ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 1, Issue 7, September 2012

Extending the UML metamodel to grant prop up for crosscutting concerns

Veluru Gowri,
Assistant Professor,
Department of Computer Science and Engineering,
Sri Venkatesa Perumal College of Engineering and Technology,
Puttur, Andhrapradesh, India,

***Abstract*— Aspect-orientation is an idiom used to describe approaches that unambiguously capture, model and implement crosscutting concerns (or aspects). There is presently a quantity of new encoding languages as well as extensions to current encoding languages, the blueprint scope of mainly of which have been predisposed by the AspectJ language through three concepts and their relevant constructs, namely join points, pointcuts and counsel which can prop up two principles recognized as being key concepts of aspect-oriented programming (AOP): quantification and obliviousness. At the design level, the response of AOP has stretched been focused on the designing of AspectJ programs, and there exists no design that is nonspecific enough to incarcerate non-AspectJ aspects either as a source language during forward engineering or as a target language during reverse engineering.**

As a solution, we present an extension to the UML metamodel to clearly incarcerate crosscutting concerns. The model is sovereign from any encoding language and vague away from platform specific details. An instantiation of the newly created metamodel can be represented in usual XMI format, which enables current CASE tools to read and to envisage the occasion models in UML. This language-independent aspectual depiction can maintain design transformations critical to software development and support, such as forward engineering, reverse engineering, and reengineering.

Index Terms—UML metamodel; aspect-oriented modeling; modularity; software maintenance;

I. INTRODUCTION

Even with the success of object-orientation in the exertion to attain separation of concerns, definite properties in object oriented systems cannot be straight mapped in a one-to-one manner from the problem sphere to the solution space, and thus cannot be restricted in single modular units [4]. Their achievement ends up cutting across the decomposition of the system. Examples of crosscutting concerns (or aspects) contain perseverance, substantiation, organization and convention checking.

Aspect-Oriented Programming (AOP) [10] clearly addresses those concerns by introducing the concept of an aspect, which is a modular unit of decomposition. Presently there exist numerous approaches and technologies to support AOP. One notable technology is AspectJ [9], a general-purpose aspect-oriented language, which has prejudiced the design scope of several other general-purpose aspect-oriented languages, and has provided the area with a common terminology based on its own linguistic constructs. In the AspectJ model, an aspect description is new units of modularity provide deeds to be inserted over purposeful apparatus. This action is defined in process like blocks called advice. Nevertheless, unlike a scheme, an advice block is not at all plainly called. Instead, it is only wholly invoked by an related create called a pointcut phrase. A pointcut phrase is a predicate above well defined points in the execution of the program called join points. Even though the measurement and the level of granularity of the join point model differ from one language to another, familiar join points in existing language stipulation consist of calls to and execution of methods and constructors. When the program execution reaches a join point captured by a pointcut phrase, the connected advice block is executed. Most aspect-oriented languages provide a level of granularity which specifies unerringly when a suggestion block should be executed, such as executing before, after, or as a substitute of the code definite at the connected join point. In addition, a number of advice blocks may pertain to the same pointcut. The organize of execution can be specific by rules of advice superiority particular by the original language [11].

The Unified Modeling Language (UML) is a de facto consistent modeling language maintained by the Object Management Group (OMG) that can be deployed to symbolize object-oriented systems. The UML metamodel is a representation of UML elements together with their interrelationships. The UML Meta Object Facility (MOF) [16] is a representation of the UML metamodel and it describes a small set of concepts (such as classes and packages) that allow one to define and manipulate models of the metamodel. It enables metamodeling of UML-level metamodels (thus allowing new metamodels, and consequently new modeling languages, to be defined).

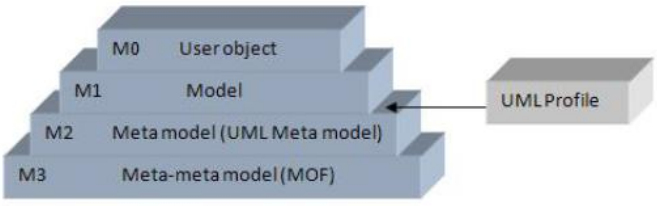

Figure 1. Meta Object Facility (MOF) 4-layered architecture.

