On the Relationship between Feature Models and Ontologies

by

Chang Hwan Peter Kim

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2006

©Chang Hwan Peter Kim, 2006

I hereby declare that I am the sole author of this thesis.
I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Chang Hwan Peter Kim

I further authorize the University of Waterloo to reproduce this thesis by photocopying or other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Chang Hwan Peter Kim

Abstract

Feature modeling is an increasingly popular domain modeling technique, particularly used in software product line development for managing commonality and variability. Ontology modeling is also an increasingly popular domain modeling technique, applicable to software engineering in general. An emerging paradigm called *model-driven software product lines (MDSPL)* proposes systematic modeling as the often sought middle ground between configuration and custom coding of software product lines. In MDSPL, both feature modeling and ontology modeling influence solution models through domain modeling, but enigmatic differences between the two techniques, for example, in modeling philosophy and descriptive power, make their relationship intriguing. This relationship is explored in three stages. Firstly, a new light is shed on the nature of feature models, which clarifies exactly what feature models are and how they may be different from ontologies. Secondly, it is proposed that feature models are views on ontologies, where the mapping is defined precisely through syntactic correspondence and semantics. Finally, two complementary domain modeling approaches, view projection and view integration, are proposed that promote agility and address the limitations of feature modeling and ontology modeling.

Acknowledgements

First and foremost, I would like to thank my supervisor, Professor Krzysztof Czarnecki, who introduced me to the world of research. I am infinitely grateful to have met such a brilliant and wonderful teacher, who is truly a *visionary* in the software engineering field. One of the most important lessons I have learned from him is that the quality of research is what is really important. It is a lesson seldom translated into research accomplishments for me, but it is a lesson that I will never forget. I look forward to learning a great deal more from him.

I would also like to thank Professor Kostas Kontogiannis and Professor Paul Dasiewicz for serving on the thesis committee. I would like to thank Professor Michael Godfrey and Professor Ladan Tahvildari for giving me the opportunity to interweave research with course work. I would also like to thank Dr. Simon Helsen in his capacity as a great course instructor.

I thank my colleagues in the Generative Software Development Laboratory for providing valuable feedback to my research work.

I thank the ECE department for inviting me into the Combined BASc./MASc. program, which was smoothly completed. I highly recommend this program for those interested in getting a head start to do research work.

I thank my family members tremendously for supporting me through this phase in life. And I thank them, especially my father, even more for pushing me to pursue education to the fullest extent. I truly believe that life is about learning. I thank God for His continued support despite my failures to Him.

I strongly believe that there exists the Holy Grail of software engineering. While we may not know exactly what it is, our relentless pursuit to realize it is what defines us as noble researchers, engineers and scientists. For me, the pursuit continues into Ph.D. studies and continues on. Until it is realized, this noble individual will not rest.

Thank you and God bless you.

Contents

1 Introduction **2**

1.1 Feature Modeling 2

1.2 Ontology Modeling 3

1.3 Model-Driven Software Product Lines (MDSPL) 6

1.4 Research Overview 9

1.5 Research Contributions 11

2 On the Nature of Feature Models **13**

2.1 Essence of Feature Models: Feature Hierarchy and Variability 13

2.2 Notational Spectrum between Feature Models and Ontologies 16

3 Feature Models as Views on Ontologies **24**

3.1 REA Case Study 25

3.1.1 REA Metamodel Explained 26

3.1.2 REA Model of a Business-to-Commerce E-commerce Product Line . . . 27

3.2 Syntactic Correspondence 31

3.3 Semantics of Mapping 34

4 Domain Modeling Approaches **40**

4.1 View Projection 40

4.1.1 Process Overview 40

4.1.2 Process Demonstration 42

4.1.3 View Projection Classification 51

4.1.4 Benefits and Limitations 56
4.2 View Integration 56
4.2.1 Process Overview 57
4.2.2 Process Demonstration 57
4.2.3 View Integration Classification 60
4.2.4 Benefits and Limitations 61

5 Discussion and Future Directions 63
5.1 Notion of Feature Models 63
5.2 Towards a Domain Modeling Paradigm 64
5.3 Tool Support 64
5.3.1 Support for Feature Models as Views on Ontologies 64
5.3.2 Support for Domain Modeling Approaches 65

6 Related Work 68
6.1 Feature Dependency Analysis 68
6.2 Work on Semantics of Feature Models 68
6.3 Ontology Views 69
6.4 Viewpoint-Oriented RE 69
6.5 Early Aspects 70
6.6 Feature-Based Configuration of Models 71
6.7 Expressing FM in Ontology Languages 71

7 Conclusion 73

List of Tables

2.1 Symbols used in cardinality-based feature modeling [15] 16

3.1 Constraints [15] 36

List of Figures

1.1 A basic feature model and its configuration 4
1.2 A simplified ontology of an electronic shop 4
1.3 Scope of a product line [9] 7
1.4 Customization cliff [9] 7
1.5 Domain modeling and solution modeling 8
1.6 The scope of research with respect to a broader perspective of MDSPL [9] 10

2.1 Features of feature modeling [15] 15
2.2 Different renderings of a basic feature model 17
2.3 Increasing descriptive power of feature modeling [15] 17
2.4 Attributes in feature models 18
2.5 Cloning in feature models [15] 19
2.6 Reference attributes in feature models [15] 20
2.7 UML model of the e-shop family from Figure 2.6 22

3.1 Basic metamodel of the REA framework [25] 26
3.2 Custom REA icons [15] 26
3.3 Overall ontology 28
3.4 SaleOrder ontology 29
3.5 Views of the REA ontology in Figure 3.3 and Figure 3.4 [15] 32
3.6 Excerpts of the REA ontology, and its Feature Model Views [15] 33
3.7 Figure 3.5(a) extended with children-independent semantics 37
3.8 Children-independent semantics 37
3.9 Figure 3.5(a) extended with children-dependent semantics 38

3.10 Children-dependent semantics without variability 38
3.11 Children-dependent semantics with variability 38

4.1 View projection process 41
4.2 Ontology and feature model metamodels 43
4.3 QVT relations for achieving syntactic correspondence 46
4.4 Step-wise projection of Figure 3.5(a) from Figure 3.3 and Figure 3.4 49
4.5 Possible results of semantic analysis of Figure 4.4(b) 50
4.6 Examples of classes of view projection 52
4.7 A technique for crosscutting ontology view projection 52
4.8 Enterprise-provided and customer-provided events 53
4.9 A technique for crosscutting feature model view projection 54
4.10 View integration process 57
4.11 Integrated viewpoints 58
4.12 Integrating ontology 58
4.13 MultipleCatalogs constraint 59
4.14 PersonalizedView constraint 60

*Approximately, 20 percent of Section 1.1 and 60 percent of Section 1.3 were taken directly from [9]. My personal contribution to [9] was approximately 30 percent.

Chang Hwan Peter Kim

As one of the authors of [9], I fully acknowledge the thesis author's statement above as indicated by * and give full permission to use any part of [9]. I also fully acknowledge that the thesis represents original research conducted by the thesis author.

Krzysztof Czarnecki

As one of the authors of [9], I fully acknowledge the thesis author's statement above as indicated by * and give full permission to use any part of [9]. I also fully acknowledge that the thesis represents original research conducted by the thesis author.

Michał Antkiewicz

Chapter 1

Introduction

Feature modeling is an increasingly popular domain modeling technique, particularly used in software product line development for managing commonality and variability. Ontology modeling is also an increasingly popular domain modeling technique, applicable to software engineering in general. An emerging paradigm called *model-driven software product lines (MDSPL)* proposes systematic modeling as the often sought middle ground between pure configuration and complete custom coding of software product lines. In MDSPL, both feature modeling and ontology modeling influence solution models through domain modeling, but enigmatic differences between the two techniques, for example, in modeling philosophy and descriptive power, make their relationship intriguing. The objective of this thesis is to explore this relationship. A background of feature modeling, ontology modeling and MDSPL is first given.

1.1 Feature Modeling

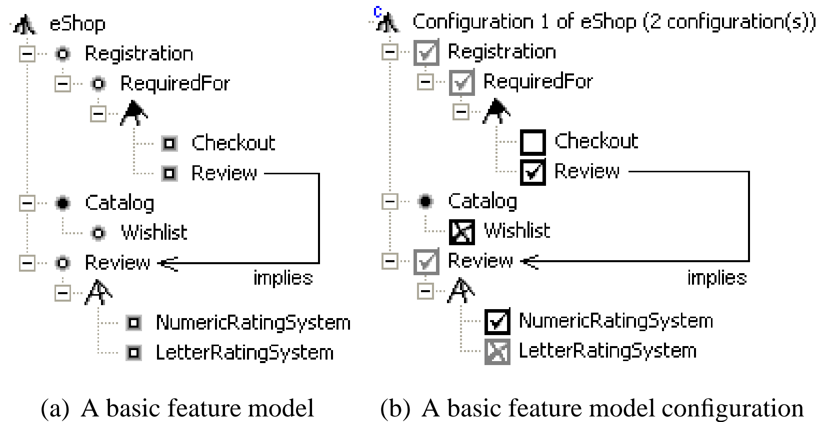
A software product line (SPL) is a set of systems with common characteristics. The goal of SPLs is to improve productivity, time-to-market, and quality of application development. These improvements are achieved by leveraging the commonalities of systems within an application domain while managing their differences. Software product line engineering (SPLE) factors out the commonalities into a *domain-specific platform* (DSP). Such a platform contains variation points, where choices from provided alternatives can be specified or new extensions can be plugged in. Examples of large, vendor-provided DSPs are IBM's WebSphere (WS) Commerce for e-

commerce applications [26], Fincentric's WealthView for banking solutions [18], and SAP's R/3 for Enterprise Resource Planning (ERP) systems [38].

A special discipline in SPLE is modeling the commonality and variability of artifacts in a SPL. Having an explicit model of what are common and what are variable helps in several activities including defining the scope of a SPL, identifying architectural variation points, and product configuration and derivation. An increasingly popular technique for commonality and variability modeling in SPLE is *feature modeling*. A feature model describes variations of a concept in increasing detail through a hierarchy of common and variable features. Figure 1.1(a) shows a basic feature model representing a family of electronic-shop storefronts (see Table 2.1 for an explanation of the notation used throughout the thesis). `Catalog` is mandatory, `WishList` is optional and `Registration`, which is also optional, may be required for `Checkout` and/or product `Review`. Naturally, if `Registration` is required for `Review`, then `Review` must be required, which is expressed as the implication. A feature model represents a set of configurations. The configuration process may be guided through constraint-based facilities [6], such as constraint checking, propagation, satisfiability, solving, and computing the number of remaining configurations [14]. The most basic constraint in a feature model is the subfeature relationship. The selection of the child feature implies the selection of the parent feature. Figure 1.1(b) shows a storefront configuration where manual choices (black colour), including the selection of `Registration/.../Review`, has propagated some automatic choices (grey colour), including the selection of the feature's parents and `Review` (due to the implication). Note that two different configurations, as indicated by the root feature label, can be made from this configuration, one by selecting `Checkout` and another one by eliminating it. A configuration with one or more undecided features like Figure 1.1(b) is more commonly referred to as a *partial configuration* or a *specialization*. A complete configuration with no undecided features is commonly referred to as just a *configuration*.

1.2 Ontology Modeling

An ontology, in information sciences and engineering, is commonly defined as "an explicit specification of conceptualization" [24]. Basically, an ontology is used to model a concept through a set of classes and associations between them. Consider Figure 1.2, which shows an extremely



(a) A basic feature model (b) A basic feature model configuration

Figure 1.1: A basic feature model and its configuration

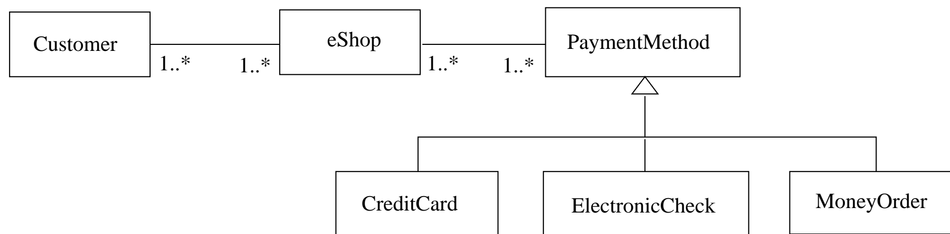


Figure 1.2: A simplified ontology of an electronic shop

simplified ontology of an electronic shop in an electronic commerce system.

A class, shown as rectangles, represents a set of individuals. `Customer`, `eShop`, and `PaymentMethod` represent the set of all customers, e-shops, and payment methods respectively. An association represents a set of *directed* and named links between individuals of one or more *domain* classes against individuals of one or more *range* classes. A binary association is between a domain class and a range class, while an N-ary association is between multiple domain classes and/or multiple range classes. The cardinality of an association may be further restricted through a minimum and maximum number of individuals of range class(es) that must be linked to an individual of a domain class. For example, `Customer.eShop` is the name of the association where the domain is `Customer` and `eShop` is the range. Each `Customer` must be linked to one or more `eShops` (a customer is naturally a customer of some e-shop(s)). Conversely, `eShop.customer` is the name of the association where the domain is `eShop` and

range is `Customer`. Each `eShop` must be linked to one or more `Customers` (ideally, every store should have at least one customer in its records).

An ontology element may denote an individual or a set of individuals, depending on the perspective. For example, for the ontology in Figure 1.2, a customer named "John Doe" is an individual of `Customer`. `Customer` is a set of individual customers, but it is just one of the classes in the ontology, i.e. an individual of `Class`. Furthermore, a metalevel above may see `Class` as an individual of some higher notion. A class is a subclass of another class if and only if the former represents a set of individuals that is a subset of the set of individuals represented by the latter. For example, `CreditCard`, `ElectronicCheck` and `MoneyOrder` are subclasses of `PaymentMethod`. Whether an ontology element should be a subclass or an individual of another class is dependent on the modeling perspective. Namely, if `PaymentMethod` were to represent the set of payment methods used at different points in time, then allowing an unbound upper limit of the multiplicity of `eShop.paymentMethod` would be appropriate, as shown in Figure 1.2. A payment method used would be an individual of one of the three subclasses of `PaymentMethod`. On the other hand, if `PaymentMethod` were to represent different payment methods available in an e-shop, then having an upper limit of the multiplicity of `eShop.paymentMethod` that is equal to the maximum number of payment methods available would be appropriate. Credit card, electronic check, and money order would represent individuals, not subclasses, of `PaymentMethod` in this case.

An important part of ontology modeling is *classification*. Classification groups individuals according to some criteria against their links to other individuals. Defining an *equivalent* class, also known as an *intentional* class, is a particular method of classification that uses conditional expressions on explicitly made classes to define a group of individuals [42]. For example, `eShopWithMoneyOrder` class may be intentionally defined as the set of `eShops` with at least one `eShop.paymentMethod` as an instance of `MoneyOrder`. An inference engine can be used to compute the individuals belonging to an intentional class. A small set of explicit or asserted classes and associations from which equivalent classes can be defined flexibly is considered a well-designed, robust ontology.

Ontology modeling vs. objected-oriented modeling Traditionally, the term *ontology modeling* is more closely associated with information management than software engineering. Ontology modeling is an essential part of *semantic web* [43], a global effort led by World Wide Web

Consortium (W3C) to define a standard framework for information reuse and sharing. OWL is widely considered as the standard ontology modeling language for semantic web. Object-oriented modeling, i.e. class diagram modeling, in software engineering is strikingly similar to ontology modeling in information management. Object-oriented modeling also uses classes, associations, and inheritance. Unified Modeling Language (UML) class diagrams [36], or Meta Object Facility (MOF) [1] for all intents and purposes, are primarily used to model object-oriented concepts. While there are technical differences between OWL and UML, such as the fact that OWL does not explicitly support containment or navigability, there does not seem to be any fundamental difference in terms of purpose, descriptive power, and even syntax. One notable difference is the explicit support for classification in information management ontology modeling. However, recent advancements in constraint (OCL [37]) and query support, which propose mechanisms powerful enough for classification, indicate that object-oriented modeling and information management ontology modeling may be converging. For the purpose of this thesis, both are considered as ontology modeling. Although UML/OCL is used throughout the thesis, other ontology modeling technologies may be used instead.

1.3 Model-Driven Software Product Lines (MDSPL)

Today's DSPs in SPLs are usually implemented using code-centric technologies such as object-oriented frameworks, components, and sometimes even #ifdef preprocessor directives. Applications built on top of such platforms represent some mixture of platform configuration settings and custom code. The ratio between these two may vary from platform to platform and application to application. For example, some deployments of SAP's R/3 may be almost 100% configuration, while solutions based on WS Commerce are usually more custom-code intensive.

