Variability Management in Aspect-Oriented Architecture Description Languages: An Integrated Approach

Eiji Adachi, Thaís Batista, Uirá Kulesza, Ana Luisa Medeiros
Computer Science Department
UFRN
Natal-RN, Brazil
eijiadachi@gmail.com, thais@ufrnet.br, uirakulesza@gmail.com, analuisafdm@gmail.com

Christina Chavez
Computer Science Department
UFBA
Salvador-BA, Brazil
flach@ufba.br

Alessandro Garcia
Informatics Department
Opus Research Group, PUC-Rio
Rio de Janeiro-RJ, Brazil
afgarcia@inf.puc-rio.br

Abstract—In this paper we propose an integrated approach for managing variabilities in architectural specifications of software product lines. Our approach combines an aspect-oriented architectural description language for product lines, PL-AspectualACME, with a variability modeling language, VML4Arch. PL-AspectualACME, also proposed in this paper, is used to specify the overall architectural description consisting of the modular representation of architectural commonalities and variabilities. VML4Arch is used to automatically derive product-specific architecture descriptions by specifying features dependencies and the variabilities that must be included in the products. We illustrate the application of our approach by using a software product line from the mobile domain.

Keywords—Architecture Description Languages; Software Product Line; ACME; AspectualACME;

I. INTRODUCTION

Software product line (SPL) engineering [1] supports the systematic development of a family of related products that share some commonalities and contains variabilities that distinguish specific products. Typically, the common and variable features are captured and modeled into a feature model[2][3]. However, commonalities and variabilities may be considered in all activities of the software lifecycle in order to narrow the gap between the feature model and the different models and code artifacts that represents the SPL requirements, architecture, design and implementation.

In fact, there is a strong demand for approaches that help the variability management along the different software development activities. This is mainly due to the large scale and high complexity of SPLs, which usually involve dealing with a large amount of features and different configurations of products. The definition of approaches that provide support to variability management at different development stages contributes to address these challenges in the following way: (i) it allows analyzing the nature and dependencies of the SPL commonalities and variabilities at a higher level of abstraction; and (ii) it facilitates the creation of traceability links between abstractions from the different artifacts developed to the SPL. These traceability links can then be used to navigate along the implementation of artifacts considering specific features, and to analyse the change impact when evolving the SPL.

Over the last years, some approaches have been proposed aiming to address the variability management at the modeling level [4][5][6][7]. These approaches propose, in general, the definition of the composition between a variability model (typically represented as a feature model) and other models specifying the SPL requirements or architecture. At the architecture level there are some initiatives [8][9][10][11] in terms of architecture description language (ADL) for SPL specification. However, most of them focus on documenting the commonalities and variabilities with limited support for managing variabilities. In addition, to the best of our knowledge, few ADLs for SPL specification address the modular representation of the architectural concerns using architectural aspects. In fact, there is a few research work [12] that applies aspect-oriented software development to improve modularity in the context of SPL development but as just incipient proposals at the ADL level.

In this paper, we propose PL-AspectualACME, a seamless extension of a general-purpose aspect-oriented ADL, AspectualACME [13]. AspectualACME is an aspect-oriented extension of ACME [14], a general purpose ADL proposed as an architectural interchange language. PL-AspectualACME
promotes a natural blending of software product line and aspect-oriented architectural abstractions. Instead of burdening the ADL with new abstractions to express product line specification, PL-AspectualACME adapts existing AspectualACME and ACME abstractions. The main aim of PL-AspectualACME is to allow the modular representation of architectural commonalities and variabilities, as well as their composition, specifying the overall architectural description. PL-AspectualACME does not aim to specify details of specific products derived from the overall architectural description. We consider that embedding this information at the architectural level will produce a very complex architectural description overloaded by product-specific information, which is difficult to understand and to evolve. In addition, for each new product, the overall architecture will be changed in order to express the specific product configuration.

In order to overcome these challenges, we decided to follow a well-known strategy [1] to the development of software product line that separates the product-line development processes in two stages: (i) the first process specifies the commonalities, the variabilities, and the overall configuration of the product line in the domain design stage of domain engineering; (ii) the second process specifies the derivation of a product-specific architectural description in application engineering from the SPL architectural specification.

