Computer Languages, Systems & Structures 52 (2018) 43–62

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

Systematic mapping study of template-based code generation

Eugene Syriani*, Lechanceux Luhunu, Houari Sahraoui

Department of computer science and operations research (DIRO), University of Montreal, Montreal, Quebec, Canada

ARTICLE INFO

Article history:
Received 18 August 2017
Revised 24 November 2017
Accepted 30 November 2017
Available online 7 December 2017

Keywords:
Code generation
Systematic mapping study
Model-driven engineering

ABSTRACT

Context: Template-based code generation (TBCG) is a synthesis technique that produces code from high-level specifications, called templates. TBCG is a popular technique in model-driven engineering (MDE) given that they both emphasize abstraction and automation. Given the diversity of tools and approaches, it is necessary to classify existing TBCG techniques to better guide developers in their choices.

Objective: The goal of this article is to better understand the characteristics of TBCG techniques and associated tools, identify research trends, and assess the importance of the role of MDE in this code synthesis approach.

Method: We survey the literature to paint an interesting picture about the trends and uses of TBCG in research. To this end, we follow a systematic mapping study process.

Results: Our study shows, among other observations, that the research community has been diversely using TBCG over the past 16 years. An important observation is that TBCG has greatly benefited from MDE. It 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 to many domains.

Conclusion: TBCG is now a mature technique and much research work is still conducted in this area. However, some issues remain to be addressed, such as support for template definition and assessment of the correctness and quality of the generated code.

1. Introduction

Code generation has been around since the 1950s, taking its origin in early compilers [1]. Since then, software organizations have been relying on code synthesis techniques in order to reduce development time and increase productivity [2]. Automatically generating code is an approach where the same generator can be reused to produce many different artifacts according to the varying inputs it receives. It also helps detecting 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 [3], using a meta-object protocol [4], or aspect-oriented programming [5]. Since the mid-1990's, 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

* Corresponding author.
E-mail addresses: syriani@iro.umontreal.ca (E. Syriani), luhunukl@iro.umontreal.ca (L. Luhunu), sahraoui@iro.umontreal.ca (H. Sahraoui).

https://doi.org/10.1016/j.cl.2017.11.003
1477-8424/

environments started to include a template mechanism in their framework such as Microsoft Text Template Transformation Toolkit (T4)[1] for .NET and Velocity[2] for Apache.

Model-driven engineering (MDE) has advocated the use of model-to-text transformations as a core component of its paradigm [6]. TBCG is a popular technique in MDE given that they both emphasize abstraction and automation. MDE tools, such as Acceleo[3] and Xpand[4], allow developers to generate code from high-level models without worrying on how to parse and traverse input models. We can find today TBCG applied in a plethora of computer science and engineering research.

The software engineering research community has focused essentially on primary studies proposing new TBCG techniques, tools and applications. However, to the best of our knowledge, there is no classification, characterization, or assessment of these studies available yet. Therefore, in this paper, we conducted a systematic mapping study (SMS) of the literature in order to understand the trends, identify the characteristics of TBCG, assess the popularity of existing tools within the research community, and determine the influence that MDE has had on TBCG. We are interested in various facets of TBCG, such as characterizing the templates, the inputs, and outputs, along with the evolution of the amount of publications using TBCG over the past 16 years.

The remainder of this paper is organized as follows. In Section 2, we introduce the necessary background on TBCG and discuss related work. In Section 3, we elaborate on the methodology we followed for this SMS, and on the results of the paper selection phase. The following sections report the results: the trends and evolution of TBCG in Section 4, the characteristics of TBCG according to our classification scheme in Section 5, the relationships between the different facets in Section 6, how TBCG tools have been used in primary studies in Section 7, and the relation between MDE and TBCG in Section 8. In Section 9, we answer our research questions and discuss limitations of the study. Finally, we conclude in Section 10.

2. Background and related work

We briefly explain TBCG in the context of MDE and discuss related work on secondary studies about code generation.

2.1. Code generation

In this paper, we view code generation as in automatic programming [1] 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 [7].

Code generation is an important model-to-text transformation that ensures the automatic transformation of a model into code. Organization are adopting the use of code generation since it reduces the development process time and increases the productivity. Generating the code using the most appropriate technique is even more crucial since it is the key to benefit from all the advantages code synthesis offers to an organization. Nowadays, TBCG has raised to be the most popular synthesis technique available. Using templates can quickly become a complex task especially when the model should satisfy a certain condition before a template fragment is executed.

As Balzer [8] 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. [9] 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 [10]. Sometimes, relying too much on code generators may produce an overly general solution that may not necessarily be optimal for a specific problem.

[1] https://msdn.microsoft.com/en-us/library/bb126445.aspx
[2] http://velocity.apache.org/
[3] http://www.eclipse.org/acceleo/
[4] http://wiki.eclipse.org/Xpand

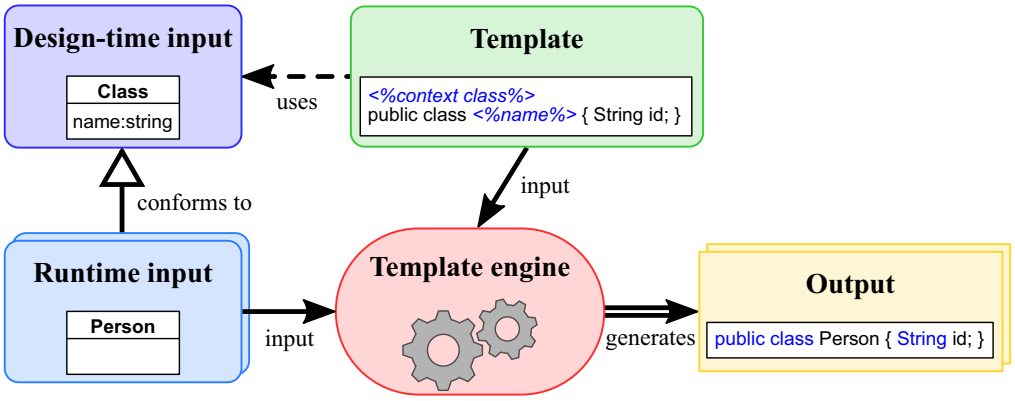

Fig. 1. Components of TBCG.