In Figure 1.3, the oval represents the scope of a DSP and the black dots represent systems that can be arrived at through platform configuration. These systems do not cover the entire DSP, as suggested by the density of the dots, and as a result, certain systems, including the one represented by the cross, cannot be arrived at through configuration only. Assuming that the customer is not willing to except the next best solution, i.e. the closest dot to the cross, custom coding is required. Unfortunately, in most DSPs, configurable systems are relatively sparse,

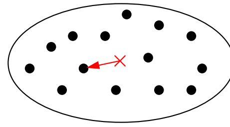


Figure 1.3: Scope of a product line [9]

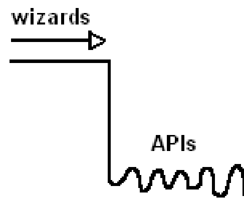


Figure 1.4: Customization cliff [9]

making the transition from configuration to custom coding rather abrupt. Steve Cook [1] refers to this situation as the "customization cliff", which is shown in Figure 1.4. The idea is that beyond the simple configuration facilities such as wizards, which are usually provided to lower the initial entry barrier, the platform user faces custom coding against the horrifying details of the platform's APIs. This idea is at odds with the intuitive principle that "easy things should be easy to do and progressively more complex tasks should only get progressively harder to do, not insurmountably harder."[2]

Model-driven software product lines (MDSPL) has the potential to eliminate the customization cliff in traditional, code-centric implementations of SPLs by offering modeling as a middle ground between configuration and custom coding. MDSPL is a rapidly growing field of intense research, with notable works being the research effort by Generative Software Development Group [9, 23], Software Factories [21], and Model-Integrated Computing [41]. MDSPL is a subset of Generative Software Development, which aims at automating the creation of a software system from a family of software systems from a specification written in textual or graphical domain-specific languages [8].

Ontology modeling and feature modeling in MDSPL. In MDSPL, models, for domain modeling, requirements, architectural design, and other purposes, are first-class citizens and are

[1]See http://blogs.msdn.com/stevecook/
[2]http://blogs.msdn.com/garethj/

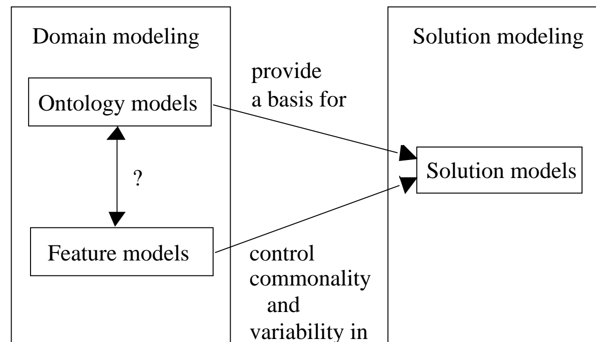


Figure 1.5: Domain modeling and solution modeling

used as main artifacts throughout the entire development process. Multiple models at multiple levels of abstraction influence one another through refinement, consistency relationships and so on.

Modeling in MDSPL can be divided into two categories: domain modeling and solution modeling. Solution modeling develops models that directly influence the end software product, including requirements, design, implementation models. On the other hand, domain modeling develops models more closely related to the problem space. In particular, as Figure 1.5 shows, feature models control commonality and variability of solution models from the viewpoint of domain-specific concepts. Ontology models provide a basis for solution models, influencing solution modeling. The influence may range anywhere from provision of common vocabulary and conventions to specification of the metamodel for solution models.

Both feature modeling and ontology modeling are domain modeling techniques and both feature models and ontologies are domain models. But, as the question mark in Figure 1.5 indicates, how are ontologies related to feature models? The similar role of feature models and ontologies in MDSPL, coupled with their enigmatic differences in modeling philosophy and descriptive power, make this question intriguing. Also, based on the relationship between ontologies and feature models, how are ontology modeling and feature modeling related? The author's main research objective has been to provide answers to these questions.

1.4 Research Overview

In Chapter 2, the essence of feature models is discussed, which sheds a light on exactly what feature models have evolved to be since its inception. Then the notational spectrum between feature models and ontologies is discussed to provide a framework for understanding the differences between feature models and ontologies.

In Chapter 3, the notion of feature models as views on ontologies is proposed using a case study of a typical business-to-commerce (B2C) e-commerce software product line, such as Amazon.com [3] or Wal-Mart [45]. Feature models from three different perspectives, i.e. business, usability, and administration, and a requirements-level ontology specified using an established framework for modeling economic concepts called REA (*R*esources, Economic *E*vents, and Economic *A*gents) [25] are used to define the precise relationship between feature models and ontologies. The case study is just a part of the broader perspective of MDSPL, applied against an e-commerce product line for example, as shown in Figure 1.6. The envisioned e-commerce product line is a set of web applications with a three-tier architecture based on the Java 2 Enterprise Edition (J2EE) framework. The top level contains three feature models of functional and non-functional requirements (business, usability, administration). The mappings among these feature models are defined as constraints over features. The feature models are mapped to the REA ontology as its views. The level below contains solution models that largely abstract from the target execution platform (such as J2EE). These models are configured by the three feature models and are based on the REA ontology above. The bidirectional mappings among the models offer consistency checking and model synchronization, meaning that the customization changes to one model are (potentially partially) propagated to its related models. The solution models are further refined into platform-specific models (which are also solution models) using model-to-model transformations [12]. Note that the web component model is based on an ontology describing web components and configured by feature models whose concepts are related to web components. Finally, application code is generated from the platform-specific models. Two rectangular areas, which enclose the feature models and the ontologies, show the scope of this thesis. While this thesis uses requirements-level ontologies and feature models as examples (the higher rectangle), the ideas presented are equally applicable to lower level domain models (the lower rectangle).

From the notion of feature models as views on ontologies, two complementary domain mod-

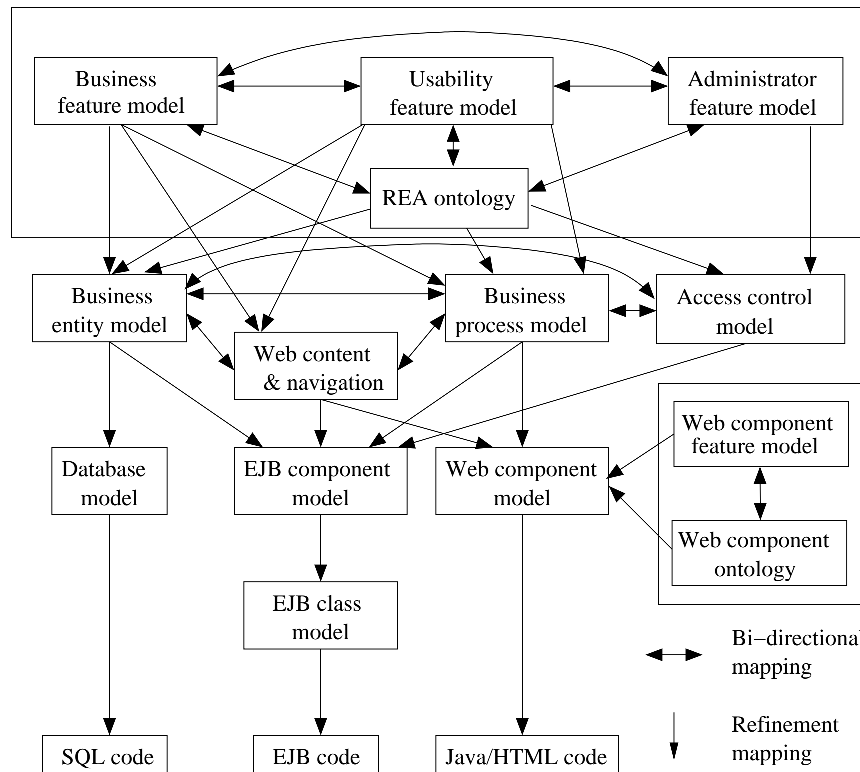


Figure 1.6: The scope of research with respect to a broader perspective of MDSPL [9]

eling approaches, view projection and view integration, are proposed in Chapter 4. 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 done to model concepts from different viewpoints for the purpose of scoping the concept modeled by the ontology. In view integration, feature models, or outlines of requirements, are first created more or less independently, with some ontology in the mind of each modeler. Ontology can be used to align the views, with a particular focus on developing the overlapping parts. For each approach, a general process is proposed and demonstrated in concrete terms and a classification of each approach based on some important dimensions is presented.

In Chapter 5, the nature of feature models, with respect to ontologies, is discussed. Possible ideas for converging towards a novel domain modeling paradigm are presented. Then tool support suggestions for supporting feature models as views on ontologies and the two proposed

domain modeling approaches, are made.

In Chapter 6, bodies of related work are discussed. Chapter 7 concludes the thesis.

1.5 Research Contributions

Contributions, all of which are novel to the author's best knowledge, can be classified into three categories.

On the nature of feature models. A new light is shed on the nature of feature models. The essence of feature modeling is spelled out and a family of feature models is described using a feature model that counteracts the popular belief of what feature models are. Also, the notational spectrum of feature models that extends to ontologies is discussed and a possible boundary between feature models and ontologies is proposed as a basis for future research.

Feature models as views on ontologies. Based on the analysis of commonalities and differences in notational, modeling philosophy and purpose in MDSPL, it is proposed that feature models are views on ontologies. A concrete mapping mechanism, i.e. feature-based restriction, that involves embedding features in ontology constraints is proposed. The mechanism is a novel way of mapping feature models to any kind of model supporting constraints.

Domain modeling approaches. Two novel domain modeling approaches, view projection of feature models from ontologies and view integration of feature models into ontologies, are proposed. A general as well as a specific process is discussed and shown through concrete examples and technologies, including QVT and OCL for UML ontologies. A classification of each approach is presented as a possible basis for future research.

*Approximately, 90 percent of Chapter 2 was taken directly from [15]. My personal contribution to the corresponding section in [15] was approximately 40 percent.

Chang Hwan Peter Kim

As one of the authors of [15], I fully acknowledge the thesis author's statement above as indicated by * and give full permission to use any part of [15]. I also fully acknowledge that the thesis represents original research conducted by the thesis author.

Krzysztof Czarnecki

As one of the authors of [15], I fully acknowledge the thesis author's statement above as indicated by * and give full permission to use any part of [15]. I also fully acknowledge that the thesis represents original research conducted by the thesis author.

Karl Trygve Kalleberg

Chapter 2

On the Nature of Feature Models

Since its inception [27], feature models have evolved considerably in terms of modeling philosophy and notation. Whereas feature models originally described software attributes that "directly affect end-users", today, feature models are more generally used to describe any commonality and variability of a concept. To this extent, feature modeling is a domain modeling technique, as ontology modeling is a domain modeling technique, and a feature model is certainly something more than what was originally conceived. Notationally, a number of extensions has been proposed that seems to push the descriptive power of feature models to that of ontologies. This chapter aims to answer the questions, "What *exactly* are feature models and how are they different from ontologies?"

2.1 Essence of Feature Models: Feature Hierarchy and Variability

A *feature model* is a hierarchy of features with variability. Figure 2.1(a) defines the concept of feature modeling notations as a feature model. *Feature hierarchy* is shown as a mandatory feature (see Table 2.1 for an explanation of the notation used through the thesis). 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 2.1(a). The most basic variability mechanism is the notion of *optional features*, which is mandatory according to Figure 2.1(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 weaker 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 2.1(a) is an example of a reference to the model in Figure 2.1(c).

Contrary to popular belief, feature models may be rendered in different forms, some of which are listed in Figure 2.1(c). *FODA-style trees* refers to notational styles resembling the original FODA diagrams [27]. *Explorer-view* style is the rendering used in this thesis. 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 2.2, which shows a basic feature model rendered in explorer-view (Figure 2.2(a)), structured document view (Figure 2.2(b)) and mindmap view (Figure 2.2(c)) [1].

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, like those in Figure 2.2(b) and Figure 2.2(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.

[1] Group constraints are not shown in the structured document and mindmap renderings for simplicity.

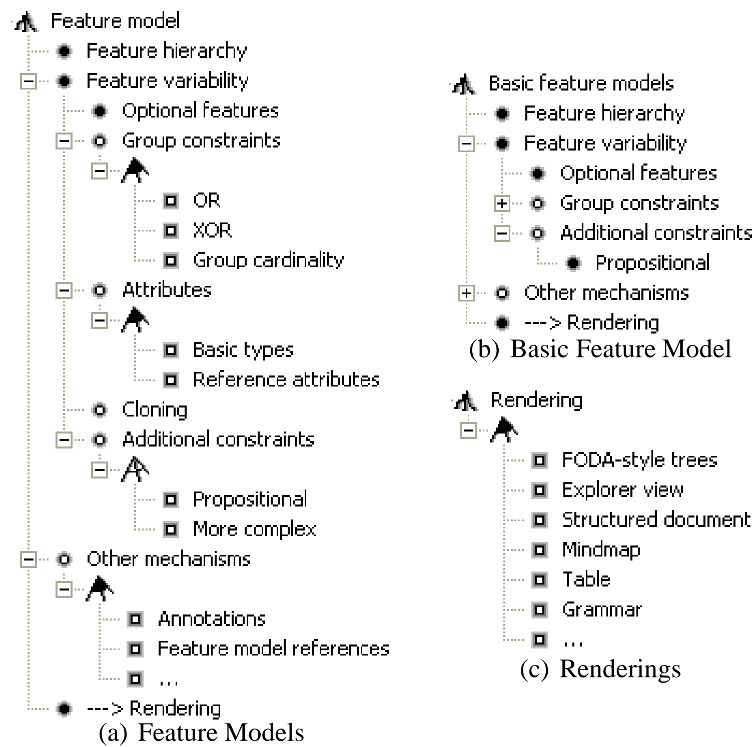


Figure 2.1: Features of feature modeling [15]

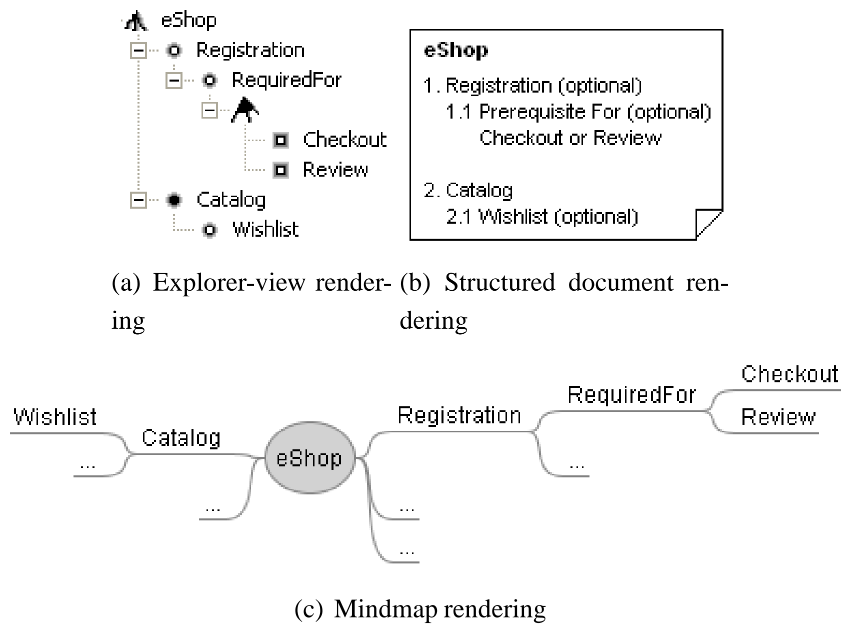
Symbol	Explanation
[symbol]	Root feature
●	Solitary feature with cardinality $[1..1]$, i.e., *mandatory* feature
○	Solitary feature with cardinality $[0..1]$, i.e., *optional* feature
[symbol] $[0..m]$	Solitary feature with cardinality $[0..m]$, $m > 1$, i.e., *optional clonable* feature
[symbol] $[n..m]$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

Table 2.1: Symbols used in cardinality-based feature modeling [15]

2.2 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" [24]. An ontology represents the semantics of classes and their associations 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 [6]. 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.



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

(c) Mindmap rendering

Figure 2.2: Different renderings of a basic feature model

These extensions bring feature models closer to that of a complete ontology formalism, and make out the spectrum in Figure 2.3. The main considered extensions to basic feature modeling include feature attributes, cloning, and reference attributes [11]. 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) and/or the expressiveness (i.e., allow expressing new mathematical concepts) [17].

Basic feature modeling. Basic feature models can be thought of as a hierarchy plus a

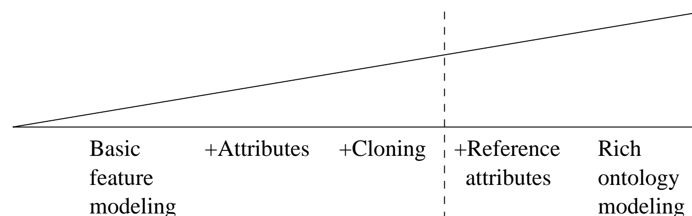


Figure 2.3: Increasing descriptive power of feature modeling [15]

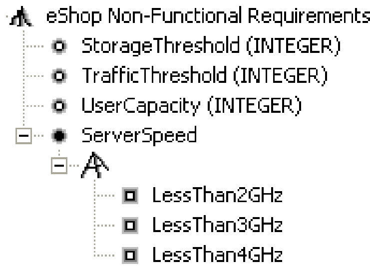


Figure 2.4: Attributes in feature models

propositional formula. Figure 2.1(b) defines this class of notations as a specialization of the feature model in Figure 2.1(a). An example of a basic feature model representing the requirements for a family of electronic shops is shown in Figure 1.1(a). A basic feature model can contain additional constraints as propositional formulas, such as `Registration/.../Review`⇒`Review`. 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 feature models. While a propositional formula allows expression of more complex feature dependencies, the essence of the basic feature model is to abstract from such expressiveness as much as possible. The basic constraint solving machinery is also applicable for consistency checking, choice propagation and auto-completion of configurations of feature models, as described in [14]. Figure 1.1(b) shows a configuration of an electronic shop storefront where some manual choices (shown in black) have propagated some automatic choices (shown in gray).

