RWTH Aachen

Department of Computer Science
Technical Report

Vergleich von Ansätzen zur Feature Modellierung bei der Softwareproduktlinienentwicklung

Horst Lichter, Thomas von der Maßen, Alexander Nyßen and Thomas Weiler

ISSN 0935–3232 · Aachener Informatik Berichte · AIB-2003-07

RWTH Aachen · Department of Computer Science · July 2003

titel

Horst Lichter, Thomas von der Maßen, Alexander Nyßen and Thomas Weiler

Lehr und Forschungsgebiet Informatik III
RWTH Aachen, Germany
Email: {lichter, vdmass, any, thweiler}@informatik.rwth-aachen.de

Zusammenfassung Die Feature Modellierung ist ein oft verwendeter Ansatz, um Variabilitäten und Gemeinsamkeiten der Produkte einer Produktlinie zu modellieren. Die einzelnen Ansätze, die die Feature-Modellierung nutzen, unterscheiden sich jedoch häufig in der Zielsetzung und Vorgehensweise bei der Modellierung. In diesem Artikel wird versucht, die verschiedenen Ansätze zur Feature-Modellierung einander gegenüberzustellen und ihre Gemeinsamkeiten und Unterschiede zu vergleichen und zu bewerten. Hierbei wird insbesondere betrachtet, welche Vorgehensweise gewählt werden muss und welche Eingaben vorliegen müssen, damit die Ergebnisse der Feature-Modellierung in den nachgelagerten Entwicklungsschritten als Eingabe weiterverwendet werden können.

1 Einleitung

Die Feature Modellierung ist ein oft verwendeter Ansatz, um Variabilitäten und Gemeinsamkeiten der Produkte einer Produktlinie zu modellieren. Die einzelnen Ansätze, die die Feature-Modellierung nutzen, unterscheiden sich jedoch häufig in der Zielsetzung und Vorgehensweise bei der Modellierung. Oft bleiben die Ziele und Methodiken der Feature-Modellierung jedoch auch weitgehend unklar.

So fehlt den Ansätzen häufig eine klar definierte Vorgehensweise zur Modellgewinnung. Die notwendigen Voraussetzungen und Eingaben für die Modellerstellung unterscheiden sich bei den verschiedenen Ansätzen oder fehlen teilweise. Auch fällt die Identifikation und Abgrenzung von Features bei den verschiedenen Ansätzen unterschiedlich aus, oft ohne dass die dazu verwendete Vorgehensweise explizit beschrieben wird.

Schließlich unterscheiden sich die Ansätze in der Verwendung der Produkte der Feature-Modellierung. Während einige Ansätze die Ergebnisse der Feature-Modellierung als Kommunikationsgrundlage der Entwickler untereinander ansehen, dienen sie anderen als Kommunikationsmedium zwischen den Kunden und den Entwicklern. Einige Ansätze versuchen wiederum die Ergebnisse der Feature-Modellierung als Grundlage oder zusätzliche Eingabe für die nachgelagerten Entwicklungsschritte, wie zum Beispiel die Architekturmodellierung, zu nutzen.

In diesem Bericht wird versucht, die verschiedenen Ansätze zur Feature-Modellierung einander gegenüberzustellen und ihre Gemeinsamkeiten und Unterschiede zu vergleichen und zu bewerten. Hierbei wird insbesondere betrachtet, welche Vorgehensweise gewählt werden muss und welche Eingaben vorliegen müssen, damit die Ergebnisse der Feature-Modellierung in den nachgelagerten Entwicklungsschritten als Eingabe weiterverwendet werden können.

Dazu wird untersucht, welche Ziele die einzelnen Ansätze zur Feature-Modellierung verfolgen und inwieweit sie die selbstgesteckten Ziele erfüllen. Daneben werden die in den verschiedenen Ansätzen verwendeten Modellierungselemente identifiziert und miteinander verglichen.

Dabei wird insbesondere betrachtet, inwieweit in den betrachteten Ansätzen domänenspezifische Modellierungselemente zum Einsatz kommen. Im weiteren wird dann die Vorgehensweise zur Modellerstellung bei den unterschiedlichen Ansätzen untersucht und anhand eines Beispiels die Verwendung der zuvor beschriebenen Modellierungselemente zu den einzelnen Ansätzen demonstriert.

Abschließend wird jeder Ansatz bewertet. Dem Vergleich der unterschiedlichen Verwendung des Feature-Begriffs sowie den Zielsetzungen und verwendeten Modellierungselemente der einzelne Ansätzen ist dabei ein eigenes Kapitel gewidmet.

2 FORM - A Feature Oriented Reuse Method

2.1 Überblick

Die Methode FORM hat zum Ziel, ein systematisches Vorgehen für die Entwicklung von Produktlinien zu definieren. Sie wird in [LKL02] und in [KKL+98] detailliert beschrieben. Abbildung 1 zeigt einen Überblick über den FORM-Entwicklungsprozess, der die für die Produktlinienentwicklung typischen Bereiche Domain und Application Engineering unterscheidet.

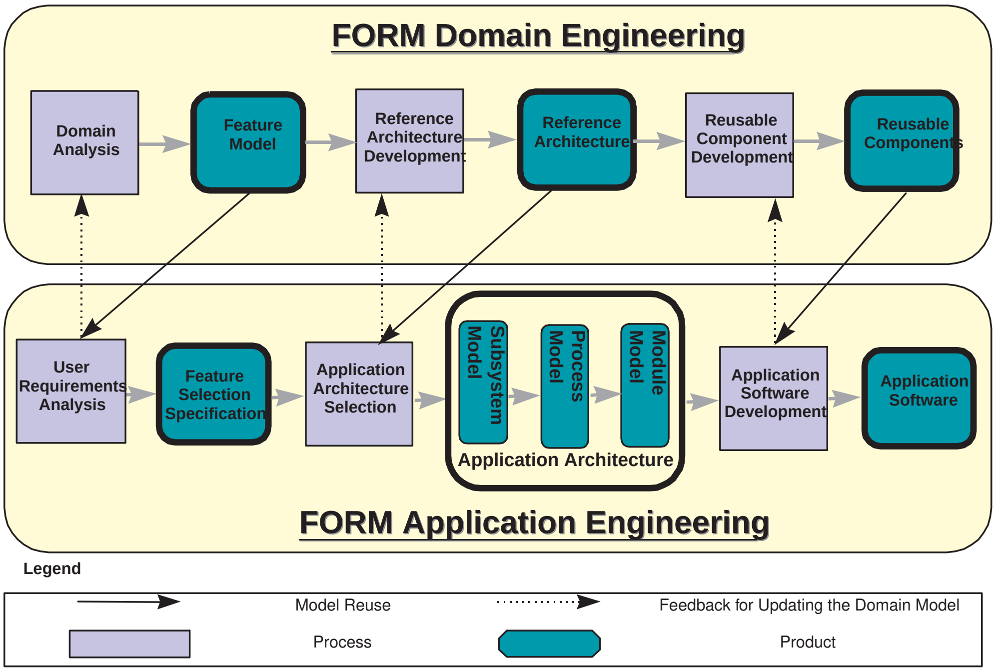


Abbildung 1. Der FORM Entwicklungsprozess (aus [KKL+98])

Die Feature-Modellierung ist ein zentraler Teil im Domain Engineering. Die erstellten Feature-Modelle werden im Application Engineering verwendet, um Produkte zu konfigurieren und abzuleiten. Die Domänenanalyse, deren Ergebnis unter anderem das Feature-Modell der Domäne ist, wird in der FORM Methode in die Teilprozesse Planung, Feature Analyse und Validierung unterteilt.

Im Kontext dieser Studie konzentrieren wir uns speziell auf den Prozess der Feature-Analyse und auf die Modellierungselemente, die FORM zur Verfügung stellt, um Feature-Modelle zu erstellen.

2.2 Ziele der Feature-Modellierung

Mit der Feature-Modellierung sollen bei der FORM Methode die folgenden Ziele adressiert werden:

– Modellierung der Funktionalitäten der Domäne. Dabei wird der Fokus auf die Modellierung der extern sichtbaren Gemeinsamkeiten und Variabilitäten gelegt.
– Modellierung der Implementierungsalternativen der identifizierten Funktionalitäten sowie der Einsatzumgebungen der resultierenden Produkte.
– Konstruktion eines Feature-Modells, das sich ohne große Brüche in eine geeignete Plattformarchitektur überführen lässt.
– Dadurch, dass Features mit Begriffen versehen werden, entsteht ein Begriffslexikon, das die Kommunikation zwischen den an der Entwicklung beteiligten Personen erleichtert.
– Da das Feature-Modell eine Domäne charakterisiert, können Feature-Modelle benutzt werden, um Domänen zu vergleichen.
– Die in einem Feature-Modell enthaltenen Gemeinsamkeiten einer Produktlinie geben einen Hinweis auf den Grad der Wiederverwendung, der erreicht werden kann.
– Das Feature-Modell dient bei der Produktentwicklung als wesentliche Grundlage, damit notwendige Entscheidungen fundiert getroffen werden können.

2.3 Modellierungselemente

Metamodell FORM definiert die Feature-Modellierungselemente nicht präzise, stützt sich jedoch auf die Modellierungselemente von FODA. Abbildung 2 zeigt das aus der FORM-Beschreibung extrahierte Metamodell.

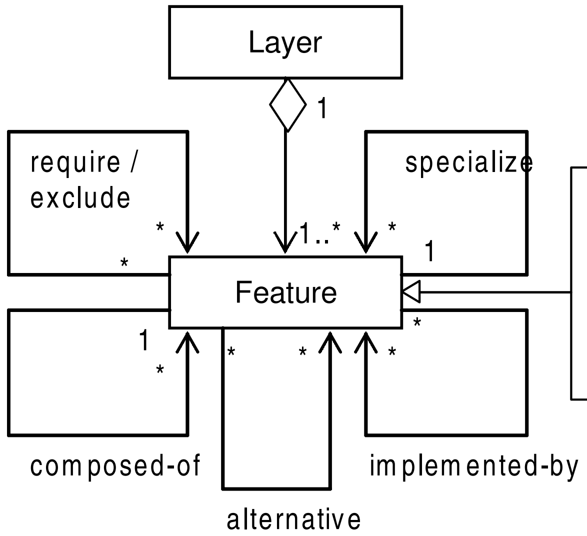
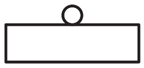


Abbildung 2. Metamodell der Feature-Modellierung der FORM-Methode

Beschreibung der Elemente

Element	Beschreibung	Symbol
Feature (F)	Dient als abstraktes Modellierungselement dazu, die Gemeinsamkeiten der beiden Arten Mantadory und Optional Feature zu beschreiben	(kein Symbol)
Mandatory Feature	Mindestens ein solches Feature muss in jedem Produkt enthalten sein.	
Optional Feature	Ein optionales Feature kann, muss aber nicht in einem Produkt enthalten sein.	
Layer	Gruppiert eine beliebige Menge von Features. Ein Feature muss in genau einem Layer enthalten sein.	
composed-of : F x F	Diese Beziehung dient dazu, Features im Sinn eine Teile-Ganzes Semantik detaillierter zu beschreiben.	
specialize : F x F	Die Beziehung erlaubt, Features im Sinn einer Spezialisierungshierarchie zu strukturieren. Für ein Feature kann es beliebig viele Spezialisierungen geben. Ein Feature ist die Spezialisierung von höchstens einem Feature. Diese Beziehung spannt einen Spezialisierungsbaum auf.	
implemented-by : F x F	Mit dieser Beziehung wird modelliert, dass ein Feature durch ein (technisches) Feature realisiert werden kann. Ein Feature kann mehrere Features implementieren und von beliebig vielen Features implementiert werden	
require : F x F	Die Existenz-Abhängigkeit zwischen Features wird durch diese Beziehung modelliert.	In Form von Kompositionsregeln
exclude : F x F	Diese Beziehung modelliert, dass zwei Features inkompatibel sind und dementsprechend nicht gemeinsam in einem Produkt enthalten sein dürfen.	In Form von Kompositionsregeln
alternative : F x F	Features können alternativ zueinander sein.	

Domänenspezifische Modellierungselemente Die gezeigten Beispiele stammen aus dem Bereich der Telefonie. Die Modellierungselemente sind jedoch allgemein gehalten. Die Autoren der FORM-Methode schlagen Standard-Schichten (Layer) für ein Feature-Modell vor. Diese Schichten sind möglicherweise nicht generisch, sondern können grob dem Bereich der eingebetteten Systeme zugeordnet werden. Bei den vorgeschlagenen Schichten handelt es sich im einzelnen um:

- *Capability Layer*
  Diese Schicht enthält alle Features, die aus der Sicht des Benutzers als unterschiedliche Dienste identifiziert werden. Demenstprechend wird in dieser Schicht ein funktionales Domänenmodell erstellt.
- *Domain Technology Layer*
  In dieser Schicht sind alle die Features enthalten, die die Dienste des Capability Layers implementieren.
- *Operating Environment Layer*
  Diese Schicht modelliert alle Umgebungen (z.B. technische Plattformen, Betriebssysteme), in denen die modellierten Anwendungen genutzt werden sollen.
- *Implementation Technique Layer*
  In dieser Schicht sind generische Techniken angesiedelt, die zur Realisierung der Dienste und Funktionen benötigt werden.

2.4 Vorgehensweise

Der Prozess der Feature-Modellierung ist in Abbildung 3 wiedergegeben. Daraus ist zu erkennen, dass sich die Feature Modellierung auf die Begriffe der Domäne und auf die Dokumente der Domäne abstützt.

Die Methode FORM definiert eine Reihe von Richtlinien, die den Feature-Modellierungsprozess leiten sollen. Insbesondere wird der Bereich der Identifikation von Features betrachtet. Folgende Richtlinien sind diesem Bereich zugeordnet:

- *Analysiere die Begriffe, die in der betrachteten Domäne benutzt werden.*
  Diese Richtlinie stützt sich auf die Erkenntnis, dass viele zentrale Konzepte einer gut verstandenen Domäne in Begriffen codiert sind. Die bei dieser Analyse gefundenen Begriffe können entlang der vorgeschlagenen Standard-Schichten klassifiziert werden. Die Methode FORM sieht vor, dass die vier Standard-Schichten durch sogenannte Feature-Kategorien feiner unterteilt werden. Die Schichten bilden zusammen mit den Feature-Kategorien ein Rahmenwerk, das bei der Identifikation und Klassifikation der Features genutzt werden kann.
- *Identifiziere zuerst die Unterschiede zwischen den einzelnen Produkten und extrahiere auf dieser Basis deren Gemeinsamkeiten.*
  Da die Gemeinsamkeiten typischerweise größer sind als die Unterschiede zwischen den einzelnen Produkten einer Produktlinie, macht es Sinn, zuerst diesen Aspekt zu betrachten und zu modellieren. Dieser Prozessschritt wird in die folgenden weiteren Teilschritte eingeteilt: Identifiziere Produktkategorien derart, dass die Produkte einer Kategorie sich nur in sehr wenigen Aspekten unterscheiden. Anschließend liste die Features auf, die jede Kategorie charakterisieren. Dadurch werden die Unterschiede zu den anderen identifizierten Kategorien sichtbar. Abschließend betrachte man die Produkte, die gleiche Funktionalitäten aufweisen.

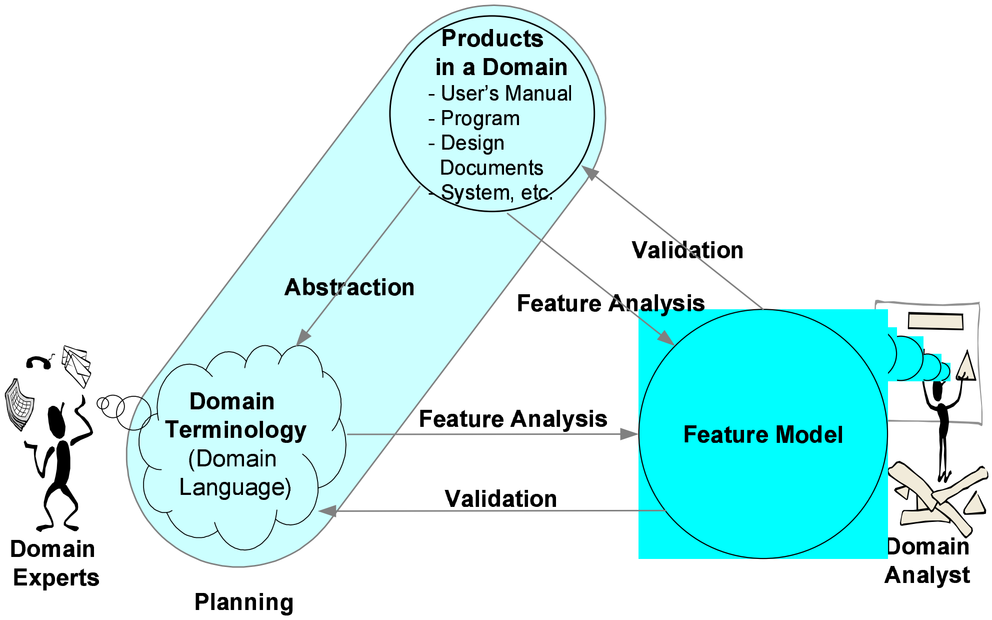


Abbildung 3. Der Feature-Modellierungsprozess der FORM-Methode (aus [KKL+98])

Diese Vorgehensweise soll dazu führen, dass die Gemeinsamkeiten schneller identifiziert werden.

– *Identifiziere nur diejenigen Implementierungstechniken, deren Einsatz auch zu unterschiedlichen Produkten der Produktlinie führt.*
Da die Menge der potentiell einsetzbaren Implementierungstechniken sehr groß sein kann, würde die umfassende Modellierung dieser Techniken zu überladenen und unübersichtlichen Modellen führen. Deshalb sollen nur die Implementierungstechniken modelliert werden, die auch zu unterschiedlichen Produkten führen.

