Feature Models are Views on Ontologies

Krzysztof Czarnecki, Chang Hwan Peter Kim
University of Waterloo, Canada
Generative Software Development Group
{kczarnec, chpkim}@swen.uwaterloo.ca

Karl Trygve Kalleberg
University of Bergen, Norway
Department of Computer Science
karltk@ii.uib.no

Abstract

Feature modeling has been proposed as an approach for describing variable requirements for software product lines. In this paper, we explore the relationship between feature models and ontologies. First, we examine how previous extensions to basic feature modeling move it closer to richer formalisms for specifying ontologies such as MOF and OWL. Then, we explore the idea of feature models as views on ontologies. Based on that idea, we propose two approaches for the combined use of feature models and ontologies: view derivation and view integration. Finally, we give some ideas about tool support for these approaches.

1 Introduction

Feature modeling is a domain modeling technique, which has generated a lot of interest in the software product line (SPL) community. Feature models can be used for modeling common and variable requirements of products in a SPL, scoping SPLs, and product configuration and derivation.

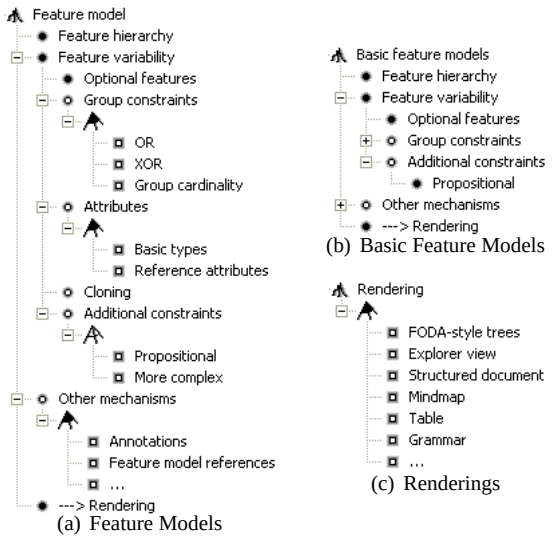
Another domain modeling technique being used in software engineering is ontology modeling, such as using OWL [23] or profiled UML class diagrams [18]. Ontology modeling is clearly also of interest to product lines; however, the relation between feature modeling and ontology modeling is not well understood today. In particular, a number of extensions to the original feature modeling notation from Feature Oriented Domain Analysis (FODA) [14] have been proposed, such as attributes and cloning, that seem to be pushing the descriptive power of feature modeling to that of ontologies.

In this paper, we explore the relationship between feature modeling and ontology modeling in two respects. First, we analyze the notational spectrum from basic feature modeling to ontology modeling. This analysis provides a framework for discussing the boundary between feature models and ontologies. Secondly, we analyze the idea of feature models as views on ontologies. We identify different mapping patterns and show how the mapping can be specified using configurable Object Constraint Language (OCL) [19] constraints. This mechanism can be used for scoping and configuring ontologies from different viewpoints. Furthermore, we suggest directions for feature modeling methods based on view projection and view integration and tool support for these methods. We believe that the paper sheds new light on the nature of feature modeling and makes a step towards establishing a comprehensive feature modeling methodology.

The remainder of the paper is organized as follows. In Section 2, we describe two essential components of a feature model: hierarchy and variability. In Section 3, we discuss the notational spectrum of feature models and ontologies, which serves as a framework for identifying the boundary between the two concepts. Then in Section 4, we propose the notion of a feature model as a view on an ontology, define the mapping, and discuss the ideas through business examples modeled using REA. In Section 5, discussion and future directions are provided. Section 6 discusses related work in depth. Section 7 concludes.

2 Essence of Feature Models: Feature Hierarchy and Variability

A *feature model* is a hierarchy of features with variability. Figure 1(a) defines the concept of feature modeling notations as a feature model. *Feature hierarchy* is shown as a mandatory feature (see Table 1 for an explanation of the notation used through the paper). The primary purpose of a hierarchy is to organize a potentially large number of features into multiple levels of increasing detail. A feature model of a concept describes a set of valid feature combinations, each representing an instance of that concept. *Variability* defines what the allowed combinations of features are. Variability in a feature model is expressed through a number of mechanisms, which are shown as its subfeatures in Figure 1(a). The most basic variability mechanism is the notion of *optional features*, which is mandatory ac-



(a) Feature Models (b) Basic Feature Models (c) Renderings

Figure 1. Features of feature modeling

Symbol	Explanation
	Root feature
	Solitary feature with cardinality $[1..1]$, i.e., *mandatory* feature
	Solitary feature with cardinality $[0..1]$, i.e., *optional* feature
$[0..m]$	Solitary feature with cardinality $[0..m]$, $m > 1$, i.e., *optional clonable* feature
$[n..m]$	Solitary feature with cardinality $[n..m]$, $n > 0 \wedge m > 1$, i.e., *mandatory clonable* feature
	Grouped feature
$f(T)$	Feature f with attribute of type T
	Feature group with cardinality $\langle 1\text{–}1 \rangle$, i.e. *xor*-group
	Feature group with cardinality $\langle 1\text{–}k \rangle$, where k is the group size, i.e. *or*-group
$\langle i\text{–}j \rangle$	Feature group with cardinality $\langle i\text{–}j \rangle$

Table 1. Symbols used in cardinality-based feature modeling

cording to Figure 1(a). More advanced variability mechanisms, such as group constraints, attributes, cloning, and additional constraints, are all optional. Additional constraints may cut across the feature hierarchy. They can be expressed in propositional logic or a richer formalism such as first-order predicate logic or some weak constraint formalism. It is important to note that the feature hierarchy has a double role in a feature model. First and most importantly, it is a structuring mechanism, but it also contributes to variability. Specifically, a feature implies its parent, and a mandatory feature is additionally implied by its parent. A feature notation may also support other mechanisms, such as annotations and feature model references. Annotations are useful to capture additional information such as priorities or relations between features. Feature model references allow splitting large feature models into smaller ones. *Rendering* in Figure 1(a) is an example of a reference to the model in Figure 1(c).

