Aspect-Oriented Code Generation for Integration of Aspect Orientation and Model-Driven Engineering

Abid Mehmood and Dayang N. A. Jawawi

Department of Software Engineering, Faculty of Computing
Universiti Teknologi Malaysia (UTM), 81310 Skudai, Johor, Malaysia
mabid4@live.utm.my, dayang@utm.my

Abstract
Software development can be improved from many perspectives by combining aspect orientation and model-driven engineering techniques. At a higher level, they can be integrated in two different ways: (1) by handling specifics of aspect orientation at modeling level and later generating object-oriented code, or (2) by transforming an aspect model directly into aspect-oriented code. The latter approach has been shown to have advantages over the former. Consequently, different aspect-oriented code generation approaches have appeared in literature. This paper comparatively evaluates some of those existing approaches, which are well-published and can be used in integration with model-driven engineering process. The results of this study indicate that in order to provide this integration, existing aspect code generation need to be improved from various perspectives. Moreover, these results also provide some insight into the prerequisites for a valuable integration of these approaches into a model-driven engineering process.

Keywords: model-driven engineering, model transformation, aspect-oriented software development, code generation, reviews

1. Introduction
Aspect-oriented techniques [1-3] are applied at all levels of system development to specify, model and encapsulate a specific type of concerns known as crosscutting concerns. These concerns, unless handled with aspect-oriented techniques, end up scattered throughout several modules of a system and tangled with other concerns. In this context, some empirical studies have reported the advantages of improved handling of crosscutting concerns through aspect orientation (cf. [4-6]) mainly from perspectives of maintainability, extensibility and reusability.

Model-Driven Engineering (MDE) techniques consider models as the primary development artifact and use them as a basis for obtaining an executable system in different ways. Automatic code generation is a well-known way of getting executable of a system from a given design model. Automatically generated code offers many advantages such as the rapid development of high quality code, reduced number of accidental programming errors, enhanced consistency between design and code to name a few [7].

An integration of Aspect-Oriented Software Development (AOSD) techniques with MDE can enhance software development through the unification of the optimum abstraction mechanisms (which are essence of MDE) and the contemporary techniques for modularization of concerns (which are the core of AO techniques). This integration can be achieved in two different ways. In the first approach, aspect orientation is handled at the modeling level only, by using a model weaver that composes crosscutting and non-crosscutting concerns in such a way that a non-aspect-oriented model is obtained. Object-oriented code generation mechanisms are then applied to obtaining an executable from this model. There is a huge volume of research that contributes to this approach of
integration of AOSD and MDE (cf. [8-10]). However, main drawback of this approach is that clear separation and modularization of concerns conceived at modeling level becomes unavailable once the model has been transformed into object-oriented code. This eventually means leaving behind all benefits of aspect orientation at the time of code generation. On the hand, in the second approach of combining AOSD and MDE, the handling of aspect orientation is passed on to programming level, by transforming aspect-oriented model into aspect-oriented code. Apart from preserving the clear boundaries of concerns, this approach provides a straightforward mapping of AO concepts at modeling level onto constructs available in an AO programming language. It has been shown in literature that maintaining aspect-oriented paradigm from model to code is preferable to transforming aspect-oriented design into object-oriented code, see for instance Ref. [5, 6, 11]. Owing to these benefits of aspect-oriented code generation in the context of MDE, the current study was conducted to investigate different approaches that address aspect-oriented code generation while keeping its integration with MDE in focus.

The rest of this paper is organized as follows. In Section 2, we briefly describe the related work in the context of this study. Section 3 provides details of the evaluation framework and the rationale behind selection of approaches. The results of applying the evaluation framework to the selected approaches are given in Section 4. A brief discussion of the studied approaches from viewpoint of their integration into MDE is presented in Section 5. Section 6 concludes our presentation.

2. Related Work

To the best of our knowledge, maturity and suitability of the aspect-oriented code generation approaches from viewpoint of their integration into MDE process has not been considered in existing literature. However, some closely as well as widely related work in this context is presented in the following.

In an effort to get an overview of existing research in the area of aspect-oriented model-driven code generation, we have conducted a systematic mapping study [12]. The results of this study suggested that further research is needed in the area of AO code generation. So far as AO code generation is concerned, we have previously compared six approaches on the basis of a set of well-defined questions [13]. To the best of our knowledge, that was the first work intended to evaluate specifically the AO code generation approaches. However, that was focused on investigating the features of studied approaches that can contribute to full code generation. The integration of existing approaches with MDE process and prospects thereof were not considered in that study. The current study focuses on the aspect code generation problem from a different perspective, and includes only the approaches that retain high likelihood of integration into a larger context of MDE. Section 3 provides more insight into the selection process.

