

A Marriage of MDD and Early Aspects in Software Product Line Development

Thais Batista

SETE — Software Engineering Team
DIMAp — Computer Science Department
Federal University of Rio Grande do Norte
thais@ufrnet.br

Sérgio Soares

SPG — Software Productivity Group
Department of Computing and Systems
University of Pernambuco
sergio@dsc.upe.br

María Cecilia Bastarrica

MaTE Lab
Computer Science Department
University of Chile
cecilia@dcc.uchile.cl

Lyrene Fernandes

Computer Science Department
State University of Rio Grande do Norte
lyrenefernandes@uern.br

Abstract

Model-driven development (MDD) shifts the development focus from code to models, allowing automatic or assisted transformations that are able to generate more refined, detailed or complete models. In Software Product Line (SPL) development, variation point implementation might inevitable lead to (crosscutting) concerns that are tangled and spread with other concerns, suggesting the use of aspect-oriented (AO) approaches. In this context, Early Aspects techniques can be applied to identify crosscutting concerns at early development process stages, positively affecting the application models as soon as possible. In this work we propose a marriage of MDD and early aspects for the development of SPL. Our approach takes an AO feature model and automatically transforms it into an AO early design model and this model into an AO architecture specification. We applied our proposal to a real-life SPL to better exemplify its use.

1. Introduction

Model-driven development (MDD) [14] is a new paradigm for developing software where the focus is shifted from code to models. Therefore, models are not exclusively documentation artifacts anymore; instead they are analysis artifacts and the input for automatic or assisted transformations that are able to generate more refined, detailed or complete models, and in particular the software code.

In software product line (SPL) development, the product line requirements usually captured in the form of a fea-

ture model, the product line architecture, and each product architecture, can be considered different models of the system family, and as such it is reasonable to try to apply MDD techniques in their generation. There have already been some attempts to use MDD as a strategy for SPL development [4].

In software development in general, some requirements, when implemented, will derive crosscutting concerns [18], i.e., concerns that are tangled and spread with other concerns. In SPL this kind of concerns usually is much more critical, since they can be part of a variation point implementation, inevitable affecting a whole series of products. This situation requires this concern to be plugged either in or out from the SPL' products.

Such a situation suggests the use of aspect-oriented programming (AOP) [18] with SPL development [1, 2]. The use of aspects to implement some kinds of product variations allows these variations to be easily added or removed from a product configuration, without polluting the code with conditional compilation code that hinders program legibility leading to maintainability' issues [1, 2].

In this context, the sooner these aspects can be identified the better because they can be incorporated as part of early models, therefore influencing the SPL and each particular product architecture upfront, instead of demanding changes only during design or implementation stages. Early aspects is a field of aspect-oriented software development (AOSD) that attempts to identify crosscutting concerns in early stages of the development process, and there are several techniques for doing so.

In the past we have developed MaRiSA (Mapping Requirements to Software Architecture) [9] where a model of a system' requirements including early aspects specified us-

ing AOV-Graph [23], an aspect-oriented intentional model, is automatically transformed into a software architecture specified using AspectualACME [8], an aspect-oriented architectural description language (ADL). This architecture is then transformed into a detailed design.

In this work we intend to extend this previous work for SPL proposing MaRiPLA (Mapping Requirements to Product Line Architecture). In order to be consistent with the common practice in SPL, requirements will be defined with a plain feature model that is then transformed to what we have called *early design*, a stage between requirements and architecture where early aspects are identified. Finally, the early design is transformed into a product line architecture. All transformations both in MaRiSA and MaRiPLA are developed using ATL (*Atlas Transformation Language*) [15]. The languages for defining both, the early design and the product line architecture are extensions of AOV-Graph and AspectualACME, respectively, in order to manage variability.

We applied the proposed methodology to the development of a meshing tool software product line. Approaching this domain as a SPL is a new field, since to the best of our knowledge, nobody has applied early aspects in their development even though its usefulness may be apparent.

The paper is structured as follows. Section 2 includes a discussion of related work about MaRiSA, feature models, the adoption of aspects in SPL development, AOV-Graph, and AspectualACME. Section 3 introduces our running example and motivates the use of the methodology. The description of the methodology together with its application to the running example is presented in Section 4. Finally, Section 5 presents some final remarks and further work.

2. Related Work

In this section we address work related to the use of aspects with features and with software product lines, aspect-oriented requirements engineering, and architecture description languages. All these issues will be considered to generate our model-driven development approach for generating architecture specifications from the requirements models.