Es werden weitere Richtlinien angegeben, die die eigentliche Modellierung, in diesem Fall also den Einsatz der Modellierungselemente, leiten sollen.

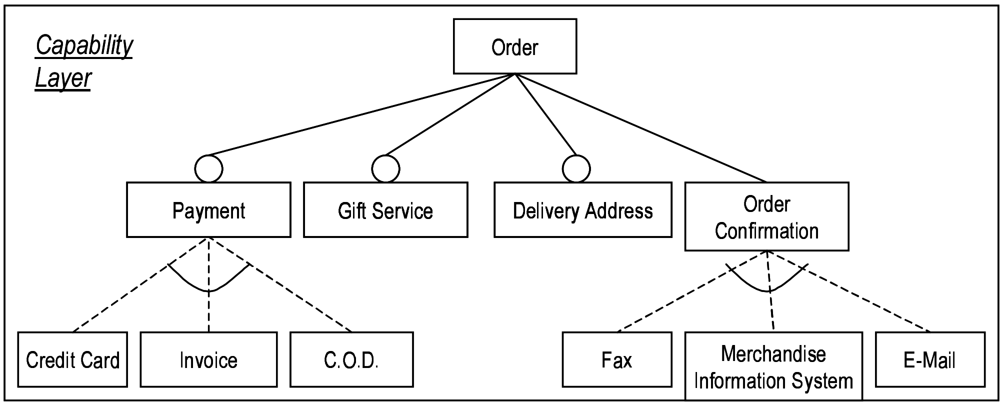
– *Ein Feature-Diagramm soll nicht die funktionalen Abhängigkeiten im Sinn einer Aufrufhierarchie modellieren.*
Die Feature-Modellierung ist nicht der geeignete Ansatz, um eine funktionale Dekomposition auszudrücken. Ziel der Modellierung muss es sein, die Gemeinsamkeiten und Variabilitäten zu modellieren.
– *Implementierungsbeziehungen sollen immer zum abstraktesten Feature einer Hierarchie gehen.*
Wenn gleiche Implementierungsbeziehungen zu allen Sub-Features einer Hierarchie eingerichtet werden, dann wird das resultierende Diagramm überladen. Dies kann dadurch vermieden werden, dass nur eine Implementierungsbeziehung zum Wurzel-Feature der Hierarchie eingerichtet wird.
– *Ein Feature soll nur bis auf die Detaillierungstufe verfeinert werden, die noch Variabilitäten zwischen den Produkten zeigt.*

Ein Feature-Modell soll nicht alle Details aufzeigen. Da das übergeordnete Ziel der Feature-Modellierung darin besteht, Gemeinsamkeiten und insbesondere die Variabilitäten zu modellieren, kann die Verfeinerung eines Features dann beendet werden, wenn keine Variabilitäten zwischen den Produkten mehr auftreten.

– *Verfeinere Features so, dass sich diese gut auf Elemente der Architektur abbilden lassen.*
  Diese Richtlinie führt dazu, dass die Nachverfolgbarkeit beginnend bei einem Feature und endend bei einer implementierten Komponente unterstützt wird. Ebenfalls wird dadurch die Wiederverwendbarkeit und die Kombinierbarkeit der Komponenten bei der Produktentwicklung verbessert.

2.5 Beispiel

Abbildung 4 zeigt beispielhaft ein Feature-Modell in der Notation der FORM-Methode. Es wird ersichtlich, dass die graphische Notation keinen Unterschied zwischen einer OR-Alternative (linke Alternative des Modells) und einer XOR-Alternative (rechte Alternative) macht. Weiterhin erkennt man die Trennung in die strukturelle graphische Sicht und in die Beschreibung der Abhängigkeiten zwischen Features.



Composition Rules
- Gift Service ***requires*** Delivery Address

Abbildung 4. Beispiel eines FORM Feature-Modells

FORM sieht zusätzlich zur graphischen Darstellung eine textuelle Darstellung der Features vor. Features werden hier nach folgendem Schema strukturiert beschrieben:

`FEATURE` – Bezeichner des Features
`DESCRIPTION` – Erklärende Beschreibung
`ISSUE AND DECISION` – Entscheidunghilfen bei Alternativen und Optionen
`TYPE` – Zugehörigkeit zu einem der definierten Layer
`COMMONALITY` – (ALTERNATIVE , MANDATORY , OPTIONAL)
`COMPOSED OF` – Liste der Sub-Features

`COMPOSITION RULE` – Aufzählung der require und exclude Beziehungen
`ALLOCATED TO SUBSYSTEM` – Trace-Information zur Architektur
`END FEATURE`

Die textuelle Darstellung ist äquivalent zur graphischen Darstellung. Feature-Modelle werden in der textuellen Notation erfasst; die graphische Notation dient der Präsentation und als Diskussionsmedium.

2.6 Zielerfüllung

In diesem Abschnitt soll untersucht werden, in wie weit der FORM-Ansatz die selbstgesteckten Ziele erfüllt. Der FORM-Ansatz gewährleistet durch die zusätzlichen textuellen Beschreibungen der Features die Modellierung der nach außen sichtbaren Gemeinsamkeiten und Variabilitäten. Jedoch kann nicht zwischen einer OR und einer XOR Auswahl unterschieden werden. Desweiteren ist es nur möglich, Variabilität in der Struktur auszudrücken, im Sinne einer Ist-Enthalten Beziehung, jedoch nicht in den dynamischen Beziehungen der Features oder ihrer Attribute.

Durch die *implements* Beziehung ist es möglich, wie in den Zielen gefordert, Implementierungsalternativen zu modellieren. Jedoch wird in dem Ansatz nicht näher erläutert, wie der bruchfreie Übergang von der Feature-Modellierung zur Produktarchitektur gestaltet werden soll. Auch bleibt in diesem Zusammenhang unklar, wie das Feature-Modell als Entscheidungsgrundlage dienen soll.

Wie in den Zielen gefordert, unterstützt die Feature-Modellierung nach dem FORM-Ansatz die Erstellung eines Begriffslexikons und erlaubt den Vergleich verschiedener Domänen anhand der zugehörigen Featuremodelle.

2.7 Bewertung

Die nachfolgenden Aussagen beziehen sich lediglich auf den Teil der Feature-Modellierung der FORM-Methode.

- Der in diesem Ansatz verwendete Feature-Begriff ist sehr weit gefasst. So werden Techniken und Implementierungsalternativen ebenfalls als Features modelliert.
- Die vorgeschlagenen Schichten führen bei der Modellierung dazu, dass Domänen- und Lösungsaspekte zusammen modelliert werden.
- Mit der vorgeschlagenen Notation können keine speziellen Selektionen modelliert werden. Es ist lediglich vorgesehen, dass Features als Alternativen gekennzeichnet werden können. Demzufolge bleibt offen, ob es sich dabei um eine OR oder XOR Beziehung handelt.
- Dadurch, dass die Abhängigkeitsbeziehungen zwischen Features (also require und exclude) nicht im gleichen Modell beschrieben werden, besteht die Gefahr, dass das gesamte Modell bei Änderungen inkonsistent wird.
- Da die Notation der Feature-Modelle von FODA übernommen wurde, werden die Beziehungen, die zur Modellierung zur Verfügung stehen, ungerichtet dargestellt. Dies kann zu Fehlinterpretationen der Modelle führen, wenn diese nicht ausschließlich von oben nach unten gezeichnet werden.
- Obwohl FORM vorsieht, dass Features in Layer gruppiert werden, gibt es keine explizite Möglichkeit, Beziehungen zwischen Layer zu modellieren. Layer sind demnach nichts anderes als Behälter.

3 Nokia - Feature Modellierung

3.1 Überblick

Fey et al [FFB02] stellen einen Ansatz zur Feature-Modellierung vor, der bei Nokia entwickelt wurde und dort im Bereich User Interface Software von Mobiltelefonen eingesetzt wird. Der vorgestellte Ansatz basiert auf FODA, soll jedoch die identifizierten Schwachstellen der FODA-Modellierung überwinden. Dabei steht im Vordergrund, dass die erstellten Feature-Modelle zumindest teilweise automatisch analysiert und weiterverwendet werden sollen. Damit dies möglich wird, muss die Semantik der Modellierungselemente präzise definiert werden. Der vorgestellte Ansatz geht über die FODA Modellierung hinaus und hat folgende charakteristische Merkmale:

– Explizite Modellierung von *Attributen*. Insbesondere sollen Beziehungen zwischen Attributen und Features modelliert werden können. Dadurch können die Abhängigkeiten zwischen Attributen und Features ausgedrückt werden.
– Modellierung von *Auswahlregeln*, die die Ableitung von Produkten entsprechend den modellierten Variabilitäten und Abhängigkeiten unterstützen.
– Features werden nicht mehr *redundant* modelliert. Die redundante Modellierung von Features ist eine Konsequenz der Baumstruktur der FODA Modellierung. Dieser Ansatz erlaubt explizit, dass Feature-Modelle eine Graphstruktur haben können.
– Die in FODA eingeführte Unterscheidung von Mandatory und Optional Features wird aufgegeben. Die dahinter stehende Semantik wird durch explizite *require* bzw. *conflict* Beziehungen zwischen Features modelliert.

3.2 Ziele der Feature-Modellierung

Dieser Ansatz sieht die erstellten Feature-Modelle als Teil des gesamten Requirements-Modells. Mithilfe von Features sollen die funktionalen und die Qualitätsanforderungen modelliert werden, wobei die Gemeinsamkeiten und die Variabilitäten einer Produktlinie den Schwerpunkt der Modellierung ausmachen. Der beschriebene Ansatz konzentriert sich besonders auf die folgenden Ziele:

– Modellierung der Produktfunktionalität.
– Konstruktion von semantisch präzisen Feature-Modellen, die automatisch weiterverarbeitet werden können. Dazu müssen alle Abhängigkeiten zwischen Features explizit modelliert werden.
– Automatische Analyse der Feature-Modelle. Dabei soll insbesondere die Ableitung von Produkten unterstützt werden. Ziel ist es, dass die für ein Produkt ausgewählten Features konsistent sind und aus den Modellen berechnet werden können.

3.3 Modellierungselemente

Metamodell Das in Abbildung 5 gezeigte Metamodell ist der Orginalarbeit entnommen. Die Elemente des Metamodells werden, im Vergleich zu anderen Ansätzen zur Feature-Modellierung, präzise eingeführt. Insbesondere werden die Beziehungen exakt definiert.

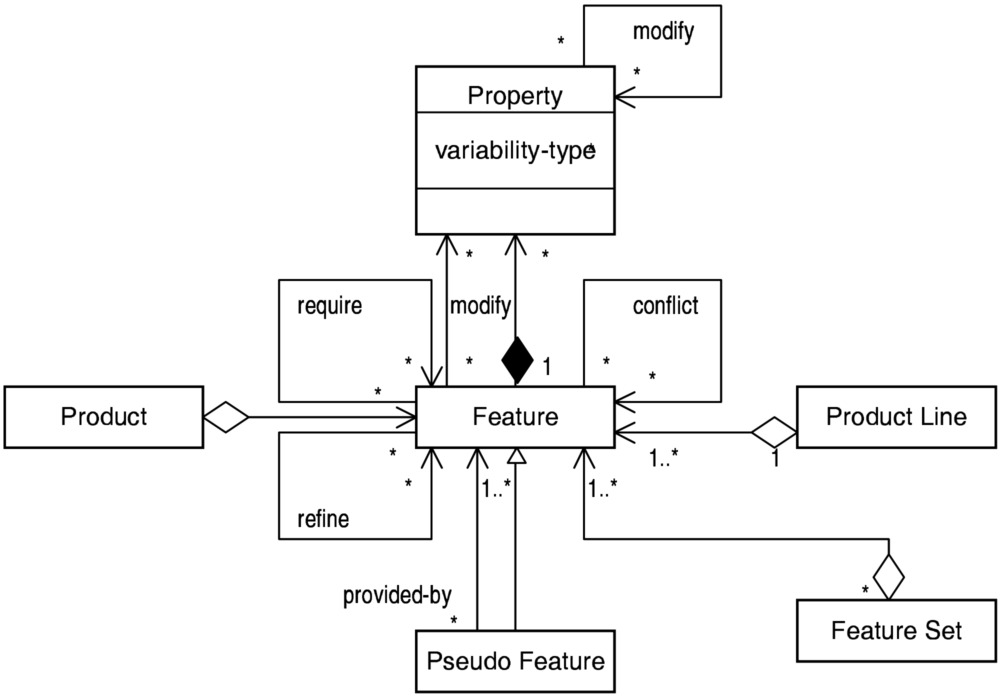
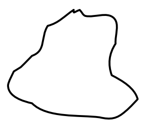


Abbildung 5. Nokia Metamodell der Feature-Modellierung (aus [FFB02])

Beschreibung der Elemente

Element	Beschreibung	Symbol
Feature (F)	Entspricht der FODA Feature Definition	●
Pseudo Feature (PF)	Pseudo-Features modellieren solche Features, die nicht direkt in einem Produkt enthalten sein können, da sie abstrakt sind. Zu jedem Pseudo-Feature muss wenigstens ein Feature angegeben werden, dass dieses konkretisiert. Diese Beziehung wird durch die provided-by Beziehung im Feature-Diagramm modelliert.	○

Element	Beschreibung	Symbol
Element	**Beschreibung**	**Symbol**
Property (P)	Features können durch Eigenschaften näher spezifiziert werden. Diese Eigenschaften werden durch das Element Property modelliert.	▭
Fixed Property	Modelliert eine Property, die in jedem Produkt den gleichen Wert besitzt. Dies entspricht der Semantik einer Konstanten.	Wert des Property-Attributes variability-type
Variable Property	Modelliert eine Property, deren Wert in einem Produkt mehrere Werte annehmen kann. Beispiel: die Länge von Eingabefeldern kann in diesem Sinn variieren.	Wert des Property-Attributes variability-type
Family-Variable Property	Modelliert eine Property, deren Wert bei jedem Produkt der Produktlinie einen speziellen (aber festen) Wert annehmen kann.	Wert des Property-Attributes variability-type
Feature Set	Gruppiert eine beliebige Menge von Features nach einem frei wählbaren Gesichtspunkt. Features können in beliebig vielen Feature Sets enthalten sein.	
Product	Besteht aus einer konsistenten Menge von Features.	keine
Product Line	Enthält alle Features.	keine
refine : F x F	Diese Beziehung dient dazu, Features detaillierter zu beschreiben. Sie entspricht der consists-of Beziehung aus FODA. Die refine-Beziehung darf nicht zyklisch sein. Es ist möglich, dass ein Feature die Verfeinerung von mehreren Features ist. Dadurch wird die redundante Modellierung von Features vermieden.	⟶
require : F x F	Die Existenz-Abhängigkeit zwischen Features wird durch diese Beziehung modelliert. Diese Beziehung ist transitiv.	
conflict : F x F	Diese Beziehung modelliert, dass zwei Features inkompatibel sind und dementsprechend nicht gemeinsam in einem Produkt enthalten sein dürfen. Diese Bezeigung ist symmetrisch.	

Element	Beschreibung	Symbol
provided-by : PF x F	Pseudo-Features müssen durch Features konkretisiert werden; dieses Feature realisiert das Pseudo-Feature. Ein Feature kann dabei mehrere Pseudo-Features realisieren. Dadurch wird die redundante Modellierung von Features vermieden. Diese Beziehung ist transitiv.	
modify : F x P	Die Existenz eines Features in einem Produkt kann Auswirkungen auf den Wert einer Property eines anderen Features haben. Dieser Sachverhalt wird durch die modify Beziehung zwischen einem Feature und einer Property modelliert.	
modify : P x P	Diese Beziehung modelliert, dass der Wert einer Property eines Features vom Wert einer Property eines anderen Features abhängt.	
isPropertyOf: F x P	Definiert eine Property als Eigenschaft eines Features.	

Domänenspezifische Modellierungselemente Als Beispiel werden Modelle der Domäne User Interface Software von Mobiltelefonen der Firma Nokia gezeigt. Die eingeführten Modellierungselemente sind jedoch nicht domänenspezifisch. Hervorzuheben ist jedoch der Ansatz, dass das resultierende Feature-Modelle in verschiedenen Sichten präsentiert werden kann. Es werden drei Sichten (Views) vorgeschlagen:

– *Hierarchy-View*
  Diese Sicht enthält alle Features und ihre strukturellen Abhängigkeiten, die durch die refine und durch die provided-by Beziehungen aufgespannt werden.
– *Dependency-View*
  Sie zeigt alle semantischen Abhängigkeiten zwischen den Features. In dieser Sicht werden alle require-, conflict- und modify-Beziehungen gezeigt. Die strukturellen Beziehungen werden ausgeblendet.
– *Set-View*
  Diese Sicht zeigt alle definierten Gruppierungen von Features. Alle Beziehungen zwischen den Features sind dabei ausgeblendet.

3.4 Vorgehensweise

Leider machen die Autoren keine Aussage, wie die Feature-Modellierung durchzuführen ist. Statt dessen werden Algorithmen angegeben, um konsistente Mengen von Features zu ermitteln, die in einem Produkt enthalten sein müssen.

3.5 Beispiel

Das Beispiel in Abbildung 6 zeigt, wie Mandatory Features und die Selektionen explizit mit require und conflict Beziehungen modelliert werden können.

- Das Feature Order Confirmation wird durch eine require Beziehung an sein Vater-Feature Order gebunden. Dementsprechend muss es in jedem Produkt enthalten sein.
- Aus den drei durch provided-by Beziehungen als Realisierungsalternativen zum Feature Order Confirmation modellierten Features Fax, Merchandise Information System und E-Mail soll genau ein Feature ausgewählt werden müssen. Um dieses zu modellieren, werden wechselseitige confict Beziehungen zwischen diesen Features eingrichtet.

