Conference 2011, August 8, 2011 - August 11, 2011*, AIAA Modeling and Simulation Technologies Conference 2011, pages 904–915. American Institute of Aeronautics and Astronautics Inc.

[300] Malohlava, M., Plasil, F., Bures, T., and Hnetynka, P. (2013). Interoperable domain-specific languages families for code generation. *Software: Practice and Experience*, 43(5):1–21.

[301] Manley, R. and Gregg, D. (2010). A program generator for intel aes-ni instructions. *Progress in Cryptology - INDOCRYPT 2010*, pages 311–327.

[302] Mao, F., Cai, X., Shen, B., Xia, Y., and Jin, B. (2016). Operational pattern based code generation for management information system: An industrial case study. In *2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 425–430.

[303] Maoz, S. and Harel, D. (2006). From multi-modal scenarios to code: Compiling lscs into aspectj. In *14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, pages 219–230. Association for Computing Machinery.

[304] Maoz, S., Harel, D., and Kleinbort, A. (2011). A compiler for multimodal scenarios: Transforming lscs into aspectj. *ACM Transactions on Software Engineering and Methodology*, 20(4).

[305] Marand, E. A., Marand, E. A., and Challenger, M. (2015). Dsml4cp: A domain-specific modeling language for concurrent programming. *Computer Languages, Systems & Structures*, pages 319–341.

[306] Marin, M., van Deursen, A., Moonen, L., and van der Rijst, R. (2009). An integrated crosscutting concern migration strategy and its semi-automated application to jhotdraw. *Automated Software Engineering*, 16(2):323–356.

[307] Martins, D., Campos, F. O., Ciuffo, L. N., Oliveira, R. S., Amorim, R. M., Vieira, V., Ebecken, N. F., Barbosa, C. B., and Dos Santos, R. W. (2007). A computational

framework for cardiac modeling based on distributed computing and web applications. *High Performance Computing for Computational Science - VECPAR 2006*, pages 544–555.

[308] Matsumoto, K., Mizuno, T., and Mori, N. (2013). A model-driven development method for applying to management information systems. *Knowledge Discovery, Knowledge Engineering and Knowledge Management*, pages 197–207.

[309] Mazzeranghi, D. (2008). Panda: A pattern-based programming system for automatic code generation. *Journal of Object Technology*, 7(4):67–99.

[310] McNaughton, M., Redford, J., Schaeffer, J., and Szafron, D. (2003). Pattern-based ai scripting using scriptease. In *16th Conference of the Canadian Society for Computational Studies of Intelligence, June 11, 2003 - June 13, 2003*, volume 2671, pages 35–49. Springer Verlag.

[311] Mezini, M. and Ostermann, K. (2005). A comparison of program generation with aspect-oriented programming. *Unconventional Programming Paradigms*, pages 342–354.

[312] Miller, T., Freitas, L., Malik, P., and Utting, M. (2005). Czt support for z extensions. In *5th International Conference on Integrated Formal Methods, IFM 2005, November 29, 2005 - December 2, 2005*, volume 3771 LNCS, pages 227–245. Springer Verlag.

[313] Mizuta, S. and Huang, R. (2005). Automation of grid service code generation with andromda for gt3. In *19th International Conference on Advanced Information Networking and Applications (AINA'05) Volume 1 (AINA papers)*, volume vol.2, pages 417–20.

[314] Modesti, P. (2016). *AnBx: Automatic Generation and Verification of Security Protocols Implementations*, pages 156–173. Springer International Publishing.

[315] Moha, N., Guéhéneuc, Y.-G., Le Meur, A.-F., and Duchien, L. (2008). A domain analysis to specify design defects and generate detection algorithms. *Fundamental Approaches to Software Engineering*, pages 276–291.

[316] Mohan, R. and Kulkarni, V. (2009). Model driven development of graphical user interfaces for enterprise business applications - experience, lessons learnt and a way forward. In *Model Driven Engineering Languages and Systems*, pages 307–321.

[317] Mohr, F. and Walther, S. (2014). Template-based generation of semantic services. *Software Reuse for Dynamic Systems in the Cloud and Beyond*, pages 188–203.

[318] Mokaddem, C. e., Sahraoui, H., and Syriani, E. (2016). *Towards Rule-Based Detection of Design Patterns in Model Transformations*, pages 211–225. Springer International Publishing.

[319] Molnár, L., Pongrácz, G., Enyedi, G., Kis, Z. L., Csikor, L., Juhász, F., Kőrösi, A., and Rétvári, G. (2016). Dataplane specialization for high-performance openflow software switching. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 539–552. ACM.

[320] Morales, Z., Magańa, C., Aguilar, J. A., Zaldívar-Colado, A., Tripp-Barba, C., Misra, S., Garcia, O., and Zurita, E. (2016). *A Baseline Domain Specific Language Proposal for Model-Driven Web Engineering Code Generation*, pages 50–59. Springer International Publishing.

[321] Moreira, T. G., Wehrmeister, M. A., Pereira, C. E., Pétin, J.-F., and Levrat, E. (2010). Generating vhdl source code from uml models of embedded systems. *Distributed, Parallel and Biologically Inspired Systems*, pages 125–136.

[322] Moreira de Sousa, L. and Rodrigues da Silva, A. (2016). A domain specific language for spatial simulation scenarios. *GeoInformatica*, 20(1):117–149.

[323] Mtsweni, J. (2012). Exploiting uml and acceleo for developing semantic web services. In *Internet Technology And Secured Transactions, 2012 International Conference for*, pages 753–758.

[324] Muñoz, J. and Pelechano, V. (2006). Applying software factories to pervasive systems: A platform specific framework. In *8th International Conference on Enterprise Information Systems, ICEIS 2006*, volume ISAS, pages 337–342.

[325] Mueller-Glaser, K. D., Reichmann, C., Kuehl, M., and Benz, S. (2006). Quality assurance and certification of software modules in safety critical automotive electronic control units using a case-tool integration platform. *Automotive Software - Connected Services in Mobile Networks*, pages 15–30.

[326] Muller, P.-A., Fondement, F., Fleurey, F., Hassenforder, M., Schnekenburger, R., Gérard, S., and Jézéquel, J.-M. (2008). Model-driven analysis and synthesis of textual concrete syntax. *Software & Systems Modeling*, 7(4):423–441.

[327] Muller, P.-A., Studer, P., Fondement, F., and Bézivin, J. (2005). Platform independent web application modeling and development with netsilon. *Software & Systems Modeling*, 4(4):424–442.

[328] Muller, T. C.-A. (2012). Exploiting model driven technology: a tale of two startups. *Software & Systems Modeling*, 11(4):481–493.

[329] Mulo, E., Zdun, U., and Dustdar, S. (2013). Domain-specific language for event-based compliance monitoring in process-driven soas. *Service Oriented Computing and Applications*, 7(1):59–73.

[330] Muresan, M. and Pitica, D. (2009). Simulating embedded targets for efficient code implementation. In *2009 32nd International Spring Seminar on Electronics Technology*, pages 1–4.

[331] Naujokat, S., Traonouez, L.-M., Isberner, M., Steffen, B., and Legay, A. (2014). Domain-specific code generator modeling: A case study for multi-faceted concurrent systems. *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, pages 481–498.

[332] Naumann, U. and Utke, J. (2005). Source templates for the automatic generation of adjoint code through static call graph reversal. In *Computational Science - ICCS 2005*, pages 338–46.

[333] Nazari, P. M. S., Roth, A., and Rumpe, B. (2016). An extended symbol table infrastructure to manage the composition of output-specific generator information. *arXiv preprint arXiv:1606.00585*.

[334] Nestor Ribeiro, A. and Rogério Araújo, C. (2016). *An Automated Model Based Approach to Mobile UI Specification and Development*, pages 523–534. Springer International Publishing.

[335] Nguyen, K. D., Sun, Z., Thiagarajan, P., and Wong, W.-F. (2005). Model-driven soc design: The uml-systemc bridge. *UML for SOC Design*, pages 175–197.

[336] Nunes, I., Cirilo, E., de Lucena, C. J., Sudeikat, J., Hahn, C., and Gomez-Sanz, J. J. (2011). A survey on the implementation of agent oriented specifications. *Agent-Oriented Software Engineering X*, pages 169–179.

[337] Núñez-Valdez, E. R., García-Díaz, V., Lovelle, J. M. C., Achaerandio, Y. S., and González-Crespo, R. (2016). A model-driven approach to generate and deploy

videogames on multiple platforms. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–13.

[338] Ogunyomi, B., Rose, L. M., and Kolovos, D. S. (2014). On the use of signatures for source incremental model-to-text transformation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8767:84–98.

[339] Ogunyomi, B., Rose, L. M., and Kolovos, D. S. (2015). Property access traces for source incremental model-to-text transformation. *Modelling Foundations and Applications*, pages 187–202.

[340] Oh, H., Dutt, N., and Ha, S. (2005). Single appearance schedule with dynamic loop count for minimum data buffer from synchronous dataflow graphs. In *Proceedings of the 2005 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, CASES '05, pages 157–165. ACM.

[341] O'Halloran, C. (2009). Guess and verify - back to the future. *FM 2009: Formal Methods*, pages 23–32.

[342] Oldevik, J., Neple, T., Grønmo, R., Aagedal, J., and Berre, A.-J. (2005). Toward standardised model to text transformations. *Model Driven Architecture - Foundations and Applications*, pages 239–253.

[343] Oppenheimer, F., Zhang, D., and Nebel, W. (2001). Modelling communication interfaces with comix. *Reliable SoftwareTechnologies—Ada-Europe 2001*, pages 337–348.

[344] Ortiz-Cornejo, A., Cuayahuitl, H., and Perez-Corona, C. (2006). Wisbuilder: A framework for facilitating development of web-based information systems. In *Elec-*

*tronics, Communications and Computers, 2006. CONIELECOMP 2006. 16th International Conference on*, pages 46–46.

[345] Paige, R. F., Kolovos, D. S., Rose, L. M., Matragkas, N., and Williams, J. R. (2013). Model management in the wild. *Generative and Transformational Techniques in Software Engineering IV*, pages 197–218.

[346] Palyart, M., Lugato, D., Ober, I., and Bruel, J.-M. (2012). Mde4hpc: An approach for using model-driven engineering in high-performance computing. *SDL 2011: Integrating System and Software Modeling*, pages 247–261.

[347] Parr, T. (2006). Web application intel-nationalization and localization in action. In *ICWE'06: 6th International Conference on Web Engineering, July 11, 2006 - July 14, 2006*, pages 64–70. Association for Computing Machinery.

[348] Pastorino, R., Cosco, F., Naets, F., Desmet, W., and Cuadrado, J. (2016). Hard real-time multibody simulations using arm-based embedded systems. *Multibody System Dynamics*, 37(1):127–143.

[349] Pavón, J., Gómez-Sanz, J., and Fuentes, R. (2006). Model driven development of multi-agent systems. In Rensink, A. and Warmer, J., editors, *Model Driven Architecture–Foundations and Applications: Second European Conference, ECMFA. Proceedings*, volume 4066 of *LNCS*, pages 284–298. Springer Berlin Heidelberg.

[350] Pękala, J. (2016). *Data Transformation Using Custom Class Generator as Part of Systems Integration in Manufacturing Company*, pages 397–409. Springer International Publishing.

[351] Pelcat, M., Aridhi, S., Piat, J., and Nezan, J.-F. (2013). Generating code from lte models. *Physical Layer Multi-Core Prototyping*, pages 173–196.

[352] Perry, T., Walke, R., and Benkrid, K. (2011). An extensible code generation framework for heterogeneous architectures based on ip-xact. In *Programmable Logic (SPL), 2011 VII Southern Conference on*, pages 81–86.

[353] Pezze, M. and Wuttke, J. (2009). *Automatic generation of runtime failure detectors from property templates*, pages 223–40. Springer-Verlag.

[354] Pezze, M. and Wuttke, J. (2016). Model-driven generation of runtime checks for system properties. *International Journal on Software Tools for Technology Transfer*, 18(1):1–19.

[355] Phillips, J., Chilukuri, R., Fragoso, G., Warzel, D., and Covitz, P. A. (2006). The cacore software development kit: Streamlining construction of interoperable biomedical information services. *BMC Medical Informatics and Decision Making*, 6:1–16.

[356] Piel, É., Marquet, P., and Dekeyser, J.-L. (2008). Model transformations for the compilation of multi-processor systems-on-chip. *Generative and Transformational Techniques in Software Engineering II*, pages 459–473.

[357] Pinheiro, L. P., Lopes, Y. K., Leal, A. B., and Rosso, R. S. U. (2015). Nadzoru: A software tool for supervisory control of discrete event systems. In *5th IFAC International Workshop on Dependable Control of Discrete Systems, DCDS*, volume 48 of *IFAC Proceedings Volumes (IFAC-PapersOnline)*, pages 182–187. IFAC Secretariat.

[358] Pokahr, A. and Braubach, L. (2009). A survey of agent-oriented development tools. *Multi-Agent Programming:*, pages 289–329.

[359] Portillo, J., Casquero, O., and Marcos, M. (2005). Loose integration of cots tools for the development of real time distributed control systems. *COTS-Based Software Systems*, pages 191–200.

[360] Possatto, M. A. and Lucrédio, D. (2015). Automatically propagating changes from reference implementations to code generation templates. *Information and Software Technology*, 67:65–78.

[361] Poveda, G. and Schumann, R. (2016). *An Ontology-Driven Approach for Modeling a Multi-agent-Based Electricity Market*, pages 27–40. Springer International Publishing.

[362] Preschern, C., Kajtazovic, N., and Kreiner, C. (2012). Applying patterns to model-driven development of automation systems: An industrial case study. In *17th European Conference on Pattern Languages of Programs, EuroPLoP 2012*, pages Springer;Wiley–Blackwell. Association for Computing Machinery.

