An Experimental Design for Evaluating the Maintainability of Aspect-Oriented Models Enhanced with Domain-Specific Constructs*

Aram Hovsepyan, Stefan Van Baelen, Riccardo Scandariato, Wouter Joosen
DistriNet, Katholieke Universiteit Leuven
Celestijnenlaan 200A
BE-3001 Leuven, Belgium
{first.last}@cs.kuleuven.be

Serge Demeyer
Universiteit Antwerpen (UA)
Department of Mathematics and Computer Science
Middelheimlaan 1,
BE-2020 ANTWERPEN
{first.last}@ua.ac.be

ABSTRACT

Abstraction, modularity and composability are considered to be the fundamental properties behind aspect-oriented software development and aspect-oriented modeling (AOM) in particular. The same properties are expected to be supported through the use of domain-specific modeling languages (DSMLs). However, little research is done to investigate the symbiosis between the two paradigms. In this position paper we firstly present the key challenges for the successful integration of DSMLs with an AOM approach. Furthermore, we elaborate in detail on the question whether leveraging on aspect-oriented programming languages offers benefits over the use of traditional model composition. We propose an experimental approach to evaluate these alternatives.

1. INTRODUCTION

Modularization and levels of *abstraction* are key software engineering concepts that can help one to master the increased complexity of software applications [22]. In this context, aspect-oriented software development and aspect-oriented modeling (AOM) in particular is an important and promising step towards a new dimension in the modularization and separation of concerns [28]. Also, domain-specific modeling languages (DSML) research community is trying to stress the importance of DSMLs in reducing and bridging the *abstraction* gap between the problem and solution domains [16]. However, to our best knowledge, little research is done to investigate the synergy between these two disciplines [14, 29].

AOM is a model-driven engineering (MDE) approach that supports the definition and use of concept-specific viewpoints (also referred to as concerns/aspects/modules/etc.) and focuses on providing support for separation of concerns at higher levels of abstraction [10, 26]. Each viewpoint describes how a concern is addressed in a design, hence, each concern can be developed individually without overwhelming the developers with irrelevant details of other modules. Moreover, in theory, each concern could be developed by the most suitable team of experts. AOM approaches typically provide support for composing the individual viewpoints to obtain an integrated design view.

In order to specify each concern in the most optimal manner, we promote to combine AOM and DSMLs [14]. This could be done by offering the developer the choice of using a DSML or a general-purpose modeling language (GPML) in case no suitable DSML exists. Most of the existing AOM approaches (e.g., [2, 12, 13, 24]) use UML and its extension mechanisms for expressing all concerns. However, given the expressiveness and the raised abstraction that the DSMLs provide [16, 19], one could potentially improve the specification of concern models in current AOM methods when the developers can use an optimal DSML for each of the concerns involved.

In this position paper we further discuss the synergy between DSMLs and AOM, and outline the key challenges that come with this combination. We elaborate further on one of the challenges, namely given a modularized design along with the composition specification that outline the correspondences between the different concerns, where the actual composition should take place. The first option is

using a model-to-model transformation and performing a **model composition**. The second alternative is transforming the design models to an aspect-oriented platform, where the low-level aspect weaver performs the composition at the byte-code level, i.e., **code weaving**.

The paper is structured as follows. In section 2, we present our position on the synergy between DSMLs and AOM and illustrate that the fundamental properties of both paradigms are similar and complementary. Furthermore, we present two alternative development process flows, namely, model composition of the modularized concerns versus model-to-code transformation where code weaving is performed by an aspect weaver. In section 3, we propose an empirical evaluation method to investigate and compare the two alternative process flows. Section 4, describes the potential challenges of the proposed method. Finally, we conclude and sketch on our future work.

2. THE SYNERGY BETWEEN DSMLS AND ASPECTS

Concerns are an important motivation for organizing and decomposing software into manageable and comprehensible parts [21]. We use the term concern to uniformly refer to what aspect-oriented software development practitioners often call an aspect concern and a base concern [17]. The base concern typically represents the functional backbone of a given application, whereas different aspect concerns represent functional and non-functional modules that augment the core. We treat **abstraction**, **modularity** and **composability** as the fundamental properties that define aspect-orientation rather than quantification and obliviousness [9, 23].