2.2. Model-to-text transformations

In MDE, model transformations can have different purposes [11], such as translating, simulating, or refining models. One particular kind of model transformation is devoted to code generation with model-to-text transformations (M2T) [12]. In general, M2T transforms a model (often in the form of a graph structure) into a linearized textual representation. M2T is used to produce various text artifacts, e.g., to generate code, to serialize models, to generate documentation and reports, or to visualize and explore models. As such, code generation is a special case of M2T, where the output artifacts are executable source code. There are commonly five different code generation approaches present in the literature [7,12]:

Visitor based approaches consist of programmatically traversing the internal representation of the input, while relying on an API dedicated to manipulate the input, to write the output to a text stream. This requires to program directly the creation of the output files and storing the strings while navigating through the data structure of the input. The implementation typically makes use of the visitor design pattern [13]. This approach is used in [3] to generate Java code from a software architecture description language.

Meta-programming is a language extension approach, such as using a meta-object protocol, that relies on introspection in order to access the abstract syntax tree of the source program. For example, in OpenJava [4], 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 expand the existing one, such as with the C++ standard template library or C macro preprocessor instructions. The generation instructions can be defined at a higher-level of abstraction, either using a dedicated language distinct from the base language (e.g., for macros) or as a dedicated library as in [14].

Code annotations are in-line descriptions that added to statement declarations (e.g., class definition) that can either be internally transformed into more expanded code (e.g., attributes in C#) or that are processed by a tool other than the compiler of the language (e.g., the specification of documentation comments processed by Javadoc). This approach is used in [15].

Template based is described below.

2.3. Template-based code generation

The literature agrees on a general definition of M2T code generation [12] and on templates. Jörges [7] 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 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* (sometimes refereed to as *template processor*) 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 consists of a set of *templates* specified in a formalism, where the specification of a template is based on a *design-time input* such that, when executed, it reads a *run-time input* to produce a textual *output*.

For example, the work in [16] 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.

2.4. Literature reviews on code generation

In evidence-based software engineering [17], 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 SMS aim at classifying and performing a thematic analysis on a topic [18]. SMS is a secondary study method that has been adapted from other disciplines to software engineering in [19] and later evolved by Petersen et al. in [20]. 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. [21] 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. [22] 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. [21], they did not follow a systematic and repeatable process.

Dominguez et al. [23] 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. [24] performed a systematic mapping study 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 M2T. 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. [12] proposed a feature model providing a terminology to characterize model transformation approaches. They distinguished two categories for M2T 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. [25] extended the feature model of Czarnecki et al. to focus on template-based M2T tools. Their classification is centered exclusively on tool-dependent features. 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 [12] but focuses only on a feature model for M2T. The difference with our study is that it focuses on a feature diagram and deals with tool-dependent features only.

There are also other systematic reviews performed on other related topics. For example, the study in [26] performed a SMS on domain-specific modeling languages (DSL). They analyzed 390 papers to portray the published literature on DSLs, which is comparable to the size of our corpus. Also, the study in [27] presents a systematic literature review on software product lines engineering in the context of DSLs. Similar to our study, they propose a definition for the life-cycle of language product lines and use it to analyze how the literature has an impact this life-cycle. Another SMS recently published in [28] presents a taxonomy of source code labeling.

3. Research methods

In order to analyze the topic of TBCG, we conducted a SMS following the process defined by Petersen et al. in [20] and summarized in Fig. 2. The definition of research question is discussed in Section 3.1. The search conduction is described in Section 3.2. We present the screening of papers in Section 3.3. The relevant papers are obtained based on the criteria presented in Section 3.3.1 and Section 3.3.2. The elaboration of the classification scheme is described in Section 3.4. Finally, we detail the selection of the papers in Section 3.5.

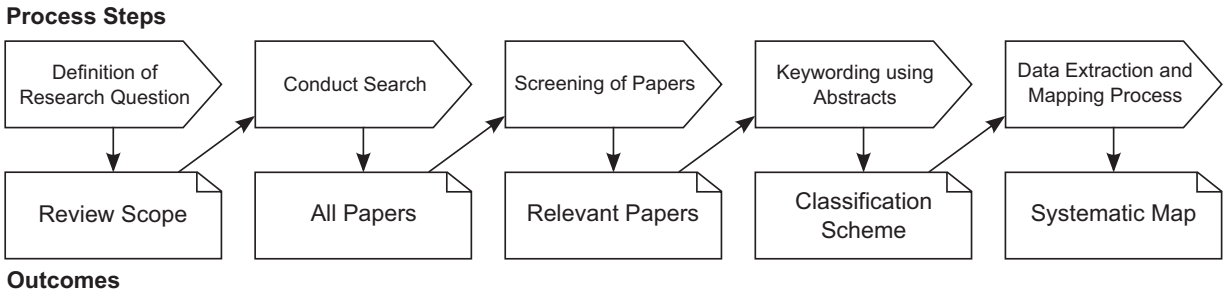

Fig. 2. The systematic mapping study process we followed from Peterson et al.

3.1. 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 through research publications.
2. *What are the characteristics of template-based code generation approaches?* We want to identify major characteristics of these techniques and their tendencies.
3. *To what extent are template-based code generation tools being used in research?* 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.

3.2. 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 necessarily the main contribution. For example, Buchmann et al. [29] 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 retrieved publications based on their title, abstract, or full text (when available) mentioning the keywords "template" and its variations, "code", and "generation" and synonyms with their variations (e.g., "synthesis"). We used the following search query for all digital libraries, using their respective syntax:

templat AND "code generat*" OR "code synthesi*"*

We validated our search with a sample of 100 pilot papers we preselected. These papers were chosen from papers we apriori knew should be included and resulting from a preliminary pilot search online. We iterated over different versions of the search strings until all pilot papers could be retrieved from the digital libraries.

3.3. Screening procedure

Screening is the most crucial phase in a SMS [20]. 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. 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.

3.3.1. 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. "Use" is taken in a large sense when it is explicit that a TBCG contributes to the core of the paper (not in the related work).

3.3.2. 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., proceed-

ings 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.

