Fraunhofer Institut
Experimentelles
Software Engineering

Feature- und Entscheidungsmodell-basierte Varianteninstanziierung im PESOA-Prozess

Autoren:
Joachim Bayer
Thomas Forster
Sebastian Kiebusch
Theresa Lehner
Alexis Ocampo
Jens Weiland

PESOA
Process Family Engineering in Service-Oriented Applications

BMBF-Project

IESE-Report Nr. 128.06/D
Version 1.0
8. September 2005

Eine Publikation des Fraunhofer IESE

Das Fraunhofer IESE ist ein Institut der Fraunhofer-Gesellschaft.
Das Institut transferiert innovative Software-Entwicklungstechniken, -Methoden und -Werkzeuge in die industrielle Praxis. Es hilft Unternehmen, bedarfsgerechte Software-Kompetenzen aufzubauen und eine wettbewerbsfähige Marktposition zu erlangen.

Das Fraunhofer IESE steht unter der Leitung von
Prof. Dr. Dieter Rombach (geschäftsführend)
Prof. Dr. Peter Liggesmeyer
Fraunhofer-Platz 1
67663 Kaiserslautern

PESOA-Report No. 21/2005

PESOA is a cooperative project supported by the federal ministry of education and research (BMBF). Its aim is the design and prototypical implementation of a process family engineering platform and its application in the areas of e-business and telematics.
The project partners are:

- DaimlerChrysler Inc.
- Delta Software Technology Ltd.
- Fraunhofer IESE
- Hasso-Plattner-Institute
- Intershop Communications Inc.
- University of Leipzig

PESOA is coordinated by
Prof. Dr. Mathias Weske
Prof.-Dr.-Helmert-Str. 2-3
D-14482 Potsdam

www.pesoa.org

Abstract

Ziel von PESOA ist die Entwicklung von Methoden und Techniken für die Produktlinienentwicklung prozessorientierter Software: Dem Process Family Engineering. Dieses umfasst die Entwicklung der Produktlinie bis zur Instanziierung konkreter Produkte auf Basis variantenreicher Prozesse. Das folgende Dokument beschreibt zwei alternative Vorgehensweisen für das Process Family Engineering. Diese basieren auf der Entscheidungs- und der Feature Modellierung. Die Beschreibung orientiert sich an der Domäne Automotive. Komplettiert wird der Fachbericht durch die Analyse von Bindezeiten für das Auflösen der Variabilität in variantenreichen Prozessen.

Schlagworte: PESOA, Domain Engineering, Scoping, Decision Modeling, Feature Modeling, Domain Modeling, Application Engineering.

Inhaltsverzeichnis

1 Einleitung

1.1 Projektkontext

PESOA ist ein Gemeinschaftsprojekt finanziert durch das Bundesministerium für Bildung und Forschung (BMBF). Das Ziel des Förderprojektes ist die Entwicklung und Implementierung einer Plattform für PESOA Produktfamilien. PESOA Produktfamilien sind gekennzeichnet durch variantenreiche Software-basierte Prozesse. Die PESOA Plattform unterstützt die Instanziierung dieser variantenreichen Prozesse in den betrachteten Anwendungsdomänen E-Business und Automotive. Im Rahmen der Forschungsarbeiten werden etablierte Technologien aus dem Bereich Domain Engineering, Produktlinien Engineering und Softwaregenerierung mit Methoden aus dem Bereich Prozess Engineering integriert.

1.2 Zielsetzung

Der Fokus in PESOA liegt auf variantenreichen Software-basierten Prozessen. In der Automotive Domäne sind dies variantenreiche eingebettete Kontrollprozesse der Steuerung und Regelung. Diese beschreiben das Verhalten Software-basierter Anwendungsfunktionen. Die Entwicklung einer Produktfamilie aus variantenreichen Prozessen geschieht hierbei im Rahmen des Domain Engineering; die Instanziierung einzelner Produkte aus der Produktfamilie im Rahmen des Application Engineering. Domain Engineering im Kontext variantenreicher Prozesse und Application Engineering sind Gegenstand des Process Family Engineering.