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 [11]. For example, Weiland et al. [48] 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 op-

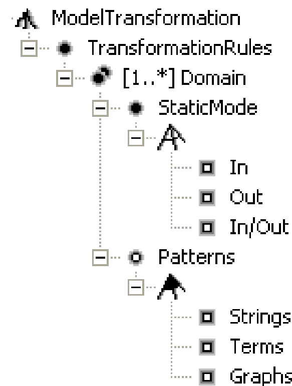


Figure 2.5: Cloning in feature models [15]

erating 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 need not be a problem, constraint solving can quickly turn into a complicated optimization problem in many cases. For example, in Figure 2.4, 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. This can often be solved with the proper tooling and machinery, but if constraints become too complex, it should be regarded as a warning sign saying that perhaps what you are doing is outside the scope of feature modeling.

Cloning. Cloning is an extension that allows the upper bound of a feature cardinality to be greater than one. Although cloning is rather an exotic mechanism in feature modeling, there are clearly cases where cloning is useful. As an example, Figure 2.5 shows a fragment of a feature model that characterizes a family of model transformation approaches [12]. 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-

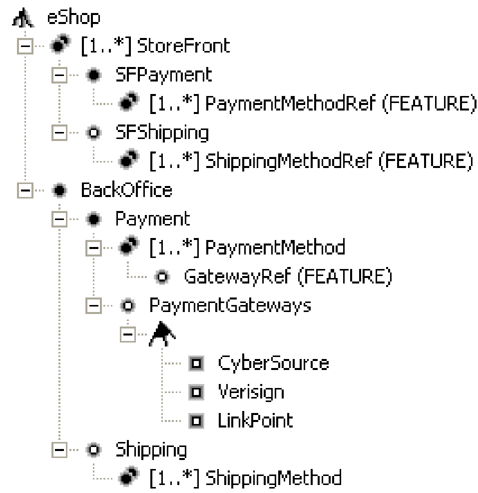


Figure 2.6: Reference attributes in feature models [15]

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, such as constraints over sets of clones [14]. Simple constraints over clones, such as constraining the size of a set of clones to be within a range, are rather unproblematic. However, more complex constraints over clones, such as requiring a clone of feature A for every clone of feature B, demand richer constraint facilities (in particular, constraint propagation and constraint solving) and essentially model associations. Allowing such relationships to be modeled is one of the strengths of 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 only of interest in the presence of clonable features. Figure 2.6 shows an excerpt from a feature model describing a family of electronic shops from an earlier thesis [14]. The feature model makes a heavy use of feature cloning. The `BackOffice` part of the feature model allows creating multiple `PaymentMethods` and `ShippingMethods`, which need to be referred to from the `StoreFront` part. This is modeled using the feature references of `PaymentMethodRef` and `ShippingMethodRef`. Ad-

ditional constraints are needed to restrict the scope of features the attributes can refer to. For example, the constraint that the attribute (`att`) of `PaymentMethodRef` should only point to clones of `PaymentMethod` can be stated using OCL as follows [14]:

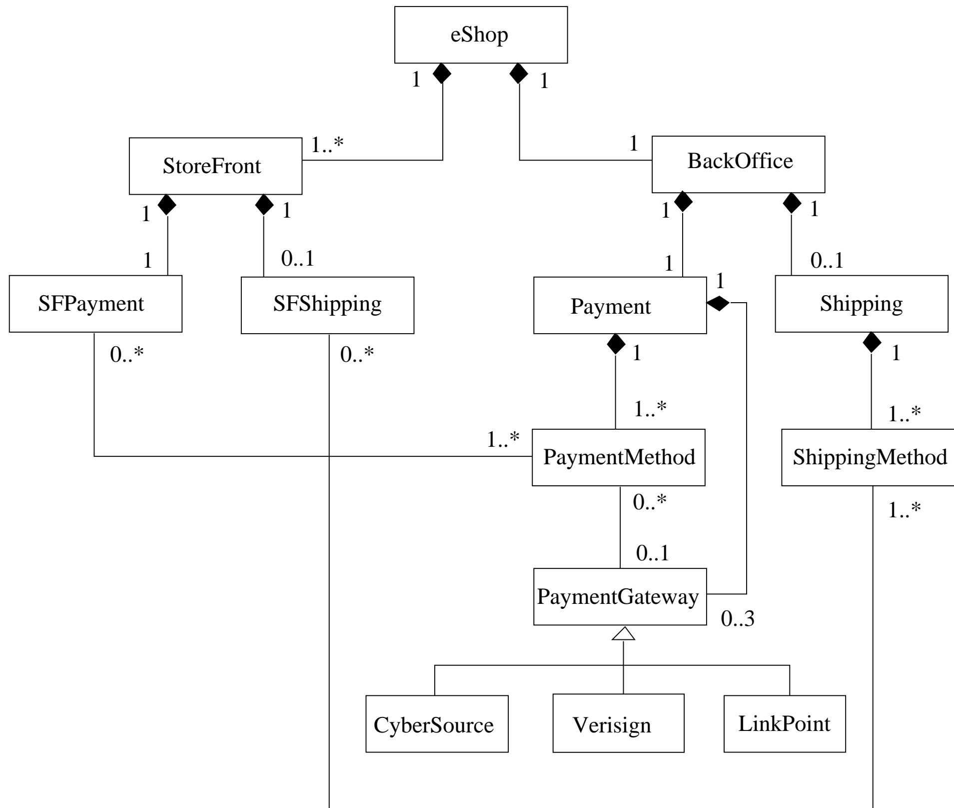
```
context PaymentMethodRef inv:
   EShop.BackOffice.Payment.PaymentMethod->includes(att)
```

In essence, reference attributes can be viewed as "a poor man's associations." Consequently, reference attributes seem to be outside of the scope of feature modeling, as indicated by the dashed line in Figure 2.3.

Rich ontology modeling. Ontologies may be described in formal languages based on first-order logic (FOL) such as KIF [16] or OWL [42]. Alternatively, ontologies may also be represented using UML class diagrams and OCL[2]. This is commonly done in software engineering. In the UML approach, individuals are represented using classes and their relationships are represented using UML associations. Domain-specific semantics can be provided by a specialized UML profile, as will be illustrated in Section 3.1. In this thesis, only ontologies represented using UML class diagrams are shown, but the presented ideas can be generalized to other ontology formalisms, too. Figure 2.7 shows a UML ontology of the e-shop family from Figure 2.6. The use of association and specializations results in a more intuitive model.

Other extensions. The discussed extensions influence the configurations, or the semantics, of a feature model. Other extensions that do not change the semantics of a feature model, namely, annotations, are non-essential. For example, a feature dependency, like a UML dependency as opposed to an association, typically has two components: formal semantics, which is an additional propositional constraint on the feature model like *requires* or *excludes*, and informal semantics, expressed through a description of a relation like *implements* or *uses*. The first component can be expressed in a basic feature model. The second component, which cannot be expressed using symbols already in the feature model, is an annotation that is an additional symbol. While non-essential, an annotation is a symbol in a configuration, like a feature name, which can be useful for interpreting the configuration.

[2]OCL is more expressive than FOL, e.g., transitive closures over object relations can be expressed in OCL but not in FOL [33].




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

Figure 2.7: UML model of the e-shop family from Figure 2.6

*Approximately, 50 percent of Chapter 3 was taken directly from [15]. My personal contribution to the corresponding section in [15] was approximately 100 percent.

Chang Hwan Peter Kim

As one of the authors of [15], I fully acknowledge the thesis author's statement above as indicated by * and give full permission to use any part of [15]. I also fully acknowledge that the thesis represents original research conducted by the thesis author.

Krzysztof Czarnecki

As one of the authors of [15], I fully acknowledge the thesis author's statement above as indicated by * and give full permission to use any part of [15]. I also fully acknowledge that the thesis represents original research conducted by the thesis author.

Karl Trygve Kalleberg

Chapter 3

Feature Models as Views on Ontologies

With an understanding of the nature of feature models, we can further explore the relationship between ontologies and feature models in three dimensions: notation, modeling philosophy, and role in MDSPL.

Notationally, as discussed in the previous chapter, feature models, on the left side of the spectrum in Figure 2.3, are less powerful than ontologies, on the right side of the spectrum. Since feature models are more appropriate for expressing a subset of what ontologies can express (for example, reference attributes seems to be outside of feature models but definitely within ontologies), feature models can be considered as a notational subset of ontologies.

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. 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 bottom-up fashion. For example, when defining a feature model of the "business" concept of a B2C e-commerce product line, the scope of the "business" concept is first set to, say, high-level business processes involving customers. Each business process is further broken down and ultimately, a feature model describing commonality and variability in each business process in increasing detail is devised. Ontology modeling may proceed in a similar top-down way, by first conceiving the business processes and the associations between them. However, as the ontology evolves, while the details of business processes may have been added, it is very likely that other high level concepts, like order fulfillment and inventory tracking, that have very little to do with customers, are modeled as well. Unlike the feature model, the scope

of “business” concept is defined by what’s in the ontology, not vice versa. Inevitably, as the ontology evolves, the scope of the concept being modeled will expand. Therefore, an ontology is likely to model a more general concept, i.e. with a larger scope, than that of a feature model.

As mentioned in Chapter 1, feature models control commonality and variability in solution models while ontologies provide a basis for solution models. These roles in MDSPL are not very different, as the two modeling techniques are both domain modeling techniques.

The fact that 1) feature models form a notational subset of ontologies, 2) feature models describe concepts more specialized than those described by ontologies, and 3) feature models and ontologies are both domain models suggests that feature models are *views* on ontologies, namely, projections of the ontologies from different viewpoints. This chapter presents a formalization of this relationship through syntactic correspondence and semantics of mapping between feature models and ontologies. Syntactically, traceability links between feature and ontology elements are achieved to increase understanding of the mapping or to represent simple dependencies like existential constraints. Semantically, a feature model restricts an ontology from a viewpoint through additional constraints. An e-commerce product line case study is used to explain the mapping.

3.1 REA Case Study

The Resource Event Agent (REA) framework [25] is an established system of guidelines, rules, patterns, and schemas for constructing an ontology of business elements. Figure 3.1 shows the basic metamodel of the REA framework. The system allows the economic world to be modeled in terms of duality of economically beneficial and detrimental `EconomicEvents` that are functions of providing/receiving `EconomicAgents` and incoming/outgoing (`stockflow`) `Resources`. Duality is a fundamental idea behind REA: in economy, an event that decreases value of some resources is inevitably linked via the `duality` association to another event that increases value of the same or other resources from a perspective. Ontologies developed using frameworks like REA are likely to be more complete and stable than the ones developed using less systematic methods.

In this case study, in the context of an electronic commerce software product line, some REA business patterns that are modeled according to a metamodel considerably more complex than

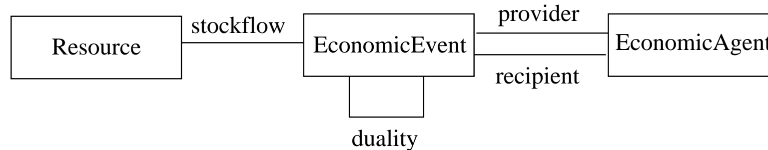


Figure 3.1: Basic metamodel of the REA framework [25]

Figure 3.2: Custom REA icons [15]

the one in Figure 3.1 are presented.

3.1.1 REA Metamodel Explained

Instances of the REA metamodel are represented using UML stereotypes and intuitive icons for classes shown in Figure 3.2. From left to right in Figure 3.2, `Resources`, which are scarce, are represented using diamonds. A `ResourceType`, which represents a set of `Resources` of the same type, is represented using a shaded diamond. For example, a particular `Resource-Type` is `ProductType`, an instance of which is a particular product description in a catalog that typifies the distinguishable products leaving the store inventory. `EconomicEvents`, which may be incremental or decremental from the perspective of an `EconomicAgent`, are represented using filled and unfilled stars respectively. `EconomicAgents`, such as customer and enterprise, are represented using stickmen. `EconomicEvents` are involved in *conversion* and *exchange* processes. In a conversion process, `EconomicEvents` *uses* and/or *consumes* some resources to *produce* some other resources. A consumption of a resource makes it disappear, while a usage does not. In an exchange process, resources flow into and out of a set of activities related through *duality* associations.

Models conforming to the pure REA metamodel only contain instances of `Resource`, `EconomicEvent` and `EconomicAgent`, hence the acronym. However, more advanced economic notions require additional constructs. A `Contract`, a legally binding agreement between

at least two parties that details Commitments and Terms, is represented using a document. A Term, a contractual condition whose violation can trigger the requirement of some Commitments, is represented using a magnifying glass. Finally, a Commitment, an agreement to fulfill an incremental or decremental EconomicEvent in the future, is represented using a filled and unfilled clock respectively.

3.1.2 REA Model of a Business-to-Commerce E-commerce Product Line

Figure 3.3 and Figure 3.4 show an ontology of system capabilities and entities of a B2C e-commerce product line modeled according to the REA metamodel and icons described in the previous subsection. The ontology is broken down into two parts to facilitate explanation: Figure 3.3 shows the overall ontology and Figure 3.4 shows the elements largely relevant to the SaleOrder contract. Two EconomicAgents are assumed to exist: customer and enterprise. All EconomicEvents are modeled from the perspective of the Enterprise. To keep the example visually clear, some trivial relationships are left out, but explained.

In Figure 3.3, from left to right, a Registration event produces an Account, which may be or may not be used by Browsing, Checkout, and Review (product review) events, which are provided by the customer. Registration, Browsing, Checkout, and Review are increment events because they increase membership base, traffic, sale orders, consumer community base respectively, which are the enterprise's net resources. On the other hand, BehaviourTracking, a system capability that monitors customer activities, is a decrement event that may be a dual of Browsing and Checkout because it uses system resources to increase usability of these two events. Similarly, SystemSettingsDetection, a system capability for analyzing properties of the customer computer, may be applied to Registration, Browsing or Checkout also to increase the usability of these activities.

The word *may* is used here to emphasize variability. Different products in the e-commerce product line may have different configurations, more specifically subsets, of the ontology. Also, note that the stereotypes on associations, like use, produce, and duality, indicate the type, not the name, of association, which is an association between EconomicEvent and Resource metaclasses as described in Section 3.1.2. Model associations are referred to using standard UML naming convention, for example, Checkout.account. In addition, while a system capability may be modeled as discrete events or a continuous event, to keep the discus-

