Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches

Krzysztof Czarnecki
University of Waterloo, Canada
czarnecki@acm.org

Paul Grünbacher
Johannes Kepler University Linz, Austria
paul.gruenbacher@jku.at

Rick Rabiser
CD Lab for Autom. Softw. Eng., JKU Linz, Austria
rabiser@ase.jku.at

Klaus Schmid
University of Hildesheim, Germany
schmid@sse.uni-hildesheim.de

Andrzej Wąsowski
IT University of Copenhagen, Denmark
wasowski@itu.dk

ABSTRACT

Variability modeling is essential for defining and managing the commonalities and variabilities in software product lines. Numerous variability modeling approaches exist today to support domain and application engineering activities. Most are based on feature modeling (FM) or decision modeling (DM), but so far no systematic comparison exists between these two major classes of approaches. Over the last two decades many new features have been added to both FM and DM and it is tough to decide which approach to use for what purpose. This paper clarifies the relation between FM and DM. We aim to systematize the research field of variability modeling and to explore potential synergies. We compare multiple aspects of FM and DM ranging from historical origins and rationale, through syntactic and semantic richness, to tool support, identifying commonalities and differences. We hope that this effort will improve the understanding of the range of approaches to variability modeling by discussing the possible variations. This will provide insights to users considering adopting variability modeling in practice and to designers of new languages, such as the new OMG Common Variability Language.

Keywords

variability modeling, feature modeling, decision modeling, product lines

1. INTRODUCTION AND MOTIVATION

Variability modeling is used to understand and define commonalities and variabilities in software product lines and to support product derivation. It also helps to bring modeling concepts into otherwise mostly code-driven projects. For example, in some real-world projects, such as the Linux kernel and the embedded operating system eCos, variability modeling is the only form of modeling that is used [13]. Among the existing approaches to variability modeling, feature modeling (FM) and decision modeling (DM) have gained most importance. Most existing FM approaches are more or less directly derived from the work on Feature-Oriented Domain Analysis (FODA) by Kang *et al.* [42]. Several papers compare FM approaches from different viewpoints [19, 61, 12]. DM exists nearly as long as FM, and, similarly, most (if not all) existing DM approaches [60] have been influenced by the Synthesis method [64]. Among other variability modeling approaches, one should mention Orthogonal Variability Management (OVM) [51]. In this paper, however, we focus solely on comparing DM and FM, as the currently most popular classes of methods, leaving a broader survey to future work.

Despite their significant role in product line research and practical applications, a systematic comparison of FM and DM approaches is still lacking. Existing general surveys of variability modeling techniques [17, 63] do not discuss the differences and commonalities of FM and DM. Over the last two decades, the original FM and DM proposals have been extended with many new features. Without a thorough understanding of the differences, deciding between FM and DM approaches is difficult. As we will see, however, the capabilities of modern FM and DM approaches have significantly converged.

Fig. 1 depicts an example of a feature model using a slightly adapted FODA notation [23] (a) and an example of a decision model in two different notations (b and c) for a fictitious mobile phone product line. Fig. 1(b) is based on the style of Mansell and Sellier [46] and Schmid and John [59]; Fig. 1(c) shows the original Synthesis notation [64]. We will refer to this example throughout the paper to clarify differences and commonalities of FM and DM.

Our aim is to clarify the relation between the two major classes of variability modeling approaches and increase clarity in the research field, in which, as we show later, a number of seemingly contradictory statements on DM and FM exist. We also aim to explore potential synergies.

The paper is based on our experiences as experts in the respective areas of DM and FM and on our knowledge of the literature in these fields. The results are based on 3 face-to-face meetings and 11 phone conferences with a typical duration of 1,5 hours in which we first agreed on the dimensions and then discussed the relation between DM and FM in detail for each dimension. We did not make the attempt to involve additional experts in this work. We also did not follow a systematic method to analyze the literature (such as a systematic literature review). Obviously, this process represents a threat to validity. However, the group of five authors carefully

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
VaMoS'12, January 25–27, 2012, Leipzig, Germany.
Copyright 2012 ACM 978-1-4503-1058-1 ...$10.00.

discussed all cases of disagreement and the authors achieved consensus on all reported results.

Our work is of interest to researchers and practitioners deciding on approaches and tools to variability modeling as well as to tool builders who want to improve the variability modeling capabilities of their tools. We also regard this comparison as an important step towards standardization of variability modeling [1].

Sect. 2 describes the background and history of FM and DM. Sect. 3 discusses the commonalities and differences of FM and DM in detail using ten dimensions defined based on comparison frameworks previously used by the authors within the areas of FM [13] and DM [60]. Sect. 4 discusses how variability modeling approaches used in real-world projects such as Kconfig and CDL [13] have applied concepts from both FM and DM in practice and what can be learned from these languages regarding potential synergies and standardization efforts, such as the OMG Common Variability Language (CVL) [1]. We summarize main findings and suggest future work in Sect. 5.

2. BACKGROUND AND HISTORICAL PERSPECTIVE

Feature Modeling.

FM was originally proposed as part of the FODA method [42]. In FODA, a domain is defined as a set of current and future systems sharing common capabilities. Domain analysis aims at discovering and representing commonalities and variabilities among them. In FODA, *feature models capture features—the end-user's (and customer's) understanding of the general capabilities of systems in the domain—and the relationships among them.* Fig. 1(a) depicts a feature model for the mobile phone example as a feature tree with mandatory and optional features and a cross-tree dependency (*requires*).

Historically, FODA builds on, among others, Neighbors's work on Draco [50], and Batory's domain analysis of DBMS and the Genesis tool [10]. Feature-orientation can be traced back to the ROSE tool [45, 14], which supported 'feature-based selection' of components, and the KAPTUR system [49], which used 'distinctive features' to distinguish systems within a domain.

FODA has spurred a multitude of works on extending the original notation and on modeling and implementing systems using feature models. Notable extensions include group cardinalities [55], feature cardinalities [25], and feature inheritance [5]. Existing surveys cover some of these notations [61] and applications [39]. FM is also an integral part of feature-oriented software development (FOSD), a paradigm focused on treating features as modular, first-class entities throughout the entire development cycle, including requirements, design, implementation, and test [2]. Thus, the meaning of the term feature has been broadened dramatically over time.

Decision Modeling.

The earliest documented approach to DM is found in the Synthesis method [64]. The method, developed by the Software Productivity Consortium for industrial use, provided an early reuse process model. Most—if not all—DM approaches can ultimately trace their heritage back to Synthesis, which defines a decision model as *a set of decisions that are adequate to distinguish among the members of an application engineering product family and to guide adaptation of application engineering work products* [64]. This early definition emphasizes product derivation—as opposed to describing the domain, which is the main purpose of FM in FODA. Fig. 1(c) depicts a Synthesis decision model for the mobile phone example, in a textual representation.