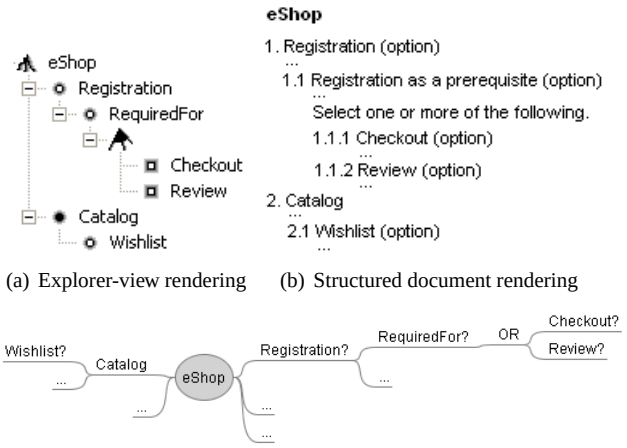
Feature models may be rendered in different forms, some of which are listed in Figure 1(c). *FODA-style trees* refers to notational styles resembling the original FODA diagrams [14]. *Explorer-view* style is the rendering used in this paper. In general, any rendering style for hierarchies is applicable to feature models, such as structured documents with sections and subsections, mindmaps, and hierarchical tables. For example, consider Figure 2, which shows a basic feature model rendered in explorer-view (Figure 2(a)), structured document view (Figure 2(b)) and mindmap view (Figure 2(c)).

The essence of a feature model is its embodiment of a hierarchy and description of variability, rather than its rendering. Indeed, artifacts that may not be commonly considered as feature models, such as those in Figure 2(b) and Figure 2(c), can arguably be considered as feature models *in disguise*. Degenerate cases include artifacts without hierarchy, such as flat lists and tables, or without variability, such as a requirements outline with no variability.

3 Notational Spectrum Between Feature Models and Ontologies

A commonly accepted definition of an *ontology* in information sciences and engineering is that by Gruber, who defines an ontology as "an explicit specification of conceptualization" [12]. An ontology represents the semantics of concepts and their relationships using some description language, which is most often coupled with first-order logic or its decidable fragment. Basic feature modeling is also a concept description technique, but is captured logically as a propositional formula [2]. In terms of descriptive power, ontologies are clearly richer and more powerful.

The space between the two techniques is not empty, however. A notational and semantic spectrum exists between basic feature models and ontologies, and this warrants exploration. Various extensions to feature modeling for increasing the descriptive power have been proposed. These extensions bring feature models closer to that of a complete ontology formalism, and make out the spectrum in Figure 3. The main considered extensions to basic feature modeling include feature attributes, cloning, and reference attributes [5]. We discuss each extension in turn, focusing on the motivation for each extension, and how it extends the constraint mechanism of basic feature models. We also comment on whether the discussed extensions increase the *succinctness* of the notation, i.e., add syntactic sugar, or



(a) Explorer-view rendering (b) Structured document rendering

(c) Mindmap rendering

Figure 2. Different renderings of a basic feature model

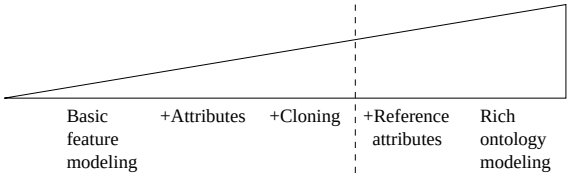


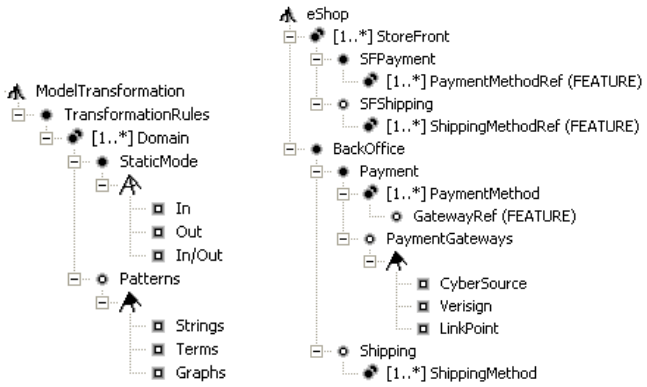
Figure 3. Increasing descriptive power of feature modeling

the expressiveness, i.e., allow expressing new mathematical concepts [10], or both.

Basic feature modeling. Basic feature models can be thought of as a hierarchy plus a propositional formula. Figure 1(b) defines this class of notations as a specialization of the feature model in Figure 1(a). An example of a basic feature model representing the requirements for a family of electronic shops is shown in Figure 6(b). A basic feature model can contain additional constraints as propositional formulas, such as `Backorder`⇒`AccountRequired`. Group constraints are syntactic sugar with respect to additional constraints since they can be captured by a propositional formula. The strength of basic feature models is their simplicity and intuitiveness. The hierarchy naturally helps the modeler explore a problem from a given perspective by allowing organization of features into levels of increasing detail. Feature optionality and feature groups allow the explicit modeling of variability, while the hierarchy implicitly encodes implications. Constraint solvers based on binary-decision diagrams (BDDs) can be used to efficiently reason over basic feature models, since a propositional formula can precisely capture the variability of such feature models. Such constraint solvers enable, among others, choice propagation and auto-completion of configurations of feature models, as described in [8].