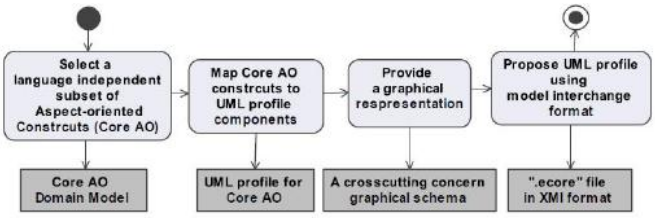

Figure 2. Methodology for creating a UML profile for crosscutting concerns

The UML is described as a 4-layered architecture shown in Figure 1. M3 is the meta-metamodel layer which is defined by MOF. M2 is the UML metamodel and it defines modeling languages (such as UML). M1 describes a user-defined UML model. All static or dynamic diagrams produced during software development or maintenance lie on this layer. Instances of the classes defined in M1 lie on M0.

In the context of software maintenance, the UML supports reverse engineering of an object-oriented program through its transformation and representation at a higher level of abstraction. With the introduction of aspects to represent crosscutting concerns, there have been in the literature a number of approaches to support aspect modeling. These approaches are classified into two different main categories [2]: 1) Deploying standard UML in a way where existing elements (e.g. classes in a class diagram) can represent aspects, or 2) Extending current UML semantics and elements to provide explicit support for crosscutting concerns. For the latter approach, there are two ways for extending UML:

Manipulating the MOF: This is a heavyweight extension mechanism, and it tends to be currently impractical due to lack of tool support.

Building a UML profile [17]: The UML profile is a subset of a UML metamodel that is defined using stereotypes, tagged values, and constraints for adapting UML meta elements to the constructs (in our case aspect-oriented constructs) of the new domain. As the UML profile does not define new elements for the UML metamodel, it can thus be considered as a lightweight extension mechanism. More specifically, UML profiles include the following:

- Stereotypes to create new model elements for a specific domain.
- Tagged values to define additional properties to model elements.
- Constraints that add rules to restrict the way the model elements operate in the context of the new domain.

An advantage of adopting a UML profile is that it allows modeling with generic UML tools. Furthermore, developers can work with well-known UML notations and concepts, reducing the need to learn new modeling languages. Additionally, it is possible to combine stereotyped and non-stereotyped elements together for proposing a complete model of a software system.

II. Motivation and Organization

Approaches in the literature to model crosscutting concerns tend to be language-specific. Our objective is to provide such modeling in a language-independent manner. This would enable model transformations during maintenance such as reverse engineering and reengineering, including the possible migration of an aspect-oriented program into a new language.

To achieve this objective, we need to specify a core model for aspectual representations, which is the topic of this paper. In particular, we present our proposal for a UML profile that models crosscutting concerns independently from any technology such as a specific programming language.

The remainder of this paper is organized as follows: In Section III we present our methodology by introducing a UML profile for modeling crosscutting concerns. In Section IV we describe two case studies to demonstrate how the proposed UML profile can be applied and how it can be used for modeling of complete AOP system. A possible application for aspect-oriented language-independent profile is proposed in Section V. In Section VI we present related work. We conclude the discussion and provide recommendations for further developments in Section VII.

III. UML Profile

The UML-based profile is built on Level 2 of the layered architecture shown in Figure1. Based on Selic[21] that describes a systematic method for defining UML profiles, our methodology for proposing a UML profile to support crosscutting concerns includes the four steps shown in Figure2:

A. *Defining a domain model*

To describe the elements of the model, we need to go back to the first principles and discuss what characteristics make a system AOP. Are there core concepts which are necessary and sufficient to qualify a system as AOP? In an article that gained high popularity, authors Filman and Friedman [4] describe two properties that are essential for AOP: Quantification and obliviousness. Quantification implies that one should be able to execute statements of the form “In program P, whenever condition C occurs, execute action A.” Obliviousness implies that components should not necessarily be built under the consideration that some aspectual behavior will be applied on them, i.e. they should maintain no visibility over aspects. As the AspectJ programming language has influenced the design dimensions of other general-purpose aspect-oriented languages, it has, in essence, dictated a collection of implementation concepts to support AOP. Along the lines of the AspectJ model, we define a domain model with the following elements: (See Figure 3).

