ASPECT-ORIENTED MODEL-DRIVEN CODE GENERATION APPROACH FOR IMPROVING CODE REUSABILITY AND MAINTAINABILITY

ABID MEHMOOD

A thesis submitted in fulfilment of the
requirements for the award of the degree of
Doctor of Philosophy (Computer Science)

Faculty of Computing
Universiti Teknologi Malaysia

MAY 2014

To my beloved parents, wife Irum, and our children Shuja and Rameen

ACKNOWLEDGEMENT

First, I am grateful to Almighty Allah (swt) for always being the source of strength, well-being, and gratification for me throughout my life and particularly through the period of this research work.

I am thankful to Dr. Dayang Jawawi, my supervisor, for her vibrant and invaluable guidance. Dr. Jawawi has continuously supported me during this research, reviewing my work, providing clear advice, and entertaining numerous questions with kind gestures.

I am highly obliged to my parents, brothers and sisters for their encouragement, endless support and compassionate prayers. Their prayers have always been a source of inspiration and encouragement for me. Here, I also acknowledge the support of my dear friend and brother, Shabir Ahmed, who has helped me along in ways he may not even know.

I owe a great debt of gratitude to my wife, Irum, for her love, huge patience, support and motivation. This work would not have been accomplished without her help and support indeed. Besides her, I also express much appreciation to our children Shuja and Rameen, for helping me get through the hard times, and understanding at the moments I was not with them.

ABSTRACT

Software development teams always need methods that can help in producing high-quality software with reduced development effort and delivery time. Model-Driven Engineering (MDE) as well as Aspect-Oriented Software Development (AOSD) techniques help in reducing the delivery time, and also positively contribute to quality of the produced software. Through the use of AOSD techniques in combination with MDE, an integration of excellent abstraction mechanisms of MDE and capabilities of AOSD with regards to modularity and composition of concerns can be perceived, which is expected to enhance the positive effects of both techniques. To this end, different integration approaches have appeared in literature, but aspect-oriented code generation has advantages over the other approaches. Consequently, a number of aspect-oriented code generation approaches have been offered, but all such approaches lack several features mandatory to materialize a workable integration of aspect technologies in the context of MDE. To address these issues, this research was conducted to present an approach for aspect-oriented model-driven code generation, which focuses on elaborating the conceptual relationship between design models and the implementation code, and exploits the same to obtain aspect-oriented code that is more reusable and maintainable. The key outcomes of this research are the elaboration of the conceptual mappings between elements of visual design and constructs of the code, mapping of the visual models to implementation-specific text-based models, and a technique for generation of aspect-oriented code. The applicability of the proposed approach is shown by the use of case studies, whereas the quality of the approach is measured using reusability and maintainability metrics. A comparison of the proposed approach with existing approaches substantiates its efficacy in terms of reusability and maintainability of code, showing an outperformance of other approaches by the proposed approach against 78% of the employed quality metrics.

ABSTRAK

Pasukan pembangunan perisian sentiasa memerlukan kaedah yang boleh membantu dalam menghasilkan perisian yang berkualiti tinggi dengan mengurangkan usaha pembangunan dan masa penghantaran. Kejuruteraan berpandukan model (MDE) dan teknik Pembangunan Perisian Berorientasikan Aspek (AOSD) membantu dalam mengurangkan masa penghantaran dan juga menyumbangkan kepada kualiti perisian yang dihasilkan. Dengan penggunaan teknik-teknik AOSD yang digabungkan dengan MDE, gabungan ini dijangka boleh memperbaiki kesan positif kedua-dua teknik dengan integrasi mekanisma pengabstrakan MDE yang baik dan kemampuan AOSD dari segi kemodularan dan komposisi yang boleh diamati. Untuk ini, pendekatan-pendekatan integrasi yang berbeza telah muncul di dalam literatur, tetapi penjanaan kod berorientasikan aspek mempunyai banyak kelebihan berbanding pendekatan yang lain. Walaupun, pelbagai pendekatan penjanaan kod berorientasikan aspek telah ditawarkan, tetapi kesemua pendekatan tersebut kurang dari segi ciri-ciri mandatori untuk membolehkan integrasi teknologi aspek di dalam konteks MDE dilaksanakan. Bagi menyelesaikan kekurangan ini, kajian ini telah dijalankan untuk menunjukkan satu pendekatan penjanaan kod berorientasikan aspek dan berpandukan model, di mana ia fokus kepada penghuraian hubungan konseptual di antara reka bentuk model dan pelaksanaan kod, dan mengolah kedua-duanya untuk mencapai kod berorientasikan aspek yang mempunyai kadar kebolehgunaan semula dan kebolehselenggaraan yang lebih. Penghasilan utama kajian ini adalah penghuraian pemetaan elemen-elemen reka bentuk visual dan pembinaan kod, pemetaan model visual untuk model berasaskan teks dan perlaksanaan-spesifik dan teknik untuk penjanaan kod berorientasikan aspek. Kebolehgunaan pendekatan yang dicadangkan ini telah ditunjukkan melalui kajian kes, manakala kualiti pendekatan diukur menggunakan metrik kebolehgunaan semula dan kebolehsenggaraan. Perbandingan di antara pendekatan yang dicadangkan dengan pendekatan-pendekatan sedia ada berkaitan keberkesanannya dari segi kebolehgunaan semula dan kebolehsenggaraan kod, menunjukkan pendekatan yang dicadangkan mengatasi pendekatan-pendekatan lain sebanyak 78% dengan menggunakan metrik pengukuran kualiti.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AO	-	Aspect-Oriented
AOM	-	Aspect-Oriented Modeling
AOMDCG	-	Aspect-Oriented Model-Driven Code Generation
AOSD	-	Aspect-Oriented Software Development
CASE	-	Computer-Aided Software Engineering
CRM	-	Conceptual Reference Framework
FDAF	-	Formal Design Analysis Framework
GTW	-	Generate-Then-Weave
JPDD	-	Join Point Designation Diagram
MATA	-	Modeling Aspects using a Transformation Approach
MDA	-	Model-Driven Archiecture
MDE	-	Model-Driven Engineering
MOF	-	Meta-Object Facility
NFR	-	Non-Functional Requirement
OCL	-	Object Constraint Language
OO	-	Object-Oriented
OBSS	-	Online Book Store System
PIM	-	Platform-Independent Model
PSM	-	Platform-Specific Model
RAM	-	Reusable Aspect Models
RSC	-	Remote Service Caller
SDL	-	Specification and Description Language
SIG	-	Sequence Integration Graph
SOAP	-	Simple Object Access Protocol
UML	-	Unified Modeling Language
UMLAUT	-	Unified Modeling Language All pUrposes Transformer
WTG	-	Weave-Then-Generate

XMI	-	XML Metadata Interchange
XML	-	Extensible Markup Language
XSLT	-	EXtensible Stylesheet Language

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Overview

Model-Driven Engineering (MDE) is an approach to software development that stresses upon making models the primary development artifact, and using these models as source in a process leading to automatic generation of the final application code. It emphasizes on subjecting models to a refinement process, through automatic transformations, until a running system is obtained. By doing so, MDE aims at providing higher level of abstraction in development of systems which further results in an improved understanding of complex systems. Moreover, it addresses problems in software systems development that originate from existence of heterogeneous platforms. It achieves this through keeping different levels of model abstractions; and by transforming models from Platform Independent Models (PIMs) to Platform Specific Models (PSMs). In this context, automatic generation of application code (i.e., automatic model-driven code generation) 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. In addition to these, several other benefits have also been reported in (Afonso *et al.*, 2006; Karakostas and Zorgios, 2008).

Aspect orientation and the related paradigm, i.e., Aspect-Oriented Software Development (AOSD) (Elrad *et al.*, 2002; Rashid *et al.*, 2006; Sánchez *et al.*, 2010;

Hoffman and Eugster, 2013) provide an approach to software engineering which allows explicit identification, separation, and encapsulation of concerns that cut across the primary modularization of a system. These c*rosscutting concerns* cannot be clearly decomposed from primary functionality (*core concerns*) of the system, and thus cannot be effectively modularized, when using well-known object-oriented development techniques. Hence these concerns end up *scattered* throughout the system and *tangled* with the core concerns of system. Even though the crosscutting concerns usually originate from non-functional requirements such as logging, security, persistence, and optimization, but the phenomenon encompasses the functional ones too, which often have their behavioral logic spread out over several modules. Using aspect-orientation, these concerns are identified, modeled and implemented independent of each other as well as separate from the main functional concerns of the system. Once separated in this way into modules, these concerns need some composition mechanism to control where and when concern behavior is applied. This effectiveness of modularization is achieved through applying aspect orientation at analysis phase using Early Aspects (Rashid *et al.*, 2006), during design using Aspect-oriented Modeling (Elrad *et al.*, 2002), and using Aspect-oriented Programming (Kiczales *et al.*, 1997) for implementation. The separation of crosscutting concerns from core functionality of the system achieved through aspect-orientation eventually results in improving the reusability and maintainability qualities of software, which in turn contribute positively to several quality factors such as understandability, flexibility and extensibility (Hannemann and Kiczales, 2002; Garcia *et al.*, 2005; Hovsepyan *et al.*, 2010; Piveta *et al.*, 2012).

In order to combine the use of aspect orientation and MDE (Amaya *et al.*, 2005; Clemente *et al.*, 2011; Pinto *et al.*, 2012), aspect-oriented models developed using Aspect-Oriented Modeling (AOM) approaches (cf. (beeck *et al.*, 2006; Wimmer *et al.*, 2011)) can be integrated and used within the context of MDE in at least two different ways. This eventually results in forming two distinct lines of research for the integration of aspect orientation and MDE. Along the first line are approaches that propose using a model weaver to compose the base model (one that models core concerns) and the aspect model (model which represents crosscutting concerns) in such a way that a non-aspect-oriented (object-oriented) model is obtained. Then, standard code generation approaches can be used to generate code into one of the object-oriented programming languages. In contrast, the second line of research comprises of approaches that explore direct transformation of the source aspect-oriented model into code of a target aspect-oriented language, and then rely on weaver provided by the target language to deal with crosscutting aspects.

While transforming aspect models into object-oriented code may provide some benefits such as using existing code generators, the technique is bound to lose the aspectual features owing to lack of their support by object-oriented programming languages. This means that the concerns that have been composed once at the model level can no longer be reproduced in a separated form. As aspect orientation is used in software development projects with the clear intention of separation of concerns (and their reuse as needed), it is unreasonable to lose this separation during model to code transformation. This is the main idea that inspires the current research to studying the direct transformation of aspect-oriented models into aspect-oriented code.

1.2 Research background

When it comes to the use of aspect-oriented technologies in the context of MDE (Amaya *et al.*, 2005; Clemente *et al.*, 2011; Pinto *et al.*, 2012), early literature has paid much attention to aspect-oriented modeling, whereas the issue of their transformation into code, i.e., model-driven code generation, has been rarely investigated. However, in order to discuss the existing research in the broader context of the integration of AOSD and MDE, it may be classified into two major groups: AOM notations along with model weavers, and mechanisms to transform AO models into AO code.