3.4. Classification scheme

We elaborated a classification scheme by combining our general knowledge with the information extracted from the abstracts during the screening phase. We classify all papers along different categories that are of interest in order to answer our research questions. This helps analyzing the overall results and gives an overview of the trends and characteristics of TBCG. The categories we classified the corpus with are the following:

- *Template style*: We characterize the level of customization and expressiveness of the templates used in the code generation approach. *Predefined* 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 construct to use for many-cardinality associations, e.g., `Array` or `ArrayList` for Java templates. *Output-based* 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 Fig. 1. *Rule-based* style puts the focus of the template 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*: We characterize the language of the design-time input that is necessary to develop templates. The run-time input is an instance that conforms to it. *General-purpose modeling language* is for generic languages reusable across different domains that are not programming languages, such as UML. *Domain-specific modeling language* is for languages targeted for a particular domain, for example, where the run-time input is a Simulink model. *Schema* is for structured data definitions, such as XML or database schema. *Programming language* is for well-defined programming languages, where the run-time input is source code.
- *Output type*: We characterize the output of the code generator (more than one category can be selected when multiple artifacts are generated). *Source code* is executable code conforming to a specific programming language. *Structured data* is for code that is not executable, such as HTML.
- *Application scale*: We characterize the scale of the artifact on which the TBCG approach is applied with respect to the evaluation or illustration cases in the paper. The presence of a formal case study that either relied on multiple data sources or subjects qualified as *large scale* application. *Small scale* was used mainly when there was only one example or toy examples in the paper that illustrated the TBCG. *No application* if for 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, Compilers, Bio-medicine*, etc.
- *Orientation*: We distinguish *industrial* papers, where at least one author is affiliated to industry, from *academic* papers otherwise.
- *Tool*: We capture the tool 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* within the research community when it is used in at least 1% of the papers. Otherwise, we classify it as *other*.
- *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.

3.5. Paper selection

Table 1 summarizes the flow of information through the selection process of this study. This section explains how we obtained the final corpus of papers to classify and analyze.

Table 1
Evolution of paper corpus during the study process.

Phase	Number of papers
Collection	
Engineering Village	4043
Scopus	932
SpringerLink	2671
Initial corpus	*5131*
Screening	
Excluded during screening	4553
Included	*578*
Classification	
Excluded during classification	99
Final corpus	*481*

3.5.1. 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[5], Scopus[6], and SpringerLink[7]. The first two cover typical software engineering editors (IEEE Xplore, ACM Digital Library, Elsevier). However, from past experiences [24], they do not include all of Springer publications. We used the search string from Section 3.2 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 3.3.2. We then removed automatically all duplicates using EndNote software. This resulted in 5 131 candidate papers for the screening phase.

3.5.2. Screening

Based on the exclusion criteria stated in Section 3.3.2, 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 [30]. After all the reviewers completed screening the papers they were assigned, the tool calculates an inter-rater agreement coefficient. In our case, the Cohens Kappa coefficient was 0.813. This high value shows that the reviewers were in almost perfect agreement.

Among the initial corpus of candidate papers, 4556 were excluded, 551 were included and 24 received conflicting ratings. During the screening, the senior reviewer systematically verified 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 4549. Almost all cases of conflicts were about a disagreement 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 4553 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 [31], 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 2.2. Also, more than a quarter of the papers were in the compilers or embedded system domains, where code generation is programmed imperatively rather than declaratively specified 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.

3.5.3. 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 2.3. As shown in [33] 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 [34].

The final corpus[8] considered for this study contains 481 papers.

[5] https://www.engineeringvillage.com/
[6] https://www.scopus.com/
[7] http://link.springer.com/
[8] The complete list of papers is available online at http://www-ens.iro.umontreal.ca/~luhunukl/survey/classification.html

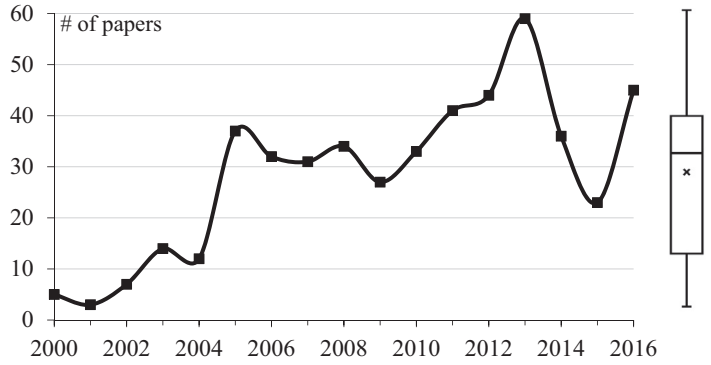

Fig. 3. Evolution of papers in the corpus.

Table 2
Most popular venues.

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

4. Evolution of TBCG

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

4.1. General trend

Figure 3 reports the number of papers per year, averaging around 28. The general trend indicates that the number of publications with at least one TBCG 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, the most prolific period was in 2013, where we notice a significant peak with 2.4 times the average numbers of publications observed in the previous years. Fig. 3 then shows sudden a 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 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, 2016 is one standard deviation above the average.

4.2. 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 2 shows the most popular venues that have at least five papers. 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, SOSYM, and ECMFA are the three most popular venues[9] with a total of 66 publications between them. This is very significant given that the

[9] We grouped UML conference with MODELS and ECMDA-FA with ECMFA.

Table 3
Distribution of the template style facet.

Output-based	Predefined	Rule-based
72%	24%	4%

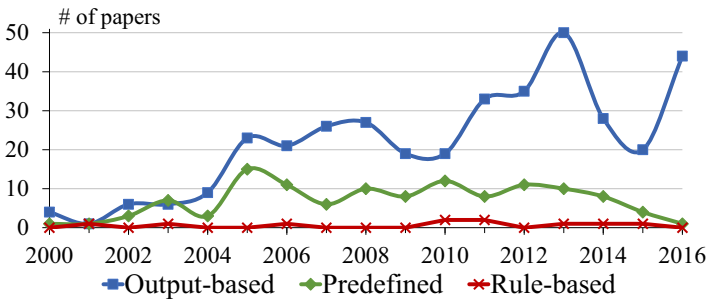

Fig. 4. Template style evolution.

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. The remaining papers were published as book chapters, workshops or other publication type. Interestingly, we notice a steady increase in journal articles, reaching a maximum of 15 in 2016.

5. Characteristics of template-based code generation

We examine the characteristics of TBCG using the classification scheme presented in Section 3.4.

5.1. Template style

As Table 3 illustrates, the vast majority of the publications follow the *output-based* style. This consists of papers like [35], where Xpand is used to generate workflow code used to automate modeling tools. There, it is the final output 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 Fig. 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 [36] 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 [37] 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. 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 [38] 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.

5.2. Input type

General purpose languages account for almost half of the design-time input of the publications, as depicted in Table 4. UML (class) diagrams, which are used as metamodels for code generation, are the most used for 87% of these papers as in [35]. Other popular general-purpose languages that were used are, for example, the architecture analysis and design language [39] and feature diagrams [40]. The *schema* category comes second with 21% of the papers. For example, a database

Table 4
Distribution of the input type facet.