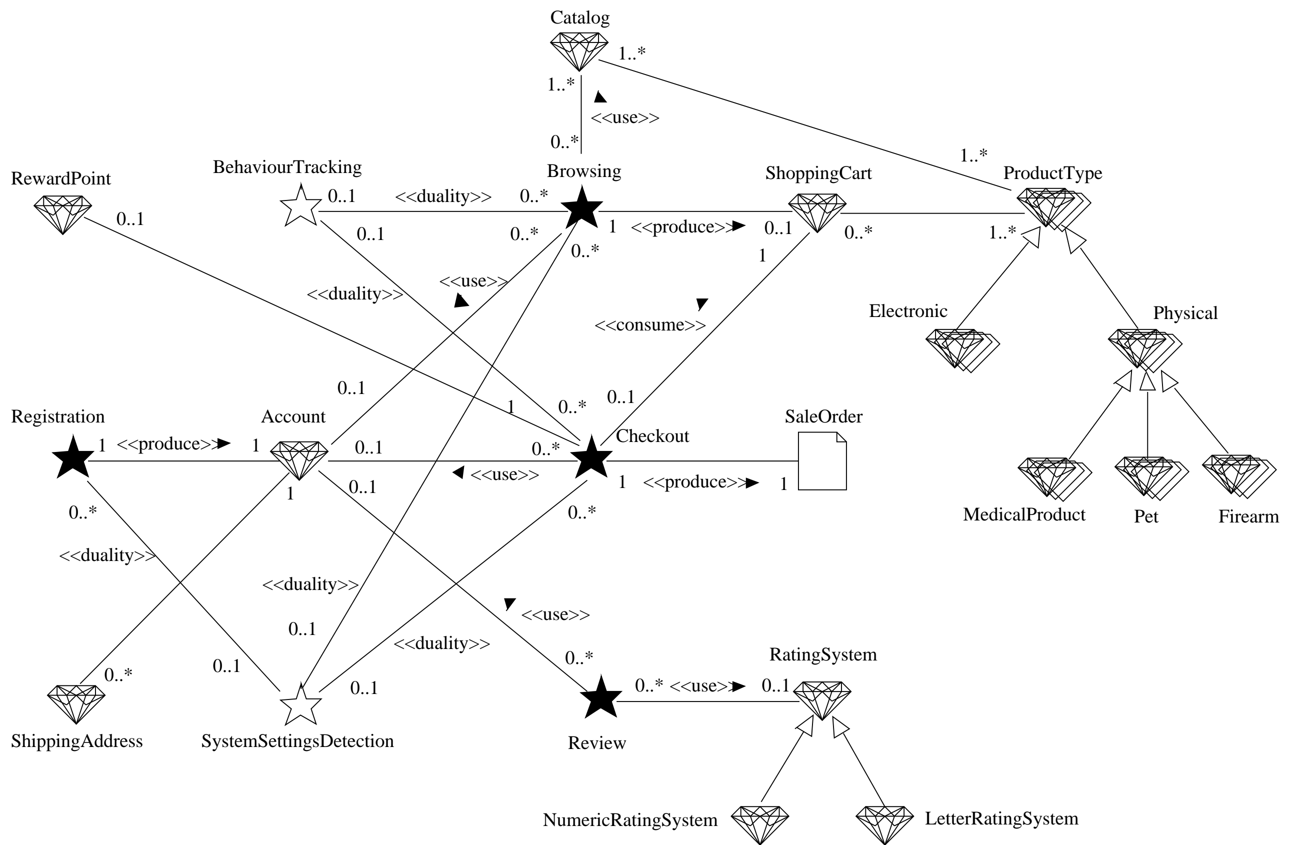


Figure 3.3: Overall ontology

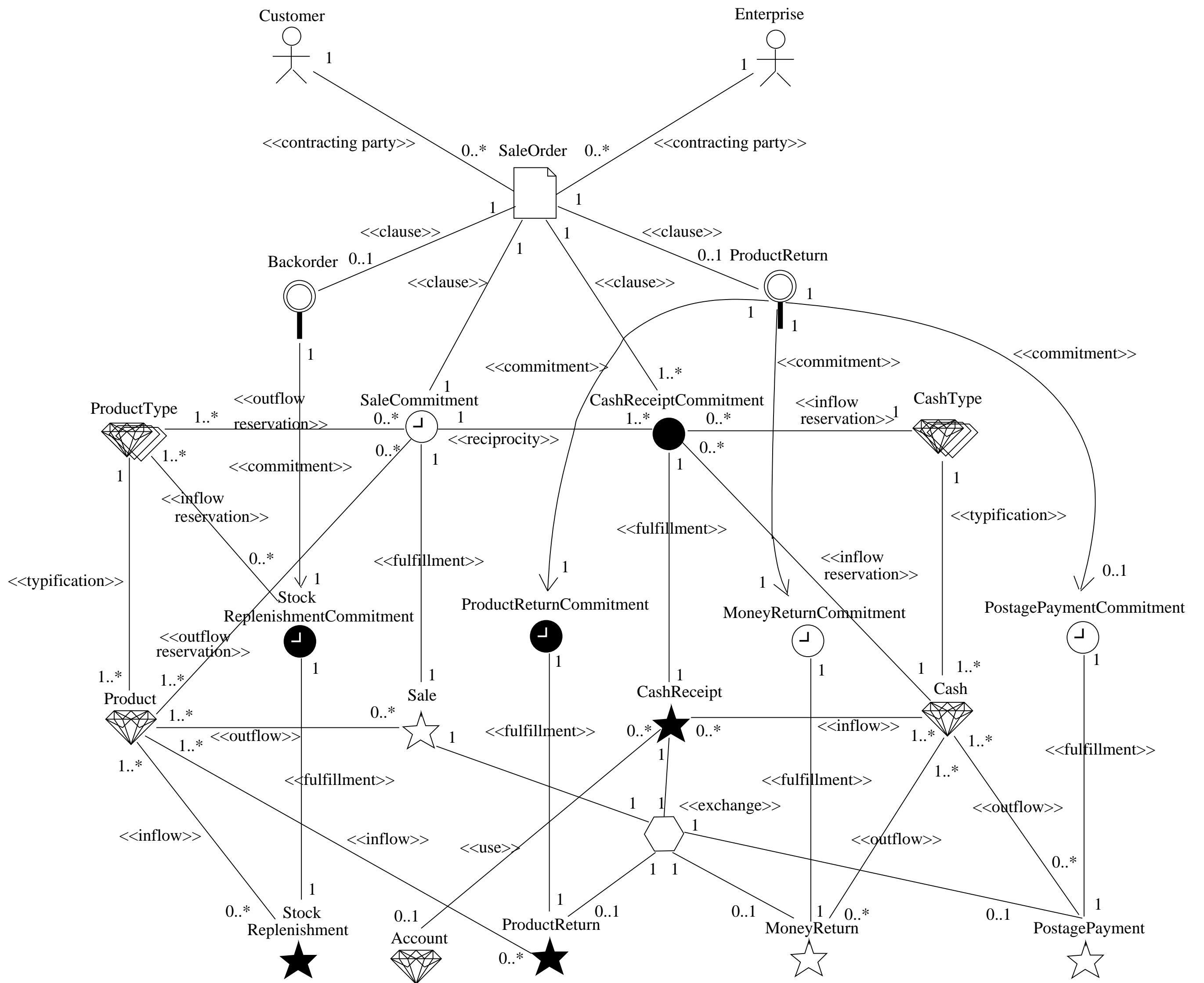


Figure 3.4: SaleOrder ontology

sion 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, such as browsing.behaviourTracking and checkout.SystemSettingsDetection in Figure 3.3, have the cardinality [0..1].

Continuing on in Figure 3.3, an Account may have multiple ShippingAddress and on a Checkout, a RewardPointAmount may be applied to the purchase being made. Browsing uses at least one Catalog, which describes at least one ProductType. [1]. ProductType is divided into two main categories: Electronic refers to digital ProductTypes while Physical refers to tangible ProductTypes. A consumer Review of ProductType may use a RatingSystem, which may allow numeric or letter rating. A Browsing event ideally produces a ShoppingCart, which is consumed by a Checkout event to produce a SaleOrder.

Figure 3.4, an extension of the advanced "Guarantee" pattern presented in [25], shows just one of the possible ways of using REA to model sale orders as contracts. A SaleOrder, also referred to as a *purchase* throughout this thesis, is a Contract between an Enterprise and a Customer in which the enterprise is committed to making a sale (SaleCommitment) and the customer is committed to making a payment or a CashReceipt (CashReceiptCommitment), typically in the near future. Note that a SaleCommitment is associated with ProductType and Product through <<outflow reservation>>. This means that in order for a Sale to be fulfilled in the future, a SaleCommitment must reserve ProductTypes, handbag descriptions in a catalog for example, and the typified Products, handbags with particular serial numbers in a warehouse for example. When the actual Sale is fulfilled, the handbags leave the warehouse due to the Sale-Product outflow association. Likewise, the actual Cash (money belonging to an entity), typified by a CashType (money amount) like *a thousand dollars*, arrives at the enterprise's possession when the actual CashReceipt is fulfilled.

A SaleOrder may have Backorder and ProductReturn in its clauses. Backorder is an implication stating that if some ProductTypes and their typified Products cannot

[1] Catalog-ProductType, which does not have any stereotype, is an example of an association that is not an instance of an REA meta-association. The example, while heavily influenced by REA, is essentially a class diagram with associations, inheritance, profiles, and etc.

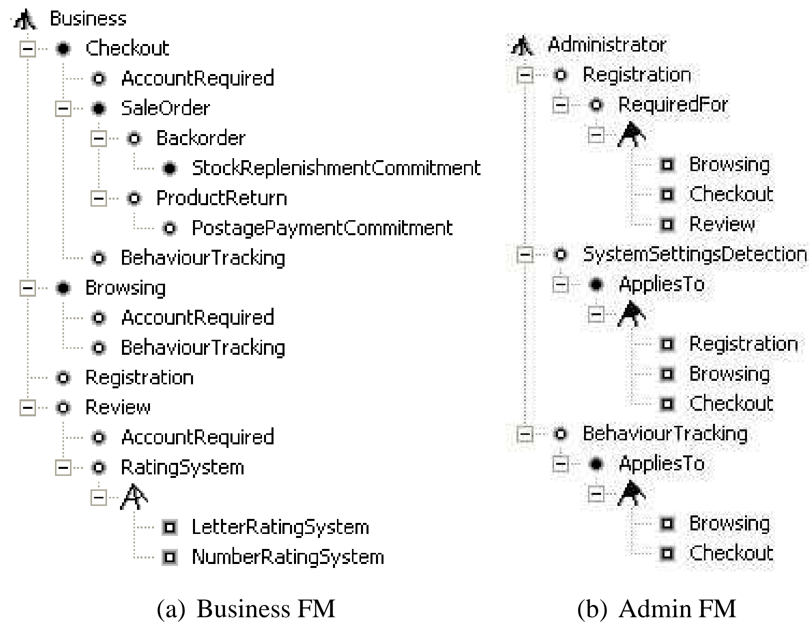
be reserved (i.e. not available at the time of purchase), then the enterprise is committed to `StockReplenishment`, which will stock the unavailable `Products` within a certain period of time. `ProductReturn` is an implication stating that if a customer wants to return purchased `Products`, then the customer must fulfill the commitment to physically return the `Products` back to the enterprise, while the enterprise must fulfill the commitment to physically return the customer's money. A `ProductReturn` may require the enterprise to make a `PostagePayment` to help the customer return the `Products`. Note that the events occurring due to `SaleOrder`, namely `Sale`, `CashReceipt`, `ProductReturn`, `MoneyReturn`, and `PostagePayment`, are related through an `exchange` process.

Two feature models are presented as views on Figure 3.3 and Figure 3.4. The business feature model, in Figure 3.5(a), views the ontology in terms of customer-provided events: `Checkout`, `Browsing`, `Registration`, and `Review`. The administrator feature model, in Figure 3.5(b), views the ontology in terms of system administration concerns, namely, `Registration`, `SystemSettingsDetection`, and `BehaviourTracking`. Syntactic correspondence and semantics of mapping between an ontology and feature model views are formalized using these examples.

3.2 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 M-N association between feature elements and ontology elements. But a typical mapping is where an element-to-element mapping is used to express an existential dependency from a feature element to an ontology element. Figure 3.6 shows examples of such mappings between the REA ontology and its two views[2]. Note that to define a uniform and precise syntactic and semantic mapping, dependencies must be represented as associations with stereotypes, as done in Figure 3.3.

[2] Ontology elements not involved in a mapping are colored lightly.

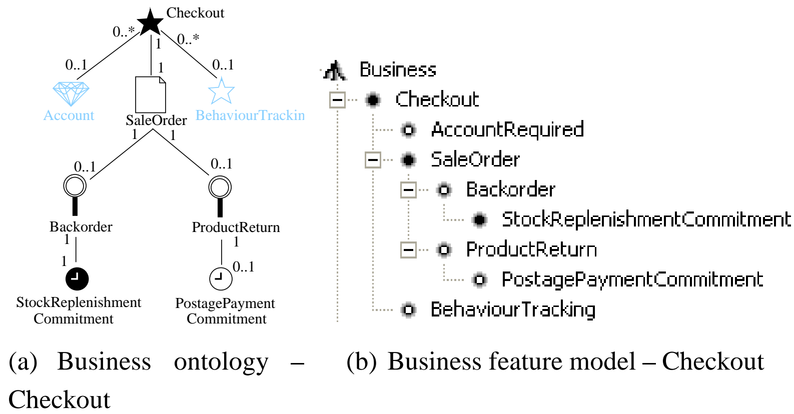


(a) Business FM (b) Admin FM

Figure 3.5: Views of the REA ontology in Figure 3.3 and Figure 3.4 [15]

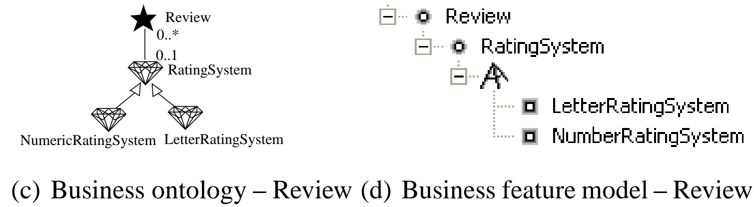
Isomorphic mapping. In Figure 3.6(a) and Figure 3.6(b), the `SaleOrder` feature is mapped to the actual `SaleOrder` class, while the subfeature relationship between `Checkout` and `SaleOrder` is mapped to the association `Checkout-SaleOrder`. Likewise, `Backorder` and `ProductReturn` features, which 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, the feature model region and the ontology region are isomorphic [3], which means that the ontology region is very much like a feature hierarchy, modeled with a single parent in mind. This makes sense in the last case, since `Terms` can only exist in the context of a `Contract`, like a `SaleOrder`. Similarly, potential `Commitments`, including `PostagePayment`, make sense only in the context of their `Term`, like `ProductReturn`. Generally, isomorphic mapping is suitable when an ontology describes partonomy, characterization, or single inheritance. Note the structural resem-

[3] "A graph isomorphism is a bijection, i.e., a one-to-one mapping, between vertices of two graphs G and H, f:V(G)⇒V(H), where any two vertices u and v from G are adjacent if and only if f(u) and f(v) from H are adjacent" [49].



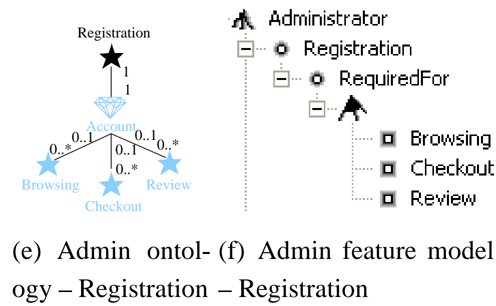
(a) Business ontology – Checkout

(b) Business feature model – Checkout



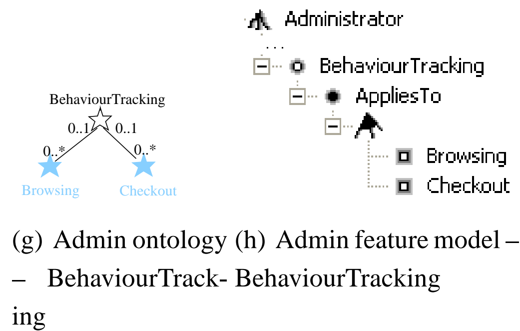
(c) Business ontology – Review

(d) Business feature model – Review



(e) Admin ontology – Registration

(f) Admin feature model – Registration



(g) Admin ontology – BehaviourTracking

(h) Admin feature model – BehaviourTracking

Figure 3.6: Excerpts of the REA ontology, and its Feature Model Views [15]

blance between Figure 3.6(d) with its ontology counterpart Figure 3.6(c). Here, a feature group maps to the specialization relationship. In isomorphic mapping, subfeature cardinalities usually correspond to association cardinalities. This is evident between `SaleOrder` and `Review` features and their ontology counterparts.

Feature-to-association mapping. In Figure 3.6(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 constraint on the association. If a feature is involved in a feature-to-association mapping, it is difficult to specify the syntactic correspondence of its child features, as the child features refine the semantics of the feature-to-association mapping. This problem is explored in detail in Section 3.3.

Different views. Different views may syntactically correspond to the same region of ontology through different traversals. For example, Figure 3.6(a) and Figure 3.6(b) show `Account` and `BehaviourTracking` being viewed from `Checkout`. Figures 3.6(e), 3.6(f), 3.6(g) and 3.6(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 3.6(b), `BehaviourTracking` capability in Figure 3.6(h) may be turned on or off for all events.

3.3 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, *feature-based restriction*: features of a feature model are embedded in ontology constraints, which are specified against the ontology and are applicable to individuals of the ontology. For a given feature model configuration, simply, the feature variables, which are typically Boolean, are replaced by their values in the OCL constraints. We first discuss some simple semantics and then discuss more complex semantics.

Simple Semantics

Table 3.1 shows some of the applicable constraints on Figure 3.3 and Figure 3.4. A feature ID, shown within `<<  >>`, is replaced with the Boolean value of the feature at configuration time. For example, constraint 1 states that if a business viewpoint configuration 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` feature in Figure 3.6(f). 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. So the constraint `AccountRequired<->Checkout` must be placed between Figure 3.6(b) and Figure 3.6(f) and evaluated to true before constraint 1 can be evaluated. Alternatively, if the constraint between the two features is not desired, perhaps because they have other semantics for which they cannot be equivalent, they can be ANDed to form the left-side of the implication in constraint 1. Constraint 2 shows a situation where the existence of an association, i.e. `SaleOrder.backOrder`, is dependent on the existence of a feature, `Backorder`. Constraint 3 shows a case where a class, `Registration`, cannot have any individuals. Due to the one-to-one association from `Registration` to `Account`, this constraint effectively eliminates `Account` as well when `Registration` feature is eliminated, which has further implications. Constraint 4 shows a type restriction of the range of an association.

Binding time. Feature model configuration is done with respect to a binding time. If a binding time is associated with a feature configuration, the same binding time should be associated with the ontology constraints written over that feature. For example, in Figure 3.6(c) and Figure 3.6(d), if `RatingSystem` has an installation binding time, the ontology constraint may state that the elimination of the feature implies that `RatingSystem` is not supported, and state nothing about its selection:

```
context RatingSystem inv:
    (not <<RatingSystem>>) implies RatingSystem.allInstances()->size()=0
```

On the other hand, if `RatingSystem` has a runtime binding, the ontology constraint may state that the selection of the feature implies that every `Review` must have an associated `RatingSystem`, and state nothing about its elimination:

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

Table 3.1: Constraints [15]

```
context Review inv:
    <<RatingSystem>> implies self.ratingSystem->size()=1
```

Therefore, different binding times can impose different mapping semantics for the same feature. An ontology constraint that involves features of different binding times is partially evaluated in stages as feature IDs become replaced by their Boolean values through a progression of time. In fact, this staged configuration notion [13] can be applied to a set of feature models and ontology constraints as a whole.

Complex Semantics

The simple semantics mostly involved effective removal of schema elements. As a feature model becomes more complex, the semantics of mapping become more complex as well. We can classify the more complex semantics into two categories: children-independent and children-dependent semantics.