[363] Prout, A., Atlee, J. M., Day, N. A., and Shaker, P. (2008). Semantically configurable code generation. In *Model Driven Engineering Languages and Systems*, pages 705–720.

[364] Puder, A. (2012). Running android applications without a virtual machine. *Mobile Wireless Middleware, Operating Systems, and Applications*, pages 121–134.

[365] Qiu, X. X. and Cheng, X. (2014). Design and implementation of a software automation development framework for management information system. In *Materials Science, Computer and Information Technology*, volume 989 of *Advanced Materials Research*, pages 4488–4492. Trans Tech Publications.

[366] Quintero, R., Zepeda, L., and Vega, L. (2010). Model driven software development of applications based on web services. *Computational Science and Its Applications - ICCSA 2010*, pages 313–330.

[367] Radosevic, D. and Magdalenic, I. (2011). Python implementation of source code

generator based on dynamic frames. In *MIPRO, 2011 Proceedings of the 34th International Convention*, pages 969–974.

[368] Radosevic, D., Magdalenic, I., and Orehovacki, T. (2011). Error messaging in generative programming. In *Proceedings of the 22nd Central European Conference on Information and Intelligent Systems*, pages 181–186.

[369] Rahmouni, M. and Mbarki, S. (2014). Model-driven generation: From models to mvc2 web applications. *International Journal of Software Engineering and its Applications*, 8(7):73–94.

[370] Rajkovic, P., Petkovic, I., and Jankovic, D. (2015). Benefits of using domain model code generation framework in medical information systems. In *Fourth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications SQAMIA 2015*, volume 1375, pages 45–52.

[371] Regensburger, M., Buckl, C., Knoll, A., and Schrott, G. (2007). Model based development of safety-critical systems using template based code generation. In *Dependable Computing, 2007. PRDC 2007. 13th Pacific Rim International Symposium on*, pages 89–92.

[372] Reis, R. (2007). Educase: automatic system for the development of educational software. In *New Horizons in Education and Eductional Technology. 6th WSEAS International Conference on Education and Educational Technology, 21-23 Nov. 2007*, pages 292–4. WSEAS Press.

[373] Riccobene, E. and Scandurra, P. (2009). Model transformations in the upes/upsoc development process for embedded systems. *Innovations in Systems and Software Engineering*, 5(1):35–47.

[374] Richmond, P. and Romano, D. (2011). *Template-driven agent-based modeling and simulation with CUDA*, pages 313–324. Elsevier Inc.

[375] Riebisch, M., Philippow, I., and Götze, M. (2003). Uml-based statistical test case generation. *Objects, Components, Architectures, Services, and Applications for a Networked World*, pages 394–411.

[376] Rios, J. L. and Machado-Piriz, F. (2007). A case study to evaluate templates metadata for developing application families. In *Advances and Innovations in Systems, Computing Sciences and Software Engineering*, pages 241–246.

[377] Rodrigues, T., Delicato, F. C., Batista, T., Pires, P. F., and Pirmez, L. (2015). An approach based on the domain perspective to develop wsan applications. *Software & Systems Modeling*, pages 1–29.

[378] Rodríguez, F. T., Reina, M., Baptista, F., Usaola, M. P., and Lamancha, B. P. (2013). Automated generation of performance test cases from functional tests for web applications. *Evaluation of Novel Approaches to Software Engineering*, pages 164–173.

[379] Romaniuk, P. (2007). Translator of hierarchical state machine from uml statechart to the event processor pattern. In *Mixed Design of Integrated Circuits and Systems, 2007. MIXDES '07. 14th International Conference on*, pages 684–687.

[380] Rose, L., Guerra, E., De Lara, J., Etien, A., Kolovos, D., and Paige, R. (2013). Genericity for model management operations. *Software & Systems Modeling*, 12(1):201–219.

[381] Rose, L. M., Paige, R. F., Kolovos, D. S., and Polack, F. A. (2008). The epsilon generation language. *Model Driven Architecture - Foundations and Applications*, pages 1–16.

[382] Roubi, S., Erramdani, M., and Mbarki, S. (2016). *Model Driven Architecture as an Approach for Modeling and Generating Graphical User Interface*, pages 651–656. Springer International Publishing.

[383] Roychoudhury, S., Gray, J., and Jouault, F. (2011). A model-driven framework for aspect weaver construction. *Transactions on Aspect-Oriented Software Development VIII*, pages 1–45.

[384] Ruiz-López, T., Rodríguez-Domínguez, C., Rodríguez, M. J., Ochoa, S. F., and Garrido, J. L. (2013). Context-aware self-adaptations: From requirements specification to code generation. *Ubiquitous Computing and Ambient Intelligence. Context-Awareness and Context-Driven Interaction*, pages 46–53.

[385] Rutherford, M. J. and Wolf, A. L. (2003). A case for test-code generation in model-driven systems. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 2830, pages 377–396. Springer Netherlands.

[386] SÃnchez, P., Jiménez, M., Rosique, F., Ã?lvarez, B. A., and Iborra, A. A. (2011). A framework for developing home automation systems: From requirements to code. *Journal of Systems and Software*, 84(6):1008–1021.

[387] Sahai, A., Pu, C., Jung, G., Wu, Q., Yan, W., and Swint, G. S. (2005). Towards automated deployment of built-to-order systems. *Ambient Networks*, pages 109–120.

[388] Sakamoto, K., Tomohiro, K., Hamura, D., Washizaki, H., and Fukazawa, Y. (2013). Pogen: A test code generator based on template variable coverage in gray-box integration testing for web applications. *Fundamental Approaches to Software Engineering*, pages 343–358.

[389] Salazar, E., Alonso, A., de Miguel, M. A., and Juan, A. (2013). A model-based framework for developing real-time safety ada systems. *Reliable Software Technologies - Ada-Europe 2013*, pages 127–142.

[390] Saurabh, A., Dahiya, D., and Mohana, R. (2012). *Maximizing Automatic Code Generation: Using XML Based MDA*, pages 283–293. Springer.

[391] Schaefer, J., Stynes, J., and Kroeger, R. (2008). Model-based performance instrumentation of distributed applications. In *Distributed Applications and Interoperable Systems*, pages 210–23.

[392] Schiffelers, R. R. H., Alberts, W., and Voeten, J. P. M. (2012). Model-based specification, analysis and synthesis of servo controllers for lithoscanners. In *6th International Workshop on Multi-Paradigm Modeling, MPM 2012, October 1, 2012 - October 1, 2012*, Proceedings of the 6th International Workshop on Multi-Paradigm Modeling, MPM 2012, pages 55–60. Association for Computing Machinery.

[393] Schippers, H., Gorp, P. V., and Janssens, D. (2005). Leveraging uml profiles to generate plugins from visual model transformations. *Electronic Notes in Theoretical Computer Science*, 127(3):5–16.

[394] Schloegel, K., Oglesby, D., Engstrom, E., and Bhatt, D. (2003). Composable code generation for model-based development. *Software and Compilers for Embedded Systems*, pages 211–225.

[395] Schneider, V., Deitsch, A., Dulz, W., and German, R. (2016). *Combined Simulation and Testing Based on Standard UML Models*, pages 499–523. Springer International Publishing.

[396] Schoeberl, M., Brooks, C., and Lee, E. A. (2010). Code generation for embedded

java with ptolemy. *Software Technologies for Embedded and Ubiquitous Systems*, pages 155–166.

[397] Schreiber, A. (2006). Automatic generation of wrapper code and test scripts for problem solving environments. *Applied Parallel Computing. State of the Art in Scientific Computing*, pages 680–689.

[398] Seo, Y.-J. and Song, Y.-J. (2006). A study on automatic code generation tool from design patterns based on the xmi. In *Computational Science and Its Applications - ICCSA 2006*, pages 864–72.

[399] Silaghi, R. and Strohmeier, A. (2005). Parallax—an aspect-enabled framework for plug-in-based mda refinements towards middleware. *Model-Driven Software Development*, pages 239–267.

[400] Simonsen, K., Kristensen, L., and Kindler, E. (2013). Generating protocol software from cpn models annotated with pragmatics. In *Formal Methods: Foundations and Applications*, pages 227–42.

[401] Simonsen, K. I. F. (2014). Petricode: A tool for template-based code generation from cpn models. In *Software Engineering and Formal Methods*, volume 8368 LNCS, pages 151–163.

[402] Simonsen, K. I. F. and Kristensen, L. M. (2014). Implementing the websocket protocol based on formal modelling and automated code generation. *Distributed Applications and Interoperable Systems*, pages 104–118.

[403] Sindico, A., Di Natale, M., and Sangiovanni-Vincentelli, A. (2012). An industrial system engineering process integrating model driven architecture and model based design. In *Model Driven Engineering Languages and Systems*, pages 810–826.

[404] Sinha, V. S., Dhoolia, P., Mani, S., and Sinha, S. (2014). Operational abstraction of model transforms. In *7th India Software Engineering Conference, ISEC 2014, February 19, 2014 - February 21, 2014*, ACM International Conference Proceeding Series, pages 3:1–3:10. Association for Computing Machinery.

[405] Siret, N., Wipliez, M., Nezan, J.-F., and Rhatay, A. (2010). Hardware code generation from dataflow programs. In *2010 Conference on Design and Architectures for Signal and Image Processing, DASIP2010, October 26, 2010 - October 28, 2010*, 2010 Conference on Design and Architectures for Signal and Image Processing, DASIP2010, pages 113–120. IEEE Computer Society.

[406] Skene, J. and Emmerich, W. (2005). Engineering runtime requirements-monitoring systems using mda technologies. *Trustworthy Global Computing*, pages 319–333.

[407] Skrobo, D., Milanovic, A., and Srbljic, S. (2006). Performance evaluation of program translation in service-oriented architectures. In *International Conference on Networking and Services 2006, ICNS*, International Conference on Networking and Services 2006, ICNS'06, pages 14–14. Inst. of Elec. and Elec. Eng. Computer Society.

[408] Smaragdakis, Y. (2004). Program generators and the tools to make them. In *Static Analysis. 11th International Symposium, SAS 2004*, Static Analysis. 11th International Symposium, SAS 2004. Proceedings (Lecture Notes in Comput. Sci. Vol.3148), pages 19–20. Springer-Verlag.

[409] Spampinato, D. G. and Puschel, M. (2014). A basic linear algebra compiler. In *12th ACM/IEEE International Symposium on Code Generation and Optimization, CGO 2014, February 15, 2014 - February 19, 2014*, Proceedings of the 12th

ACM/IEEE International Symposium on Code Generation and Optimization, CGO 2014, pages 23–32. Association for Computing Machinery.

[410] Spanoudakis, N. and Moraitis, P. (2009). Gaia agents implementation through models transformation. *Principles of Practice in Multi-Agent Systems*, pages 127–142.

[411] Sridhara, G., Pollock, L., and Vijay-Shanker, K. (2011). Automatically detecting and describing high level actions within methods. In *33rd International Conference on Software Engineering, ICSE 2011*, Proceedings - International Conference on Software Engineering, pages 101–110. IEEE Computer Society.

[412] Steffen, B., Isberner, M., Naujokat, S., Margaria, T., and Geske, M. (2014). Property-driven benchmark generation: synthesizing programs of realistic structure. *International Journal on Software Tools for Technology Transfer*, 16(5):465–479.

[413] Strommer, M. and Wimmer, M. (2008). A framework for model transformation by-example: Concepts and tool support. *Objects, Components, Models and Patterns*, pages 372–391.

[414] Stroulia, E., Bazelli, B., Ng, J. W., and Ng, T. (2015). Wl++: Code generation of multi-platform mobile clients to restful back-ends. In *Mobile Software Engineering and Systems (MOBILESoft), 2015 2nd ACM International Conference on*, pages 136–137.

[415] Sturm, T., von Voss, J., and Boger, M. (2002). Generating code from uml with velocity templates. *?UML? 2002—The Unified Modeling Language*, pages 150–161.

[416] Sun, K., Fryer, D., Brown, A. D., and Goel, A. (2013). Annotation for automation: Rapid generation of file system tools. In *7th Workshop on Programming Languages*

*and Operating Systems, PLOS 2013 - In Conjunction with the 24th ACM Symposium on Operating Systems Principles, SOSP 2013, November 3, 2013 - November 3, 2013*, pages 4:1–4:6. Association for Computing Machinery.

[417] Surla, B. D. and Ivanovic, D. (2012). Using templates for presenting publication references in cris. In *11th International Conference on Current Research Information Systems, CRIS 2012*, pages 61–66. Zeithamlova Milena, ING - Agentura Action M.

[418] Swertz, M. A., Dijkstra, M., Adamusiak, T., van der Velde, J. K., Kanterakis, A., Roos, E. T., Lops, J., Thorisson, G. A., Arends, D., Byelas, G., Muilu, J., Brookes, A. J., de Brock, E. O., Jansen, R. C., and Parkinson, H. (2010). The molgenis toolkit: rapid prototyping of biosoftware at the push of a button. *BMC Bioinformatics*, 11(suppl12):1–9.

[419] Systa, T., Koskimies, K., and Muller, H. (2001). Shimba - an environment for reverse engineering java software systems. *Software - Practice and Experience*, 31(4):371–394.

[420] Taentzer, G. and Carughi, G. T. (2006). A graph-based approach to transform xml documents. In *9th International Conference on Fundamental Approaches to Software Engineering, FASE 2006. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, March 27, 2006 - March 28, 2006*, volume 3922 LNCS of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 48–62. Springer Verlag.

[421] Taguchi, M., Suzuki, T., and Tokuda, T. (2003). A visual approach for generating server page type web applications based on template method. In *Human Centric*

*Computing Languages and Environments, 2003. Proceedings. 2003 IEEE Symposium on*, pages 248–250.

[422] Tamayo, A., Granell, C., Díaz, L., and Huerta, J. (2014). Personalised code generation from large schema sets for geospatial mobile applications. *Computing*, 96(5):355–379.