General purpose	Schema	Domain specific	Programming language
48%	22%	20%	10%

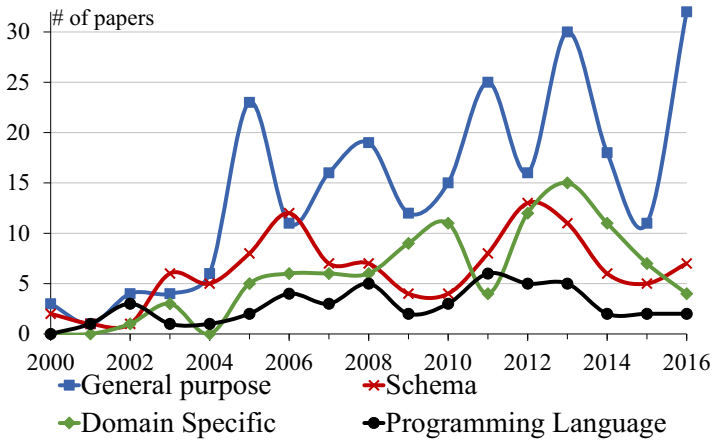

Fig. 5. Evolution of the design-time input type.

schema is used as input at design-time in [41] to generate Java for a system that demonstrates that template can improve software development. Also, an XML schema is used in [42] as design-time input to 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 schema. They have been gaining popularity and gradually reducing the gap with general-purpose languages. For example in [43], 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 [44] 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, as depicted in Fig. 5. 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.

5.3. Output type

An overwhelming majority of the papers use TBCG to generate *source code* (81%), in contrast with 19% was structured data (though some papers output both types). Table 5 shows the distribution of the output languages that appeared in more than five papers, representing 74% of the corpus. This includes papers like [45] where Java code is generated an adaptable access control tool for electronic medical records. Java and C are the most targeted programming languages. Writing a program manually often requires proved abilities, especially with system and hardware languages, such as VHDL [46]. This is why 8% of all papers generate low level source codes. Generation of *structured data* includes TBCG of mainly XML and HTML files. For example [47] produces both HTML and XML as parts of the web component to ease regression testing. In addition, we found that around 4% of the papers generate combinations of at least two output types. This includes papers such as [48] that generate both C# and HTML from a domain-specific model.

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

5.4. Application scale

As depicted in Table 6, 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 [49] that uses Acceleo to generate hundreds of lines

Table 5
Distribution of the most popular output languages.

Java	C	HTML	C++	C#	XML	VHDL	SQL	AspectJ	SystemC
33%	11%	7%	7%	4%	3%	3%	2%	2%	2%

Table 6
Distribution of application scale facet

Large scale	Small scale	No application
63%	32%	5%

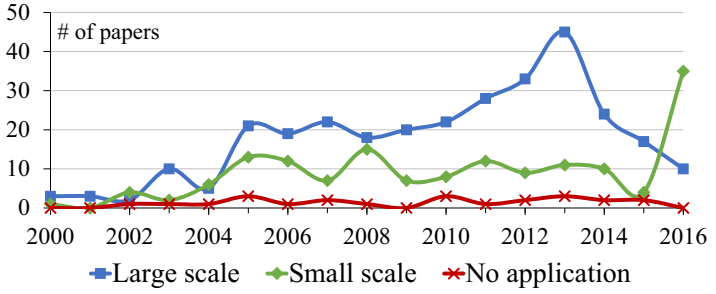

Fig. 6. Application scale evolution.

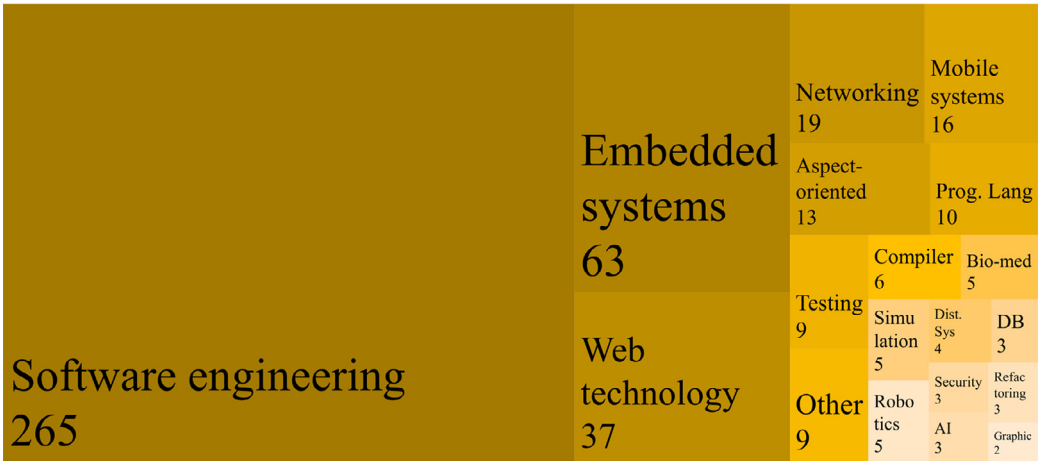

Fig. 7. Distribution of application domain facet.

of aspect-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 [50] in which a small concocted example shows the generation process with the Epsilon Generation Language (EGL)[10]. *No application* was used in 5% of the publications. This includes papers like [51] 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.

As depicted in Fig. 6, most papers illustrated their TBCG using large scale applications up to 2015. In this period, while it follows the general trend of papers, the other two categories remained constant over the years. However, in 2016, we observe a major increase of small scale for TBCG.

5.5. Application domain

The tree map in Fig. 7 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 [52] 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 automatic programming. In this category, we found papers like [53] 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 [54] where the authors worked to enhance the development dynamic web sites. *Networking* obtains 4% of the papers, such as [55] where code is generated for a telephony service network. *Compiler* obtains 1% of the papers, such as [56] 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* [57], *artificial intelligence* [58], and *graphics* [59].

[10] http://www.eclipse.org/epsilon/doc/egl/

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.

5.6. 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 [16]. Industry oriented papers have gradually increased since 2003 until they reached a peak in 2013.

6. Relations between characteristics

To further characterize the trends observed in Section 5, we identified significant and interesting relations between the different facets of the classification scheme, using SPSS.

6.1. 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 5.2. An example is when the design-time input is UML, the run-time input is always a UML diagram as in [57]. Such a strong relationship is also noticeable in [60] with programming languages and source code, as well as in [58] 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.