Der folgende Fachbericht beschreibt alternative Vorgehensweisen für das Process Family Engineering am Beispiel der PESOA Anwendungsdomäne Automotive. Als Alternativen werden die Entscheidungsmodellierung auf Basis des beim Fraunhofer IESE entwickelten PuLSE Ansatzes und die Variantenkonfiguration auf Basis von FODA und dem generativen Domänenmodell vorgestellt. Beide Alternativen orientieren sich hierbei am PESOA Framework zur Modellierung variantenreicher Prozesse, wie in [BBG05] vorgestellt.

1.3 Übersicht

Kapitel 2) motiviert die Beschreibung alternativer Vorgehensweisen zum Process Family Engineering. Nachfolgend werden die einzelnen Schritte bis zur Instanziierung konkreter Produkte aus einer Prozessfamilie anhand der beiden Alternativen vorgestellt. Zunächst wird in Kapitel 3) das Domain Engineering auf Basis

von Entscheidungs- und Feature Modellen beschrieben. Die Instanziierung gültiger Produktvarianten, d.h. das Auflösen der Variabilität in variantenreichen Prozessen, wird im Rahmen des Application Engineering durchgeführt. Kapitel 4) beschreibt die Vorgehensweisen zur Instanziierung gültiger Produktvarianten anhand der beiden Alternativen. In Kapitel 5) werden Bindezeiten zur Auflösung von Variabilität in den variantenreichen Prozessen betrachtet und ein Modell für PESOA-Bindezeiträume vorgestellt.

2 Motivation

Die Automotive Domäne ist gekennzeichnet durch eine Vielzahl von Produktvarianten auf Basis elektronischer Systeme. Eingebettete Steuergeräte bilden die Kernkomponenten dieser elektronischen Systeme. Sie implementieren die Funktionslogik: Steuergeräte empfangen Sensordaten, verarbeiten diese und senden das Ergebnis der Verarbeitung an Aktuatoren. Bussysteme verbinden die Steuergeräte zu einem komplexen Netzwerk. In heutigen Oberklassefahrzeugen sind bereits mehr als 100 Steuergeräte verbaut.

Prozesse beschreiben mögliche Konsequenzen und die Ausführung zugehöriger Kontrollfunktionen. In der Benzinmotorsteuerung werden beispielsweise zur Kontrolle des Zündwinkels, zur Öffnung der Drosselklappe und zur Einspritzung eine Reihe von (Sub-)prozessen ausgeführt. Die Ausführung hängt hierbei von Sensorsignalen und dem Systemzustand ab. Die Signale werden beispielsweise von Temperatur- oder Drucksensoren ermittelt [BO02, EL03]. Unterschiede zwischen verschiedenen Benzinmotorsteuerungen, die in unterschiedlichen Motoren zum Einsatz kommen, führen zu Variabilitäten in den entsprechenden Prozessen, die sich wiederum als Variabilitäten in der embedded Software niederschlagen.

Aufgrund des steigenden Einflusses der Modell-basierten Entwicklung von Automotive embedded Software spielen Konzepte zur Modellierung von Variabilität in Software Architekturmodellen und der Auflösung dieser Variabilität eine besondere Rolle. Ein wichtiger Vertreter der Modell-basierten Softwareentwicklung ist hierbei Matlab/Simulink [MW04].

Die Konzepte zur Modellierung von Variabilität in Software Architekturmodellen und der Auflösung dieser Variabilität werden in unserem Kontext als Process-Family-Engineering bezeichnet. Process-Family-Engineering basiert hierbei auf Domain- und Application-Engineering, wie in [BBG05] beschrieben.

Der Fokus dieses Berichts liegt auf der Identifikation und Dokumentation von Gemeinsamkeiten und Unterschieden innerhalb einer Prozessfamilie im Domain Engineering, sowie deren Nutzung zur Erstellung konkreter Produkte im Application-Engineering. Die folgenden Kapitel stellen alternative Vorgehensweisen für das Process-Family-Engineering vor. Diese werden am Beispiel variantenreicher Architekturmodelle in Matlab/Simulink erläutert.