[423] Tang, P. and Moulliet, R. (2015). Automatic generation of tuners for intel concurrent collections programs.

[424] Tatsubori, M. and Suzumura, T. (2009). Html templates that fly a template engine approach to automated offloading from server to client. In *18th International World Wide Web Conference, WWW 2009*, WWW'09 - Proceedings of the 18th International World Wide Web Conference, pages 951–960. Association for Computing Machinery.

[425] Tichy, P., Kadera, P., Staron, R. J., Vrba, P., and Marik, V. (2012). Multi-agent system design and integration via agent development environment. *Engineering Applications of Artificial Intelligence*, 25(4):846–852.

[426] Tonella, P. and Ricca, F. (2005). Web application slicing in presence of dynamic code generation. *Automated Software Engineering*, 12(2):259–288.

[427] Topalidou-Kyniazopoulou, A., Spanoudakis, N. I., and Lagoudakis, M. G. (2013). A case tool for robot behavior development. *RoboCup 2012: Robot Soccer World Cup XVI*, pages 225–236.

[428] Topçu, O., Durak, U., Oğuztüzün, H., and Yilmaz, L. (2016). *Implementation, Integration, and Testing*, pages 203–230. Springer International Publishing.

[429] Touraille, L., Traoré, M. K., and Hill, D. R. C. (2011). A model-driven software environment for modeling, simulation and analysis of complex systems. In *Proceed-*

*ings of the 2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, TMS-DEVS '11, pages 229–237. Society for Computer Simulation International.

[430] Travkin, O. and Wehrheim, H. (2016). *Verification of Concurrent Programs on Weak Memory Models*, pages 3–24. Springer International Publishing.

[431] Tsai, W., Fan, C., Chen, Y., and Paul, R. (2006). A service-oriented modeling and simulation framework for rapid development of distributed applications. *Simulation Modelling Practice and Theory*, 14(6):725–739.

[432] Ünsal, E. and Sevilgen, F. E. (2007). Bemga: A hla based simulation modeling and development tool. *Advances and Innovations in Systems, Computing Sciences and Software Engineering*, pages 115–119.

[433] Valderas, P., Pelechano, V., and Pastor, O. (2006). Towards an end-user development approach for web engineering methods. In *18th International Conference on Advanced Information Systems Engineering, CAiSE 2006*, pages 528–543. Springer Verlag.

[434] van Reeuwijk, C. (2003). Rapid and robust compiler construction using template-based metacompilation. In *12th International Conference on Compiler Construction, CC 2003 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003*, volume 2622, pages 247–261. Springer Verlag.

[435] Vaupel, S., Taentzer, G., Gerlach, R., and Guckert, M. (2016). Model-driven development of mobile applications for android and ios supporting role-based app variability. *Software & Systems Modeling*, pages 1–29.

[436] Vazhenin, D., Mirenkov, N., and Vazhenin, A. (2011). Movie-based represen-

tation of reduction operations in numerical computing. *Knowledge-Based Systems*, 24(7):977–988.

[437] Viana, M. C., Penteado, R. A., Do Prado, A. F., and Durelli, R. S. (2015). F3t: a tool to support the f3 approach on the development and reuse of frameworks. *Journal of Software Engineering Research and Development*, 3:1–26.

[438] Viana, M. C., Penteado, R. A. D., and Prado, A. F. (2013). *Building Domain-Specific Modeling Languages for Frameworks*, pages 191–206. Springer Berlin Heidelberg.

[439] Vijayakumar, A., Abhishek, D., and Chandrasekaran, K. (2016). *DSL Approach for Development of Gaming Applications*, pages 199–211. Springer India.

[440] Vinogradov, S., Ozhigin, A., and Ratiu, D. (2015). Modern model-based development approach for embedded systems practical experience. In *1st IEEE International Symposium on Systems Engineering, ISSE 2015*, 1st IEEE International Symposium on Systems Engineering, ISSE 2015 - Proceedings, pages 56–59. Institute of Electrical and Electronics Engineers Inc.

[441] Visser, E. (2008). Webdsl: A case study in domain-specific language engineering. *Generative and Transformational Techniques in Software Engineering II*, pages 291–373.

[442] Voelter, M. (2010). Embedded software development with projectional language workbenches. In *Model Driven Engineering Languages and Systems*, pages 32–46.

[443] Voelter, M. (2013). Language and ide modularization and composition with mps. *Generative and Transformational Techniques in Software Engineering IV*, pages 383–430.

[444] Voelter, M., Ratiu, D., Kolb, B., and Schaetz, B. (2013). mbeddr: instantiating a language workbench in the embedded software domain. *Automated Software Engineering*, 20(3):339–390.

[445] Voelter, M., Salzmann, C., and Kircher, M. (2005). Model driven software development in the context of embedded component infrastructures. *Component-Based Software Development for Embedded Systems*, pages 143–163.

[446] Vokac, M. and Glattetre, J. M. (2005). Using a domain-specific language and custom tools to model a multi-tier service-oriented application - experiences and challenges. In *Model Driven Engineering Languages and Systems*, volume 3713 LNCS, pages 492–506.

[447] Vollebregt, T., Kats, L. C. L., and Visser, E. (2012). Declarative specification of template-based textual editors. In *12th Workshop on Language Descriptions, Tools, and Applications, LDTA 2012, March 31, 2012 - April 1, 2012*, Proceedings of the 12th Workshop on Language Descriptions, Tools, and Applications, LDTA 2012, pages 8:1–8:7. Association for Computing Machinery.

[448] Walsh, J., Roche, D., and Foping, F. (2012). Nitroscript: A php template engine for customizing of e-commerce applications. In *Internet Technology And Secured Transactions, 2012 International Conference for*, pages 459–464.

[449] Wang, G., Di Natale, M., Sangiovanni-Vincentelli, A., and Mosterman, P. J. (2009). Automatic code generation for synchronous reactive communication. In *International Conference on Education and Social Sciences*, pages 40–47.

[450] Wang, Q., Zhang, X., Zhang, Y., and Yi, Q. (2013). Augem: Automatically generate high performance dense linear algebra kernels on x86 cpus. In *Proceedings of*

*the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 25:1–25:12.

[451] Wang, Y., Zhou, L., Zheng, Q., Zhang, Z., and Wu, G. (2010). An approach of code generation based on model integrated computing. In *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, volume vol.15, pages 114–17.

[452] Wang, Z. and Chalmers, K. (2013). Evolution feature oriented model driven product line engineering approach for synergistic and dynamic service evolution in clouds:four kinds of schema. *Procedia Computer Science*, 19:889–894.

[453] Warken, M. (2008). From testing to anti-product development. *International Journal on Software Tools for Technology Transfer*, 10(4):297–307.

[454] Watanobe, Y., Mirenkov, N., and Watanabe, Y. (2012). Aida compiler: A code synthesizer from programs in pictures. In *Proceedings of the 2012 Joint International Conference on Human-Centered Computer Environments*, pages 76–83.

[455] Weber, D., Scheidgen, M., and Fischer, J. (2016). *Exchanging the Target-Language in Existing, Non-Metamodel-Based Compilers*, pages 196–210. Springer International Publishing.

[456] Weisemöller, I., Klar, F., and Schürr, A. (2010). 16 development of tool extensions with moflon. *Model-Based Engineering of Embedded Real-Time Systems*, pages 337–343.

[457] Welling, G. and Ott, M. (2000). Customizing idl mappings and orb protocols. *Middleware 2000*, pages 396–414.

[458] Whittle, J., Hutchinson, J., Rouncefield, M., Burden, H., and Heldal, R. (2013). Industrial adoption of model-driven engineering: Are the tools really the problem? In *Model Driven Engineering Languages and Systems*, pages 1–17.

[459] Whittle, J., Hutchinson, J., Rouncefield, M., Burden, H., and Heldal, R. (2015). A taxonomy of tool-related issues affecting the adoption of model-driven engineering. *Software & Systems Modeling*, pages 1–19.

[460] Wimmer, M. and Burgueño, L. (2013). Testing m2t/t2m transformations. In *Model Driven Engineering Languages and Systems*, pages 203–219.

[461] Winetzhammer, S. and Westfechtel, B. (2015). Staged translation of graph transformation rules. *Model-Driven Engineering and Software Development*, pages 134–152.

[462] Winkler, U., Fritzsche, M., Gilani, W., and Marshall, A. (2012). Bob the builder: A fast and friendly model-to-petrinet transformer. *Modelling Foundations and Applications*, pages 416–427.

[463] Wu, G., Cheng, D., and Zhang, Z. (2009). A solution based on modeling and code generation for embedded control system. *Journal of Software Engineering and Applications*, 2(3):160–4.

[464] Xiaohong, L., Zhiyong, F., and Li, L. e. a. (2006). A template language for agent construction. *Computational Science and Its Applications - ICCSA 2006*, pages 32–38.

[465] Xue-Bin, W., yua n, W. Q., Huai-Min, W., and Dian-Xi, S. (2007). Research and implementation of design pattern-oriented model transformation. In *Computing in the Global Information Technology, 2007. ICCGI 2007. International Multi-Conference on*, pages 24–24.

[466] Yahya, I., Turki, S. H., Charfi, A., Kallel, S., and Bouaziz, R. (2013). An aspect-oriented approach to enforce security properties in business processes. *Service-Oriented Computing - ICSOC 2012 Workshops*, pages 344–355.

[467] Yen, I.-L., Goluguri, J., Bastani, F., Khan, L., and Linn, J. (2002). A component-based approach for embedded software development. In *Object-Oriented Real-Time Distributed Computing, 2002. (ISORC 2002). Proceedings. Fifth IEEE International Symposium on*, pages 402–10.

[468] Yoong, L. H., Bhatti, Z. E., and Roop, P. S. (2012). Combining iec 61499 model-based design with component-based architecture for robotics. *Simulation, Modeling, and Programming for Autonomous Robots*, pages 349–360.

[469] Yu, H., Gamatié, A., Rutten, E., and Dekeyser, J.-L. (2008). Model transformations from a data parallel formalism towards synchronous languages. *Embedded Systems Specification and Design Languages*, pages 183–198.

[470] Yu, Y., Lin, Y., Hu, Z., Hidaka, S., Kato, H., and Montrieux, L. (2012). Maintaining invariant traceability through bidirectional transformations. In *Proceedings of the 34th International Conference on Software Engineering*, pages 540–50.

[471] Yu, Y., Tun, T. T., Bandara, A. K., Zhang, T., and Nuseibeh, B. (2014). From model-driven software development processes to problem diagnoses at runtime. In *Dagstuhl Seminar 11481 on Models@run.time, November 27, 2011 - December 2, 2011*, pages 188–207. Springer Verlag.

[472] Zäschke, T., Zimmerli, C., Leone, S., Nguyen, M. K., and Norrie, M. C. (2013). Adaptive model-driven information systems development for object databases. *Information Systems Development*, pages 513–525.

[473] Zhang, C., Bakshi, A., and Prasanna, V. K. (2007). Modelml: A markup language for automatic model synthesis. In *2007 IEEE International Conference on Information Reuse and Integration, IEEE IRI-2007*, 2007 IEEE International Conference on Information Reuse and Integration, IEEE IRI-2007, pages 317–322. Inst. of Elec. and Elec. Eng. Computer Society.

[474] Zhang, J., Buy, U., and Liu, X. (2003). A framework for the efficient production of web applications. In *Computers and Communication, 2003. (ISCC 2003). Proceedings. Eighth IEEE International Symposium on*, pages 44–49.

[475] Zhao, Q., Amagasaki, M., Iida, M., Kuga, M., and Sueyoshi, T. (2013). An fpga design and implementation framework combined with commercial vlsi cads. In *Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2013 8th International Workshop on*, pages 1–7.

[476] Zheng, Y. and Taylor, R. N. (2013). A classification and rationalization of model-based software development. *Software & Systems Modeling*, 12(4):669–678.

[477] Zhou, G., Leung, M.-K., and Lee, E. A. (2007). A code generation framework for actor-oriented models with partial evaluation. *Embedded Software and Systems*, pages 193–206.

[478] Zimmermann, O., Gschwind, T., Küster, J., Leymann, F., and Schuster, N. (2007). Reusable architectural decision models for enterprise application development. *Software Architectures, Components, and Applications*, pages 15–32.

[479] Zolotas, C., Diamantopoulos, T., Chatzidimitriou, K. C., and Symeonidis, A. L. (2016). From requirements to source code: a model-driven engineering approach for restful web services. *Automated Software Engineering*, pages 1–48.

[480] Zook, D., Huang, S. S., and Smaragdakis, Y. (2004). Generating aspectj programs with meta-aspectj. In *3rd International Conference on Generative Programming and Component Engineering, GPCE 2004, October 24, 2004 - October 28, 2004*, volume 3286 of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 1–18. Springer Verlag.

[481] Zschaler, S. and Rashid, A. (2011). Towards modular code generators using symmetric language-aware aspects. In *Proceedings of the 1st International Workshop on Free Composition*, pages 6:1–6:5.