2.1. MaRiSA

MaRiSA (Mapping Requirements into Software Architecture) [9] defines a MDD-based transformation process between AOV-Graph and AspectualACME models. It also offers a tool that implements such a process; it receives as input an AOV-Graph specification and automatically transforms it into a corresponding AspectualACME model. MaRiSA does not support product family specification as its languages do not offer elements to represent variabilities. In

Section 4 we propose MaRiPLA that extends AOV-Graph and AspectualACME to support the representation of product lines and also extending its associated tool by inheriting the existing transformation rules and defining new rules related to the extensions for product lines specification.

2.2. Features Model with Aspects

A feature is a system property that is relevant to some stakeholder. Features are organized in feature diagrams. A feature diagram is a tree with the root representing a concept and its descendent nodes are features. Feature models are feature diagrams plus additional information such as feature descriptions, binding times, priorities, or stakeholders, among others.

Feature modeling is a key approach to capturing and managing common and variable features in a software product line [11]. They are used during early stages of SPL development for scoping the system family, later as a basis for building the product line architecture, and finally during the application engineering for guiding the requirement elicitation and analysis. Feature models were proposed as part of the Feature-Oriented Analysis method (FODA) [16], and since then a series of extensions have been proposed.

In [11] there is a series of distinctive types of features identified:

- Concrete features such as data storage or functions that may be realized as individual components.
- Aspectual features that may affect several components and can be modularized using aspect technology.
- Abstract features such as performance requirements that are usually mapped to a configuration of components and/or aspects.
- Grouping features may represent variation points and they are mapped to a common interface of plug-compatible components.

Kulesza et al. [19] extended the feature model by introducing a new type of relation between features, called crosscuts relation, to support the representation of cross-cutting features in a generative approach to the context of families of multi-agent systems. In this approach a feature A crosscuts a feature B, when either A or one of its subfeatures depends on and inspects B or one of the subfeatures of B.

In this work we identify concrete and aspectual features in order to generate the early design. We also include cross-cutting relations as part of the feature model. Considering abstract features for designing the product line architecture remains part of our future work.

2.3. Product Lines and Aspects

Some works address the use of aspect-oriented techniques with software product lines. Alves et al. [1, 2] have worked on refactoring variations implemented with conditional compilation to use aspects instead. A tool was developed to support such refactorings [10]. Another work [20] deals with a model-based generative approach to map features to aspects in order to enable the derivation of an aspect-oriented software family architecture. The work defines an extended generative model that allows feature modeling representing crosscutting relationships, mapping rules and guidelines for product derivation, and an architecture model. An evaluation on how suitable some aspect-oriented techniques are for software product lines [3] had concluded that each technique should provide a set of guidelines and criteria that support developers in applying the techniques in a systematic and unified way.

In our work, we aim to provide support for the automatic generation of the product line architecture from the requirements and early design models: feature and PL-AOV-Graph.

2.4. Aspect-oriented requirements engineering AORE

A survey on aspect-oriented requirements engineering approaches was conducted [17] and has compared several models. The author has concluded that in general there is no formalism for conflict resolution techniques, and neither for adding new requirements. Models do not deal with real time constraints. Most models provide a systematic procedure for separation between crosscutting and core concerns. This shows that the models have similar aims and constraints.

AOV-Graph [23] is an aspect-oriented intentional model, represented by AND/OR decomposition graphs. Its relationships map not just positive and negative conflicts between requirements (goals, softgoals and tasks), they map how these requirements crosscut each other and they also represent choices of different options of how a given requirement may be achieved, being a reasonable choice to represent variability [5].

In this work we extend AOV-Graph to consider issues of the software product lines domain, such as variability, in the SPL requirement model.

2.5. Architecture description languages ADLs

Mae [13], Koala [24] and x.ADL 2.0 [12] are architecture description languages that support the modeling of product family architectures. They share the idea that a

product family architecture is a common architecture (composed by components and connectors) extended with some well-defined variation points. Such variation points are represented, in Mae, by a variant component. In Koala each component that contains variability has a special kind of interface (called *diversity interface*) through which selections are performed to choose the variant for a specific instance. To capture the evolution of a product family architecture Mae uses a version management approach while Koala defines an external configuration management system. xADL 2.0 defines three elements to represent variability: variants, options and versions. Each variant is associated with a boolean condition (a guard condition) that is evaluated when the system is executed and a concrete element is chosen according to the satisfied condition. xADL 2.0 also offers a versioning schema to represent a full version graph of any component, connector, or interface type.