3 Domain Engineering

Ziel des Domain Engineering ist es eine Anwendungsdomäne aus Sicht der Gemeinsamkeiten und Variabilitäten von Softwareprodukten der Domäne zu beschreiben. Die identifizierten Softwareprodukte bzw. ihre ablaufenden Prozesse ergeben eine Prozessfamilie, deren gemeinsamen und variablen Eigenschaften abgebildet werden. Aus Sicht von PESOA werden diese Eigenschaften durch variantenreiche Prozesse implementiert. Diese Prozesse sind Teil der Prozessfamilien-Infrastruktur [BBG05].

Aus Sicht des Domain Engineering sind im Kontext von PESOA von besonderem Interesse

a) Scoping der Prozessfamilie und Identifizieren der gemeinsamen und variablen Eigenschaften (engl. Features) der Prozessfamilie. Im Rahmen der Domain Analyse werden die Features mittels Feature Modellen modelliert.

b) Modellieren einer gemeinsamen Prozessfamilien-Architektur. Diese wird im Rahmen des Domain Designs ausgearbeitet. Das Ergebnis sind variantenreiche Prozesse, deren Variationspunkte und das Configuration Model, welches das Konfigurationswissen zwischen den Features und den variantenreichen Prozessen der Prozessfamilie beinhaltet.

c) Implementieren der Prozessfamilie durch domänen-spezifische Komponenten und Generatoren.

Im Rahmen des Application Engineerings werden aus den Featuren, den variantenreichen Prozessen und dem Konfigurationswissen gültige Produktvarianten abgeleitet (siehe Kapitel 4).

Die folgenden Abschnitte beschreiben zwei alternative Vorgehensweisen innerhalb des Domain Engineering. Zur Modellierung variantenreicher Prozesse sei an dieser Stelle auf [PS+05] verwiesen.

In Abschnitt 3.1 wird der Domain Engineering Prozess, der vom Fraunhofer IESE auf Basis von PuLSE verfolgt wird, definiert. Anschließend wird in Abschnitt 3.2 die in der DaimlerChrysler Forschung verfolgte Vorgehensweise auf Basis von FODA und dem generativen Domänenmodell vorgestellt.

3.1 Entscheidungsmodell-basiertes Domain Engineering

Der in [BBG05] definierte PESOA Prozess ist unterteilt in Scoping, Domain und Application Engineering. In diesem Abschnitt werden Vorgehensweisen für die Aktivitäten „Domain Scoping", „Model Features" und „Model Configurations" beschrieben. Diese Vorgehensweisen basieren auf PuLSE, der Produktlinienentwicklungs-Methode des Fraunhofer IESE.

„Domain Scoping" hat zum Ziel die Prozessfamilie und ihre Grenzen zu definieren [BBG05]. Ausgangspunkt für das Scoping ist eine Menge von existierenden oder geplanten Produkten, die mit Hilfe einer Prozessfamilie erstellt werden sollen. Aus dieser Menge werden diejenigen Produkte ausgewählt, die wirtschaftlich rentabel als Prozessfamilie entwickelt werden können, da sie einen hohen Grad an Wiederverwendung versprechen. Ein zentraler Punkt sind hierbei auftretende Gemeinsamkeiten und Unterschiede zwischen den Produkten. Die ausgewählten Produkte werden mit den Eigenschaften (so genannten Features) in Beziehung gesetzt, die die Produkte bereitstellen.

