Survey on Template-based Code Generation

Lechanceux Luhunu DIRO University of Montreal Montreal, QC, Canada Email: lechanceux.luhunu.kavuya@umontreal.ca Eugene Syriani DIRO University of Montreal Montreal, QC, Canada Email: syriani@iro.umontreal.ca

Abstract—Among the various model-to-text transformation paradigms, template-based code generation (TBCG) is the most popular in MDE. Given the diversity of tools and approaches, it is necessary to classify and compare existing TBCG techniques to provide appropriate support to developers. We conduct a systematic mapping study of the literature to better understand the trends and characteristics of TBCG techniques over the past 16 years. We also evaluate the expressiveness, performance and scalability of the associated tools based on a range of models that implement critical patterns.

Index Terms—model-driven engineering, code generation, systematic mapping study, performance study, expressiveness evaluation

I. MOTIVATION AND GOALS

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, and generate documentation or reports. Among the various model-totext 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. 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", and a dynamic part embedded with splices of meta-code that encode the generation logic. It is a popular technique in MDE, as they both emphasize abstraction and automation. Given the diversity of tools and approaches, it is necessary to classify and compare existing TBCG techniques and tools to provide appropriate support to developers.

In this work, we conduct 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, and determine the influence that MDE has had on TBCG over the past 16 years. Based on this SMS, 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.

II. TRENDS OF TBCG

We first present the results of the SMS we conducted on TBCG.

A. Systematic Mapping Study

We followed the process defined in [1] to portray the literature on TBCG. The protocol we followed is described in [2]. The research questions guiding the SMS are: (RQ1) What are the trends in template-based code generation? (RQ2) What are the characteristics of TBCG approaches? (RQ3) To what extent are TBCG tools being used? (RQ4) What is the place of MDE in TBCG? We collected 5 131 papers published between 2000–2016 from online databases that matched the keywords we searched for. After screening all these papers, we obtained a final corpus of 481 papers. We then classified each paper according to a classification scheme (available in [3]) that helps answering our research questions.

B. Evolution of TBCG

Fig. 1 reports the number of papers per year, averaging around 28. This significantly large sample of papers clearly suggests 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 (24%). The only exceptions were a significant boost in 2013 and a dip in 2015. The most popular venues are MODELS, SOSYM, ECMFA. However, we noticed a decrease of publications in MDE venues, indicating that TBCG is now applied in development projects rather than being a critical research problem to solve. 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.

C. Characteristics of TBCG

Output-based templates have always been the most popular style from the beginning (72%). This template style is when the template is syntactically based on the actual target output, such as in [4] that uses Xpand. Nevertheless, there have been some attempts to propose other template styles, like the rulebased style in [5], but they did not catch on (4%). Because of its simplicity to use, the predefined style is probably still popular in practice, e.g., in CASE tools, but less in research papers (24%).

TBCG has been used to synthesize a variety of application code or documents. As expected, the study shows that high-level language inputs (general purpose 48% or domain-specific 22% modeling languages) have prevailed over any other type (schema 20% or programming languages 10%). Specifically for MDE approaches to TBCG, the input to transform is moving from general purpose to domain-specific models.

The study confirms that the community uses TBCG to generate mainly source code (81%), rather than structured data e.g., XML (16%) or natural language documents (3%). This trend is set to continue since the automation of computerized tasks is continuing to gain ground in all fields. TBCG has been implemented in many domains, software engineering (55%) and embedded systems (13%) being the most popular, but also unexpectedly in unrelated domains, such as bio-medicine and finance.

The study revealed a total of 77 different tools for TBCG. Many studies implemented code generation with a custommade tool that was never or seldom reused. This indicates that the development of new tools is still very active. Modelbased tools are the most popular (49%). 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.

D. Role of MDE

The burst of papers in 2005 coincides with the transition form the UML to MODELS conference. MDE venues have 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 [6]. These are, in fact, the pillar principles of MDE and domain-specific



Fig. 2: Invoice metamodel

modeling [7]. 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. These three are the top ranked venues in terms of number of TBCG paper published. All this analysis clearly concludes that the advent of MDE has been driving TBCG research.

III. TOOL EXPRESSIVENESS

We evaluate and compare the nine most popular tools found in the SMS with respect to metamodel patterns that drive the implementation of the dynamic part of the template. The complete evaluation methodology is described in [8].

A. Metamodel Patterns for TBCG

To evaluate the expressiveness of TBCG tools, we identify a minimal set of four common structures found in metamodels that influence TBCG. This is the result of analyzing a plethora of metamodels that were used for TBCG from repositories [9], [10], known metamodel patterns [11], and industrial experiences [12]. We evaluated the tools on the common running example of invoice production, for which the metamodel is depicted in Fig. 2.

1) Navigation: this pattern is when there is a navigable relation between two classes. A template uses it to access the data of a target class related to the class of the current context.

2) Variable dependency: this pattern is like the navigation pattern, but when the template desires to output a value that depends on variables present in other classes.

3) Polymorphism: this pattern takes advantage of an inheritance relationship to reuse parts of the template. The template implements the output for the super class and only what varies for the subclass(es).

4) *Recursion:* this pattern consists of a recursive selfrelation of a class. The template can be reapplied on objects of the same type in a transparent way.