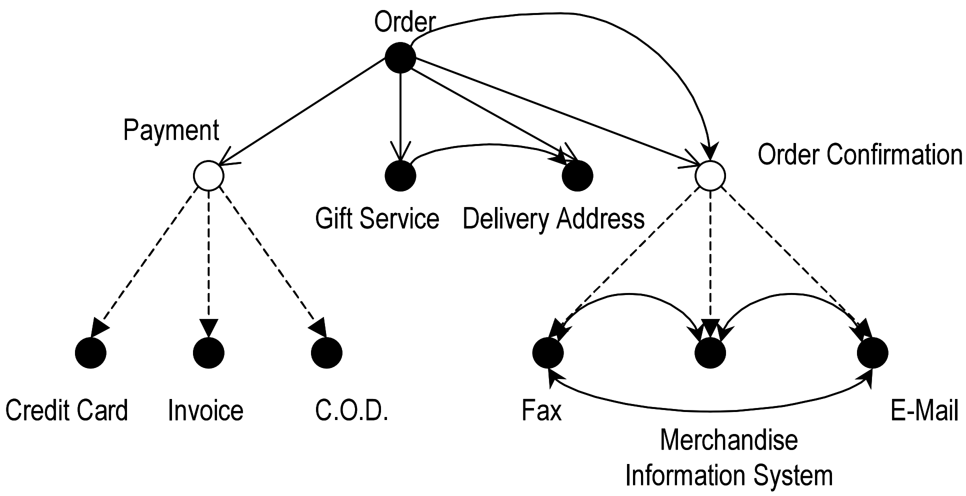


Abbildung 6. Beispiel eines Nokia Feature-Modells

3.6 Zielerfüllung

In diesem Abschnitt soll untersucht werden, in wie weit der Ansatz von Fey et. al. die selbstgesteckten Ziele erfüllt. Das Ziel, die Produktfunktionalität mit Hilfe von Features modellieren zu können, kann nur eingeschränkt erreicht werden, wenn es keine zustätzlichen, beschreibenden Dokumente gibt. Die in den Feature-Modellen enthaltene Information reicht dazu nicht aus.

Durch die präzise Formulierung der Modellierungselemente erlaubt der Ansatz nach Fey et.al. die Konstruktion semantisch präziser Feature-Modelle, die automatisch analysiert und weiterverarbeitet werden können.

3.7 Bewertung

Der betrachtete Modellierungsansatz ist dadurch, dass die automatische Analyse und Weiterverarbeitung der Feature-Modelle das zentrale Ziel ist, präzise formuliert. Weitere positive Aspekte sind:

- Attribute werden explizit modelliert. Beziehungen zwischen Attributen und Features sind möglich.
- Pseudo-Features sind geeignet, um explizit Abstraktionen zu modellieren.
- Die vorgeschlagenen Sichten helfen, komplexe Feature-Modelle überschaubar zu präsentieren.

Demgegenüber können folgende Schwächen aufgeführt werden:

- Es werden keine Muster angegeben, um typische Selektionen (1 aus n, n aus m, etc) mit Hilfe der require und conflict Beziehungen zu modellieren.
- Optionale und Mandatory Features müssen explizit durch (viele) require und conflict Beziehungen modelliert werden. Dies führt zu sehr unüberschaubaren Diagrammen. Die damit modellierten Informationen sind jedoch notwendig, um die resultierenden Modelle automatisch analysieren zu können.
- Die Mischung aus refine und provided-by Beziehungen führt dazu, dass Features, die ein Pseudo-Feature realisieren, die Verfeinerungshierarchien des Pseudo-Features erben. Das macht die Modelle komplex und für den Menschen schwer nachvollziehbar.
- Da zu jedem Pseudo-Feature wenigstens ein realisierendes Feature angegeben werden muss (mit Hilfe der provided-by Beziehung), ist man bei der Modellierung in manchen Situationen gezwungen ein an sich abstraktes Features, das durch ein Pseudo-Feature dargestellt werden sollte, durch ein normales Feature im Diagramm abzubilden.

4 FeatuRSEB

4.1 Überblick

Griss et. al. [G$^+$98] beschreiben, wie die Methode *Reuse-Driven Software Engineering Business (RSEB)* [JGJ97] um die Modellierung von Features auf der Basis des FODA-Ansatzes erweitert wird. Dabei entsteht das Feature-Modell auf der Grundlage des *Domänen Use Case Modells*, das zentral für den RSEB-Ansatz ist. Das Feature-Modell soll nicht dazu dienen, die Funktionalität einer Produktlinie im Detail zu beschreiben. Dazu werden die Use Case Modelle verwendet. Das Feature-Modell dient im Sinne eines Konfigurationsmodells dazu, die Anwendungsentwickler bei der Ableitung von Produkten aus der gemeinsamen Plattform zu unterstützen.

4.2 Ziele der Feature-Modellierung

Mit der Feature-Modellierung sollen die folgenden Ziele erreicht werden:

- Das Feature-Modell ergänzt die entwickelten Use Case Modelle und dabei insbesondere das Domain Use Case Modell.
- Das Feature-Modell dient im wesentlichen dazu, die Konfiguration von Produkten zu unterstützen. Da es die Gemeinsamkeiten und die Variablitäten der Domäne explizit zeigt, soll es bei produktspezifischen Entscheidungen herangezogen werden, wenn es darum geht, aus den zur Verfügung stehenden Alternativen auszuwählen bzw. Features zu kombinieren.
- Das Feature-Modell soll nicht-funktionale Features beschreiben, die nicht oder nur schwer in Use Case Modellen ausgedrückt werden können.
- Das Feature-Modell soll explizit nicht vollständig sein, sondern nur die markanten Eigenschaften der Dömane zeigen. Es soll die Essenz der Domäne modelliert werden. Die vollständige Sicht auf die Domäne liefert bei FeatuRSEB das Domänen Use Case Modell. Das Feature-Modell ergänzt dieses.
- Der Entwickler soll im Analyseprozess methodisch unterstützt werden. Dazu wird die RSEB-Methode um Tätigkeiten der Feature-Modellierung erweitert, die mit der Entwicklung des Domänen Use Case Modells verzahnt sind. Weiterhin werden Feature-Arten vorgeschlagen, die die identifizierten Features in die Kategorien Funktionale Features, Architekturelle Features und Implementierungs-Features einteilen.

4.3 Modellierungselemente

Metamodell Abbildung 7 zeigt das Metamodell, das aufgrund der Beschreibung der Feature-Modellierung in [G$^+$98] entstanden ist. Auffällig bei diesem Metamodell sind die folgenden Aspekte:

- Variationspunkte werden explizit modelliert. Dazu wird das Modellierungselement Variation Point Feature zur Verfügung gestellt.
- Nicht alle Features können hierarchisch verfeinert werden. Dies geht nur bei Mandatory und bei Variation Point Features.
- Das Metamodell sieht unterschiedliche Zeitpunkte für die Auswahl der Alternativen vor. Es wird zwischen Alternativen unterschieden, die zur Konfigurationszeit (reuse time) und zur Laufzeit (use-time) ausgewählt werden können.

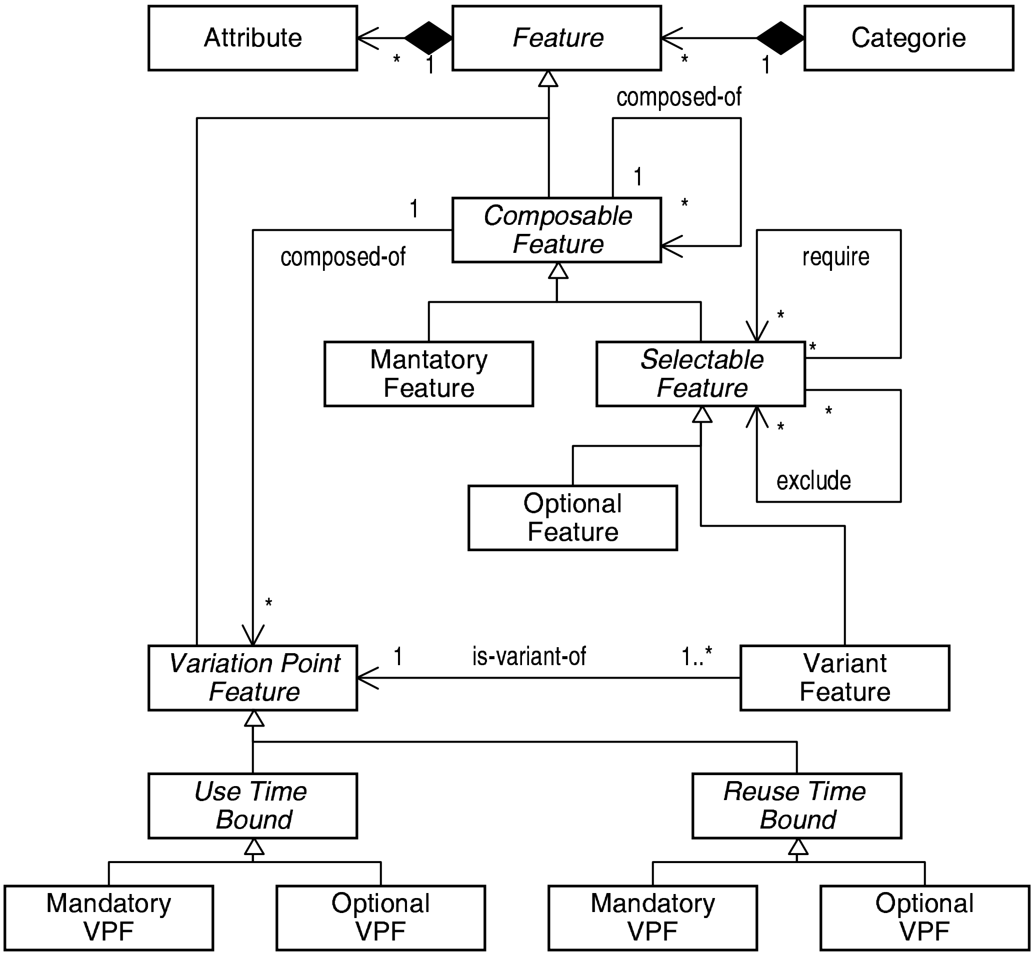


Abbildung 7. Metamodell der Feature-Modellierung der erweiterten RSEB-Methode

Beschreibung der Elemente

Element	**Beschreibung**	**Symbol**
Feature	Dient als abstraktes Modellierungselement dazu, die Gemeinsamkeiten der beiden Arten Mantadory und Optional Feature zu beschreiben	-
Composable Feature (CF)	Dient als abstraktes Modellierungselement dazu, die Feature-Arten zu beschreiben, die mit Hilfe der composed-of Beziehung hierarchisch verfeinert werden können.	-
Selectable Feature (SF)	Dient als abstraktes Modellierungselement dazu die Feature-Arten zu beschreiben, aus denen bei der Produktbildung ausgewählt werden kann.	-
Mandatory Feature	Ein solches Feature muss in jedem Produkt enthalten sein.	<Bezeichner>
Optional Feature	Ein optionales Feature kann, muss aber nicht in einem Produkt enthalten sein.	○ <Bezeichner>
Variation Point Feature (VPF)	Dieses abstrakte Element modelliert einen Variationspunkt, für den es verschiedene Alternativen (Varianten) gibt, aus denen ausgewählt werden kann.	-
Mandatory Variation Point Feature - use-time bound	Modelliert einen Variationspunkt, dessen Alternativen zur Ausführungszeit ausgewählt werden können. Dadurch wird eine OR-Auswahl modelliert	<Bezeichner> ◆
Mandatory Variation Point Feature - reuse-time bound	Modelliert einen Variationspunkt, desses Alternativen zur Konfigurationszeit ausgewählt werden können. Dadurch wird eine XOR-Auswahl modelliert.	<Bezeichner> ◇
Optional Variation Point Feature - use-time bound	Modelliert einen optionalen Variationspunkt, dessen Alternativen zur Ausführungszeit ausgewählt werden können.	○ <Bezeichner> ◇

Element	Beschreibung	Symbol
Optional Variation Point Feature - reuse-time bound	Modelliert einen optionalen Variationspunkt, desses Alternativen zur Konfigurationszeit ausgewählt werden können.	○ <Bezeichner> ◆
Attribute	Modelliert eine charakteristische Eigenschaft eines Features.	textlich
Categorie	Kategorien dienen dazu, zusammen gehörende Features zu gruppieren	-
composed-of : CF x CF , composed-of : CF x VPF	Diese Beziehung dient dazu, Features im Sinn eine Teile-Ganzes Semantik detaillierter zu beschreiben. Sind die Sub-Features Mandatory Features, dann müssen diese gewählt werden, wenn das Superfeature für ein Produkt gewählt wurde. Die Beziehung ist azyklisch und spannt einem Feature-Baum auf. Sind als Subfeatures eines Mandatory Features ausschließlich optionale Features angegeben, dann muss wenigstens eines der optionalen Features in jedem Produkt gewählt werden (1 aus n Selektion)	
require : SF x SF	Die Existenz-Abhängigkeit zwischen Features wird durch diese Beziehung modelliert. Diese Beziehung ist transitiv.	in Form von Regeln
exclude : SF x SF	Diese Beziehung modelliert, dass zwei Features inkompatibel sind und dementsprechend nicht gemeinsam augewählt werden dürfen. Diese Beziehung ist symmetrisch.	in Form von Regeln
is-variant-of : VF x VPF	Damit wird modelliert, dass ein Feature eine wählbare Alternative eines Variationspunktes ist.	

Die RSEB Erweiterung sieht vor, dass Feature-Modelle nicht nur graphisch dargestellt werden. Jedes Feature wird durch eine Reihe von Attributen beschrieben. Diese werden in der FeatuRSEB-Methode durch UML-Stereotypes definiert. Das Feature-Stereotype definiert die folgenden Attribute:

- *description*: Kurze Beschreibung des Features
- *source*: Wer hat dieses Feature eingebracht
- *nature*: Festlegung der Feature-Art (funktional, architekturell, Implementierung)
- *existence*: Optional oder Mandatory
- *alternative*: Gibt es Alternativen

- *category*: Name der Kategorie (siehe Vorgehensweise)
- *binding time*: reuse oder use
- *issuesAndDecisions*: Entscheidungshilfe bei Alternativen und Optionen
- *notes*: Zusätzliche Informationen

Die graphische Darstellung ist eine kondensierte Sicht auf das Feature-Modell.

Domänenspezifische Modellierungselemente Die FeatuRSEB-Methode definiert keine domänenspezifischen Modellierungselemente.

4.4 Vorgehensweise

Zentral für die RSEB-Methode ist die Entwicklung von Use Case Modellen. Diese werden anschließend im sogenannten Domänen Use Case Modell zusammgengeführt. Dieses Use Case Modell erlaubt es bereits, Variationspunkte explizit zu modellieren, um damit die Unterschiede ausdrücken zu können. Die vorhandene Methode wird nun um einige Schritte erweitert, um zusätzlich ein Feature-Modell zu erstellen. Im Einzelnen werden die folgenden Schritte vorgeschlagen:

- Erzeuge ein initiales Feature-Modell auf der Basis des vorhandenen Domänen Use Case Modells. Dieses initiale Feature-Modell enthält nur funktionale Features.
- Erzeuge das RSEB Objekt-Modell. Schritthaltend dazu wird das initiale Feature-Modell um architekturelle Features erweitert. Diese Features modellieren die Systemstruktur und dessen Konfiguration.
- Entwickle das RSEB Entwurfsmodell. Schritthaltend dazu wird das Feature-Modell um sogenannte Implementierungs-Features erweitert.

Der zweite Schritt *Extrahieren der Funktionalen Features auf Basis des Domänen Use Case Modells* wird weiter verfeinert. Dabei wird vorausgesetzt, dass jedes funktionale Feature genau auf einen Use Case zurückgeführt werden kann. Es ist allerdings möglich, dass ein Use Case zu mehreren funktionalen Features führt. Im einzelnen sind die folgenden Aktivitäten durchzuführen:

- *Identifiziere Mandatory und Optionale Features* Dazu sind aus der Menge der Use Cases die Top-Level Use Cases heraus zu finden. Diese können anschließend nach der Häufigkeit der Ausführung geordnet werden. Top-Level Use Cases, die sehr häufig ausgeführt werden, sind Kandidaten für Mandatory Features. Top-Level Use Cases, die nur selten ausgeführt werden, können zu optionalen Features werden. Dieser Schritt ist natürlich nicht automatisierbar. Um die Features aus den Use Cases zu identifizieren, braucht man entsprechendes Domänen Know-how.
- *Unterteile die Features in Sub-Features* Leitlinie für die Dekomposition der einzelnen Features in Sub-Features sind die im Use Case Diagramm vorhandenen extend- und include-Beziehungen.
- *Identifiziere Variation Point Features* Wenn der Use Case bereits Variationspunkte definiert, dann kann das entsprechende Feature in Sub-Features unterteilt werden, die selbst Variation Point Features sind.
- *Führe eine Robustheitsanalyse des Feature-Modells durch* Das enstandene Feature-Modell wird auf Konsistenz und Integrität geprüft. Bei diesem Prozess werden

Abhängigkeiten zwischen den Features identifiziert, die durch exclude und require-Beziehungen modelliert werden können. Weiterhin werden die Features in sogenannten Kategorien zusammengefasst. Kategorien gruppieren logisch zusammengehörende Features.