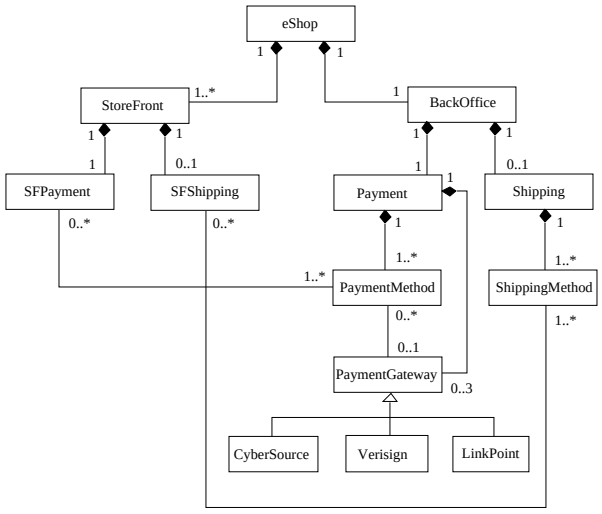
Attributes. A useful extension is to allow *feature attributes*. Here we consider attributes of basic types such as numbers and strings. Feature attributes have been particularly useful in the context of embedded software [5]. For example, Weiland et al. [27] use feature models to configure a Matlab/Simulink model of automotive engine control software. In that application, different characteristic parameters of the Matlab/Simulink model such as engine mass, target operating temperatures, etc., are exposed as feature attributes in the corresponding feature model. In the context of business applications, feature attributes would seem to be useful for modeling non-functional requirements such as server throughput or capacity. However, such requirements can usually be modeled by a few alternative features, such as "capacity up to 1,000 users" or "capacity up to 100,000 users." Adding attributes invites complex constraints. While constraint checking is usually not a problem, constraint solving can quickly turn into a complicated optimization problem in many cases, for example, for continuous or infinite domains. When an attribute value, like the user capacity in an electronic commerce product line, is dependent on other features, like thresholds of storage, traffic, and server speed, appropriate solutions may be computationally difficult to find. Such constraint problems can often be solved with specialized tools and algorithms, but if the constraints become too complex, it may indicate that the problems are outside the scope of feature modeling.

Cloning. Cloning allows a feature to be replicated during configuration. The number of possible replications is constrained by the feature cardinality, which must have an upper bound greater than one for cloning. Although cloning is rather an exotic mechanism in feature modeling, there are clearly cases where cloning is useful. As an example, Figure 4(a) shows a fragment of a feature model that characterizes a family of model transformation approaches [6]. According to the model, each approach supports transformation rules, which in turn may have one or more domains. Thus, the model can account for approaches with multiple domains, each having a different selection of features. For example, a graph-based model-to-model transformation approach may have just one in/out-domain with graph patterns, whereas a template-based model-to-text approach may have two domains: an in-domain with graph patterns and an out-domain with string patterns. Bounded cloning can be expressed by a propositional formula; unbounded cloning means increased expressiveness. In practice, one can always make cloning bounded by putting a sufficiently large number on the upper bound. However, adding cloning potentially invites a new class of constraints, namely constraints over sets of clones [8]. Simple constraints such as



(a) Cloning in feature models (b) Reference attributes in feature models

Figure 4. Cloning and reference attributes in feature models




```
context Payment inv:
  self.paymentGateway->forAll(gateway1, gateway2: PaymentGateway |
        gateway1<>gateway2 implies gateway1.getType() <> gateway2.getType())
```

Figure 5. UML model of the e-shop family from Figure 4(b)

constraining the size of a set of clones to be within two constants are rather unproblematic. However, constraints such as requiring a clone of feature *A* for every clone of feature *B* essentially model associations. Such relationships are the strength of rich ontology modeling and, thus, they are arguably outside the scope of feature modeling.

Reference attributes. A reference attribute is an attribute that may point to another feature in a feature configuration. Reference attributes are of interest only in the presence of clonable features. Figure 4(b) shows an excerpt from a feature model describing a family of electronic shops from an earlier paper [8]. The feature model makes a heavy use of feature cloning. The back office part of the feature model allows creating multiple payment methods and shipment methods, which need to be referred to from the store front part. This is modeled using the feature references of `PaymentMethodRef` and `ShippingMethodRef`. Additional constraints are needed to restrict the scope of features the attributes can refer to. For example, the constraint that the attribute of `PaymentMethodRef` should only point to clones of `PaymentMethod` can be stated using OCL as follows [8]:

```
context PaymentMethodRef inv:
EShop.BackOffice.Payment.PaymentMethod->includes(att)
```

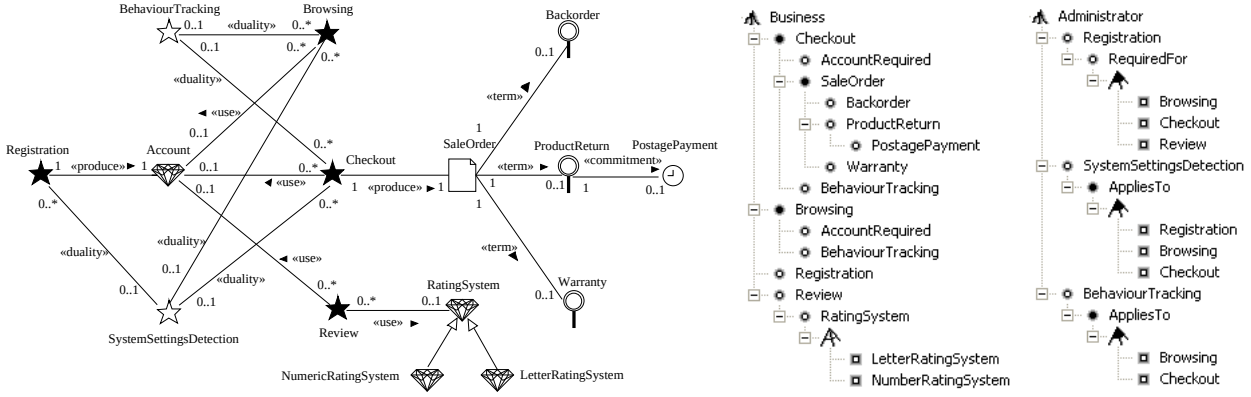
In essence, reference attributes can be viewed as a poor man's associations. A more elegant model using a richer ontology notation will be shown next. Consequently, we currently feel that reference attributes are outside of the scope of feature modeling, as indicated by the dashed line in Figure 3.

Rich ontology modeling. Ontologies may be described in formal languages based on first-order logic (FOL) such as KIF [9] or OWL [23]. Alternatively, ontologies may also be represented using UML class diagrams and OCL.[1] This is commonly done in software engineering. In the UML approach, concepts are represented as classes and the ontological relationships are represented using UML relationships. Domain-specific semantics can be provided by a specialized UML profile, as will be illustrated in Section 4.1. In this paper, we only consider ontologies represented using UML class diagrams, although we believe that the presented ideas can be generalized to other ontology formalisms, too. Figure 5 shows a UML model of the e-shop family from Figure 4(b). The use of associations and specializations results in a more intuitive model.