- A set of language constructs that are core concepts in our domain. We choose aspect, join point, pointcut and advice as a subset of aspect-oriented constructs which we refer to as core AOP model. The core AOP model contains the constructs that conceptually introduce AOP [4], [6], [24].
- A set of valid relationships between the core AOP concepts.
- A set of constraints.

- A concrete syntax or graphical representation of the domain model.
- Semantics of the domain model representation.

B. *Mapping the domain model to the UML metamodel*

To map the domain model to the UML metamodel, we need to identify the most suitable UML metamodel base concepts for each element in the domain model. Each element of the domain model of Figure 3 must be individually mapped. The mapping is discussed subsequently:

Aspect: An aspect definition encapsulates static and dynamic features. Additionally much like a class, a concrete aspect can support inheritance whereas an abstract aspect can enforce inheritance2. Therefore, the UML metamodel Class is a good candidate for representing an aspect. This is because the Class metaclass is a classifier and subsequently a generalizable model element in the UML metamodel. Being a classifier it enforces the extended element to have a name and if it is inherited from a generalizable element it can be abstract or can be used as a base class for extension.

Advice: An advice block encapsulates behavior. It indicates what happens whenever the program reaches specific points during its execution. In the proposed profile, we model advice as a stereotyped behavioral feature. In the UML class diagram metamodel, there is a behavioral feature that can be either a method or an operation. An advice is never invoked explicitly and it does not have any parameters. Therefore, it cannot be modeled as a method. In addition, as discussed in [12], "a UML operation is a declaration with name and parameters" and as such it is an abstract definition without implementation. Therefore, semantically, an advice is not an operation because it is not only the declaration but it also contains the implementation. In Figure 3 we show how to extend the metaclass behavioral feature for modeling advice. Therefore, an advice is a stereotyped behavioral feature. An advice has different types, and it can be executed after, before or instead of specific events defined in pointcut expressions. Consequently we can add an enumeration type that contains different types of advice, and call it adviceType.

Pointcut and Join point: Each join point specifies a welldefined place during the execution of the program where the aspect interacts with the core functionality.

Along the lines of the work described in Fuentes et al. [7], we deploy sequence diagrams for modeling join points where messages are stereotyped as join points. Sequence diagrams provide a graphical representation for displaying join points that is simple to understand. Wildcards can be used for addressing classes and methods names. Here we have a star (*) representing any sequence of characters and a double-dot (..) representing any sequence of arguments. A pointcut is a predicate of join points. A pointcut can include a name and as a result it can be easily reused (for example, to be attached to various advice blocks which provide different behavior for it).

We therefore modeled a pointcut as a stereotyped peration. An operation has no body and it can be abstract. Semantically and conceptually it can be good choice for modeling pointcuts. This modeling is simple. Moreover, there is an advantage to modeling pointcuts and join points with definitions that are independent from advice as this decoupling can promote reuse of the former.

In the UML metamodel, BehavioralFeature is owned by Class. Due to the fact that Aspect is an extension of the Class metaclass, and Advice and Pointcut are extensions of the BehavioralFeature metaclass, Aspect already contains Advice and Pointcut because of the association that exists between the metaclasses Class and BehavioralFeature in the UML metamodel.

In other words, there is no need to explicitly create an association that relates an aspect to its advice and pointcuts. We impose two constraints during the definition of Advice and Pointcut. Only classes that are stereotyped as aspects can have behavioral features that are stereotyped as advice or pointcuts. With appropriate constraints defined in the Object Constraint Language (OCL), we confine advice and pointcuts to aspects. In Figure 3, we illustrate the UML profile for modeling crosscutting concerns. Highlighted items are added to the UML class diagram metamodel for proposing a UML profile for modeling crosscutting concerns.