4.5 Beispiel

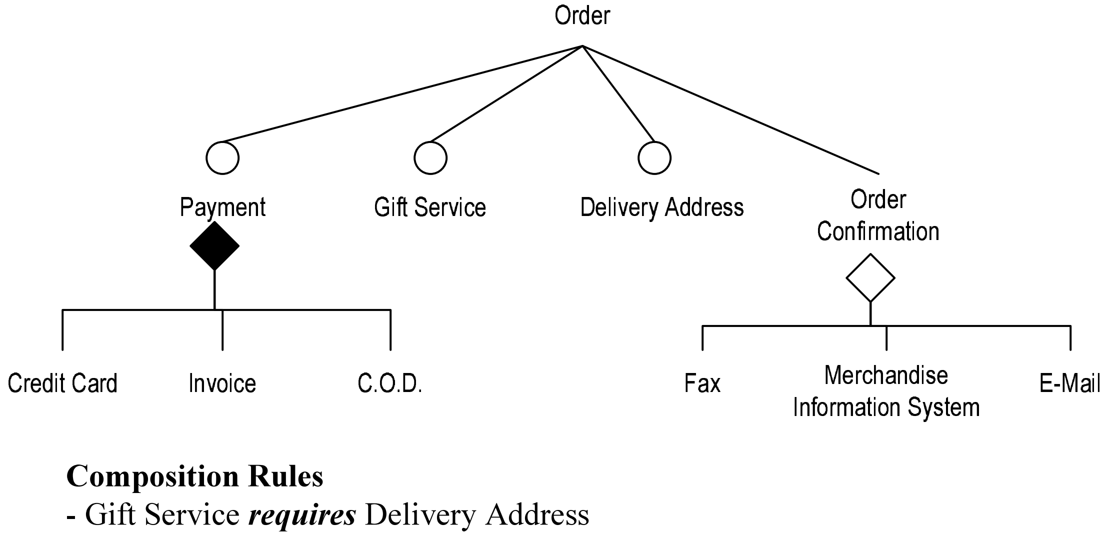


Abbildung 8. Beispiel eines Feature-Modells in der FeatuRSEB Notation

4.6 Zielerfüllung

In diesem Abschnitt soll untersucht werden, in wie weit der FeatuRSEB-Ansatz die selbstgesteckten Ziele erfüllt. Die Feature-Modelle ergänzen - wie gefordert - die Use Case Modelle, indem sie den Schwerpunkt auf die strukturellen Beziehungen zwischen den Features legen. Darüberhinaus können in den Feature-Modellen Features modelliert werden, welche nicht-funktionale Bestandteile des zu modellierenden Systems beschreiben, die bei der Use Case Modellierung nicht berücksichtigt werden. Durch die Ausstattung der Features mit Attributen kann außerdem eine vollständige Beschreibung der Eigenschaften der Features erreicht werden.

Es bleibt jedoch unklar, wie die Featuremodelle die Produktkonfiguration unterstützen sollen. Außerdem beschränkt sich die in dem Ansatz beschriebene methodische Unterstützung auf die Konstruktion von Feature-Modellen aus den domain use cases. Ein methodisches Vorgehen zur Analyse der Gemeinsamkeiten und Variabilitäten innerhalb der Domäne wird nicht beschrieben.

4.7 Bewertung

Dadurch dass mit RSEB eine bestehende Methode um die Feature-Modellierung erweitert wird, findet man bei diesem Ansatz eine Integration der Feature-Modellierung in die vorhandenen Tätigkeiten und Artefakte. Aus diesem Grund ist die Feature-Modellierung keine insuläre Tätigkeit, sondern wird in den Kontext der Domänenmodellierung der Methode RSEB eingebettet. Folgende Schwachpunkte lassen sich jedoch identifizieren:

- Obwohl methodische Schritte vorgegeben werden, wie auf der Basis des Domänen Use Case Modells ein Feature-Modell entwickelt werden kann, wird nicht deutlich, wie die Analyse der Gemeinsamkeiten und der Variabilitäten durchzuführen ist.
- Ein wesentliches Ziel der Feature-Modellierung bei diesem Ansatz ist die Unterstützung bei der Konfiguration von Produkten. Dieser sehr wichtige Schritt wird jedoch nicht behandelt und bleibt dementsprechend unklar.
- Die gewähle Notation für die Feature-Diagramme ist sehr speziell. Es ist nicht klar, wie und wo die einzelnen Informationen im Modell abgelegt sind.

Positiv kann aus unserer Sicht bewertet werden, dass die Autoren erkennen, dass Feature-Modelle nur dann effektiv und effizient erstellt und genutzt werden können, wenn eine dazu geeignete Werkzeugunterstützung vorhanden ist. Folgende wesentliche Anforderungen an diese Werkzeuge sind:

- Die gesamte Feature-Modell muss in einem zentralen Repository abgelegt sein. Dieses enthält die Datenbasis für alle Werkzeuge zur Feature-Modellierung und dient als datenorientierte Integrationsplattform.
- Es muss möglich sein, verschiedene Sichten auf das Feature-Modell zu erzeugen. Darunter fällt neben einer graphischen Repräsentation der Features in einem Feature-Diagramm auch eine Sicht, die die Abhängigkeiten zwischen den Features zeigt.
- Da die Methode verschiedene Modelle vorsieht (z.B. Domänen Use Case Modell, Feature Modell) muss die Nachverfolgbarkeit (traceability) zwischen den verschiedenen Modellen sichergestellt werden.
- Die entstehenden Modelle müssen - wie alle Produkte und Artefakte der Entwicklung von Software - einem rigorosen Konfigurationsmanagement unterstellt sein.

5 FODA -Feasibility Study

5.1 Überblick

Kang et.al. [K$^+$90] beschreiben die FODA-Methode (*Feature-Oriented Domain Analysis*), die dazu dient, Gemeinsamkeiten von verwandten Softwaresystemen zu ermitteln und zu beschreiben. Der Fokus liegt dabei auf der Identifizierung prominenter bzw. unterscheidbarer *features* der Softwaresysteme einer gegebenen Domäne. Diese Features beschreiben sowohl die gemeinsamen als auch die variablen Aspekte der Systeme einer Domäne. Die Machbarkeit des beschriebenen Ansatzes wird dabei am Beispiel einer Domäne für Window Manager Software untersucht.

5.2 Ziele der Feature-Modellierung

Die Feature-Modellierung ist laut den Autoren in den *domain analysis* Prozess eingebettet. Sie dient in erste Linie dazu, die Gemeinsamkeiten und Unterschiede der verschiedenen Softwaresystem einer Domäne zu identifizieren und zu dokumentieren. Die dabei entstehenden Modelle dienen dann als Kommunikationsmedium zwischen den Benutzern, Domänenexperten, Domänenanalysten, Systemanalysten und Entwicklern.

Ein Feature Modell soll laut Kang et.al. vor allem die folgenden Aspekte einer Domäne beschreiben:

- Funktionale Fähigkeiten (*capabilies*)
- Bereitgestellte Services
- Performance
- Benötigte Hardware-Plattform
- Kosten

5.3 Modellierungselemente

Metamodell Das in Abbildung 9 dargestellte Metamodell wurde auf Basis der Beschreibung der FODA-Methode entwickelt [K$^+$90]. Jedes Feature ist durch eine *bindtime* sowie den Typ des von ihm bereitgestellten Dienstes gekennzeichnet. Darüber hinaus besteht die Möglichkeit, dass Features *optional* sind. Weiterhin dienen *alternative features* dazu, eine exklusive Auswahl zwischen Features zu beschreiben. Features können an zwei verschiedenen Arten von Beziehungen beteiligt sein:

- Strukturelle Beziehungen
- Abhängigkeiten

Die strukturellen Beziehungen werden weiter unterschieden in eine *composedOf*-Beziehung, die eine Teil-Ganzes Beziehung beschreibt und eine *implementedBy*-Beziehung, die das Verhältnis von bereitgestelltem Dienst und dessen Realisierung beschreibt.

Bei den Abhängigkeiten wird zwischen einer *mutex* und einer *requires* Beziehung unterschieden, je nachdem, ob Features sich gegenseitig ausschließen oder ein Feature ein anderes Feature bedingt.

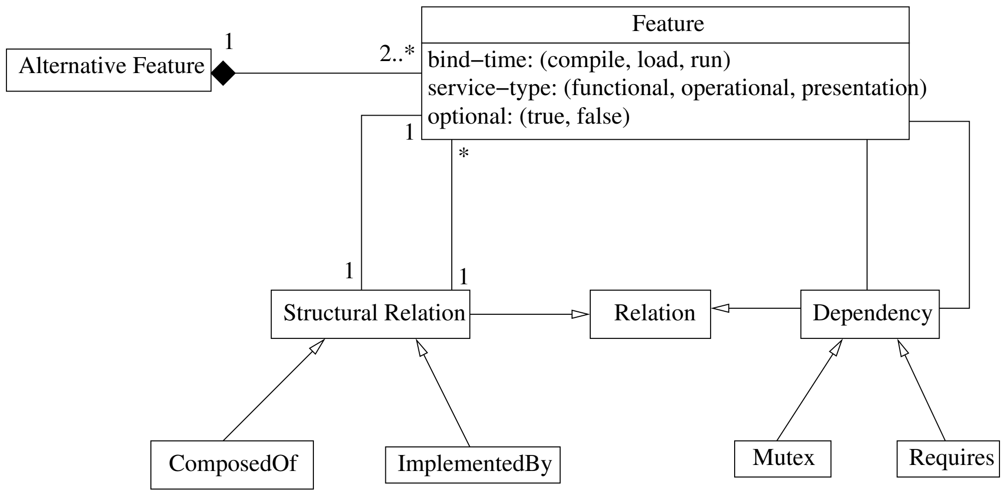


Abbildung 9. FODA Metamodell

Beschreibung der Elemente

Element	**Beschreibung**	**Symbol**
Feature	Attribute eines Systems, die den Endbenutzer direkt betreffen	Name des Features
Compile-time Feature	Features, die zu unterschiedlichen Konfigurationen der Produkte führen	keine Notation
Load-time Feature	Features, die zu Beginn der Ausführung des Programms ausgewählt werden, danach aber fest bleiben (z.B. Terminaltyp oder Missionsparameter von Waffen)	keine Notation
Runtime Feature	Features, die interaktiv oder automatisch während der Ausführung geändert werden können	Keine Notation
Functional Features	Feature, das einen Service beschreibt	Keine Notation
Operational Features	Feature, das eine Benutzerinteraktion beschreibt	Keine Notation
Presentation Features	Feature, das beschreibt, welche Information wie dargestellt wird?	Keine Notation
Generalisierung / Spezialisierung: F x F	Beschreibt eine Spezialisierung bzw. Generalisierung von Features	--------
ComposedOf (Aggregation): F x F	Beschreibt eine Teil-Ganzes Beziehung	———

Element	Beschreibung	Symbol
ImplementedBy: F x F	Beschreibt eine Realisierungsbeziehung zwischen Features	
Optional Feature	Ein Feature, dass in einem Produkt enthalten sein kann, aber nicht muss	
Alternative Feature	Beschreibt eine exklusive Auswahl zwischen Features	
Gegenseitiger Ausschluss: F x F	Beziehung zwischen zwei Features, die beschreibt, dass nicht gleichzeitig beide Features in einem Produkt vorhanden sein können	Statement
Requires: F x F	Beziehung zwischen zwei Features, die beschreibt, dass ein Feature die Existenz des anderen Features voraussetzt	Statement

Domänenspezifische Modellierungselemente In dem FODA-Ansatz werden keine domänenspezifischen Modellierungselemente verwendet.

5.4 Vorgehensweise

Die Autoren legen der Domänenanalyse, in die die Feature-Modellierung eingebettet ist, die in Abbildung 10 dargestellte Vorgehensweise zugrunde. Der Domänenanalyst erzeugt auf Basis der Erfahrungen und Wünsche der Domänenexperten und Endbenutzer sowie der Analyse existierender Systeme und Standards zunächst den *scope* der Domäne, d.h. die Abgrenzung der zu untersuchenden Domäne gegenüber anderen Domänen. Daraufhin erstellt er einen Domänen- sowie ein Architekturmodell, welche unter anderem ein Featuremodell beinhalten. Die dabei entstehenden Modelle dienen den Requirementsanalysten und Software Designern als Grundlage für die Erstellung neuer Systeme.

Laut den Autoren läßt sich mit dem FODA-Ansatz eine akkurate Beschreibung der Domäne produzieren. Der FODA-Ansatz wurde jedoch bis zur Erstellung des Berichts noch nicht für die Implementierung neuer Produkte verwendet.

5.5 Beispiel

Abbildung 11 zeigt die Verwendung der von FODA bereitgestellten Modellierungselemente.

5.6 Zielerfüllung

In diesem Abschnitt soll untersucht werden, in wie weit der FODA-Ansatz die selbstgesteckten Ziele erfüllt. FODA bietet die Möglichkeit, Gemeinsamkeiten und Variabilitäten innerhalb der Domäne abzubilden. Jedoch bleibt zu fragen, ob alle möglichen

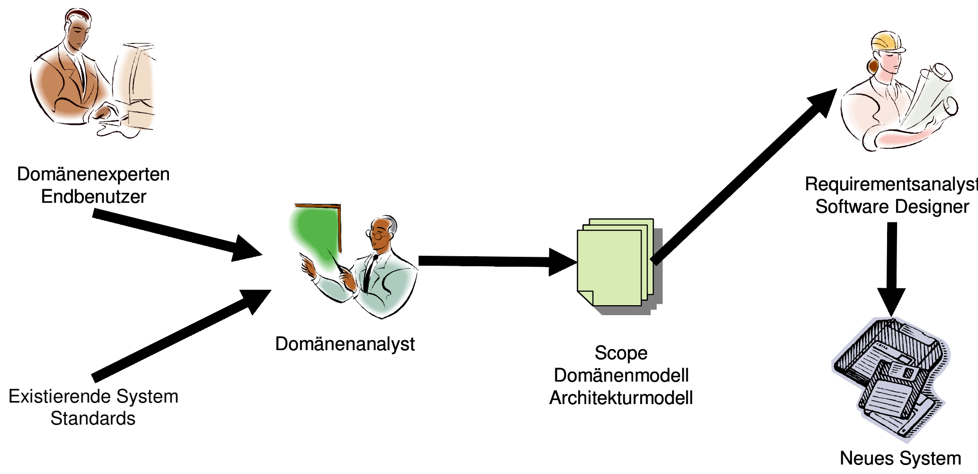


Abbildung 10. FODA: Vorgehensweise

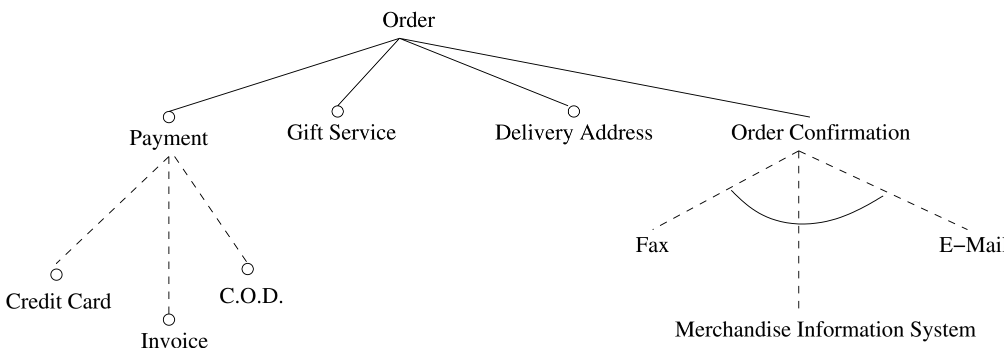


Abbildung 11. Beispiel eines FODA Feature-Modells

und notwendigen Arten von Variabilität berücksichtigt werden. Insbesondere ist es nicht möglich, Variabilitäten oder Abhängigkeiten in den Attributen der Features zu modellieren. Außerdem fehlt die Möglichkeit einer OR Auswahl.

Mit Hilfe des FODA-Ansatzes können zwar, wie gefordert, die Fähigkeiten und Dienste, die ein System zur Verfügung stellt, abgebildet werden. Jedoch beschränkt sich dies auf die Angabe eines Begriffes für diese Funktionalitäten. Es existieren keine weiteren Dokumente, die diese Funktionalitäten näher beschreiben.

Mit Hilfe der *implementedBy* Beziehung lassen sich die in den Zielen geforderten Unterschiede in den Hardware Plattformen modellieren. Es bleibt jedoch unklar, wie die unterschiedlichen Anforderungen an die Performance eines Systems modelliert werden sollen. Ebenso wird nicht erläutert, wie mit Hilfe der Feature-Modellierung nach FODA die Kosten der Systementwicklung berücksichtigt werden sollen, wie es in den Zielen gefordert wird.

Schließlich bleibt auch unberücksichtigt, wie die Feature-Modellierung als Basis für ein Architekturmodell dienen kann.

5.7 Bewertung

Mit Hilfe der Feature-Modellierung nach FODA lassen sich die Gemeinsamkeiten und Unterschiede der Systeme einer Domäne strukturiert beschreiben. Es bleibt jedoch offen, wie Features identifiziert werden können. Des weiteren kommt es - insbesondere auch durch die verwendete *implementedBy*-Beziehung - zu einer Vermischung von Problem- und Lösungsbeschreibung. Die Autoren beschreiben außerdem, dass die Feature-Modellierung als Basis der Architekturmodellierung dienen soll. Es wird allerdings nicht erläutert, wie die Architektur aus dem Feature-Modell entwickelt werden kann.

6 Lee - Domain-Oriented Engineering of Elevator Control Software

6.1 Überblick

Lee et.al. schildern in einer Fallstudie [L^+00] ihre Erfahrungen mit dem Einsatz von Feature-Modellierung zur Domänenmodellierung von Aufzugssteuerungssoftware. Dabei gliedern sie ein Feature-Modell in vier *layers*: Capability layer, operating environment layer, domain technology layer und implementation technique layer.

6.2 Ziele der Feature-Modellierung

Ziel der Feature-Modellierung ist laut den Autoren die Analyse der Gemeinsamkeiten und Variabilitäten innerhalb einer Familie von Produkten. Die Ergebnisse dieser Analyse werden in einem Feature-Modell organisiert, welches die strukturellen Aspekte der Features beschreiben soll.