In domain engineering for the first process, we adopt PL-AspectualACME. In order to decide which variability has to be bound to realize a product-specific architecture description, the variability modeling language (VML4Arch) is used [7] to establish the links between the feature model and software architecture models by providing primitives to describe product-specific configurations - the set of features that must be included and the features dependencies. By combining PL-AspectualACME and VML4Arch we guarantee that both product line architecture and product line instantiation are
handled in a modular way and, as a consequence, both the overall architecture and the product-specific architectural description can be easily changed and evolved. In order to illustrate our approach we apply it to MobileMedia [15], a software product line application for media management on mobile devices.

This paper is structured as follows. Section II introduces the case study used through the paper and the basic concepts of ACME, AspectualACME and VML4Arch. Section III presents PL-AspectualACME, the architectural description of MobileMedia in PL-AspectualACME, and also a discussion about the strategy proposed by PL-AspectualACME. Section IV describes the integration of PL-AspectualACME and VML4Arch and illustrates the MobileMedia specification with VML4Arch. Section V compares our proposal with related work. Finally, Section VI contains the final remarks.

II. BACKGROUND

A. Mobile Media

MobileMedia (MM) is a software mobile product line that provides support to manage (create, delete, visualize, play, send) different kinds of media (photo, music) on mobile devices. It extends an existing software product line, MobilePhoto, by including mandatory, optional and alternative features. There are different MobileMedia implementations in Java, AspectJ and CaesarJ. Each implementation has 10 releases, in which each release contains additional functionalities with respect to its predecessor release. Releases 1 to 5 deal only with photos and release 6 includes new features: store, play, and organise music. The management of photo (e.g. create, delete and labelling) was turned into an alternative feature.

Figure 1 illustrates the MM Feature Model of release 6. This model contains mandatory features representing commonalities such as AlbumManager, CreateAlbum, Search Album, MediaManager, DeleteAlbum, AddMedia, ExcludeMedia, LabelMedia, SearchMedia, DisplayMediaList, PlayMedia, and DisplayMedia. The optional features are: LinkMediawithPhoneBookEntry, DisplayPhotoofIncomingCaller, PlayMelodyofIncomingCaller, DisplayFavouriteMediaList, SetAsFavouriteMedia, CountNumberofTimesaMediahasbeenViewed, SortMediabyHighestViewingFrequency, CopyMedia, SearchMediaforLabel, SendMediaviaE-mail, and SendMediaviaSMS. Finally, the alternative features are: (i) the media types: photo, music; and (ii) the communication service: SMS, e-mail and photo viewer.

B. ACME and AspectualACME

ACME is an ADL that supports the definition of: (i) an architectural structure, that is, the organization of a system into its constituent parts; (ii) properties of interest, information about a system or its parts that allow one to reason abstractly about overall behavior, both functional and non-functional; (iii) types and styles, defining classes and families of architecture; and (iv) constraints, guidelines for how the architecture can change over time [14]. Architectural structure is described in ACME by Components, Connectors, Systems, Attachments, Ports, Roles, and Representations. Components are composite computational encapsulations that support multiple interfaces known as ports. Ports are bound to ports on other components using first-class intermediaries called Connectors, which support so-called Roles that attach directly to ports. Systems are the abstractions that represent configurations of components and connectors. A system includes a set of components, a set of connectors, and a set of attachments that describe the topology of the system. Attachments define a set of port/role associations. Representations are alternative decompositions of a given element (component, connector, port or role) to describe it in greater detail. Thus, the representation may be seen as a more refined depiction of an element. For instance, ports may have a representation to encapsulate a large set of API calls as a single port. Inside the representation, a set of ports is used to represent individual API calls. A binding defines an association between a port on a component with some port within the representation. Properties of interest are <name, type, value> triples that can be attached to any of the above ACME elements as annotations, except attachments. Properties are a mechanism for annotating designs and design elements with detailed, generally nonstructural, information.