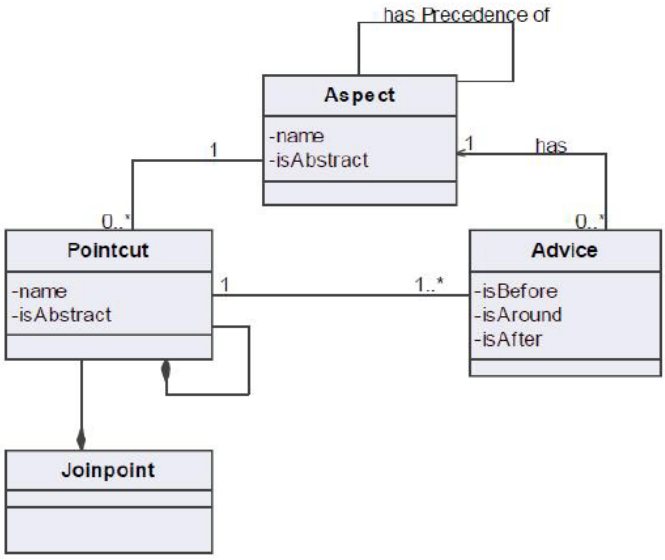

Figure 3. Core AOP domain model for crosscutting concerns.

C. *Providing a graphical representation*

Figure 4 illustrates how to model crosscutting concerns in UML tools that supports our UML profile. Using stereotyping, a new model element, Aspect, is added to the UML class diagram. Aspect has a behavioral feature that is stereotyped as Advice and also an operation that is stereotyped as Pointcut.

An advice is defined by its type (After, Before, Around) and it is attached to a pointcut. It has a name that is similar to its attached pointcut. An OCL invariant constraint is defined for the name of the advice to enforce this property. In addition, the AdviceType is shown in the definition of an aspect in this graphical representation.

A pointcut is defined by its name and the related join point's name. A join point is defined in a sequence diagram that has a name and a specific message that is stereotyped as a join point. One or more sequence diagrams displaying join points are attached to a specific pointcut.

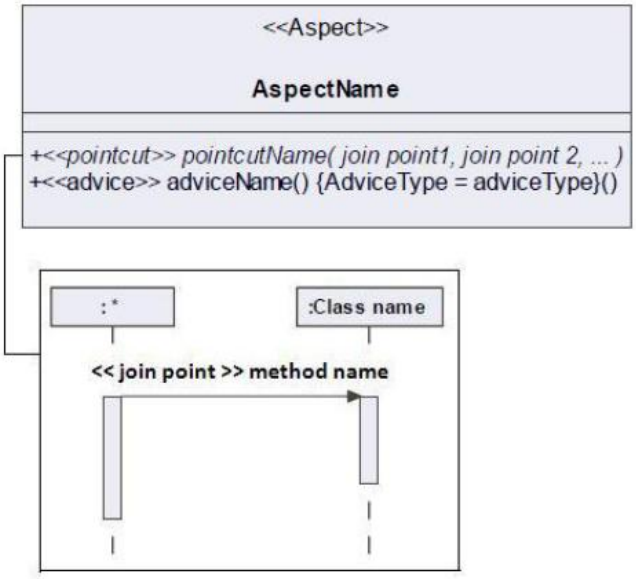

Figure 4. A graphical representation for modeling crosscutting concerns.

This representation hides unnecessary details and represents crosscutting concerns in a manner where there is no need for any additional (e.g. textual) specification. Thus the produced model can be manipulated or verified automatically by any tool that works on UML diagrams. Using this model, crosscutting concerns and their relationships with other model elements can be displayed.

D. The proposed profile using a model interchange format

To deploy the profile in available CASE tools (such as Eclipse's EMF), it is necessary to provide a persistent interchange format. St-Denis et al. [23] define a list of requirements for model interchange formats (RSF [25], RDF [13], XMI [18], etc.) and discuss their advantages and disadvantages. We have decided to adopt XMI (XML Metadata Interchange). First of all it has wide industry and tool support. Furthermore, XMI is a metamodel to describe model elements and therefore it is completely compatible with UML. Additionally, XMI uses XML syntax and therefore has all the advantages of XML.

The XMI format allows one to capture our metamodel in a specific, formal, persistent form that is required for defining model to model transformations [22]. In the EMF framework, for example, each metamodel is represented as an .ecore file in XMI format. We have created an aspect-core.ecore file that contains all the core AOP constructs, so it can be used as a metamodel for any aspect-oriented model. The XMI file will be made available from the authors online for download.