2.1 Domain-Specific Models are Aspects

DSMLs are often small languages providing a notation that is very close to the problem domain and quite intuitive for the domain experts. This is why DSMLs are said to raise the **abstraction** level of the language [16]. A DSML provides a language to describe a view of the system, focusing on the elements that are relevant to that particular view. A single DSML is rarely sufficient to describe all the system views. This is why we believe that **modularity** is a fundamental property of the DSML paradigm as one would typically need a number of DSMLs to describe each view of a complex system. Finally, given the modular and abstract system viewpoints, one would eventually need to specify the correspondences between the different viewpoints, i.e., specify their **composition** [5, 14, 29]. Thus, a systematic approach that allows the specification of a complex system using a number of viewpoints expressed in DSMLs is by definition an AOM approach. On the other hand, an AOM approach may benefit from the abstraction raise that DSMLs provide by specifying certain concerns in a dedicated DSML.

We believe there are three key issues for the successful integration of DSMLs and AOM.

2.1.1 DSML Availability

Although DSMLs for various domains have started to emerge (e.g., XACML for the specification of access control policies [20], WSLA for service-level agreements [15]), the number of standardized DSMLs (or at least recognized by a community) is limited. The development of a DSML requires highly specialized skills and may prove to be very costly [19]. In addition, GPMLs (e.g., UML) remain very suitable for the specification of certain system concerns, such as most functionality related issues. Hence, ideally, an integrated AOM-DSML approach should allow the developer to select either a suitable DSML for a given concern or to fall back on the default setting of using a GPML.

2.1.2 Composition Specification

Even if suitable DSMLs are available, the matter of specifying the composition of concerns expressed in different modeling languages is extremely challenging [29]. There is a growing number of approaches that address the problem of DSML composition (e.g., [4, 7, 31]), however, they require that the input DSMLs belong to the same technical space, i.e., share a common metamodel [18]. In [18] Kurtev et al. stress the importance and the need for concern specification and composition across technical spaces. So far, a limited number of approaches have addressed the problem of bridging technical spaces [14, 18].

2.1.3 Composition Output

Finally, even if all the building blocks for the specification and composition of the heterogeneous domain-specific concerns are present, it is still unclear what would be the next step in the development process. The traditional AOM approaches (e.g., [2, 24]) typically implicitly suggest performing the composition at the modeling level and obtaining an integrated system view. However, the output formalism of the model composition step involving different modeling languages (i.e., a mix of a GPML and DSMLs) is not obvious. Firstly, if the input DSMLs do not share a common metamodel, having an integrated output modeling language may be impossible (e.g., consider combining an XSD-based and an EMF-based modeling language). Moreover, even if the input modeling languages share a common metamotel and it would be possible to combine them, it could become too complex to be usable by developers [29].

A possible solutions is the homogeneous transformation of all DSML specifics matters to a GPML or a general-purpose programming language. This option would require a transformation bridge to the selected GPML for each DSML. An alternative approach is keeping the domain specific concerns as they are and introducing DSML interpreters written in a general purpose programming language (e.g., Java). Concerns that are specified in a GPML are transformed into the same general purpose programming language in which the interpreters are implemented.

However, in both cases it still remains unclear where the actual composition should take place. One of the possible choices is to perform the composition at the modeling level (or just before code generation inside the model-to-code transformation tool) and produce code that fully composes all concerns. On the other hand, one could also preserve the modularization by leveraging on aspect-oriented implementation platforms, thereby leaving the actual composition to be performed by the aspect weaver. Clearly, this choice is not neutral. Rather, it has a direct impact on several dimensions, such as productivity of developers, quality of the final product, ease of maintenance of the artifacts, and several other matters. In the remainder of this paper we focus on this issue in the context of system maintainability and we propose a detailed empirical evaluation strategy to compare
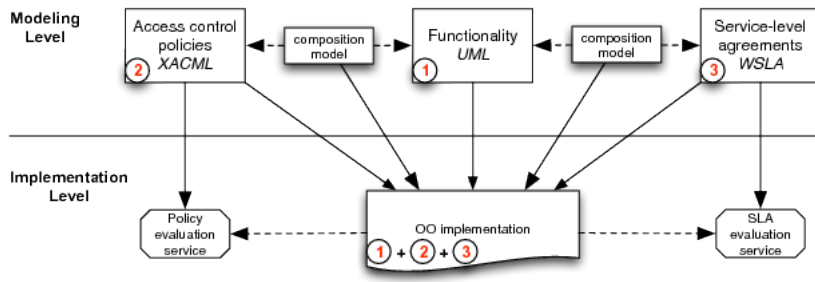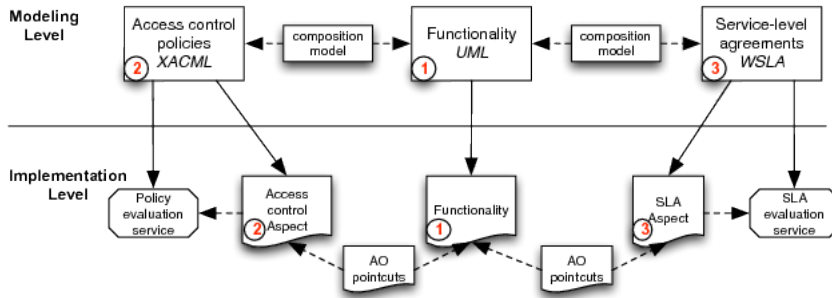
Figure 1: Model composition