ADLARS [6] is an ADL that considers a feature model as a predecessor and represents architectures as a collection of tasks that follows a task template. This template defines mandatory, optional and alternative features and components. A component template contains a set of possible component configurations and directly relates them to features of the feature model.

AspectualACME [8] is a general-purpose ADL that employs the following constructs: *components* and *connectors* (elements of computation and interaction), *ports* and *roles* (interfaces of components and connectors respectively), *attachments* (associations between ports and roles), *representations* (internal decompositions of components/ connectors), *properties* (annotations on other Acme constructs), *systems* (configurations of components and connectors), and *families* (architectural styles). Crosscutting concerns are modeled using components and a special kind of connector – an *aspectual connector* (AC) – is used for representing crosscutting interactions. The AC interface makes a distinction between the elements playing different roles in a crosscutting interaction i.e. affected base components and aspectual components. The AC interface contains: (i) a *base role*, (ii) a *crosscutting role*, and (iii) a *glue* clause. The *base role* may be connected to the port of a component (provided or required) and the *crosscutting role* may be connected to a port of an aspectual component. The composition is expressed by the *glue* clause. It specifies the details about the composition between components and aspectual components, such as the place where the port from an aspectual component will affect the regular component.

In this work we extend AspectualACME to represent variable features and to provide a selection mechanism that chooses the variant for a specific architecture.

3. Running Example

Mesheres are used for numerical modeling, visualizing and/or simulating objects or phenomena. A mesh is a discretization of certain domain geometry. This discretization can be either composed by a unique type of element, such as triangles, tetrahedra or hexahedra, or a combination of different types of elements. There exist several meshing tools that generate and manage these discretizations [21].

Meshering tools are inherently sophisticated software due to the complexity of the concepts involved, the large number of interacting elements they manage, and the application domains where they are used. Their complexity mainly relies on the components involved as is the case for most scientific computing software. Provided that meshing tools are used in a variety of different application domains, they may require slightly different functionalities.

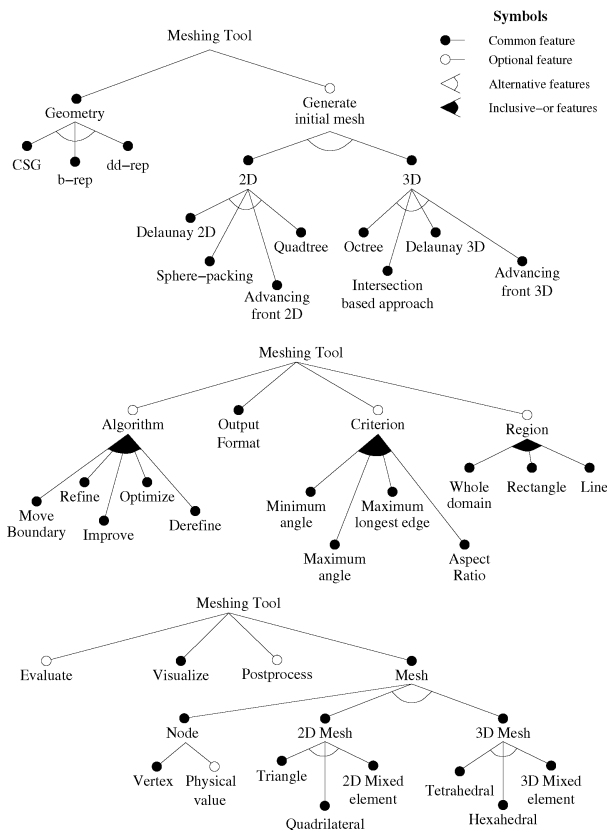


Figure 1. Feature model for the Meshing Tool SPL [22].

As these tools have usually been developed with ad hoc methodologies and without taking reuse as a goal, every new tool usually needs to be developed from scratch even though it may involve algorithms already implemented and

data structures already designed all of them also used and tested. Meshing tools have a good opportunity for reuse, but a clear reuse framework is required if the expected gains in productivity and quality are to be achieved [7].

In Figure 1 we show a complete feature model for the meshing tool SPL. There are some common features such as Geometry, Mesh or Output Format, others that are optional such as Evaluate or Postprocess, and others where a series of alternative features may be chosen such as Algorithm, Criterion or Region. Most of these features are concrete according to the taxonomy in [11]: Refine, Optimize, and Visualize are functions, while Mesh is a data storage. There are some other features that are aspectual features such as Criterion and Region. These features crosscut the execution of any Algorithm chosen to be part of a particular product of the SPL.