## Appendix II

## Classification table of the SMS

| Ref. number | Year | Tool | Design-time input | Output | MDE | Runtime input | Template style |
|---|---|---|---|---|---|---|---|
| 1 | 2010 | Rhapsody | General purpose | Source code | Yes | General purpose | Predefined |
| 2 | 2010 | Unspecified | General purpose | Source code | Yes | General Purpose | Predefined |
| 3 | 2010 | Fujaba | General purpose | Source code | Yes | General Purpose | Predefined |
| 4 | 2008 | Other | General purpose | Source code | Yes | General purpose | Predefined |
| 5 | 2012 | Acceleo | General purpose | Structured data | Yes | General Purpose | Output-based |
| 6 | 2007 | Xpand | General purpose | Source code | Yes | General purpose | Output-based |
| 7 | 2012 | Xpand | General purpose | Structured data | Yes | General purpose | Output-based |
| 8 | 2013 | Other | Programming Language | Source code | No | Source code | Predefined |
| 9 | 2005 | XSLT | General purpose | Structured data | Yes | General purpose | Output-based |
| 10 | 2006 | XSLT | Schema | Structured data | Yes | Structured data | Output-based |
| 11 | 2007 | Unspecified | General purpose | Source code | Yes | General Purpose | Output-based |
| 12 | 2012 | Xpand | Domain Specific | Natural language | Yes | Domain specific | Output-based |
| 13 | 2013 | Xpand | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 14 | 2013 | Xpand | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 15 | 2007 | MOFScript | General purpose | Source code | Yes | General Purpose | Output-based |
| 16 | 2013 | XSLT | Schema | Source code | No | Structured data | Output-based |
| 17 | 2011 | Other | General purpose | Source code | Yes | General Purpose | Output-based |
| 18 | 2006 | Fujaba | General purpose | Source code | Yes | General purpose | Predefined |
| 19 | 2016 | Acceleo | General purpose | Structured data | Yes | General purpose | Output-based |
| 20 | 2004 | Other | General purpose | Source code | Yes | General Purpose | Output-based |
| 21 | 2014 | Acceleo | General purpose | Natural language | Yes | General Purpose | Output-based |
| 22 | 2013 | StringTemplate | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 23 | 2012 | XSLT | Programming Language | Source code | No | Source code | Output-based |
| 24 | 2006 | Other | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 25 | 2008 | JET | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 26 | 2013 | Xpand | General purpose | Source code | Yes | General Purpose | Output-based |
| 27 | 2007 | Other | Schema | Structured data | No | Structured data | Output-based |
| 28 | 2014 | Unspecified | Domain specific | Source code | Yes | Domain specific | Output-based |
| 29 | 2011 | Unspecified | Programming Language | Source code | No | Source code | Rule-based |
| 30 | 2013 | Other | Schema | Structured data | No | Structured data | Output-based |
| 31 | 2012 | Unspecified | Domain Specific | Source code | No | Domain specific | Output-based |

Table II.I: Classification table 1–31

| Ref. number | Year | Tool | Design-time input | Output | MDE | Runtime input | Template style |
|---|---|---|---|---|---|---|---|
| 32 | 2014 | Rational | General purpose | Source code | Yes | General purpose | Predefined |
| 33 | 2014 | Rational | General purpose | Source code | Yes | General purpose | Predefined |
| 34 | 2013 | Unspecified | General purpose | Source code | No | General Purpose | Output-based |
| 35 | 2012 | Programmed | Schema | Source code | Yes | Structured data | Output-based |
| 36 | 2007 | Other | General purpose | Source code | Yes | General purpose | Predefined |
| 37 | 2014 | Acceleo | General purpose | Source code | Yes | General Purpose | Output-based |
| 38 | 2006 | Other | General purpose | Source code | Yes | General Purpose | Output-based |
| 39 | 2010 | Other | General purpose | Structured data | Yes | General purpose | Output-based |
| 40 | 2014 | Other | Programming Language | Source code | Yes | Source code | Predefined |
| 41 | 2013 | Acceleo | General purpose | Source code | Yes | General purpose | Output-based |
| 42 | 2010 | Other | Schema | Source code | No | Structured data | Predefined |
| 43 | 2005 | Rhapsody | General purpose | Source code | Yes | General purpose | Predefined |
| 44 | 2000 | Rational | General purpose | Source code | Yes | General Purpose | Predefined |
| 45 | 2005 | Velocity | General purpose | Source code | Yes | General Purpose | Output-based |
| 46 | 2008 | Unspecified | Programming Language | Source code | No | Source code | Predefined |
| 47 | 2016 | Acceleo | General purpose | Structured data | Yes | General purpose | Output-based |
| 48 | 2011 | StringTemplate | Programming Language | Structured data | No | Source code | Output-based |
| 49 | 2014 | Acceleo | General purpose | Source code | Yes | General purpose | Output-based |
| 50 | 2006 | Programmed | Programming Language | Structured data | No | Source code | Output-based |
| 51 | 2002 | Unspecified | Domain Specific | Structured data | Yes | Domain specific | Output-based |
| 52 | 2014 | Xtend | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 53 | 2016 | Other | General purpose | Source code | Yes | General purpose | Output-based |
| 54 | 2009 | Unspecified | Programming Language | Structured data | No | Source code | Output-based |
| 55 | 2015 | Unspecified | Schema | Source code | No | Structured data | Output-based |
| 56 | 2008 | Velocity | General purpose | Source code | Yes | General Purpose | Output-based |
| 57 | 2014 | Unspecified | General purpose | Source code | Yes | General purpose | Predefined |
| 58 | 2000 | Other | Schema | Source code | Yes | Structured data | Output-based |
| 59 | 2004 | Rational | General purpose | Source code | Yes | General Purpose | Predefined |
| 60 | 2005 | Rational | General purpose | Source code | Yes | General purpose | Predefined |
| 61 | 2013 | Simulink TLC | Domain Specific | Source code | Yes | Domain specific | Rule-based |
| 62 | 2008 | Unspecified | General purpose | Source code | Yes | General Purpose | Output-based |
| 63 | 2006 | Simulink TLC | Domain Specific | Source code | Yes | Domain specific | Rule-based |

Table II.II: Classification table 32–63

| Ref. number | Year | Tool | Design-time input | Output | MDE | Runtime input | Template style |
|---|---|---|---|---|---|---|---|
| 64 | 2013 | Acceleo | General purpose | Source code | Yes | General Purpose | Output-based |
| 65 | 2005 | Other | Programming Language | Source code | No | Source code | Output-based |
| 66 | 2005 | Other | Programming Language | Source code | No | Source code | Output-based |
| 67 | 2006 | Other | Programming Language | Natural language | No | Source code | Output-based |
| 68 | 2007 | Other | Programming Language | Natural language | No | Source code | Output-based |
| 69 | 2007 | Xpand | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 70 | 2008 | Xpand | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 71 | 2013 | EGL | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 72 | 2008 | Other | Schema | Source code | No | Structured data | Output-based |
| 73 | 2011 | Xpand | General purpose | Source code | Yes | General Purpose | Output-based |
| 74 | 2005 | Fujaba | General purpose | Source code | Yes | General Purpose | Predefined |
| 75 | 2005 | Fujaba | General purpose | Source code | Yes | General Purpose | Predefined |
| 76 | 2011 | Simulink TLC | Domain Specific | Source code | Yes | Domain specific | Rule-based |
| 77 | 2013 | Unspecified | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 78 | 2011 | Unspecified | Programming Language | Source code | No | Source code | Output-based |
| 79 | 2012 | Unspecified | General purpose | Source code | Yes | General Purpose | Output-based |
| 80 | 2009 | Xpand | General purpose | Source code | Yes | General Purpose | Output-based |
| 81 | 2003 | XSLT | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 82 | 2006 | JET | General purpose | Source code | Yes | General purpose | Output-based |
| 83 | 2003 | XSLT | Schema | Structured data | No | Structured data | Output-based |
| 84 | 2016 | Unspecified | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 85 | 2013 | Other | Schema | Source code | No | Structured data | Output-based |
| 86 | 2006 | Unspecified | Schema | Source code | No | Structured data | Predefined |
| 87 | 2007 | Simulink TLC | Domain Specific | Source code | Yes | Domain specific | Rule-based |
| 88 | 2010 | Programmed | Schema | Source code | Yes | Structured data | Output-based |
| 89 | 2011 | XSLT | Schema | Source code | No | Structured data | Output-based |
| 90 | 2012 | XSLT | Schema | Source code | No | Structured data | Output-based |
| 91 | 2012 | Xpand | General purpose | Source code | Yes | General purpose | Output-based |
| 92 | 2008 | JET | General purpose | Source code | Yes | General purpose | Output-based |
| 93 | 2005 | Other | Schema | Source code | Yes | Structured data | Predefined |
| 94 | 2004 | XSLT | Schema | Source code | No | Structured data | Output-based |
| 95 | 2013 | Acceleo | General purpose | Structured data | Yes | General purpose | Output-based |

Table II.III: Classification table 64–95

| Ref. number | Year | Tool | Design-time input | Output | MDE | Runtime input | Template style |
|---|---|---|---|---|---|---|---|
| 96 | 2008 | Xpand | General purpose | Source code | Yes | General purpose | Output-based |
| 97 | 2014 | Other | Programming Language | Source code | No | Source code | Predefined |
| 98 | 2010 | EGL | General purpose | Structured data | Yes | General purpose | Output-based |
| 99 | 2013 | Unspecified | General purpose | Source code | Yes | General Purpose | Predefined |
| 100 | 2015 | Acceleo | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 101 | 2007 | Unspecified | General purpose | Structured data | Yes | General purpose | Output-based |
| 102 | 2014 | Unspecified | General purpose | Source code | Yes | General Purpose | Output-based |
| 103 | 2008 | Other | General purpose | Source code | Yes | General Purpose | Predefined |
| 104 | 2009 | Xpand | General purpose | Source code | Yes | General purpose | Output-based |
| 105 | 2009 | Unspecified | Domain Specific | Source code | No | Domain specific | Predefined |
| 106 | 2011 | Xpand | General purpose | Natural language | Yes | General Purpose | Output-based |
| 107 | 2012 | Programmed | Schema | Source code | No | Structured data | Output-based |
| 108 | 2002 | Unspecified | Programming Language | Source code | No | Source code | Predefined |
| 109 | 2005 | Other | General purpose | Source code | Yes | General Purpose | Predefined |
| 110 | 2016 | Acceleo | General purpose | Structured data | Yes | General purpose | Output-based |
| 111 | 2007 | StringTemplate | Schema | Source code | No | Structured data | Output-based |
| 112 | 2005 | Other | Domain Specific | Source code | No | Domain specific | Output-based |
| 113 | 2005 | Fujaba | General purpose | Source code | Yes | General Purpose | Predefined |
| 114 | 2010 | Simulink TLC | Domain Specific | Source code | Yes | Domain specific | Rule-based |
| 115 | 2016 | Acceleo | General purpose | Source code | Yes | General purpose | Output-based |
| 116 | 2014 | Acceleo | General purpose | Structured data | Yes | General Purpose | Output-based |
| 117 | 2008 | Other | Schema | Structured data | No | Structured data | Output-based |
| 118 | 2007 | Velocity | General purpose | Source code | Yes | General Purpose | Output-based |
| 119 | 2013 | Other | Schema | Source code | No | Structured data | Rule-based |
| 120 | 2005 | XSLT | General purpose | Source code | Yes | General Purpose | Output-based |
| 121 | 2011 | Xpand | General purpose | Source code | Yes | General Purpose | Output-based |
| 122 | 2010 | Simulink TLC | Domain Specific | Source code | Yes | Domain specific | Rule-based |
| 123 | 2011 | Simulink TLC | Domain Specific | Source code | Yes | Domain specific | Rule-based |
| 124 | 2012 | Velocity | General purpose | Source code | Yes | General Purpose | Output-based |
| 125 | 2014 | Other | General purpose | Structured data | Yes | General Purpose | Output-based |
| 126 | 2016 | Other | General purpose | Source code | Yes | General purpose | Output-based |
| 127 | 2014 | Unspecified | Schema | Source code | No | Structured data | Output-based |

Table II.IV: Classification table 96–127

| Ref. number | Year | Tool | Design-time input | Output | MDE | Runtime input | Template style |
|---|---|---|---|---|---|---|---|
| 128 | 2012 | JET | Domain Specific | Structured data | Yes | Domain specific | Output-based |
| 129 | 2008 | Programmed | Schema | Structured data | No | Structured data | Output-based |
| 130 | 2010 | Unspecified | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 131 | 2016 | Other | General purpose | Source code | Yes | General purpose | Output-based |
| 132 | 2007 | JET | General purpose | Source code | Yes | General purpose | Output-based |
| 133 | 2013 | Other | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 134 | 2002 | Unspecified | Programming Language | Source code | No | Source code | Predefined |
| 135 | 2009 | XSLT | Schema | Natural language | Yes | Structured data | Output-based |
| 136 | 2016 | XSLT | General purpose | Structured data | Yes | General purpose | Output-based |
| 137 | 2011 | Unspecified | General purpose | Source code | Yes | General purpose | Output-based |
| 138 | 2015 | Unspecified | General purpose | Source code | Yes | General Purpose | Output-based |
| 139 | 2006 | XSLT | Schema | Source code | Yes | Structured data | Output-based |
| 140 | 2016 | Acceleo | General purpose | Source code | Yes | General purpose | Output-based |
| 141 | 2007 | Unspecified | General purpose | Source code | Yes | General purpose | Predefined |
| 142 | 2002 | Unspecified | Schema | Structured data | Yes | Structured data | Output-based |
| 143 | 2004 | Other | Schema | Source code | No | Structured data | Predefined |
| 144 | 2014 | Other | Programming Language | Source code | No | Source code | Output-based |
| 145 | 2007 | Other | General purpose | Source code | Yes | General purpose | Output-based |
| 146 | 2009 | Unspecified | General purpose | Structured data | Yes | General purpose | Predefined |
| 147 | 2000 | Unspecified | Schema | Source code | No | Structured data | Output-based |
| 148 | 2011 | Xpand | General purpose | Source code | Yes | General Purpose | Output-based |
| 149 | 2012 | XSLT | Schema | Source code | No | Structured data | Output-based |
| 150 | 2009 | Other | Domain Specific | Structured data | Yes | Domain specific | Output-based |
| 151 | 2010 | Other | Domain Specific | Structured data | Yes | Domain specific | Output-based |
| 152 | 2008 | XSLT | Schema | Structured data | No | Structured data | Output-based |
| 153 | 2006 | Unspecified | Schema | Source code | No | Structured data | Predefined |
| 154 | 2008 | Unspecified | General purpose | Source code | Yes | General Purpose | Predefined |
| 155 | 2010 | Unspecified | Programming Language | Source code | No | Source code | Predefined |
| 156 | 2011 | Programmed | General purpose | Source code | Yes | General Purpose | Predefined |
| 157 | 2014 | MOFScript | General purpose | Source code | Yes | General Purpose | Output-based |
| 158 | 2016 | Xpand | General purpose | Structured data | Yes | General purpose | Output-based |
| 159 | 2011 | Unspecified | General purpose | Source code | Yes | General Purpose | Predefined |