The core idea behind the use of model weavers with AO models is to provide composition mechanism so that models can be simulated, tested and debugged prior to execution. The Motorola experience (Baker *et al.*, 2005) and the Motorola WEAVR report (Cottenier *et al.*, 2007b) have presented many benefits associated with simulation, testing and debugging of models. In this context, significant work has been accomplished in the domain of aspect-oriented modeling and composition approaches. Majority of aspect-oriented modeling approaches extend UML by modeling either structure or behavior, or both, using UML diagrams extended with aspect-oriented concepts. However, there is comparatively little number of proposals that address extending behavior diagrams. As far as the composition approaches for aspect-oriented modeling are concerned, there are two broader categories: a*symmetric* and *symmetric* composition approaches. An asymmetric composition refers to an approach which models aspects separate from other components in the system. Aspects are used to model crosscutting concerns, whereas components are used to model non-crosscutting concerns (the base). In order to obtain a composed view of the system, aspects are applied to (i.e., woven into) the base. On the other hand, symmetric composition approach does not make any distinction between aspects and the base. In this approach, a system is simply regarded as a set of concerns that are to be composed. Therefore, these approaches separate all concerns that exist in a system, rather than separating crosscutting concerns from non-crosscutting ones.

The initial work on aspect-oriented modeling was presented in (Grundy, 2000). This initial proposal performed structure modeling using extended class and component diagrams, whereas only the communication diagram was extended for behavior modeling. It provided asymmetric composition mechanism. Some other works that have provided asymmetric composition include Aspect-Oriented Design Modeling (Stein *et al.*, 2002b; Stein *et al.*, 2002a), Aspect-Oriented Architecture Models (France *et al.*, 2004), UML for Aspect-Oriented Software Development (Pawlak *et al.*, 2002), Dynamic Component and Aspect-oriented Platform (Pinto *et al.*, 2005), Aspect-oriented Executable Modeling (Fuentes and Sanchez, 2007a), and Motorola WEAVR (Cottenier *et al.*, 2007b). For structure, all these approaches primarily rely on extension of class diagram. In addition to class diagrams, only one approach (Aspect-oriented Architecture Models) supports the package diagram, and one (Dynamic Component and Aspect-

Oriented Platform) supports component diagrams. Motorola WEAVR supports composite structure diagram and deployment diagram.

Theme/UML (Clarke and Walker, 2001; Clarke, 2002; Clarke and Walker, 2002; Baniassad and Clarke, 2004) was the first approach to aspect-oriented modeling which used symmetric composition. It provides support for extended class and package diagrams for structure modeling, and sequence diagrams for behavior modeling. It can be regarded as a leading aspect-oriented approach among symmetric ones, since it provides complete mapping of the design to implementation (Clarke and Baniassad, 2005), and initial results on its integration with model-driven engineering have been presented in literature (Carton *et al.*, 2009). Other significant work that provides symmetric composition includes the State Charts UML Profile (Aldawud *et al.*, 2002; Aldawud, 2003), Architectural Views of Aspects (Katara and Katz, 2003), and Aspect Modeling Language (Groher and Baumgarth, 2004). All these aspect-oriented extensions use the class diagrams for structure modeling, while latter two extend the package diagram as well. The former two approaches contribute mainly to behavior modeling in a sense that they have provided support for state chart diagram and sequence diagram. A combination of state charts and sequence diagrams in this way can be extended to represent complete behavior of a system in aspect-oriented way. Another recent and a prominent approach to aspect-oriented modeling is Reusable Aspect Models (RAM) (Klein and Kienzle, 2007; Abed and Kienzle, 2009; Kienzle *et al.*, 2009; Kienzle, 2013). RAM is a multi-view aspect modeling approach which is unique in the sense that it provides one coherent approach to aspect-oriented modeling by integrating existing class diagram, state chart diagram, and sequence diagram approaches. Moreover, reuse of aspects is the core idea of RAM.

Regardless of the specific features supported by each of the AOM notations discussed above, the respective weaving mechanisms provided by all these approaches result in a woven (object-oriented) model. Therefore, these approaches may be extended to work in integration with existing object-oriented code generation techniques (e.g., (Chauvel and Jézéquel, 2005; Niaz and Tanaka, 2005; Pilitowski and Dereziñska, 2007;

Stavrou and Papadopoulos, 2009; Badreddin *et al.*, 2014)) in order to obtain code from the woven model. Nevertheless, even though the modeling approaches may work effectively in combination with their associated weaving mechanisms (integrated with OO code generators) for the purpose of model analysis and execution, they are not expected to provide effective support with regards to the development of software. This is for the reason that: (1) the generated code is object-oriented, and hence, lacks the support for aspectual features envisioned at the model level and (2) the code is usually generated with the intention of model analysis and execution only, and therefore, results in maintenance and other related problems if it was to be maintained manually (Simmonds and Reddy, 2009; Hovsepyan *et al.*, 2010; Papotti *et al.*, 2013). It has to be emphasized here that one would expect the need for manual code maintenance and evolution until MDE becomes an extremely mature discipline. These problems arise owing to the ineffective handling of crosscutting concerns by these approaches. As they do not retain the separation of concerns after a model has been woven (and further transformed into code), they result in implementation code that is difficult to reuse and maintain.

The approaches that propose transformation of an aspect-oriented model directly into aspect-oriented code essentially do not suffer from problems mentioned above, as they tend to maintain the separation of concerns from model to code. Moreover, these approaches are mainly inspired by benefits resulting from existence of a direct mapping between constructs of design model and the programming language. In this regard, a number of empirical studies, for example (Hannemann and Kiczales, 2002; Garcia *et al.*, 2005; Cacho *et al.*, 2006; Fuentes and Sánchez, 2007; Greenwood *et al.*, 2007) have reported potential benefits of using aspect-oriented techniques in software development. Another study (Hovsepyan *et al.*, 2010) has discovered that approaches pursuing aspect-oriented programming languages result in compact, smaller, less complex and more modular implementations. Beside academia, the use of aspect technologies at the implementation level is now well-established in industrial circles, as the designers of mainstream implementation frameworks (e.g., JBoss, Spring) are increasingly adopting it.

The initial work on mapping of aspect-oriented design to aspect-oriented programming language was initially presented in 2002 by mapping Theme/UML models to AspectJ (Kiczales *et al.*, 2001) code (Clarke and Walker, 2002). Besides providing traceability between constructs of both languages, this work majorly contributed to providing means for assessing the two languages and their incompatibilities. However, some decisions in the mapping process resulted in code that imposed stronger restrictions on reusing the modules. A similar but enhanced mapping has been provided for Reusable Aspect Models (RAM) recently (Kramer, 2010; Kramer and Kienzle, 2011). This mapping has provided several improvements to approach of Theme/UML mapping by using AspectJ's recent support for *annotations*. Enhanced flexibility and better support for reuse are major distinctive features of RAM's mapping approach as compared to Theme/UML. Mapping of Theme/UML to CaesarJ code has also been explored by Jackson *et al.* (2008). The mapping of Theme/UML to CaesarJ, is very similar to the mapping done for AspectJ, but this work does not address various problems that arise from specific properties of CaesarJ in the context of mapping from model to code. Recently, in (Loukil *et al.*, 2013), architectural aspects described in AO4AADL have been mapped to AspectJ aspects with the help of transformation rules based on Real-Time Separation for Java (RTSJ) (Autret, 2009) rules. However, all these works mainly focus on highlighting the conceptual relationship between constructs of design and code, and make use of sequence diagrams for behavior implementation. Therefore, details related to implementation of the code generation such as an implementation savvy representation of visual models, or details of code generation process have not been addressed by these approaches. Moreover, the use of only sequence diagrams limits the extent of code to be generated. This is for the reason that sequence diagrams are considered suitable for modeling behavior of controller objects that involve sequence of method calls. They cannot effectively model the detailed object behavior.

Some other work focuses on generating the aspect-oriented code from extended UML models and addresses some other concerns as well. All work that follows in this category, however, lacks in at least two aspects: (1) they have not discussed the detailed mapping as to how the artifacts at model level are translated to code level constructs, and (2) they have not addressed behavior diagrams at all. In this regard, the work of Bennett *et al.* (2010) has provided an approach that uses graph-based transformation algorithms to transform aspect design into AspectJ code. For implementation, they have used XML schemas to textually represent their architectural design models. However, they address only the class diagrams for this purpose. Groher and Schulze (2003) propose an experimental design notation based on the standard UML. This notation enhances reuse of aspect code by clearly separating the reusable programming-language-independent design from language-dependent crosscutting one. In (Evermann, 2007; Evermann *et al.*, 2011), a template-based approach to generating AspectJ code has been proposed. This is different from previous work in the sense that it generates code from a comprehensive specification of the AspectJ meta-model. It defines a meta-model for the AspectJ language in the form of a UML profile which utilizes the built-in features of UML by using stereotypes and tags to specify the meta-model of AspectJ. By implementing specific constructs of AspectJ instead of defining new element types into UML meta-model, this AspectJ meta-model specification becomes compatible for use with any of the existing CASE tools that support XMI interface. Hecht *et al.* (2005) have proposed an approach to generating code from Theme/UML models in AspectJ. They develop XML representation of Theme/UML models and use the Theme approach to mapping from model to code. Furthermore, these approaches do not provide sufficient details with regards to the comparison of the generated code with that obtained using other mechanisms such as object-oriented implementation of the same concept.

To conclude, aspect-oriented modeling and model-driven code generation are extensive study fields and much effort has been put to improving the mechanisms of exploiting aspect-orientation in the context of MDE. Despite several existing proposals, the efforts need furtherance with respect to the specific case in which the integration is carried out using aspect-oriented code generation.

1.3 Motivation

Software development projects are aimed at producing high-quality software within allocated time. However, as the projects grow in size and complexity, achieving the goals of quality and on-time delivery become more challenging. For this reason, software projects often end up running over schedule, as found in software project management studies such as (Brooks, 1995). Moreover, these off-schedule projects often relinquish quality in order to meet the project deadlines, further leading to software products which are less reliable, less maintainable, and less adaptable. Therefore, to prevent them ending up running over schedule, or even worse, relinquishing quality in order to meet the deadlines, software teams are always in need of techniques that can help reducing delivery time, and also lend to raising the quality of the product.