For example, the purpose of the Refine algorithm is to modify the Mesh so that it contains smaller elements that better describe the geometry, as shown in Figure 2 where certain geometry is refined twice. In this case the refinement is applied to the whole geometry, i.e., all elements in the whole mesh are refined. However, when the mesh already contains a large number of elements, this algorithm may be too time consuming. In this case it may be better to constrain the region where the refinement would be applied. A region may be defined as the intersection of the mesh and certain Rectangle or Line. Defining the region as an aspect allows us to decide whether to include this concept or not, and thus we may be able to restrict the area where the Refine algorithm is applied without invading the refinement code.

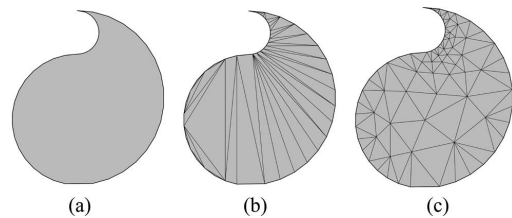


Figure 2. Successively refined mesh.

4. MaRiPLA

MaRiPLA (Mapping Requirements to Product Line Architecture) is a software development process framework that defines three layers that support early aspect-oriented product line specification ranging from requirements to architecture description, as illustrated in Figure 3.

The mapping between two adjacent layers is defined through ATL transformation rules that transform models according to the metamodels specification. MaRiPLA defines

a set of rules to transform the Feature model to PL-AOV-Graph and another set of rules to transform PL-AOV-Graph to PL-AspectualACME.

4.1. Feature Model

SPL requirement specification will be approached using regular feature models for usability reasons. These models will be annotated with two extra pieces of information: an attribute for each identified feature indicating if it is concrete or aspectual. In the latter case it should also be indicated which other feature would be crosscut by the aspectual feature.

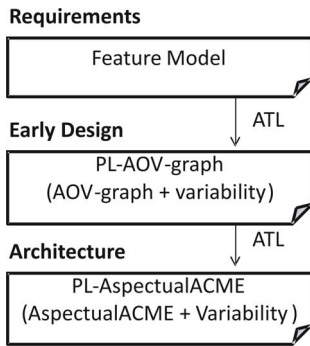


Figure 3. MaRiPLa's transformation phases.

4.2. Early Design

In order to represent variability we propose PL-AOV-Graph, an AOV-Graph extension. AOV-Graph uses AND/OR relations: AND relations represent mandatory requirements and OR relations represent optional requirements.

Figure 4 illustrates, in AOV-Graph, some tasks of a Meshing tool: Change geometry and Select [region] are optional tasks while Model [geometry] is a mandatory task. Cross relationship between Select [region] and Change [geometry] means that whenever one of the Region subfeatures is selected as part of a feature configuration, it will affect the execution of the tasks in Change [geometry].

This representation is not enough to easily represent a set of alternative features and neither a set of inclusive-or features, such as it can be shown in a feature model such as the one in Figure 1.

Therefore, we identify the need for developing PL-AOV-Graph. AOV-Graph models can be extended without violating its foundations and making the model harder, in two ways: (i) by creating a new attribute in goals, softgoals and tasks to represent with which other elements that element is combined; or (ii) by using the property attribute of goals,

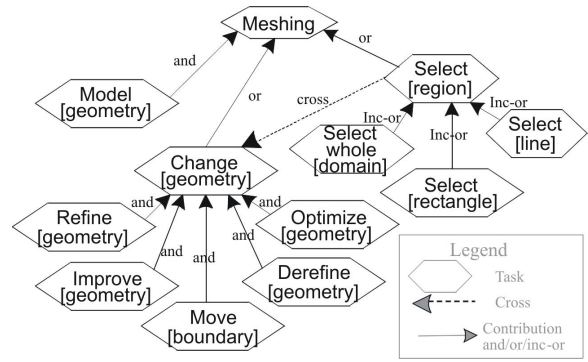


Figure 4. AOV-Graph for the Meshing Tool SPL.

softgoals and tasks to represent this information. Both, using a new attribute and using a property attribute, we can graphically represent alternative and inclusive-or tasks using arcs between relationships such as those shown in Figure 1.