Children-independent semantics. Consider Figure 3.7, which shows an extension of Figure 3.5(a), which concerns Figure 3.4, to give the business perspective the options to allow multiple payments in general and multiple payments from different accounts if payments are made from accounts at all. Clearly, if payments can be made from different accounts, then multiple payments can be made. However, the option of allowing multiple payments is not dependent on the option of allowing multiple payments from different accounts or accounts. Namely, while

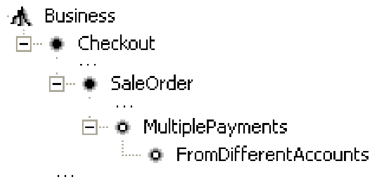


Figure 3.7: Figure 3.5(a) extended with children-independent semantics

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

context SaleOrder inv:
 (<<MultiplePayments>> and not <<FromDifferentAccounts>>) implies
   (self.cashReceiptCommitment->forAll(commit1, commit2: CashReceiptCommitment |
commit1 <> commit2 implies commit1.cashReceipt.account=commit2.cashReceipt.account))
```

Figure 3.8: Children-independent semantics

mapping semantics of `FromDifferentAccounts` depends on `MultiplePayments`, the reverse is not true. In children-independent semantics, the mapping semantics of the parent feature are independent of those of a child feature. In these cases, we can write separate constraints for the parent feature and the child feature, as shown in Figure 3.8.

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. Figure 3.9 extends Figure 3.5(a) to define what `Account`-requiring `Checkouts` are in Figure 3.3. `Checkouts` must be restricted using the configurations of features below `AccountRequired`. Constraints for children-dependent semantics can become complex if there is complex variability in the subtree and/or if the subtree is deep. It may require incrementally building the constraint with variability encoded and/or breaking down a large constraint without variability consideration and inserting variability. The latter approach is shown. First, the constraint is specified without inserting variability, as shown in Figure 3.10.

Then variability is inserted by breaking down the constraint, for example, into understandable functions, as shown in Figure 3.11.

Note that `AccountRequired` and `ProductType` features are not specified in the con-

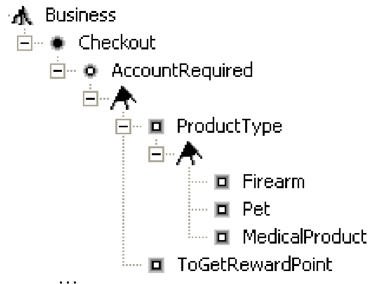


Figure 3.9: Figure 3.5(a) extended with children-dependent semantics

```
context Checkout inv:
    (self.shoppingCart.productType.select(p: ProductType |
        p.oclIsTypeOf(MedicalProduct) or
        p.oclIsTypeOf(Pet) or
        p.oclIsTypeOf(Firearm))->notEmpty() or
           self.rewardPointAmount->notEmpty())
              implies (self.account->notEmpty())
```

Figure 3.10: Children-dependent semantics without variability

```
context Checkout inv:
  (self.isProductTypeApplicable() or
      self.isRewardPointApplicable())
         implies (self.account->notEmpty())

context Checkout::isProductTypeApplicable(): Boolean
post: result = (shoppingCart.productType.select(
   p: ProductType | (p.oclIsTypeOf(MedicalProduct) and
     <<MedicalProduct>>) or
      (p.oclIsTypeOf(Pet) and <<Pet>>>) or
        (p.oclIsTypeOf(Firearm) and <<Firearm>>))->notEmpty())

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

Figure 3.11: Children-dependent semantics with variability

straints in Figure 3.11. 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 (for example, eliminating `ProductType` makes `isProductTypeApplicable()` false), 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 each node of 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.

Chapter 4

Domain Modeling Approaches

The fact that feature models are views on ontologies strongly suggests that feature modeling and ontology modeling are closely intertwined as domain modeling techniques in the MDSPL approach. In this chapter, two novel domain modeling approaches that incorporate feature modeling and ontology modeling in synergetic ways are proposed. Basically, the view projection approach projects feature models from already existing ontologies, while the view integration approach integrates already existing feature models to form an ontology. For each approach, the process involved is outlined and demonstrated. Then variations of each approach are discussed for further insight.

4.1 View Projection

In view projection, a rich and mature ontology is assumed to exist. Feature models, which provide outlines of different themes on an ontology as its views, are syntactically constructed using the ontology. Semantics of mapping are then specified, which may in turn allow the feature models to be modified syntactically and semantically. Note that Figure 3.5(a) and Figure 3.5(b) are views projected from Figure 3.3 and Figure 3.4.

4.1.1 Process Overview

Figure 4.1 shows the steps involved in the view projection process.

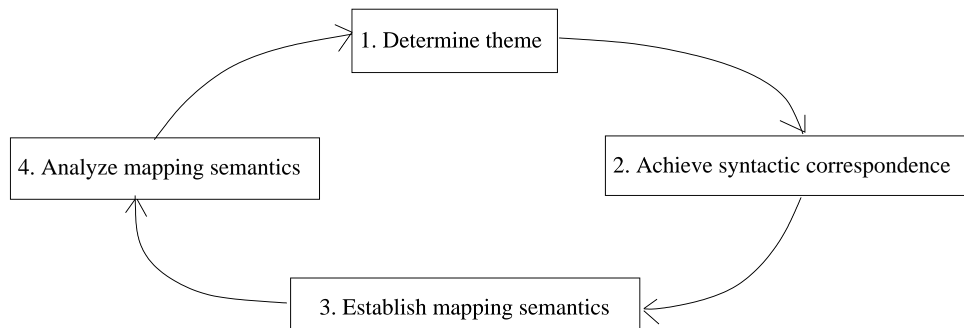


Figure 4.1: View projection process

In step 1, a theme on the ontology is determined by the domain modeler. This theme, a specialization of the overall concept modeled by the ontology, represents the concept of the feature model to be projected, which in turn sets a scope for the feature model. The root of the feature model, which represents its concept, is created.

In step 2, features underneath the root are created. Collectively, the features should be conceptually related to the ontology. While semantics of mapping may be required to precisely define this relation, at this step, achieving syntactic correspondence is sufficient. Syntactic correspondence here can, but does not have to be, a physical traceability link; it can also be a more logical one, such as a rule or a constraint, that gives an insight on how mapping semantics should be specified. There are two ways of creating syntactic correspondence. One way is to derive the features from the ontology, for example, by exploring the ontology syntax (elements like classes and associations) and semantics (constraints on individuals). Exploring the ontology may give hints on appropriate feature names, feature hierarchy and other basic constraints. Naturally, syntactic correspondence is achieved when features are derived from the ontology. For example, in Section 3.2, Figure 3.6 shows feature models that could have been derived by exploring their respective ontology regions. Another way is to first create a feature without considering the ontology and then to achieve syntactic correspondence. This requires finding ontology elements that can eventually map to the feature semantically, which arguably requires more effort than deriving features from the ontology. Also, feature derivation using ontology facilities may be automated. An initial feature model, with adequate feature names, feature hierarchy, and other basic constraints like cardinalities, is produced in this step.

In step 3, the semantics of mapping between the feature model and the ontology are established from the syntactic correspondence achieved in the previous step. New ontology constraints are specified, which are written over some of the features. This step presents an opportunity to improve the initial feature model, for example, by making cardinalities appropriate and moving features to proper locations. While this step is largely a manual process, if the mapping semantics are very simple, like existential dependencies between ontology elements and features, then they may be derived from the syntactic correspondence automatically. For example, for an ontology element and a feature element that syntactically correspond in an isomorphic way, such as `ProductReturn` class and feature in Figure 3.4, a constraint that prohibits the instances of the class if the feature is false may be automatically derived.

In step 4, mapping semantics are analyzed in order to improve the feature model. For example, analysis may reveal that two features are equivalent, since they imply the same ontology constraint, or that one implies or excludes another, due to some relationship between their ontology constraints. This step yields additional constraints on the feature model. These additional constraints not only influence the semantics of feature model, but may also influence its syntax, which represents the feature hierarchy and feature variability. For instance, a feature that was directly under the root may be moved to more precise locations due to new implications. Also, a feature may be prevented from appearing under some features, using constraints.

If the feature model still needs to be improved, step 1 follows for improvement of the theme and the process repeats. Additional feature models may be projected as well. Eventually, a set of coherent projections should provide outlines of unique themes on the ontology.

4.1.2 Process Demonstration

The process described is demonstrated by projecting Figure 3.5(a) from Figure 3.3 and Figure 3.4. For achieving syntactic correspondence through ontology exploration, the state-of-the-art Meta Object Facility (MOF) Query/View/Transformation (QVT) standard [2] for Model Driven Architecture [35] is used. UML/OCL and propositional constraint facilities are used for analyzing mapping semantics.

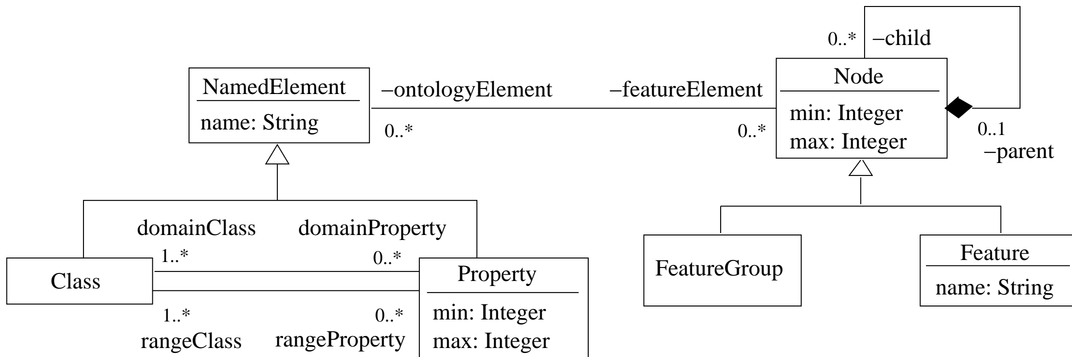


Figure 4.2: Ontology and feature model metamodels

Step 1

A feature model (i.e. Figure 3.5(a)) that outlines the "business" theme on the REA ontology in Figure 3.3 and Figure 3.4 and is scoped to consider customer activities is desired.

Step 2

Using the QVT *Relations* language, traceability links between ontology elements and features are specified declaratively, which allows automatic achievement of syntactic correspondence. Before the language can be used however, the metamodels of the elements to be related, i.e. ontology metamodel and feature metamodel, need to be specified.

Metamodels Figure 4.2 shows the ontology metamodel and the feature metamodel, minimally designed to allow Relations to be used. `Class` and `Property`, subclasses of `Named-Element` related through domain and range relationships, form the ontology metamodel. For example, for `Checkout-SaleOrder` association in Figure 3.3, `Checkout` and `SaleOrder` are instances of `Class` and `Checkout.saleOrder` and `SaleOrder.checkOut` are instances of `Property` with `min` and `max` equal to one. While not shown in the metamodel, the classes denoted by the custom icons in Figure 3.2, such as `EconomicEvent` and `Con-tract`, and associations between them, like `produce` and `clause`, are subclasses of `Class` and `Property` respectively.

`Node`, `Feature` and `FeatureGroup` form the feature model metamodel, which is a simplified version of the one in [28]. `Feature` and `FeatureGroup` are subclasses of `Node`,

which has min and max cardinality and parent-child association. For example, for Checkout-SaleOrder subfeature relationship in Figure 3.5(a), Checkout and SaleOrder are instances of Feature with min and max equal to one and Checkout and SaleOrder are linked by parent-child. ontologyElement-featureElement association defines M-N traceability links between ontology and feature elements.

QVT Relations Figure 4.3 shows the relations between ontology elements and feature elements specified using QVT Relations. Metamodel classes are assumed to be in one of two models, namely, OntologyMetamodel for ontology metaclasses and FeatureMetamodel for feature model metaclasses. A *transformation* (Figure 4.3, line 2) and its *relation*s (Figure 4.3, lines 4, 17 and 31) define traceability links between classes in OntologyMetamodel and FeatureMetamodel models, which must be satisfied by the conforming ontology elements and feature model elements. While feature model elements are created or modified to satisfy the traceability definition, ontology elements are never altered, making the transformation unidirectional. To ensure this, OntologyMetamodel is qualified as read-only (checkonly domain) and FeatureMetamodel is qualified as read-write (enforce domain) throughout the *relations*. The user specifies rules (Figure 4.3, lines 48, 58, and 67) that must be satisfied by the *relations*. Figure 4.3 is explained in detail. For more information on QVT, the reader is referred to [2].

The transformation syntacticCorrespondence (Figure 4.3, line 2) takes the two models as input. The transformation requires that its top relation query (Figure 4.3, line 4) hold, which in turn requires that some other relations hold. In query, ontoElement (Figure 4.3, line 6) represents each instance of NamedElement and fmFeature (Figure 4.3, line 7) represents each instance of Feature. The user specifies what ontoElements are queried. Features corresponding to these ontoElements are created under desirable parent features, or fmFeatures. More specifically, if an ontology element matches the query specified by the user (Figure 4.3, line 10) and a feature is considered a desirable parent (Figure 4.3, line 11), then elementCorrespondence needs to hold between the ontology element and a child of the feature (Figure 4.3, line 14). The user defines matchesQuery (Figure 4.3, line 48) and matchesParent (Figure 4.3, line 58) as OCL functions. For the particular ontology in Figure 3.3 and Figure 3.4, EconomicEvents provided by Customer are queried using matchesQuery (Figure 4.3, lines 50-54), which returns Checkout, Browsing,

Registration and Review from Figure 3.3. Corresponding features are created under each of the parents queried using matchesParent; in this particular example, they are created only under the feature model root (Figure 4.3, lines 60-63). Each pair of ontology element and its corresponding feature is passed on as arguments (Figure 4.3, line 14) to ontoElement and fmFeature respectively in elementCorrespondence (Figure 4.3, lines 21-22). The corresponding feature is made to have the same name as the ontology element (Figure 4.3, line 26) and a traceability link to the ontology element (Figure 4.3, line 27). For example, a new feature Checkout, with default cardinality [1..1] is created with Checkout as its ontologyElement.

matchesSubfeature is used to map ontology properties (i.e. associations) to feature parent-child associations. When a correspondence is achieved between a class and a feature, they are passed as arguments to the subfeature relation (Figure 4.3, line 28), which uses matchesSubfeature to check if properties of that class should be mapped to parent-child links (Figure 4.3, line 41). For a property that matches (in this particular example, kinds of Produce, Term, and Commitment due to Figure 4.3, lines 69-73), a new child feature is created that corresponds to the property's range class and has the property's minimum cardinality (Figure 4.3, lines 44-45) [1]. For example, for Checkout feature, whose corresponding class has Checkout.saleOrder property that is of the type Produce, a mandatory child named SaleOrder is created. Due to recursion (Figure 4.3, line 44), further children, including the optional Backorder, are created.

Figure 4.4(a) shows the generated feature hierarchy. Some uninteresting features are crossed out, while some interesting features, bound by the rectangle, are added manually. Assume that the new features are added underneath the root as their exact locations are not known yet.

Note that the demonstration, meant to give a taste of how ontology facilities can be used to achieve syntactic correspondence, was considerably simplified. Feature groups, which are not very difficult to consider, were left out as they would have convoluted the example. The transformation relations did not take into account seriously the consequences of yielding conflicting or redundant results. Often, inconsistencies in model relations may be fixed in many, sometimes infinitely many, possible ways; therefore, model synchronization typically requires manual intervention. However, being more precise and constraining when specifying relations can greatly

[1] Since basic feature models are desired, maximum feature cardinality is always assumed to be 1.

```
1  transformation syntacticCorrespondence(ontoMetamodel: OntologyMetamodel,
2      fmMetamodel: FeatureMetamodel)
3  {
4     top relation query
5     {
6        checkonly domain ontoMetamodel ontoElement: NamedElement{}
7        enforce domain fmMetamodel fmFeature: Feature{child=c:Feature{}}
8
9        when
10       {  matchesQuery(ontoElement);
11          matchesParent(fmFeature); }
12
13       where
14       {  elementCorrespondence(ontoElement, c); }
15    }
16
17    relation elementCorrespondence
18    {
19       featureName, ontoName: String;
20
21       checkonly domain ontoMetamodel ontoElement: NamedElement{name=ontoName}
22       enforce domain fmMetamodel fmFeature: Feature
23          {ontologyElement=o:NamedElement{}, name=featureName}
24
25       where
26       {  featureName=ontoName;
27          o = ontoElement;
28          subfeature(ontoElement, fmFeature); }
29    }
30
31    relation subfeature
32    {
33       propertyMin, featureMin: Integer;
34
35       checkonly domain ontoMetamodel ontoElement: Class
36          { domainProperty=dp:Property{rangeClass=rc:Class{}, min=propertyMin} }
37       enforce domain fmMetamodel fmFeature: Feature
38          { child=c:Feature{min=featureMin, max=featureMax} }
39
40       when
41       {  matchesSubfeature(dp); }
42
43       where
44       {  elementCorrespondence(rc, c);
45          featureMin=propertyMin; }
46    }
```

Figure 4.3: QVT relations for achieving syntactic correspondence

```
47    // user specifies what ontology elements are desired
48    function matchesQuery(namedElement: NamedElement): Boolean
49    {
50       if (namedElement.oclIsTypeOf(EconomicEvent) and
51          namedElement.oclAsType(EconomicEvent).provider.name=Customer) then
52             true
53       else false
54       endif;
55    }
56
57    // user specifies where the corresponding features will be located
58    function matchesParent(node: Node): Boolean
59    {
60       if (self.parent->isEmpty()) then
61          true
62       else false
63       endif;
64    }
65
66    // user specifies
67    function matchesSubfeature(property: Property): Boolean
68    {
69       if (property.oclIsTypeOf(Produce) or property.oclIsTypeOf(Term) or
70             property.oclIsTypeOf(Commitment)) then
71          true
72       else false
73       endif;
74    }
75 }
```

QVT relations for achieving syntactic correspondence (continued)

reduce the number of resolution possibilities and thus facilitate automation. An overview of model synchronization in the context of software product lines is presented in [28].