4 Feature Models as Views on Ontologies

Firstly, since feature models, on the left side of the spectrum in Figure 3, are less powerful than ontologies, on the right side of the spectrum, feature models form a notational subset of ontologies. Secondly, in terms of modeling philosophy, in feature modeling, a concept is described by first setting its scope and hierarchically adding its details in a top-down fashion. For example, for a feature model concerning the "business" concept, the scope is first restricted to "business processes" before the details are constructed. On the other hand, in ontology modeling, a concept is described by adding its details and implicitly defining the scope of the concept through the details in a botttom-up

[1] OCL is more expressive than FOL, e.g., transitive closures over object relations can be expressed in OCL but not in FOL [17].



(a) Ontology (b) Business FM (c) Admin FM

Figure 6. An REA ontology and its views

fashion. For example, the scope of "business" concept is defined through what exists in the ontology, such as business resources, events, and agents. As a result, feature models are more likely to describe concepts more specialized than those of ontologies. Thirdly, feature models and ontologies, as conceptual models, play the similar role of providing meta-information, such as metamodels, common vocabulary, and variability information, to design models throughout software development. These three points form a basis for considering feature models as *views* on ontologies. This view relationship is precisely defined as a mapping characterized by syntactic correspondence and semantics. We use the following example to explain the mapping.

4.1 REA Example

The Resource Event Agent (REA) framework [13] is an established system of guidelines, rules, patterns, and schemas for constructing an ontology of business concepts. The framework allows the economic world to be modeled from the perspective of the enterprise in terms of duality of economically beneficial and detrimental events representing exchanges and transformations of resources. The framework claims at least two clear benefits. Firstly, the framework, which prescribes business modeling patterns, is designed to facilitate elicitation and complete specification of business requirements. For example, both the provider and the the receiver of a resource must be specified for an event. Secondly, the framework allows the modeler to first model the logical structure and causal dependencies and to defer implementation-dependent details including ordering of events that are typically the focus of traditional business modeling techniques like business process modeling and workflow modeling. The resulting business models are likely to be less brittle under evolution.

Figure 6(a) shows a partial ontology of system capabilities and entities of a B2C e-commerce product line modeled according to REA. An REA profile with intuitive icons is used. Resources, which are economically scarce, are represented using diamonds. For example, an `Account` in an electronic shop (eShop) is a resource as it is a valuable commodity to eShop owners. Economic events, represented using stars, are involved in *conversion* and *exchange* processes. In a conversion process, economic events *use* or *consume* some resources to *produce* other resources. A consumption of a resource makes it disappear, while a usage does not. For example, `Registration` event uses input data like name and address to produce an `Account`. In turn, `Browsing`, `Checkout`, and `Review` events may use `Account` to produce internet traffic, `SaleOrder`s, and product reviews respectively.[2] In an exchange process, resources flow into and out of a set of economic events related through duality. Duality is an ontological constraint between two events enforcing an exchange of resources. For example, `SystemSettingsDetection` may complete registration information, search input fields, and checkout data with the information residing in the client's computer and thus increase immediate usability of `Registration`, `Browsing`, and `Checkout` events for the customers in exchange for consumption of enterprise system resources, like the development and computing power required to perform the system settings detection.[3] Similarly, `BehaviourTracking`, which may increase the usability of `Browsing` and `Checkout` in a less imminent way, is related through duality to the two processes. In duality, events are distinguished as being economically beneficial or detrimental from the perspective of an agent

[2] The word *may* is used to emphasize the optionality, which is indicated by the lower bound 'zero' of an association.

[3] Note that while a system capability may be modeled as discrete events or a continuous event, to keep the discussion more clear, we take the latter view and assume that there can be at most one instance of the capability. As a result, all the associations to a system capability have cardinality 0..1.

as *increment* and *decrement* events respectively. Increment and decrement events are represented using filled stars and empty stars respectively. *Customer* and *enterprise* are the only agents in the example and the increment/decrement aspect is modeled from the perspective of the enterprise. For example, `Registration`, `Browsing`, `Checkout`, and `Review` events are increment events because they increase membership base, traffic, sale orders, and consumer community base respectively, which positively increase the enterprise's net resources. Agents are always involved in conversion or exchange processes as providers or recipients of resources. Agents, as well as some resources related to the events, are not explicitly modeled to keep the example scoped.

Business concepts beyond the basic resources, events, and agents can be modeled by extending the REA framework. A *sale order* is such a concept.[4] In Figure 6(a), a `SaleOrder` is a legal contract, represented by a document. A legal contract states one or more commitments the involved agents must fulfill, as well as additional terms. A commitment, represented by a clock, is an event that must occur in the future. A term, represented by a magnifying glass, is a conditional commitment. In a `SaleOrder`, the enterprise has the committment to provide sale of the product and to receive cash receipt for it, while the customer has the committment to provide cash receipt and receive sale of the product. But more interestingly, a `SaleOrder` may involve `Backorder`, `ProductReturn`, or `Warranty` terms, which describe the required committments of the enterprise should a product be not immediately available, wished to be returned, or defective respectively. For `ProductReturn`, the enterprise may be committed to provide the postage payment required for shipping the undesired product back. In addition to advanced business concepts, general ontology facilities, including inheritance, may also be incorporated. A `Review` may or may not use a `RatingSystem`, which can be either a `NumericRatingSystem` or a `LetterRatingSystem`.

The business feature model, in Figure 6(b), views the ontology in Figure 6(a) in terms of customer-concerning activities, which can be broken down into `Checkout`, `Browsing`, `Registration`, and `Review`. The administrator feature model, in Figure 6(b), views the ontology in terms of system administration concerns, namely, `Registration`, `SystemSettingsDetection`, and `BehaviourTracking`.

[4]The sale order business concept is presented as an advanced business pattern in [13]. The business pattern is considerably simplified in this paper for the sake of clarity of discussion.