6.2. Other interesting relations

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

6.2.1. Template style

Figure 8 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

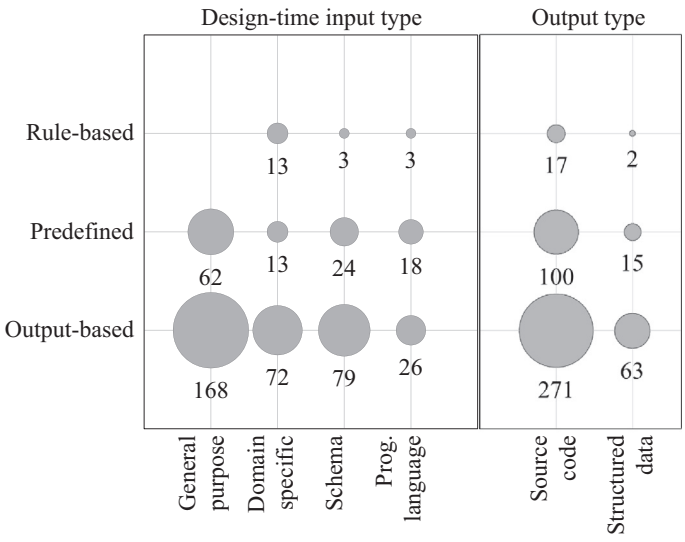

Fig. 8. Relation between template style (vertical) and input/output types (horizontal).

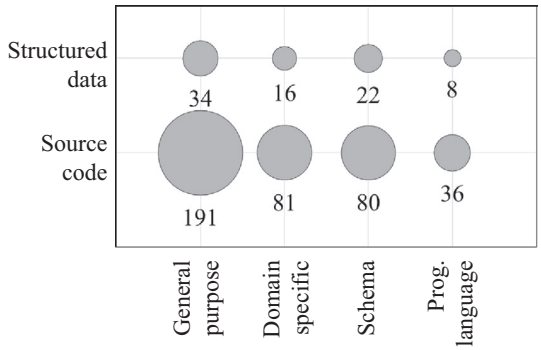

Fig. 9. Relation between output (vertical) and design-time input (horizontal) types showing the number of papers in each intersection.

Table 7
Distribution of the tool facet.

Named MDE	Unspecified	Other	Named not MDE
41%	28%	23%	8%

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.

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 for output-based (27%).

We found that popular tools were used twice as 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.

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

6.2.2. Input type

The bubble chart in Fig. 9 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.

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%.

6.2.3. Output type, application scale, and orientation

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 than industrials: 18% vs. 6% and 3% vs. 1%., respectively. We found that 65% of the papers without application are from the academy.

7. Template-based code generation tools

Table 7 shows that half of the papers used a popular TBCG tool (*named*). The other half used less popular tools (the *other* category), did not mention any TBCG tool, or implemented the code generation directly for the purpose of the paper.

7.1. Popular tools in research

Figure 10 shows the distribution of popular tools used in at least 1% of the papers, i.e., five papers. We see that only 6/14 popular tools follow MDE approaches. 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 [61] and the fact that they are MDE tools [16]. They both have an OCL-like language for the dynamic part and rely on a metamodel specified in Ecore as design-time input.

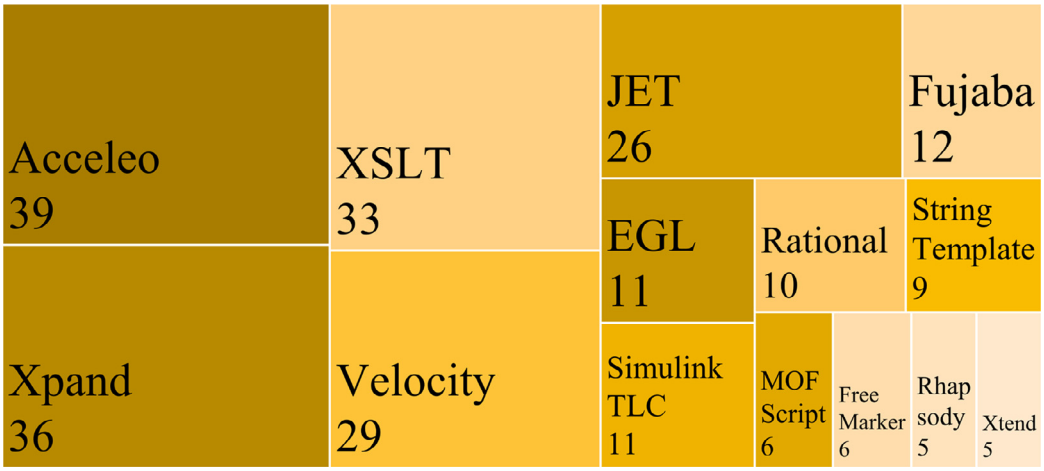

Fig. 10. Popular tools.

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 [62]. XSLT follows the template and filtering strategy. It matches each tag of the input document and applies the corresponding template.

JET [63] and Velocity [53] 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 developers generate the Java code specific for the synthesis of code to help developers implement the code generation.

StringTemplate [64] 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 [65], Rational [66], and Rhapsody [67]. 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 [7,8].

7.2. Unspecified and other tools

As depicted in Table 7, 28% of the papers did not specify the tool that was used, as in [68] where the authors introduce the concept of a meta-framework to resolve issues involved in extending the life of applications. Furthermore, 23% of the papers used less popular tools, present in less than five papers, such as T4 [44] and Cheetah [56], 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 [69]. Other examples of less popular tools are Groovy template [47], Meta-Aspect-J [70], and Jinja2 [71]. 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.

7.3. Trends of tools used

Each one of these tools had a different evolution over the years. Unspecified tools were prevalent 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.

7.4. Characteristics of tools

We have also analyzed each popular tool with respect to the characteristics presented in Section 5. 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 JET and Cheetah. 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. T4 can be integrated with MDE artifacts (e.g., generate code based on a domain-specific language in Visual studio). However, all four papers in the corpus that implemented TBCG in T4 did not use an MDE approach.

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 [72]. Among the most popular tools, Xpand is the most in the industry.

7.5. Application scale

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[11] use large scale applications. Small scale is likely to be used in unpopular tools rather than popular tools.

7.6. Tool support

In total, we found 82 different tools named in the corpus. Among them 54% are no longer supported (i.e., no release, update or commit since the past two years). Interestingly, 55% of the tools have been developed by the industry. 70% of these industry tools are still supported, in contrast with 51% for the academic tools. As one would expect, the tendency shows that tools that are frequently mentioned in research papers are still supported and are developed by industry.