4.3. Product Line Architecture

In this work we identify the need of developing PL-AspectualACME that extends AspectualACME in order to represent SPL architectures. Such architecture is a normal architecture with aspects and variation points. In order to represent variability, PL-AspectualACME employs the idea of representation that already exists in AspectualACME. Using the representation construct it is possible to define multiple representations for a given element (optional and alternative features). This means that each representation exhibits a product variation. Thus, the commonalities of a product line are modeled as components and each representation of a component characterizes the possibility of instantiating a specific product. We extend AspectualACME by defining special types of ports (Select-Alternative, Select-Inclusive-Or, Select-Optional), that we call select ports. These ports use conditional statements to select the proper representation (variation) for a given product.

Figure 5 contains a piece of PL-AspectualACME description of the Meshing Tool SPL. It specifies the Geometry component, the Algorithm component and the Region component. This last component contains three representations (Rectangle, Line, and WholeDomain).

The Select – Inclusive – or port implements the selection mechanism (omitted for the sake of brevity). The AC aspectual connector defines the crosscutting relationship where the glue clause specifies that the element attached to the RegionRole acts before the element at

```

System Meshing = {
  Component Geometry = { Port Model; }
  Component Algorithm = {
    Port Change; ... }
  Component Region = {
    Port Select-Inclusive-or;
    Representation Rectangle = {...}
    Representation Line = {...}
    Representation WholeDomain = {...}
}
AspectualConnector AC = {
  baseRole AlgorRole;
  crosscuttingRole RegionRole;
  glue RegionRole before AlgorRole;
}
Attachments
  Algorithm.Change to AC.AlgorRole;
  AC.RegionRole to Region.Select-Inclusive-or;

```

Figure 5. AspectualACME description for the Meshing Tool SPL.

tached to the AlgorRole. The attachments part specifies that the Change port of the Algorithm component is attached to the AlgorRole of the aspectual connector and the RegionRole of the aspectual connector is attached to the Select – Inclusive – or port of the Region component. This means that the Select – Inclusive – or port will select a representation of the Region component before the execution of the change service of the Algorithm component.

4.4. Model Transformations

Each feature in the feature model will be transformed into a task in PL-AOVgraph. Common features are connected with and edges, while optional features are connected with or edges. Whenever there is an inclusive-or in the feature model (feature group), all the subfeatures will be transformed to tasks connected with the inc – or edge. Figure 6 includes part of the transformation from the feature model to PL-AOVgraph implemented in ATL.

Regarding to the transformation from PL-AOVgraph and PL-AspectualACME, tasks are transformed into Components. Whenever there is an inc – or edge connecting subtasks to a task, each subtask is transformed into a Representation of the Component and it is inserted a Select – Inclusive – Or port in the Component. We omit the transformation rule for the sake of brevity.

5. Final Remarks and Future Work

MDD and AOSD are new software engineering paradigms that have emerged as interesting alternatives to improve the development process of complex systems. In

```

rule Aspectual2Task {
  from
    af : FeatureModel!Aspectual
  to
    t : SP-AOVgraph!Task(id <- af-name,
      contribution_label <-
        if !oclIsUndefined(af.lower) then
          if af.lower=0 then "or"
          else "and" endif
        else "" endif,
      component <- af.getMembers()),
    cr : SP-AOVgraph!CrosscuttingRel(
      source <- t,
      pointcut <- Set { pc },
    pc : SP-AOVgraph!Pointcut(joinpoint <- jp),
    jp : SP-AOVgraph!Joinpoint(component <- af.cross)
}

```

Figure 6. Transformation from feature to PL-AOVgraph.

this paper we argued that the marriage of these two approaches can bring benefits to the development of software product lines. An additional level of separation of concerns can be achieved with this marriage as AOSD supports the separation of crosscutting concerns in different activities of the software lifecycle and MDD supports the systematic integration of these activities through model transformation rules. In order to make this marriage concrete, in this paper we proposed MaRiPLA, a process framework that integrates requirements (feature model), early design (PL-AOV-Graph) and architecture (PL-AspectualACME) activities by establishing associated metamodels and a set of ATL rules to allow the automatic transformation between them, and forward and backward traceability.

As part of the envisioned future work we intend to implement the extensions of PL-AOV-Graph and PL-AspectualACME. We also aim to conclude the implementation and to test MaRiPLA in a broader domain.