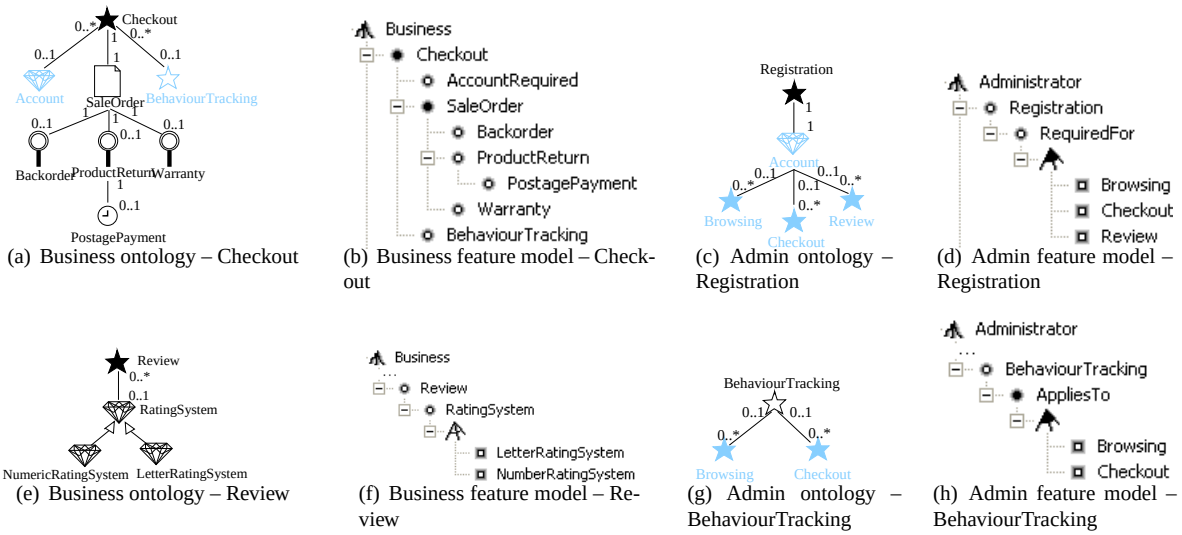
4.2 Syntactic correspondence

Syntactic correspondence between a feature model and an ontology establishes traceability links between feature model and ontology elements. Traceability is useful for increasing understanding of mapping and for representing simple constraints like existential dependencies. In general, an arbitrary set of feature elements, i.e., features and relationships (subfeature or feature group), may be mapped to an arbitrary set of ontology elements, i.e., classes and associations. There is an many-to-many association between feature elements and ontology elements, but a typical mapping is where an one-to-one mapping is used to express an existential dependency from a feature element to an ontology element. Figure 7 shows examples of such mappings between the REA ontology and its two views, which can be classified.[5] Ontology elements not involved in a mapping are specifically colored lightly.

Isomorphic mapping. `SaleOrder` is mapped to the actual `SaleOrder` class, while the subfeature relationship between `Checkout` and `SaleOrder` is mapped to the association `Checkout-SaleOrder`, as shown in Figures 7(a) and 7(b). Likewise, `Backorder`, `ProductReturn` and `Warranty` features, which describe whether every sale order must include or exclude such terms, are mapped to the corresponding ontology classes. When a feature–subfeature–feature pattern is mapped to a class–association–class pattern as in the last two cases, there is isomorphism between the feature model region and the ontology region, which means that the ontology region is modeled according to a single context. This makes sense in the last case, since terms can only exist in the context of a contract, such as a `SaleOrder`. Similarly, potential commitments, such as `PostagePayment`, make sense only in the context of their term, such as `ProductReturn`. Generally, isomorphism occurs when an ontology describes partonomy, characterization, and single inheritance. Note the structural resemblance between Figure 7(f) with its ontology counterpart Figure 7(e). Here, a feature group maps to the specialization relationship. In isomorphic mapping, subfeature cardinalities usually correspond to association cardinalities. This is evident in `SaleOrder`, `Review`, and their counterparts.

Feature-to-association mapping. In Figure 7(b), `AccountRequired` states whether or not a checkout event requires an account. This feature is mapped to the association `Checkout-Account`, not to `Account`, since the feature expresses an existential dependency on the association. If a feature is involved in a feature-to-association mapping, it is difficult to specify syntactic correspondence

[5]To define a uniform and precise syntactic and semantic mapping, dependencies must be represented as associations with stereotypes, which is trivial to do.



(a) Business ontology – Checkout (b) Business feature model – Checkout (c) Admin ontology – Registration (d) Admin feature model – Registration

(e) Business ontology – Review (f) Business feature model – Review (g) Admin ontology – BehaviourTracking (h) Admin feature model – BehaviourTracking

Figure 7. Excerpts of the REA ontology, and its Feature Model Views

of its children features, as the children features refine the semantics of the feature-to-association mapping. This issue is explored in detail in the next section.

Different views. Different views may syntactically correspond to the same region of ontology through different traversals. For example, Figures 7(a) and 7(b) show `Account` and `BehaviourTracking` being viewed from `Checkout`. Figures 7(c), 7(d), 7(g), and 7(h) show `Checkout` being viewed from `Account` and `BehaviourTracking`. Such opposite traversals mean that while each feature model may provide overlapping restrictions, each can always provide a unique restriction. For example, while the `BehaviourTracking` capability may be turned on or off only for `Checkout` events in Figure 7(b), `BehaviourTracking` capability in Figure 7(h) may be turned on or off for all events.

4.3 Semantics of mapping

Semantically, a feature model, through its configurations, represents the set of viewpoint restrictions that can be applied to an ontology. The restricted ontology must represent at least one valid set of ontology individuals. We propose a natural mechanism for ontology restriction: a feature model represents a family of ontology constraints, in the form of OCL embedded with feature identifiers (IDs). For a given feature configuration, simply, the feature variables referenced by the feature IDs, which are typically Boolean, are replaced by their values in the OCL constraints. We first discuss some simple mapping semantics and then discuss more complex semantics.

Simple semantics. Table 2 shows some of the applicable constraints on Figure 6(a). A feature ID, shown within `<< >>` as an XPath-like expression of feature name hierarchy, is replaced with the Boolean value of the feature at configuration time. The example constraints represent existential dependencies between features and ontology elements, which can be either positive, negative, or full [7]. In a positive existential dependency, if a feature is selected, then the corresponding ontology element must exist. For example, constraint 1 states that if the business viewpoint indicates that an account is required for checkout, then every checkout event must have an associated account. Note that we have not included here the same decision from the admin viewpoint, i.e. `Checkout` in Figure 7(d). To avoid redundancy, we only include one of the features since we're assuming that both viewpoints must agree on the capability. Before configuring the ontology constraints, feature constraints between feature models must be placed and a valid configuration of the feature models as a whole must exist. In a negative existential dependency, if a feature is eliminated, then the corresponding ontology element must not exist. For example, constraint 2 shows a case where a class, `Registration`, cannot have any individuals when the corresponding feature is eliminated. In a full existential dependency, the positive and the negative existential dependency semantics are combined. For example, constraint 3 shows a situation where the existence of an association, i.e. `SaleOrder.backOrder`, is dependent on the existence of a feature, `Backorder` and vice versa. Semantics more complex than existential dependencies may also be expressed. For example, constraint 4 shows a type restriction on `Review.ratingSystem`.