Most DM approaches were either inspired by industrial applications or developed in close collaboration with industry. Several of them are discussed in a recent comparative analysis [60]. In VManage [31, 46] and the approach by Weiss and Lai [69], a decision model is *a document defining the decisions that must be made to specify a member of a domain* [46]. The tool-supported DOPLER approach [28] has been developed to guide the derivation of customer-specific products. The work of Schmid and John [59] provides a common modeling foundation that can be mapped to a wide range of notations. Fig. 1(b) depicts a decision model for the mobile phone example in a tabular representation that combines elements from several previous works [46, 59, 28].

3. COMPARISON

We compare FM and DM along ten dimensions, extracted from two earlier frameworks for discussing FM and DM approaches [13, 60]. These dimensions are the following *applications*, *unit of variability* (feature vs. decision), *orthogonality*, *data types*, *hierarchy*, *dependencies and constraints*, *mapping to artifacts*, *binding time and mode*, *modularity*, and *tool aspects*.

The first of the two frameworks underlying our dimensions [13] was used to study Kconfig and Component Description Language (CDL), which are two real-world variability languages, and compare them with FM. Its dimensions included feature kinds, feature representation, hierarchy, constraints, code mappings, binding modes, and modularization.

The second framework [60] was used to compare different decision modeling approaches. Its dimensions were decision data type, constraints and dependencies, artifact types and relations, product derivation support (including binding time), aggregation of decisions and tool support.

We integrated and generalized the two frameworks to adequately address the needs of both DM and FM, arriving at the ten dimensions used in this paper. The integration was straightforward as the frameworks overlap significantly, giving us eight of the ten dimensions. We then extended this set with two additional dimensions: *applications* and *orthogonality*. We added applications since one of our goals was to clarify the applicability of FM and DM. We added orthogonality of variability modeling since several authors mentioned this aspect when discussing FM and DM (e.g., [51, 33]).

The remainder of this section compares DM and FM according to these ten dimensions. The three left-most columns of Table 1 summarize this comparison; the remaining columns will be discussed in Sect. 4).

3.1 Applications

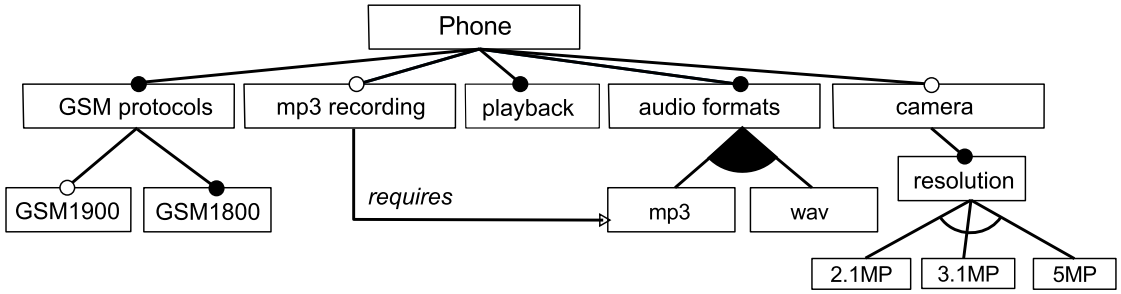
We will first focus on the purposes to which the approaches are most commonly used. The emphasis is slightly different for the two approaches.

DM focuses on variability modeling and derivation support; FM focuses on commonality and variability modeling, but it also provides derivation support. Fig. 1 shows that FM and DM both represent variabilities, but only FM also presents the commonalities: GSM Protocol GSM 1800 and playback are part of the feature model but are intentionally not represented in the decision models, as no decision needs to be made for these common features.

The key applications of DM include planning and managing variabilities in SPLs and derivation support, i.e., supporting application engineers in configuring and creating products based on software product lines [69, 46, 59, 28]. The application scope has also broadened over time and includes efforts to integrate DM with goal

Table 1: *Commonalities* and **differences** between FM and DM in diverse dimensions. The rightmost columns, Kconfig, CDL and CVL initial are discussed in Sect. 4.

dimension	decision modeling	feature modeling	Kconfig	CDL	CVL initial
applications	*variability modeling; derivation support*	**diverse applications: concept modeling (e.g., domain modeling)**, *variability* **and commonality** *modeling; derivation support*	modeling variability in the kernel; derivation support	modeling variability in eCos; derivation support	variability modeling; derivation support
unit of variability	**decisions to be made in derivation**	**features are properties of concepts, e.g., systems**	drivers, subsystems, kernel options, build option	drivers, subsystems, kernel options, build option	VSpecs: essentially decisions in derivation; pre-made decisions (mandatory features)
orthogonality	orthogonal	mostly used in orthogonal fashion	orthogonal (added configuration UI concepts, e.g., menus)	orthogonal (added architectural concepts, e.g., packages, components)	orthogonal (but admitting non-orthogonal uses is discussed)
data types	*comprehensive set of basic types*; *composite*: **sets, records, arrays**	*comprehensive set of basic types*; **references**; *composite*: **via hierarchy, group and feature cardinalities**	Boolean, tristate, numbers and strings; choices	none, bool, data (dynamically typed values incl. int, string, real), booldata	choices; parameters with comprehensive set of basic types; classifiers
hierarchy	**secondary concept; diverse approaches**, e.g., **visibility or relevance** hierarchy (no decomposition)	**essential concept; single approach**: tree hierarchy modeling, parent-child configuration constraints and **decomposition**	characteristics of FM&DM: essential organization means (FM), visibility induced, driven by UI concepts (DM)	like in FM (essential organization means; decomposition hierarchy)	essential concept; vspec tree, like in FM
dependencies and constraints	*no standard constraint language but similar range of approaches (Boolean, numeric, sets)*	*no standard constraint language but similar range of approaches (Boolean, numeric, sets, quantifiers)*	propositional three-valued logics with comparisons	propositional Boolean logics with expressions on data	propositional and predicate logic with expressions on data
mapping to artifacts	**essential concept**; *no standard mechanism*	**optional concept**; *no standard mechanism*	mapping to C preprocessor via a custom build system (no explicit mapping model)	explicit mapping in the variability model; variability symbols available to C preprocessor	essential concept; mapping model, base-model independent
binding time and mode	*not standardized, occasionally supported*	*not standardized, occasionally supported*	static or dynamic binding decided at compile time	static binding	not included in CVL (dependent on application)
modularity	*no standard mechanism;* **decision groups play partly this role**	*no standard mechanism;* **feature hierarchy plays partly this role**	model is split into files; no modularization beyond hierarchy in the language	loadable packages, reparenting	explicit support (packages, configurable units)
tool aspects	*representation of models as lists, tables, trees, and graphs*; **configuration UI: an (ordered) list of questions**	*representation of models as lists, tables, trees, and graphs*; **configuration UI: usually a tree (unordered)**	modeling in textual syntax; configuration UI: a tree with controlled visibility	modeling in textual syntax; configuration UI: a tree with controlled visibility	user interfaces are the domain of vendors; basic concrete syntax for VSpecs close to FM
	diverse solutions for configuration workflows **(essential)**	*diverse solutions for supporting configuration workflows* **(secondary concept)**	no support for configuration workflows; reconfiguration scripts	no support for configuration workflows beyond visibility conditions	