The architectural elements outlined above are sufficient for defining the structure of software architecture as a graph of components and connectors. However, they do not provide the adequate means to describe the composition mechanisms required for aspect-oriented composition. In order to address this issue, AspectualACME [13] extends ACME to support the modular specification of architectural aspects. It basically defines two extensions: (i) a special kind of architectural connector, the Aspectual Connector, that encapsulates aspect-component interaction, and (ii) a quantification mechanism that simplifies, syntactically, the reference of a set of join points in an architectural description. The basic elements of AspectualACME are components, connectors and attachments. The attachments section (configuration) defines a set of port/role associations. Thus, it defines the place where the structural join points are identified. The quantification mechanism is used in the configuration through wildcards (*) that represent part of the component name or ports.

The aspectual connector defines an interface that distinguishes the different elements that participate in a crosscutting interaction (Base and Crosscutting roles) and expresses the way that these elements interact (Glue clause). A base role represents one or more ports of components affected by a crosscutting concern; a crosscutting role represents a port of some aspectual component; and a glue clause specifies the way a service of an aspectual component affects one or
more services of regular components. There are three types of aspectual glue: after, before, and around.

Figure 2 contains a textual specification of the Aspectual Connector in the context of the MobileMedia SPL, including the ServiceCommunication crosscutting concern. The ServiceCommunication component (lines 02-06) provides three services of communication with external softwares through the ports PhotoViewerSoftware (line 03), SMSSoftware (line 04) and EmailSoftware (line 05). The MediaManager component (lines 08-12) provides two services to manage medias through the ports DisplayPhotoOfIncomingCaller (line 09) and DisplayMedia (line 10). The SendMedia (lines 13-16) component provides two services for sending medias through the ports SendMediaViaSMS (line 14) and SendMediaViaEmail (line 15). The aspectual connector ServiceCommunicationCross (lines 17-21) abstracts the crosscutting relationship among ServiceCommunication, MediaManager and SendMedia components. Its glue clause (line 20) specifies that the port attached to the crosscutting role acts around the port attached to the base role, which means that each invocation to the port attached to the base role will be intercepted to be realized by the port attached to the crosscutting role. The attachment section (lines 22-34) bounds the ServiceCommunication to the components MediaManager and SendMedia through the aspectual connector ServiceCommunicationCross. The quantification mechanism (*) is used within the attachments to specify more concisely that the crosscutting concern affects all ports ending with the suffix “Email” (line 24) and “SMS” (line 26).

Figure 3 illustrates the graphical notation of the aspectual composition of ServiceCommunication, MediaManager and SendMedia components through the aspectual connector ServiceCommunicationCross. The pair base role - crosscutting role does not impose the constraint, common in others ADLs, of only connecting input ports (provided ports) to output ports (required ports). The aspectual connector connects the SMSSoftware provided port of the ServiceCommunication to the SendMediaViaSMS provided port of the SendMedia component, for instance.

C. VML4Arch

The Variability Modelling Language 4 Architecture - VML4Arch - [7] aims at expressing how variabilities are realised by elements of product line architectural models. It establishes the links between features models and software architectural models, specifying which operations must be executed when features are selected or unselected. In order to achieve this, VML4Arch provides a set of primitives for referencing elements and invoking actions, which result in architectural transformations. The main elements in the language are concerns, variation points and variants. Concerns are high-level abstractions encapsulating variation points, which relate to a particular feature of the system or any other architectural concern. A variation point identifies a particular concept within a concern as being variable. It has a name and a kind (optional, inclusive-or, alternative or parameter). A variant describes a specific variability decision.

VML4Arch provides a basic set of actions for activating decisions which will result in architectural compositions between architectural variants and the common core elements. The key actions provided are: add, remove, connect, deploy and merge. The language also provides the reference primitive, which associates architectural elements with a name in order to simplify expressions when used within actions.
The language also permits specifying dependencies between variabilities. Dependencies are expressed using requires and excludes identifiers with parameters pointing the name of the corresponding variant or variation point.