Figure 2: Code weaving

the two alternatives.

2.2 AOM-DSML approach

In order to illustrate the two alternatives we will build up on an approach that allows the combination of core system structure and functionality (expressed in UML) with modularized access control policies and service-level agreements expressed in a domain-specific language [14]. We have selected to use XACML, which is a de facto standard domain-specific language for expressing access control policies [20]. Furthermore, we have chosen WSLA for the specification of service-level agreements [15]. WSLA is used by many projects, in particular in academia. Moreover, WSLA is the predecessor of WS-Agreement that is currently the new standard initiative in the domain of service-level agreements.

The top part of figure 1 illustrates schematically how the concerns can be modularized and modeled. The base structure and functionality (1) is specified using UML class diagrams. The access control policies (2) are specified in XACML. Analogously, WSLA language is used to specify service-level agreements (3) involving certain services that are to be offered by the application. In order to specify the composition between the three concerns, we follow the approach presented in [14]. This AOM-DSML approach requires the creation of the so-called **concern interfaces** for both the access control concern and the SLA concern. Each concern interface specifies the information that is required by the concern from the base application. Finally, two composition models specify the correspondences between the viewpoints (functionality, access control and SLA viewpoint).

2.3 Model Composition

The first alternative is to merge the concerns at the modeling level by performing model composition and producing object-oriented source code. Even though model composi-

tion does not necessarily imply the use of object-oriented programming languages, it is the closest match in terms of abstractions used. A number of existing AOM approaches imply that this option is followed (e.g., see [3, 13, 24]). The main advantage of this approach is that the composition complexities are not propagated to the code so that the developer has to face it only at the modeling level.

2.4 Code Weaving

The second process alternative (depicted in figure 2) starts from the same aspectual model. This time, however, the concerns are kept modularized all the way down to the source code. A model-to-code transformation tool is used to transform each modularized concern to the code level. The composition models are reflected at the code level by a set of aspect-oriented pointcuts . The composition itself is performed by the aspect code weaver that produces a composed byte-level code (not shown on the figure). Several AOM approaches follow this alternative (e.g., see [2, 12]).

Note that both alternatives start from the same specification at the modeling level. In addition, the presented AOM-DSML approach only allows for a semi-automatic code generation.

So far we have described the need for AOM approaches to support the selective use of a GPML or a more suitable DSML for specifying concerns. Given that such DSMLs do not always share a common metamodel we are interested what is the best way to implement the composition logic. The first alternative is performing the composition at the modeling level (or during the code generation) whereas the second option is leveraging on an AOP weaver by generating suitable pointcuts and advice. In the next section we present a roadmap of investigating this problem using empirical evaluation techniques.

3. EMPIRICAL EVALUATION

In order to evaluate the two process flow alternatives presented in the previous section, namely model composition and code weaving, we propose to conduct an empirical investigation that focuses on the problem statement in a certain perspective, i.e. maintainability. In a typical software life-cycle, it is the maintenance that brings the highest cost in terms of overall effort and time spent.

Ideally, any empirical evaluation should be performed using a variety of different techniques that build up the "weight of evidence" in support of a certain hypothesis. We propose two quantitative methods and a qualitative method as a blueprint for our experiment.

3.1 Quantitative 1: Internal Metrics

The first track of the quantitative investigation involves a static analysis of the final systems obtained by the alternative approaches. The analysis focuses on determining the internal quality attributes of the systems (e.g., size, complexity, modularity) by collecting a number of metrics well-known in the literature (e.g., lines of code, coupling, cohesion, scattering, tangling) [6, 25].

3.1.1 Motivation

The main motivation for using this technique lies in its simplicity. Given the availability of tools that can automatically perform a static analysis of a system, it costs virtually no effort to collect and compare these metrics. In addition, this evaluation technique is widely used in the literature (e.g., [8, 11]).