(a) Feature model in a tree notation—slightly adapted from FODA [42]

decision name	description	type	Range	cardinality/constraint	visible/relevant if
GSM_Protocol_1900	Support GSM 1900 protocol?	Boolean	true | false		
Audio_Formats	Which audio formats shall be supported?	Enum	WAV | MP3	1:2	
Camera	Support for taking photos?	Boolean	true | false		
Camera_Resolution	Required camera resolution?	Enum	2.1MP | 3.1MP | 5MP	1:1	Camera == true
MP3_Recording	Support for recording MP3 audio?	Boolean	true | false	ifSelected Audio_Formats.MP3 = true	

(b) Decision model in a tabular notation [59, 28]

```
GSM_Protocol_1900: one of (GSM_1900, NO_GSM_1900)        {indicates whether support for making and receiving calls using
                                                          GSM 1900 is available}
Audio: list of (WAV, MP3)                                {indicates the types of supported audio formats}
Camera: composed of
  Presence: one of (Camera, NO_Camera)                   {indicates whether camera support is available}
  Resolution: one of (2.1MP, 3.1MP, 5MP)                 {resolution of the camera}
MP3_Recording: one of (MP3, NO_MP3)                      {indicates whether MP3 recording is available}

Constraints
Resolution is available only if Presence has the value Camera
MP3_Recording requires that also MP3 Audio is supported
```

(c) Decision model in the textual notation of Synthesis [64]

Figure 1: A feature model and two decision models for a fictitious mobile phone product line; same variability, while commonalities are only shown in the feature model.

modeling in service-oriented systems [34], to use DM with model-driven architectures [31], to use DM for code generation [70], and to support personalization of ERP software by end-users [53].

FM has targeted a broader set of roles in the development life cycle, starting with the original application in FODA—domain analysis and scoping, but also including design and representation of product line architectures and evolution—helping to see what features are available, which new features should be added and where, and which existing features might need to be retired.

FM has also been used like DM—as a central variability model and a basis for derivation (e.g., [38, 48, 47]). Interestingly, Gears and Pure::Variants, which are industrial SPL tools supporting FM, have been used predominantly for variability modeling and derivation support.[1] However, FM is also used as a general concept modeling technique, e.g., in comparative surveys [24]. Thus, in contrast to DM, derivation support is not the essential application of FM.

3.2 Unit of variability

This dimension looks at the key concepts that are used to model variability in both types of approaches.

The units of variability are *decisions* for DM and *features* for FM. *Decisions are differences among systems*; they can be anything that an application engineer needs to decide during derivation. For example, the engineer needs to decide whether a particular phone will support the GSM 1900 protocol (cf. Fig. 1(b) and (c)).

The term "feature" is highly overloaded among different FM approaches [19] and also in the wider context of software engineering. FODA defines a feature as "a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems." [42] Some authors define features as requirements-level entities, e.g., "a cohesive set of individual requirements" [16] or "a set of related requirements, domain properties, and specifications" [19]. FOSD considers a feature as "an increment in product functionality" [11], and focuses on representing features explicitly throughout the development life cycle, including requirements, design, implementation, and tests. Thus, FM has represented a wide range of system and environment properties as features—functional or non-functional, and pertaining to different life cycle activities. This usage is consistent with the definition of a feature as a "characteristic of a concept (e.g., system, component, etc.) that is relevant to some stakeholder of the concept" [23].

Clearly, *anything that FM captures as variable features can be captured by DM as decisions and vice versa*. Essentially, a decision in DM reifies the need to decide some variable feature (property) of a system, its environment or both. The essential difference between FM and DM is that common features, e.g., GSM 1800 in Fig. 1(a), are outside the scope of DM and thus missing in Fig. 1(b) and (c).

Finally, it should be emphasized that features in FM or decisions

[1] According to personal communication with the CEOs of the respective tool vendors.

in DM are typically *abstractions* over other artifacts, such as requirements, designs, and implementations (rather than, say, directly representing requirements [15]). Consequently, features and decisions need to be *mapped to artifacts*, which we discuss later in Section 3.7.

3.3 Orthogonality

Orthogonality of variability modeling is the degree to which variability is modeled as a separate concern. An *orthogonal variability model* is devoted primarily to capturing variability—it offers, as Pohl *et al.* put it, "a cross-sectional view of the variability across all software-development artifacts" [51]. Orthogonality is lost when variability is defined as an integral part of development artifacts; for instance, by directly defining variability within code.

Gomaa [32] models variability directly within UML models of requirements and design. He uses class diagrams syntax to represent feature models and gives them special semantics to capture variant derivation. He also realizes variability in requirements and design models by exploiting existing syntactic mechanisms of UML, such as conditions on transitions and stereotypes.

Orthogonal variability modeling has been the essence of DM and is the predominant application of FM today. Decisions focus purely on variability—representing what needs to be decided on. Feature models are used today typically as orthogonal representations of commonality and variability or just variability. For example, Groher and Völter use feature models, represented in Pure::Variants, for orthogonal variability modeling [33].

The orthogonality of FM is sometimes questioned in the literature. For example, Pohl *et al.* state that "software development models (e.g., feature models) are already complex, and they get overloaded by adding variability information." [51] This statement relates to features as user-visible system capabilities—the original meaning of features in FODA. Lists of such capabilities, which are often packages of requirements, may be useful artifacts on their own, even if no variability is involved. Thus, adding variability to such artifacts could be classified as non-orthogonal variability modeling; however, as argued above, feature models are used today mainly as orthogonal representations of commonality and variability (or just variability) over other artifacts.

3.4 Data types

A decision or feature model denotes a set of structures representing configurations and thus can be viewed as a collection of type definitions. These types determine the available primitive values and composite structures that can be selected or constructed during configuration.