MDE techniques not only help in reducing the delivery time but also positively contribute to overall quality of the produced software (Fleurey *et al.*, 2007; Aranda *et al.*, 2012). In this context, at the design level, visual modeling languages such as (Rumbaugh *et al.*, 1991; Jacobson, 1992; Booch, 1993; Group, 2007) support by providing modeling and model-checking capabilities (Fuentes and Sánchez, 2009). On the other hand, during the implementation and maintenance phases, the same effect is achieved by applying automatic code generation (Sánchez *et al.*, 2010; Rahmouni and Mbarki, 2013). Automatically generated code, if correctly obtained, enhances the benefits of high-level modeling and analysis (Carton *et al.*, 2009). Hence, in the past, it has been deemed ideal to develop approaches that generate or help to generate executable code from high level design models. So far as the benefits of automatic model-driven code generation are concerned, the most significant advantages include reduction in development time, and improvement in quality (Hovsepyan *et al.*, 2006; Alonso *et al.*, 2007; Papotti *et al.*, 2013). The majority of automatic code generation approaches have addressed automatic code generation for object-oriented analysis and design models. Moreover, code generation has been presented using formal notations. Examples of code generation using formal notations include Petri Nets (Philippi, 2006), Software Cost Reduction (SCR) (Rauchwerger *et al.*, 2005), and Cinderella SLIPPER

(Rauchwerger *et al.*, 2005). These approaches have achieved complete code generation and they have proposed techniques for optimized code generation. In some other works such as (Chauvel and Jézéquel, 2005; Niaz and Tanaka, 2005; Pilitowski and Dereziñska, 2007; Stavrou and Papadopoulos, 2009), models represented in UML have been used to generate fully executable object-oriented code. UML models have also been used to generate code for web applications (Rahmouni and Mbarki, 2013). Advanced issues such as handling of class associations with the help of model-oriented languages have been explored in the perspective of code generation (Badreddin *et al.*, 2014). Further, many of currently available commercial (e.g., IBM Rational Software Architect (Leroux *et al.*, 2006), AjileJ StructureViews (AjileJ, 2011), MagicDraw UML (NoMagic, 2011)) as well as open source (e.g., ArgoUML (Tigris.org, 2012), Eclipse UML2 Tools (Eclipse.org, 2012)) object-oriented CASE tools support the generation of code stubs. However, all this work has been carried out in object-oriented paradigm, and thus eventually results in scattered and tangled code.

Aspect oriented techniques too, just like the MDE, aim to positively affect the delivery time and quality of the software products. Specifically, they achieve this goal by providing better modularization of components leading to improving their reuse and enhancing other quality factors such as maintainability (Burrows *et al.*, 2010b; Giunta *et al.*, 2012). Thus, MDE and aspect orientation possess some complementary properties (Pinto *et al.*, 2012). While modeling increases the level of abstraction, it suffers from difficulties when it comes to the refinement and integration of system perspectives. Aspect orientation, on the other hand, allows better modularization and composition of concerns, but lacks appropriate abstraction mechanisms (Cottenier *et al.*, 2007b). Therefore, an integration of these two technologies can increase the benefits of both (Pinto *et al.*, 2012). This is because, on one hand, excellent abstraction mechanisms of MDE will become available to aspect-oriented techniques and on the other hand, MDE will be augmented by the capabilities of aspect-oriented techniques with regards to modularity and composition of concerns. Such an integration can be realized by subjecting aspect-oriented models to a transformation process that leads to generation of application code into a target AO programming language. Therefore, owing to the benefits associated with direct transformation of aspect-oriented models into aspect-oriented code (Hannemann and Kiczales, 2002; Garcia *et al.*, 2005; Cacho *et al.*, 2006; Fuentes and Sánchez, 2007; Greenwood *et al.*, 2007), a few efforts have been made to achieve the same and initial results have been reported in the literature, for example, (Whittle *et al.*, 2009; Bennett *et al.*, 2010; Evermann *et al.*, 2011; Kramer and Kienzle, 2011; Loukil *et al.*, 2013). All such approaches are naturally based on model-driven architecture, meaning that they use a source model developed in some notation extended from UML as input and generate the target aspect-oriented code according to some transformation definition. However, each of these approaches generates code in a way that it supports certain specific features of aspect-oriented code generation (e.g., limited structure), while eliminating others (e.g., behavioral part, reuse of functionality at code level, implementation details etc.). Therefore, the use of aspect orientation in integration with MDE cannot be exploited to an extent that it actually results in aspect-oriented code, which is more maintainable and reusable for developers, and possesses both the structure as well as behavior. Consequently, there is a need for an approach that bases upon a mature AOM design notation and eventually produces executable aspect code, to realize a practicable integration of aspect orientation in the larger context of MDE.

1.4 Problem statement

The software development process can be significantly improved, and the effort involved in writing and maintaining software can be reduced by employing aspect orientation in integration with MDE techniques. The past research has mostly focused on applying aspect orientation in the context of MDE by providing only the model composition facilities for AO models, which results in a common object-oriented model and leads to object-oriented code. However, this approach of integrating aspect orientation and MDE does not effectively solve the problem of supporting software development teams in reducing their effort for writing and maintaining code. The main reason for the ineffectiveness of this approach is that it results in the loss of clear relationship between the elements of AO design and the generated code (consequently making the reuse and maintenance of code more difficult) on one hand, and elimination of numerous benefits of applying aspect-orientation on the other hand. As far as the possibility of generating AO code is concerned, it has not been given much attention. Therefore, the approaches that consider aspect-oriented model-driven code generation either do not elaborate the relationship between constructs of model and code, or do not effectively address the issues involved in actual generation of code such as elaboration of implementation models and code generation process. Furthermore, all current approaches suffer from at least one common problem, which is their inability to generate comprehensive object behavior. Hence a practicable integration of aspect orientation and MDE cannot be provided.

In this study, we intend to propose an approach for the integration of aspect orientation and MDE that solves the problems mentioned above. The main problem to be addressed can be specified as:

"How can we apply aspect orientation in combination with MDE, by generating aspect-oriented code that is more reusable and maintainable than its object-oriented counterpart, for both structure and behavior specified in the input aspect-oriented models?"

In order to address the main research problem given above, we need to provide answers to the following research questions as a pre-requisite.

1) What is an effective AOM approach that possesses the ability to model a system in a comprehensive manner, makes the reuse of modeled functionality straightforward, and lends itself to code generation?
2) What are the elements of an aspect-oriented design that are vital to generation of behavioral code, in order to maximize the amount of generated code, and thus produce a workable combination of aspect orientation with MDE?

3) How can the coherence between the aspect-oriented design and the target aspect-oriented programming language be considered, and how can the elements of the former be mapped to the constructs of the latter?
4) How can the common problems in aspect code generation benefit from existing solutions in other disciplines like object-oriented?
5) What are different options to transforming the visual model into a computer-understandable text-based model?
6) How can the aspect-oriented code be actually obtained?
7) What is the effect of proposed integration approach on the quality of final code in terms of its reusability and maintainability?

1.5 Research objectives

The final goal of this research is to propose an approach for the integration of aspect orientation and MDE through aspect-oriented code generation. The approach, to be referred to as the aspect-oriented model-driven code generation approach throughout this thesis, has to take a source AO model and generate AO code that contains implementation of both structure as well as behavior represented in the design. Moreover, the approach has to support the code generation for aspectual as well as non-aspectual parts of the model. The specific objectives aimed at achieving this goal are:

1) To elaborate a method for mapping AO models representing the system structure and object behavior to AO code.
2) To define a text-based implementation model that transforms the visual AO design model into a formal and equivalent textual representation, and supports its systematic translation into code.
3) To develop an aspect-oriented code generation technique that applies the mapping method to the textual representation of design models, and generates structural as well as behavioral code for both aspectual and non-aspectual parts.

4) To evaluate the proposed approach of integration against the other approach in terms of reusability and maintainability of the generated code.

1.6 Research scope

The scope of this research study is limited to:

1) This research is related to supporting the software application development through AO design and code generation. Therefore, other issues that may have impact on the development such as requirements engineering are not dealt with in this study.
2) For the purpose of AO design, this research is intended to determine an effective AOM notation from the whole corpus of existing ones. Thus, proposing a new design notation is not part of the scope of this research.
3) This research focuses on generating aspect-oriented code for Java/ AspectJ languages only.
4) The proposed integration approach has been validated for small to medium-sized general-purpose software applications.
5) Metrics-based measurement of performance of the integration approach against other approach has been conducted by applying *reusability* and *maintainability* metrics only.

1.7 Thesis outline

This thesis is organized as follows: Chapter 2 provides the basic background for this study and sets the research in context. Specifically, it reviews different approaches for the use of aspect technologies in MDE context, and determines the prerequisites for the current research. This chapter also evaluates the corpus of existing AOM notations in the context of their integration in MDE process through code generation and makes selection of the approach to be employed in this work. Research methodology employed to conducting this research is presented in Chapter 3. The conceptual mapping method for implementation of various elements of design model at the code level is discussed in Chapter 4. A prerequisite to the systematic code generation, the text-based representation of the aspect models is presented in Chapter 5. Chapter 6 provides the details of the code generation technique, by introducing the code generation algorithm and describing application of the same on the textual representation of aspect models. The applicability of the proposed integration approach is demonstrated with the help of two case studies in Chapter 7 and Chapter 8. Chapter 9 discusses the results by explaining the measurement process used to validate the results and by comparing the proposed integration approach with the other existing approach. Chapter 10 concludes the thesis while highlighting the findings, resolved issues and future work.

REFERENCES

Abed, W. A. and Kienzle, J. (2009). Information Hiding and Aspect-Oriented Modeling. *Proceedings of the 14th Aspect-Oriented Modeling Workshop*, Denver, CO, USA, 1–6.

Afonso, M., Vogel, R. and Teixeira, J. (2006). From code centric to model centric software engineering: practical case study of MDD infusion in a systems integration company. *Model-Based Development of Computer-Based Systems and Model-Based Methodologies for Pervasive and Embedded Software*, *2006. MBD/MOMPES 2006. Fourth and Third International Workshop on*, 10 pp.-134.

AjileJ (2011). AjileJ StructureViews www.ajilej.com. Accessed January 2013.

Al Abed, W., Bonnet, V., Schöttle, M., Yildirim, E., Alam, O. and Kienzle, J. (2013). *TouchRAM: A Multitouch-Enabled Tool for Aspect-Oriented Software Design*. In Czarnecki, K. and Hedin, G. (Ed.) *Software Language Engineering*. (275-285). Springer Berlin Heidelberg.

Al Abed, W. and Kienzle, J. (2011). *Aspect-Oriented Modelling for Distributed Systems*. In Whittle, J., Clark, T. and Kühne, T. (Ed.) *Model Driven Engineering Languages and Systems*. (123-137). Springer Berlin / Heidelberg.

Aldawud, O., Bader, A. and Elrad, T. (2002). Weaving With Statecharts. *Proceedings of the Aspect-Oriented Modeling with UML workshop (at AOSD)*, The Netherlands, 41-47.

Aldawud, T., Bader, A.,Tzilla Elrad (2003). UML profile for aspect-oriented software development. *The Third International Workshop on Aspect Oriented Modeling.*

Alhalabi, F., Vienne, P., Maranzana, M. and Sourrouille, J. L. (2006). Code Generation from the Description of QoS-Aware Applications. *Information and Communication Technologies, 2006. ICTTA '06. 2nd*, 3216-3221.

Ali, J. (2010a). Implementing statecharts using Java enums. *2010 2nd International Conference on Education Technology and Computer (ICETC)*, 413-417.

Ali, J. (2010b). Using Java Enums to implement concurrent–hierarchical state machines. *Journal of Software Engineering* 4(3), 215–230.