Table II.V: Classification table 128–159

| Ref. number | Year | Tool | Design-time input | Output | MDE | Runtime input | Template style |
|---|---|---|---|---|---|---|---|
| 160 | 2014 | Other | Schema | Source code | No | Structured data | Output-based |
| 161 | 2008 | Other | General purpose | Source code | No | General Purpose | Predefined |
| 162 | 2012 | Programmed | Schema | Source code | Yes | Structured data | Output-based |
| 163 | 2006 | Fujaba | General purpose | Source code | Yes | General purpose | Predefined |
| 164 | 2010 | Fujaba | General purpose | Source code | Yes | General purpose | Predefined |
| 165 | 2012 | Other | Domain Specific | Source code | Yes | Domain specific | Predefined |
| 166 | 2013 | Simulink TLC | Domain specific | Source code | Yes | Domain specific | Rule-based |
| 167 | 2006 | JET | Schema | Source code | No | Structured data | Output-based |
| 168 | 2008 | Other | General purpose | Source code | Yes | General Purpose | Predefined |
| 169 | 2008 | Unspecified | Programming Language | Source code | No | Source code | Predefined |
| 170 | 2005 | Other | General purpose | Source code | Yes | General Purpose | Predefined |
| 171 | 2008 | Velocity | General purpose | Source code | Yes | General Purpose | Output-based |
| 172 | 2016 | Velocity | General purpose | Source code | No | General purpose | Output-based |
| 173 | 2011 | Other | Schema | Source code | Yes | Structured data | Output-based |
| 174 | 2011 | Other | Schema | Source code | Yes | Structured data | Output-based |
| 175 | 2005 | JET | General purpose | Source code | Yes | General purpose | Output-based |
| 176 | 2004 | Unspecified | General purpose | Source code | Yes | General purpose | Output-based |
| 177 | 2015 | Unspecified | General purpose | Source code | Yes | General purpose | Output-based |
| 178 | 2016 | Xpand | General purpose | Source code | Yes | General purpose | Output-based |
| 179 | 2015 | Other | Programming Language | Source code | No | Source code | Rule-based |
| 180 | 2009 | Xpand | General purpose | Source code | Yes | General purpose | Output-based |
| 181 | 2004 | Other | General purpose | Source code | No | General Purpose | Predefined |
| 182 | 2005 | Fujaba | General purpose | Source code | Yes | General purpose | Predefined |
| 183 | 2014 | Acceleo | General purpose | Source code | Yes | General purpose | Output-based |
| 184 | 2013 | Acceleo | General purpose | Source code | Yes | General Purpose | Output-based |
| 185 | 2013 | Xpand | General purpose | Structured data | Yes | General purpose | Output-based |
| 186 | 2016 | Acceleo | General purpose | Source code | Yes | General purpose | Output-based |
| 187 | 2008 | Unspecified | General purpose | Source code | Yes | General purpose | Output-based |
| 188 | 2011 | Unspecified | Schema | Source code | No | Structured data | Predefined |
| 189 | 2010 | Rational | General purpose | Source code | Yes | General purpose | Predefined |
| 190 | 2010 | Xpand | General purpose | Source code | Yes | General purpose | Output-based |
| 191 | 2007 | JET | Schema | Source code | Yes | Structured data | Output-based |

Table II.VI: Classification table 160–191

| Ref. number | Year | Tool | Design-time input | Output | MDE | Runtime input | Template style |
|---|---|---|---|---|---|---|---|
| 192 | 2005 | Velocity | Schema | Structured data | No | Structured data | Output-based |
| 193 | 2010 | Other | General purpose | Source code | Yes | General Purpose | Output-based |
| 194 | 2014 | Xpand | General purpose | Source code | Yes | General purpose | Output-based |
| 195 | 2005 | Unspecified | General purpose | Structured data | Yes | General purpose | Output-based |
| 196 | 2013 | Rational | General purpose | Source code | Yes | General purpose | Predefined |
| 197 | 2010 | Other | Domain specific | Source code | Yes | Domain specific | Predefined |
| 198 | 2013 | JET | Domain specific | Source code | Yes | Domain specific | Output-based |
| 199 | 2004 | XSLT | Schema | Source code | Yes | Structured data | Output-based |
| 200 | 2010 | Other | Domain Specific | Source code | No | Domain specific | Rule-based |
| 201 | 2010 | StringTemplate | Domain specific | Source code | Yes | Domain specific | Output-based |
| 202 | 2003 | XSLT | Schema | Source code | No | Structured data | Output-based |
| 203 | 2009 | Other | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 204 | 2009 | Other | Domain Specific | Source code | No | Domain specific | Output-based |
| 205 | 2007 | Unspecified | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 206 | 2013 | Other | Schema | Source code | No | Structured data | Output-based |
| 207 | 2016 | Other | Schema | Source code | Yes | Structured data | Output-based |
| 208 | 2015 | XSLT | Schema | Source code | Yes | Structured data | Output-based |
| 209 | 2013 | EGL | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 210 | 2005 | Other | Schema | Source code | No | Structured data | Output-based |
| 211 | 2008 | Other | Programming Language | Source code | No | Source code | Output-based |
| 212 | 2012 | Unspecified | Programming Language | Source code | No | Source code | Output-based |
| 213 | 2012 | Other | Schema | Natural language | No | Structured data | Output-based |
| 214 | 2016 | Other | General purpose | Source code | Yes | General purpose | Predefined |
| 215 | 2003 | Unspecified | Programming Language | Source code | No | Source code | Output-based |
| 216 | 2014 | Other | Schema | Source code | Yes | Structured data | Output-based |
| 217 | 2016 | StringTemplate | Schema | Source code | No | Structured data | Output-based |
| 218 | 2011 | XSLT | General purpose | Source code | Yes | General purpose | Output-based |
| 219 | 2007 | Unspecified | Programming Language | Source code | No | Source code | Output-based |
| 220 | 2004 | Programmed | General purpose | Structured data | Yes | General Purpose | Output-based |
| 221 | 2009 | Velocity | Domain Specific | Structured data | Yes | Domain specific | Output-based |
| 222 | 2016 | Acceleo | General purpose | Source code | Yes | General purpose | Output-based |
| 223 | 2012 | XSLT | Schema | Structured data | No | Structured data | Output-based |

Table II.VII: Classification table part 192–223

| Ref. number | Year | Tool | Design-time input | Output | MDE | Runtime input | Template style |
|---|---|---|---|---|---|---|---|
| 224 | 2004 | Other | Schema | Source code | No | Structured data | Output-based |
| 225 | 2007 | Unspecified | Domain Specific | Source code | Yes | Domain specific | Predefined |
| 226 | 2012 | Other | Schema | Source code | No | Structured data | Predefined |
| 227 | 2013 | Other | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 228 | 2006 | Velocity | Schema | Source code | No | Structured data | Output-based |
| 229 | 2013 | Other | Schema | Structured data | No | Structured data | Predefined |
| 230 | 2015 | Other | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 231 | 2008 | Velocity | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 232 | 2011 | StringTemplate | Domain Specific | Structured data | No | Domain specific | Output-based |
| 233 | 2012 | Velocity | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 234 | 2013 | Xpand | General purpose | Source code | Yes | General Purpose | Output-based |
| 235 | 2014 | Other | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 236 | 2016 | Other | General purpose | Source code | Yes | General purpose | Output-based |
| 237 | 2010 | Unspecified | Domain Specific | Source code | Yes | Domain specific | Predefined |
| 238 | 2011 | Xpand | General purpose | Source code | Yes | General Purpose | Output-based |
| 239 | 2007 | Unspecified | Schema | Source code | No | Structured data | Predefined |
| 240 | 2012 | Fujaba | General purpose | Source code | Yes | General Purpose | Predefined |
| 241 | 2013 | Acceleo | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 242 | 2009 | Other | General purpose | Source code | Yes | General Purpose | Predefined |
| 243 | 2009 | MOFScript | General purpose | Structured data | Yes | General purpose | Output-based |
| 244 | 2001 | Programmed | Schema | Structured data | No | Structured data | Output-based |
| 245 | 2012 | StringTemplate | Domain Specific | Structured data | No | Domain specific | Output-based |
| 246 | 2011 | Acceleo | General purpose | Source code | Yes | General Purpose | Output-based |
| 247 | 2011 | Acceleo | General purpose | Source code | Yes | General Purpose | Output-based |
| 248 | 2016 | Other | General purpose | Source code | Yes | General purpose | Output-based |
| 249 | 2009 | JET | General purpose | Source code | Yes | General Purpose | Output-based |
| 250 | 2015 | Xtend | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 251 | 2011 | Other | General purpose | Source code | Yes | General purpose | Output-based |
| 252 | 2006 | Other | Programming Language | Source code | No | Source code | Predefined |
| 253 | 2016 | FreeMarker | Schema | Source code | No | Structured data | Output-based |
| 254 | 2005 | Velocity | General purpose | Source code | Yes | General purpose | Output-based |
| 255 | 2015 | EGL | General purpose | Source code | Yes | General purpose | Output-based |

Table II.VIII: Classification table part 224–255

| Ref. number | Year | Tool | Design-time input | Output | MDE | Runtime input | Template style |
|---|---|---|---|---|---|---|---|
| 256 | 2014 | Other | Domain Specific | Source code | No | Domain specific | Output-based |
| 257 | 2007 | JET | General purpose | Source code | Yes | General purpose | Output-based |
| 258 | 2004 | Other | General purpose | Source code | Yes | General Purpose | Output-based |
| 259 | 2006 | Velocity | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 260 | 2016 | Other | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 261 | 2008 | Unspecified | Programming Language | Source code | No | Source code | Predefined |
| 262 | 2005 | JET | Domain Specific | Source code | No | Domain specific | Output-based |
| 263 | 2007 | JET | General purpose | Source code | Yes | General Purpose | Output-based |
| 264 | 2011 | Unspecified | General purpose | Source code | Yes | General Purpose | Predefined |
| 265 | 2011 | Other | General purpose | Source code | Yes | General purpose | Output-based |
| 266 | 2013 | Unspecified | General purpose | Source code | Yes | General Purpose | Output-based |
| 267 | 2013 | Unspecified | General purpose | Source code | Yes | General Purpose | Predefined |
| 268 | 2014 | Unspecified | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 269 | 2016 | Xtend | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 270 | 2011 | MOFScript | General purpose | Source code | Yes | General Purpose | Output-based |
| 271 | 2011 | Other | General purpose | Source code | Yes | General purpose | Output-based |
| 272 | 2013 | EGL | General purpose | Source code | Yes | General Purpose | Output-based |
| 273 | 2015 | EGL | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 274 | 2006 | Unspecified | Schema | Source code | No | Structured data | Predefined |
| 275 | 2006 | Unspecified | Domain Specific | Source code | No | Domain specific | Output-based |
| 276 | 2010 | Unspecified | Schema | Source code | No | Structured data | Output-based |
| 277 | 2012 | Unspecified | General purpose | Structured data | Yes | General purpose | Output-based |
| 278 | 2003 | Unspecified | Schema | Source code | No | Structured data | Rule-based |
| 279 | 2013 | Acceleo | General purpose | Source code | Yes | General Purpose | Output-based |
| 280 | 2016 | Other | General purpose | Source code | Yes | General purpose | Output-based |
| 281 | 2010 | Unspecified | Domain Specific | Source code | Yes | Domain specific | Predefined |
| 282 | 2012 | Unspecified | Programming Language | Source code | No | Source code | Output-based |
| 283 | 2012 | Other | Schema | Source code | No | Structured data | Output-based |
| 284 | 2013 | Unspecified | Programming Language | Source code | No | Source code | Predefined |
| 285 | 2009 | Other | Schema | Source code | No | Structured data | Output-based |
| 286 | 2008 | Other | Schema | Source code | No | Structured data | Output-based |
| 287 | 2013 | Other | Schema | Source code | No | Structured data | Output-based |

Table II.IX: Classification table 256–287

| Ref. number | Year | Tool | Design-time input | Output | MDE | Runtime input | Template style |
|---|---|---|---|---|---|---|---|
| 288 | 2009 | Velocity | Domain Specific | Structured data | Yes | Domain specific | Output-based |
| 289 | 2012 | Xpand | General purpose | Source code | Yes | General Purpose | Output-based |
| 290 | 2002 | Unspecified | General purpose | Structured data | Yes | General Purpose | Predefined |
| 291 | 2003 | Other | General purpose | Structured data | Yes | General Purpose | Predefined |
| 292 | 2014 | Unspecified | Domain Specific | Structured data | No | Domain specific | Predefined |
| 293 | 2015 | Unspecified | General purpose | Source code | Yes | General Purpose | Predefined |
| 294 | 2010 | Xpand | Domain specific | Source code | Yes | Domain specific | Output-based |
| 295 | 2010 | Other | General purpose | Source code | Yes | General Purpose | Output-based |
| 296 | 2012 | Other | Schema | Source code | No | Structured data | Output-based |
| 297 | 2011 | Unspecified | Schema | Source code | No | Structured data | Output-based |
| 298 | 2011 | Simulink TLC | Domain Specific | Source code | Yes | Domain specific | Rule-based |
| 299 | 2013 | Other | Schema | Source code | No | Structured data | Output-based |
| 300 | 2010 | Other | Programming Language | Structured data | No | Source code | Output-based |
| 301 | 2016 | Other | Schema | Source code | No | Structured data | Output-based |
| 302 | 2006 | Rhapsody | General purpose | Source code | Yes | General Purpose | Predefined |
| 303 | 2011 | Rhapsody | General purpose | Source code | Yes | General Purpose | Predefined |
| 304 | 2015 | Xpand | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 305 | 2009 | Unspecified | Schema | Source code | No | Structured data | Predefined |
| 306 | 2007 | Unspecified | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 307 | 2013 | JET | General purpose | Source code | Yes | General Purpose | Output-based |
| 308 | 2008 | XSLT | Schema | Source code | No | Structured data | Output-based |
| 309 | 2003 | Unspecified | Domain Specific | Source code | No | Domain specific | Predefined |
| 310 | 2005 | Unspecified | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 311 | 2005 | XSLT | Schema | Source code | No | Structured data | Output-based |
| 312 | 2005 | Velocity | General purpose | Source code | Yes | General Purpose | Output-based |
| 313 | 2016 | Other | Programming Language | Source code | No | Source code | Output-based |
| 314 | 2008 | Programmed | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 315 | 2009 | Other | General purpose | Source code | Yes | General Purpose | Output-based |
| 316 | 2014 | Unspecified | Domain specific | Structured data | Yes | Domain specific | Rule-based |
| 317 | 2016 | Acceleo | General purpose | Structured data | Yes | General purpose | Output-based |
| 318 | 2016 | Other | Domain Specific | Source code | No | Domain specific | Output-based |
| 319 | 2016 | Acceleo | General purpose | Structured data | Yes | General purpose | Output-based |