8. MDE and template-based code generation

Overall, 64% of the publications followed MDE techniques and principles. For example in [73], 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 6. For example, the authors in [40] developed a system that handles the implementation of dependable applications and offers a better certification process for the fault-tolerance mechanisms.

As Fig. 11 shows, the evolution of the MDE category reveals that MDE-based approach started over passing non MDE-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, MDE-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. 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 MDE-based papers. We only found one paper that uses a programming language as design-time input with MDE [74]. This analysis also shows that the year 2005 clearly marked the shift from schema to domain-specific design-time inputs, as witnessed in Section 5.2. 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.

[11] Refers to the union of other and unspecified categories of the tool facet.

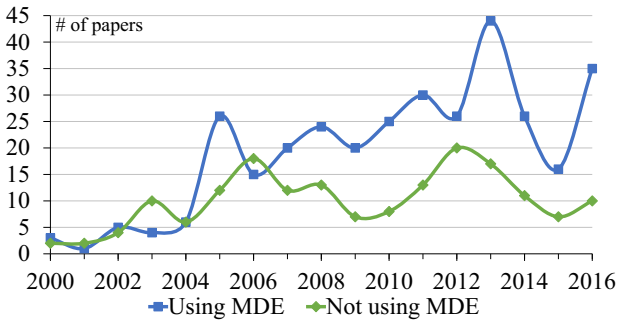

Fig. 11. Evolution of the MDE facet.

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.

9. Discussion

9.1. 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.

9.2. 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 modeling 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.

9.3. RQ3: To what extent are TBCG tools being used in research?

In this study, we discovered a total of 82 different tools for TBCG that are mentioned in research papers. 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. Most popular tool are actively supported by 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 There are many TBCG tools that are popular in industry that fall under the “Other tools” category because they are rarely reported in the scientific literature (under 1% of the papers in our corpus). Since this study is a literature review, the presence of a tool in this study is biased towards the what is published, and may not reflect the reality in industry. This is a common threat of SMS.

9.4. 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 [8]. These are, in fact, the pillar principles of MDE and domain-specific modeling [2]. 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.

9.5. Identified challenges

After thoroughly analyzing each paper in the corpus, we noted several problems that the research community has neglected in TBCG. First, we found 66% of the papers did not provide any assessment of the code generation process or the generated output. We only found one paper with a formal verification of the generated code using non-functional requirement analysis [75]. Furthermore, the TBCG can be verified through benchmarks as in [55]. Second, we found no paper that investigates efficiency of code generation. Researchers may be inspired from other related communities, such as compiler optimization [60]. Third, designing templates requires skillful engineering. Good practices, design patterns and other forms of good reusable idioms would be of great value to developers [76].

9.6. Threats to validity

The results presented in this survey have depended on many factors that could potentially threaten its validity.

9.6.1. Construction validity

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 Fig. 3, 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 standard deviation 15.76. A more systematic selection process would have been difficult to be exhaustive about TBCG. We chose to obtain the best possible coverage at the cost of duplications. Nevertheless, the size of the corpus we classified is about ten times larger than other systematic reviews related to code generation (see Section 2.4). We are, therefore, confident that this sample is a representative subset of all relevant publications on TBCG.

Another potential limitation is the query formulation for the keyword search. 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. In this study, we are only interested in code generation. Therefore we discarded articles where TBCG was used for reporting or mass mailing, for example. We believe that our sample is large enough that additional papers will not significantly affect the general trends and results. We are fully aware that, since TBCG is widely used in practice, industries have their own tools and many have not been published in academic venues. Our goal was not to be exhaustive, but to get a representative sample.

9.6.2. 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 [77], 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 [55], 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.

9.6.3. External validity

The results we obtained are based on TBCG only. Even though our classification scheme includes facets like 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 2.2.

9.6.4. Conclusion validity

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. Also, our study follows the methodology described in [20].

10. Conclusion

This paper reports the results of a large survey we conducted 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. 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 modeling languages as input. It 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.

The study also revealed that, although the research in TBCG is mature enough, there are many open issues that can be addressed in the future, upstream and downstream the generation itself. Upstream, the definition of templates is not a trivial task. Supporting the developers in such a definition is a must. Downstream, methods and techniques need to be defined to assess the correctness and quality of the generated code.

We believe this survey will be of benefit to someone familiar with code generation by knowing how their favorite tool ranks in popularity within the research community, the relevance and importance of the use of templates, and in which context TBCG has been applied (application domain). The relations across categories in Section VI show non-intuitive results as well. The paper also promotes MDE in fields that have not been traditionally exposed to it.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.cl.2017.11.003.

References

[1] Rich C, Waters RC. Automatic programming: myths and prospects. Computer 1988;21(8):40–51.
[2] Kelly S, Tolvanen J-P. Domain-specific modeling: enabling full code generation. John Wiley & Sons; 2008.
[3] Bonta E, Bernardo M. Padl2java: a Java code generator for process algebraic architectural descriptions. In: Proceedings of the european conference on software architecture. IEEE; 2009. p. 161–70.
[4] Tatsubori M, Chiba S, Killijian M-O, Itano K. OpenJava: a class-based macro system for Java. In: Reflection and software engineering. In: LNCS, 1826. Springer; 2000. p. 117–33.
[5] Lohmann D, Blaschke G, Spinczyk O. Generic advice: on the combination of AOP with generative programming in AspectC++. In: Proceedings of international conference on generative programming and component engineering. In: LNCS, 3286. Berlin Heidelberg: Springer; 2004. p. 55–74.
[6] Kleppe AG, Warmer J, Bast W. MDA explained. The model driven architecture: practice and promise. Addison-Wesley; 2003.
[7] Jörges S. Construction and evolution of code generators 7747. Ch 2 The state of the art in code generation. Berlin Heidelberg: Springer; 2013. p. 11–38.
[8] Balzer R. A 15 year perspective on automatic programming. Trans Softw Eng 1985;11(11):1257–68.
[9] Floch A, Yuki T, Guy C, Derrien S, Combemale B, Rajopadhye S, et al. Model-driven engineering and optimizing compilers: a bridge too far?. In: Model Driven Engineering Languages and Systems. In: LNCS, 6981. Springer Berlin Heidelberg; 2011. p. 608–22.
[10] Stahl T, Voelter M, Czarnecki K. Model-driven software development – technology, engineering, management. John Wiley & Sons; 2006.
[11] Lúcio L, Amrani M, Dingel J, Lambers L, Salay R, Selim GM, et al. Model transformation intents and their properties. Softw Syst Model 2014;15(3):685–705.
[12] Czarnecki K, Helsen S. Feature-based survey of model transformation approaches. IBM Syst J 2006;45(3):621–45.
[13] Gamma E, Helm R, Johnson R, Vlissides J. Design patterns: elements of reusable object-oriented software. Addison Wesley Professional; 1994.
[14] Beckmann O, Houghton A, Mellor M, Kelly PH. Runtime code generation in C++ as a foundation for domain-specific optimisation. In: Domain-Specific Program Generation. In: LNCS, 3016. Berlin Heidelberg: Springer; 2004. p. 291–306.
[15] Córdoba I, de Lara J. ANN: a domain-specific language for the effective design and validation of Java annotations. Comput Lang Syst Struct 2016;45:164–90.
[16] Jugel U, Preußner A. A case study on API generation. In: System analysis and modeling: about models. In: LNCS, 6598. Springer; 2011. p. 156–72.
[17] Kitchenham BA, Dyba T, Jorgensen M. Evidence-based software engineering. In: Proceedings of international conference on software engineering. Washington, DC, USA: IEEE Computer Society; 2004. p. 273–81.
[18] Kitchenham BA, Budgen D, Brereton OP. Using mapping studies as the basis for further research - a participant-observer case study. Inf Softw Technol 2011;53(6):638–51.
[19] Brereton P, Kitchenham BA, Budgen D, Turner M, Khalil M. Lessons from applying the systematic literature review process within the software engineering domain. J Syst Softw 2007;80(4):571–83.
[20] Petersen K, Feldt R, Mujtaba S, Mattsson M. Systematic mapping studies in software engineering. In: Proceedings of the 12th international conference on evaluation and assessment in software engineering, EASE'08, 17. British Computer Society; 2008. p. 68–77.
[21] Mehmood A, Jawawi DN. Aspect-oriented model-driven code generation: a systematic mapping study. Inf Softw Technol 2013;55(2):395–411.