III. ARCHITECTURAL DESCRIPTION FOR MOBILE MEDIA WITH PL-ASPECTUALACME

In this section, we present a brief overview of PL-AspectualACME. Initially, we present its main concepts (Section III-A), and then we apply it to specify the Mobile Media architectural description (Section III-B). Finally, we discuss the challenges faced by our group while specifying the MM architecture with PL-AspectualACME (Section III-C).

A. PL-AspectualACME Overview

PL-AspectualACME extends AspectualACME in order to provide mechanisms to represent variability in software product line architectures. The idea is to use the existing abstractions and extend AspectualACME in a lightweight fashion. PL-AspectualACME does not add any new architectural element; it just enriches the semantics of some existing elements. Variabilities are represented in PL-AspectualACME using the representation construct. Originally designed in ACME as a mechanism for hierarchical decomposition of components and connectors into subsystems, the representation construct is used in PL-AspectualACME to modularize architectural variabilities that are related to specific product variation. Hence, the commonalities of a product line are modeled as components / ports and each representation element of a component / port identifies the possibility of instantiating a specific product.

PL-AspectualACME also extends the concept of port, defining a new port type – the Select port. It is used as a kind of special port responsible for selecting the proper variation for a given product. Three new conceptual ports are introduced: Select-Alternative, Select-Inclusive-Or and Select-Optional. The Select-Alternative port implements the selection mechanism for alternative features. It selects exactly one representation from the set of available representations. The Select-Inclusive-Or port defines the selection mechanism for inclusive-or features, which selects one or more representations from the set of available representations. Finally, the Select-Optional port specifies the selection mechanism for optional features, i.e., it specifies if a given representation is selected or not. Annotations within the element Property are also used to explicitly categorize at the architectural level the features into mandatory, optional, alternative or inclusive-or features. Table I summarizes the main concepts used in PL-AspectualACME.

Figure 4 presents an example of variability description in PL-AspectualACME. The Media component (line 1) abstracts the variation point related to the types of media supported by the MM. It provides two inclusive-or variants, which are abstracted by the representation elements: the first, models the Photo variant (lines 3-10); the latter, models the Music variant (lines 11-18). The Select-Inclusive-Or port (line 2) implements the selection mechanism of the variants. It is connected to the inner ports of the Photo and Music variants at the Bindings sections (lines 8-10 and 16-18). Table II links the architectural elements used in PL-AspectualACME to describe variabilities in SPL and their respective semantics in the context of each feature category.

B. Mobile Media specification

This subsection presents the specification of MM architecture in PL-AspectualACME. The specification relates to the release 6. All the commonalities and variabilities of this release were presented in the feature model of Section II-A. In the interest of brevity, components previously explained will not be detailed again in this section.

In the partial specification depicted in Figure 5, properties are used to categorize the features into mandatory, optional and inclusive-or. The AlbumManager component (line 4) models three mandatory features (lines 6-8) and has MediaManager as a sub-system. All the services provided by the MediaManager sub-system are exposed through the MediaManagerFacade port (line 8). The MediaManager component (line 12) models the core features of MM (lines 13-15) and also the optional features. Each optional feature is modeled as one distinct representation element (lines 18-44). The Select-Optional port (line 18) abstracts the selection mechanism of the optional features. Thus, it is connected to

<table>
<thead>
<tr>
<th>Concepts</th>
<th>Related Semantic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Representation</td>
<td>Represents product variation at the architectural level</td>
</tr>
<tr>
<td>Select-Alternative Port</td>
<td>Implements the selection mechanism for alternative features</td>
</tr>
<tr>
<td>Select-Inclusive-Or Port</td>
<td>Implements the selection mechanism for inclusive-or features</td>
</tr>
<tr>
<td>Select-Optional Port</td>
<td>Implements the selection mechanism for optional features</td>
</tr>
</tbody>
</table>

Figure 4. Example of PL-AspectualACME textual description

Table I

MAIN CONCEPTS OF PL-ASPECTUALACME
the inner ports of each variant within their Binding sections.