Die beiden Domain Engineering Aktivitäten „Model Features" und „Model Configurations" werden durch die in PuLSE definierte Entscheidungsmodellierung unterstützt. Die Entscheidungsmodellierung dokumentiert und wartet die Beziehung zwischen Variationspunkten und den Eigenschaften einer Prozessfamilie [BBG05]. Ausgangspunkt der Entscheidungsmodellierung sind die variantenreichen Prozesse einer Prozessfamilien-Infrastruktur mit ihren Variationspunkten und die im Domain Scoping definierten Features. Das resultierende Entscheidungsmodell beinhaltet zum einen die Abhängigkeiten zwischen den Produkteigenschaften und zum anderen das Konfigurationswissen bezüglich den variantenreichen Prozessen. Die Konfigurationsmöglichkeiten der Variationspunkte sind an die vorhandenen Features gebunden.

3.1.1 PuLSE Scoping

Zur Definition einer Produktfamilie bzw. Prozessfamilie und zur Bewertung deren Wiederverwendungspotential verfolgt das Fraunhofer IESE die in PuLSE™ Eco [Sch03] definierte Methode.

Ausgangspunkt für PuLSE Eco sind Dokumentation von existierenden, geplanten und zukünftigen Produkten. Das Ergebnis der Scoping Aktivität ist eine so genannte Produktkarte, die die einzelnen Produkte der Produktfamilie mit ihren entsprechenden Eigenschaften auflistet, und eine Bewertung des Produktlinienpotenzials.

Die Erstellung der Produktkarte, das so genannte Product Line Mapping (siehe Abbildung 1) erfolgt in Zusammenarbeit mit verschiedenen Domänenexperten

und Produktexperten der Produktfamilie. Um das Potenzial der einzelnen Domänen und der gesamte Produktlinien zu bewerten, werden die entsprechenden Experten interviewt.

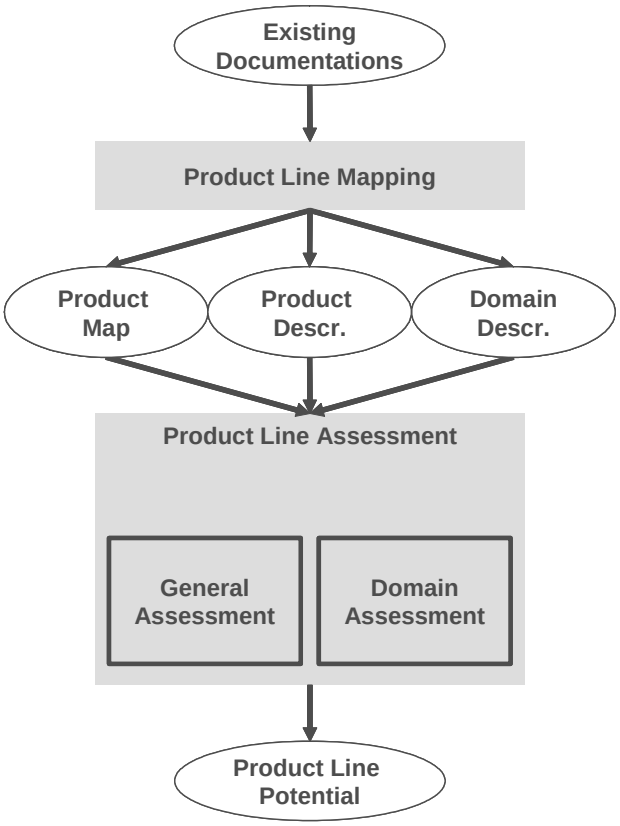

Abbildung 1: Scoping

Die folgenden Abschnitte stellen die Aktivitäten „Product Line Mapping“ und „Product Line Assessment“ detaillierter vor.

3.1.1.1 Product Line Mapping

Das Ziel der Product Line Mapping Aktivität ist es die Produkte, die Produkteigenschaften und die Domänen der Produktfamilie zu identifizieren und zu dokumentieren.

Als Input werden zum einen existierende Dokumentationen der möglichen Produkte und zum anderen vorhandenes Expertenwissen verwendet. Das Ergebnis wie oben erwähnt ist die so genannte Produktkarte, die die Produkte, deren Eigenschaften und die Domänen beinhaltet.

Für die Durchführung des Product Line Mapping bietet sich ein Workshop an, an dem sowohl die Domänen- und Produktexperten als auch das Management der Produktfamilie teilnehmen.