Туре	Tool	Navigation	Variable dependency	Polymorphism	Recursion
Model- based	Acceleo	\checkmark	\checkmark	\checkmark	\checkmark
	Xpand	\checkmark	×	\checkmark	×
	EĜL	\checkmark	\checkmark	\checkmark	\checkmark
	Xtend2	\checkmark	\checkmark	\checkmark	\checkmark
Code- based	JET	\checkmark	\checkmark	×	Х
	Velocity	\checkmark	\checkmark	×	×
	T4	\checkmark	\checkmark	×	\checkmark
	StringTemplate	\checkmark	×	×	×
	XSLT	\checkmark	\checkmark	×	×

TABLE I: Summary of the qualitative evaluation of the tools expressiveness

B. Template expressiveness

Table I summarizes the qualitative evaluation of the expressiveness of each TBCG tools, showing whether it successfully implemented each pattern or not.

All tools successfully implement the trivial *navigation pattern*. For example, to access the meta-data date from the invoice object, all tools use the dot operator. In XSLT, navigating through a composition relation is accomplished with the xsl:value-of expression. It also requires a different strategy when the relation is an association.

We implemented the *variable dependency pattern* to output and calculate the total of the invoice. Acceleo and XSLT have powerful built-in mathematical functions, especially for collection types. EGL, JET, Velocity, T4, and Xtend2 rely on the use of global variables and statement blocks. It was not possible to implement this pattern with StringTemplate (ST) and Xpand "natively". We resorted to extend the template with a Java program to handle the calculations.

We used the *polymorphism pattern* to process the priced items of an invoice, that are subtypes of the abstract item type. 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. In EGL, the content of the superclass template definition block is output, along with the content of the one for the subclass. In JET, Velocity, T4, XSLT, and ST, no template code can be defined for abstract classes. Thus, the developer must replicate the common template code for all possible subclasses.

We implemented the *recursion 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. The dedicated language of T4 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 accumulate a value in a variable. XSLT does not implement this pattern either because there is no trace between the argument that is passed to the function and the variable passed in the initial invocation. It is not possible to implement this pattern in JET, ST, and Velocity due to the absence of typed definition block or function.



Fig. 3: Tool performance, no recursion

IV. PERFORMANCE EVALUATION

To compare the performance of the tools, we generated 10 models conforming to Fig. 2, with a size varying from 10 to 10^5 classes. There are 3 instances of navigation, 7 to 10^5 variable dependencies, 6 to 10^5 ploymorphisms, and 1 to 10^2 recursions. Fig. 3 shows that the execution time increases with the size of the model for all tools. The complete evaluation methodology is described in [8].

Overall, JET is the fastest tool, completing the whole experiment. 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, Velocity and ST are the fastest. T4 is as efficient as JET for smaller models. However, for larger models, it becomes slower than Velocity and ST. Xtend2 outperforms T4 for these models tool, making it the fastest model-based tool. Xpand and XSLT come next. The slowest tools are EGL followed by Acceleo.

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, ST, and XSLT with around a factor of 25. For the remaining tools, the size of the model has a significant effect on their performance.T4 and Acceleo have the worst scale factor.

Enabling the recursion pattern gives a similar trend for the four tools concerned. It did not influence significantly the performance of Acceleo and T4, but Xtend2 performed 10% slower than in Fig. 3. However, EGL performed 10% faster because the dedicated language EOL supports caching [13].

V. CONCLUSION

The community has been diversely using TBCG over the past 16 years, and that research and development is still very active. TBCG has been greatly influenced by MDE. Both model-based and code-based tools are becoming effective development resources in industry. The former are the most capable tools since most of them successfully implemented all the metamodel patterns. However, the latter performed much faster. Although JET is the fastest tool, Xtend2 offers the best compromise between the expressiveness and the performance.

REFERENCES

- [1] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic Mapping Studies in Software Engineering," in *Evaluation and Assessment in Software Engineering*, ser. EASE'08, vol. 17. British Computer Society, 2008, pp. 68–77.
- [2] E. Syriani, L. Luhunu, and H. Sahraoui, "Systematic Mapping Study of Template-based Code Generation," Tech. Rep. arXiv:1703.06353, 2017.
- [3] http://www-ens.iro.umontreal.ca/~luhunukl/survey/classification.html.
- [4] W. Dahman and J. Grabowski, "UML-based specification and generation of executable web services," in *System Analysis and Modeling*, ser. LNCS, vol. 6598. Springer, 2010, pp. 91–107.
 [5] Z. Hemel, L. C. Kats, D. M. Groenewegen, and E. Visser, "Code
- [5] Z. Hemel, L. C. Kats, D. M. Groenewegen, and E. Visser, "Code generation by model transformation: a case study in transformation modularity," *Software & Systems Modeling*, vol. 9, no. 3, pp. 375–402, 2010.
- [6] R. Balzer, "A 15 Year Perspective on Automatic Programming," Transactions on Software Engineering, vol. 11, no. 11, pp. 1257–1268, 1985.
- [7] S. Kelly and J.-P. Tolvanen, Domain-Specific Modeling: Enabling Full Code Generation. John Wiley & Sons, 2008.
- [8] L. Luhunu and E. Syriani, "Comparison of the expressiveness and performance of template-based code generation tools," in *Software Language Engineering*, ser. LNCS. Springer, 2017.
- [9] http://web.emn.fr/x-info/atlanmod/index.php?title=Zoos.
- [10] http://www.remodd.org.
- [11] H. Cho and J. Gray, "Design Patterns for Metamodels," in *Domain-Specific Modeling workshop*, ser. SPLASH '11 Workshops. ACM, 2011, pp. 25–32.
- [12] V. Sousa, E. Syriani, and M. Paquin, "Feedback on How MDE Tools are Used Prior to Academic Collaboration," in *Symposium On Applied Computing*, 2017.
- [13] D. Kolovos, L. Rose, R. F. Paige, and A. Garcia Dominguez, *The Epsilon Book*. Eclipse, 2010.