IV. CASE STUDIES

To demonstrate the applicability of our proposal we have selected two cases studies, both of which included in Eclipse AspectJ Development Tools project [1]. The first one, elecom, simulates a telecommunication system and it contains 731 ines of code. The second one, Spacewar, simulates a game and it contains 2605 lines of code.

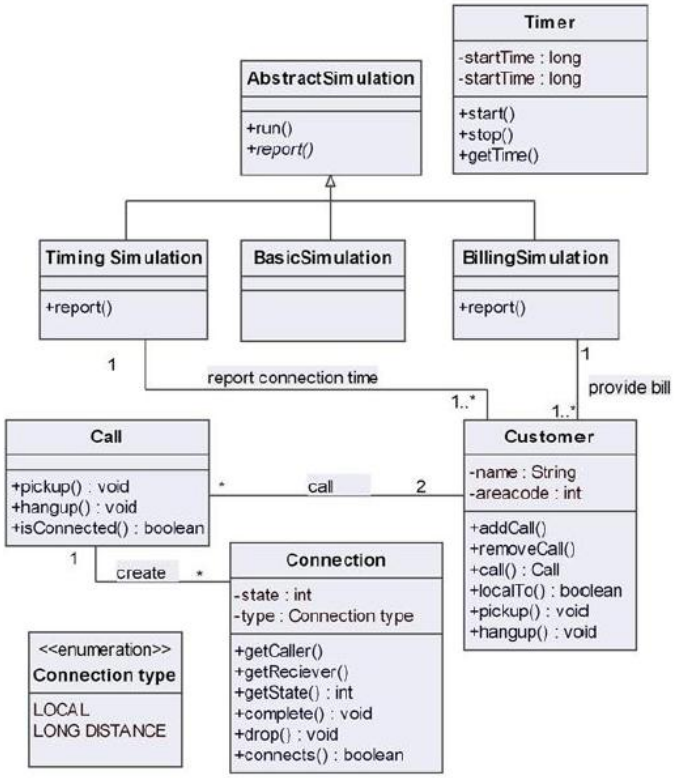

Figure 5. Telecom case study class diagram.

A. Case study: Telecom

In this system, there is a connection between two customers that can be either over long or local distance. The Telecom class diagram is shown in Figure 6. To log the activity and to provide appropriate billing for each customer, three different aspects are presented:

- Timing aspect: calculates for each customer the duration of connection and total time for each call.
- TimerLog aspect: provides a complete report of Timer's class activity. Timer class simulates a simple timer machine for calculating the elapsed time of each call.
- Billing aspect: is responsible for providing a complete call report that includes call details, duration and expenses for customers based on the timing and the type of connection.

To demonstrate the use of our profile, we show in this case study how it can represent the three Telecom aspects. As modeled in Figure 5, there are three classes which are stereotyped as aspects. If we look at the Timing aspect, it has two operations that are abstract and stereotyped as pointcuts. The name of the corresponding join points for these pointcuts are shown as operation arguments as illustrated in Figure 5. This aspect has two methods that are stereotyped as advice and their names are identical to their join point names (complete and endTiming). Additionally, the type of each advice is shown as a method property.

As the collaboration between an aspect and other components of the system is performed through join points,

we attach a UML interaction diagram to each aspect definition (See Figure 5) to illustrate this collaboration. In the timing aspect, after a call to the drop() method of class Connection, the endTiming advice executes. Furthermore, after a call to method complete() of class Connection, the complete advice executes.

V. Discussion

Several applications can be proposed for an aspect-oriented language-independent profile. First, the UML activity diagram shown in Figure 5 outlines the procedure by which the ecore file that is proposed by XMI format is the metamodel for modeling AOP constrcuts. A genmodel file is created automatically from an ecore file using EMF which in turn produces three Eclipse plugins: Edit, Editor and Core model. The Edit plugin provides a flexible layer between the model and the EMF editor while the Editor plugin provides additional model-specific UI contributions to EMF. By running the generated plugins together, the proposed metalmodel can be used for modeling any AOP project.