Both DM and FM cover a comparable range of data types. Individual approaches cover different subsets of this range, however, and *Boolean and composite types are represented differently* across the two classes of approaches. In FM, the Boolean type is implicit in *optional features*—e.g., GSM 1900 in Fig. 1(a); in DM, the type is either explicit (cf. Fig. 1(b)) or encoded as an enumeration (cf. Fig. 1(c)). Many FM and DM notations support additional primitive types, including string, integers, and reals. Synthesis [64] includes even date and time, e.g., to model a license expiry date. All DM notations offer enumerations as primitive data types and some offer records or sets or both. FM supports these composite types by relying on hierarchy, group constraints, and—if supported—feature cardinalities. Enumerations are often represented as *xor-groups* in FM (cf. resolution in Fig. 1(a)), but some languages, e.g., TVL [18], have an explicit enumeration type. Finally, some DM and FM approaches support multiple instantiation of a decision or feature. In cardinality-based FM, features having cardinality with an upper bound higher than one can be instantiated multiple times [25]. The DM approach V-Manage [31] provides this functionality as well. Some FM notations, e.g., Clafer [7], support *reference* types—their values are references to instances of other features.

3.5 Hierarchy

All *FM notations support a tree-like organization of features as an essential concept*. This organization allows decomposing higher-level features into more detailed ones as shown in Fig. 1(a). Hierarchy is explicit both in feature models and in configuration views, e.g., as graphical trees or indented text. Semantically, feature hierarchy imposes configuration constraints, where selecting a feature implies selecting its parent. Hierarchies in feature models are typically not very deep on average, but some models can have deeper parts. For example, the median hierarchy depth in the models collected in the SPLOT repository (http://www.splot-research.org/) is 3; maximum hierarchy depth is 10.

In DM, hierarchy is a secondary concept, as DM models are thought of primarily as flat lists or tables of decisions, with dependencies. However, already Synthesis [64] supported groups of decisions (cf. Fig. 1(c): 'Camera: composed of'), a concept that directly corresponds to hierarchy in feature models. Hierarchies of decisions are mainly used to guide the configuration process. In contrast to FM, there is no standard hierarchy mechanism or hierarchy semantics across DM approaches. Some notations, e.g., [6], support nesting of records, which induces a hierarchy similar to that of FM. In DOPLER [28], hierarchy is induced in the configuration view by *visibility conditions* of decisions. The visibility of a decision implies the visibility of its parents, which is not a configuration constraint. Visibility thus helps to structure the configuration process but has no impact on the definition of legal configurations. In the approach by Schmid and John [59], a hierarchy is induced by *relevancy conditions*—only values of relevant decisions become part of a configuration description. Hierarchies in real-world decision models [28] are typically quite flat, i.e., the maximum depth in the DOPLER models occurring in a multiple case study in the domains of industrial automation systems and business software [28] is 3.

3.6 Dependencies and constraints

Both FM and DM allow expressing dependencies that exist in the domain or its implementation or both. Most important are configuration constraints, which restrict the legal configurations of features or decisions. FODA has a very simple constraint language, which allows saying that a feature requires or excludes another feature (cf. Fig. 1(a)). Even though FODA included examples of numeric features, most formalizations of constraints has focused on supporting arbitrary Boolean restrictions [9, 37]. Synthesis [64] anticipated richer constraints, e.g., over numeric domains, but did not offer a precisely defined constraint language (cf. Fig. 1(c)). Subsequent DM methods, including KobrA [6], Schmid and John [59], and DOPLER [28], define their constraint languages much more precisely. Each of these languages surpasses the previous in expressiveness. For example, Schmid and John admit logic formulae with arithmetic operations and set operations, and DOPLER allows escaping to pure Java code to formulate specific conditions (cf. Fig. 1(b)). *Modern FM languages*, e.g., TVL [18], Clafer [7], and those in Gears and pure:variants, *allow a range of constraints comparable to that of modern DM languages*. Constraint support for references and multiple instantiation of features, which requires some form of predicate logic or relational algebra, is rare, but present in some, e.g., in Clafer.

Beyond configuration constraints, DM languages, such as DOPLER, allow formulating visibility conditions on decisions. These conditions control which decision prompts are visible to the application engineer during derivation (cf. Fig. 1(b)). Also, *virtually all DM and some FM languages support specifying default values as either constants or computed defaults*.

3.7 Mapping to artifacts

When modeling variability, features or decisions are just abstractions of the variabilities realized in other development artifacts. Understanding how features or decisions map to these artifacts is an important aspect of both DM and FM. Since derivation support is an essential application of decision modeling, *mapping to artifacts is an essential aspect of DM*. Some uses of FM, such as domain modeling, do not necessarily assume derivation; thus, *artifact mapping is optional in FM*.

In practice, a wide range of mapping techniques are used in both DM and FM. They typically relate decisions or features (high-level variability abstractions) to *variation points* (locations in artifacts where variability occurs). Schmid and John [59] provide a set of artifact-notation-independent primitives for expressing variability in artifacts, such as optionality, alternative, set selection, and value reference. Some approaches associate artifacts with *inclusion conditions* (e.g., [36, 21, 28]), and some associate features or decisions with the artifacts to be included (e.g., [6]). A more expressible mechanism is fragment substitution [35], which allows relatively rich artifact transformations. Some DM and FM approaches define a separate artifact model, which exposes artifact abstractions to the decision or feature model (e.g., DOPLER and pure::variants). Finally, FOSD [2] research has looked into different approaches of representing variability in artifacts, including conditions as annotations on model or program elements [21, 36, 44] and compositional approaches (e.g., [8, 4]).

The mapping between features and varaibility realization can also be implicit. For example, the PLUS method of Gomaa [32] assumes a variant of the annotative approach, where the mapping of features to variation points is achieved by using feature names directly in UML models representing the system—such references are resolved by derivation tools just as if they were annotations. The approach uses the existing UML syntactic means to express variability, such as state diagram transitions depending on feature selection, rather than on model variables.

Finally, the variable artifacts can be organized to reflect and modularize their variability. For example, ADORA [65] organizes requirements models into hierarchical aspect containers that reflect the model variability—instead of using separate feature models with explicit mappings. Features are modeled as aspects that package and modularize variable model elements; they are composed by weaving. The approach uses view generation to generate abstract FM-like views and decision models for variability binding and configuration.

3.8 Binding time and mode

Mapping variability to artifacts may not be enough to define how products are built. In particular, variability models may need to include *further realization information*, such as described by a product plan [43]. An example of such information is the *binding time*, i.e., when a certain variability is bound, and the *binding mode*, i.e., whether the binding is static (fixed) or dynamic (modifiable) [23]. Examples of binding time are compile time, link time, deployment time, etc. So far *support for such information is rare in both FM and DM*, probably because it is highly application- and technology-dependent. Early FM and DM approaches mention the possibility to specify binding information [42, 59], but lack realization details. Work on concrete realization exists, though. Examples include parameterized binding mode via templates and inheritance [23, p. 234], binding time variability via aspect-oriented techniques [58, 57], time-line variability with constraints on transitions between configurations [30], and any-time variability in architectural description languages [67].