Der erste Schritt innerhalb des Product Line Mappings, wie in Abbildung 2 beschrieben, hat die Identifikation von existierenden, geplanten und zukünftigen Produkten zum Ziel. Existierende Dokumente bieten erste Information über Produkte und deren Eigenschaften. Diese Informationen werden innerhalb des „Product Identification" Schrittes zusammen mit dem Workshopteilnehmern zu Produktbeschreibungen konkretisiert.

Das Ziel von „Feature Identification" ist es die Eigenschaften innerhalb der Produktfamilie zu definieren und aufzulisten. Ausgangspunkt ist die Produktbeschreibung der identifizierten Produkte. Das Ergebnis ist eine Liste von Eigenschaften. Die Eigenschaften werden zum einen aus der Produktbeschreibung entnommen, zum anderen werden weitere Eigenschaften von den Domänen-, Produktexperten und Managern, zum Beispiels mittels eines Brainstormings zusammengetragen.

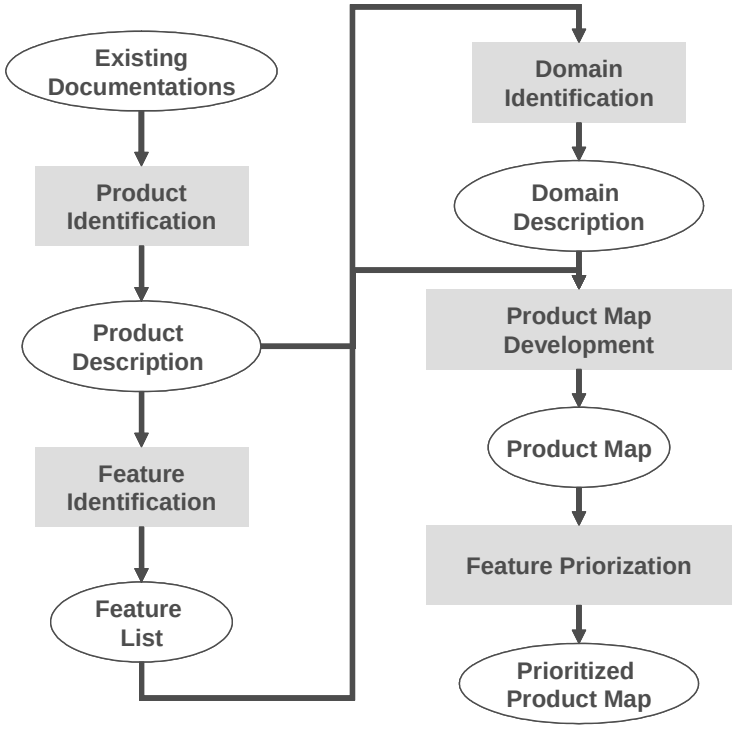

Abbildung 2: Product Line Mapping

In der Aktivität „Domain Identification" werden die einzelnen Domänen der Produktfamilie identifiziert und dokumentiert. Als Input für diese Aktivität dient die Liste der Eigenschaften. Das Ergebnis sind Beschreibungen der identifizier-

ten Domänen. Die Liste der Eigenschaften wird mit Hilfe der Domänen- und Produktexperten gruppiert. Aus den Gruppierungen ergeben sich dann die einzelnen Domänen der Produktfamilie.

Das Ziel des nächsten Schrittes ist die Erstellung der Produktkarte. Die einzelnen identifizierten Produkte, die Liste der Eigenschaften und die identifizierten Domänen sind Input für diese Aktivität. Das Resultat ist die Produktkarte, die eine Zuordnung der Eigenschaften zu den Produkten und zu den Domänen darstellt. Dabei wird die Gruppierung der Eigenschaften in Domänen in die Produktkarte übertragen. Die Zuordnung der Eigenschaften zu Produkten basiert zum einen auf den Produktbeschreibungen, zum anderen werden die Domänen- und Produktexperten zu Rate gezogen.