There is a crosscutting relationship between Media and MediaManager components, which is modeled by the MediaCross aspectual connector (line 52). The glue clause (line 55) of the MediaCross connector defines that the element attached to the chooseMedia role (line 54) must act before the element attached to the operation role (line 53). In the attachment section (lines 58-61), MediaCross attaches all ports from MediaManager (and also from its sub-systems) containing the word “Media” to the operation role (line 59); it also attaches the chooseMedia role to the Select-Inclusive-Or port of the Media component (line 63). In other words, the Select-Inclusive-Or port must choose one or more of the Media variants (Music and Photo) before MediaManager performs its service. Figure 6 illustrates the graphical description of a partial MM architecture.

C. Discussion

The seamless extensions proposed by PL-AspectualACME enabled it to represent variability in software product line architectures and avoid that software architects be burdened with a large amount of new ADL abstractions. The combination of Representation to identify product variation and Select ports to implement the selection mechanism has made the language very expressive, without losing simplicity. However, there are important issues, especially those related to the specification of feature dependencies and the selection mechanism of variabilities during product derivation, which we consider that PL-AspectualACME should not address. Both feature dependencies and the variabilities selection would be specified in a separated model in order to guarantee a clear separation between problem and solution space.

Traditionally, a feature model [2][3] is used in SPL development to model the commonalities and variabilities. It represents the features on a tree-shaped diagram, in which the root node represents the concept being modeled and the other nodes represent its features. The hierarchical organization describes the node concept from general to specific concepts. The parent-child relationship between the nodes defines an implicit dependency relationship: the child node is present in a given description instance if and only if its parent is included. We say that the child node requires its parent node presence. There is also an implicit dependency relationship among features when we use alternative-features: the features in one alternative features set are mutual exclusive among them. We say that one alternative feature excludes the other alternative features. Dependencies like requires and excludes are likely to occur with features that do not share a parent-child relationship or are not alternative features. They are used to specify constraints between features. During a process of instantiation of a product (also called product derivation), application engineers select which features will be part of the product desired, respecting the conditions and constraints defined in the feature model. There are some existing product derivation tools that already automate this process [16][17]. Recently, new languages and mechanisms have been explored to address the instantiation of product-
specific requirements or architectural models. Although a similar approach based on feature model would be used to automatically generate product-specific AspectualACME specifications, we decide to explore a new approach, based on a Variability Modeling Language (VML) [7]. Next section details our integrated approach.

The select ports are used to represent at the architecture level the mechanism responsible for choosing the variabilities included in a given instance of a product. They provide a mechanism to determine which features can be selected, but they do not specify how and under what circumstances those features will be selected. They lack an explicit support to address the selection of variabilities considering a specific product. This support is also addressed by VML in the approach presented in the next section.

The original proposal of ACME and AspectualACME do not provide any specific model for describing system behavior or its functional properties. Instead, an annotation mechanism within Property elements is proposed to express this sort of information. Following the idea of a seamless extension of AspectualACME and also the fact that information about specific instances of a product line should be inherited from the feature model or another similar language/mechanism, PL-AspectualACME maintains ACME’s original proposal and no new abstraction are added to describe features dependencies and the protocol for the selection mechanism.

IV. AUTOMATIC DERIVATION OF PL-ASPECTUALACME SPECIFICATIONS

This section explores the synergy between PL-AspectualACME and VML4Arch, a language that allows relating variability and architectural models. First, we present an approach that promotes the integration between VML4Arch and PL-AspectualACME with the aim to automatically derive product-specific ADL specifications (Section IV-A). After that, the modeling of MM architecture is presented using a combination of VML4Arch and PL-AspectualACME (Section IV-B).

A. Integrating VML4Arch and PL-AspectualACME

PL-AspectualACME is used to modularize variations occurring in the specification of software architectures. We have previously presented how specific elements from PL-AspectualACME were adopted for such purpose (Section III). MM was used as a case study to illustrate how these elements can help the modularization of variations related to components and configurations in the PL-AspectualACME specifications. Although, PL-AspectualACME promotes an adequate modularization of variabilities from architectural specifications, it does not address the automatic derivation/instantiation of product-specific ADL specifications.