3.1.2 Drawbacks

Unfortunately, up till now there are no universally accepted and crisp models for predicting externally observable system qualities (e.g., maintainability) based on the internal quality attributes. Even though certain papers succeed in providing such predictability models (e.g., [11]), they are typically limited in their scope and can rarely be reused in other studies. In general, the link between the external quality attributes and the internal metrics has only been hypothesized so far, but never proven. In addition, given the presence of models and partially automatic generation of the code, the internal source code quality has even smaller impact on the external quality attributes. Hence, the results of the static system analysis will generate an initial intuition, but are unlikely to provide a solid evidence.

3.2 Quantitative 2: User Study

The second quantitative investigation requires a carefully designed process that provides measurements for the externally observable attributes of the system (e.g., maintainability, understandability). Such process is important as it can be used as checklist and guideline of what to do and how to do it. Figure 3 illustrates the process that we will use in designing the user experiment [30].

Definition. The first step is definition of the experiment where a hypothesis has to be clearly defined. The study of the internal quality attributes from section 3.1 typically provides an initial idea which of the two alternatives under investigation is likely to be better. Hence, it is possible to state the null hypothesis along with the hypothesis. The main intention of the exper-
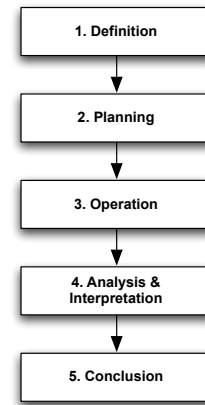

Figure 3: Process [30]

iment is to try to refute the null hypothesis in favor of the hypothesis.

Planning. During this step we will determine the context of the experiment and the overall experiment design including the measurement scales. It is essential to provide background information on experiment subjects. Furthermore, instrumentation should be discussed as part of the design. Finally, at this stage it is important to consider the validity of the results.

Operation. This activity consists of preparation, execution and data validation. At the preparation step, one must prepare the subjects as well as the materials and instruments needed. The actual execution typically does not raise any crucial issues. Finally, one must ensure that the collected data is correct and provide a valid picture of the experiment.

Analysis & Interpretation. The data collected during the experiment operation is the input to this activity. Typically the data is first analyzed using descriptive statistics, which allows one to understand and interpret the data informally. Then using associative statistics one could reflect on the null hypothesis and reject or accept it.

Conclusion Finally, given the results of the previous step, one can reflect on the experiment goals and build a conclusion.

3.3 Qualitative: User Interviews

Virtually any software engineering issue is best investigated using both quantitative and qualitative methods. Qualitative methods can actually help the researcher explain the quantitative results. The two most common means for qualitative data collection are interviews and questionnaires [1].

We plan on using the structured interviewing technique to gain some insights into the results. A structured interview follows a fixed list of carefully selected questions. Questions can be either closed or open. Closed questions are similar to the questionnaires where the participant has to chose between a list of possible answers. Open questions allow for greater interaction and can help one to obtain less expected responses from the participants.

4. CHALLENGES

Similar to any empirical experiment there are a number of threats to the validity of the user study as described in section 3.2. In this section we outline three of the most important challenges concerning the validity of the experiment.

4.1 Selection of the Maintenance Tasks

The maintenance tasks selected for the experiment should be as realistic and representative as possible. However, the potential number of such tasks for any given system is virtually endless. We believe that it is essential to select a mix of tasks based on the idea of a cause and effect relationship. As an experiment designer one has certain beliefs about the relationship between a cause construct and an effect construct. For instance:

- we expect that the *code weaving* treatment would require less effort for a broad class of tasks that affect the crosscutting concerns;

- we expect that there would be no differences in the treatments that affect non-crosscutting concerns;

- we expect that the *model composition* treatment provides a more explicit overview of the program execution flow and would result in less errors in certain situations involving crosscutting concerns where the *code weaving* treatment suffers from the fragile point-cut problem [27].

Based on these expectations we will formulate a number of hypotheses and select tasks that would tests them.

Note that many real-life maintenance tasks require days or even weeks before their completion. Unfortunately, it is not feasible to have such tasks in our experiment, hence, we will have to design tasks of relatively short duration. Given this constraint it is also likely that the selected tasks will also have a rather local impact on the system. Tasks that require some global refactoring will typically last substantially longer.