Ali, J. and Tanaka, J. (1998). An Object Oriented Approach to Generate Executable Code from OMT-Based Dynamic Model. *Journal of Integrated Design and Process Science* 2(4), 65-77.

Ali, J. and Tanaka, J. (2000). Converting Statecharts into Java Code. *Fourth World Conf. on Integrated Design and Process Technology (IDPT'99)*, Dallas, Texas, USA,

Ali, J. and Tanaka, J. (2001). Implementing the dynamic behavior represented as multiple state diagrams and activity diagrams. *ACIS Int. J Comp. Inf. Sci.* 2(1), 24-36.

Alonso, D., Vicente-Chicote, C., Sánchez, P., Álvarez, B. and Losilla, F. (2007). Automatic Ada code generation using a model-driven engineering approach. *Proceedings of the 12th international conference on Reliable software technologies*, Geneva, Switzerland, 168-179.

Amálio, N., Kelsen, P., Ma, Q. and Glodt, C. (2010). *Using VCL as an Aspect-Oriented Approach to Requirements Modelling*. In Katz, S., Mezini, M. and Kienzle, J. (Ed.) *Transactions on Aspect-Oriented Software Development VII*. (151-199). Springer Berlin / Heidelberg.

Amaya, P., González, C. and Murillo, J. (2005). MDA and Separation of Aspects - An Approach based on Multiple Views and Subject Oriented Design. *Proceedings of the 6th International Workshop on Aspect-Oriented Modeling held in conjunction with the 4th International Conference on Aspect-Oriented Software Development (AOSD'05)*, Chicago, Illinois, USA,

Aranda, J., Damian, D. and Borici, A. (2012). Transition to model-driven engineering: what is revolutionary, what remains the same? *Proceedings of the 15th international conference on Model Driven Engineering Languages and Systems*, 692-708. Springer-Verlag.

Araújo, J. and Whittle, J. (2013). *Aspect-Oriented Compositions for Dynamic Behavior Models*. In Moreira, A., Chitchyan, R., Araújo, J. and Rashid, A. (Ed.) *Aspect-Oriented Requirements Engineering*. (45-60). Springer Berlin Heidelberg.

Autret, T. (2009). *Code Generation of Real-Time Java for Real-time Systems*. Masters Thesis, Pierre & Marie Curie University, Paris.

Badreddin, O., Forward, A. and Lethbridge, T. C. (2014). Improving code generation for associations: Enforcing multiplicity constraints and ensuring referential integrity. 496, 129-149.

Baker, P., Loh, S. and Weil, F. (2005). *Model-Driven Engineering in a Large Industrial Context — Motorola Case Study*. In Briand, L. and Williams, C. (Ed.) *Model Driven Engineering Languages and Systems*. (476-491). Springer Berlin / Heidelberg.

Ballal, R. and Hoffman, M. A. (2009). Extending UML for Aspect Oriented Software Modeling. *Computer Science and Information Engineering, 2009 WRI World Congress on*, 488-492.

Baniassad, E. and Clarke, S. (2004). Theme: an approach for aspect-oriented analysis and design. *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*, 158-167.

Barais, O., Klein, J., Baudry, B., Jackson, A. and Clarke, S. (2008). Composing Multi-view Aspect Models. *Proceedings of the Seventh International Conference on Composition-Based Software Systems (ICCBSS 2008)*, 43-52. IEEE Computer Society.

Bartolomei, T. T., Garcia, A., Sant'Anna, C. and Figueiredo, E. (2006). Towards a unified coupling framework for measuring aspect-oriented programs. *Proceedings of the 3rd international workshop on Software quality assurance*, 46-53. ACM.

Basili, V., Briand, L. and Melo, W. (1996). A Validation of Object-Oriented Design Metrics as Quality Indicators. *Software Engineering* 22(10), 751-761.

beeck, S. O. d., Truyen, E., Bouck'e, N., Sanen, F., Bynens, M. and Joosen, W. (2006). A Study of Aspect-Oriented Design Approaches. Department of Computer Science K.U. Leuven.

Bennett, J., Cooper, K. and Dai, L. (2010). Aspect-oriented model-driven skeleton code generation: A graph-based transformation approach. *Science of Computer Programming* 75(8), 689-725.

Bennett, J. D. (2007). *An approach to aspect-oriented model-driven code generation using graph transformation. MS Thesis*. MS, The University of Texas at Dallas.

Booch, G. (1993). *Object-Oriented Analysis and Design with Applications (2nd Edition)*: Addison-Wesley Professional.

Brooks, F. (1995). *The mythical man-month : essays on software engineering*: Addison-Wesley Pub. Co.

Buckl, C., Regensburger, M., Knoll, A. and Schrott, G. (2007). Models for automatic generation of safety-critical real-time systems. *Second International Conference on Availability, Reliability and Security '07*, 580-587.

Burrows, R., Ferrari, F. C., Garcia, A. and Taiani, F. (2010a). An empirical evaluation of coupling metrics on aspect-oriented programs. *Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics*, 53-58. ACM.

Burrows, R., Garcia, A. and Taïani, F. (2010b). *Coupling Metrics for Aspect-Oriented Programming: A Systematic Review of Maintainability Studies*. In Maciaszek, L. A., González-Pérez, C. and Jablonski, S. (Ed.) *Evaluation of Novel Approaches to Software Engineering*. (277-290). Springer Berlin Heidelberg.

Cacho, N., Sant'Anna, C., Figueiredo, E., Garcia, A., Batista, T. and Lucena, C. (2006). Composing design patterns: a scalability study of aspect-oriented programming. *Proceedings of the 5th international conference on Aspect-oriented software development*, 109-121. ACM.

Carlson, D. (2001). *Modeling Xml Applications With Uml: Practical E-Business Applications*: Addison Wesley Publishing Company Incorporated.

Carton, A., Driver, C., Jackson, A. and Clarke, S. (2009). *Model-Driven Theme/UML*. In Katz, S., Ossher, H., France, R. and Jézéquel, J.-M. (Ed.) *Transactions on Aspect-Oriented Software Development VI*. (238-266). Springer Berlin / Heidelberg.

Cetina, C., Serral, E., Muñoz, J. and Pelechano, V. (2007). Tool Support for Model Driven Development of Pervasive Systems. *Fourth International Workshop on*

Model-Based Methodologies for Pervasive and Embedded Software (MOMPES '07), Portugal, 33-44.

Chauvel, F. and Jézéquel, J.-M. (2005). *Code Generation from UML Models with Semantic Variation Points*. In Briand, L. and Williams, C. (Ed.) *Model Driven Engineering Languages and Systems*. (54-68). Springer Berlin / Heidelberg.

Chavez, V. F. G. (2004). *A model-driven approach for aspect-oriented design*. Ph.D. dissertation, Pontif´ıcia Universidade Cat´olica do Rio de Janeiro.

Chidamber, S. R. and Kemerer, C. F. (1994). A Metrics Suite for Object Oriented Design. *IEEE Trans. Softw. Eng.* 20(6), 476-493.

Chitchyan, R. (2013). *Semantics-Based Composition for Textual Requirements*. In Moreira, A., Chitchyan, R., Araújo, J. and Rashid, A. (Ed.) *Aspect-Oriented Requirements Engineering*. (61-75). Springer Berlin Heidelberg.

Chitchyan, R., Rashid, A., Sawyer, P., Garcia, A., Alarcon, M. P., Bakker, J., Tekinerdogan, B., Clarke, S. and Jackson, A. (May 2005). Survey of Aspect-Oriented Analysis and Design Approaches. Technical Report AOSD. Europe Deliverable D11, AOSD-Europe-ULANC-9. Lancaster University.

Ciraci, S., Havinga, W., Aksit, M., Bockisch, C. and van den Broek, P. (2010). *A Graph-Based Aspect Interference Detection Approach for UML-Based Aspect-Oriented Models*. In Katz, S., Mezini, M. and Kienzle, J. (Ed.) *Transactions on Aspect-Oriented Software Development VII*. (321-374). Springer Berlin / Heidelberg.

Clarke, S. (2002). Extending standard UML with model composition semantics. *Sci. Comput. Program.* 44(1), 71-100.

Clarke, S. and Baniassad, E. (2005). *Aspect-Oriented Analysis and Design: The Theme Approach*: Addison Wesley Object Technology.

Clarke, S., Harrison, W., Ossher, H. and Tarr, P. (1999). Subject-oriented design: towards improved alignment of requirements, design, and code. *SIGPLAN Not.* 34(10), 325-339.

Clarke, S. and Walker, R. J. (2001). Composition patterns: an approach to designing reusable aspects. *Proceedings of the 23rd International Conference on Software Engineering*, 5-14. IEEE Computer Society.

Clarke, S. and Walker, R. J. (2002). Towards a standard design language for AOSD. *Proceedings of the 1st international conference on Aspect-oriented software development,* 113-119. ACM.

Clemente, P. J., Hernandez, J., Conejero, J. M. and Ortiz, G. (2011). Managing crosscutting concerns in component based systems using a model driven development approach. *J. Syst. Softw.* 84(6), 1032-1053.

Clifton, C. and Leavens, G. (2005). A Design Discipline and Language Features for Formal Modular Reasoning in Aspect-Oriented Programs. *Technical Report 05-23*.

Cohen, J. (1990). Constraint logic programming languages. *Communications of the ACM* 33(7), 52-68.

Conrad, R., Scheffner, D. and Freytag, J. C. (2000). XML conceptual modeling using UML. *Proceedings of the 19th international conference on Conceptual modeling,* 558-574. Springer-Verlag.

Cottenier, T., Berg, A. v. d. and Elrad, T. (2007a). *Joinpoint Inference from Behavioral Specification to Implementation*. In Ernst, E. (Ed.) *ECOOP 2007 – Object-Oriented Programming*. (476-500). Springer Berlin / Heidelberg.

Cottenier, T., Berg, A. v. d. and Elrad, T. (2007b). Motorola WEAVR: Aspect Orientation and Model-Driven Engineering. *Journal of Object Technology* 6(7), 51–88.

Cottenier, T., Berg, A. v. d. and Elrad, T. (2007c). The Motorola WEAVR: Model Weaving in a Large Industrial Context. *Proceedings of the 6th International Conference on Aspect-oriented Software Development (AOSD '07),*

Cottenier, T., Berg, A. v. d. and Elrad, T. (2007d). Stateful Aspects: The Case for Aspect-Oriented Modeling. *10th AOM Workshop*.

Creswell, J. W. (2003). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*: SAGE Publications.

Culwin, F. (2004). The statechart design of a novel date input mechanism. *Innovation in Teaching and Learning in Information and Computer Sciences* 3(1).

Dai, L. (2005a). *(Defense) Formal design analysis framework: an aspect-oriented architectural framework,* University of Texas at Dallas, Ph.D. Dissertation.

Dai, L. (2005b). *Formal design analysis framework: An aspect-oriented architectural framework*. Ph.D. 3224352, The University of Texas at Dallas.