As a feature model becomes deeper, the semantics of mapping become more complicated. We can classify the

No.	Constraint
1	**context** *Checkout* **inv**: <<Business//AccountRequired>> **implies** (self.account->size() <> 0)
2	**context** *Registration* **inv**: (not <<Business/Registration>>) **implies** (self->size() = 0)
3	**context** *SaleOrder* **inv**: <<Business//Backorder>> = (self.backorder->size() <> 0)
4	**context** *Review* **inv**: (not <<Business//LetterRatingSystem>>) **implies** ((self.ratingSystem->size() <> 0) **implies** (self.ratingSystem.oclIsTypeOf(LetterRatingSystem) = **false**))

Table 2. Constraints

(a) Independent Ontology (b) Independent Feature Model

Figure 8. Independent Ontology and Feature Model

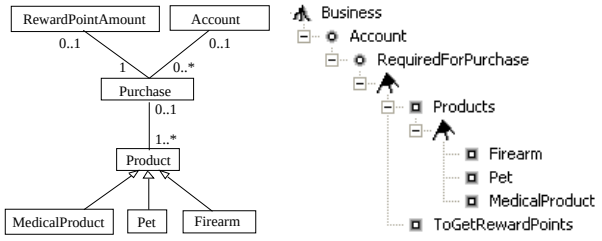
more complex semantics into two categories: those where a parent does not require its children in the mapping and those where a parent does require them.

Children-independent semantics. Figure 8(a) and Figure 8(b) show an ontology involving `Purchase`, `Payment` and `Account` and a feature model restricting it. `Payment.account` has the cardinality [0..1] because we assume that an account is not required for payment. The feature model states that allowing multiple payments in a purchase is optional and allowing such payments from different accounts is optional. Note that while `FromDifferentAccounts` only makes sense if `MultiplePayments` are allowed, `MultiplePayments` makes sense independently of `FromDifferentAccounts`. The mapping semantics of the parent feature are independent of those of the child feature. In these cases, we can write separate constraints for the parent feature and the child feature:

```
context Purchase inv:
    (not <<MultiplePayments>>) implies
         (self.payment->size() <= 1)

context Purchase inv:
   (<<MultiplePayments>> and
     (not <<FromDifferentAccounts>>)) implies
       (self.payment->forAll(payment1,
           payment2: Payment |
           payment1<>payment2 implies
           payment1.account = payment2.account))
```

Children-dependent semantics. In other cases, the mapping semantics of the parent feature is incomplete without the mapping semantics of the child features. It typically occurs when a feature hierarchy represents a hierarchy of refinements of an ontology association. For example, Figure 9(a) shows an ontology where a purchase is on multiple



(a) Dependent Ontology (b) Dependent Feature Model

Figure 9. Dependent Ontology and Feature Model

products of varying types, where a purchase may or may not yield a reward point amount, and where a purchase may or may not have an associated account. Figure 9(b) describes whether or not accounts may exist in the system and what account-requiring purchases are.

Purchases must be restricted using the configurations of features below `RequiredForPurchase`. Constraints for children-dependent semantics can become complex if there is complex variability in the subtree. It may require incrementally building the constraint with variability encoded and/or breaking down a large constraint without variability consideration and inserting variability. We show the latter approach. First, the constraint is specified without inserting variability:

```
context Purchase inv:
    (self.product.select(p: Product |
        p.oclIsTypeOf(MedicalProduct) or
        p.oclIsTypeOf(Pet) or
        p.oclIsTypeOf(Firearm))->notEmpty() or
           self.rewardPointAmount->notEmpty())
              implies (self.account->notEmpty())
```

Then we insert variability by breaking down the constraint, for example, into understandable functions:

```
context Purchase inv:
  (self.isProductApplicable() or
      self.isRewardPointApplicable())
implies self.account->notEmpty()

context Purchase::isProductApplicable(): Boolean
post: result = (product.select(
   p: Product | (p.oclIsTypeOf(MedicalProduct) and
     <<MedicalProduct>>) or
      (p.oclIsTypeOf(Pet) and <<Pet>>>) or
        (p.oclIsTypeOf(Firearm) and
         <<Firearm>>))->notEmpty())

context Purchase::isRewardPointApplicable(): Boolean
post: result = rewardPointAmount->notEmpty()
and <<ToGetRewardPoint>>
```

Note that `AccountRequired` and `ProductType` features are not specified in the constraints. While the Boolean values of these two features influence the Boolean values of the features specified in the constraints due to the propositional constraints in the feature model, the two

features do not need to be specified in the constraints. This suggests that children-dependent semantics can be kept as simple as possible by specifying constraints using a minimal set of features in the hierarchy, while variability can be specified in a very flexible manner through variability in the hierarchy. Nevertheless, it seems that children-dependent semantics are more difficult to handle than children-independent semantics as they require understanding the entire hierarchy as a whole.

5 Discussion and future directions

The analysis of the relationship between feature models and ontologies are notable in three respects. Firstly, it sheds new light on the nature of feature models. Secondly, it suggests new directions for the methodology of feature modeling: view projection and view integration. Thirdly, it suggests opportunities for tool support required for feature modeling and ontology development. We discuss each of these points in turn.

5.1 Notion of feature models

The analysis of the notational spectrum ranging from basic feature models to ontologies and of feature models as views on ontology advances our understanding of feature models. The notational spectrum analysis provides a framework for discussing the discipline of feature modeling with respect to the discipline of ontology modeling. There are clearly unanswered questions. For example, while we have made some suggestions for what the boundary between feature modeling and ontology modeling may be, for example, that reference attributes are probably outside of feature models, there are remaining issues, including the expressiveness of constraints on attributes and clones. A feature modeling notation may be domain-specific: different applications may require different combinations of language features. Also, the framework may evolve, as our understanding of feature modeling has evolved since its inception.