In the context of aspect orientation, some widely related work has presented comparison of aspect-oriented modeling (AOM) approaches. Wimmer et al. [14] have defined a detailed evaluation framework to evaluate existing AOM approaches with focus on comparability in general. Reina, et al., [15] have investigated some AOM approaches with specific goal of evaluating dependency of each approach on particular platform and on specific concerns. Op de beeck, et al., [16] have presented a comparison of AOM approaches within the context of product line engineering and with the goal to position AOM approaches within software development life cycle for large-scale system. However, the scope of all this work is limited to aspect-oriented modeling notations.

3. Evaluation Framework

In this section, we describe the evaluation framework used for this review. For this purpose, first the rationale behind the selection of approaches is given. This is followed by a description of the comparison approach used.
3.1. Methodology

3.1.1. Selection of Approaches: As we described in previous section, the current study is focused on integration of AOSD into MDE process through generating AO code, rather than considering approaches that generate AO code only. Therefore, the studies that contribute only to AO code generation have been excluded from this study.

Another factor that influenced the selection of approaches was their relation to UML. In fact, the extent and quality of code generated from a modeling notation is directly linked with the comprehensiveness and expressiveness of the underlying modeling notation. This model-code relationship makes the modeling notation one of the decisive factors in selection of approaches. A vast majority of AOM approaches possesses at least one common characteristic, which is their extension from UML. Extending UML for aspect-orientation seems quite natural and convincing since aspect-orientation can be seen as an extension to object-orientation, for which UML is the standard and most widely used modeling language. Therefore, for this study, we have focused only on approaches that use a UML-based modeling notation as their input.

As a result, we have identified five approaches as a representative of our selection criteria described above. These approaches include: mapping MATA models to AspectWerkz code [17], mapping Theme/UML models to AspectJ [18], mapping Reusable Aspect Models (RAM) to AspectJ [19], AspectJ skeleton code generation from Formal Design Analysis Framework (FDAF) models [20] and translation of Join Point Designation Diagrams (JPDDs) to AspectJ [21]. In the remainder of this paper, we refer to these approaches as MATA, Theme, RAM, FDAF and JPDDs, respectively.

3.1.2. Comparison Approach: Our previous experience with classification of research during the mapping study [12] and surveying of aspect code generation [13] have provided us with some insight into how these approaches differ. Moreover, the study of existing literature on aspect-oriented MDE approaches and related surveys given above suggested that there are different perspectives that constitute the prerequisite for an in-depth evaluation of approaches. Therefore, for each perspective, we have constituted a set of fine-grained criterion, which has helped us compare the selected approaches. It has to be emphasized here that the only purpose of the set of criteria given in this section is to provide an objective evaluation of approaches that already present some information on their integration into MDE. That is to say that the criteria distinguishing whether an approach integrates itself with MDE or not has not been included here; it is implicit that all selected approaches support this integration in some way. A summary of different criteria applied for each perspective is given in Figure 1. The findings against each criterion may have three possibilities: ‘yes’, ‘no’, or ‘to some extent’, denoted using the symbols ‘✓’, ‘✗’, and ‘~’ respectively.

The perspectives that serve as basis of our comparison are described in the following sections.

3.2. Transformation Criteria

The criteria under this category refer to approach taken towards model transformation and code generation. These criteria will identify the approach and explore various factors associated with it. They will also address the limitations of approach.

Approach Intelligibility (T.IA): This criterion considers the clarity of the transformation approach used by the surveyed work, through asking questions like if the advantages of transformation / code generation approach over other existing approaches were clearly identified or not, and so on.
Supported Views (T.SV): Transformation approach may support static and/or dynamic views of the system. This criterion investigates which view has been supported by the surveyed approach.

Supported Concerns (T.SC): Transformation may be addressed for crosscutting concerns (i.e., aspects) only, or it may include the support for non-crosscutting concerns (i.e., the base). This criterion refers to investigating whether only aspects are transformed or the base system is also supported.

Approach Advanced-ness (T.AN): Support for advanced features such as existence of interaction mechanisms other than point cuts; the support for ADTs, Collections and Arrays is explored through this criterion.