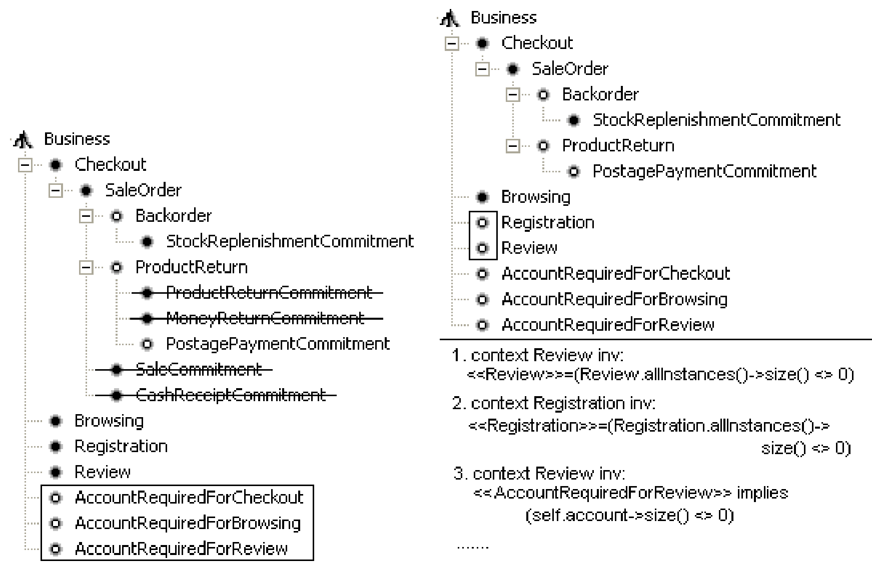
Step 3

With syntactic correspondence achieved, mapping semantics are established. Figure 4.4(b) shows Figure 4.4(a) with some minor changes that have made `Registration` and `Review` optional. Most of the features in Figure 4.4(b) influence the existence of their corresponding classes with respect to the entire e-commerce product line. For example, the selection or elimination of `Backorder` implies that every sale order must either support or not support backorders. Mapping semantics for some of the features are shown as additional ontology constraints below the feature model. Consider constraint 2. If `Registration` is selected, then there must be at least one instance of `Registration` in the system. Otherwise, there cannot be any instance of it. This constraint, more restrictive than the third constraint in Table 3.1, states that the system resources devoted to registration cannot go wasted by enforcing that they must be used at least for one registration event. Constraint 1 similarly influences `Review` instances. Constraint 3 states that if an account is required to perform reviews, then a review must be associated with an account. It is assumed that constraints for `Checkout` and `AccountRequiredForCheckout` and constraints for `Browsing` and `AccountRequiredForBrowsing` are specified in an identical manner as constraints 1 and 3 for `Review` and `AccountRequiredForReview`.

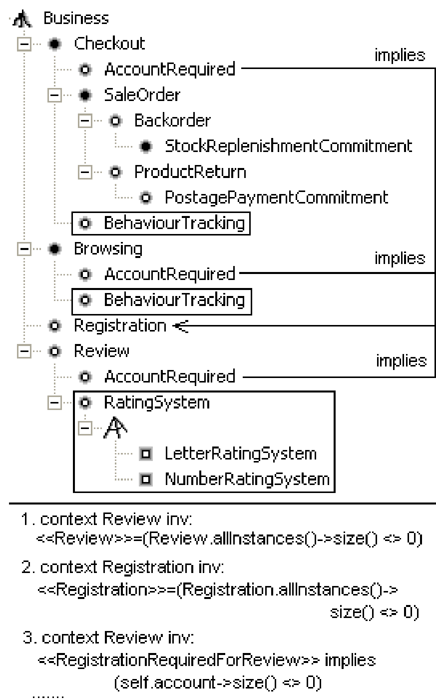
Simple mapping semantics may be derived purely from syntactic correspondence achieved in the previous step. This can be easily achieved for constraints 1 and 2. However, for more complex features like `AccountRequiredForReview`, establishing mapping semantics is largely a manual process.

Step 4

The established mapping semantics are analyzed to improve the quality of the feature model. The ontology constraints in Figure 4.4(b) suggest that there are unspecified connections between `Review`, `Registration` and `AccountRequiredForReview`. Consider constraint 3. If `AccountRequiredForReview` is selected, then there must be at least one `Review` instance (otherwise, the constraint would not be needed at all) and there must be at least one

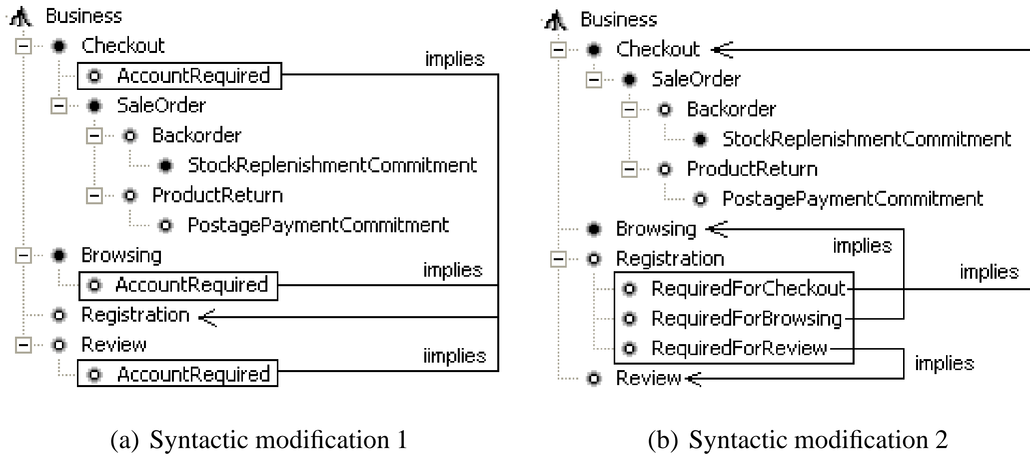


(a) Step 2 result (b) Step 3 result



(c) Step 4 result

Figure 4.4: Step-wise projection of Figure 3.5(a) from Figure 3.3 and Figure 3.4



(a) Syntactic modification 1 (b) Syntactic modification 2

Figure 4.5: Possible results of semantic analysis of Figure 4.4(b)

`Account` instance (otherwise, `self.account <> 0` would not be possible), which means that there must be at least one `Registration` instance (since `Registration-Account` is 1 to 1). Two constraints can be derived from this observation: `AccountRequiredFor-Review⇒Registration` and `AccountRequiredForReview⇒Review`. While these constraints may simply be added to the feature model in Figure 4.4(b), we can go one step further and make a syntactical change to Figure 4.4(b) since implications can be modeled as sub-features [6]. One possibility is to change it to Figure 4.5(a), where `AccountRequiredFor-Review` is placed under `Review` as `AccountRequired` (with the name change intended to improve readability) and an implication to `Registration` is added. Another possibility is to change it to Figure 4.5(b), where the feature is placed under `Registration` as `RequiredForReview` (again, the name change) and an implication to `Review` is added. Similarly, `AccountRequiredForCheckout` and `AccountRequiredForBrowsing` are changed. Figure 4.5(a) is chosen as it closely reflects Figure 3.5(a), the desired end result. Interestingly, the `Registration` subtree in Figure 4.5(b) closely reflects the counterpart of Figure 3.5(a), Figure 3.5(b).

Despite its simplicity, the example highlights the important notion that analysis of mapping semantics can improve a feature model not only semantically through additional constraints, but also syntactically, possibly allowing the user to choose from many desirable hierarchies. Simple mapping semantics may not be difficult to analyze and may even allow automatic analysis. More

complex mapping semantics may require more sophisticated binary and ontology constraint facilities.

Through another iteration of the view projection process, changes, including the addition of a feature group and some features, are made to Figure 4.5(a) to yield the intended result, Figure 4.4(c).

4.1.3 View Projection Classification

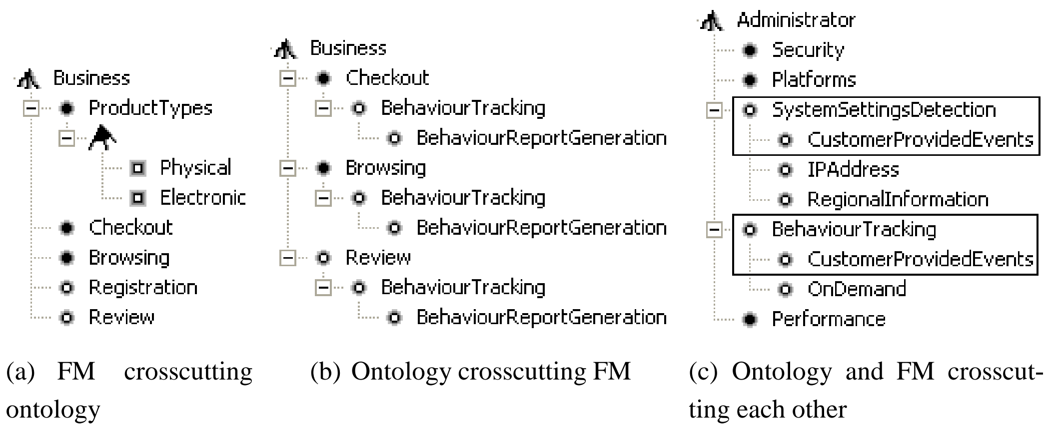
It seems that feature models, as views on ontologies, should be at a similar level of abstraction as the ontologies. However, what defines abstraction similarity is not clear and consequently, what defines appropriate or desirable view projections is not clear. A classification of view projection aims to provide a basis for understanding this issue and to give a further insight on the boundary of view projection. Four categories of view projection are discussed: localized, crosscutting in ontology, crosscutting in feature model, and crosscutting in both.

Localized

In localized view projection, a feature model has features that bear strong syntactical and semantical resemblance to their ontology counterparts. A feature typically maps to a well-defined and coherent region in the ontology. Feature models shown up to this point have been projected as views using this type of view projection.

Crosscutting in Ontology

Typically, there are a small number of key, very high-level decisions that influence virtually every aspect of a domain. In the Business-to-Consumer e-commerce domain, deciding what the supported product types are, for example, including electronic and physical product types, influences numerous aspects. For example, eliminating `Physical` in Figure 4.6(a) eliminates `ShippingAddress`, `Physical`, `Backorder`, and `PostagePaymentCommitment` in Figure 3.3 and Figure 3.4, which in turn eliminates some related elements. Syntactic correspondence between `Physical` and these ontology elements can be achieved by manually setting `Node.ontologyElement` link to these elements on the `Physical` feature. While mapping semantics can be established by having a separate constraint for each of the ontology elements,



(a) FM crosscutting ontology (b) Ontology crosscutting FM (c) Ontology and FM crosscutting each other

Figure 4.6: Examples of classes of view projection

```
1 context NamedElement::matchesQuery(): Boolean
2   post: result = (self.name='ShippingAddress' or
3     self.name='Physical' or self.name='Backorder' or
4       self.name='PostagePaymentCommitment')
5
6 context REAObject inv:
7   (not <<Physical>>) implies (not self.type.oclAsType(NamedElement).matchesQuery())
```

Figure 4.7: A technique for crosscutting ontology view projection

Figure 4.7 shows a more elegant technique. `matchesQuery` (Figure 4.7, line 1) is used to query desired ontology elements, which is done rather explicitly for this example. We assume that there exists a global superclass, like the Java `Object` for Java classes, for the ontology elements, called `REAObject`. A constraint is specified on this class that ensures that if `Physical` feature is eliminated, no REA object can be of the queried classes (Figure 4.7, line 6).

Arguably, any set of scattered ontology elements may become a 'coherent' region by connecting them by a single ontology element through mandatory associations. One could imagine such associations emanating from `Physical` class to `ShippingAddress`, `Backorder` and `PostagePaymentCommitment` and `Physical` feature could simply control its existence. However, the `Physical` feature here represents a key business decision that is in fact at a higher level of abstraction than the domain concepts that are closely related to the physical product type. In this sense, the feature does crosscut the ontology and it is appropriate to have crosscutting

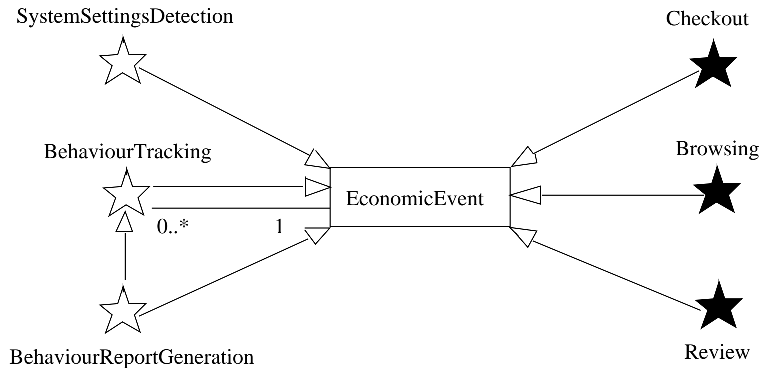


Figure 4.8: Enterprise-provided and customer-provided events

mapping semantics.

In this type of projection, a feature model is at a higher level of abstraction than the ontology. Whether or not such feature models can be called *views* on ontologies remains to be seen.

Crosscutting in Feature Model

Conversely, ontology elements may crosscut a feature model. In Figure 3.3, `BehaviourTracking` is associated with `Checkout` and `Browsing`. Compare this to Figure 4.8, where `BehaviourTracking` is a more general concept that can apply to any `EconomicEvent` [2]. `BehaviourReportGeneration` is seen as a kind of `BehaviourTracking`. In Figure 4.6(b), the events to which `BehaviourTracking` and `BehaviourReportGeneration` may apply, i.e. `Checkout`, `Browsing`, and `Review`, are made explicit through the feature hierarchy. Figure 4.9 shows the view projection mechanism, which uses OCL constraints and QVT relations in Figure 4.3, with the three functions in Figure 4.3 replaced by the three functions in Figure 4.9, lines 1, 8, and 17. Assuming that `Checkout`, `Browsing` and `Review` features already exist and are associated with their corresponding ontology elements, we can query them as desired parent features (Figure 4.9, line 8) under which `BehaviourTracking` will be added (Figure 4.9, line 1). `BehaviourReportGeneration` is added as a subfeature of `BehaviourTracking` since inheritance is specified as a property corresponding to the subfeature relationship (Figure 4.9, line 17).

[2] EconomicEvent, previously modeled as a metaclass, is modeled here as a class for the sake of the example.

```
1  function matchesQuery(namedElement: NamedElement): Boolean
2  {
3     if (namedElement.name = 'BehaviourTracking') then true
4     else false
5     endif;
6  }
7
8  function matchesParent(feature: Feature): Boolean
9  {
10    if(feature.ontologyElement->select
11       (o  o.oclIsTypeOf(EconomicEvent) and
12          o.provider='Customer').notEmpty()) then true
13    else false
14    endif;
15 }
16
17 function matchesSubfeature(): Boolean
18 {
19    if (property.oclIsTypeOf(Inheritance)) then true
20    else false
21    endif;
22 }
23
24 context Feature inv:
25    ((self.name='BehaviourTracking' and not self.selected)
26        implies (self.parent.ontologyElement->forAll(o: EconomicEvent
27           o.behaviourTracking.isEmpty()))) and
28
29    ((self.name='BehaviourReportGeneration' and not self.selected)
30        implies (self.parent.ontologyElement->forAll(o: EconomicEvent
31           not o.behaviourTracking.oclIsTypeOf(BehaviourReportGeneration))))
```

Figure 4.9: A technique for crosscutting feature model view projection

Mapping semantics can be specified through a constraint on feature configurations (Figure 4.9, line 24). A feature in a configuration is modeled almost identically to a feature in a feature model [28], with the addition of `selected` Boolean flag indicating its configuration value. The constraint states that if a `BehaviourTracking` feature is eliminated, the `EconomicEvent` corresponding to its parent must have a null `behaviourTracking` link (Figure 4.9, lines 25-27). Also, if `BehaviourReportGeneration` is eliminated, then the appropriate `behaviour-Tracking` link cannot be connected to an instance of `BehaviourReportGeneration` (Figure 4.9, lines 29-31).

This type of projection highlights an important philosophical difference between configuration modeling and solution modeling. In configuration modeling, specialization is often used to spell out the variability, while in solution modeling, generalization is often used to encourage reusability and to be concise. Whether or not specialized feature models can be considered as *views* on more generalized ontologies remains to be seen.