Second, the language-independency makes it suitable for providing proper model transformations for maintenance activities such as reverse engineering, forward engineering, language migration and reengineering. For example, the proposed UML profile provides a template for modeling crosscutting concerns and their relationship in a generic form. Therefore, this template can be forward engineered into a language specific model by adding details that are related to a specific platform (such as AspectJ or AspectC++).

Additionally, if we have a model that is based on a specific aspect-oriented programming language (e.g. the AspectJ profile discussed in [5]), we can abstract away details that are specific to the language and preserve the main concepts to accomplish reverse engineering.

In the case of reengineering to perform language migration, two transformations are necessary: 1) Reverse engineering into a language-independent model (for example, transforming AspectJ code into a class diagram), and 2) Forward engineering back into a different language dependent model (for example, into AspectC++). Language migration can be performed using the profile proposed in this paper as it supports reverse and forward engineering.

VI. Related work

In the last few years, UML profiles have been proposed for modeling crosscutting concerns. Aldawud et al. [3] argue how UML profiling would be a viable approach for modeling crosscutting concerns. Reina et al. [20] provide a survey of some other works. These were found to be either language dependent or provide no support for aspectual behavior. Other works such as [19], [5], [8] are based on the AspectJ programming language as a popular general-purpose aspect-oriented programming language.

Albunni et al. [2] propose the use of a UML activity diagram for modeling aspects in web applications. However, they do not provide a UML profile for modeling crosscutting concerns. Zhou et al. [26] model dynamic behavior of crosscutting concerns using sequence diagrams. In contrast with our approach, they do not provide a model for static behavior of crosscutting concerns. Since the dynamic behavior of crosscutting concerns affects the execution of core components, they modify existing sequence diagrams by introducing additional crosscutting bars.

A platform-independent behavioral model is proposed in [15]. In this model, advice is modeled as a stereotyped operation. Additionally, the authors follow a rather complex expression-based approach for modeling join points and it contains specific details about join points that are based on different aspect-oriented technologies. Fuentes et al. [7] model advice as a common procedure using a UML activity diagram without an input object. In this model, an advice is placed in an aspect definition as a stereotyped method. The authors provide actions for the advice that are used for retrieving data related to the join point. Our work is partially similar to [7], [8] though it differs conceptually regarding the modeling of crosscutting concerns.

Advice cannot be considered as a method or operation hence our proposed profile models advice as a stereotyped behavioral feature. Furthermore, in order to produce an executable model, they propose a weaving procedure on the model itself for combining crosscutting concerns and non-crosscutting concerns.

The idea to define a profile to support AspectJ modeling in UML was inspired by [5]. However, we eliminated all constructs specific to AspectJ programming languages and support only the most essential constructs that make a system AOP. This approach enables more kinds of reengineering and presumably makes the profile easier to explain, understand and use. Also, the profile described in this paper can easily be extended to support any additional features specific to a particular aspect-oriented language. What's more, in contrast with their profile, we dedicate a specific icon for displaying crosscutting concerns.

VII. Conclusion

Modularization of crosscutting concerns at premature stages of the software development process leads to more dependable, reusable and maintainable artifacts. In this paper we propose a language-independent UML profile for modeling both core and crosscutting concerns. The constructs in this profile are independent from any specific programming language and thus can be used to support generic aspect-oriented modeling.

Also in this paper is a graphical notation schema to display crosscutting concerns. Being UML, it is already accessible to a wide user base, yet still powerful enough to model crosscutting concerns precisely. Additionally, this proposal specified crosscutting concerns without requiring any textual specification. This, in conjunction with the decision to use the XMI format, means that it is possible to manipulate or verify a produced model using an existing UML CASE tool. Finally, due to the fact that we selected only the essential aspect-oriented language constructs, the most important characteristic of our UML extension is the language independence property,

which can be used to support reverse engineering, forward engineering, and reengineering (including migration of aspect-oriented programs from one language to another).

REFERENCES

[1] "AspectJ Development Tools," http://www.eclipse.org/ajdt/.

[2] N. Albunni and M. Petridis, "Using UML for modeling cross-cutting concerns in aspect oriented software engineering," in Proceedings of the 3rd International Conference on Information and Communication Technologies: From Theory to Applications (ICTTA), 2008.