3.9 Modularity

Modularity mechanisms are needed as it would be inconvenient to define and maintain variability in a single model due to the large scale and complexity of many systems. Modularity is also fundamental in multi product lines, an emerging area addressing variability management for large-scale systems that comprise several self-contained but still interdependent product lines. For instance, Bühne et al. [15] describe several issues of feature modeling in multi product lines using examples from the automotive industry.

Feature hierarchy and decision grouping provide basic structuring; however, dedicated modularization mechanisms for FM and DM also exist. For example, DOPLER allows dividing decision models into a number of interrelated model fragments [29]. In FM several authors have proposed approaches to define a set of interrelated feature models, such as *multi-level feature models* [54, 26]. Schirmeier and Spinczyk [56] discuss top-down, bottom-up, and hybrid forms of composing feature models in multi-layered product lines. An upcoming survey gives a detailed account of modularization techniques in FM [40].

3.10 Tool aspects

Important aspects of FM and DM tooling include editing, exploration, and analysis of models and configurations. A detailed account of specific tools is out of scope; however, we make some general observations.

As expected, *UIs for model and configuration editing in FM usually emphasize hierarchy, but a similar range of representations, including lists, tables, trees, and graphs, have been used in both FM and DM tools.* For example, decision models are often represented as a list of questions to be answered (cf. description attribute in Fig. 1(b)), but may also show hierarchy [31]. Feature models are typically shown as trees, but specific views may show feature lists or tables, e.g., summarizing feature-to-artifact mappings, or graphs, e.g., showing cross-tree dependencies.

Due to their derivation focus, *DM tools pay special attention to decision filtering and workflows during configuration.* DOPLER [28] supports filtering using visibility conditions; Schmid and John [59] accomplish this task using relevancy conditions. DOPLER also allows grouping decisions to configuration tasks, which can be assigned to users involved in product derivation [52]. FM approaches have not addressed visibility conditions yet; however, extensive work exists on staged and workflow-based feature configuration [26, 38, 48].

Many researchers have worked on reasoning support for model and configuration analysis in FM. This body of work includes 1) finding and diagnosing inconsistencies in feature models and artifact mappings (e.g., [41]), 2) verification of product line assets in presence of variability (e.g., [20, 3, 27]) and 3) automated guidance during configuration. An extensive survey of 1) and 3) is available [12]. Many of these results likely carry over to DM. In fact, DOPLER provides configuration guidance, such as decision propagation.

4. VARIABILITY MODELING IN PRACTICE

Over the last two decades variability modeling has been embraced in practice. Commercial tools like Pure::Variants and Gears exist with their respective specification languages. Independently, software engineers have designed their own modeling languages. For instance, the Linux kernel project uses the home-grown variability specification language Kconfig [71, 62], which has been created independently of FM and DM research. Similarly, the eCos operating system uses its *Component Description Language* (CDL) [68]. Linux and eCos are open source projects, but similar languages appear in commercial systems, for instance OSEK [22]. All these languages can be placed into the category of FM and DM languages. Here we show how the main properties of DM and FM are reflected in practice, by comparison with (primarily) Kconfig and CDL. Simultaneously we relate to the current proposal of the *Common Variability Language* (CVL) standard [1]. The three rightmost columns in Table 1 summarize these three languages with respect to the same dimensions already discussed for DM and FM.

Applications and *orthogonality.* All mentioned languages (Kconfig, CDL, pure::variants, Gears, and CVL) target orthogonal variability modeling. These are all quite technical, heavy-caliber, tools that are meant to support end-to-end modeling and derivation, all the way to compilation and deployment. As such they are not indispensable at early design activities—other uses of FM, such as domain modeling, may be adequately supported with lightweight tools such as mind-map editors or spreadsheets.

Interestingly, CVL avoids the FM terminology, while retaining FODA-like concrete syntax of FM. We shall see below that nomenclature for *variability specifications* (VSpecs) in CVL resembles DM more than FM, while a VSpec tree looks like a feature model. Thus CVL avoids the confusion caused by the ambiguous meaning of the term feature. Whether a dedicated support for modeling features as high level, user-visible capabilities should be part of CVL is debated [1, p. 80]. Design alternatives for such support include dedicated native notation, applying variability orthogonally to (such) feature models, or complete lack of (dedicated) support.

Units of variability and *data types*. In practice, often rather simple and relatively small Boolean models are applied to requirements artefacts.[2] In contrast, models used at the implementation level can become very large, and involve very complex constraints. For instance, the Linux kernel Kconfig model and the eCos CDL model comprise thousands of features or decisions about different implementation and architecture aspects, including complex constraints, some relating 20 or more features or decisions [13]. They often include other types of variability than Boolean. For example, in the operating systems domain, integers are used to express sizes of resource pools such as threads, open files, or memory. Low level models occasionally also require the ability to instantiate features in multiple copies (see for example [22] about OSEK).

Since CVL is domain independent, it supports a broad range of types, including multiple instantiation, and a constraint language for expressing dependencies over these types. CVL distinguishes three types of *variability specifications* (VSpecs), essentially linked to their types: choice (Boolean), variable (other primitive types) and classifier (multiple instantiations). This terminology avoids the term feature. In CVL VSpecs are organized in trees (VSpec trees), which are shown using FODA-like concrete syntax [1].

Hierarchy. Hierarchy is uniformly present in all mentioned languages. Kconfig's hierarchy, although visibility-induced like in DOPLER, is still used much in FM spirit [13]. CDL hierarchy is used very much like in FM—it enforces parent-child implications and serves to organize the model.

Following its use of FODA syntax, CVL adopts hierarchy in the style of FM: it is used to decompose and organize the model and it also enforces parent-child constraints.

Mapping variability to artifacts. Kconfig and CDL have been designed as parts of build systems for their host projects. Thus they both support mapping of variability to artifacts. They follow the annotative approach, exposing names of features or decisions to the C preprocessor controling presence code fragments. Besides that, the mapping model is hidden in the build system (Kbuild) or represented in the variability model itself (CDL).

Mapping to artifacts is a core objective of CVL, which aims at external (orthogonal) definition of variability for any models described in MOF-based languages. CVL has a three-tier architecture: variability specifications (1) are mapped to artifacts (3) using a mapping model, called variability realization (2). Variability realization allows a number of variation point types: presence of objects, change of constant values in the models, object substitution and fragment substitution in style of [35]. Importantly, CVL stores the mappings outside the artifacts. Thus the main artifacts, the so called *base models*, remain unchanged by introduction of variability, and can be processed by the native tools supporting the languages in which the artifacts are expressed.