The analysis of feature models as views on ontologies gives us a new insight on the nature of feature models in two respects. Firstly, a feature model represents a set of possible restrictions on ontologies. Secondly, the feature hierarchy provides a mechanism of imposing a perspective, and in this sense, a feature model is an outline exploring a theme through an ontology. Mappings were explored using examples, where feature models and ontologies were at similar levels of abstraction. As a result, despite some complex mappings, most of the mappings were manageable. While the mapping mechanism works for all kinds of mappings, we can imagine, for example, when ontologies are closer to implementation and feature models are closer to requirements, the mapping would become very complex and less manageable.

5.2 Towards methodology of view projection and view integration

The analysis of feature models as views suggests two potential approaches to feature modeling: view projection and view integration. In view projection, we assume that a comprehensive ontology is available or being constructed using ontology-oriented domain analysis, like REA. Feature model projection on ontology is desired for modeling from different perspectives for the purpose of scoping the domain model. In view integration, first, feature models, or outlines of requirements, are created more or less independently, with some ontology in the mind of each developer. Ontology can be used to align the views, with a particular focus on developing the overlapping parts. A hybrid approach may also be explored, where both view projection and integration are used. We leave these ideas for future work.

5.3 Tool support

Based on the ideas proposed in the previous section, we can define a wish list of tool support. For view projection, feature model construction guidance, based on existing syntax and semantics of ontologies, is desired. Various kinds of techniques, like traversals and queries, can be used not only to achieve syntactical correspondence, but also for incrementally adding mapping semantics through ontology exploration. Furthermore, constraint-solving facilities should be available for checking and computing the residual of the specialized ontology. Ideally, view projection tool capabilities should support round-tripping for the sake of usability. Tool support for view integration is harder to imagine, as it is a more manual process by nature. Synonym detection for identifying common features, simplification of semantics of mapping, and conflict detection and resolution are potential wish list items.

6 Related work

Bodies of related work can be classified into feature dependency analysis, viewpoint-oriented requirements engineering (RE), early aspects, feature-based configuration of models, encoding feature models in ontology languages, and work on semantics of feature modeling. We discuss each in turn.

Feature dependency analysis. Notable works in this area include those by Lee et al. [15] and Zhang et al. [28] Both provide classifications of feature dependencies, such as *refinement*, *usage*, and *modification*. In general, dependencies have two components: configuration semantics,

which can be captured through propositions for basic feature models, and extra semantics that are beyond feature configurations, which can be captured through annotations. While the ability to capture such dependencies is convenient, if there is an emphasis on the extra semantics, using ontology and feature models as views on ontologies seems to be more appropriate.

Viewpoint-oriented RE. There is a large body of work on viewpoints in RE [22, 11, 21]. A viewpoint sets a perspective with respect to a stakeholder's interests. A view is a projection of a problem with respect to a viewpoint. Treating feature models as views on ontologies is an example of a viewpoint-oriented approach, with the special focus on product lines. Viewpoint integration has been explored in the context of requirements engineering [20]. Understanding how this relates to our proposed view integration remains future work.

Early aspects. Early aspects [1] builds on viewpoint-oriented RE by considering crosscutting concerns and identifying candidates for implementation using Aspect-Oriented Programming (AOP) techniques. The closest work to our research is by Loughran et al. [16], which provides ways to automatically extract views from requirements according to viewpoints, which, rather than being limited to stakeholders, are more broad as they represent concepts of interest, like feature model roots. While mechanisms may differ, both our approach and this related work perform view extraction by imposing a viewpoint on a more general artifact.

Feature-based configuration of models. Three notable works exist in this area: feature-based model templates [4], UML profile for software product lines [29] and automatic specialization of state charts [26]. The model templates work explores mapping feature models to other models like UML class diagrams, which may represent business entities, and UML activity diagrams, which may represent business work flow. Mapping between feature models and ontologies is similar to feature-based model templates in the sense that the mapping gives semantics to features, and a feature model is used to configure the UML models in both cases. However, the two approaches are different in three respects. In this paper, feature models and ontologies are very close together in terms of abstraction while in the model template work, models have more implementation detail. Also, this paper considers multiple perspectives on an ontology, which was not considered in the model templates work. The mapping mechanism is different: in model templates, presence conditions are placed on actual model elements, while in this work, we use configurable ontology constraints. UML profile for software product lines models variability in class diagrams and state diagrams through stereotypes, tagged values, and structural constraints on the stereotypes and the tagged values. It is closer to the model templates work than the work described in this paper, as the models themselves are annotated, rather than their constraints. Automatic specialization of state charts also considers configurations of models, but uses partial evaluation techniques. Arguably, our technique can be considered as a partial evaluation technique because of the restriction approach rather than presence conditions. The work only considers state charts, which are more in line with mapping to implementation.

Expressing FM in ontology languages. In this line of research, basic feature models are expressed in ontology languages like OWL mainly for the purpose of using the ontology reasoning framework over feature models, for example, to check consistency [25]. There is also work to intentionally define propositional feature dependencies through an extensionally defined context for the purpose of providing automatic synchronization of feature models against changes made to the their underlying artifacts [24] using the reasoning framework. This is quite opposite to our approach, as the feature model semantics are restricted by the ontology, and not vice versa.

Work on semantics of feature models. Several authors have proposed formal semantics of feature models. Batory [2] proposed formal semantics for feature models based on propositional formulas and grammars. Bontemps et al. [3] explore the succinctness and expressiveness of feature modeling notations. However, all the notations considered here are in the left part of the spectrum in Figure 3.

7 Conclusion

In this paper, we explored the relation between feature modeling and ontology modeling by analyzing the notational spectrum from basic feature models to ontologies and the notions of feature models as views on ontologies. The notational spectrum provides a framework for exploring the boundary between feature modeling and ontology modeling. For example, based on our analysis, reference attributes seemed to be outside of the scope of feature modeling. However, the boundary is not clear, e.g., what kinds of constraints over attributes and clones should be supported, and requires further exploration. We believe that the notion of feature models as views on ontologies is appealing. The mapping can give semantics to potentially overlapping feature models, whereas feature models allows scoping and configuring ontologies from different perspectives. Furthermore, the combination of feature models and ontologies suggests a spectrum of feature modeling methods, from view projection to view integration, and gives ideas for tool support utilizing query and constraint mechanisms.