Algorithmic Maturity (T.AM): This criterion is intended to evaluate the maturity of the transformation algorithms used for model-code transformations. However, the specific perspectives with which maturity is linked for the purpose of this study include the correctness and performance of the algorithms.

3.3. Models Criteria

From a broader perspective, a system can be completely modeled using a combination of structural and behavioral representations. Code generation from structural representation of a system would essentially mean that the code specifying the behavior of entities cannot be obtained. In this context, this category is dedicated to evaluate what different types of models have been supported by the reviewed approach.

Structure Models (M.SM): This criterion evaluates if structure models are supported, and in case they are supported, it identifies the specific structural diagrams that are supported by the surveyed approach.
Behavior Models (M.BM): This criterion refers to support for behavioral models. In case the support for behavior models is found, it further investigates which specific diagrams are supported.

3.4. Validation Criteria

This category considers criteria derived to evaluate the approach used for checking the correctness of approach as a whole.

Approach Transparency (V.AT): The transparency and clarity of the validation mechanism is assessed through the measure of reliability and maturity of technique (e.g. its presence in literature (in contrast with a self-defined, proof-of-concept mechanism), standard of inputs and outputs, and maturity of the comparison approach.

3.5. Extent of Code Criteria

This category considers criteria derived in order to evaluate the approaches from perspective of extent of code generation. An important consideration in this category is the trade-off between the level of details included at the design level and the resultant amount of code generated.

Structure Code (E.SC): This criterion refers to the support for code generation from Class (or other structure) diagrams. In particular, it will ask questions such as whether the Class diagrams are fully implemented for code generation or not.

Behavior Code (E.BC): This criterion investigates the integration of one or more behavior diagrams into the code generation approach. It will specifically focus on the behavior diagrams that are used for the purpose, as well as if full code generation support has been enabled for them or not. In case behavior is not currently supported, this criterion will further investigate to see if a clear path to implementation of behavior has been provided by the approach.

Full Code (E.FC): Code generation approaches may be distinguished between those which support aspect code generation only and the ones that address code generation of both aspect as well as base code. The latter type is deemed capable of full code generation. In cases where base code is not generated by the approach on its own, this criterion will further investigate whether a clear path for integration with an existing code generation approach has been provided. Moreover, since there is always a trade-off between the amount of code generation and the extent of implementation details included in design, this criterion will also investigate if this trade-off has explicitly been considered and reported by the surveyed approach.

3.6. Tool-support Criteria

This category considers criteria that refer to the tool support provided to use and validate the approach.

Tools Sufficiency (S.TS): This criterion investigates whether the tool support provided by the reviewed approach is sufficient to validate it. In case, tool support is sufficient for validation, it further studies the approach to find out whether the provided tools are reliable (not prototypes) to be used in a real-world project environment.

Tools Integration (S.TI): This criterion is intended to evaluate questions such as “Are the tools developed using standard development frameworks” and “Are the tools extendable and allow integration with some IDE/ other tools”.

211
4. Review of Aspect-oriented Code Generation Approaches

This section presents the results of evaluation of the selected approaches. It has to be emphasized here that since results presented in tables are mostly self-explanatory, this section is not intended to discuss them in detail. Rather, we will focus more on the implications of presence or absence of some features on the maturity of the approach. The maturity of the code generation approaches is particularly relevant to reach the long-term goal of fully executable code generation from design models.

Table 1. Comparison on Basis of Transformation Criteria

<table>
<thead>
<tr>
<th>Approach</th>
<th>Intelligibility (T.IA)</th>
<th>Supported Views (T.SV)</th>
<th>Supported Concerns (T.SC)</th>
<th>Approach Advancement (T.AN)</th>
<th>Algorithmic Maturity (T.AM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Theme</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>RAM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>FDAF</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>JPDDs</td>
<td>✓</td>
<td>✓</td>
<td>~</td>
<td>✓</td>
<td>x</td>
</tr>
</tbody>
</table>

4.1. Transformation

Table 1 presents the results of comparison of approaches when transformation criteria are applied. All approaches have reported some distinctions with regard to their model to code transformation mechanism (T.IA). This also applies to approaches which only define mapping from model to code (MATA, Theme, and RAM). It is important to note here that these approaches have not implemented the actual code generation approach. Dynamic views are commonly supported by mapping approaches and not by code generation systems (T.SV). Hanenberg et al.’s work is the only one that supports both aspect and base concerns for code generation (T.SC). However, as we describe in the next section, it is limited to generation of code for Join Point Designation Diagrams (JPDDs) [22] only. JPDDs provide a graphical representation of the join point selections in a way that is independent of the programming language implementation. Interactions other than join point are not considered by any of the approaches (T.AN). Correctness and performance of the transformation algorithms have not been focused on (T.AM).