The *binding time and mode* are highly dependent on the application domains and technology applied. CDL, which supports an operating system for small embedded devices, only has static, compile-time binding. Interestingly, Kconfig provides native support for binding mode, by means of so called tri-state features or decisions. It allows to decide at compile time whether a given functionality should be linked statically or whether it should stay dynamically loadable. Thus, Kconfig implements support for binding mode from the model and the configuration UI through to realization in the build process (unlike other variability modeling languages that treat binding as a simple parameter in the high level model).

Since CVL is domain independent, it follows both FM and DM and does not standardize this aspect.

Modularity is essential to any models of realistic size. Kconfig allows distributing a monolithic model over multiple files—a form of primitive untyped modules. CDL provides packaging and import constructs, with an interesting ability to inject new model fragments into an existing hierarchy (reparenting). This enables model extensions that plug into an existing model decomposition hierarchy.

The initial CVL proposal allows both for simple aggregation (packaging) and encapsulation of *configurable units* (CU). CUs wrap variability realization together with variability specification. CVL also supports configuration interfaces (simply VSpec trees), which specify variability allowed for an encapsulated CU. The intended uses of CUs include supplying configurable reusable third-party components, staged configuration and multi-level product lines.

Tool aspects. CDL and Kconfig are both textual languages, with no specific editor support, besides usual text editors. They both have tools supporting product derivation, providing user guidance. In particular, CDL's configurator can propose fixes for constraint violations. Gears and pure::variants have tool support for model editing (for multiple representations) and configuration. Other tools, such as analyzers, are rare in the industry, but quite widespread in research [12]. This gap is probably caused by lack of proper

[2] Personal communication with BigLever Inc., the vendor of Gears.

technology transfer. Recent successful attempts to introduce model analysis tools to practitioners indicate that these tools are indeed useful [66].

Three aspects of tool integration stand out in CVL's design. First, the language-independent approach to adding variabilit y allows processing models produced by CVL using regular modeling tools (although to exploit the variability information, plugins for modeling tools will be needed). Second, CVL will need tools supporting 1) configuration of the VSpec tree and 2) materialization (resolution of variation points in the base models based on configuration). Finally, CUs will allow encapsulation of variability realized by some other, non-CVL tools, and interfacing it to CVL infrastructure using variability interfaces.

5. CONCLUSIONS AND FUTURE WORK

We have provided a detailed structured comparison of feature and decision modeling, including a discussion of our findings in the light of other related practical variability modeling approaches like Kconfig, CDL, and CVL.

Based on our analysis, the main difference between FM and DM is that FM supports both commonality and variability modeling, whereas DM focuses exclusively on variability modeling. In practice, this difference is limited by two factors: 1) FM is often used, just like DM, to capture variabilities that need to be resolved during product derivation; 2) commonality modeling has limited impact on variability languages and tools—as the constructs used to model variability can also be used to model commonality. The latter point is reflected by the fact that mandatory features can be used to model both common features, such as GSM 1800 in Fig. 1(a), and groupings of variable features, such as GSM Protocols; however, many DM approaches also provide grouping nodes, like the mandatory feature GSM Protocols. Thus, the main difference is largely one of methodology: if FM is used to model both commonalities and variabilities, common features such as GSM 1800 are also captured.

The remaining differences are either minor or largely historical. The initial use of FM in FODA was to capture user-visible system capabilities; DM did not have such a restriction. Today, FM is mostly used, just like DM, as an orthogonal variability modeling technique, where features, like decisions, abstract over variabilities pertaining to different types of properties (functional and non-functional), levels of abstraction (environment, system, subsystem) and lifecycle stages (requirements, design, implementation, and test). Other differences are minor: 1) hierarchy is essential and has uniform semantics (child-presence-to-parent-presence implications) in FM and is secondary and has varied semantics (child-presence-to-parent-presence or child-visibility-to-parent-visibility implications) in DM; 2) mapping to artifacts is essential in DM and, depending on the use case, optional in FM. On the other hand, both FM and DM have a similar range of data types, constructs for expressing constraints, and modularization mechanisms and they both lack standardized support for binding time and mode.

Thus, our analysis uncovers a significant convergence between FM and DM. This conclusion is also supported by the fact that practical variability modeling approaches, such as Kconfig and CDL, combine concepts from FM and DM. For example, both Kconfig and CDL support hierarchy (FM), group constraints (FM), and visibility conditions (DM). Consequently, the specific capabilities of a variability modeling approach, such as the data types offered, the expressiveness of the constraint language, support for modularity, and the available tool support, are much more important factors when selecting an approach than its classification as DM or FM.

We hope that our study can fruitfully influence the CVL standardization process, by improving clarity of the design space and terminology of variability modeling. Our analysis of the initial CVL proposal reveals no glaring omissions or misguided decisions in its design, while it does single out clear advantages that CVL has, due to influence of research on its design (esp. constructs for encapsulations and interfacing, or the non-invasive, language-independent approach).

Possible extensions of this work include conducting an expert survey on FM and DM with a wide set of respondents. Another work would be to compare FM and DM by comparing the formal semantics of the existing FM and DM languages. Since only selected dimensions of our comparison can be effectively formalized, such an approach would be restricted to these dimensions. It would also be interesting to expand the analysis with a systematic artifact study of models developed within FM and DM communities, including a qualitative and statistical comparison.

6. ACKNOWLEDGEMENTS

This work has been partially supported by the Christian Doppler Forschungsgesellschaft, Siemens VAI Metals Technologies, and Siemens Corporate Technology. This work has been partially supported by the INDENICA project, funded by the European Commission grant 257483, area Internet of Services, Software & Virtualisation (ICT-2009.1.2) in the 7th framework programme.

Czarnecki and Wąsowski thank the other members of the CVL design team for insightful discussions on variability modeling.

7. REFERENCES

[1] Common variability language (CVL). OMG Initial Submission. Available on request., 2010.

[2] S. Apel and C. Kästner. An Overview of Feature-Oriented Software Development. *Journal of Object Technology*, 8(5):49–84, 2009.

[3] S. Apel, C. Kästner, A. Größlinger, and C. Lengauer. Type safety for feature-oriented product lines. *Automated Software Engineering*, 17(3):251–300, 2010.

[4] S. Apel, C. Kästner, and C. Lengauer. FEATUREHOUSE: Language-independent, automated software composition. In *Proc. of the 31st International Conference on Software Engineering*, pages 221–231. IEEE, 2009.