[22] Gurunule D, Nashipudimath M. A review: analysis of aspect orientation and model driven engineering for code generation. Procedia Comput Sci 2015;45:852–61.
[23] Domíguez E, Pérez B, Rubio AL, Zapata MA. A systematic review of code generation proposals from state machine specifications. Inf Softw Technol 2012;54(10):1045–66.
[24] Batot E, Sahraoui H, Syriani E, Molins P, Sboui W. Systematic mapping study of model transformations for concrete problems. In: Model-driven engineering and software development. IEEE; 2016. p. 176–83.
[25] Rose LM, Matragkas N, Kolovos DS, Paige RF. A feature model for model-to-text transformation languages. In: Modeling in software engineering. IEEE; 2012. p. 57–63.
[26] Kosar T, Bohra S, Mernik M. Domain-specific languages: a systematic mapping study. Inf Softw Technol 2016;71:77–91.
[27] Méndez-Acuña D, Galindo JA, Degueule T, Combemale B, Baudry B. Leveraging software product lines engineering in the development of external DSLs: a systematic literature review. Comput Lang Syst Struct 2016;46:206–35.
[28] Matús Sulír JP. Labeling source code with metadata: a survey and taxonomy. In: Proceedings of federated conference on computer science and information systems. In: Workshop on Advances in Programming Languages (WAPL'17), CFP1785N-ART. IEEE; 2017. p. 721–9.
[29] Buchmann T, Schwägerl F. Using meta-code generation to realize higher-order model transformations. In: Proceedings of international conference on software technologies; 2013. p. 536–41.
[30] Seriai A, Benomar O, Cerat B, Sahraoui H. Validation of software visualization tools: a systematic mapping study. In: Proceedings of IEEE working conference on software visualization. VISSOFT; 2014. p. 60–9.
[31] Liu Q. C++ techniques for high performance financial modelling. WIT Trans Model Simul 2006;43:1–8.
[32] Fang M, Ying J, Wu M. A template engineering based framework for automated software development. In: Proceeding of the 10th international conference on computer supported cooperative work in design. IEEE; 2006. p. 1–6.
[33] Singh A, Schaeffer J, Green M. A template-based approach to the generation of distributed applications using a network of workstations. IEEE Trans Parallel Distrib Syst 1991;2(1):52–67.
[34] O'Halloran C. Automated verification of code automatically generated from simulink ®. Autom Softw Eng 2013;20(2):237–64.
[35] Dahman W, Grabowski J. UML-based specification and generation of executable web services. In: System analysis and modeling. In: LNCS, 6598. Springer; 2011. p. 91–107.
[36] Gessenharter D. Mapping the UML2 semantics of associations to a Java code generation model. In: Proceedings of international conference on model driven engineering languages and systems. In: LNCS, 5301. Springer; 2008. p. 813–27.
[37] Valderas P, Pelechano V, Pastor O. Towards an end-user development approach for web engineering methods. In: Proceedings of international conference on advanced information systems engineering, 4001. Springer; 2006. p. 528–43.
[38] Hemel Z, Kats LC, Groenewegen DM, Visser E. Code generation by model transformation: a case study in transformation modularity. Softw Syst Model 2010;9(3):375–402.
[39] Brun M, Delatour J, Trinquet Y. 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. IEEE; 2008. p. 257–62.
[40] Buckl C, Knoll A, Schrott G. Development of dependable real-time systems with Zerberus. In: Proceedings of the 11th IEEE pacific rim international symposium on dependable computing; 2005. p. 404–8.
[41] Li J, Xiao H, Yi D. Designing universal template for database application system based on abstract factory. In: Computer science and information processing. IEEE; 2012. p. 1167–70.
[42] Gopinath VS, Sprinkle J, Lysecky R. Modeling of data adaptable reconfigurable embedded systems. In: International conference and workshops on engineering of computer based systems. IEEE; 2011. p. 276–83.
[43] Buckl C, Regensburger M, Knoll A, Schrott G. Models for automatic generation of safety-critical real-time systems. In: Availability, reliability and security. IEEE; 2007. p. 580–7.
[44] Fischer T, Kollner C, Hardle M, Muller Glaser KD. Product line development for modular FPGA-based embedded systems. In: Proceedings of symposium on rapid system prototyping. IEEE; 2014. p. 9–15.
[45] Chen K, Chang Y-C, Wang D-W. Aspect-oriented design and implementation of adaptable access control for electronic medical records. Int J Med Inform 2010;79(3):181–203.
[46] Brox M, Sánchez-Solano S, del Toro E, Brox P, Moreno-Velo FJ. CAD tools for hardware implementation of embedded fuzzy systems on FPGAs. IEEE Trans Ind Inform 2013;9(3):1635–44.
[47] Fraternali P, Tisi M. A higher order generative framework for weaving traceability links into a code generator for web application testing. In: Proceedings of international conference on web engineering. In: LNCS, 5648. Springer; 2009. p. 340–54.
[48] Vokáč M, Glattetre JM. 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, 3713. Springer; 2005. p. 492–506.
[49] Kokar M, Baclawski K, Gao H. Category theory-based synthesis of a higher-level fusion algorithm: an example. In: Proceedings of international conference on information fusion; 2006. p. 1–8.
[50] Hoisl B, Sobernig S, Strembeck M. Higher-order rewriting of model-to-text templates for integrating domain-specific modeling languages. In: Model–Driven Engineering and Software Development. SCITEPRESS; 2013. p. 49–61.
[51] Ecker W, Velten M, Zafari L, Goyal A. The metamodeling approach to system level synthesis. In: Proceedings of design, automation & test in Europe conference & exhibition. IEEE; 2014. p. 1–2.
[52] Behrens T, Richards S. Statelator-behavioral code generation as an instance of a model transformation. In: Proceedings of international conference on advanced information systems engineering. In: LNCS, 1789. Springer; 2000. p. 401–16.
[53] Durand SH, Bonato V. A tool to support Bluespec SystemVerilog coding based on UML diagrams. In: Proceedings of annual conference on IEEE industrial electronics society. IEEE; 2012. p. 4670–5.
[54] Schattkowsky T, Lohmann M. Rapid development of modular dynamic web sites using UML. In: Proceedings of international conference on the unified modeling language. In: LNCS, 2460. Springer; 2002. p. 336–50.
[55] Buezas N, Guerra E, de Lara J, Martín J, Monforte M, Mori F, et al. Umbra designer: graphical modelling for telephony services. In: Proceedings of european conference on modelling foundations and applications. In: LNCS, 7949. Berlin Heidelberg: Springer; 2013. p. 179–91.
[56] Manley R, Gregg D. A program generator for intel AES-NI instructions. In: Proceedings of international conference on cryptology. In: LNCS, 6498. Springer; 2010. p. 311–27.
[57] Phillips J, Chilukuri R, Fragoso G, Warzel D, Covitz PA. The caCORE software development kit: streamlining construction of interoperable biomedical information services. BMC Med Inform Decis Mak 2006;6(2):1–16.
[58] Fu J, Bastani FB, Yen I-L. Automated AI planning and code pattern based code synthesis. In: Proceedings of international conference on tools with artificial intelligence. IEEE; 2006. p. 540–6.
[59] Possatto MA, Lucrédio D. Automatically propagating changes from reference implementations to code generation templates. Inf Softw Technol 2015;67:65–78.
[60] Ghodrat MA, Givargis T, Nicolau A. Control flow optimization in loops using interval analysis. In: Proceedings of international conference on compilers, architectures and synthesis for embedded systems. ACM; 2008. p. 157–66.
[61] Guduvan A-R, Waeselynck H, Wiels V, Durrieu G, Fusero Y, Schieber M. A meta-model for tests of avionics embedded systems. In: Proceedings of international conference on model-driven engineering and software development; 2013. p. 5–13.
[62] Adamko A. Modeling data-oriented web applications using UML. In: Proceedings of the international conference on computer as a tool, EUROCON 2005, 1. IEEE; 2005. p. 752–5.