4.2 Selection of the Participants

Provided the mix of the technologies and expertise required for the use of AOM-DSML approach outlined in section 2, it is obvious that the selection of the participants is crucial. Ideally, participants should be experts in UML, Java, AspectJ, XACML and WSLA. However, it is highly unlikely that we will be able to find such participants. Hence, it is essential to minimize the risk of obtaining results that will not be valid in case the experiment is repeated with experts. There are a number of different techniques that could help minimize this threat.

Blocking is a technique that is used to systematically eliminate the undesired effect (e.g., learning factor for a certain technology) in comparison among the treatments. Within one block, the undesired effect is the same and one could study the effect of the treatments on that block. More specifically, we will block the subjects into pairs where within each pair we will try to match the two participants as closely as possible with respect to their skills. This will lower the effect of the difference in expertise amongst the subjects. Moreover, we will require the subjects within each pair to perform half of the maintenance tasks using one treatment and the rest using the other treatment. Both subjects will do all the maintenance tasks in the same order but using opposite treatments. We will use statistical analysis methods that perform a pair-wise comparison (e.g., Wilcoxon signed-rank test) in order to further increase the precision of the experiment.

Each maintenance task requires both modeling and coding effort. Given that the modeling effort in both alternative treatments is the same, in theory, we could split the two efforts and discard the modeling effort. We could go even further by actually providing the modifications required at the modeling level. This will lower the impact of having participants that are novices in modeling technologies.

4.3 Investigating the Cause and Effect Relationship

Although as experiment designers have an idea of the cause and effect relationships, sometimes unforeseen, but interesting factors could arise and play a crucial role. In order to investigate the cause and effect relationship in greater detail we will opt for individually supervised experiment. With this setup the supervisor gets the opportunity to observe the course of the experiment and note any interesting actions or events. The individual observations could rise new hypotheses. While in the long-term the experiment could be repeated with the new hypotheses, the qualitative study (see section 3.3) could be used as a short-term technique to confirm or refute the new hypotheses.

Experiment calibration is another technique that we could use in order to explore the cause and effect relationship before performing the user study. During a calibration step, a pair of participants is asked to perform the selected maintenance tasks with the sole intention of monitoring the course of the experiment.

5. CONCLUSION

In this position paper we have presented the importance and the advantages of investigating a symbiosis between aspect-oriented modeling (AOM) and domain-specific modeling languages (DSML). Given the limited scope of the typical DSMLs, each domain-specific model can be considered being an aspect as it is modular and focused around a certain concern in the system. Moreover, a single DSML is rarely sufficient on its own to allow one to describe the whole system. Hence, there is a need for the composition operation whenever DSMLs are used. Precisely the three properties of modularity, abstraction and composition are defined as fundamental for aspect-oriented software development. Furthermore, given an integrated AOM approach that allows the use of various DSMLs from different technical spaces for concern specification, we have proposed an empirical research roadmap in order to evaluate whether keeping the concerns modularized in the implementation offers benefits compared to the concern composition at the modeling level. In the near future we will perform a thorough empirical investigation using the presented strategy. We are extremely eager to discuss this approach at the AOM @ MoDELS'10 workshop.

6. REFERENCES

[1] E. Babbie. Survey research methods. 1990.
[2] E. Baniassad and S. Clarke. *Aspect-Oriented Analysis and Design: The Theme Approach*. Addison-Wesley, 2005.

[3] O. Barais, J. Klein, B. Baudry, A. Jackson, and S. Clarke. Composing multi-view aspect models. In *Proceedings of the 7th International Conference on Composition-Based Software Systems*, pages 43–52. IEEE Computer Society, 2008.

[4] J. Bézivin, S. Bouzitouna, M. D. D. Fabro, M.-P. Gervais, F. Jouault, D. S. Kolovos, I. Kurtev, and R. F. Paige. A canonical scheme for model composition. In *ECMDA-FA*, pages 346–360, 2006.

[5] J. Bézivin and I. Kurtev. Model-based technology integration with the technical space concept. In *Metainformatics Symposium 2005*, Esbjerg, Denmark, 2005.

[6] S. Chidamber and C. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.

[7] M. Emerson and J. Sztipanovits. Techniques for metamodel composition. In *OOPSLA – 6th Workshop on Domain Specific Modeling*, pages 123–139, 2006.

[8] E. Figueiredo, N. Cacho, C. Sant'Anna, M. Monteiro, U. Kulesza, A. Garcia, S. Soares, F. Ferrari, S. Khan, F. C. Filho, and F. Dantas. Evolving software product lines with aspects: an empirical study on design stability. In *Proceedings of the 30th International Conference on Software Engineering*, pages 261–270. ACM, 2008.