[5] T. Asikainen, T. Mannisto, and T. Soininen. A Unified Conceptual Foundation for Feature Modelling. In *Proc. of the 10th International Software Product Line Conference*, pages 31–40. IEEE, 2006.

[6] C. Atkinson, J. Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wüst, and J. Zettel. *Component-Based Product Line Engineering with UML*. Addison-Wesley, 2002.

[7] K. Bąk, K. Czarnecki, and A. Wąsowski. Feature and meta-models in Clafer: Mixed, specialized, and coupled. In *Proc. of the 3rd International Conference on Software Language Engineering*, pages 102–122. Springer, 2010.

[8] D. S. Batory. Feature-Oriented Programming and the AHEAD Tool Suite. In *Proc. of the 26th International Conference on Software Engineering*, pages 702–703. IEEE, 2004.

[9] D. S. Batory. Feature Models, Grammars, and Propositional Formulas. In *Proc. of the 9th International Software Product Line Conference*, pages 7–20. Springer, 2005.

[10] D. S. Batory, J. R. Barnett, J. F. Garza, K. P. Smith, K. Tsukuda, B. C. Twichell, and T. E. Wise. GENESIS: An Extensible Database Management System. *IEEE TSE*, 14(11):1711–1730, 1988.

[11] D. S. Batory, D. Benavides, and A. R. Cortés. Automated analysis of feature models: challenges ahead. *Communications of the ACM*, 49(12):45–47, 2006.

[12] D. Benavides, S. Segura, and A. R. Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, 2010.

[13] T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki. Variability Modeling in the Real: A Perspective from the Operating Systems Domain. In *Proc. of the 25th IEEE/ACM Conference on Automated Software Engineering*, pages 73–82. ACM, 2010.

[14] T. J. Biggerstaff. Design Recovery for Maintenance and Reuse. *IEEE Computer*, 22(7):36–49, 1989.

[15] S. Bühne, K. Lauenroth, and K. Pohl. Why is it not Sufficient to Model Requirements Variability with Feature Models? In *Proc. of the Workshop on Automotive Requirements Engineering (AURE04). Nanzan University, Nagoya, Japan*, pages 5–12, 2004.

[16] K. Chen, W. Zhang, H. Zhao, and H. Mei. An Approach to Constructing Feature Models Based on Requirements Clustering. In *Proc. of the 13th IEEE International Conference on Requirements Engineering*, pages 31–40. IEEE, 2005.

[17] L. Chen and M. Babar. A systematic review of evaluation of variability management approaches in software product lines. *Information and Software Technology*, 53:344–362, 2011.

[18] A. Classen, Q. Boucher, and P. Heymans. A Text-based Approach to Feature Modelling: Syntax and Semantics of TVL. *Science of Computer Programming*, 76(12):1130–1143, 2010.

[19] A. Classen, P. Heymans, and P.-Y. Schobbens. What's in a Feature: A Requirements Engineering Perspective. In *Proc. of the 11th International Conference on Fundamental Approaches to Software Engineering*, LNCS 4961/200, pages 16–30. Springer, 2008.

[20] A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, and J.-F. Raskin. Model checking lots of systems: efficient verification of temporal properties in software product lines. In *Proc. of the 32nd ACM/IEEE International Conference on Software Engineering*, pages 335–344. ACM, 2010.

[21] K. Czarnecki and M. Antkiewicz. Mapping Features to Models: A Template Approach Based on Superimposed Variants. In *Proc. of the 4th International Conference on Generative Programming and Component Engineering*, pages 422–437. ACM, 2005.

[22] K. Czarnecki, T. Bednasch, P. Unger, and U. W. Eisenecker. Generative Programming for Embedded Software: An Industrial Experience Report. In *Proc. of the ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering*, pages 156–172. ACM, 2002.

[23] K. Czarnecki and U. W. Eisenecker. *Generative Programming: Methods, Techniques, and Applications*. Addison-Wesley, 2000.

[24] K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–646, 2006.

[25] K. Czarnecki, S. Helsen, and U. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10:7–29, 2005.

[26] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10:143–169, 2005.

[27] K. Czarnecki and K. Pietroszek. Verifying feature-based model templates against well-formedness OCL constraints. In *Proc. of the 5th International Conference on Generative Programming and Component Engineering*, pages 211–220. ACM, 2006.

[28] D. Dhungana, P. Grünbacher, and R. Rabiser. The DOPLER Meta-Tool for Decision-Oriented Variability Modeling: A Multiple Case Study. *Automated Software Engineering*, 18(1):77–114, 2011.

[29] D. Dhungana, P. Grünbacher, R. Rabiser, and T. Neumayer. Structuring the Modeling Space and Supporting Evolution in Software Product Line Engineering. *Journal of Systems and Software*, 83(7):1108–1122, 2010.

[30] E. Dolstra, G. Florijn, M. de Jonge, and E. Visser. Capturing timeline variability with transparent configuration environments. In *International Workshop on Software Variability Management*. ICSE Workshop, 2003.

[31] European Software Institute Spain and IKV++ Technologies AG Germany. *MASTER: Model-driven Architecture inSTrumentation, Enhancement and Refinement*, 2002.

[32] H. Gomaa. *Designing Software Product Lines with UML*. Addison-Wesley, 2005.

[33] I. Groher and M. Völter. Aspect-Oriented Model-Driven Software Product Line Engineering. *Transactions on Aspect-Oriented Software Development*, 6:111–152, 2009.

[34] P. Grünbacher, D. Dhungana, N. Seyff, M. Quintus, R. Clotet, X. Franch, L. Lopez, and J. Marco. Goal and Variability Modeling for Service-oriented System: Integrating i* with Decision Models. In *Proc. of the Workshop on Software and Services Variability Management*, pages 99–104. Helsinki Unversity of Technology, 2007.

[35] O. Haugen, B. Moller-Pedersen, J. Oldevik, G. Olsen, and A. Svendsen. Adding Standardized Variability to Domain Specific Languages. In *Proc. of the 12th International Software Product Line Conference*, pages 139–148, Limerick, Ireland, 2008. IEEE.

[36] F. Heidenreich, J. Kopcsek, and C. Wende. FeatureMapper: mapping features to models. In *Proc. of the 30th International Conference on Software Engineering, ICSE Companion*, pages 943–944. ACM, 2008.

[37] P. Heymans, P.-Y. Schobbens, J.-C. Trigaux, Y. Bontemps, R. Matulevicius, and A. Classen. Evaluating formal properties of feature diagram languages. *IET Software*, 2(3):281–302, 2008.

[38] A. Hubaux, A. Classen, and P. Heymans. Formal modelling of feature configuration workflows. In *Proc. of the 13th International Software Product Line Conference*, pages 221–230. ACM, 2009.