6.3 Modellierungselemente

Metamodell Das in Abbildung 12 dargestellte Metamodell wurde aus der Beschreibung der Feature-Modellierung in [L^+00]extrahiert. Ein Feature-Modell besteht bei dem betrachteten Ansatz aus vier *layers*, deren innere Struktur mit Hilfe von Features modelliert wird:

- Capability
- Operating Environment
- Domain Technology
- Implementation Technique

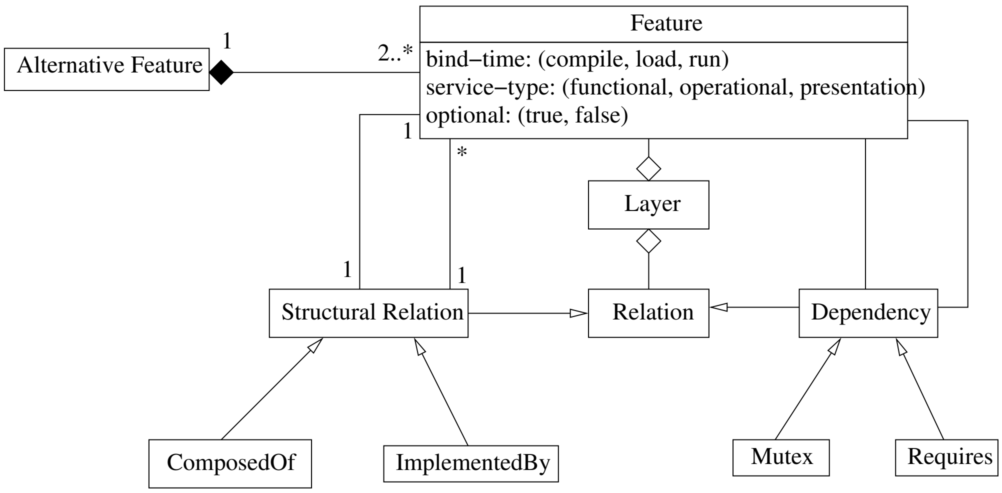


Abbildung 12. Metamodell

Der *capability layer* beschreibt das Verhalten eines Systems auf abstrakter Ebene im Sinne von bereitgestellten Diensten, Operationen und nicht-funktionalen Aspekten.

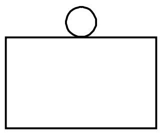
Der interne Aufbau der Features dieses layer wird mit Hilfe von *message sequence diagrams* (MSDs) und Zustandsdiagrammen sowie *data flow diagrams* (DFDs) modelliert.

Auf Ebene des *operating environment layer* werden Features modelliert, die spezifisch für eine bestimmte Hardware- oder Softwareplattform - zum Beispiel betriebssystemspezifisch - sind.

Die Features des *domain technology layer* beschreiben die Arten der Implementierung der Services und Operationen des *capability layer*. Features dieses layer werden als Komponenten modelliert.

Im *implementation technique layer* werden ähnliche Features, wie im *domain technology layer* beschrieben. Jedoch sind die Features auf dieser Ebene generischer, das heißt, sie können auch in anderen Domänen wiederverwendet werden. Ein Beispiel für ein Feature dieser Ebene wäre zum Beispiel ein Feature *ComputeQuadraticEquation*.

Beschreibung der Elemente

Element	Beschreibung	Symbol
Mandatory Feature	Ein Feature, dass in jedem Produkt der PL enthalten sein muss	
Generalisierung / Spezialisierung: F x F	Beschreibt eine Spezialisierung bzw. Generalisierung von Features	
ComposedOf (Aggregation): F x F	Beschreibt eine Teil-Ganzes Beziehung	
ImplementedBy: F x F	Beschreibt eine Realisierungsbeziehung zwischen Features	
Gegenseitige Abhängigkeiten	Nicht näher erläutert	unbekannt
Optional Feature	Ein Feature, dass in einem Produkt enthalten sein kann, aber nicht muss	
Alternative Feature	Beschreibt eine exklusive Auswahl zwischen Features	

Domänenspezifische Modellierungselemente Der beschriebene Ansatz zur Feature-Modellierung verwendet keine expliziten domänenspezifischen Modellierungselemente. Lediglich die Einteilung eines Feature-Modells in die vier *layers* kann vielleicht als domänenspezifisch angesehen werden, da zum Beispiel der *operating environment layer* eher für eingebettete Systeme von Bedeutung ist.

6.4 Vorgehensweise

Auch in diesem Ansatz ist die Feature-Modellierung in den Prozess des *domain engineering* eingebettet, vergleiche Abbildung 6.4. Zu Beginn des *domain engineering* findet eine *context analysis* statt, in der die Grenzen der Domäne ermittelt werden.

Die Ergebnisse der *context analysis* gehen in die Feature-Modellierung ein, auf deren Basis dann die operationales Modell des Systems ermittelt wird, welches als Eingabe für die nachgelagerte Architekturmodellierung dient. Dabei bilden die entstehenden Feature-Modelle die Basis für die Identifikation und Konstruktion wiederverwendbarer Komponenten.

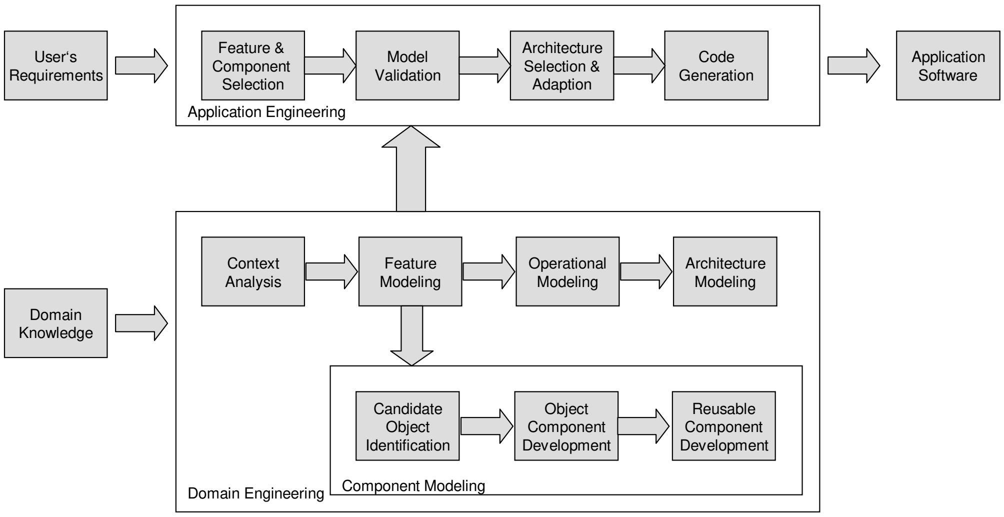
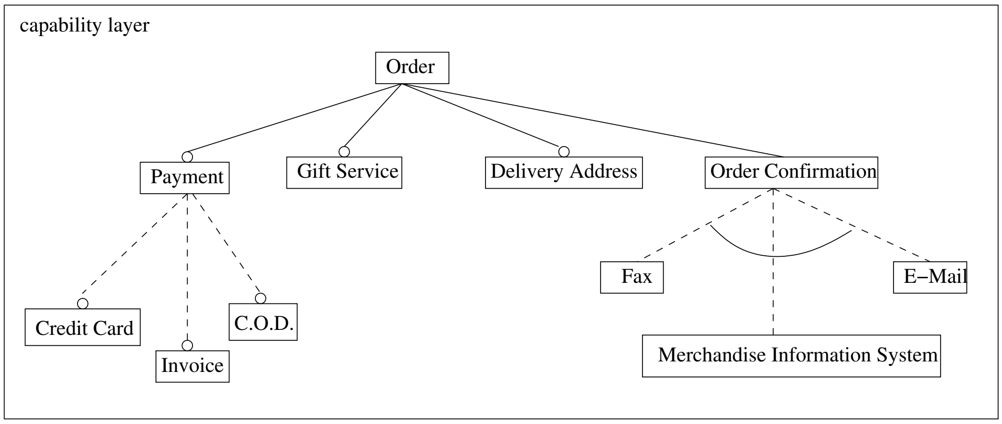


Abbildung 13. Vorgehensweise

6.5 Beispiel



6.6 Zielerfüllung

In diesem Abschnitt soll untersucht werden, in wie weit der Ansatz von Lee et.al. die selbstgesteckten Ziele erfüllt. Da der Ansatz von Lee et. al. auf dem FODA-Ansatz basiert, gelten auch hier die bereits in Abschnitt 5.6 erwähnten Einschränkungen bezüglich der Zielerfüllung. Insbesondere wird auch in diesem Ansatz keine Methodik vorgegeben, mit deren Hilfe die Gemeinsamkeiten und Variabilitäten innerhalb der Domäne ermittelt werden können.

6.7 Bewertung

Wenngleich in der vorgestellten Fallstudie durch Einsatz der Feature-Modellierung die Wiederverwendung gesteigert und die Fehlerrate gesenkt werden konnte, so wird doch keine Methodik vorgestellt, wie Features während der *context analysis* und der nachgelagerten Feature-Modellierung identifiziert werden können. Auch ist keine systematische Vorgehensweise für die Konstruktion wiederverwendbarer Komponenten auf Basis der Feature-Modellierung erkennbar. Darüberhinaus kommt es bei dem vorgestellten Ansatz zur Vermischung von Problem- und Lösungsbeschreibung, wodurch keine klare Trennung zwischen *domain engineering* und *application engineering* möglich ist.

7 Czarnecki - Feature Modellierung

7.1 Überblick

In seiner Dissertation beschreibt Czarnecki einen allgemeinen Ansatz zur Domänenmodellierung mit Hilfe der Feature-Modellierung [Cza98]. Dabei versucht er, Muster von Variabilität im Featurebaum zu identifizieren und diese auf die Architekturebene zu übertragen.

7.2 Ziele der Feature-Modellierung

Ziel der Feature-Modellierung in dem von Czarnecki beschriebenen Ansatz ist die Identifizierung und Dokumentation der Gemeinsamkeiten und Variabilitäten, sowie der Abhängigkeiten innerhalb einer Familie von Produkten.

7.3 Modellierungselemente

Metamodell Das in Abbildung 7.3 dargestellte Metamodell ist auf Basis der Beschreibung des Ansatzes von Czarnecki entwickelt worden [Cza98].

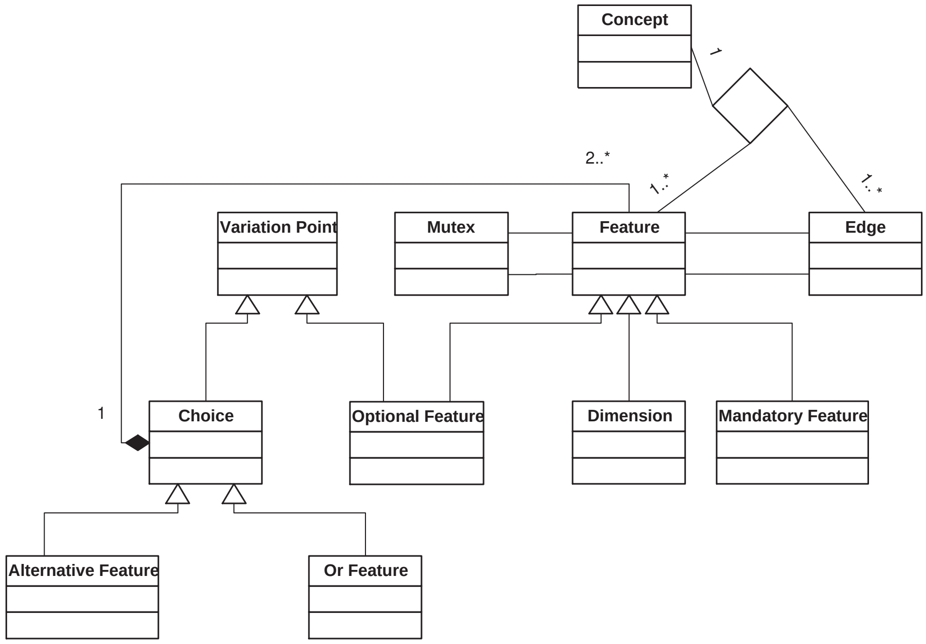


Abbildung 14. Metamodell

Die Wurzel eines Featurebaums wird in Czarnecki's Ansatz als *concept* bezeichnet. Ein *concept* ist über *edges* mit ein oder mehreren *features* verbunden. Diese *feature* sind wiederum mit Hilfe der *edges* untereinander verbunden. Die Features werden weiter unterschieden in:

- *mandatory features*
- *optional features*
- *dimensions*

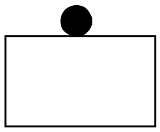
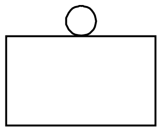
Ein *mandatory feature* ist ein Feature, dass in jedem Produkt der Produktlinie enthalten sein muss. Demgegenüber kann ein *optional feature* in einem Produkt der Produktlinie enthalten sein, muss es aber nicht. Eine *dimension* beschreibt ein Feature mit einer Menge direkter alternativer Subfeatures und ein oder mehrerer *mandatory* Subfeatures.

Neben den *optional features* identifiziert Czarnecki zwei weitere *variation points*, an denen innerhalb eines Featurebaums Variabilität auftreten kann:

- *Alternative feature*
- *Or feature*

Dabei kennzeichnet ein *alternative feature* ein Feature aus einer Menge von Features, aus der genau ein Feature ausgewählt werden muss. Ein *or feature* ist ein Feature aus einer Menge von Features, aus der ein oder mehrere Features ausgewählt werden müssen.

Beschreibung der Elemente

Element	**Beschreibung**	**Symbol**
Concept	Die Wurzel eines Featurebaums	
Mandatory Feature	Ein Feature, dass in jedem Produkt der PL enthalten sein muss	
Optional Feature	Ein Feature, dass in einem Produkt enthalten sein kann, aber nicht muss	
Kanten: F x F	Partitionierung der Unterknoten eines Knoten	
Alternative Features	Beschreibt eine exklusive Auswahl zwischen Features	

Element	Beschreibung	Symbol
Or Features	Beschreibt eine 1..m Auswahl	
Dimension	Features mit einer Menge direkter alternativer Subfeatures und ein oder mehreren mandatory Subfeatures	keine Notation
Variation Point	Feature oder Konzept, das mindestens ein direktes variables Subfeature besitzt	keine Notation
Mutually exclusive	Gegenseitiger Ausschluss zweier Features	k.A.

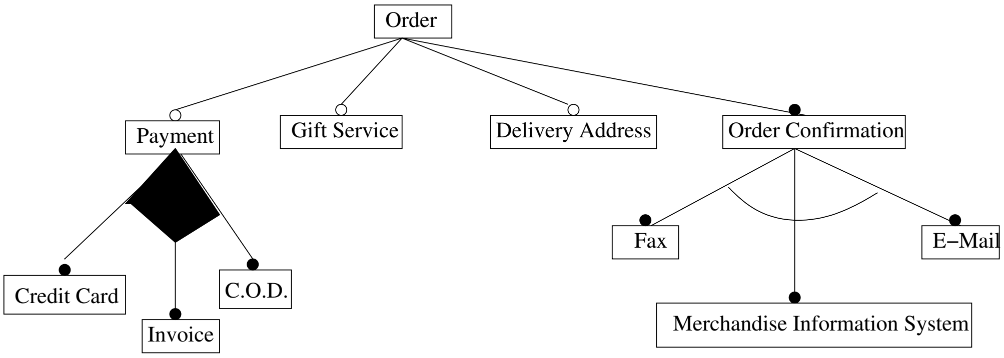
Domänenspezifische Modellierungselemente Der Ansatz von Czarnecki ist allgemein gehalten und enthält keine domänenspezifische Modellierungselemente.

7.4 Vorgehensweise

Czarnecki beschreibt keine explizite Vorgehensweise für die Identifizierung von Features und deren Anordnung in einem Featurebaum. Er gibt jedoch Richtlinien vor, die bei der Featuremodellierung beachtet werden sollen:

1. Als Eingabe für die Feature-Modellierung dienen existierende und potentielle *stakeholders*, Domänenexperten, Domänenliteratur und existierende Systeme
2. Features sollen präzise und aussagekräftige Namen - ähnlich den *design patterns* - besitzen
3. Features existieren auf jeder Ebene (Anforderungs-, Architektur-, Komponenten- und Implementierungsebene)
4. Es soll Ausschau nach Domain Terminologie gehalten werden, die Variabilität beinhaltet, zum Beispiel *thread-safe* gegenüber *not thread-safe*
5. Sogenannte *feature starter sets* sollen gebildet werden, die unterschiedliche Sichten auf die Domäne abbilden
6. Es sollen mehr Features aufgeschrieben werden, als später realisiert werden. Dies führt zu Erweiterungsmöglichkeiten der Domäne, erfordert aber gleichzeitig eine Priorisierung der Features
7. Features sollen in einem *brainstorming*-Verfahren gesammelt werden und erst danach zu einem Featurebaum organisiert werden.
8. Wo möglich, soll der Featurebaum normalisiert werden. Das heißt, eine Alternative mit einem optionalen Feature wird zu einer Alternativen mit ausschließlich optionalen Features und eine *or*-Auswahl mit einem optionalen Feature wird zu einem Diagramm mit ausschließlich optionalen Subfeatures

7.5 Beispiel



7.6 Zielerfüllung

In diesem Abschnitt soll untersucht werden, in wie weit der Ansatz von Czarnecki die selbstgesteckten Ziele erfüllt. Der Ansatz von Czarnecki erlaubt in umfassenderer Weise als z.B. FODA die Dokumentation von Gemeinsamkeiten und Variabiltiäten innerhalb einer Domäne. So ist es in Czarnecki's Ansatz möglich, eine OR Auswahl zwischen Features zu modellieren. Allerdings ist es nicht möglich, Abhängigkeiten zwischen den Features zu modellieren. Auch bleibt zu fragen, ob der Ansatz alle möglichen und notwendigen Variabilitäten berücksichtigt. Insbesondere gibt es keine Möglichkeiten Variabilität in den Attributen oder dem Verhalten der Features zu modellieren.

Auch wenn Czarnecki einige Richtlinien zur Identifikation der Gemeinsamkeiten und Variabilitäten innerhalb einer Domäne vorgibt, fehlt dem Ansatz eine methodische Vorgehensweise.