[9] R. Filman and D. Friedman. Aspect-oriented programming is quantification and obliviousness. In *Workshop on Advanced Separation of Concerns, OOPSLA 2000, Minneapolis.*, 2000.

[10] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In *Proceedings of the 29th International Conference on Software Engineering*, pages 37–54. IEEE Computer Society, 2007.

[11] P. Greenwood, T. Bartolomei, E. Figueiredo, A. Garcia, N. Cacho, C. Sant'Anna, P. Borba, U. Kulesza, and A. Rashid. On the impact of aspectual decompositions on design stability: An empirical study. In *Proceedings of the 21st European Conference on Object-Oriented Programming*, pages 176–200. Springer-Verlag, 2007.

[12] S. Hanenberg, D. Stein, and R. Unland. From aspect-oriented design to aspect-oriented programs: tool-supported translation of JPDDs into code. In *Proceedings of the 6th International Conference on AOSD*, pages 49–62. ACM, 2007.

[13] A. Hovsepyan, S. Van Baelen, Y. Berbers, and W. Joosen. Generic reusable concern compositions. In *Proceedings of the 4th European Conference on Model Driven Architecture Foundations and Applications*, pages 231–245. Springer, 2008.

[14] A. Hovsepyan, S. Van Baelen, Y. Berbers, and W. Joosen. Specifying and composing concerns expressed in domain-specific modeling languages. In *47th International Conference Objects, Models, Components, Patterns*, pages 116–135. Springer, June 2009.

[15] A. Keller, E. Keller, and H. Ludwig. Defining and monitoring service level agreements for dynamic e-business. In *in Proceedings of the 16th USENIX System Administration Conference (LISA)*. The

USENIX Association, 2002.

[16] S. Kelly and J.-P. Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation.* Wiley-IEEE Computer Society Press, 2008.

[17] G. Kiczales, J. Irwin, J. Lamping, J.-M. Loingtier, C. V. Lopes, C. Maeda, and A. Mendhekar. Aspect-oriented programming, 1997.

[18] I. Kurtev, J. Bézivin, and M. Aksit. Technological spaces: An initial appraisal. In *CoopIS, DOA'2002 Federated Conferences, Industrial track*, 2002.

[19] M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344, 2005.

[20] OASIS. Core specification: Extensible access control markup language (XACML) v2.0. www.oasis-open.org/commitees/xacml.

[21] H. Ossher and P. Tarr. Multi-Dimensional Separation of Concerns and The Hyperspace Approach. In *Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development*, 2000.

[22] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, 1972.

[23] A. Rashid and A. Moreira. Domain models are not aspect free. In *Proceedings of the 9th International Conference On Model Driven Engineering Languages And Systems*, pages 155–169. Springer, 2006.

[24] Y. R. Reddy, S. Ghosh, R. B. France, G. Straw, J. M. Bieman, N. McEachen, E. Song, and G. Georg. Directives for composing aspect-oriented design class models. *Transactions on Aspect-Oriented Software Development*, pages 75–105, 2006.

[25] C. Sant'anna, A. Garcia, C. Chavez, C. Lucena, and A. von Staa. On the reuse and maintenance of aspect-oriented software: An assessment framework. In *Proceedings of the 17th Brazilian Symposium on Software Engineering*, 2003.

[26] A. Schauerhuber, W. Schwinger, E. Kapsammer, W. Retschitzegger, M. Wimmer, and G. Kappel. A survey on aspect-oriented modeling approaches. Technical report, 2006.

[27] K. Sullivan, W. G. Griswold, Y. Song, Y. Cai, M. Shonle, N. Tewari, and H. Rajan. Information hiding interfaces for aspect-oriented design. In *Proceedings of the 10th European Software Engineering Conference*, pages 166–175. ACM, 2005.

[28] P. L. Tarr, H. Ossher, W. H. Harrison, and S. M. S. Jr. N degrees of separation: Multi-dimensional separation of concerns. In *Proceedings of the 21st International Conference on Software Engineering*, pages 107–119, 1999.

[29] A. Vallecillo. On the combination of domain specific modeling languages. In *ECMFA*, pages 305–320, 2010.

[30] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering An Introduction.* Kluwer Academic Publishers, 2000.

[31] A. Zito, Z. Diskin, and J. Dingel. Package merge in uml 2: Practice vs. theory? In *MoDELS*, pages 185–199, 2006.