Debnath, N., Baigorria, L., Riesco, D. and Montejano, G. (2008). Metrics applied to Aspect Oriented Design using UML profiles. *Computers and Communications, 2008. ISCC 2008. IEEE Symposium on*, 654-657.

Demchak, B., Ermagan, V., Farcas, E., Huang, T.-J., Kruger, I. H. and Menarini, M. (2008). *A Rich Services Approach to CoCoME*. In Andreas, R., Ralf, R., Raffaela, M.et al (Ed.) *The Common Component Modeling Example*. (85-115). Springer-Verlag.

Derezinska, A. and Pilitowski, R. (2008). Correctness issues of UML class and state machine models in the C# code generation and execution framework. *Computer Science and Information Technology, 2008. IMCSIT 2008. International Multiconference on*, 517-524.

Domı´nguez, E., Pérez, B., Rubio, Á. L. and Zapata, M. a. A. (2012). A systematic review of code generation proposals from state machine specifications. *Information and Software Technology* 54(10), 1045-1066.

Domínguez, E., Lloret, J., Pérez, B., Rodrı´guez, Á., Rubio, Á. L. and Zapata, M. a. A. (2011). Evolution of XML schemas and documents from stereotyped UML class models: A traceable approach. *Information and Software Technology* 53(1), 34-50.

Easterbrook, S., Singer, J., Storey, M.-A. and Damian, D. (2008). *Selecting Empirical Methods for Software Engineering Research*. In Shull, F., Singer, J. and Sjøberg, D. K. (Ed.) *Guide to Advanced Empirical Software Engineering*. (285-311). Springer London.

Eclipse.org (2012). Open Source Software Engineering Tools, Eclipse UML2 Tools.

Elrad, T., Aldawud, O. and Bader, A. (2002). *Aspect-Oriented Modeling: Bridging the Gap between Implementation and Design* In Batory, D., Consel, C. and Taha, W. (Ed.) *Generative Programming and Component Engineering*. (189-201). Springer Berlin / Heidelberg.

Evermann, J. (2007). A meta-level specification and profile for AspectJ in UML. *Proceedings of the 10th international workshop on Aspect-oriented modeling*, 21-27. ACM.

Evermann, J., Fiech, A. and Alam, F. E. (2011). A platform-independent UML profile for aspect-oriented development. *Proceedings of The Fourth International C* Conference on Computer Science and Software Engineering*, 25-34. ACM.

Fenton, N. and Pfleeger, S. (1998). *Software Metrics: A Rigorous and Practical Approach*: PWS Publishing Co.

Fleurey, F., Baudry, B., France, R. and Ghosh, S. (2008). A Generic Approach For Automatic Model Composition. *11th AOM Workshop*.

Fleurey, F., Breton, E., Baudry, B., Nicolas, A. and J´ez´equel, J.-M. (2007). Model-driven engineering for software migration in a large industrial context. *Proceedings of the 10th international conference on Model Driven Engineering Languages and Systems*, 482-497. Springer-Verlag.

France, R., Ray, I., Georg, G. and Ghosh, S. (2004). Aspect-oriented approach to early design modelling. *Software, IEE Proceedings* - 151(4), 173-185.

Fuentes, L. and S´anchez, P. (2006). Elaborating UML 2.0 Profiles for AO Design. *8th Workshop on AOM, 5th Int. Conference on AOSD*.

Fuentes, L. and Sanchez, P. (2006). A generic MOF metamodel for aspect-oriented modelling. *Model-Based Development of Computer-Based Systems and Model-Based Methodologies for Pervasive and Embedded Software, 2006. MBD/MOMPES 2006. Fourth and Third International Workshop on*, 10 pp.-124.

Fuentes, L. and Sanchez, P. (2007a). Designing and Weaving Aspect-Oriented Executable UML models. *Journal of Object Technology* 6(7), 109-136.

Fuentes, L. and Sanchez, P. (2007b). Towards executable aspect-oriented UML models. *Proceedings of the 10th international workshop on Aspect-oriented modeling*, 28-34. ACM.

Fuentes, L. and Sánchez, P. (2007). *Execution of Aspect Oriented UML Models*. In Akehurst, D., Vogel, R. and Paige, R. (Ed.) *Model Driven Architecture-Foundations and Applications*. (83-98). Springer Berlin / Heidelberg.

Fuentes, L. and Sánchez, P. (2009). *Dynamic Weaving of Aspect-Oriented Executable UML Models*. In Katz, S., Ossher, H., France, R. and Jézéquel, J.-M. (Ed.) *Transactions on Aspect-Oriented Software Development VI*. (1-38). Springer Berlin / Heidelberg.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*: Addison-Wesley.

Garcia, A. (2004). *From Objects to Agents: An Aspect-Oriented Approach*. Ph.D. Thesis, PUC-Rio, Rio de Janeiro, Brazil.

Garcia, A., Sant'Anna, C., Figueiredo, E., Kulesza, U., Lucena, C. and Staa, A. v. (2005). Modularizing design patterns with aspects: a quantitative study. *Proceedings of the 4th international conference on Aspect-oriented software development*, 3-14. ACM.

Garcia, A., Sant’Anna, C., Chavez, C., Silva, V., Lucena, C. P. and Staa, A. (2004). *Separation of Concerns in Multi-agent Systems: An Empirical Study*. In Lucena, C., Garcia, A., Romanovsky, A., Castro, J. and Alencar, P. C. (Ed.) *Software Engineering for Multi-Agent Systems II*. (49-72). Springer Berlin Heidelberg.

Giunta, R., Pappalardo, G. and Tramontana, E. (2012). AODP: refactoring code to provide advanced aspect-oriented modularization of design patterns. *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, 1243-1250. ACM.

Grace, P., Truyen, E., Lagaisse, B. and Joosen, W. (2007). The case for aspect-oriented reflective middleware. *Proceedings of the 6th international workshop on Adaptive and reflective middleware: held at the ACM/IFIP/USENIX International Middleware Conference*, 1-6. ACM.

Greenwood, P., Bartolomei, T., Figueiredo, E., Dosea, M., Garcia, A., Cacho, N., Sant’Anna, C., Soares, S., Borba, P., Kulesza, U. and Rashid, A. (2007). *On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study*. In Ernst, E. (Ed.) *ECOOP 2007 – Object-Oriented Programming*. (176-200). Springer Berlin / Heidelberg.

Groher, I. and Baumgarth, T. (2004). Aspect-Orientation from Design to Code. *Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design*, 62-68.

Groher, I. and Schulze, S. (2003). Generating aspect code from UML models. *The Third International Workshop on Aspect-Oriented Modeling*,

Groher, I. and Voelter, M. (2007). XWeave: models and aspects in concert. *Proceedings of the 10th international workshop on Aspect-oriented modeling*, 35-40. ACM.

Grønmo, R., Sørensen, F., Møller-Pedersen, B. and Krogdahl, S. (2008). *Semantics-Based Weaving of UML Sequence Diagrams*. In Vallecillo, A., Gray, J. and Pierantonio, A. (Ed.) *Theory and Practice of Model Transformations*. (122-136). Springer Berlin / Heidelberg.

Grose, T. J., Doney, G. C. and Brodsky, S. A. (2002). *Mastering XMI: Java Programming with XMI, XML and UML*: Wiley.

Group, O. (2007). OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2.

Grundy, J. (2000). Multi-perspective specification, design and implementation of software components using aspects. *International Journal of Software Engineering and Knowledge Engineering* 10(6).

Haitao, S., Zhumei, S. and Shixiong, Z. (2006). Mapping Aspect-Oriented Domain-Specific Model to Code for Real Time System. *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on*, 6426-6431.

Hanenberg, S., Hirschfeld, R. and Unland, R. (2004). Morphing aspects: incompletely woven aspects and continuous weaving. *Proceedings of the 3rd international conference on Aspect-oriented software development*, 46-55. ACM.

Hanenberg, S., Stein, D. and Unland, R. (2007). From aspect-oriented design to aspect-oriented programs: tool-supported translation of JPDDs into code. *Proceedings of the 6th international conference on Aspect-oriented software development*, 49-62. ACM.

Hannemann, J. and Kiczales, G. (2002). Design pattern implementation in Java and aspectJ. *SIGPLAN Not.* 37(11), 161-173.

Harada, M., Fujisawa, T., Teradaira, M., Yamamoto, K. and Hamada, S. (1996). Refinement of Dynamic Modeling of Some Automatic Layouting of Object Oriented Design Schema and Reverse Engineering of Design Schema from C++ Program. *IPSJ Object-Oriented Symposium*, Tokyo, Japan, 111-118.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.* 8(3), 231-274.

Harrison, W., Ossher, H. and Tarr, P. (2002). Asymmetrically vs. symmetrically organized paradigms for software composition.

Hecht, M. V., Piveta, E. K., Pimenta, M. S. and Price, R. T. (2005). Aspect-oriented Code Generation. *XX Brazilian Conference on Software Engineering*.

Ho, W.-M., Jezequel, J.-M., Pennaneac'h, F. and Plouzeau, N. (2002). A toolkit for weaving aspect oriented UML designs. *Proceedings of the 1st international conference on Aspect-oriented software development*, 99-105. ACM.

Hoffman, K. and Eugster, P. (2013). Trading obliviousness for modularity with cooperative aspect-oriented programming. *ACM Trans. Softw. Eng. Methodol.* 22(3), 1-46.

Hohenstein, U. D. and Gleim, U. (2011). Using aspect-orientation to simplify concurrent programming. *Proceedings of the tenth international conference on Aspect-oriented software development companion*, 29-40. ACM.

Hohenstein, U. D. C. and Jager, M. C. (2009). Using aspect-orientation in industrial projects: appreciated or damned? *Proceedings of the 8th ACM international conference on Aspect-oriented software development*, 213-222. ACM.

Hovsepyan, A., Scandariato, R., Baelen, S. V., Berbers, Y. and Joosen, W. (2010). From aspect-oriented models to aspect-oriented code?: the maintenance perspective. *Proceedings of the 9th International Conference on Aspect-Oriented Software Development*, 85-96. ACM.

Hovsepyan, A., Van Baelen, S., Vanhooff, B., Joosen, W. and Berbers, Y. (2006). *Key Research Challenges for Successfully Applying MDD Within Real-Time Embedded Software Development*

Embedded Computer Systems: Architectures, Modeling, and Simulation. In Vassiliadis, S., Wong, S. and Hämäläinen, T. (Ed.). (49-58). Springer Berlin / Heidelberg.

Jackson, A., Casey, N. and Clarke, S. (2008). Mapping design to implementation. AOSD-Europe TDC-D111. http://www.aosd-europe.net/deliverables/d111.pdf.

Jackson, A., Klein, J., Baudry, B. and Clarke, S. (2006). KerTheme: Testing Aspect Oriented Models. *Workshop on Integration of Model Driven Development and Model Driven Testing (ECMDA'06),* Bilbao, Spain,

Jacobson, I. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach*: Addison-Wesley Professional.

Jacobson, I. and Ng, P.-W. (2004). *Aspect-Oriented Software Development with Use Cases*: Addison-Wesley Professional.