Czarnecki versucht auch, wie in den Zielen gefordert, Variabilitätsmuster innerhalb der Featuremodelle zu erkennen und diese auf Architekturebene abzubilden. Hierbei vermischt er unsere Meinung nach jedoch verschiedene Ebenen der Analyse bzw. Modellierung, wenn er z.B. eine OR Auswahl zwischen drei Features auf eine Klassenhierarchie abbildet, in der die drei Features Unterklassen einer zu dem übergeordneten Feature entsprechenden Klasse sind. Hierbei wird nicht berücksichtigt, dass es sich bei der OR Auswahl auf Ebene der Feature-Modellierung um eine Variabilität bis zum Zeitpunkt der Produktkonfiguration handelt, so dass die nicht ausgewählten Features nicht mehr in der abgeleiteten Produktarchitektur auftreten würden.

7.7 Bewertung

Der Ansatz von Czarnecki erlaubt im Gegensatz zu FODA [$K^{+}90$] die explizite Modellierung von exklusiven Alternativen sowie einer Oder-Auswahl. Es ist jedoch kein Modellierungselement vorgesehen, um Abhängigkeiten der Features untereinander modellieren zu können. Obgleich Czarnecki einige Richtlinien vorgibt, fehlt eine systematische Vorgehensweise für die Identifizierung von Features.

8 Bosch - Applying Feature Models in Industrial Settings

8.1 Überblick

Hein et.al. beschreiben einen Ansatz zur Feature-Modellierung, der die Beschränkungen des FODA-Ansatzes überwinden soll [HSVM00]. Der vorgestellte Ansatz soll es insbesondere erlauben, das Features an unterschiedlichen Stellen des Feature-Modells in unterschiedlichen Rollen auftreten können. Dazu wird der Featurebaum zu einem Featuregraph erweitert. Desweiteren wird angestrebt, die UML so zu erweitern, dass sie die Featuremodellierung unterstützt. Auf diese Weise soll eine ausreichende Werkzeugunterstützung für die Featuremodellierung sichergestellt werden.

8.2 Ziele der Feature-Modellierung

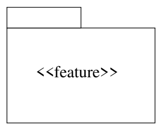
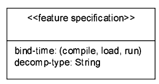
Mit ihrem Ansatz zur Feature-Modellierung verfolgen Hein et.al. fünf Ziele:

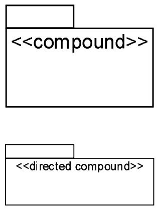
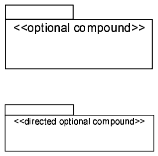
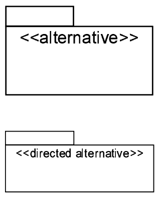
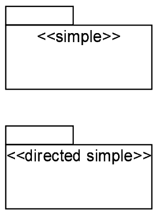
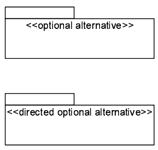
1. Modellierung von Variabilität: Modellierung der Gemeinsamkeiten und Unterschiede von Produkten einer Produktlinie
2. Wiederverwendung von Requirements
3. Konfigurationsunterstützung: Konfiguration der Produkte auf Requirementsebene mit anschließender (halb-)automatischer Codegenerierung.
4. Verkaufsunterstützung: Verkäufer haben mit den Featuremodellen eine Diskussionsgrundlage für die Kommunikation mit den Kunden.
5. Unterstützung für neue Entwicklungen: Bei der Entwicklung eines neuen Produkts vereinfacht die Feature-Modellierung die Analyse der Punkte, in denen sich das neue Produkt von den existierenden unterscheidet.

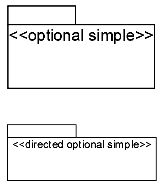
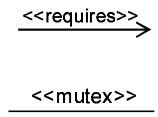
8.3 Modellierungselemente

Metamodell Das in Abbildung 8.3 dargestellte Metamodell ist der Originalarbeit entnommen.

Beschreibung der Elemente

Element	Beschreibung	Symbol
Feature	Enthält einer Menge von Rollen	
Role	Besitzt eigene binding-time und eigenen decomposition type	

Element	**Beschreibung**	**Symbol**
Compound	Container additiver Features	
Optional Compound (OC)	Optionaler Container additiver Features	
Alternative	Container exlusiver Alternativen	
Simple	Blatt des Feature-Graphen	
Mandatory Feature (MF)		
Optional Alternative (OA)	Optionaler Container exlusiver Alternativen	
Optional Feature		

Element	Beschreibung	Symbol
Optional Simple	Optionales Blatt des Feature-Graphen	
alternative: (O)A x MF	(Un)gerichtete Beziehung zwischen (optional) alternative und mandatory feature	
consists of: (O)C x R	(Un)gerichtete Beziehung zwischen (optional) compound und role	
Crosslinks	k.A.	

Domänenspezifische Modellierungselemente Der von Hein et.al. vorgestellte Ansatz wurde bei Bosch im Bereich der *car periphery supervision* (CPS) angewandt. Es werden jedoch keine domänenspezifischen Modellierungselemente vorgestellt.

8.4 Vorgehensweise

Hein et.al. beschreiben einige Richtlinien, die bei der Featuremodellierung ihrer Meinung nach berücksichtigt werden sollen. Diese Richtlinien sind aus den Erfahrungen entstanden, die die Autoren mit der Featuremodellierung bei Bosch gewonnen haben:

1. Die Textstruktur der Anforderungsspezifikation dient als Basis einer ersten Version des Feature Modells.
2. Parameter innerhalb der Anforderungen sind ein Hinweis auf Variabilität im Feature Modell und werden daher zu Features.
3. Anforderungen sind oft nach Funktionen gruppiert. Features werden im Feature Modell nach Services gruppiert, die verschiedene Funktionen zusammenfassen.
4. Service-Gruppen im Feature Modell werden weiter untergliedert, zum Beispiel nach Funktionalität oder Verantwortung. Der jeweilige *decomposition-type* wird im Modell festgehalten.
5. Es werden Zweige im Feature Modell erzeugt, welche die Variabilität angrenzender Domänen oder von Hardware beschreiben, die das Domänen-Modell beeinflussen.
6. Variabilität, die von anderen Domänen kommt, wird mit Features verbunden, die von diesen abhängen. Die Untersuchung dieser *interface features* dient als Ausgangspunkt für die Suche nach fehlenden Features.
7. Falls ähnliche Features in verschiedenen Zweigen des Modells auftauchen (ähnliche Namen), dann werden diese zuerst modelliert.

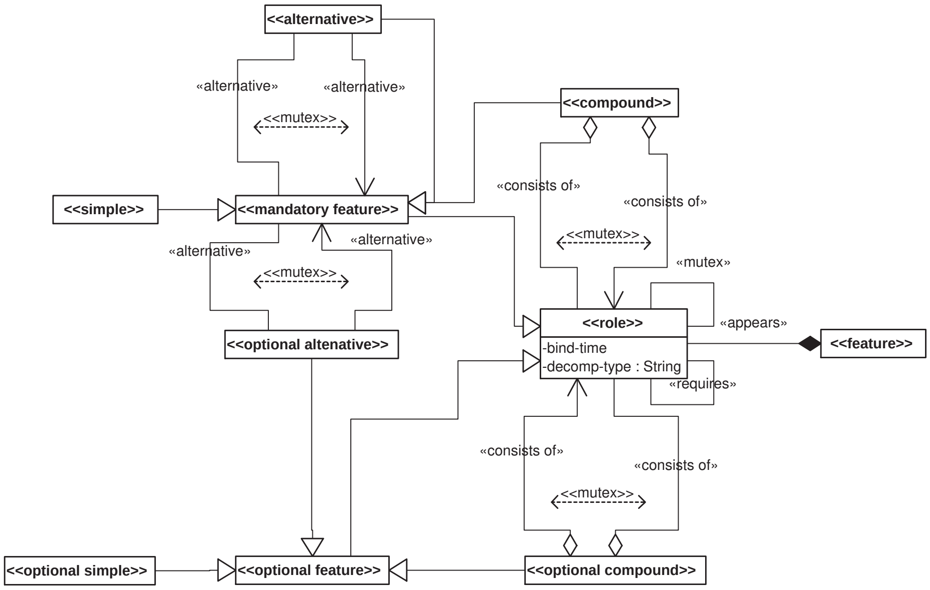


Abbildung 15. Metamodell aus [HSVM00]

8. Features, die gewöhnlich bei allen Varianten ausgewählt werden, sollten als *common features* aufgefasst und modelliert werden.
9. Falls Anforderungen oder Features einzelner Anwendungen nicht repräsentiert werden, weil die Anforderungsspezifikation unvollständig ist, dann werden sie als separate Zweige im Feature Modell modelliert.
10. Bei der Erzeugung eines neuen Features wird eine Verbindung zu dem entsprechenden Parameter in den Anforderungen hergestellt.

8.5 Beispiel

Da die Verwendung der in diesem Ansatz erläuterten Modellierungselemente weitgehend unklar ist, kann ihre Verwendung nicht am bisher verwendeten Beispiel demonstiert werden.

8.6 Zielerfüllung

In diesem Abschnitt soll untersucht werden, in wie weit der Ansatz von Hein et.al. die selbstgesteckten Ziele erfüllt. Wie in den Zielen gefordert, erlaubt der Ansatz von Hein et. al. durch die Verwendung einer Graphstruktur für die Featuremodelle, dass Features nicht mehr redundant auftreten, sondern ihre vielfältigen Beziehungen zu anderen Features über die Graphstruktur abgebildet werden können. Inwiefern dies jedoch die verschiedenen Rollen, die ein Feature annehmen kann, modelliert, bleibt unklar.

Der Forderung nach UML Unterstützung kommt der Ansatz von Hein et. al. nach, indem *stereotypes* verwendet werden, um die unterschiedlichen Modellierungselemente zu symbolisieren. Dabei bleibt es jedoch unklar, wie die beschriebenen Modellierungselemente zu verwenden sind. Insofern muss auch das Ziel, den Verkauf von Produkten mit Hilfe der Featuremodelle zu unterstützen, als nicht erfüllt angesehen werden, da die resultierenden Featuremodelle insbesondere durch die ausschließliche Verwendung von *stereotypes* in den Modellierungselementen zu schlecht lesbaren Diagrammen führt.

In dem Ansatz von Hein et.al. bleibt darüberhinaus unklar, wie die in den Zielen geforderte Wiederverwendung von Requirements sowie die Unterstützung neuer Entwicklungen erfolgen soll. Auch wird nicht erläutert, wie die Featuremodellierung als Konfigurationsunterstützung für abgeleitete Produkte dienen soll. Schließlich geht der Ansatz auch nicht näher auf die (halb)automatische Codegenerierung aus den Featuremodellen ein, welche in den Zielen gefordert wurde.

8.7 Bewertung

Das von Hein et.al. vorgestellte Metamodell ist in vielen Teilen nicht nachvollziehbar. Auch das von den Autoren angegebene Beispiel bleibt abstrakt und dient nicht der Erklärung der beschriebenen Konzepte. Des weiteren scheint die ausschließlich auf dem Konzept der *stereotypes* beruhende Erweiterung der UML zur Unterstützung der Featuremodellierung wenig geeignet, übersichtliche und leicht verständliche Modelle zu konstruieren.

9 ESAPS - Feature Modeling

9.1 Überblick

Savolainen und Vehkomäki beschreiben einen Ansatz zur Domänen Analyse [S+01], in dem mit Hilfe der Feature Modellierung die Gemeinsamkeiten und Variabilitäten innerhalb der Anforderungsspezifikationen der Produkte einer Produktlinie analysiert und repräsentiert werden. Die dabei geschilderte Vorgehensweise wurde laut den Autoren nicht an realen Produkten angewandt sondern nur exemplarisch zur Verdeutlichung der Konzepte durchgeführt.

9.2 Ziele der Feature-Modellierung

Laut Savolainen und Vehkomäki dient der vorgestellte Ansatz zur Feature Modellierung der Analyse und Repräsentation der Gemeinsamkeiten und Variabilitäten innerhalb der Anforderungsspezifikationen der Produkte einer Produktlinie. Dabei werden verschiedene Sichten definiert, welche durch die feature-basierte Domänen Analyse beschrieben werden sollen:

- Liste von *high-level* features: Eine Auflistung der Dienste und Qualitätsattribute, die den Benutzern der Domäne zur Verfügung gestellt werden
- Gruppierung der *high-level* features in *feature areas*
- Beziehungen zwischen den *feature areas* (*feature area graph*): Es werden sowohl strukturelle (Komposition) als auch dynamische Beziehungen (Uses) modelliert.
- Variabilitätssicht: Diese Sicht beschreibt die identifizierten Variabilitäten
- Quellen von Variabilität: Diese Sicht beschreibt die Ursachen für identifizierte Variabilitäten
- Produkt-basierte Priorisierung: Zuordnung einer Priorität für jedes Feature bei jedem Produkt der Produktlinie
- Konfigurationseinschränkungen: Beschreibung der Abhängigkeiten bei der Auflösung von Variabilitäten
- Konfigurationspfade: nicht näher erläutert.

Abbildung 16 stellt die Zusammenhänge zwischen den verschiedenen Sichten und der Feature Modellierung in dem betrachteten Ansatz dar.

9.3 Modellierungselemente

Der Ansatz von Savolainen und Vehkomäki definiert für die verschiedenen Sichten unterschiedliche Modellierungselemente, welche im folgenden vorgestellt werden.

Liste der high level features Die *high-level* features werden in dem Ansatz von Savolainen und Vehkomäki in Tabellenform dargestellt, ähnlich dem folgenden Beispiel (Auszug aus [S+01]).

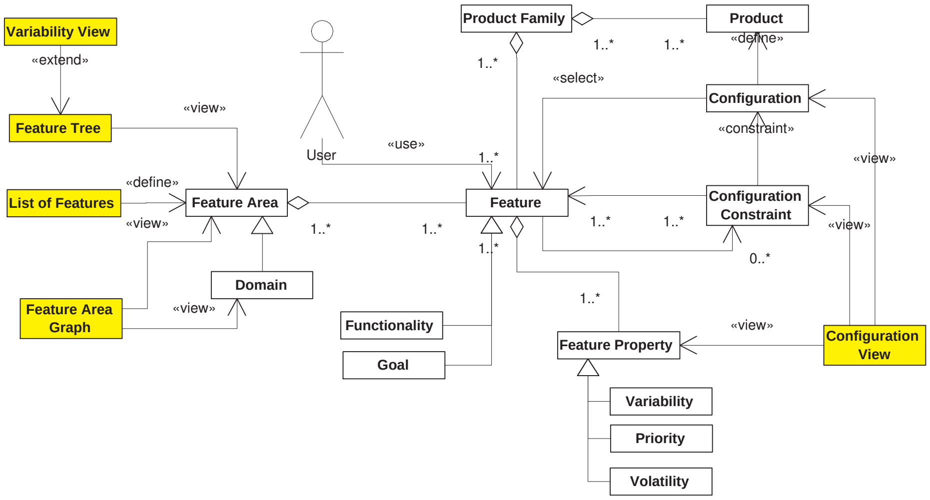


Abbildung 16. Begriffsnetz aus [S$^+$01]

ID	Feature Name	Description	Products		
			A	B	C
F1.	Calls per hour	Switching capacity as determined by the processing power of the computing platform	?	?	?
F2.	Number of subscriber lines	Total number of subscriber lines	?	?	?
F3.	Number of concurrent calls	Maximum number of concurrent calls as determined by the transmission capacity	?	?	?
F4.	Billing	Functionality to collect calls data needed for billing.	X	X	X
F5.	Remote management	Support for switch software and hardware management from a remote site	-	X	X
F6.	Software upgrading	Support for dynamic upgrading of the system software	-	-	X
F7.	ATM transmission	Support for transmission carried by the ATM protocol	X	X	-

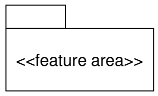
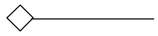
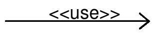
Feature Areas Die *feature areas* werden in einer Kombination aus Tabellen und Listen dargestellt. Zunächst werden in einer Tabelle die *feature areas* kurz erläutert:

Feature Area Name	Description
Telephone Switch	The entire domain covered by this model
Switching	Functionality related to call switching
Transmission	Functionality related to transmitting calls between telephone switches
Reliability	Special goal that suports the reliability of the switch
Operation and Maintenance	Functionality related to monitoring and maintenance of the switch operation.

Danach werden in einer Liste die in der jeweiligen *feature area* enthaltenen Features aufgeführt:

- **Switching**
  - F1. Calls per hour
  - F2. Number of subscriber lines

Beziehungen zwischen Feature Areas Um strukturelle sowie dynamische Beziehungen zwischen den *feature areas* zu modellieren, werden in dem betrachteten Ansatz die folgenden Modellierungselemente verwendet.

Element	**Beschreibung**	**Symbol**
Feature area	Gruppierung zusammengehöriger Features	
Komposition: FA x FA	Beschreibt eine Teil-Ganzes Beziehung von feature areas	
Uses-Beziehung: FA x FA	Beschreibt eine Benutzt-Beziehung zwischen feature areas	

Feature Tree Der Ansatz von Savolainen und Vehkomäki sieht für den *feature tree* sowohl eine textuelle Repräsentation als Liste als auch eine graphische Darstellung vor. Für das bisher betrachtete Beispiel ergibt sich folgende Darstellung als Liste:

- **Switching**
  - Call capacity
    - F1. Calls per hour
    - F2. Number of subscriber lines
  - F12. Operator specific services
    - Emergency calls
      - High priority
    - Call forwarding
    - Call waiting tone
    - Caller ID
    - Answering machine
    - Multiparty calls
- **Transmission**
  - Protocol
    - F7. ATM
    - F8. IPv4
    - IPv6

- Security
- Capacity
  - F3. Number of concurrent calls

Für die grafische Notation des *feature tree* werden die folgenden Modellierungselemente verwendet.