These results have some obvious implications on suitability of these approaches to use them as an instrument to manage the integration between AOSD and MDE. Support for dynamic views is vital in this regard. Moreover, in order to make these approaches reliable, further research is required on algorithmic maturity for the transformation process.

4.2. Models

Results of comparison on basis of the models criteria are shown in Table 2. Mapping of both structure and behavior models have been provided by approaches that consider only the mapping of design to code level constructs (first three in table) (M.SM & M.BM). So
far as the detailed implementation of the approach is concerned, only class diagrams are supported by FDAF.

4.3. Validation

As evident from the results in Table 2 under validation criteria, the validation approach used by almost all studied approaches is not generally transparent (V.AT). There is only one exception however, in which a defined validation mechanism was systematically applied to three open-source, publicly available systems.

When it comes to a reliable and practical integration of a code generation system into MDE process, one has to look beyond the validation mechanisms that only rely on some trivial case studies. In this way, it is important that a standard validation approach be applied that compares the generated code to that of an open source development project using well-defined comparison parameters.

4.4. Extent of code

Results of comparison on basis of extent of code generated shown in Table 3 reveal that only the code skeletons are supported currently and that full code has not been determined even when applying the proposed mapping techniques (E.SC). Mapping of behavior diagrams to code has been proposed (first three approaches) but not yet applied by code generation approaches (E.BC). Similarly, full code has been supported in terms of mapping only, and it has not been implemented in the code generation systems (E.FC).

4.5. Tool-support

Table 3 lists the results of applying tool-support criteria. The results are self-explanatory. Mostly, the tool support is not sufficient and not integrated with any of the standard development frameworks (S.TS & S.TI).

In order to support full code generation and thereby deliver a true integration of AOSD and MDE, incorporation of diagrams that model the system from dynamic perspective (e.g. sequence and state diagrams) is indispensable.

Table 2. Comparison on basis of Models & Validation Criteria

<table>
<thead>
<tr>
<th>Approach</th>
<th>Models criteria</th>
<th>Validation criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Class diagrams</td>
<td>Other diagrams</td>
</tr>
<tr>
<td></td>
<td>Structure Models (M SM)</td>
<td>Behavior Models (M BM)</td>
</tr>
<tr>
<td></td>
<td>Statechart</td>
<td>Sequence</td>
</tr>
<tr>
<td>MATA</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Theme</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RAM</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FDAF</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>JPDDs</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Table 3. Comparison on basis of Extent of Code and Tool Support criteria

<table>
<thead>
<tr>
<th>Approach</th>
<th>Extent of code criteria</th>
<th>Tool Support criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Structure Code (ESC)</td>
<td>Behavior Code (E.BC)</td>
</tr>
<tr>
<td></td>
<td>Full Code (E.FC)</td>
<td>Path for behavior diag.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Trade-off analysis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sufficient for validation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reliable for Real-world apps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Standard dev frameworks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Allow integration</td>
</tr>
<tr>
<td>MATA</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Theme</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RAM</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FDAF</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>JPDDs</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

In order to be used practically, a code generation approach has to provide an integrated system that works in conjunction with existing development IDEs and frameworks to deliver quality code. In this regard, we have carefully analyzed the tool-support provided by Bennett, et al., and found that it may be interesting and useful to enhance the existing tools to support dynamic views as well.

5 Discussion on the Integration

In the previous section, we have highlighted different features of the studied approaches to report their maturity or suitability if they were to be used in a setting that is intended to reach the long-term goal of obtaining complete and executable code from a design model. In this section, we briefly discuss each of the studied approach in terms of its likelihood to integrate into such a setting (i.e., the MDE process).

5.1. MATA: Mapping MATA Models to AspectWerkz Code [17]

Modeling Aspects Using a Transformation Approach (MATA) is essentially a graph-transformation based approach for composition of UML aspect models. This work has also elaborated the mapping of MATA models to aspect-oriented code. From the perspective of integration of AOSD and MDE, this approach is interesting and essentially different from all other aspect modeling approaches, making it a good choice for this purpose. Basically, it defines no constructs for defining aspects. Rather, both aspect and base models are standard UML, complemented with an additional specification for composition of both. This specification simply defines the relationship between base and aspect models. Therefore, when developing a code generation system, existing approaches for generation of base and/or aspect code can be exploited well and complemented with generation of the code for the additional specification defined using MATA.