Product derivation [18] is a fundamental activity to allow the automatic synthesis of products from the set of assets specified or implemented for a SPL. It has been mainly explored to support the automatic instantiation of code assets from a SPL architecture produced in domain engineering, in order to derive (customize, compose and/or generate) the code of a specific product in application engineering. Recent approaches [4][5][6] [7] have emphasized the importance of product derivation in activities related to requirements and architectural models. The automatic derivation of product-specific architectural (ADLs) specifications can help different activities, such as: (i) to improve the traceability between different models (requirements, architecture, design, code, testing) representing a product; and (ii) to help the static or dynamic (re)configuration of a software architecture. Additionally, the product-specific ADL specifications can also be used as an initial step during the process of producing and customize code assets from a SPL architecture to derive the code of a product.

PL-AspectualACME proposes the use of the Representation and Select port concepts to explicitly modularize and isolate variabilities occurring in a SPL architecture specification. However, it does not provide support to automatically decide which of these concepts will be part of ADL specifications for each different product. The integration of PL-AspectualACME and VML4Arch aims to address this requirement. VML4Arch can be used to explicitly decide which representations will be part of an ADL specification generated for a specific product. Each VML4Arch specification allows relating features from feature models (problem space) to representations from PL-AspectualACME specifications (solution space).

Figure 7 shows an overview of our approach by illustrating how PL-AspectualACME and VML4Arch can be used together to enable the derivation of product-specific ADL specifications. Each VML4Arch specification defines a series of variation points and variants. Inside these elements can be defined SELECT statements. Each SELECT statement defines a set of actions that will be executed only if the variation point or variants related are selected during the process of product derivation. Two kinds of
actions are used in our integration: (i) add element - that is used to decide which representations will be part of the ADL specification being generated; and (ii) connect - that is used to define configurations in attachments, such as to connect two components using a specific interface. It is worth mentioning that there is a slight difference when using the add primitive within different kinds of variation points. When used within an alternative or an inclusive-or variation point, the add primitive incorporates one component to a specific architectural element. When used within an optional variation point, the add primitive composes one port to a specific component. During the product derivation process, these actions are processed to explicitly decide which representations will be part of a product-specific ADL specification generated. The invoke statement of VML4Arch specifications is used to express the selection of features desired for a specific product. It defines which variation points and variants from the VML4Arch specification will be considered when deriving a PL-AspectualACME specification.

B. Modeling MobileMedia with VML4Arch

The VML4Arch specification for the MobileMedia defines how different variabilities (representations and configurations) in its PL-AspectualACME specification must be instantiated when specific features are selected, thus characterizing a product.

Figure 8 presents a partial view of the VML4Arch specification for MM. It depicts two main concerns: MediaSupport and MediaManagement. The first concern encapsulates the variation point which describes the types of media supported by MM. The latter concern describes the optional features. The first concern defines an inclusive-or variation point, named TypeOfMedia (line 2), which provides two variants: Music (line 4) and Photo (line 9). A number of actions are created for both variants in their SELECT blocks. For the Music variant, the first action adds the Music representation to the System (line 6). Next, the port \( m \) of Music is connected to all ports containing the word “Media” of the MediaManager component (line 12). The actions created for the Photo variant are equivalent to those created for the Music variant. The MediaManagement concern (line 15) describes three optional variation points. For each variation point is defined only one action, and every action has the same behavior: it adds one specific representation to the MediaManager component. The SetAsFavouriteMedia optional variation point (line 21), for example, adds a representation with the same label to the MediaManager component (line 24).

Figure 9 schematically presents our integrated approach with VML4Arch and PL-AspectualACME. Four distinct files are illustrated: (i) file 1 is a VML configuration specification in which a set of invokes are defined, i.e., variants and variation points are selected for a specific product.
Figure 9. Integrating VML4Arch and PL-AspectualAcme

instantiation; (ii) file 2 describes the SPL variabilities in an VML specification; (iii) file 3 describes the MM architecture in an PL-AspectualACME specification; and (iv) file 4 is the product architectural description in PL-AspectualACME containing all the components, ports and attachments that are directly associated to the variants and variation points requested (file 1).