[63] Kövi A, Varró D. An eclipse-based framework for AIS service configurations. In: Proceedings of the 4th international symposium on service availability, ISAS. In: LNCS, 4526. Springer; 2007. p. 110–26.
[64] Anjorin A, Saller K, Rose S, Schürr A. A framework for bidirectional model-to-platform transformations. In: Proceedings of the 5th international conference on software language engineering, SLE 2012. In: LNCS, 7745. Berlin Heidelberg: Springer; 2013. p. 124–43.
[65] Burmester S, Giese H, Schäfer W. Model-driven architecture for hard real-time systems: from platform independent models to code. In: Proceedings of European conference on model driven architecture-foundations and applications. In: LNCS, 3748. Berlin Heidelberg: Springer; 2005. p. 25–40.
[66] Brown AW, Conallen J, Tropeano D. Introduction: models, modeling, and model-driven architecture (MDA). In: Proceedings of international conference on model-driven software development. Berlin Heidelberg: Springer; 2005. p. 1–16.
[67] Basu AS, Lajolo M, Prevostini M. A methodology for bridging the gap between UML and codesign. In: UML for SOC design. US: Springer; 2005. p. 119–46.
[68] Furusawa T. Attempting to increase longevity of applications based on new SaaS/cloud technology. Fujitsu Sci Tech J 2010;46:223–8.
[69] Muller P-A, Studer P, Fondement F, Bézivin J. Platform independent web application modeling and development with Netsilon. Softw Syst Model 2005;4(4):424–42.
[70] Antkiewicz M, Czarnecki K. Framework-specific modeling languages with round-trip engineering. In: Model driven engineering languages and systems. In: LNCS, 4199. Berlin Heidelberg: Springer; 2006. p. 692–706.
[71] Hinkel G, Denninger O, Krach S, Groenda H. Experiences with model-driven engineering in neurorobotics. In: Proceedings of the 12th european conference on modelling foundations and applications, ECMFA 2016. Cham: Springer International Publishing; 2016. p. 217–28.
[72] Kulkarni V, Barat S, Ramteerthkar U. Early experience with agile methodology in a model-driven approach. In: Model driven engineering languages and systems. In: LNCS, 6981. Springer; 2011. p. 578–90.
[73] Touraille L, Traoré MK, Hill DR. A model-driven software environment for modeling, simulation and analysis of complex systems. In: Proceedings of symposium on theory of modeling & simulation. SCSC; 2011. p. 229–37.
[74] Fertalj K, Kalpic D, Mornar V. Source code generator based on a proprietary specification language. In: Proceedings of Hawaii international conference on system sciences, 9. IEEE; 2002. p. 3696–704.
[75] Yen I-L, Goluguri J, Bastani F, Khan L, Linn J. A component-based approach for embedded software development. In: International symposium on object-oriented real-time distributed computing. ISORC 2002. IEEE Computer Society; 2002. p. 402–10. doi:10.1109/ISORC.2002.1003805.
[76] Luhunu L, Syriani E. Comparison of the expressiveness and performance of template-based code generation tools. In: Software Language Engineering. ACM; 2017.
[77] Ma M, Meissner M, Hedrich L. 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. IEEE; 2012. p. 9–12.