Table II.X: Classification table 288–319

| Ref. number | Year | Tool | Design-time input | Output | MDE | Runtime input | Template style |
|---|---|---|---|---|---|---|---|
| 320 | 2010 | Velocity | General purpose | Source code | Yes | General purpose | Output-based |
| 321 | 2016 | Acceleo | General purpose | Source code | Yes | General purpose | Output-based |
| 322 | 2012 | Acceleo | General purpose | Source code | Yes | General purpose | Output-based |
| 323 | 2006 | Velocity | General purpose | Source code | Yes | General Purpose | Output-based |
| 324 | 2005 | Other | General purpose | Source code | Yes | General Purpose | Predefined |
| 325 | 2008 | Other | General purpose | Source code | Yes | General Purpose | Predefined |
| 326 | 2012 | Rational | General purpose | Source code | Yes | General Purpose | Predefined |
| 327 | 2013 | Unspecified | Domain Specific | Structured data | Yes | Domain specific | Output-based |
| 328 | 2006 | FreeMarker | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 329 | 2009 | Simulink TLC | Domain Specific | Source code | Yes | Domain specific | Rule-based |
| 330 | 2012 | Xpand | General purpose | Source code | Yes | General purpose | Output-based |
| 331 | 2014 | StringTemplate | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 332 | 2005 | Unspecified | Schema | Source code | No | Structured data | Predefined |
| 333 | 2016 | FreeMarker | General purpose | Source code | Yes | General purpose | Output-based |
| 334 | 2016 | Velocity | Schema | Source code | No | Structured data | Output-based |
| 335 | 2005 | Velocity | General purpose | Source code | Yes | General purpose | Output-based |
| 336 | 2011 | Rational | General purpose | Source code | Yes | General Purpose | Predefined |
| 337 | 2016 | Unspecified | Schema | Source code | No | Structured data | Output-based |
| 338 | 2014 | Acceleo | General purpose | Natural language | Yes | General purpose | Output-based |
| 339 | 2015 | EGL | General purpose | Natural language | Yes | General Purpose | Output-based |
| 340 | 2005 | Unspecified | General purpose | Source code | No | General purpose | Predefined |
| 341 | 2009 | Rational | General purpose | Source code | Yes | General purpose | Predefined |
| 342 | 2005 | MOFScript | General purpose | Source code | Yes | General Purpose | Output-based |
| 343 | 2001 | Other | Programming Language | Structured data | No | Source code | Rule-based |
| 344 | 2006 | XSLT | Schema | Structured data | No | Structured data | Output-based |
| 345 | 2013 | EGL | General purpose | Natural language | Yes | General purpose | Output-based |
| 346 | 2012 | Xpand | General purpose | Source code | Yes | General Purpose | Output-based |
| 347 | 2006 | StringTemplate | Schema | Source code | No | Structured data | Output-based |
| 348 | 2016 | Other | Programming Language | Source code | No | Source code | Output-based |
| 349 | 2006 | Unspecified | General purpose | Source code | Yes | General Purpose | Predefined |
| 350 | 2016 | XSLT | Schema | Natural language | No | Structured data | Output-based |
| 351 | 2013 | XSLT | Schema | Source code | No | Structured data | Output-based |

Table II.XI: Classification table 320–351

| Ref. number | Year | Tool | Design-time input | Output | MDE | Runtime input | Template style |
|---|---|---|---|---|---|---|---|
| 352 | 2011 | Other | Programming Language | Source code | No | Source code | Output-based |
| 353 | 2009 | Unspecified | Schema | Structured data | No | Structured data | Predefined |
| 354 | 2016 | JET | General purpose | Source code | Yes | General purpose | Output-based |
| 355 | 2006 | JET | General purpose | Source code | Yes | General Purpose | Output-based |
| 356 | 2008 | JET | General purpose | Source code | Yes | General Purpose | Output-based |
| 357 | 2015 | Other | Schema | Source code | No | Structured data | Output-based |
| 358 | 2009 | Velocity | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 359 | 2005 | Unspecified | Schema | Natural language | No | Structured data | Predefined |
| 360 | 2015 | JET | Schema | Source code | No | Structured data | Output-based |
| 361 | 2016 | Velocity | General purpose | Source code | Yes | General purpose | Output-based |
| 362 | 2012 | Unspecified | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 363 | 2008 | Unspecified | Programming Language | Source code | No | Source code | Output-based |
| 364 | 2012 | XSLT | Programming Language | Source code | No | Source code | Output-based |
| 365 | 2014 | Other | Schema | Source code | No | Structured data | Output-based |
| 366 | 2010 | Velocity | General purpose | Source code | Yes | General purpose | Output-based |
| 367 | 2011 | Unspecified | Programming Language | Structured data | No | Source code | Output-based |
| 368 | 2014 | JET | General purpose | Structured data | Yes | General purpose | Output-based |
| 369 | 2015 | Unspecified | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 370 | 2011 | Unspecified | Schema | Source code | No | Structured data | Output-based |
| 371 | 2003 | Other | Schema | Source code | No | Structured data | Predefined |
| 372 | 2007 | Xpand | General purpose | Source code | Yes | General Purpose | Output-based |
| 373 | 2007 | Other | Domain Specific | Structured data | No | Domain specific | Predefined |
| 374 | 2009 | Acceleo | General purpose | Source code | Yes | General purpose | Output-based |
| 375 | 2011 | XSLT | Schema | Source code | No | Structured data | Output-based |
| 376 | 2003 | Unspecified | General purpose | Structured data | Yes | General Purpose | Predefined |
| 377 | 2007 | Unspecified | Schema | Source code | No | Structured data | Output-based |
| 378 | 2015 | Acceleo | General purpose | Source code | Yes | General purpose | Output-based |
| 379 | 2013 | Acceleo | General purpose | Structured data | Yes | General Purpose | Output-based |
| 380 | 2007 | Velocity | General purpose | Source code | Yes | General Purpose | Output-based |
| 381 | 2008 | EGL | General purpose | Source code | Yes | General purpose | Output-based |
| 382 | 2013 | EGL | General purpose | Source code | Yes | General purpose | Output-based |
| 383 | 2016 | Acceleo | General purpose | Source code | Yes | General purpose | Output-based |

Table II.XII: Classification table 352–383

| Ref. number | Year | Tool | Design-time input | Output | MDE | Runtime input | Template style |
|---|---|---|---|---|---|---|---|
| 384 | 2011 | Acceleo | General purpose | Source code | Yes | General Purpose | Output-based |
| 385 | 2013 | Acceleo | General purpose | Source code | Yes | General Purpose | Output-based |
| 386 | 2003 | XSLT | Schema | Structured data | No | Structured data | Output-based |
| 387 | 2005 | XSLT | Schema | Source code | No | Structured data | Output-based |
| 388 | 2013 | Other | Schema | Structured data | No | Structured data | Predefined |
| 389 | 2013 | Unspecified | General purpose | Source code | Yes | General purpose | Output-based |
| 390 | 2011 | JET | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 391 | 2012 | Unspecified | Schema | Source code | No | Structured data | Predefined |
| 392 | 2008 | Xpand | General purpose | Source code | Yes | General Purpose | Output-based |
| 393 | 2012 | Unspecified | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 394 | 2005 | Fujaba | General purpose | Source code | Yes | General Purpose | Predefined |
| 395 | 2003 | Unspecified | General purpose | Source code | Yes | General Purpose | Predefined |
| 396 | 2016 | Acceleo | General purpose | Source code | Yes | General purpose | Output-based |
| 397 | 2010 | Unspecified | Domain Specific | Source code | No | Domain specific | Output-based |
| 398 | 2006 | Velocity | Schema | Source code | No | Structured data | Output-based |
| 399 | 2006 | Other | General purpose | Structured data | Yes | General purpose | Predefined |
| 400 | 2005 | Other | General purpose | Source code | Yes | General purpose | Predefined |
| 401 | 2013 | Unspecified | General purpose | Source code | Yes | General Purpose | Output-based |
| 402 | 2014 | Other | General purpose | Source code | Yes | General Purpose | Output-based |
| 403 | 2014 | Other | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 404 | 2012 | Acceleo | General purpose | Source code | Yes | General purpose | Output-based |
| 405 | 2014 | XSLT | General purpose | Source code | Yes | General purpose | Output-based |
| 406 | 2010 | Unspecified | Programming Language | Source code | No | Source code | Predefined |
| 407 | 2005 | Velocity | General purpose | Source code | Yes | General purpose | Output-based |
| 408 | 2006 | Unspecified | Domain Specific | Source code | No | Domain specific | Output-based |
| 409 | 2004 | Other | Schema | Source code | No | Structured data | Output-based |
| 410 | 2014 | Other | Domain Specific | Source code | No | Domain specific | Predefined |
| 411 | 2009 | Rhapsody | General purpose | Source code | Yes | General Purpose | Predefined |
| 412 | 2011 | Programmed | Programming Language | Natural language | No | Source code | Predefined |
| 413 | 2014 | Unspecified | Schema | Source code | No | Structured data | Predefined |
| 414 | 2008 | Unspecified | General purpose | Source code | Yes | General purpose | Output-based |
| 415 | 2015 | Other | Schema | Source code | No | Structured data | Predefined |

Table II.XIII: Classification table 384–415

| Ref. number | Year | Tool | Design-time input | Output | MDE | Runtime input | Template style |
|---|---|---|---|---|---|---|---|
| 416 | 2002 | Velocity | General purpose | Source code | Yes | General Purpose | Output-based |
| 417 | 2013 | Unspecified | Programming Language | Structured data | No | Source code | Output-based |
| 418 | 2012 | FreeMarker | General purpose | Structured data | Yes | General purpose | Output-based |
| 419 | 2010 | FreeMarker | General purpose | Structured data | Yes | General purpose | Output-based |
| 420 | 2001 | Fujaba | General purpose | Source code | Yes | General purpose | Predefined |
| 421 | 2006 | XSLT | Schema | Source code | No | Structured data | Output-based |
| 422 | 2003 | Other | Schema | Structured data | No | Structured data | Predefined |
| 423 | 2014 | FreeMarker | Schema | Source code | No | Structured data | Predefined |
| 424 | 2015 | Other | Programming Language | Source code | No | Source code | Predefined |
| 425 | 2009 | Unspecified | Programming Language | Structured data | No | Source code | Output-based |
| 426 | 2012 | Unspecified | Schema | Structured data | No | Structured data | Predefined |
| 427 | 2005 | Unspecified | Schema | Structured data | No | Structured data | Output-based |
| 428 | 2013 | Xpand | General purpose | Source code | Yes | General Purpose | Output-based |
| 429 | 2016 | Simulink TLC | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 430 | 2011 | Xpand | General purpose | Source code | Yes | General Purpose | Output-based |
| 431 | 2016 | Acceleo | General purpose | Structured data | Yes | General purpose | Output-based |
| 432 | 2006 | Unspecified | Programming Language | Source code | No | Source code | Predefined |
| 433 | 2007 | Other | General purpose | Source code | No | General purpose | Predefined |
| 434 | 2006 | Programmed | Schema | Structured data | No | Structured data | Predefined |
| 435 | 2016 | Xtend | General purpose | Source code | Yes | General purpose | Output-based |
| 436 | 2011 | Unspecified | General purpose | Source code | Yes | General Purpose | Output-based |
| 437 | 2013 | JET | Domain specific | Source code | Yes | Domain specific | Output-based |
| 438 | 2015 | JET | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 439 | 2016 | Xtend | General purpose | Source code | Yes | General purpose | Output-based |
| 440 | 2015 | Unspecified | General purpose | Source code | No | General purpose | Predefined |
| 441 | 2008 | Unspecified | Domain specific | Structured data | Yes | Domain specific | Output-based |
| 442 | 2005 | Xpand | General purpose | Source code | Yes | General Purpose | Output-based |
| 443 | 2010 | Unspecified | Domain Specific | Source code | No | Domain specific | Rule-based |
| 444 | 2013 | Other | General purpose | Source code | Yes | General purpose | Output-based |
| 445 | 2013 | Unspecified | General purpose | Source code | Yes | General purpose | Output-based |
| 446 | 2005 | Unspecified | Domain Specific | Structured data | Yes | Domain specific | Output-based |
| 447 | 2012 | Unspecified | Domain specific | Structured data | Yes | Domain specific | Predefined |