Crosscutting in Each Other

Combining the previous two categories, a projection may incorporate features crosscutting in ontology and ontology elements crosscutting in feature model. Figure 4.6(c) shows a feature model describing system capabilities relevant to e-commerce administrators. View projection is specified to relate customer-provided events to administration events. A `CustomerProvided-Events` feature crosscuts Figure 4.8 by collecting `EconomicEvents` provided by customers. This feature also crosscuts Figure 4.6(c), being placed under the two administration events. Mapping semantics, although a bit complex, can be specified using the techniques shown for the last two categories of projection.

From these four categories, it is evident that view projection does not merely involve deriving feature models from ontologies, but that it also involves relating feature models and ontologies using composition mechanisms.

4.1.4 Benefits and Limitations

View projection addresses a serious limitation of ontology modeling: its tendency to produce monolithic ontologies that are difficult to modularize. A feature model allows ontology modularization from a viewpoint by providing a set of constraints that is intuitively configurable through a hierarchy. Projecting multiple feature models from an ontology enables collaboration by allowing different viewpoints to make unique contributions to a shared knowledge repository. View projection also enables lightweight feature model development. Ontology-based facilities can be used to specify queries and rules that can automate much of feature model development, as was shown using QVT and OCL.

View projection is not without limitations, however. Firstly, a mature ontology needs to exist. This is not a problem for mature industries in which richly specified documents can be converted into a mature ontology. Also, it is not a problem when such rich documents are required anyway, for example, for standards compliance. However, developing a mature ontology is generally a very difficult task. Also, view projection needs to be a controlled process. Without control, views that are incoherent and conflicting may be projected, which may actually hinder collaboration. It seems that the views should be at similar levels of abstraction, which requires that the ontology is properly constructed and that boundaries of projections be understood.

4.2 View Integration

For carrying out tasks in general, in many cases, devising an outline before delving into details may be more practical. Similarly, in domain modeling, having feature models before an ontology may be more suitable. For such cases, a domain modeling approach, called *view integration*, that is opposite but complementary to view projection is proposed. In view integration, feature models at similar levels of abstraction are assumed to exist and are used to develop and refine an ontology. This feature-driven domain modeling approach allows the development of shared understanding between feature modelers, enabling collaboration. Also, it allows agile ontology development, as only the concepts that are common to the multiple feature models need to be modeled in the ontology.

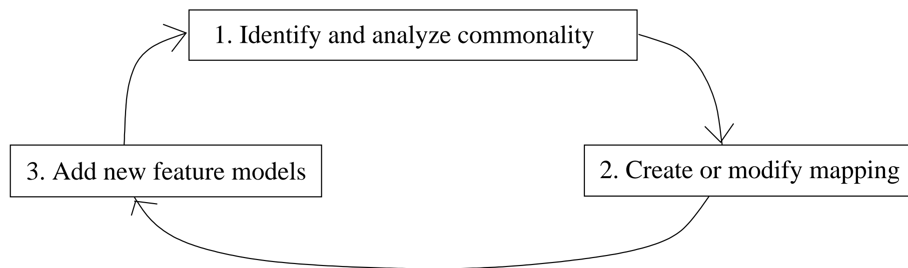


Figure 4.10: View integration process

4.2.1 Process Overview

Figure 4.10 shows the steps involved in the view integration process.

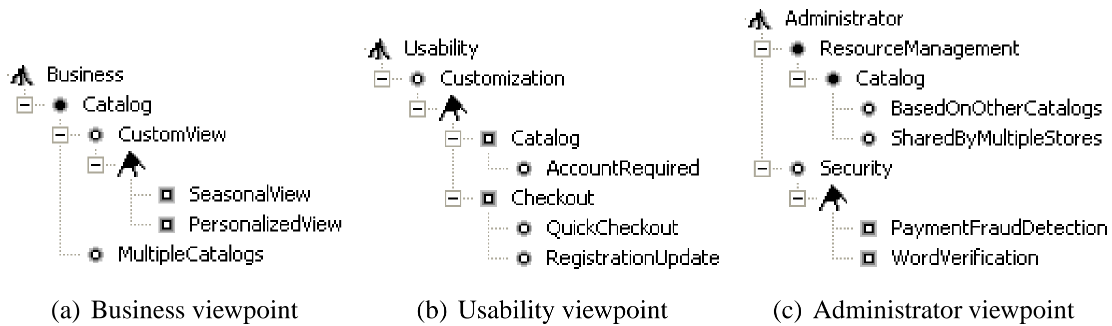
In step 1, the common notions between the feature models are identified and analyzed. This is largely a manual process that requires understanding the semantics of the feature models as a whole. For example, two features from two different viewpoints (represented by feature model roots), despite their name differences, may have identical semantics in terms of their common notions. They may be tied in a less obvious way, for example, through relationships between the common notions. The goal of this step is to yield an understanding of the commonality that is adequate to allow the commonality to be modeled.

In step 2, the acquired understanding of the commonality is modeled explicitly by creating or modifying the mapping between the feature models and the ontology. The ontology may need to be structurally and/or semantically changed to allow the change in mapping. Feature models may be slightly changed, for example, to reflect constraints between features that are evident through the new mapping to the ontology.

In step 3, a feature model may be added to the list of feature models being integrated and the process may be repeated.

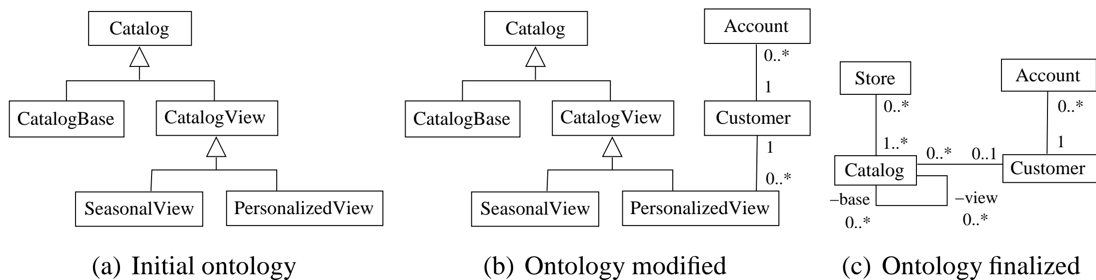
4.2.2 Process Demonstration

The process described is demonstrated for ontologies expressed using UML class diagrams and basic feature models. The ontologies, which involve only `Resources`, are shown using the regular class diagram notation, rather than the REA profile, to keep the example more uniform. Feature models of the viewpoints *business*, *usability*, and *administrator*, shown in Figure 4.11,



(a) Business viewpoint (b) Usability viewpoint (c) Administrator viewpoint

Figure 4.11: Integrated viewpoints



(a) Initial ontology (b) Ontology modified (c) Ontology finalized

Figure 4.12: Integrating ontology

are integrated.

Business

The business viewpoint in Figure 4.11(a) models variability in `Catalog`. Two kinds of `CustomViews` may be allowed. While a `CustomView` may be considered a `Catalog` itself, `MultipleCatalogs` refers to having multiple non-view `Catalogs`, which may be referred to as `CatalogBases`. The corresponding ontology, Figure 4.12(a), is modeled in an analogous way as the feature model, with most of the feature decisions having an analogous effect on the existence of their corresponding ontology elements. An exception is the feature `MultipleCatalogs`, which constrains `CatalogBase` as shown in Figure 4.13.

```
context CatalogBase inv:
   (not <<MultipleCatalogs>>) implies (CatalogBase.allInstances()->size() <= 1)
```

Figure 4.13: MultipleCatalogs constraint

Usability

A feature model concerning the usability of end-user interfaces of an e-commerce product line, shown in Figure 4.11(b), is to be integrated with the business feature model. By simply looking at the feature names and hierarchies of the two feature models, it can be concluded that the only commonality is between `PersonalizedView` in the business viewpoint and `Catalog` and `AccountRequired` in the usability viewpoint. `Customer` and `Account` classes are added in Figure 4.12(a) to model `Catalog` and `AccountRequired` features, producing Figure 4.12(b). `PersonalizedView <-> Catalog` feature constraint is added, since personalizing a catalog is no different than customizing a catalog for the end-user. Note that features unique to the usability perspective, those in the `Checkout` subtree, are not mapped.

Administrator

Another feature model, this time concerning the administrator, shown in Figure 4.11(c), is integrated with the other two feature models. It is obvious that only the two features under `Catalog` are common to the other feature models. While `BasedOnOtherCatalogs` may seem to be synonymous with the `CatalogView` class in Figure 4.12(b) at first glance, it seems to imply something more general upon analysis. The literal translation seems to refer to the capability of defining a catalog using the contents of other catalogs and it seems to emphasize these origins. In this sense, the feature encodes a more implementation-oriented notion than the feature `CatalogView`, which is appropriate since the viewpoint, administrator, should rightfully be more concerned about catalog implementation than the business viewpoint. `base-view` association on `Catalog` is introduced to give semantics to `BasedOnOtherCatalogs`, as shown in Figure 4.12(c). Since the distinction between base and view catalogs, as well as between seasonal and personalized catalogs, can be made using this association and some extra information, the `Catalog` inheritance hierarchy is removed. The constraint `(SeasonalView or PersonalizedView)`⇒`BasedOnOtherCatalogs` is added. `Customer` class is now as-

```
context Catalog inv:
   (<<PersonalizedView>> implies (self.customer->notEmpty() implies self.base->notEmpty())) and
   ((not <<PersonalizedView>>) implies (self.customer->isEmpty()))
```

Figure 4.14: PersonalizedView constraint

sociated with `Catalog` class, making `PersonalizedView` semantics, shown in Figure 4.14, more complex than before.

The mapping semantics of `MultipleCatalogs` in Figure 4.13 is no longer valid since `CatalogBase` no longer exists. Fortunately, this problem can be solved through `Store-Catalog` association, which is needed anyway to express the semantics of `SharedByMultipleStores`, which state that `Catalog.store <= 1` upon the feature's elimination. Assuming that a store is associated only with base catalogs, elimination of `MultipleCatalogs` implies `Store.catalog <= 1`, preventing multiple base catalogs.

4.2.3 View Integration Classification

While facilities like vocabulary and taxonomy analysis may be used for identifying and analyzing commonality between feature models, this step (step 1) seems to be largely a manual process that is difficult to understand in a systematic way. On the other hand, creating/modifying mapping (step 2) seems more understandable. For further insight on view integration, the approach is classified in terms of different kinds of mapping modifications.

Additive modification. In additive modification, new mapping semantics are established. For example, to establish mapping for the new feature `AccountRequired` in Figure 4.11(b), `Customer` and `Account` classes were added in Figure 4.12(b). While existing ontology structure and semantics may be used to establish the mapping for a new feature, typically, especially when the ontology is in early stages of development, new ontology elements are likely to be needed.

Altering modification. In altering modification, existing mapping semantics are altered. Most, if not all, of the semantics are preserved since alteration merely involves specifying existing semantics in another way. For example, the semantics of `PersonalizedView` and `MultipleCatalogs` in Figure 4.11(a), although changed due to the introduction of `Based-`

OnOtherCatalogs in Figure 4.11(c), are not altered. This type of modification typically involves re-establishing mapping semantics after the ontology is restructured.

Subtractive modification. In subtractive modification, existing mapping semantics are removed. For example, from Figure 4.12(b) to Figure 4.12(c), PersonalizedView and SeasonalView classes are removed. While the semantics of PersonalizedView feature is preserved, as evident in Figure 4.14, the semantics of SeasonalView feature is not. Arguably, seasonal views may be considered as views that are not personalized views. However, there is some semantic loss since the notion of *season* is not present anywhere in Figure 4.12(c), whereas there was a symbol (feature name) explicitly representing the notion in Figure 4.12(b). Subtractive modification may be suitable when the commonality between feature models is being reduced to make the feature models more independent. Indeed, a seasonal catalog may be a highly business-specific concept that does not need to be a part of the shared understanding with the usability and administrator viewpoints.

Understanding the evolution of mapping in terms of these dimensions may help us better understand view integration as a systematic approach. It may be hypothesized that additive modification and major altering modification should occur in early stages of view integration, promoting expansion of an ontology, while only minor altering modification and subtractive modification should occur in the later stages of view integration, for manageable maintenance.

4.2.4 Benefits and Limitations

View integration addresses a serious limitation of feature modeling: composition. By having a common ontology through which feature semantics can be specified, features from different feature models can be related through a concrete representation for increased understanding. Also, analyzing mapping semantics of the different feature models to the common ontology can alter both configurations (semantics) and syntax of the feature models. View integration also enables agile ontology modeling. Only the intersection of the feature models needs to be modeled, while features specific to a feature model may or may not be modeled. Ontology modeling in view integration is feature-driven. Deriving and evolving an ontology from several outlines only as much as necessary seems to be a much more agile process than working out details from scratch.

View integration is not without limitations, however. Feature models need to be constructed

before any ontology can be created. These feature models must be coherent and ideally, at similar levels of abstraction. Otherwise, an ontology will be developed that will continue to promote integration of misfitting feature models, ultimately disabling collaboration. Creating such feature models requires a good understanding of the domain being modeled in the first place, which can be considered as an *implicit* ontology in the mind of the modeler [3]. As a result, this approach seems to be more appropriate for a group of experts than novices.

[3] After all, how can outlines be developed without a good understanding of what the outlines represent?

Chapter 5

Discussion and Future Directions

The research presented in this thesis is notable in three respects. Firstly, it sheds new light on the nature of feature models. Secondly, it suggests a novel domain modeling paradigm. Thirdly, it suggests opportunities for tool support required for feature modeling and ontology modeling. Each of these points is discussed 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 whether or not cloning should be allowed. The framework 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. A feature model represents a set of possible restrictions on ontologies. 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 more or less similar levels of abstraction. As a result, despite some complex mappings, most of the mappings were manageable. While the mapping mechanism, feature-based restriction, 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 mechanism would become very complex and less manageable.

5.2 Towards a Domain Modeling Paradigm

View projection and view integration are opposite, but complementary domain modeling approaches. While they were described separately, a hybrid domain modeling approach is imaginable. Feature models, projected from an ontology to scope it from different viewpoints, could be evolved separately from the ontology, which could in turn be integrated to evolve the ontology. Conversely, a set of coherent feature models may initially drive ontology development, but later be evolved through evolution of ontology. Devising a roadmap for collaborative domain modeling that incorporates both view projection and view integration is a future work item. Such a roadmap could form a basis for what could be a novel domain modeling paradigm that drives solution modeling in MDSPL.

5.3 Tool Support

Based on the ideas presented in the previous chapters, a list of desirable tool support capabilities can be divided into two categories. The first category contains capabilities required for allowing feature models to be considered as views on ontologies. The second category contains capabilities required for view projection and view integration.

5.3.1 Support for Feature Models as Views on Ontologies

Firstly, a tooling environment that incorporates both feature models and ontologies (but not necessarily feature modeling and ontology modeling) must be conceived. In a heterogeneous environment, the metamodel of feature models is significantly different than the metamodel of

ontologies. For example, feature models may be defined using UML class diagrams while ontologies are defined using OWL. On the other hand, in a homogeneous environment, the metamodels are similar, allowing a common set of technologies to be used. In this thesis, UML was used throughout, which enabled use of closely related technologies, like OCL and QVT. On the other hand, in practice, the metamodels are more likely to be different, requiring some bridging mechanism.

With such an environment in place, tooling should support writing ontology constraints over features, which is trivial. Less trivial is providing constraint facilities for checking and computing the residual of the ontology. The residual of the ontology is the ontology with the feature-configured constraints, which may be transformed into a syntactically simplified ontology. For example, a configured constraint may enforce the cardinality of an association to be [0..0] or enforce a class to be an empty set, in which case the association or the class may be syntactically removed. Partial evaluation techniques, including rule-based simplification of OCL [20], may be used.

Advanced tool support capabilities, like synchronization between ontologies and feature models, may be supported as well. For example, change in the ontology can invalidate mapping. While visible changes to the ontology, like syntactic removal of elements, may be detected easily, subtle changes, like altering semantics of the ontology through constraints, may be very difficult to detect. Also, change in a feature model could invalidate the feature-based constraints, both syntactically and semantically. In a sense, there is an overlap between synchronization and constraint facilities: constraint facilities may be used to address synchronization issues. Roundtripping is a desirable synchronization trait for increased usability. Another advanced capability is binding time support. To associate binding times with both features and feature-based constraints seems to be a non-trivial task that first requires some foundational ideas.

5.3.2 Support for Domain Modeling Approaches

Capabilities supporting view projection and view integration can be distinguished. For achieving syntactic correspondence in view projection, exploration facilities for ontologies and feature models, traceability and guidance rules are desired. Queries, traversals and equivalent classes are some of the exploration techniques available. Traceability, which requires expressing associations between metamodel elements, can most easily be achieved in a homogeneous environment.

Guidance rules, which are user-specified constraints for facilitating syntactic correspondence achievement for particular projections, can be implemented as restrictions on exploration of ontologies and feature models. Fortunately, as shown previously, QVT offers a uniform mechanism for exploration, traceability and guidance rules. But unfortunately, at the time of writing, to the author's best knowledge, despite some prototypes, including IBM's Model Transformation Framework [22], no practical implementation of QVT exists. Support for establishing mapping semantics trivially requires processing feature-based ontology constraints. Analyzing mapping semantics is more difficult. It may be possible to express some OCL constraint fragments as Boolean variables, which are often implied by features, to use binary constraint facilities, like BDDs [4], for deriving constraints between features. However, in many cases, this approach may not scale, requiring more sophisticated constraint facilities. Ultimately, a uniform user interface that supports all of the mechanisms shown for all four kinds of view projection (introduced in Section 4.1.3) is desired.