[3] O. Aldawud, T. Elrad, and A. Bader, "UML profile for aspect-oriented software development," in Proceedings of the 3rd International Workshop on Aspect-Oriented Modeling (AOM), 2003.

[4] T. Elrad, R. E. Filman, and A. Bader, "Theme section on aspect-oriented programming," Communications of ACM, vol. 44, pp. 29–32, 2001.

[5] J. Evermann, "A meta-level specification and profile for AspectJ in UML," in Proceedings of the 10th International Workshop on Aspect-Oriented Modeling (AOM), 2007.

[6] R. E. Filman and D. P. Friedman, "Aspect-oriented programming is quantification and obliviousness," in Proceedings of the OOPSLA Workshop on Advanced Separation of Concerns, 2000.

[7] L. Fuentes and P. S´anchez, "Towards executable aspect-oriented UML models," in Proceedings of the 10th International Workshop on Aspect-Oriented Modeling (AOM), 2007.

[8] J. U. J´unior, V. V. Camargo, and C. V. F. Chavez, "UML-AOF: A profile for modeling aspect-oriented frameworks," in Proceedings of the 13th International Workshop on Aspect-Oriented Modeling (AOM), 2009.

[9] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, "An overview of AspectJ," in Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP), 2001.

[10] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M.Loingtier, and J. Irwin, "Aspect-oriented programming," in Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP), 1997.

[11] J. Kienzle, Y. Yu, and J. Xiong, "On composition and reuse of aspects," in Proceedings of the 2nd AOSD Workshop on Foundations of Aspect-Oriented Languages (FOAL), 2003.

[12] C. Larman, Applying UML and patterns: An introduction to objectoriented analysis and design and iterative development (3rd edition). Prentice Hall PTR, 2004.

[13] O. Lassila and R. R. Swick, "Resource description framework (RDF) model and syntax specification," http://www.w3.org/TR/PR-rdf-syntax, Tech. Rep., 1998.

[14] T. Mens and P. V. Gorp, "A taxonomy of model transformation,"Electronic Notes in Theoretical Computer Science, vol. 152, pp. 125–142, 2006.

[15] M. Mosconi, A. Charfi, J. Svacina, and J. Wloka, "Applying and evaluating AOM for platform independent behavioral UML models,"in Proceedings of the 7th AOSD International Workshop on Aspect-Oriented Modeling (AOM), 2008.

[16] OMG, "MOF core specification,"
http://www.omg.org/spec/MOF/2.0/,Tech. Rep.

[17] "UML 2.0 infrastructure final adopted specification,"
http://www.omg.org/cgi-bin/doc?ptc/2003-08-02,Tech. Rep.

[18] "XML metadata interchange (XMI),"
http://www.omg.org/cgibin/doc?formal/2007-12-02, Tech. Rep.

[19] A. Przybylek, "Separation of crosscutting concerns at the design level: An extension to the UML metamodel," in Proceedings of the 2nd International Multiconference on Computer Science and Information Technology (IMCSIT), 2008.

[20] A. Reina, J. Torres, and M. Toro, "Towards developing generic solutions with aspects," in Proceedings of the 5th International Workshop on Aspect Oriented Modeling (AOM), 2004.

[21] B. Selic, "A systematic approach to domain-specific language design using UML," in Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), 2007.

[22] S. Sendall and W. Kozaczynski, "Model transformation: The heart and soul of model-driven software development," IEEE Software, vol. 20,no. 5, pp. 42–45, 2003.

[23] G. St-Denis, R. Schauer, and R. K. Keller, "Selecting a model interchange format: The SPOOL case study," in Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS), 2000.

[24] F. Steimann, "The paradoxical success of aspect-oriented programming," in Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), 2006.

[25] K. Wong and H. Mller, "Rigi users manual, version 5.4.4," University of Victoria, Victoria, Canada, ftp://ftp.rigi.csc.uvic.calpub/, Tech. Rep., 1998.

[26] X.-C. Zhou, C. Liu, Y.-T. Niu, and T.-Z. Lai, "Towards a framework of aspect-oriented modeling with UML," in Proceedings of the 2008 International Symposium on Computer Science and Computational Technology (ISCSCT), 2008.15717