Table II.XIV: Classification table part 416–447

| Ref. number | Year | Tool | Design-time input | Output | MDE | Runtime input | Template style |
|---|---|---|---|---|---|---|---|
| 448 | 2012 | Other | Schema | Source code | No | Structured data | Output-based |
| 449 | 2009 | Unspecified | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 450 | 2010 | Unspecified | Schema | Source code | Yes | Structured data | Output-based |
| 451 | 2013 | Other | Programming Language | Source code | No | Source code | Predefined |
| 452 | 2013 | Unspecified | General purpose | Source code | Yes | General purpose | Output-based |
| 453 | 2008 | Unspecified | Domain specific | Structured data | Yes | Domain specific | Predefined |
| 454 | 2012 | Programmed | Programming Language | Source code | No | Source code | Predefined |
| 455 | 2016 | Other | General purpose | Source code | Yes | General purpose | Output-based |
| 456 | 2010 | Fujaba | General purpose | Source code | Yes | General purpose | Predefined |
| 457 | 2000 | Unspecified | General purpose | Source code | No | General purpose | Output-based |
| 458 | 2013 | Unspecified | General purpose | Source code | Yes | General Purpose | Output-based |
| 459 | 2015 | Unspecified | General purpose | Source code | Yes | General Purpose | Output-based |
| 460 | 2013 | MOFScript | General purpose | Source code | Yes | General Purpose | Output-based |
| 461 | 2015 | Xpand | General purpose | Source code | Yes | General purpose | Output-based |
| 462 | 2012 | Unspecified | Domain Specific | Source code | Yes | Domain specific | Predefined |
| 463 | 2009 | Other | General purpose | Source code | Yes | General purpose | Predefined |
| 464 | 2006 | Unspecified | Schema | Source code | No | Structured data | Rule-based |
| 465 | 2007 | XSLT | General purpose | Structured data | Yes | General purpose | Output-based |
| 466 | 2013 | Xpand | Domain Specific | Source code | Yes | Domain specific | Output-based |
| 467 | 2002 | Unspecified | Schema | Structured data | No | Structured data | Output-based |
| 468 | 2012 | Unspecified | General purpose | Source code | Yes | General purpose | Predefined |
| 469 | 2008 | JET | General purpose | Source code | Yes | General Purpose | Output-based |
| 470 | 2012 | JET | General purpose | Source code | Yes | General Purpose | Output-based |
| 471 | 2014 | Unspecified | General purpose | Source code | Yes | General purpose | Output-based |
| 472 | 2013 | Acceleo | General purpose | Source code | Yes | General purpose | Output-based |
| 473 | 2003 | Unspecified | Schema | Source code | No | Structured data | Predefined |
| 474 | 2007 | XSLT | Schema | Structured data | No | Structured data | Output-based |
| 475 | 2013 | Other | Programming Language | Source code | No | Source code | Predefined |
| 476 | 2013 | Unspecified | General purpose | Structured data | Yes | General purpose | Output-based |
| 477 | 2007 | Unspecified | Programming Language | Source code | No | Source code | Output-based |
| 478 | 2007 | JET | General purpose | Source code | No | General purpose | Output-based |
| 479 | 2016 | Acceleo | General purpose | Structured data | Yes | General purpose | Output-based |
| 480 | 2004 | Other | Programming Language | Source code | No | Source code | Output-based |
| 481 | 2011 | EGL | General purpose | Source code | Yes | General Purpose | Output-based |

Table II.XV: Classification table 448–481

# Appendix III

## Templates of each tool for the experiments

#### III.0.0.0.1 Templates for experiment A

#### III.0.0.0.2 EGL

The following template was executed as presented.

```
Store: FooBar

[% var total : Integer = 0 ; %]
[%for (modl in Model) { %]
[%for (invoice in modl.invoice) { %]
Cashier: [%=invoice.meta.cashier%]
Transaction date: [%=invoice.meta.date%]

************************
[%for (items in invoice.items) { %]
Name: [%=items.name%]
Type: [%=items.type%]
Price: [%=items.price%]

[% total += items.price; %]
[%}%]
************************

[% if (invoice.category.eContainer().isTypeOf(Model)) { %]
Category: [%=invoice.category.name%]
Discount:15%
[% total = total * (1 - 15 / 100); %]
Tax: [%=invoice.taxRate%]%
[% total = total * (1 + invoice.taxRate / 100); %]
Total: [%= total %]$
[% } %]
[% else { %]
Category: [%=invoice.category.name%]
Discount:10%
[% total = total * (1 - 10 / 100); %]
Tax: [%=invoice.taxRate%]%
[% total = total * (1 + invoice.taxRate / 100); %]
Total: [%= total %]$
[%}%]
[%}%]
[%}%]
```

Listing III.1: EGL template for the experiment A

##### III.0.0.0.3 Acceleo

The following template was executed as presented.

```
[comment encoding = UTF-8 /]
[module modelTest('http://www.InvoiceMM.org/generate','http://www.eclipse.org/emf/2002/
    Ecore')]

[template public generateElement(model : Model)]
[comment @main/]
[file ('invoice.txt', false, 'UTF-8')]
Store: FooBar

[for (invoice : invoice | self.invoice)]
[generateinvoice(invoice)/]
[/for]
[/file]
[/template]

[template public generateinvoice(invoice : invoice)]
[for (meta : Metadata | invoice.meta)]
[generateMetadata(meta)/]
[/for]

***********************
[for (item : Item | invoice.items)]
[generatedItem(item)/]
[/for]
***********************

[if (invoice.category->size() > 0)]
Category:[invoice.category.name/]
[if (invoice.category.eContainer().oclIsTypeOf(Model))]
```

```
Discount: 15%
Tax:[invoice.taxRate/]%
Total:[(invoice.items.oclAsType(Item).price -> sum() * (1 - 15 / 100)) * (1 + invoice.
    taxRate / 100)/]$
[else]
Discount: 10%
Tax:[invoice.taxRate/]%
Total: [(invoice.items.oclAsType(Item).price -> sum() * (1 - 10 / 100)) * (1 + invoice.
    taxRate / 100)/]$
[/if]
[/if]

[/template]

[template public generatedItem(elements : Item)]
Name:[elements.name/]
Type:[elements.type/]
[/template]

[template public generatedItem(elements : PricedItem)]
Name:[elements.name/]
Type:[elements.type/]
Price:[elements.price/]

[/template]

[template public generateMetadata(meta : Metadata)]
Cashier: [meta.cashier/]
Transaction date: [meta.date/]
[/template]
```

Listing III.2: Acceleo template for the experiment A

**III.0.0.0.4 Xpand** The following template was executed as presented, a long with the Java and Xtend extensions.

```
«IMPORT InvoiceMM»

«DEFINE main FOR Model»
```

```
«FILE "invoice.txt"»
Store: FooBar
«EXPAND invoice FOR invoice»
«ENDFILE»
«ENDDEFINE»

«DEFINE invoice FOR Invoice»
«EXPAND meta FOR meta»
***********************
«EXPAND item FOREACH items»
***********************
Category: «category.name»
«IF this.category.eContainer.toString() == "Model"»
Discount: 15%
Tax: «taxRate»%
Total: «getTotal()»$
«ELSE»
Discount: 10%
Tax: «taxRate»%
Total: «getTotal()»$
«ENDIF»
«ENDDEFINE»

«DEFINE meta FOR Metadata»
Cashier: «cashier»
Transaction date: «date»
«ENDDEFINE»

«DEFINE item FOR Item»
Nom: «name»
Type: «type»

«ENDDEFINE»

«DEFINE item FOR PricedItem»
Name: «name»
Type: «type»
Price: «price»
«addPrice()»

```

```
«ENDDEFINE»
```

Listing III.3: Xpand template for the experiment A

**III.0.0.0.5 Xtend2** The following template was executed as presented.

```
package template

import org.eclipse.emf.common.util.URI
import org.eclipse.emf.ecore.resource.impl.ResourceSetImpl
import org.eclipse.emf.ecore.xmi.impl.XMIResourceFactoryImpl
import org.eclipse.emf.ecore.resource.Resource

import InvoiceMM.Model
import InvoiceMM.Invoice
import InvoiceMM.Item
import InvoiceMM.Category
import InvoiceMM.Metadata
import InvoiceMM.PricedItem
import InvoiceMM.impl.InvoiceMMPackageImpl

class MyCodeGenerator {

static long sum = 0
static long total = 0
static long subtotal = 0

def Object generate(String file) {
InvoiceMMPackageImpl.init();
doEMFSetup
val resourceSet = new ResourceSetImpl
val resource = resourceSet.getResource(URI.createURI(file), true)
for (content : resource.contents.filter(typeof(Model))) {
return generateCode(content)
}
}

def generateCode(Model model){
'''
Store: FooBar
«generateinvoice(model.invoice)»
'''
}

def generateinvoice(invoice invoice){
'''
«generateMetadata(invoice.meta)»
***************
«FOR items : invoice.items»
«generateItems(items as Item)»
«ENDFOR»
***************
Category: «invoice.category.name»
«IF invoice.category.eContainer instanceof Model»
Discount: 15%
Subtotal after discount: «subtotal = sum * (1 - 15 / 100)»$
Tax: «invoice.taxRate»%
Total: «total = sum * (1 + invoice.taxRate / 100)»$
«ELSE»
Discount: 10%
Subtotal after discount: «subtotal = sum * (1 - 10 / 100)»$
Tax: «invoice.taxRate»%
Total: «total = subtotal * (1 + invoice.taxRate / 100)»$
«ENDIF»
'''
}

def generateMetadata(Metadata meta) {
'''
Cashier: «meta.cashier»
Transaction date: «meta.date»
'''
}

def generateItem(Item items){
'''
Nom:«items.name»
Type:«items.type»
'''
}
```

```

def generateItems(PricedItem items){
'''
Name: «items.name»
Type: «items.type»
Price: «items.price»
«subTotal(items)»

'''
}

def static subTotal(Item item) {
sum += item.price
}

def doEMFSetup() {
// EPackage$Registry.INSTANCE.put(MyPackage.eINSTANCE.nsURI, MyPackage.eINSTANCE)
Resource$Factory.Registry.INSTANCE.extensionToFactoryMap.put("xmi", new
    XMIResourceFactoryImpl);
// InvoiceMMPackage mp = InvoiceMMPackage.eINSTANCE;
}
}
```

Listing III.4: Xtend2 template for the experiment A

**III.0.0.0.6 JET** The following template was executed as presented. The argument that is passed to the template is the Java object *model*.

```
<%@ jet package="generator"
class="ModelTestJet" imports ="java.util.Iterator
org.eclipse.emf.common.util.EList
InvoiceMM.Model
InvoiceMM.Invoice
InvoiceMM.Item
InvoiceMM.PricedItem"
%>
<% Model model = (Model) argument; %>
<% long subtotal = 0; %>
<% long total = 0; %>
```

```

Store: FooBar

<% invoice invoice = model.getinvoice(); %>
Cashier: <%= invoice.getMeta().getCashier() %>
Transaction date: <%= invoice.getMeta().getDate() %>
********************************
<% EList<Item> items = invoice.getItems();
for(Iterator<Item> iterator = items.iterator(); iterator.hasNext();) {
Item item = iterator.next();
%>
Name: <%= item.getName() %>
Type: <%= item.getType() %>
Price: <%= ((PricedItem)item).getPrice() %> CAD
<% subtotal += ((PricedItem)item).getPrice(); %>

<%}%>
********************************
<% if(invoice.getCategory().eContainer().eClass().getName().equals("Model")) { %>
Category: <%= invoice.getCategory().getName() %>
Discount: 15%
<% subtotal = subtotal * (1 - 15 / 100); %>
Tax: <%= invoice.getTaxRate() %>%
<% total = subtotal * (1 + invoice.getTaxRate()/100); %>
Total: <%= total %>$
<%}%>
<% else { %>
Category: <%= invoice.getCategory().getName() %>
Discount: 10%
<% subtotal = subtotal * (1 - 10 / 100); %>
Tax: <%= invoice.getTaxRate() %>%
<% total = subtotal * (1 + invoice.getTaxRate()/100); %>
Total: <%= total %>$
<%}%>
```

Listing III.5: JET template for the experiment A

**III.0.0.0.7 Velocity** The following template was executed as presented. The argument that is passed to the template is the Java object *invoice*.

```
#set ($subtotal = 0 )
#set ($total = 0)
#set ($d = "$")
FooBar
Cashier: $invoice.meta.cashier
Transaction date: $invoice.meta.date

*******************************

#foreach ($item in $invoice.items)
Name: $item.name
Type: $item.type
Price: $item.price $d
#set($subtotal = $subtotal + $item.price)

#end
*******************************

#if ($invoice.category.eContainer().eClass().name == "Model")
Category: $invoice.category.name
Discount: 15%
#set ($subtotal = $subtotal * (1 - 15 / 100))
Tax: $invoice.taxRate %
#set ($total = $subtotal * (1 + $invoice.taxRate / 100))
Total: $total $d
#else
Category: $invoice.category.name
Discount: 10%
#set ($subtotal = $subtotal * (1 - 10 / 100))
Tax: $invoice.taxRate %
#set ($total = $subtotal * (1 + $invoice.taxRate / 100))
Total: $total $d
#end
```