Next, we describe the process of derivation of product-specific PL-AspectualACME specifications. Before the VML configuration file (Figure 9, file 1) is processed, all mandatory features from the SPL architectural description (Figure 9, file 3) are copied into the product architectural description file (Figure 9, file 4). Then for each invoke statement in the configuration file, the instantiation process occurs as follows:

1.A. The first invoke statement in the configuration file is processed and one specific variation point in the SPL variabilities description file (Figure 9, file 2) is pointed;
1.B. The set of actions within the SELECT block of the SendMediaViaSMS variation point are then transformed into model transformations instructions;
1.C. Based on the SPL architectural description file, the transformation instructions compose the SendMediaViaSMS port with the common core elements of the MediaManager component.

The same steps for the first invoke statement can be taken for the second invoke. The steps taken for the third invoke statement are:

2.A. The first invoke statement points to the Photo variant within the TypeOfMedia variation point;
2.B. The first action within the SELECT block is trans-
formed into a model transformation instruction;

2.C. Based on the SPL architectural description, the transformation instruction composes the Photo component with the common core elements of the System;

2.D. The second action within the SELECT block is transformed into a model transformation instruction;

2.E. Based on the SPL architectural description, the transformation instruction connects the MediaManager and Photo components through the MediaCross connector.

Finally, after all those steps, the architectural description for a given product is obtained (Figure 9, file 4). It covers only the variabilities selected using the invoke statements (Figure 9, file 1).

V. RELATED WORK

There are some initiatives in terms of ADL to specify product lines. However, none of them focus on managing variability by combining: (i) an aspect-oriented ADL, that supports the modular specification of a SPL architecture; and (ii) a variability modeling language that links the feature model and the software architecture model.

ADLARS [8] is an architectural description language that proposes a relational model which links features with architectural structure. Differently from our approach, the authors argue that capturing feature relationships in the architecture description makes the process of deriving product-specific architectures easier. We follow an approach that separates the product line architectural description from the specification of a product-specific architecture. We advocate that our proposal also promotes better separation of concerns (domain engineering separated from application engineering), and offers a seamless extension of an existing ADL, which avoids that software architects be burdened with a large amount of new abstractions.

Koala [10] and Mae [9] are ADLs that define special abstractions to capture variation points. Mae uses a new type of component, a variant component; Koala uses a design pattern. In contrast, PL-AspectualACME explores existing ADL abstractions, such as representations, ports and properties. While we adopt an architectural description and a variability modeling language, Koala embeds a special element, the diversity interface, to choose which variant appears in a product-specific architecture. xADL 2.0 [11] associates each variant with a Boolean condition that is evaluated and a concrete element is chosen according to the condition.

In terms of languages to express variability, Bachmann et al. [19] propose an orthogonal variability modeling (OVM) that defines a separate architectural view to express the variability model. It deals only with document variability, it does not define the composition of architectural variabilities as VML4Arch does.

VI. FINAL REMARKS

In this paper, we presented an integrated approach to variability management in software architecture specifications. PL-AspectualACME, an aspect-oriented architecture description language, was proposed to provide support for modeling an SPL architecture considering their commonalities and variabilities (Section III). VML4Arch was used to enable the derivation of product-specific architectural description based on a PL-AspectualACME specification (Section IV). The integration of both approaches improves the variability management at the architecture level by providing mechanisms to explicitly modularize, manage, and derive the variabilities encountered in SPL architectures. Additionally, it can also bring benefits to the traceability of variabilities along architecture representations. The approach was developed based on existing work developed in the research area of architectural description languages, specifically ACME. As PL-AspectualACME is a seamless adaptation of an existing ADL, the architects deal with architectural abstractions that are already intuitive for them.

As a future work, we intend to extend ACME Studio and integrate it with the VML4Arch tool suite in order to provide a complete tool support to PL-AspectualACME. We will also explore the adoption of our approach in the modeling of systems and product lines that require automatic and dynamic adaptations. Finally, we plan to investigate the possibility to use our integrated approach to generate initial versions of a SPL architecture or a product.

REFERENCES