5.2. Theme: Mapping Theme/UML Models to AspectJ [18]

This work has presented a detailed mapping from a well-known aspect modeling technique Theme/UML to AspectJ. Basically, in Theme approach, system concerns are represented in the form of themes, depicted using Theme/Doc at the analysis phase and
modeled using Theme/UML at the design phase. In this sense, the Theme approach covers both analysis and design phases of the system lifecycle, and thus can be considered as a robust candidate to be used for the integration addressed in this study. As far as the integration problem is concerned, the existing mapping may be enhanced to support recent developments in the AspectJ language (e.g., support for Annotations can be exploited to map templates). Obviously, this has to be followed by development of a code generation system. In this case, details of the code generation approach will need to be researched from scratch.

5.3. RAM: Mapping Resuable Aspect Models to AspectJ [19]

This work presents an elaborated mapping from aspect-oriented models developed using Reusable Aspect Models (RAM) approach. RAM is an interesting multi-view modeling approach that combines three different view types (i.e., structure, state and message views) to allow representation of sub-concerns with a modeling notations which is most appropriate to model it. The approach is different from other aspect-oriented modeling techniques in the sense that it emphasizes on reusability of aspects. Any concern in the system can be considered a reusable concern and modeled using the aspect notation. This makes it a powerful notation and thus a carrier of great potential for integration. As far as the mapping approach is concerned, at first glance, it seems comprehensive enough to generate a larger extent of code since it incorporates sequence diagrams. However, from perspective of its use for integration of AOSD and MDE, just like Theme/UML it also needs an ample amount of research to accomplish the fine-grained specifics of code generation.


If we consider the finer details of the implementation of a code generation system, this work can be considered the most mature approach presented so far. However, if used to the purpose of integration discussed in this paper, this approach has two drawbacks. First, it is based on Formal Design Analysis Framework (FAF) [23] which works mostly at architectural design level only. Second, currently the amount of generated code is rather insignificant. However, as described in previous section, this work has its own plus points. The transformation algorithms are well-defined and detailed for the model-to-code transformation of structure diagrams. Further research can extend them to support behavior diagrams. The most significant contribution of this work, in our opinion, is its graph-based transformation approach. It can be extended to work as a uniform approach to handling transformation of all types of diagrams. Similarly, the elementary work on validation and tool-support can be extended to support integration of the approach into a true MDE environment.

5.5. JPDDs: Translation of Join Point Designation Diagrams to AspectJ [21]

Join Point Designation Diagrams (JPDDs) basically intend to provide a graphical notation for developers to communicate the details of their joint point selections. Their purpose is not to provide a general-purpose and complete aspect-oriented design notation. However, in the context of integration of AOSD and MDE, this work may provide some insight through the set of principles devised to consider during the mapping of JPDDs to a programming language.
6. Conclusions and Future Work

The research on integration of aspect-oriented software development and model-driven engineering process can result in enabling software development to produce more maintainable, extensible and reusable systems. In this paper, we have evaluated existing aspect-oriented code generation approaches which can be used for this integration.

In order to achieve a practical and useful integration of AOSD and MDE using aspect-oriented code generation, this study investigated two (non-exclusive) relevant perspectives. First, it examined current approaches to aspect-oriented code generation to find whether they seem mature and reliable enough to be able to generate fully executable code. Second, the study focused on the integration view to find what enhancements are essential if some of the current approaches were to be integrated into MDE process.

The study determined that current approaches to aspect-oriented code generation still lack several mandatory features. For examples, code generation for behavioral models is currently unavailable, correctness of transformation process has not been established using some standard validation mechanism, and no sufficient integrated tool-support has been provided.

So far as the integration of approaches into MDE process is concerned, all approaches possess some good prospects, but require a reasonable amount of effort and research. Specifically, approaches that provide basic, tool-supported code generation need enhancements in terms of support for advanced feature set. On the other hand, approaches that cover advanced features in their model-to-code mappings need detailed elaboration and development of functional code generation systems.

This study suggests several lines of future work. An important issue is the transformation and code generation thereby of behavior diagrams. To make this transformation reliable, some existing model-to-code techniques need to be enhanced for this purpose, elaboration of which is another significant problem. We intend to work further on mapping techniques discussed in this paper to enhance and integrate them into the MDE process.

References