Listing III.6: Velocity template for the experiment A

#### III.0.0.0.8 T4 The following template was executed as presented.

```
<#@ template debug="true" hostspecific="false" language="C#" #>
<#@ assembly name="System.Core" #>
<#@ import namespace="System.Linq" #>
<#@ import namespace="System.Text" #>
<#@ import namespace="System.Collections.Generic" #>
<#@ output extension=".txt" #>
<#@ import namespace="System.Xml.Linq" #>
<#@ import namespace="System.IO" #>
<#@ assembly name="System.Xml" #>
<#@ assembly name="System.Xml.Linq" #>
<#@ assembly name="System" #>

Store: FooBar

<# long subtotal = 0; long total = 0;#>
<# XElement element = getModel();
  string cashier = element.Element("meta").Attribute(XName.Get("cashier")).Value;
  string date = element.Element("meta").Attribute(XName.Get("date")).Value;#>

Cashier: <#= cashier #>
Transaction date: <#= date#>

******************

<# foreach (XElement item in element.Elements("items"))
{
  string name = item.Attribute(XName.Get("name")).Value;
  string type = item.Attribute(XName.Get("type")).Value;
  string price = item.Attribute(XName.Get("price")).Value;
  subtotal += long.Parse(price); #>
  Name: <#= name #>
  Type: <#= type #>
  Price: <#= price #>$

<# } #>

******************
<# XElement category = getCategory(); #>
<# string invoice_category = category.Attribute(XName.Get("name")).Value; #>
Category: <#= invoice_category #>
Discount: 15%
<#= subtotal = subtotal *- (1 - 15 / 100) #>
Tax: 15%
Total: <#= total = subtotal * (1 + 15 / 100) #>$

<#+

// getting all the elements from model reference
private XElement getModel()
{
 string modelpath = GetModelFilePath(fileName);
 XDocument model = XDocument.Load(modelpath);
 XElement elements = model.Element("InvoiceMM");
 if(elements == null)
 {
 Error("No such elements");
 return null;
 }
 return elements.Element("invoice");
}

private XElement getCategory()
{
 string modelpath = GetModelFilePath(fileName);
 XDocument model = XDocument.Load(modelpath);
 XElement elements = model.Element("InvoiceMM");
 if(elements == null)
 {
 Error("No such elements");
 return null;
 }
 return elements.Element("rootCat");
}

// getting the input model
private string GetModelFilePath(string fileName)
{
 FileInfo fi = new FileInfo("R:\\Lechanceux\\T4\\visual studio 2012\\Projects\\
     modelTest\\modelTest\\" + fileName);
 string currentFolder = fi.Directory.FullName;
 string mod = Path.Combine(currentFolder, fileName);
 if (File.Exists(mod))
 {
 return mod;
 }
 Error ("File does not exist");
 return string.Empty;
}
#>
```

Listing III.7: T4 template for the experiment A

**III.0.0.0.9 StringTemplate** The following template was executed as presented. The argument that are passed to the template are the Java objects *invoice* and *total*.

```
Store: fooBar
Cashier: <invoice.meta.cashier>
Transaction date: <invoice.meta.date>
*******************
<invoice.items: { item | Name: <item.name> <\n>Type: <item.type> <\n>Price: <item.price
    > <\n><\n>}>
*******************
Category: <invoice.category.name>
Discount: 15%
Tax: <invoice.taxRate>%
Total: <total>
```

Listing III.8: StringTemplate template for the experiment A

**III.0.0.0.10 XSLT** The following template was executed as presented.

```
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:my="my:my">
<xsl:script implements-prefix="my"/>
<xsl:template match="/">
<!-- TODO: Auto-generated template -->
```

```
<xsl:apply-templates/>
</xsl:template>

<xsl:template match="TestModel">
Best Buy
<xsl:for-each select="invoice">
Cashier: <xsl:value-of select="meta/@cashier"/>
Transaction date: <xsl:value-of select="meta/@date"/>
*******************
<xsl:for-each select="items">
Name: <xsl:value-of select="@name"/>
Type: <xsl:value-of select="@type"/>
Price: <xsl:value-of select="@price"/> CAD
<xsl:text>&#xa;</xsl:text>
</xsl:for-each>
*******************
<xsl:variable name="invoice_cat_id" select="@category"/>
<xsl:for-each select="../rootCat">
<xsl:choose>
<xsl:when test="attribute::id = $invoice_cat_id">
Category: <xsl:value-of select="attribute::name"/>
</xsl:when>
</xsl:choose>
<xsl:variable name="category" select="../rootCat"/>
<xsl:choose>
<xsl:when test="parent::category = parent::invoice>
Discount: 15%
Tax: 15%
Total: <xsl:value-of select="(sum(items/@price) * (1 + 15 div 100)) * (1 + 15 div 100)"
    />$
</xsl:when>
<xsl:otherwise>
Discount: 10%
Tax: 15%
Total: <xsl:value-of select="(sum(items/@price) * (1 + 15 div 100)) * (1 + 15 div 100)"
    />$
</xsl:otherwise>
</xsl:choose>

</xsl:for-each>
```

```
</xsl:template>
```

Listing III.9: XSLT template for the experiment A

#### III.0.0.0.11 Templates for experiment B

#### III.0.0.0.12 EGL

The following template was executed as presented.

```
Store: FooBar
[% var total : Integer = 0 ; %]
[%for (modl in Model) { %]
[%for (model in Model) { %]
[%for (invoice in Invoice) { %]
Cashier: [%=invoice.meta.cashier%]
Transaction date: [%=invoice.meta.date%]

*************************
[%for (items in invoice.items) { %]
Name: [%=items.name%]
Type: [%=items.type%]
Price: [%=items.price%]

[% total += items.price; %]
[%}%]
*************************

Category: [%=invoice.category.name%]
[% invoice.category.getCategoryLevel(model.rootCat, invoice.category, 0, total, invoice
    .taxRate); %]
[%}%]
[%}%]
[% operation Category getCategory(catRef:Category, cat:Category, index:Integer, total:
    Integer, taxRate:Integer):Integer{%]
[% if (cat.name = catRef.name) { %]
[% if (index > 30) { %]
Discount: 15%
[% total = total * (1 - 15 / 100); %]
Tax: [%=taxRate%]%
[% total = total * (1 + taxRate / 100); %]
Total: [%= total %]$
[% } %]

[% else { %]
Discount: 10%
[% total = total * (1 - 10 / 100); %]
Tax: [%=taxRate%]%
[% total = total * (1 + taxRate / 100); %]
Total: [%= total %]$
[% } %]
[% } %]

[% for (subCat in catRef.subs) { %]
[% subCat.getCategory(subCat, cat, index+1); %]
[% } %]
[% } %]
```

Listing III.10: EGL template for the experiment B

**III.0.0.0.13 Acceleo** The following template was executed as presented.

```
[comment encoding = UTF-8 /]
[module modelTest('http://www.InvoiceMM.org/generate','http://www.eclipse.org/emf/2002/
    Ecore')]

[template public generateElement(model : Model)]
[comment @main/]
[file ('invoice.txt', false, 'UTF-8')]
Store: FooBar

[for (invoice : invoice | self.invoice)]
[generateinvoice(invoice)/]
[/for]
[/file]
[/template]

[template public generateinvoice(invoice : invoice, model : Model)]
[for (meta : Metadata | invoice.meta)]
[generateMetadata(meta)/]
[/for]
```

```

***********************
[for (item : Item | invoice.items)]
[generatedItem(item)/]
[/for]
***********************

[for (cat : Category | invoice.category)]
[getCategoryLevel(model.rootCat,invoice.category,0,invoice.items.oclAsType(Item).price
    -> sum(),invoice.taxRate)/]
[/for]
[/template]

[template public getCategory(catRef:Category, cat:Category,index:Integer,total:Integer,
    taxRate:Integer)]
[if (cat.name = catRef.name)]
[if (index > 30)]
Discount: 15%
Tax:[taxRate/]%
Total:[(total * (1 - 15 / 100)) * (1 + taxRate / 100)/]$
[else]
Discount: 10%
Tax:[taxRate/]%
Total: [(total * (1 - 10 / 100)) * (1 + taxRate / 100)/]$
[/if]
[/if]

[for (subCat : Category | cat.subs)]
[getCategory(subCat, cat, index+1, total,taxRate)/]
[/for]
[/template]

[template public generatedItem(elements : Item)]
Name:[elements.name/]
Type:[elements.type/]
[/template]

[template public generatedItem(elements : PricedItem)]
Name:[elements.name/]
Type:[elements.type/]
```

```
Price:[elements.price/]

[/template]

[template public generateMetadata(meta : Metadata)]
Cashier: [meta.cashier/]
Transaction date: [meta.date/]
[/template]
```

Listing III.11: Acceleo template for the experiment B

#### III.0.0.0.14 Xtend2 The following template was executed as presented.

```
package template

import org.eclipse.emf.common.util.URI
import org.eclipse.emf.ecore.resource.impl.ResourceSetImpl
import org.eclipse.emf.ecore.xmi.impl.XMIResourceFactoryImpl
import org.eclipse.emf.ecore.resource.Resource

import InvoiceMM.Model
import InvoiceMM.Invoice
import InvoiceMM.Item
import InvoiceMM.Category
import InvoiceMM.Metadata
import InvoiceMM.PricedItem
import InvoiceMM.impl.InvoiceMMPackageImpl

class MyCodeGenerator {

static long sum = 0
static long total = 0


def Object generate(String file) {
InvoiceMMPackageImpl.init();
doEMFSetup
val resourceSet = new ResourceSetImpl
val resource = resourceSet.getResource(URI.createURI(file), true)
for (content : resource.contents.filter(typeof(Model))) {
```

```
return generateCode(content)
}}

def generateCode(Model model){
'''
Store: FooBar
«generateinvoice(model.invoice, model)»
'''
}

def generateinvoice(Invoice invoice, Model model){
'''
«generateMetadata(invoice.meta)»
***************
«FOR items : invoice.items»
«generateItems(items as Item)»
«ENDFOR»
***************
Category: «invoice.category.name»
«generateCategory(model.rootCat, invoice.category, 0, sum, invoice.taxRate)»
'''
}

def getCategory(Category catRef, Category cat, int index, long total, int taxRate){
'''
«IF cat.name.equals(catRef.name)»
«IF index > 30»
Discount: 15%
Tax: «taxRate»%
Total: «total * (1 + taxRate / 100)»$
«ELSE»
Discount: 10%
Tax: «taxRate»%
Total: «total * (1 + taxRate / 100)»$
«ENDIF»
«ENDIF»

«FOR subCat : catRef.subs»
«generateCategory(subCat, cat, index+1, total, taxRate)»
«ENDFOR»
```

```
'''
}

def generateMetadata(Metadata meta) {
'''
Cashier: «meta.cashier»
Transaction date: «meta.date»
'''
}

def generateItem(Item items){
'''
Nom:«items.name»
Type:«items.type»
'''
}

def generateItems(PricedItem items){
'''
Name: «items.name»
Type: «items.type»
Price: «items.price»
«subTotal(items)»

'''
}

def static subTotal(Item item) {
sum += item.price
}

def doEMFSetup() {
// EPackage$Registry.INSTANCE.put(MyPackage.eINSTANCE.nsURI, MyPackage.eINSTANCE)
Resource$Factory.Registry.INSTANCE.extensionToFactoryMap.put("xmi", new
    XMIResourceFactoryImpl);
// InvoiceMMPackage mp = InvoiceMMPackage.eINSTANCE;
}}
```

Listing III.12: Xtend2 template for the experiment B

**III.0.0.0.15 T4** The following template was executed as presented.

```
<#@ template debug="true" hostspecific="false" language="C#" #>
<#@ assembly name="System.Core" #>
<#@ import namespace="System.Linq" #>
<#@ import namespace="System.Text" #>
<#@ import namespace="System.Collections.Generic" #>
<#@ output extension=".txt" #>
<#@ import namespace="System.Xml.Linq" #>
<#@ import namespace="System.IO" #>
<#@ assembly name="System.Xml" #>
<#@ assembly name="System.Xml.Linq" #>
<#@ assembly name="System" #>

Store: FooBar

<# long subtotal = 0; long total = 0;#>
<# XElement element = getModel();
string cashier = element.Element("meta").Attribute(XName.Get("cashier")).Value;
string date = element.Element("meta").Attribute(XName.Get("date")).Value;#>

Cashier: <#= cashier #>
Transaction date: <#= date#>

******************

<# foreach (XElement item in element.Elements("items"))
{
string name = item.Attribute(XName.Get("name")).Value;
string type = item.Attribute(XName.Get("type")).Value;
string price = item.Attribute(XName.Get("price")).Value;
subtotal += long.Parse(price); #>
Name: <#= name #>
Type: <#= type #>
Price: <#= price #>$

<# } #>

******************
<# XElement category = getCategory(); #>
<# string invoice_category = category.Attribute(XName.Get("name")).Value; #>
Category: <#= invoice_category #>
<# int index = getIndex(category, "last", 0); #>
<# if(index > 30)
{ #>
Discount: 15%
<#= subtotal = subtotal * (1 - 15 / 100) #>
Tax: 15%
Total: <#= total * (1 + 15 / 100) #>$
<# } #>
<# else
{ #>
Discount: 10%
<#= subtotal = subtotal * (1 - 10 / 100) #>
Tax: 15%
Total: <#= total * (1 + 15 / 100) #>$
<# } #>
<#+

//recursive function
private int getIndex(XElement category, string mod_cat, int index)
{
//Console.WriteLine(category.Attribute(XName.Get("name")).Value + " " + mod_cat);
if(category.Attribute(XName.Get("name")).Value.Equals(mod_cat))
{
return index;
}
foreach(XElement cat in category.Elements("subs"))
{
 getIndex(cat, mod_cat, index+1);
}
return 0;
}

// getting all the elements from model reference
private XElement getModel()
{
string modelpath = GetModelFilePath(fileName);
XDocument model = XDocument.Load(modelpath);
XElement elements = model.Element("InvoiceMM");
if(elements == null)
{
Error("No such elements");
return null;
}
return elements.Element("invoice");
}

private XElement getCategory()
{
string modelpath = GetModelFilePath(fileName);
XDocument model = XDocument.Load(modelpath);
XElement elements = model.Element("InvoiceMM");
if(elements == null)
{
Error("No such elements");
return null;
}
return elements.Element("rootCat");
}

// getting the input model
private string GetModelFilePath(string fileName)
{
FileInfo fi = new FileInfo("R:\\Lechanceux\\T4\\visual studio 2012\\Projects\\modelTest
    \\modelTest\\" + fileName);
string currentFolder = fi.Directory.FullName;
string mod = Path.Combine(currentFolder, fileName);
if (File.Exists(mod))
{
return mod;
}
Error ("File does not exist");
return string.Empty;
}#>
```

Listing III.13: T4 template for the experiment B