Acknowledgments

The authors would like to thank Ulrich W. Eisenecker, Anders Hessellund, and Dan Berry for their valuable comments on an earlier draft of the paper.

References

[1] E. Baniassad, P. C. Clements, J. Araujo, A. Moreira, A. Rashid, and B. Tekinerdogan. Discovering early aspects. *IEEE Software*, 23(1):61–70, Feb. 2006.

[2] D. Batory. Feature models, grammars, and propositional formulas. Technical Report TR-05-14, University of Texas at Austin, Texas, Mar. 2005.

[3] Y. Bontemps, P. Heymans, P.-Y. Schobbens, and J.-C. Trigaux. Semantics of FODA feature diagrams. In T. Männistö and J. Bosch, editors, *Proceedings SPLC 2004 Workshop on Software Variability Management for Product Derivation – Towards Tool Support*, pages 48–58. Technical Report 6 – HUT-SoberIT-C6, Aug. 2004. Available from `http://www.soberit.hut.fi/SPLC-DWS/`.

[4] K. Czarnecki and M. Antkiewicz. Mapping features to models: A template approach based on superimposed variants. In R. Glück and M. Lowry, editors, *GPCE'05*, volume 3676 of *LNCS*, pages 422–437. Springer, 2005.

[5] K. Czarnecki, T. Bednasch, P. Unger, and U. W. Eisenecker. Generative programming for embedded software: An industrial experience report. In D. Batory, C. Consel, and W. Taha, editors, *Proceedings of GPCE'02, Pittsburgh, October 6-8, 2002*, volume 2487 of *LNCS*, pages 156–172, Heidelberg, Germany, 2002. Springer-Verlag.

[6] K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3), 2006. To appear.

[7] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration through specialization and multi-level configuration of feature models. *Software Process Improvement and Practice*, 10(2):143–169, 2005. `http://swen.uwaterloo.ca/~kczarnec/spip05b.pdf`.

[8] K. Czarnecki and C. H. P. Kim. Cardinality-based feature modeling and constraints: a progress report. In *International Workshop on Software Factories*, San Diego, California, Oct 2005. Paper available at `http://softwarefactories.com/workshops/OOPSLA-2005/SoftwareFactoryWorkshopAnnouncement.htm`.

[9] DARPA. Knowledge interchange format. `http://logic.stanford.edu/kif/kif.html`.

[10] M. Felleisen. On the expressive power of programming languages. In N. Jones, editor, *ESOP '90 3rd European Symposium on Programming, Copenhagen, Denmark*, volume 432, pages 134–151. Springer-Verlag, New York, N.Y., 1990.

[11] A. Finkelstein and I. Sommerville. The viewpoints FAQ. *BCS/IEE Software Engineering Journal*, 11(1):2–4, 1996.

[12] T. R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. Technical Report KSL93-04, Stanford University, Stanford, August 1993.

[13] P. Hruby. *Model-Driven Design Using Business Patterns*. Springer-Verlag, 2006.

[14] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Nov. 1990.

[15] K. Lee and K. C. Kang. Feature dependency analysis for product line component design. In *Proc. of the 8th Int'l Conference on Software Reuse (ICSR'04)*, volume 3107 of *LNCS*, pages 69–85, Madrid, Spain, Jul 2004. Springer.

[16] N. Loughran, A. Sampaio, and A. Rashid. From requirements documents to feature models for aspect oriented product line implementation. In *Workshop on MDD in Product Lines (held with MODELS 2005)*, Montego Bay, Jamaica, Oct 2005.

[17] L. Mandel and M. V. Cengarle. On the expressive power of OCL. In *FM'99 - Formal Methods. Toulouse, France, September 1999. Proceedings, Volume I*, volume 1708 of *LNCS*, pages 854–874. Springer, 1999.

[18] Object Management Group. *Unified Modeling Language 2.0*, 2004. `http://www.omg.org/cgi-bin/apps/doc?ptc/04-10-02.zip`.

[19] OMG. *UML 2.0 OCL Specification*, 2003. `http://www.omg.org/docs/ptc/03-10-14.pdf`.

[20] I. Sommerville and P. Sawyer. PREview viewpoints for process and requirements analysis. Technical Report REAIMS/WP5.1/LU060, Lancaster University, Lancaster, May 1996.

[21] I. Sommerville and P. Sawyer. *Requirements Engineering: A Good Practice Guide*. Wiley, Apr. 1997.

[22] A. van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proceedings of the 5th International Symposium on Requirements Engineering*, pages 249–261, Toronto, Canada, Sep 2001. IEEE CS Press.

[23] W3C. OWL web ontology language. Document available at `http://www.w3.org/TR/owl-features/`.

[24] D. Wagelaar. Towards context-aware feature modelling using ontologies. In *MoDELS 2005 workshop on 'MDD for Software Product Lines: Fact or Fiction?'*, Montego Bay, Jamaica, Oct 2005. Position paper.

[25] H. Wang, Y. F. Li, J. Sun, H. Zhang, and J. Pan. A semantic web approach to feature modeling and verification. In *Workshop on Semantic Web Enabled Software Engineering (SWESE'05)*, Galway, Ireland, Nov 2005.

[26] A. Wasowski. Automatic generation of program families by model restrictions. In R. L. Nord, editor, *SPLC 2004*, volume 3154 of *Lecture Notes in Computer Science*, pages 73–89, Heidelberg, Germany, 2004. Springer-Verlag.

[27] J. Weiland and E. Richter. Konfigurationsmanagement variantenreicher Simulink-Modelle. In *GI-Jahrestagung (2)*, pages 176–180, 2005.

[28] W. Zhang, H. Mei, and H. Zhao. A feature-oriented approach to modeling requirements dependencies. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE 2005)*, pages 273–284, Paris, France, Aug 2005. IEEE Computer Society.

[29] T. Ziadi, L. Hélouët, and J.-M. Jézéquel. Towards a UML profile for software product lines. In *PFE*, pages 129–139, 2003.