[39] A. Hubaux, A. Classen, M. Mendonça, and P. Heymans. A Preliminary Review on the Application of Feature Diagrams in Practice. In *Proc. of the 4th International Workshop on Variability Modelling of Software-Intensive Systems*, pages 53–59. University Duisburg-Essen, 2010.

[40] A. Hubaux, T. T. Tun, and P. Heymans. Separation of

concerns in feature diagram languages: A systematic survey. 2011. Under review.

[41] M. Janota and G. Botterweck. Formal Approach to Integrating Feature and Architecture Models. In *Proc. of the 11th International Conference on Fundamental Approaches to Software Engineering*, pages 31–45. Springer, 2008.

[42] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical report, CMU/SEI-90TR-21, 1990.

[43] K. Kang, J. Lee, and P. Donohoe. Feature-Oriented Product Line Engineering. *IEEE Software*, 19(4):58–65, 2002.

[44] C. Kästner. CIDE: Decomposing Legacy Applications into Features. In *Proc. of the 11th International Software Product Line Conference, vol 2.*, pages 149–150. IEEE, 2007.

[45] M. D. Lubars. Wide-Spectrum Support for Software Reusability. In *Proc. of the Workshop on Software Reusability and Maintainability*. National Institute of Software Quality and Productivity, 1987.

[46] J. Mansell and D. Sellier. Decision Model and Flexible Component Definition Based on XML Technology. In *Proc. of the 5th International Workshop on Software Product Family Engineering*, pages 466–472. Springer, 2003.

[47] S. Maoz, J. O. Ringert, and B. Rumpe. Semantically Configurable Consistency Analysis for Class and Object Diagrams. In *Proc. of the 14th International Conference on Model Driven Engineering Languages and Systems*, pages 153–167. Springer, 2011.

[48] M. Mendonca and D. Cowan. Decision-making coordination and efficient reasoning techniques for feature-based configuration. *Science of Computer Programming*, 75(5):311–332, 2009.

[49] J. M. Moore and S. C. Bailin. The KAPTUR Environment: An Operations Concept. Technical report, CTA Incorporated, 1989.

[50] J. M. Neighbors. The Draco approach to constructing software from reusable components. *IEEE TSE*, 10(5):564–574, 1984.

[51] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, 2005.

[52] R. Rabiser, P. Grünbacher, and D. Dhungana. Supporting Product Derivation by Adapting and Augmenting Variability Models. In *Proc. of the 11th International Software Product Line Conference*, pages 141–150. IEEE, 2007.

[53] R. Rabiser, R. Wolfinger, and P. Grünbacher. Three-level Customization of Software Products Using a Product Line Approach. In *Proc. of the 42nd Annual Hawaii International Conference on System Sciences*, pages 1–10. IEEE, 2009.

[54] M.-O. Reiser and M. Weber. Managing Highly Complex Product Families with Multi-Level Feature Trees. In *Proc. of the 14th IEEE International Conference on Requirements Engineering*, pages 149–158, Minneapolis, USA, 2006. IEEE.

[55] M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow. Extending Feature Diagrams with UML Multiplicities. In *Proc. of the 6th World Conference on Integrated Design & Process Technology*, 2002.

[56] H. Schirmeier and O. Spinczyk. Challenges in Software Product Line Composition. In *Proc. of the 42nd Annual Hawaii International Conference on System Sciences*, page 7. IEEE, 2009.

[57] K. Schmid and H. Eichelberger. From Static to Dynamic Software Product Lines. In *Proc. of the Workshop on Dynamic Software Product Lines (DSPL) at SPLC 2008*, volume 2, pages 33–38. Lero Tech Report, 2008.

[58] K. Schmid and H. Eichelberger. Model-Based Implementation of Meta-Variability Constructs: A Case Study using Aspects. In *Proc. of the Second International Workshop on Variability Modelling of Software-Intensive Systems*, number 22 in ICB-Research Report, pages 63–71. University Duisburg-Essen, 2008.

[59] K. Schmid and I. John. A Customizable Approach to Full-Life Cycle Variability Management. *Science of Computer Programming*, 53(3):259–284, 2004.

[60] K. Schmid, R. Rabiser, and P. Grünbacher. A Comparison of Decision Modeling Approaches in Product Lines. In *Proc. of the Fifth International Workshop on Variability Modelling of Software-Intensive Systems*, pages 119–126. ACM, 2011.

[61] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. Bontemps. Feature Diagrams: A Survey and a Formal Semantics. In *Proc. of the 14th IEEE International Conference on Requirements Engineering*, pages 139–148, Minneapolis, USA, 2006. IEEE.

[62] J. Sincero and W. Schröder-Preikschat. The Linux Kernel Configurator as a Feature Modeling Tool. In *Proc. of the 1st Workshop on Analyses of Software Product Lines (ASPL'08) at SPLC 2008*, pages 257–260, Limerick, Ireland, 2008. Lero.

[63] M. Sinnema and S. Deelstra. Classifying variability modeling techniques. *Information and Software Technology*, 49(7):717–739, 2006.

[64] Software Productivity Consortium Services Corporation, Technical Report SPC-92019-CMC. *Reuse-Driven Software Processes Guidebook, Version 02.00.03*, 1993.

[65] R. Stoiber and M. Glinz. Supporting Stepwise, Incremental Product Derivation in Product Line Requirements Engineering. In *Proc. of the 4th International Workshop on Variability Modelling of Software-Intensive Systems*, ICB-Research Report, pages 77–84. University Duisburg-Essen, 2010.

[66] R. Tartler, D. Lohmann, J. Sincero, and W. Schröder-Preikschat. Feature consistency in compile-time-configurable system software: facing the Linux 10000 feature problem. In *EuroSys'11*, pages 47–60. ACM, 2011.

[67] A. van der Hoek. Design-Time Product Line Architectures for Any-Time Variability. *Science of Computer Programming*, 53(30):285–304, 2004.

[68] B. Veer and J. Dallaway. The eCos Component Writer's Guide. Seen Mar. 2010 at ecos.sourceware.org/ecos/docs-latest/cdl-guide/cdl-guide.html.

[69] D. Weiss and C. Lai. *Software Product-Line Eng.: A Family-Based Software Development Process*. Addison-Wesley, 1999.

[70] D. Weiss, J. Li, H. Slye, and H. Sun. Decision-Model-Based Code Generation for SPLE. In *Proc. of the 12th International Software Product Line Conference*, pages 129–138. IEEE, 2008.

[71] R. Zippel and contributors. kconfig-language.txt. seen in the kernel tree at kernel.org, 2011-05/01.