Jakimi, A. and Elkoutbi, M. (2009). An object-oriented approach to UML scenarios engineering and code generation. *International Journal of Computer Theory and Engineering (IJCTE)* 1(1), 35-41.

Jalali, A., Wohed, P., Ouyang, C. and Johannesson, P. (2013). *Dynamic Weaving in Aspect Oriented Business Process Management*. In Meersman, R., Panetto, H., Dillon, T.et al (Ed.) *On the Move to Meaningful Internet Systems: OTM 2013 Conferences*. (2-20). Springer Berlin Heidelberg.

Jézéquel, J.-M. (2008). Model driven design and aspect weaving. *Software and Systems Modeling* 7(2), 209-218.

Jingjun, Z., Yuejuan, C. and Guangyuan, L. (2009). Modeling Aspect-Oriented Programming with UML Profile. *Education Technology and Computer Science, 2009. ETCS '09. First International Workshop on*, 242-245.

Kagdi, H., Collard, M. L. and Maletic, J. I. (2007). A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *J. Softw. Maint. Evol.* 19(2), 77-131.

Karakostas, B. and Zorgios, Y. (2008). *Engineering Service Oriented Systems: A Model Driven Approach*: IGI Global.

Katara, M. and Katz, S. (2003). Architectural views of aspects. *Proceedings of the 2nd international conference on Aspect-oriented software development*, 1-10. ACM.

Katara, M. and Katz, S. (2007). A concern architecture view for aspect-oriented software design. *Software and Systems Modeling* 6(3), 247-265.

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J. and Griswold, W. G. (2001). An Overview of AspectJ. *Proceedings of the 15th European Conference on Object-Oriented Programming*, 327-353. Springer-Verlag.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M. and Irwin, J. (1997). *Aspect-oriented programming*. In Aksit, M. and Matsuoka, S. (Ed.) *ECOOP'97 — Object-Oriented Programming*. (220-242). Springer Berlin / Heidelberg.

Kiczales, G. and Mezini, M. (2005). Aspect-oriented programming and modular reasoning. *Proceedings of the 27th international conference on Software engineering*, 49-58. ACM.

Kienzle, J. (2013). Reusing software design models with TouchRAM. *Proceedings of the 12th annual international conference companion on Aspect-oriented software development*, 23-26. ACM.

Kienzle, J., Abed, W. A. and Klein, J. (2009). Aspect-oriented multi-view modeling. *Proceedings of the 8th ACM international conference on Aspect-oriented software development*, 87-98. ACM.

Kienzle, J., Al Abed, W., Fleurey, F., Jézéquel, J.-M. and Klein, J. (2010). *Aspect-Oriented Design with Reusable Aspect Models*. In Katz, S., Mezini, M. and Kienzle, J. (Ed.) *Transactions on Aspect-Oriented Software Development VII*. (272-320). Springer Berlin / Heidelberg.

Klein, J., Fleurey, F. and Jézéquel, J.-M. (2007). *Weaving Multiple Aspects in Sequence Diagrams*. In Rashid, A. and Aksit, M. (Ed.) *Transactions on Aspect-Oriented Software Development III*. (167-199). Springer Berlin / Heidelberg.

Klein, J., Helouet, L. and Jezequel, J.-M. (2006). Semantic-based weaving of scenarios. *Proceedings of the 5th international conference on Aspect-oriented software development*, 27-38. ACM.

Klein, J. and Kienzle, J. (2007). Reusable Aspect Models. *11th Workshop on Aspect-Oriented Modeling*.

Klein, J., Kienzle, J., Morin, B. and Jézéquel, J.-M. (2009). *Aspect Model Unweaving*. In Schürr, A. and Selic, B. (Ed.) *Model Driven Engineering Languages and Systems*. (514-530). Springer Berlin / Heidelberg.

Knapp, A. and Merz, S. (2002). Model checking and code generation for UML state machines and collaborations. *Proceedings of 5th Workshop on Tools for System Design and Verification, Technical Report* 11, 59-64.

Knapp, A., Merz, S. and Rauh, C. (2002). Model Checking - Timed UML State Machines and Collaborations. *Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems: Co-sponsored by IFIP WG 2.2,* 395-416. Springer-Verlag.

Kon, F., Costa, F., Blair, G. and Campbell, R. H. (2002). The case for reflective middleware. *Commun. ACM* 45(6), 33-38.

Kramer, M. and Kienzle, J. (2011). *Mapping Aspect-Oriented Models to Aspect-Oriented Code*. In Dingel, J. and Solberg, A. (Ed.) *Models in Software Engineering*. (125-139). Springer Berlin / Heidelberg.

Kramer, M. E. (2010). *Mapping Reusable Aspect Models to aspect-oriented code*, Karlsruhe Institute of Technology, Germany.

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

Lamancha, B., Reales, P., Polo, M. and Caivano, D. (2013). *Model-Driven Test Code Generation*. In Maciaszek, L. and Zhang, K. (Ed.) *Evaluation of Novel Approaches to Software Engineering*. (155-168). Springer Berlin Heidelberg.

Leroux, D., Nally, M. and Hussey, K. (2006). Rational Software Architect: A tool for domain-specific modeling. *IBM Systems Journal* 45(3), 555-568.

Long, E., Misra, A. and Sztipanovits, J. (1998). Increasing productivity at Saturn. *Computer* 31(8), 35-43.

Lopes, C. and Kiczales, G. (1997). D: A Language Framework for Distributed Programming. (SPL97-010, P9710047).

Losavio, F., Matteo, A. and Morantes, P. (2009). UML Extensions for Aspect Oriented Software Development. *Journal of Object Technology* 8(5), 105-132.

Loukil, S., Kallel, S., Zalila, B. and Jmaiel, M. (2013). AO4AADL: Aspect oriented extension for AADL. *Central European Journal of Computer Science* 3(2), 43-68.

Mandić, V., Markkula, J. and Oivo, M. (2009). *Towards Multi-Method Research Approach in Empirical Software Engineering*. In Bomarius, F., Oivo, M., Jaring, P. and Abrahamsson, P. (Ed.) *Product-Focused Software Process Improvement*. (96-110). Springer Berlin Heidelberg.

Maoz, S. and Harel, D. (2006). From multi-modal scenarios to code: compiling LSCs into aspectJ. *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering,* 219-230. ACM.

Mehner, K., Monga, M. and Taentzer, G. (2009). *Analysis of Aspect-Oriented Model Weaving*. In Rashid, A. and Ossher, H. (Ed.) *Transactions on Aspect-Oriented Software Development V*. (235-263). Springer Berlin / Heidelberg.

Mellor, S. J. and Balcer, M. (2002). *Executable UML: A Foundation for Model-Driven Architectures*: Addison-Wesley Longman Publishing Co., Inc.

Morin, B., Vanwormhoudt, G., Lahire, P., Gaignard, A., Barais, O. and Jézéquel, J.-M. (2008). *Managing Variability Complexity in Aspect-Oriented Modeling*. In Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A. and Völter, M. (Ed.) *Model Driven Engineering Languages and Systems*. (797-812). Springer Berlin / Heidelberg.

Mosconi, M., Charfi, A., Svacina, J. and Wloka, J. (2008). Applying and evaluating AOM for platform independent behavioral UML models. *Proceedings of the 12th workshop on Aspect-oriented modeling,* 19-24. ACM.

Mouheb, D., Talhi, C., Nouh, M., Lima, V., Debbabi, M., Wang, L. and Pourzandi, M. (2010). *Aspect-Oriented Modeling for Representing and Integrating Security Concerns in UML*. In Lee, R., Ormandjieva, O., Abran, A. and Constantinides, C. (Ed.) *Software Engineering Research, Management and Applications 2010*. (197-213). Springer Berlin / Heidelberg.

Mussbacher, G., Amyot, D., Whittle, J. and Weiss, M. (2007). *Flexible and Expressive Composition Rules with Aspect-oriented Use Case Maps (AoUCM)*. In Moreira, A. and Grundy, J. (Ed.) *Early Aspects: Current Challenges and Future Directions*. (19-38). Springer Berlin Heidelberg.

Neto, P. A. d. M. S., Machado, I. d. C., McGregor, J. D., Almeida, E. S. d. and Meira, S. R. d. L. (2011). A systematic mapping study of software product lines testing. *Information and Software Technology* 53(5), 407-423.

Niaz, I. A. (2005). *Automatic Code Generation From UML Class and Statechart Diagrams*. PhD PhD Thesis, University of Tsukuba, Ph.D. Thesis.

Niaz, I. A. and Tanaka, J. (2003). Code Generation from UML Statecharts. *7th IASTED International Conf. on Software Engineering and Application (SEA 2003)*, Marina Del Rey, USA, 315-321.

Niaz, I. A. and Tanaka, J. (2004). Mapping UML Statecharts to Java Code. *IASTED International Conf. on Software Engineering (SE 2004)*, Innsbruck, Austria, 111-116.

Niaz, I. A. and Tanaka, J. (2005). An Object-Oriented Approach to Generate Java Code from UML Statecharts. *International Journal of Computer & Information Science* 6(2).

NoMagic (2011). MagicDraw UML. www.magicdraw.com/.

Oldevik, J., Menarini, M. and Krüger, I. (2009). *Model Composition Contracts*. In Schürr, A. and Selic, B. (Ed.) *Model Driven Engineering Languages and Systems*. (531-545). Springer Berlin / Heidelberg.

OMG (2007). MOF 2.0/XMI Mapping, Version 2.1.1.

OMG (2009). Unified Modelling Language Specification: Superstructure v2.2

OMG (2010). UML 2.3 Superstructure Specification Document Formal/2010-05-05. http://www.omg.org/. (Accessed May 2013).

Papotti, P., Prado, A., Souza, W., Cirilo, C. and Pires, L. (2013). *A Quantitative Analysis of Model-Driven Code Generation through Software Experimentation*. In Salinesi, C., Norrie, M. and Pastor, Ó. (Ed.) *Advanced Information Systems Engineering*. (321-337). Springer Berlin Heidelberg.

Pawlak, R., Duchien, L., Florin, G., Legond-Aubry, F., Seinturier, L. and Martelli, L. (2002). A UML Notation for Aspect-Oriented Software Design. *AO modeling with UML workshop at the AOSD'02.*

Pawlak, R., Seinturier, L., Duchien, L., Martelli, L., Legond-Aubry, F. and Florin, G. (2005). *Aspect-Oriented Software Development with Java Aspect Components*. In Filman, R., Elrad, T., Clarke, S. and Aksit, M. (Ed.) *Aspect-oriented software development*. (343-369). Addison-Wesley.

Petter, S. C. and Gallivan, M. J. (2004). Toward a framework for classifying and guiding mixed method research in information systems. *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, 10 pp.

Philippi, S. (2006). Automatic code generation from high-level Petri-Nets for model driven systems engineering. *Journal of Systems and Software* 79(10), 1444-1455.

Pilitowski, R. and Dereziñska, A. (2007). *Code Generation and Execution Framework for UML 2.0 Classes and State Machines*. In Sobh, T. (Ed.) *Innovations and Advanced Techniques in Computer and Information Sciences and Engineering*. (421-427). Springer Netherlands.