Tool support for view integration is harder to imagine since it is a more manual process by nature. Vocabulary and taxonomy analysis may be useful for identifying and analyzing commonality between feature models. For example, basic vocabulary analysis may reveal that "Feedback" and "Review" are synonymous words whose corresponding features may be deemed equivalent. Also, taxonomy analysis may reveal that "Raincheck" is a kind of "Backorder", so adding the constraint `Raincheck⇒Backorder` or adding `Raincheck` under `Backorder` may be suggested. A reference ontology may be used to perform vocabulary and taxonomy analysis. Using the REA ontology in Figure 3.3 and Figure 3.4 as the reference ontology, `Checkout` and `Purchase` (a synonym of `SaleOrder`) features may become equivalent. However, if the reference ontology is used heavily, view projection may be more appropriate. Simplification of mapping semantics, conflict detection and resolution are potential wish list items for creating/modifying mapping.

*Approximately, 60 percent of Chapter 6 was taken directly from [15]. My personal contribution to the corresponding section in [15] was approximately 50 percent.

Chang Hwan Peter Kim

As one of the authors of [15], I fully acknowledge the thesis author's statement above as indicated by * and give full permission to use any part of [15]. I also fully acknowledge that the thesis represents original research conducted by the thesis author.

Krzysztof Czarnecki

As one of the authors of [15], I fully acknowledge the thesis author's statement above as indicated by * and give full permission to use any part of [15]. I also fully acknowledge that the thesis represents original research conducted by the thesis author.

Karl Trygve Kalleberg

Chapter 6

Related Work

Bodies of related work can be classified into feature dependency analysis, work on semantics of feature models, ontology views, viewpoint-oriented requirements-engineering (RE), early aspects, feature-based configuration of models, and expressing feature models in ontology languages. Each is discussed in turn.

6.1 Feature Dependency Analysis

Notable works in this area include those by Lee et al. [30] and Zhang et al. [51]. 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.

6.2 Work on Semantics of Feature Models

Several authors have proposed formal semantics of feature models. Batory [6] proposed formal semantics for feature models based on propositional formulas and grammars. Bontemps et al. [7]

explores the succinctness and expressiveness of feature modeling notations. However, all the notations considered here are in the left part of the spectrum in Figure 2.3.

6.3 Ontology Views

An ontology view is an ontology that is defined using some other ontology. Typically, it is a subset of the original ontology defined through traversals or queries. Notable works involving traversal include those by Noy et al. [34] and Lieberherr et al. [31]. Both works propose graph traversal specifications for extracting portions of an ontology. Besides the difference that the former deals with ontologies in information management while the latter deals with ontologies in software engineering, the latter is considerably more technical. There are numerous notable works involving queries, most of which are focused on query specification. For OWL ontologies, SPARQL (SPARQL Protocol And RDF Query Language) [50], an active W3C working draft, is the most notable work. For UML ontologies, QVT [2], an OMG adopted specification, is the most notable work. Feature models, as views on ontologies, are different from ontology views created from traversals and queries in two respects. Firstly, a feature model, due to its hierarchical nature, is much more viewpoint-oriented than an ontology view. Secondly, a feature model is not a view in a traditional sense as it is constructed specifically (in view projection) from a viewpoint to model variations in the ontology. However, techniques for extracting ontology views have been used for achieving syntactic correspondence in view projection. While UML/QVT was used to demonstrate view projection, note that OWL/SPARQL could have also been used. In fact, the author's earlier work was done using OWL/SPARQL.

6.4 Viewpoint-Oriented RE

There is a large body of work on viewpoints in RE [29, 19, 40]. Viewpoint integration has been explored in the context of requirements engineering, with PREview (*P*rocess and *RE*quirements *view*points) [39] being a notable work. A PREview viewpoint consists of, among other things, *focus*, the perspective of the problem domain taken by the viewpoint, *concerns*, system-crosscutting dimensions that may influence the viewpoint, and *requirements*, the set of requirements relevant to the viewpoint. For example, for an on-board train protection system, which takes corrective

action when a train driver breaks some rules, “safe state assurance” viewpoint may have “detection of dangerous conditions” focus, “safety” and “compatibility” concerns, and “detection of excess speed” and “detection of overshooting” requirements [39]. Integrating two viewpoints, such as “safe state assurance” and “safety standards”, involves first finding the intersecting requirements of two viewpoints. A comparison of the focii of the viewpoints guides this largely manual process. Then the intersecting requirements from one viewpoint are tabulated against those from the other and each pair of requirements is rated as *reinforcing*, *conflicting* or *neutral*. This PREview viewpoint integration is very similar to the proposed view integration. Finding intersecting requirements in PREview is analogous to identifying and analyzing commonality between feature models. Also, rating the intersection of requirements in PREview is analogous to creating/modifying mapping in view integration. There are clear differences, however. Most notably, PREview viewpoint integration does not explicitly consider software product line variability, while view integration does. View integration is more precise and systematic in terms of micro-level activities, as it involves analysis and modification of formally defined models, i.e. feature models and ontologies, while PREview involves analysis and modification of natural language descriptions. However, at the macro-level, as a methodology, PREview seems to be more systematic. For example, in PREview, a viewpoint can be categorized into *interactors* (interfacing components in a system), *stakeholders* (having socio-technical influence on system requirements), and *domain phenomena* (important domain concepts, like “braking” for trains and “saleorders” for e-commerce). Unfortunately, in feature modeling, the root-defining criteria are still not well-understood. As a result, view-defining criteria on an ontology is not clear either.

6.5 Early Aspects

Early aspects [5] 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. [32], 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 the mechanisms may differ, both our approach and this related work performs view extraction by imposing a viewpoint on a more general artifact.

6.6 Feature-Based Configuration of Models

Three notable works exist in this area: feature-based model templates [10], UML profile for software product lines [52], and automatic specialization of state charts [47]. 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 a 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 thesis, 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 thesis 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 feature-based ontology constraints, or more generally, *feature-based restriction* of models. UML profile for software product lines models variability in class diagrams and state diagrams mainly through stereotypes and inheritance. It is closer to the model templates work than the work described in this thesis, 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. However, the work only considers state charts, which are more in line with mapping to implementation. In addition, it does not use explicit feature models for configuration.

6.7 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 [46]. 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 [44]

using the reasoning framework. Determining semantics (and possibly syntax) of feature models from ontologies is also done in the semantic analysis step in view projection. But this idea is secondary to the essential notion of feature models as viewpoint-specific restrictions on ontologies.

Chapter 7

Conclusion

The main research objective of this thesis was to explore the intriguing relationship between ontology and feature modeling in the context of MDSPL. The exploration was done in three stages. Firstly, a comprehensive explanation of the fundamental notions behind feature models was given. Essentially, feature modeling is a technique for modeling concepts in increasing detail through a hierarchy with an explicit distinction between commonalities and variabilities. A family of feature models can be described according to several dimensions. Contrary to popular belief, feature models may embody several implicit forms, including mindmaps and textual outlines. The notational spectrum of feature models, which starts from basic feature models and extends to ontologies, and the suggestion that reference attributes are probably outside the scope of feature modeling provide a framework for discussing the boundary between feature models and ontologies.

Secondly, the fact that 1) feature models form a notational subset of ontologies, 2) feature models describe concepts more specialized than those described by ontologies, and 3) feature models and ontologies are both domain models suggests that feature models are views on ontologies, namely, projections of the ontologies from different viewpoints. Using an established methodology of ontology-oriented domain analysis, REA, this "view" relationship was precisely defined in two dimensions: syntactic correspondence and semantics of mapping. Syntactically, traceability links may be established between feature models and ontologies for increasing understanding of the mapping and for expressing simple constraints like existential dependencies. Semantically, a feature model provides a set of configurable constraints on an ontology. No-

tably elegant is the fact that the feature hierarchy allows both concise expression *and* flexible configuration of constraints.

Thirdly, based on the notion that feature models are views on ontologies, two complementary domain modeling approaches, view projection and view integration, were proposed. In view projection, feature models of different but similar-in-abstraction viewpoints are developed from an already-existing, mature ontology. Syntactic correspondence may be automatically achieved using ontology exploration facilities like QVT and OCL. Semantic analysis may also be done to improve both the semantics and syntax of the projected feature model. A classification of view projection based on different kinds of mapping semantics gives a further insight on the approach. The approach addresses the difficulty of modularizing ontologies and promotes agile feature modeling. In view integration, independently created feature models, which are typically outlines of requirements created by a group of experts, are integrated to form a coherent ontology. The commonality between the feature models is identified and analyzed and translated into creation or modification of mapping semantics. Change in mapping semantics may be additive, altering or subtractive, which typically expands, restructures, or contracts the ontology respectively. The approach addresses the difficulty of composing feature models and promotes agile ontology modeling.

As for future work, while possible differences between feature models and ontologies were proposed in terms of descriptive power and modeling philosophy, understanding exactly what they are and gaining community consensus on them remains a vital work item. Taking the two proposed domain modeling approaches and devising a comprehensive domain modeling paradigm for MDSPL is an important work item as well. But perhaps the most immediate future direction is to develop a tool supporting the notion of feature models as views on ontologies and the two proposed approaches. As the ideas were demonstrated using established technologies including UML, QVT and OCL, while the task may be challenging, it is certainly doable.

Bibliography

[1] Meta object facility 2.0 core. OMG Document ptc/03-10-04, Object Management Group, October 2003. Available from `http://www.omg.org/cgi-bin/doc?ptc/03-10-04`.

[2] MOF 2.0 query / views / transformations specification (final adopted). OMG Document ptc/05-11-01, Object Management Group, November 2005. Available from `http://www.omg.org/docs/ptc/05-11-01.pdf`.

[3] Amazon. Amazon.com: Online shopping for electronics, apparel, computers, books, dvds and more. At `http://www.amazon.com`.

[4] Henrik Reif Andersen. *Binary Decision Diagrams*. Department of Information Technology, Technical University of Denmark, Lyngby, Denmark, 1997. Lecture notes for 49285 Advanced Algorithms E97, `http://www.itu.dk/people/hra/notes-index.html`.

[5] Elisa Baniassad, Paul C. Clements, Joao Araujo, Ana Moreira, Awais Rashid, and Bedir Tekinerdogan. Discovering early aspects. *IEEE Software*, 23(1):61–70, February 2006.

[6] Don S. Batory. Feature models, grammars, and propositional formulas. In J. Henk Obbink and Klaus Pohl, editors, *Software Product Lines, 9th International Conference, SPLC 2005, Rennes, France, September 26-29, 2005, Proceedings*, volume 3714 of *Lecture Notes in Computer Science*, pages 7–20. Springer, 2005.

[7] Y. Bontemps, P. Heymans, P.Y. Schobbens, and J.C. Trigaux. Semantics of FODA feature diagrams. In *Third Software Product Line Conference (SPLC'04)*, Boston, USA, Aug 2004.

[8] Krzysztof Czarnecki. Overview of Generative Software Development. In *Proceedings of Unconventional Programming Paradigms (UPP) 2004, 15-17 September, Mont Saint-Michel, France, Revised Papers*, volume 3566 of *Lecture Notes in Computer Science*, pages 313–328. Springer-Verlag, 2004. http://www.swen.uwaterloo.ca/~kczarnec/gsdoverview.pdf.

[9] Krzysztof Czarnecki, Michal Antkiewciz, and Chang Hwan Peter Kim. Model-driven development of software product lines. Technical report, University of Waterloo, Waterloo, March 2006. Available from http://swen.uwaterloo.ca/~chpkim/mdspl06.pdf.

[10] Krzysztof Czarnecki and MichałAntkiewicz. Mapping features to models: A template approach based on superimposed variants. In Robert Glück and Michael Lowry, editors, *GPCE 2005 - Generative Programming and Component Enginering. 4th International Conference, Tallinn, Estonia, Sept. 29 – Oct. 1, 2005, Proceedings*, volume 3676 of *LNCS*, pages 422–437,. Springer, 2005.

[11] Krzysztof Czarnecki, Thomas Bednasch, Peter Unger, and Ulrich W. Eisenecker. Generative programming for embedded software: An industrial experience report. In Don Batory, Charles Consel, and Walid Taha, editors, *Proceedings of the ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering (GPCE'02), Pittsburgh, October 6-8, 2002*, volume 2487 of *Lecture Notes in Computer Science*, pages 156–172, Heidelberg, Germany, 2002. Springer-Verlag.

[12] Krzysztof Czarnecki and Simon Helsen. A characterization and categorization of model transformation approaches. *IBM Systems Journal*, 2006. To appear.

[13] Krzysztof Czarnecki, Simon Helsen, and Ulrich 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.

[14] Krzysztof Czarnecki and Chang Hwan Peter 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/Papers/Czarnecki.pdf.

[15] Krzysztof Czarnecki, Chang Hwan Peter Kim, and Karl Trygve Kalleberg. Feature models are views on ontologies. In Frank van der Linden and Robert L. Nord, editors, *Software Product Lines, 10th International Conference, SPLC 2006, Baltimore, USA, August 21-24, 2006*, Computer Society. IEEE, 2006.

[16] DARPA. Knowledge interchange format. Documents available at `http://logic.stanford.edu/kif/kif.html`.

[17] Matthias 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.

[18] Fincentric. Wealthview for banking. Available at `http://www.fincentric.com/products/banks/default.htm`.

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

[20] Martin Giese and Daniel Larsson. Simplifying transformations of ocl constraints. In *MoDELS*, pages 309–323, 2005.

[21] Jack Greenfield and Keith Short. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley, Indianapolis, IN, 2004.

[22] Catherine Griffin. Model transformation framework, 2000-2004. Tool available at `http://www.alphaworks.ibm.com/tech/mtf`.

[23] Generative Software Development Group. University of waterloo. At `http://gp.uwaterloo.ca`.

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

[25] Pavel Hruby and Jesper Kiehn. *Model-Driven Design of Software Applications with Business Patterns*. Springer-Verlag, 2006.

[26] IBM. WebSphere Commerce. Available at `www.ibm.com/websphere`.

[27] Kyo Kang, Sholom Cohen, James Hess, William Nowak, and Spencer Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, November 1990.

[28] Chang Hwan Peter Kim and Krzysztof Czarnecki. Synchronizing cardinality-based feature models and their specializations. In *Proceedings of ECMDA'05*, 2005. `swen.uwaterloo.ca/~kczarnec/ecmda05.pdf`.

[29] A. 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.

[30] Kwanwoo Lee and Kyo Chul Kang. Feature dependency analysis for product line component design. In *Proceedings of the 8th International Conference on Software Reuse (ICSR'04)*, volume 3107 of *LNCS*, pages 69–85, Madrid, Spain, Jul 2004. Springer.

[31] Karl Lieberherr, Boaz Patt-Shamir, and Doug Orleans. Traversals of object structures: Specification and efficient implementation. *ACM Trans. Program. Lang. Syst.*, 26(2):370–412, 2004.

[32] Neil Loughran, Américo Sampaio, and Awais Rashid. From requirements documents to feature models for aspect oriented product line implementation. In *MoDELS Satellite Events*, pages 262–271, 2005.

[33] Luis Mandel and María Victoria Cengarle. On the expressive power of OCL. In *FM'99 - Formal Methods. World Congress on Formal Methods in the Development of Computing Systems, Toulouse, France, September 1999. Proceedings, Volume I*, volume 1708 of *LNCS*, pages 854–874. Springer, 1999.

[34] Natalya Fridman Noy and Mark A. Musen. Specifying ontology views by traversal. In *International Semantic Web Conference*, pages 713–725, 2004.

[35] Object Management Group. *Model-Driven Architecture*, 2004. http://www.omg.org/mda.

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

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

[38] SAP. R/3. Available at http://www.sap.com/index.epx.

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

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

[41] Janos Sztipanovits and Gabor Karsai. Model-Integrated Computing. *IEEE Computer*, 30(4):110–111, 1997.

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

[43] W3C. Semantic web. Document available at http://www.w3.org/2001/sw/.

[44] 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.

[45] Wal-Mart. Walmart.com. At http://www.walmart.com.

[46] Hai Wang, Yuan Fang Li, Jing Sun, Hongyu Zhang, and Jeff Pan. A semantic web approach to feature modeling and verification. In *Workshop on Semantic Web Enabled Software Engineering (SWESE'05)*, Galway, Ireland, Nov 2005.

[47] Andrzej Wasowski. Automatic generation of program families by model restrictions. In Robert L. Nord, editor, *Software Product Lines: Third International Conference, SPLC 2004, Boston, MA, USA, August 30-September 2, 2004. Proceedings*, volume 3154 of

Lecture Notes in Computer Science, pages 73–89, Heidelberg, Germany, 2004. Springer-Verlag.

[48] Jens Weiland and Ernst Richter. Konfigurationsmanagement variantenreicher simulink-modelle. In *GI Jahrestagung (2)*, pages 176–180, 2005.

[49] Wikipedia. Graph isomorphism. At `http://en.wikipedia.org/wiki/Graph_isomorphism`.

[50] World Wide Web Consortium. *SPARQL Query Language for RDF*, 2006. `http://www.w3.org/TR/rdf-sparql-query/`.

[51] Wei Zhang, Hong Mei, and Haiyan 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.

[52] Tewfik Ziadi, Loïc Hélouët, and Jean-Marc Jézéquel. Towards a uml profile for software product lines. In *PFE*, pages 129–139, 2003.