Element	**Beschreibung**	**Symbol**
Mandatory Feature	Ein Feature, dass in jedem Produkt enthalten sein muss	
Optional Feature	Ein Feature, dass in einem Produkt enthalten sein kann	<<optional>>
Alternative Feature	Ein Feature aus einer Menge von Features, aus der genau ein Feature ausgewählt werden muss	<<Alternative>>
Multiple Feature	Ein Feature aus einer Menge von Features, aus der ein oder mehrer Features ausgewählt werden müssen	<<Multiple>>

Abbildung 17 zeigt für das Beispiel die entsprechende grafische Notation des feature tree.

Variation Sources Auch die *variation sources* werden in Tabellenform notiert, vergleiche folgende Tabelle:

Variation Source	**Main Variations**
Capacity	Number of subscriber lines Call handling capacity Call transmitting capacity
Operator	User applications such as call forwarding multiparty calls, and answering machines

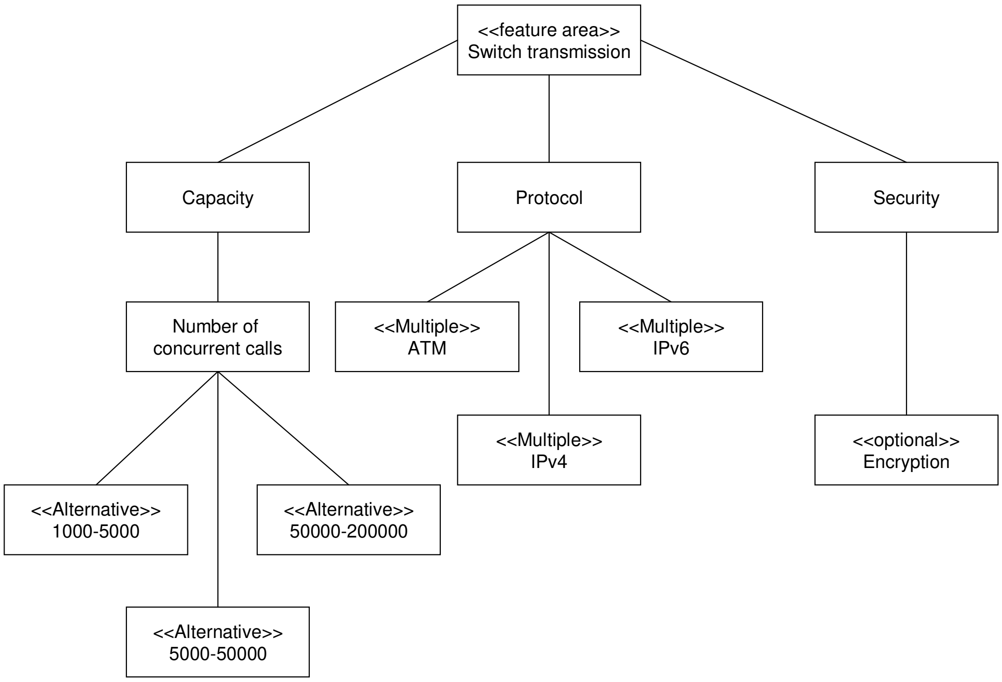


Abbildung 17. Feature Tree

Gemeinsamkeiten und Variabilitäten Die Zuordnung der Gemeinsamkeiten und Variabilitäten zu den einzelnen Produkten einer Produktlinie wird in dem betrachteten Ansatz in Tabellenform dargestellt, wobei folgende Modellierungselemente genutzt werden.

Element	**Beschreibung**	**Symbol**
Mandatory Feature	Ein Feature, dass in jedem Produkt enthalten sein muss	m
Optional Feature	Ein Feature, dass in einem Produkt enthalten sein kann	o
Alternative Feature	Ein Feature aus einer Menge von Features, aus der genau ein Feature ausgewählt werden muss	a
Multiple Feature	Ein Feautre aus einer Menge von Features, aus der ein oder mehrer Features ausgewählt werden müssen	u

Für das bisher verwendete Beispiel aus [S^+01] ergibt sich folgende Tabelle:

	Transmission			**Product**			
					A	**B**	**C**
	Protocol			m			
F7		ATM		u	X	X	-
F8		IPv4		u	-	X	X
		IPv6		u	-	-	X
	Security			m			
		Encryption		o	-	X	X
	Capacity			m			
F3		Number of concurrent calls		m			
			1000-5000	a			
			5000-50000	a		X	X
			50000-200000	a	X		

Priorisierung Für die Prisorisierung von Features geben Savolainen und Vehkomäki folgendes Klassifizierungsschema vor:

Prioritätswert	Klassifikation
0	Not allowed
1	Don't care
2	Might be useful
3	Wish
4	Useful
5	Somewhat important
6	Quite important
7	Important
8	Very important
9	Mandatory

Hierbei wird wiederum eine Tabellenform genutzt, um die Priorisierung der Features in den einzelnen Produkten darzustellen:

Feature	Max number of calls	Priorization (A)	Priorization (B)	Priorization (C)
F3	1000 - 5000	Not acceptable	Mandatory	Mandatory
F3	5000 - 50000	Mandatory	Important	Important
F3	50000 - 200000	Important	Wish	Wish
F3	200000 - 500000	Wish	Not needed	Not needed

Domänenspezifische Modellierungselemente Dem Ansatz von Savolainen und Vehkomäki liegt als Beispiel eine Domäne für *telephone switch* Systeme zugrunde. Allerdings werden in dem Ansatz keine domänenspezifischen Modellierungselemente eingeführt.

9.4 Vorgehensweise

Bei der Erläuterung der Vorgehensweise bleiben Savolainen und Vehkomäki vage:

By performing various analyses - commonality and variability assumptions can be automatically gathered from the requirement specifications.

Eine genauere Beschreibung der notwendigen Analyseschritte unterbleibt.

9.5 Zielerfüllung und Bewertung

Der Ansatz von Savolainen und Vehkomäki ermöglicht durch die Definition der verschiedenen Sichten eine umfassende Darstellung der betrachteten Domäne aus verschiedenen Blickwinkeln. Allerdings fehlt auch in diesem Ansatz eine systematische Vorgehensweise zur Identifikation der Features, wodurch in dem angegebenen Beispiel nicht immer erkennbar ist, wann eine Systemeigenschaft als Feature oder als Attribut eines Features modelliert werden soll, z.B. im Fall der unterschiedlichen Anzahl gleichzeitiger Anrufe in Abbildung 17.

10 Der Feature-Begriff

Um einen detaillierten Vergleich der Ansätze zu ermöglichen, ist auch eine eingängigere Betrachtung des in den Ansätzen verwendeten *Feature*-Begriffs sinnvoll. Dabei fällt auf, dass nahezu alle Ansätze durch den FODA-Ansatz [K$^+$90] geprägt oder doch zumindest stark beeinflusst wurden.

Die Definition des Begriffs *Feature* betreffend äußert sich dies darin, dass, sofern eine explizite Definition des Feature-Begriffs gegeben wird, diese entweder die Begriffs-Definition des FODA-Ansatzes direkt referenziert [LKL02][FFB02][L$^+$00] [HSVM00] oder sich bei einer eigenen Definition des Begriffs sehr stark auf diejenige des FODA-Ansatzes stützt [G$^+$98]. Auch wenn eine solch explizite Definition nicht gegeben wird [S$^+$01] und man die Bedeutung des Begriffs *Feature* nur implizit ableiten kann, lässt sich dies beobachten.

Daher ist es nicht verwunderlich, dass der Mehrheit der Ansätze ein - wenn auch weit gefasstes - gemeinsames Verständnis des Begriffs *Feature* zugrunde liegt, dass ein *Feature* entsprechend der FODA-Definition als eine(n)

> *hervorstechende(n) oder unterscheidende(n) für den Endnutzer sichtbare(n) Aspekt, Qualität oder Charakteristik einer Software oder eines Systems*

bezeichnet. Diese recht allgemeine Definition wird in FODA durch eine Klassifizierung von Features nach der Zugehörigkeit zu

1. Einsatzumgebung
2. Fähigkeiten (engl. *capabilities*)/Dienste
3. Domänentechnik
4. Implementierungstechnik

näher präzisiert. Die Ausprägung des Feature-Begriffs in den einzelnen Ansätzen unterscheidet sich insbesondere dadurch, dass entweder eine bestimmte Klasse von Features besonders betont oder der Feature-Begriff sogar vollständig auf eine bestimmte Klasse von Features eingeschränkt wird. Dies ist teilweise bereits beim FODA-Ansatz selbst zu beobachten, wo in der validierenden Fallstudie nur die Dienst-Features untersucht und in

1. Funktionale Features
2. Operationale Features
3. Präsentations-Features

weiter unterschieden werden. Erst in einer zweiten, in deutlichem Zeitabstand nachgelieferten Fallstudie zum FODA-Ansatz [L$^+$00] werden auch die übrigen Feature-Klassen berücksichtigt.

Wenn auch bereits im ursprünglichen FODA-Ansatz explizit darauf hingewiesen wird, dass die dort getroffene Einschränkung nur aus rein zeitlichen Gründen stattfand, so hat dies die nachfolgenden Ansätze in sofern entscheidend mitgeprägt, dass sich eine starke Betonung der funktionalen Features bei nahezu allen anderen Ansätzen abzeichnet. Besonders stark kommt dies in [KKL$^+$98] zum Ausdruck, wo ein Feature aus Sicht des Endanwenders als eine *unterscheidbare funktionale Abstraktion eines Systems* betrachtet wird (abweichend davon wird der Feature-Begriff im Ansatz aber durchaus anders eingesetzt, so dass beispielsweise auch Implementierungstechniken berücksichtigt werden). Auch in *griss98* wird diese Sicht vertreten, wobei als Abgrenzungskriterium eines Features gegenüber einem Use-Case der systemübergreifende Charakter betont wird, d.h. ein Feature beschreibt eine Funktionalität einer bestimmten Domäne, nicht eines einzelnen Systems.

Neben dieser sehr starken Betonung der funktionalen Features läßt sich, getrieben durch den Einsatz der Feature-Modellierung in der Produktlinienentwicklung, bei nahezu allen Ansätzen eine starke Fokussierung auf den Unterscheidungscharakater eines Features beobachten, d.h. ein Feature wird als ein unterscheidendes Merkmal von Produkten betrachtet. Dies hängt damit zusammen, dass die Feature-Modellierung zur Modellierung der Variabilitäten in einer Produktlinie eingesetzt wird und kommt vor allem in [HSVM00] besonders zum Ausdruck, wo die Unterscheidung zwischen einem Feature und einer *Anforderung* darin gesehen wird, dass ein Feature zur Modellierung der Variabilitäten und eine Anforderung zur Beschreibung der Gemeinsamkeiten der Produkte einer Produktlinie eingesetzt wird.

Eine Sonderstellung bezüglich der Begriffsdefinition nimmt der von Czarnecki [Cza98] beschriebene Ansatz ein, der die FODA-Begriffsdefinition nicht weiter einschränkt, sondern noch erweitert. Die von Czarnecki gelieferte Begriffsdefinition sieht ein Feature als eine *unterscheidende Eigenschaft eines Konzepts* und legt den Schwerpunkt nicht auf die Sichtbarkeit eines Features für den Endnutzer sondern allgemeiner auf die Relevanz eines Features für eine Gruppe von Interessenten (*Stakeholdern*). Diese etwas weiter gefasste Begriffsdefinition ist stark durch den ODM-Ansatz (Organizational Domain Modeling) geprägt, wo die Nachverfolgbarkeit (engl. *traceability*) jedes Konzepts oder Features bezüglich der zugehörigen Interessensgruppe explizit gefordert wird.

11 Vergleich von Zielsetzung und verwendeten Modellierungselementen

Im folgenden Abschnitt werden die in diesem Bericht vorgestellten Ansätze für die Feature-Modellierung dahingehend untersucht, in wie weit sie in ihrem Verwendungszweck und den bereitgestellten Modellierungselementen übereinstimmen.

11.1 Ziele der Feature-Modellierung

Vergleicht man nun die hier beschriebenen Ansätze und Konzepte der Feature-Modellierung, so lassen sich eine Reihe von Gemeinsamkeiten, aber auch Unterschiede bezüglich der Motivation für die Feature-Modellierung feststellen. Die mit der Feature-Modellierung verfolgten Ziele spiegeln sich ebenfalls in den entsprechenden Modellierungselementen wieder, welche speziell für die entsprechenden Ansätze eingeführt worden sind.

Alle hier vorgestellten Ansätze möchten Funktionalitäten und qualitative Charakteristiken eines Anwendungsbereichs abstrakter modellieren, als es auf der konkreten Anforderungsebene möglich wäre. Dazu ist das Modellierungselement *Feature* gewählt worden. Alle Autoren verstehen Features als abstraktes Hilfsmittel, um eine Menge von Anforderungen zu beschreiben. Neben der Identifizierung und Analyse der Features steht des weiteren die Modellierung struktureller Beziehungen im Vordergrund. Dies bedeutet, dass die identifizierten Features hierarchisch in einer Baum- bzw. Graphstruktur angeordnet werden. Gemeinsamer Konsens aller Ansätze ist die Motivation, Gemeinsamkeiten und Variabilitäten zu modellieren. Hierbei erhalten beide Phänomene die gleiche Priorität. Gemeinsamkeiten dienen dazu, die entsprechenden Charakteristiken wiederzuverwenden, während das Wissen darüber, an welchen Stellen für eine Charakteristik unterschiedliche Varianten gewählt werden können, ebenfalls von großer Bedeutung ist, um die komplexe Domänen besser verstehen zu können.

Große Unterschiede zeigen sich jedoch in den Zielen, die mit der Feature-Modellierung verfolgt werden. Die Autoren von [KKL$^+$98], [K$^+$90] und [LKL02] verfolgen mit der Feature-Modellierung das Ziel, eine Domänen-Beschreibung zu erhalten sowie ein Begriffslexikon aufzubauen, welches die Kommunikation zwischen den Entwicklern und den beteiligten Domänenexperten unterstützen soll. Die Feature-Modellierung wird somit als Technik für die Domänenanalyse und -dokumentation eingesetzt.

Mit einem anderen Ziel, nämlich der Dokumentation einer Produktlinienbeschreibung, setzen die Autoren von [FFB02], [LKL02], [Cza98], [HSVM00] und [S$^+$01] die Feature-Modellierung ein. Die entwickelten Modelle dienen somit dazu, potentielle Produktfunktionalitäten und -qualitäten zu identifizieren. Neben der Identifizierung, werden die Modelle ebenfalls herangezogen, um Produktkonfigurationen zu ermitteln. Fey et al [FFB02] haben ihren Feature-Ansatz soweit formalisiert, dass sogar eine automatische Analyse der Feature-Modelle durchgeführt werden kann, um festzustellen, ob die abgeleiteten Produkte in sich konsistent sind. Neben der Dokumentation der Domäne(n), für die eine Produktlinie entwickelt werden soll, verfolgen diese Ansätze zusätzlich das Ziel, das Konfigurationsmanagement auf abstrakter Ebene zu unterstützen.

Interessant ist allerdings die Beobachtung, dass gerade die Methoden *FORM* und *FODA* sehr stark lösungsorientierte Modelle in ihren Arbeiten vorstellen. Die modellierten Features beschreiben somit (hauptsächlich funktionale) Charakteristiken auf implementierungstechnischer Ebene. Es werden also für den im Feature-Baum weiter oben modellierten Problemraum, Lösungen in den Blättern des Baums mitmodelliert.

Eine Sonderstellung nimmt der Ansatz von Griss et al [G+98] ein, der die Feature-Modellierung als unterstützende Methode zur Use Case Modellierung einsetzt und in die *RSEB*-Methode eingebettet wurde. Die Feature-Modelle dienen hauptsächlich dazu, qualitative Charakteristiken einer Domäne zu beschreiben, da die funktionalen Aspekte bereits durch die Use Case Beschreibungen und Diagramme dokumentiert sind. Die Feature-Modelle sollen ebenfalls nicht vollständig sein, sondern allein die funktionale und qualitative Essenz ausdrücken. Ergänzend sollen die Feature-Modelle aber ebenfalls dazu dienen, Produktkonfigurationen aufgrund von Gemeinsamkeiten und Variabilitäten abzuleiten.

11.2 Modellierungselemente

Die eingesetzten Modellierungselemente zur Visualisierung der Feature-Modelle in Diagrammen lehnen sich sehr stark an die durch die *FODA*-Methode ursprünglich vorgeschlagenen Elemente an. Da alle Ansätze die Feature-Modellierung einsetzen, um Gemeinsamkeiten und Variabilitäten auszudrücken, finden sich in allen Diagrammen *mandatory, optional* und *alternative* Features wieder. Ebenso wird zwischen verschiedenen Beziehungen differenziert, die ausdrücken, in welcher Beziehung ein Vater-Feature zu einem Kind-Feature im Feature-Baum steht: Teile-Ganzes-Beziehungen, Generalisierungs- bzw. Spezialisierungs-Beziehungen sowie Implementierungs-Beziehungen sollen die entsprechenden Hierarchiebeziehungen typisieren. Neben den Beziehungen sehen es ebenfalls alle Ansätze vor, Abhängigkeiten zwischen Features zu definieren, die nicht zwingend im gleichen Unterbaum liegen, um somit Implizitäten und wechselseitige Ausschlüsse zu modellieren. Abweichungen lassen sich hier nur in der Form der Modellierung erkennen. So unterstützen die Ansätze von [K+90], [KKL+98] und [G+98] die Abhängigkeiten durch Angabe von textuell beschriebenen Regeln, während die anderen Methoden dafür explizite Modellierungselemente (gerichtete Pfeile) eingeführt haben.