Pintér, G. and IstvánMajzik (2003). Program Code Generation Based On UML Statechart Models. *Periodica Polytechnica* 47(3), 187-204.

Pinto, M., Fuentes, L. and Fernández, L. (2012). Deriving detailed design models from an aspect-oriented ADL using MDD. *Journal of Systems and Software* 85(3), 525-545.

Pinto, M., Fuentes, L. and Troya, J. M. (2005). A Dynamic Component and Aspect-Oriented Platform. *The Computer Journal* 48(4), 401-420.

Pitkänen, R. and Selonen, P. (2004). *A UML Profile for Executable and Incremental Specification-Level Modeling*. In Baar, T., Strohmeier, A., Moreira, A. and Mellor, S. J. (Ed.) *UML 2004 - The Unified Modelling Language*. (158-172). Springer Berlin / Heidelberg.

Piveta, E. K., Moreira, A., Pimenta, M. S., Araújo, J., Guerreiro, P. and Price, R. T. (2012). An empirical study of aspect-oriented metrics. *Science of Computer Programming* 78(1), 117-144.

Pleuss, A., Wollny, S. and Botterweck, G. (2013). Model-driven development and evolution of customized user interfaces. *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems*, 13-22. ACM.

Rahmouni, M. and Mbarki, S. (2013). An end-to-end code generation from UML diagrams to MVC2 web applications. *International Review on Computers and Software* 8(9), 2123-2135.

Rasche, A., Schult, W. and Polze, A. (2005). Self-adaptive multithreaded applications: a case for dynamic aspect weaving. *Proceedings of the 4th workshop on Reflective and adaptive middleware systems*. ACM.

Rashid, A., Moreira, A., Araujo, J., Clements, P., Baniassad, E. and Tekinerdogan, B. (2006). Early aspects: Aspect-oriented requirements engineering and architecture design. Electronic Document. http://www.early-aspects.net/.

Rauchwerger, Y., Kristoffersen, F. and Lahav, Y. (2005). *Cinderella SLIPPER: An SDL to C-Code Generator*. In Prinz, A., Reed, R. and Reed, J. (Ed.) *SDL 2005: Model Driven*. (1159-1165). Springer Berlin / Heidelberg.

Reddy, Y., Ghosh, S., France, R., Straw, G., Bieman, J., McEachen, N., Song, E. and Georg, G. (2006). *Directives for Composing Aspect-Oriented Design Class Models*. In Rashid, A. and Aksit, M. (Ed.) *Transactions on Aspect-Oriented Software Development I*. (75-105). Springer Berlin / Heidelberg.

Reina, A. M., Torres, J. and Toro, M. (2004). Separating concerns by means of UML-profiles and metamodels in PIMs. *The 5th Aspect-Oriented Modeling Workshop In Conjunction with UML 2004,*

Routledge, N., Bird, L. and Goodchild, A. (2002). UML and XML schema. *Aust. Comput. Sci. Commun.* 24(2), 157-166.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorenson, W. (1991). *Object-Oriented Modeling and Design*: {Prentice Hall, Inc.}.

Samek, M. and Montgomery, P. (2000). State-oriented programming. *International Journal of Embedded Systems* 13(8), 22-43.

Sánchez, P., Fuentes, L., Stein, D., Hanenberg, S. and Unland, R. (2008). *Aspect-Oriented Model Weaving Beyond Model Composition and Model Transformation*. In Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A. and Völter, M. (Ed.) *Model Driven Engineering Languages and Systems*. (766-781). Springer Berlin / Heidelberg.

Sánchez, P., Moreira, A., Fuentes, L., Araújo, J. and Magno, J. (2010). Model-driven development for early aspects. *Information and Software Technology* 52(3), 249-273.

Sant'anna, C., Garcia, A., Chavez, C., Lucena, C. and von Staa, A. (2003). On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework. *Proceedings XVII Brazilian Symposium on Software Engineering,* Saraiva, J., Barreiros, E., Almeida, A., Lima, F., Alencar, A., Lima, G., Soares, S. and Castor, F. (2012). Aspect-oriented software maintenance metrics: A systematic mapping study, 253-262.

Saurabh, A., Dahiya, D. and Mohana, R. (2012). *Maximizing Automatic Code Generation: Using XML Based MDA*. In Parashar, M., Kaushik, D., Rana, O.et al (Ed.) *Contemporary Computing*. (283-293). Springer Berlin Heidelberg.

Simmonds, D. M. (2008). Aspect-oriented Approaches to Model Driven Engineering. *International Conference on Software Engineering Research and Practice*, Las Vegas, Nevada, USA,

Simmonds, D. M. and Reddy, Y. R. (2009). A Comparison of Aspect-Oriented Approaches to Model Driven Engineering. *Conference on Software Engineering Research and Practice*, 327–333

Sommerville, I. (2010). *Software Engineering*: Pearson.

Stavrou, A. and Papadopoulos, G. A. (2009). *Automatic Generation of Executable Code from Software Architecture Models*. In (Ed.) *Information Systems Development*. (447-458). Springer US.

Stein, D., Hanenberg, S. and Unland, R. (2002a). Designing Aspect-Oriented Crosscutting in UML. *AOSD-UML Workshop at AOSD '02*.

Stein, D., Hanenberg, S. and Unland, R. (2002b). An UML-based aspect-oriented design notation for AspectJ. *Proceedings of the 1st international conference on Aspect-oriented software development*, 106-112. ACM.

Stein, D., Hanenberg, S. and Unland, R. (2006). Expressing different conceptual models of join point selections in aspect-oriented design. *Proceedings of the 5th international conference on Aspect-oriented software development*, 15-26. ACM.

Tarr, P., Ossher, H., Harrison, W. and Stanley M. Sutton, J. (1999). N degrees of separation: multi-dimensional separation of concerns. *Proceedings of the 21st international conference on Software engineering*, 107-119. ACM.

Thongmak, M. and Muenchaisri, P. (2002). Design of Rules for Transforming UML Sequence Diagrams into Java code. *Proceedings of the Ninth Asia-Pacific Software Engineering Conference*, 485. IEEE Computer Society.

Thongmak, M. and Muenchaisri, P. (2011). *Measuring Understandability of Aspect-Oriented Code*. In Cherifi, H., Zain, J. and El-Qawasmeh, E. (Ed.) *Digital Information and Communication Technology and Its Applications*. (43-54). Springer Berlin Heidelberg.

Tigris.org (2012). Open Source Software Engineering Tools, ArgoUML Modeling Tool. http://argouml.tigris.org.

Tun, T., Yu, Y., Jackson, M., Laney, R. and Nuseibeh, B. (2013). *Aspect Interactions: A Requirements Engineering Perspective*. In Moreira, A., Chitchyan, R., Araújo, J. and Rashid, A. (Ed.) *Aspect-Oriented Requirements Engineering*. (271-286). Springer Berlin Heidelberg.

Usman, M. and Nadeem, A. (2009). Automatic Generation of Java Code from UML Diagrams using UJECTOR. *International Journal of Software Engineering and Its Applications* 3(2), 21-37.

Vanderperren, W., Suvée, D., Cibrán, M. and Fraine, B. (2005). *Stateful Aspects in JAsCo*. In Gschwind, T., Aßmann, U. and Nierstrasz, O. (Ed.) *Software Composition*. (167-181). Springer Berlin Heidelberg.

W3C (2013). Extensible Markup Language (XML) 1.0 (Fifth Edition). World Wide World Consortium. Available at: http://www.w3.org/TR/xml/.

Wang, R., Mao, X.-G., Dai, Z.-Y. and Wang, Y.-N. (2009). Extending UML for Aspect-Oriented Architecture Modeling. *Computer Science and Engineering, 2009. WCSE '09. Second International Workshop on*, 362-366.

Wehrmeister, M. A., Freitas, E. P., Pereira, C. E. and Rammig, F. (2008). GenERTiCA: A Tool for Code Generation and Aspects Weaving. *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, 234-238.

White, J., Gray, J. and Schmidt, D. (2009). *Constraint-Based Model Weaving*. In Katz, S., Ossher, H., France, R. and Jézéquel, J.-M. (Ed.) *Transactions on Aspect-Oriented Software Development VI*. (153-190). Springer Berlin / Heidelberg.

Whittle, J. and Jayaraman, P. (2008). MATA: A Tool for Aspect-Oriented Modeling based on Graph Transformation. *11th AOM Workshop*.

Whittle, J., Jayaraman, P., Elkhodary, A., Moreira, A. and Araújo, J. (2009). *MATA: A Unified Approach for Composing UML Aspect Models Based on Graph Transformation*. In Katz, S., Ossher, H., France, R. and Jézéquel, J.-M. (Ed.) *Transactions on Aspect-Oriented Software Development VI*. (191-237). Springer Berlin / Heidelberg.

Whittle, J., Moreira, A., Arajo, J., Jayaraman, P., Elkhodary, A. and Rabbi, R. (2007). An Expressive Aspect Composition Language for UML State Diagrams. *MoDELS*, 514-528.

Wimmer, M., Schauerhuber, A., Kappel, G., Retschitzegger, W., Schwinger, W. and Kapsammer, E. (2011). A survey on UML-based aspect-oriented design modeling. *ACM Comput. Surv.* 43(4), 1-33.

Wu, I.-C. and Hsieh, S.-H. (2002). An UML-XML-RDB Model Mapping Solution for Facilitating Information Standardization and Sharing in Construction Industry. *Proceeding of the 19th International Symposium on Automation and Robotics in Construction (ISARC)* Maryland, 317-321.

Xu, D. and Xu, W. (2006). State-based incremental testing of aspect-oriented programs. *Proceedings of the 5th international conference on Aspect-oriented software development*, 180-189. ACM.

Zakaria, A. A., H. Hosny, A. Zeid (2002). A UML Extension for Modeling Aspect-Oriented Systems. *Fifth International Conference on the Unified Modeling Language - the Language and its Applications*

Zhang, G. (2005). Towards aspect-oriented class diagrams. *Software Engineering Conference, 2005. APSEC '05. 12th Asia-Pacific*, 6 pp.

Zhang, G. (2012). *Aspect-Oriented Modeling of Mutual Exclusion in UML State Machines*. In Vallecillo, A., Tolvanen, J.-P., Kindler, E., Störrle, H. and Kolovos, D. (Ed.) *Modelling Foundations and Applications*. (162-177). Springer Berlin Heidelberg.

Zhang, G., Hölzl, M. and Knapp, A. (2007). *Enhancing UML State Machines with Aspects*. In Engels, G., Opdyke, B., Schmidt, D. and Weil, F. (Ed.) *Model Driven Engineering Languages and Systems*. (529-543). Springer Berlin / Heidelberg.

Zhou, X., Liu, C., Niu, Y. and Lai, T. (2008). Towards a Framework of Aspect-Oriented Modeling with UML. *Computer Science and Computational Technology, 2008. ISCSCT '08. International Symposium on*, 738-741.