Our experience with MaRiSA showed that the automatic transformation between the models of different software lifecycle activities guarantees that the early decisions are propagated to subsequent activities within the development process. We also observed that an automatic generation of the architecture from the requirements can result in an incomplete architectural model because some architectural decisions are not represented in the requirements model. We have worked with the idea that there is an information baseline, and it has to be repeated (and mapped to) in all models created during the development process; therefore, architects do not have to write this common information from scratch, they can add or change a partial structure that can be obtained from the requirements. Thus, the intervention of the architecture is needed in order to improve the generated architecture with new architectural elements.

Acknowledgment

We thank CNPq for funding the Latin-American Aspect-Oriented Software Development network (LA-AOSD). We also thank Andrés Vignaga for implementing the transformations.

References

- [1] V. Alves et al. From conditional compilation to aspects: A case study in software product lines migration. In *1st Workshop on Aspect-Oriented Product Line Engineering at GPCE'06*, October 2006.
- [2] V. Alves et al. Extracting and evolving code in product lines with aspect-oriented programming. In *Transactions on Aspect-Oriented Software Development (TAOSD): Special Issue on Software Evolution*, 2007.
- [3] M. Anastasopoulos and D. Muthig. An evaluation of aspect-oriented programming as a product line implementation technology. In *International Conference on Software Reuse (ICSR'04)*, pages 141–156, July 2004.
- [4] O. Ávila García et al. Using software product lines to manage model families in model-driven engineering. In *Symposium of Applied Computing (SAC07)*, pages 1006–1011, 2007.
- [5] B. Baixauli et al. Using goal-models to analyze variability. In *First International Workshop on Variability Modelling of Software-intensive Systems*, 2007.
- [6] R. Bashroush et al. Adlars: An architecture description language for software product lines. In *Annual IEE Software Engineering Workshop (SEW05)*, 2005.
- [7] M. C. Bastarrica and N. Hitschfeld-Kahler. Designing a product family of meshing tools. *Advances in Engineering Software*, 37(1):1–10, January 2006.
- [8] T. Batista et al. Reflections on architectural connection: Seven issues on aspects and adls. In *Early Aspects (EA) at ICSE'06*, pages 3–9, May 2006.
- [9] T. Batista et al. On a bi-directional mapping between aspect-oriented requirements and architectural descriptions. *IET Software Journal*, 2008. submitted.
- [10] P. Borba et al. Flip product line derivation tool. In *Forum Demonstration at AOSD'08*, April 2008.
- [11] K. Czarnecki et al. Staged configuration using feature models. In *Third Software Product Line Conference (SPLC'04)*, pages 266–283. LNCS 3154, 2004.
- [12] E. Dashofy and A. van der Hoek. Representing product family architectures in an extensible architecture description language. In *4th International Workshop on Product Family Engineering*, page 330341. LNCS 2290, 2001.
- [13] A. V. der Hoek et al. Taming architectural evolution. In *Joint 8th European Software Engineering Conference*, 2001.
- [14] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In *Future of Software Engineering (FOSE'07)*, pages 37–54. IEEE Computer Society, 2007.
- [15] F. Jouault et al. Atl: a model transformation tool. *Science of Computer Programming Special Issue: Experimental Software and Toolkits*, 2008.
- [16] K. C. Kang et al. Feature-oriented domain analysis feasibility study. Technical Report CMU/SEI-90-TR-21, ADA235785, Software Engineering Institute, 1990.
- [17] S. Khan et al. A survey on early separation of concerns. In *12th Asia-Pacific Software Engineering Conference (APSEC'05)*. IEEE, 2005.
- [18] G. Kiczales et al. Aspect-oriented programming. In *European Conference on ObjectOriented Programming (ECOOP'97)*, pages 220–242. Springer, 1997.
- [19] U. Kulesza et al. Integrating generative and aspect-oriented technologies. In *Brazilian Conference on Software Engineering (SBES'04)*, October 2004.
- [20] U. Kulesza et al. Mapping features to aspects: A model-based generative approach. In *10th Internacional Workshop on Early Aspects at AOSD'07*, March 2007.
- [21] S. J. Owen. <http://www.andrew.cmu.edu/user/sowen/mesh.html>, 2007.
- [22] P. Rossel et al. Supporting reuse in meshing tool development using domain analysis. Technical Report TR/DCC2008-11, Universidad de Chile, July 2008.
- [23] L. F. Silva et al. On the symbiosis of aspect-oriented requirements and architectural descriptions. In *10th International Workshop on Early Aspects at AOSD'07*, pages 75–93. LNCS 4765, Springer, 2007.
- [24] A. van der Hoek. Capturing product line architectures. In *Fourth International Software Architecture Workshop*, pages 95–98, 2000.