Ebenso stellen alle Ansätze heraus, dass Elemente bzw. Techniken benötigt werden, die es erlauben, eine Menge von Features zu gruppieren. Die Intentionen, die hinter der Gruppierungen liegen unterscheiden sich allerdings wieder. Während die Autoren von [FFB02], [G+98] und [HSVM00] frei wählbare, logische Gruppierungen ermöglichen, versuchen die Autoren von [L+00] und [LKL02] Features in sogenannte *Layers* einzuordnen. Features in den jeweiligen Layern kennzeichnen somit Dienste in den Bereichen *Capability, Domain Technology, Operating Environment* und *Implementation Technique*. Die Zuordnung von Features in diese Layer zeigen ebenfalls den sehr lösungsorientierten Modellierungsgedanken, der weiter oben beschrieben wurde.

Neben dem Strukturierungselement der Gruppierung schlagen einige Ansätze vor, unterschiedliche Sichten auf die Feature-Modelle zu nutzen, um die Komplexität der Diagramme zu reduzieren. So schlagen beispielsweise Fey et. al. [FFB02] vor, Sichten zu verwenden, um nur die Informationen in einer Sicht darzustellen, für die man sich momentan interessiert.

Alle hier vorgestellten Ansätze sind Domänen-unabhängig spezifiziert, so dass keine domänenspezifischen Modellierungselemente eingeführt werden. Allein die *FORM*-

Methode versucht Features in die oben erwähnten Layern einzuordnen, welches man als Ansatz für eine Domänenspezifizierung deuten kann.

Neben dem großen Konsens der Ansätze bei den Modellierungselementen zeigen sich hingegen Unterschiede im Detaillierungsgrad der Modellierung. So bieten die Ansätze von [FFB02], [G^+98] und [S^+01] die Möglichkeit, Features zu attributieren. Dies spiegelt sich ebenfalls in den dafür benötigten Modellierungselementen wider: Spezielle Beziehungen und Abhängigkeiten werden benötigt, um auszudrücken, dass die Auswahl bzw. die Parametrisierung eines Features eine bestimmte Wertbelegung eines Attributs eines anderen Features impliziert.

FODA und *FeatuRSEB* unterscheiden zudem den Auflösungszeitpunkt der Variationspunkte, um auszudrücken, ob der Variationspunkt bei der Produktinstanziierung oder erst zur Laufzeit aufgelöst wird. Dieser Mechanismus erlaubt eine sehr viel detailliertere Modellierung von Variabilität und unterstützt den Entwickler bei der Produktkonfiguration und der Überprüfung von Abhängigkeiten zwischen variablen Features.

11.3 Fazit

Insgesamt lässt sich festhalten, dass zwei große Ziele mit der Feature-Modellierung verfolgt werden. Zum einen dient sie als Methode für die Domänenanalyse und zum zweiten zur Modellierung von Produktlinien mit gleichzeitiger Unterstützung für die Produktkonfigurationen.

Alle Ansätze basieren in den Modellierungselementen im wesentlichen auf denen, die im ursprünglichen *FODA*-Ansatz zur Modellierung von Gemeinsamkeiten und Variabilitäten vorgeschlagenen wurden. Abweichungen finden sich vor allem in der Einführung von speziellen Abhängigkeitsbeziehung und Feature-Attributen. Diese werden insbesondere für die automatische Konsistenzprüfung der Modelle für die Produktkonfigurationen benötigt.

12 Bewertung

Die betrachteten Ansätze zur Featuremodellierung erwiesen sich in unseren Augen als geeignet, um die Gemeinsamkeiten und Variabilitäten einer Domäne zu modellieren. Jedoch fehlt den Ansätzen eine Methodik zur zielgerichteten Modellierung. So mangelt es den Ansätzen beispielsweise an einer systematischen Vorgehensweise zur Identifizierung von Features. Auch bleibt die gegenseitige Abgrenzung von Features unklar.

Des weiteren wurde bei der Betrachtung der verschiedenen Ansätze die Notwendigkeit erkannt, mit Hilfe von unterschiedlichen *Sichten* auf ein Featuremodell die Handhabbarkeit und Skalierung zu gewährleisten. Als weiterer Abstraktionsmechanismus müssen Modelle hierarchisch organisiert und strukturiert werden können, damit eine bessere Skalierung der Modellierungstechnik erreicht wird.

Weiterhin wurde in den einzelnen Ansätzen zur Featuremodellierung oft der Eindruck erweckt, die Featuremodelle dienten als alleinige Grundlage für die nachgelagerten Modellierungsschritte. Die Featuremodelle können jedoch nur eine Ergänzung eines umfassenderen Requirementsmodells sein, welches durch weitere Modelle und Beschreibungen, wie beispielsweise Anforderungsspezifikationen und Dokumentation der Features ergänzt werden muss. So beschreibt das Featuremodell nur eine statische Sicht auf die Problemdomäne. Für die nachgelagerten Modellierungsschritte müssen jedoch auch Variabilitäten im dynamischen Verhalten modelliert werden können.

Schließlich blieb bei allen Ansätzen eine Möglichkeit, produktspezifische Erweiterungspunkte zu modellieren, unberücksichtigt. Um eine durchgängige Modellierungstechnik zu ermöglichen, ist die Berücksichtigung dieser produktspezifischen Variationspunkte in unseren Augen nicht vernachlässigbar.

Zusammenfassend kann die Feature-Modellierung als adäquate Technik angesehen werden, um die Modellierung der Variabilitäten und Gemeinsamkeiten einer Produktlinie zu unterstützen. Dabei muss sie jedoch von weiteren Tätigkeiten und Modellierungstechniken des Requirements Engineering ergänzt werden, damit das Gesamtresultat dieses Modellierungsprozesses als Ausgangsbasis für die nachgelagerten Entwicklungsphasen dienen kann.

Literatur

[Cza98] K. Czarnecki. *Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models*. PhD thesis, Technische Universität Illmenau, 1998.

[FFB02] D. Fey, R Fajta, and A. Boros. Feature modeling: A meta-model to enhance usability and usefullness. In *SPLC2*, LNCS 2379, pages 198–216. Springer, 2002.

[FHS02] S. Ferber, J. Haag, and J. Savolainen. Feature interaction and dependencies: Modeling features for reengineering a legacy product line. In *SPLC2*, LNCS 2379, pages 235–256. Springer, 2002.

[G+98] Griss et al. Integrating feature modeling with the RSEB. In *Proceedings of the Fifth International Conference on Software Reuse*. IEEE Computer Society, Los Alamitos, CA, USA, 1998.

[HSVM00] A. Hein, M. Schlick, and R. Vinga-Martins. Applying feature models in industrial settings. In P. Donohoe, editor, *Software Product Lines. Experience and Research Directions*, pages 47–70. Kluwer Academic Publishers, 2000.

[JGJ97] Ivar Jacobson, Martin Griss, and Patrik Jonsson. *Software Reuse*. ACM Press, 1997.

[K+90] K.C. Kang et al. Feature-oriented domain anaylsis (foda) feasibility study. Technical report, CMU/SEI, November 1990.

[KKL+98] K.C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. Form: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, 5:143–168, 1998.

[L+00] K. Lee et al. Domain-oriented engineering of elevator control software. In P. Donohoe, editor, *Software Product Lines. Experience and Research Directions*, pages 3–22. Kluwer Academic Publishers, 2000.

[LKL02] K. Lee, C. Kang, and J. Lee. Concepts and guidelines of feature modeling for product line software engineering. In *ICSR-7*, LNCS 2319, pages 62–77. Springer, 2002.

[S+01] J. Savolainen et al. Feature analysis. Technical report, ESAPS, June 2001.

Aachener Informatik-Berichte

This is a list of recent technical reports. To obtain copies of technical reports please consult http://aib.informatik.rwth-aachen.de/ or send your request to: Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Email: `biblio@informatik.rwth-aachen.de`

95-11 *	M. Staudt / K. von Thadden: Subsumption Checking in Knowledge Bases
95-12 *	G.V. Zemanek / H.W. Nissen / H. Hubert / M. Jarke: Requirements Analysis from Multiple Perspectives: Experiences with Conceptual Modeling Technology
95-13 *	M. Staudt / M. Jarke: Incremental Maintenance of Externally Materialized Views
95-14 *	P. Peters / P. Szczurko / M. Jeusfeld: Business Process Oriented Information Management: Conceptual Models at Work
95-15 *	S. Rams / M. Jarke: Proceedings of the Fifth Annual Workshop on Information Technologies & Systems
95-16 *	W. Hans / St. Winkler / F. Sáenz: Distributed Execution in Functional Logic Programming
96-1 *	Jahresbericht 1995
96-2	M. Hanus / Chr. Prehofer: Higher-Order Narrowing with Definitional Trees
96-3 *	W. Scheufele / G. Moerkotte: Optimal Ordering of Selections and Joins in Acyclic Queries with Expensive Predicates
96-4	K. Pohl: PRO-ART: Enabling Requirements Pre-Traceability
96-5	K. Pohl: Requirements Engineering: An Overview
96-6 *	M. Jarke / W. Marquardt: Design and Evaluation of Computer–Aided Process Modelling Tools
96-7	O. Chitil: The ς-Semantics: A Comprehensive Semantics for Functional Programs
96-8 *	S. Sripada: On Entropy and the Limitations of the Second Law of Thermodynamics
96-9	M. Hanus (Ed.): Proceedings of the Poster Session of ALP'96 — Fifth International Conference on Algebraic and Logic Programming
96-10	R. Conradi / B. Westfechtel: Version Models for Software Configuration Management
96-11 *	C. Weise / D. Lenzkes: A Fast Decision Algorithm for Timed Refinement
96-12 *	R. Dömges / K. Pohl / M. Jarke / B. Lohmann / W. Marquardt: PRO-ART/CE* — An Environment for Managing the Evolution of Chemical Process Simulation Models
96-13 *	K. Pohl / R. Klamma / K. Weidenhaupt / R. Dömges / P. Haumer / M. Jarke: A Framework for Process-Integrated Tools
96-14 *	R. Gallersdörfer / K. Klabunde / A. Stolz / M. Eßmajor: INDIA — Intelligent Networks as a Data Intensive Application, Final Project Report, June 1996

96-15 * H. Schimpe / M. Staudt: VAREX: An Environment for Validating and Refining Rule Bases
96-16 * M. Jarke / M. Gebhardt, S. Jacobs, H. Nissen: Conflict Analysis Across Heterogeneous Viewpoints: Formalization and Visualization
96-17 M. Jeusfeld / T. X. Bui: Decision Support Components on the Internet
96-18 M. Jeusfeld / M. Papazoglou: Information Brokering: Design, Search and Transformation
96-19 * P. Peters / M. Jarke: Simulating the impact of information flows in networked organizations
96-20 M. Jarke / P. Peters / M. Jeusfeld: Model-driven planning and design of cooperative information systems
96-21 * G. de Michelis / E. Dubois / M. Jarke / F. Matthes / J. Mylopoulos / K. Pohl / J. Schmidt / C. Woo / E. Yu: Cooperative information systems: a manifesto
96-22 * S. Jacobs / M. Gebhardt, S. Kethers, W. Rzasa: Filling HTML forms simultaneously: CoWeb architecture and functionality
96-23 * M. Gebhardt / S. Jacobs: Conflict Management in Design
97-01 Jahresbericht 1996
97-02 J. Faassen: Using full parallel Boltzmann Machines for Optimization
97-03 A. Winter / A. Schürr: Modules and Updatable Graph Views for PROgrammed Graph REwriting Systems
97-04 M. Mohnen / S. Tobies: Implementing Context Patterns in the Glasgow Haskell Compiler
97-05 * S. Gruner: Schemakorrespondenzaxiome unterstützen die paargrammatische Spezifikation inkrementeller Integrationswerkzeuge
97-06 M. Nicola / M. Jarke: Design and Evaluation of Wireless Health Care Information Systems in Developing Countries
97-07 P. Hofstedt: Taskparallele Skelette für irregulär strukturierte Probleme in deklarativen Sprachen
97-08 D. Blostein / A. Schürr: Computing with Graphs and Graph Rewriting
97-09 C.-A. Krapp / B. Westfechtel: Feedback Handling in Dynamic Task Nets
97-10 M. Nicola / M. Jarke: Integrating Replication and Communication in Performance Models of Distributed Databases
97-13 M. Mohnen: Optimising the Memory Management of Higher-Order Functional Programs
97-14 R. Baumann: Client/Server Distribution in a Structure-Oriented Database Management System
97-15 G. H. Botorog: High-Level Parallel Programming and the Efficient Implementation of Numerical Algorithms
98-01 * Jahresbericht 1997
98-02 S. Gruner/ M. Nagel / A. Schürr: Fine-grained and Structure-oriented Integration Tools are Needed for Product Development Processes
98-03 S. Gruner: Einige Anmerkungen zur graphgrammatischen Spezifikation von Integrationswerkzeugen nach Westfechtel, Janning, Lefering und Schürr
98-04 * O. Kubitz: Mobile Robots in Dynamic Environments

98-05 M. Leucker / St. Tobies: Truth — A Verification Platform for Distributed Systems

98-07 M. Arnold / M. Erdmann / M. Glinz / P. Haumer / R. Knoll / B. Paech / K. Pohl / J. Ryser / R. Studer / K. Weidenhaupt: Survey on the Scenario Use in Twelve Selected Industrial Projects

98-08 * H. Aust: Sprachverstehen und Dialogmodellierung in natürlichsprachlichen Informationssystemen

98-09 * Th. Lehmann: Geometrische Ausrichtung medizinischer Bilder am Beispiel intraoraler Radiographien

98-10 * M. Nicola / M. Jarke: Performance Modeling of Distributed and Replicated Databases

98-11 * A. Schleicher / B. Westfechtel / D. Jäger: Modeling Dynamic Software Processes in UML

98-12 * W. Appelt / M. Jarke: Interoperable Tools for Cooperation Support using the World Wide Web

98-13 K. Indermark: Semantik rekursiver Funktionsdefinitionen mit Striktheitsinformation

99-01 * Jahresbericht 1998

99-02 * F. Huch: Verifcation of Erlang Programs using Abstract Interpretation and Model Checking — Extended Version

99-03 * R. Gallersdörfer / M. Jarke / M. Nicola: The ADR Replication Manager

99-04 M. Alpuente / M. Hanus / S. Lucas / G. Vidal: Specialization of Functional Logic Programs Based on Needed Narrowing

99-07 Th. Wilke: CTL+ is exponentially more succinct than CTL

99-08 O. Matz: Dot-Depth and Monadic Quantifier Alternation over Pictures

2000-01 * Jahresbericht 1999

2000-02 Jens Vöge / Marcin Jurdzinski: A Discrete Strategy Improvement Algorithm for Solving Parity Games

2000-04 Andreas Becks / Stefan Sklorz / Matthias Jarke: Exploring the Semantic Structure of Technical Document Collections: A Cooperative Systems Approach

2000-05 Mareike Schoop: Cooperative Document Management

2000-06 Mareike Schoop / Christoph Quix (eds.): Proceedings of the Fifth International Workshop on the Language-Action Perspective on Communication Modelling

2000-07 * Markus Mohnen / Pieter Koopman (Eds.): Proceedings of the 12th International Workshop of Functional Languages

2000-08 Thomas Arts / Thomas Noll: Verifying Generic Erlang Client-Server Implementations

2001-01 * Jahresbericht 2000

2001-02 Benedikt Bollig / Martin Leucker: Deciding LTL over Mazurkiewicz Traces

2001-03 Thierry Cachat: The power of one-letter rational languages

2001-04 Benedikt Bollig / Martin Leucker / Michael Weber: Local Parallel Model Checking for the Alternation Free mu-Calculus

2001-05 Benedikt Bollig / Martin Leucker / Thomas Noll: Regular MSC Languages

2001-06 Achim Blumensath: Prefix-Recognisable Graphs and Monadic Second-Order Logic
2001-07 Martin Grohe / Stefan Wöhrle: An Existential Locality Theorem
2001-08 Mareike Schoop / James Taylor (eds.): Proceedings of the Sixth International Workshop on the Language-Action Perspective on Communication Modelling
2001-09 Thomas Arts / Jürgen Giesl: A collection of examples for termination of term rewriting using dependency pairs
2001-10 Achim Blumensath: Axiomatising Tree-interpretable Structures
2001-11 Klaus Indermark / Thomas Noll (eds.): Kolloquium Programmiersprachen und Grundlagen der Programmierung
2002-01 * Jahresbericht 2001
2002-02 Jürgen Giesl / Aart Middeldorp: Transformation Techniques for Context-Sensitive Rewrite Systems
2002-03 Benedikt Bollig / Martin Leucker / Thomas Noll: Generalised Regular MSC Languages
2002-04 Jürgen Giesl / Aart Middeldorp: Innermost Termination of Context-Sensitive Rewriting
2002-05 Horst Lichter / Thomas von der Maßen / Thomas Weiler: Modelling Requirements and Architectures for Software Product Lines
2002-06 Henry N. Adorna: 3-Party Message Complexity is Better than 2-Party Ones for Proving Lower Bounds on the Size of Minimal Nondeterministic Finite Automata
2002-07 Jörg Dahmen: Invariant Image Object Recognition using Gaussian Mixture Densities
2002-08 Markus Mohnen: An Open Framework for Data-Flow Analysis in Java
2002-09 Markus Mohnen: Interfaces with Default Implementations in Java
2002-10 Martin Leucker: Logics for Mazurkiewicz traces
2002-11 Jürgen Giesl / Hans Zantema: Liveness in Rewriting
2003-01 * Jahresbericht 2002
2003-02 Jürgen Giesl / René Thiemann: Size-Change Termination for Term Rewriting
2003-03 Jürgen Giesl / Deepak Kapur: Deciding Inductive Validity of Equations
2003-04 Jürgen Giesl / René Thiemann / Peter Schneider-Kamp / Stephan Falke: Improving Dependency Pairs
2003-05 Christof Löding / Philipp Rohde: Solving the Sabotage Game is PSPACE-hard
2003-06 Franz Josef Och: Statistical Machine Translation: From Single-Word Models to Alignment Templates

* These reports are only available as a printed version.
Please contact biblio@informatik.rwth-aachen.de to obtain copies.