APPENDIX A

AOM APPROACHES EVALUATED IN THE CONTEXT OF CODE GENERATION

A.1) The Aspect-Oriented Design Model Notation for AspectJ

Aspect-Oriented Design Model (AODM) notation of Stein *et al.* (2002b) provides a design technique specific to AspectJ. Therefore, it extends UML with the only intention to support AspectJ's concepts at the design level. To exploit the huge resemblance between the core concepts of AspectJ and UML, it provides UML representation for basic constructs of AspectJ, namely join points, pointcuts, introductions, and aspects. Mainly the class, statechart and sequence diagrams are used for structure and behavior modeling. They represent join points using UML links, and apply the concept of adopted links to different diagrams in a way specific to each diagram. Similarly, an advice in AspectJ is viewed as analogous to an operation in UML. Aspects are represented as classes of a special stereotype named `<<aspect>>`. In principle, AODM has been specified using the UML's standard extension mechanism, but for certain specifications meta-model has also been extended. For example, the UML extend relationship, from which the `<<crosscut>>` stereotype has been derived originally, can be specified between use cases only.

A.2) The UMLAUT Framework

UML All pUrpose Transformer (UMLAUT) is an open framework proposed by Ho *et al.* (2002) for developing application-specific weavers that can generate detailed aspect-oriented design from a high level design modeled using UML. The UML model elements can be input using various formats such as XMI and Java source. Extensions to UML are done through the UML Profile mechanism. The weaving process is implemented as a model transformation applied to the UML model. Specifically, the weave operation is defined as a transformation rule from an initial model to a final one.

A.3) The UML Profile for AOSD

The UML Profile for Aspect-Oriented Software development has been presented by Aldawud (2003). It extends the standard UML package structure to define an AOSD package, which is used to encapsulate all elements defined by the AOSD profile. Crosscutting concerns are modeled using aspects, which are extensions to UML core classes. A new stereotype `<<aspect>>` is used to model aspects, which are further classified into synchronous or asynchronous aspects. In this profile, synchronous aspects are distinguished from asynchronous ones in that they usually control the behavior of the core classes. The `<<crosscut>>` stereotype is used to model crosscutting relationships. For behavior modeling, this profile does not dictate any specific behavioral package; however, currently only the use of Collaboration and State machine packages has been outlined for this profile.

A.4) aSideML Notation

The aSideML's Notation of Chavez (2004) is a meta-model extension of UML to support aspect-oriented concepts. Aspects are defined by parameterizing different model elements, and one or more crosscutting interface is defined to organize join point description and the crosscutting behavior of the aspect. Crosscutting features are defined as an extension to the original features of the class. Specifically, to model structure, a new construct called aspect diagram is introduced which extends features of a UML class diagram. Collaboration and sequence diagrams are extended for modeling behavior of the aspect. The join points are defined by means of an enhanced sequence diagram. Weaving of models is also provided which supports the same set of diagrams and generates woven class diagrams, woven collaboration diagrams and woven sequence diagrams.

A.5) Theme/UML

Theme/UML (Clarke and Walker, 2002; Baniassad and Clarke, 2004; Clarke and Baniassad, 2005) has basically evolved from work on composition patterns (Clarke and Walker, 2001; Clarke and Walker, 2002), and is considered one of the early approaches to aspect-oriented modeling. In this approach, a new declaratively complete unit named “Theme” is proposed at the design level to represent system concerns, which are essentially collections of structures and behaviors inferred from requirements. A distinction has been made between the “base” themes and the “aspect” themes, where aspect themes refer to crosscutting behavior. In Theme/UML approach, an aspect theme is differentiated from a base theme in the sense that in addition to other behavior, it may define some behavior that is triggered by behavior in some other theme. As far as modeling is process is concerned, first the triggered behavior needs to be identified and captured in the form of templates and then the crosscutting behavior related to those templates is modeled. Later, the base themes which are not affected by the crosscutting themes are modeled using the standard UML design process. A different approach is used to modeling of aspect themes by representing them using a new complete unit of modularization similar to a package in standard UML, with stereotype `<<theme>>.` This theme may comprise of any of the standard UML diagrams to model different views of the structure and behavior required for a concern to execute. Essentially, the aspect theme design is similar to a standard UML package that contains structural and behavioral diagrams. The only difference is the specification of templates listed inside the theme package notation and a sequence diagram for each of the templates grouping in the theme package. Even though Theme/UML allows any kind of UML diagrams to be used for aspect-theme design, package and class diagrams are currently used for structure modeling, whereas sequence diagrams are used for behavior modeling.

A.6) Aspect-Oriented Software Development with Use Cases

Jacobson and Ng (2004) have proposed an approach which is based on the idea of using a development process where concerns are kept separated from requirements specification to the implementation phase. For this purpose, they define use case slices to specify high-level design and then refine this design to obtain detailed design. At detailed design level, they represent structure by means of class diagrams and behavior by means of sequence diagrams. Model weaving is not supported. One distinguishing characteristic of this approach is its support for traceability of models pertaining to a specific concern along different phases of software development.

A.7) Aspect-Oriented Architecture Models

The Aspect-Oriented Architecture Model approach of France *et al.* (2004) is based on composing model elements that present a single concept using different views. The model elements that are composed using this approach are needed to be of the same type. Aspects may specify concepts that are not present in a target model. Templates are used in conjunction with package diagrams, class diagrams, communication diagrams and sequence diagram to represent aspects. In this respect, this approach is similar to Theme/UML approach described previously. The compositor composition mechanism is used to provide the concern composition. Just like Theme/UML, primary models and aspect models are distinguished, where the latter represent crosscutting behavior. Later, a tool called Kompose (Fleurey *et al.*, 2008) has also been developed which uses the composition technique proposed by Aspect-Oriented Architecture Models approach.

A.8) The UML Notation for AOSD

The notation of (Pawlak *et al.* (2002); Pawlak *et al.* (2005)) is a UML profile based on UML 1.x to model a design using JAC Framework, which is a middleware to support concerns such as persistence, security, fault tolerance etc. in J2EE applications. Currently, the profile does not support behavior modeling, whereas the support for structure modeling is provided by means of class diagrams. `<<aspect>>` stereotype is used to represent aspects, and they are linked with a target class using `<<pointcut>>` stereotypes. The association between the operations of base and aspect classes (i.e., the join point) is specified with the help of a proprietary language. This approach is similar to that of Jacobson et al. described previously in that it does not support model weaving.

A.9) The Motorola WEAVR Approach

The Motorola WEAVR approach (Cottenier *et al.*, 2007a; Cottenier *et al.*, 2007c) has been developed in an industrial setting, specifically in context of telecommunication industry. It uses Specification and Description Language (SDL) to specify models, which has partly been adopted in UML 2.x. The WEAVR approach is thus based on UML 2.x. The approach uses class diagrams and composite structure diagrams to represent structure. State machines, action language of SDL and sequence diagrams are used to model behavior. Individual aspects are represented using `<<aspect>>` stereotype and a pointcut-advice mechanism is used for composition of aspects and target models. Model execution and code generation are also supported.

A.10) The Aspect-Oriented Executable Models Notation

Aspect-Oriented Executable UML Models (AOEM) (Fuentes and Sanchez, 2007a; Fuentes and Sanchez, 2007b) is a UML profile that extends the UML and its action semantics to construct aspect-oriented executable models. In AOEM, an aspect is represented by a UML class stereotyped as `<<aspect>>`, and comprises special operations to model advices. Specifically, advices are modeled using activity diagrams without input objects and any number of output pins to modify values of intercepted objects. In (Fuentes and Sánchez, 2009), a dynamic weaving mechanism has also been provided by authors of the AOEM to enhance its models.

A.11) The Concern Architecture View Approach

Katara and Katz (2007) have provided a conceptual model for design of aspects by designing a concern architecture model. The approach can handle the specification of aspects both in symmetric as well as asymmetric manner. The conceptual model has been implemented as a UML Profile. Aspects are defined as augmentations to target model, and the composed model is obtained by mapping a non-aspect model to a new model containing aspect descriptions. The aspects are parametric in nature and thus can be bound or instantiated several times. To make them generic in this way, they are explicitly split into two parts: required (which defines the join point) and provided (which defines the augmentation to original model). To compose an aspect into a target model, a special operation called superimposition is used, which allows an aspect to augment another one.

A.12) The Behavioral Aspect Weaving Approach

The approach of Klein *et al.* (2007) is based on the concept of scenarios which are basically sequence diagrams or Message Sequence Charts. A pair of scenarios is used to define an aspect, one scenario representing the pointcut, and the other representing the advice. Just like AspectJ, the advice in this behavioral aspect weaving approach can be inserted "`around`", "`before`", or "`after`" a join point. In order to weave an aspect model into a target model, first a generic detection strategy is used which identifies all the join points in the target model, then a generic composition mechanism is applied to compose advice model with the target model at the identified join points.

A.13) Reusable Aspect Models

Reusable Aspect Models (RAM) (Klein and Kienzle, 2007; Kienzle *et al.*, 2009; Kienzle *et al.*, 2010) is a multi-view modeling approach that combines existing AO approaches to model class, sequence and state diagrams into a single approach. Multi-view modeling, in its essence, provides means to describing a system from multiple points of view, using different modeling notations, and thus allowing the use of the most appropriate modeling notation to describe facets different views of a system. RAM is different from all other AOM approaches in a sense that it views aspects as concerns that are reused many times in an application or across several applications. Therefore, this approach models any functionality that is reusable by means of an aspect. Hence, different views (i.e., structure, message, and state views) of a reusable concern are encapsulated in the form of an aspect model which is essentially a special UML package. This aspect model comprises of three different compartments representing the structural view, state view and message view. These views are expressed using a UML class diagram, state diagram and sequence diagrams respectively.

A.14) MATA notation

Modeling Aspects using a Transformation Approach (MATA) (Whittle *et al.*, 2007; Whittle and Jayaraman, 2008; Whittle *et al.*, 2009) is a graph transformation based approach to modeling and composing aspects. Both the base and aspect models are represented in the form of well-formed graphs. Since the idea of using graph rules is broadly applicable, MATA is seen as an approach which can be extended to any modeling diagrams, and even to other modeling languages. The only condition in this regard is that the modeling language to be represented using MATA must have a well-defined meta-model. UML meta-model can be represented in the form of a graph by making each meta-class a node in the type graph, and making each meta-association an edge in the type graph. In this way, any UML model can be represented as an instance of this type graph. Aspects are defined using graph transformation rules, where the left-hand-side (LHS) of a transformation rule is a pattern that defines the pointcuts which are to be matched, whereas the right-hand-side (RHS) defines new elements to be added (or removed) at these pointcuts. MATA provides a convenient way of writing the graph rules by proposing that the rule be given on one diagram only, rather than writing graph rules using both LHS and RHS, since this needs repetition of unchanged elements on both sides of the rule. To this purpose, it defines three new stereotypes namely: (1) `<<create>>` to specify that a new element must be created by the graph rule, (2) `<<delete>>` to identify deletion of an element by a graph rule, and (3) `<<context>>` to avoid effect of (1) and (2) on elements.