Aus Sicht des Marketings bzw. der Entwicklung haben die identifizierten Eigenschaften und Domänen unterschiedlichen Stellenwert. „Feature Priorization" hat zum Ziel diesen Stellenwert zu identifizieren. Diese Aktivität wird auf Basis der Produktkarte durchgeführt. Die Ergebnisse der Priorisierung werden in zusätzlichen Spalten der Produktkarte dokumentiert. Die Priorisierung erfolgt nach einem Schema wie in Tabelle 1 dargestellt. Es existieren zwei Bewertungskriterien. Für das Marketing ist es die Kundenzufriedenheit bzw. - Unzufriedenheit; für die Entwicklung der Implementierungsaufwand bzw. das damit verbundene Risiko. Anhand dieser Bewertungskriterien werden die Eigenschaften mit Hilfe des angegebenen Punktesystems von Marketing und Entwicklungsexperten bewertet.

Tabelle 1: Skala der Priorisierung

	Marketing	Entwicklung/Technik
1. Bewertung	Wie sehr hält ein Fehlen dieses Features vom Kauf ab?	Welchen Aufwand kostet die Implementierung dieses Feature?
2.Bewertung	Wie wichtig ist das Feature für die Kundenzufriedenheit / USP?	Welches Risiko trägt die Implementierung dieses Feature?
Skala: 0 Punkte	überhaupt nicht	
1 Punkt	etwas	
2 Punkte	viel/wichtig	
3 Punkte	sehr viel/wichtig; essentiell	

Abschließend soll ein Beispiel aus der Automotive Domäne die Produktkarte, das Resultat von „Domain Scoping", veranschaulichen. Der in Abbildung 3 ausgewählte Ausschnitt beinhaltet die Domäne „Benzinmotorsteuerung" mit ihren Sub-Domänen „Automobilsysteme" und „Motor". Für jede (Sub-) Domäne wurden die Feature und ihre möglichen Werte spezifiziert. Die Produktkarte definiert auch welches existierende, geplante oder hypothetische Produkt welche Eigenschaften mit welchen Werten hat.

Product Map				existing				planned			hypothetical	
Domain	Nr	Feature	Values	P1	P2			P3	P4		P5	
	1	Schubabschaltung			X							
	2	Leerlaufstrategie	Drosselklappensteuerung(D), Leerlaufregelungsventilsteuerung (L), Zündwinkelveränderung (Z)	(L)	(Z)							
	3	Geschwindigkeitsbegrenzung										
	4	Abgasrückführsystem			X							
	5	Fahrmodus	sportlich, eco, komfortable	s	k							
Benzinmotorsteuerung	6	Kraftstoffgemischstrategie	stöchiometrisch, magermix	s	s							
	7	Drosselklappe		x	x							
	8	Klimaanlage	Typ1, Typ2									
	9	Turbolader		x								
	10	Tempomat			x							
	11	Antischlupfregelung		x								
	12	Katalysator		x								
Automobilsysteme	13	Wegfahrsperre	electronisch		x							
	14	Typ	M1, M2	M1	M2							
	15	Klopfsensor		x								
Motor	16	Einspritzung	Saugrohreinspritzung, Direkteinspritzung	s	x							

Abbildung 3: Beispiel-Produktkarte der Domäne „Bezinmotorsteuerung“

3.1.1.2 Product Line Assessment

Das Ziel von Product Line Assessment ist das Produktfamilienpotenzial der identifizierten Domänen und der gesamten Produktfamilie zu bewerten.

Ausgangspunkt dafür sind die Produktkarte, die Domänenbeschreibungen und die Produktbeschreibungen, die in der vorangegangenen Product Line Mapping Aktivität erstellt wurden. Das Ergebnis ist die Bewertung der Domänen bzw. Produktfamilie hinsichtlich ihres Potentials auf einer Skala von „teilweise“ bis „fortgeschritten“.

Für die Bewertung des Produktfamilienpotentials werden Interviews mit Domänenexperten, Produktexperten und dem Manager der Produktfamilie durchgeführt.

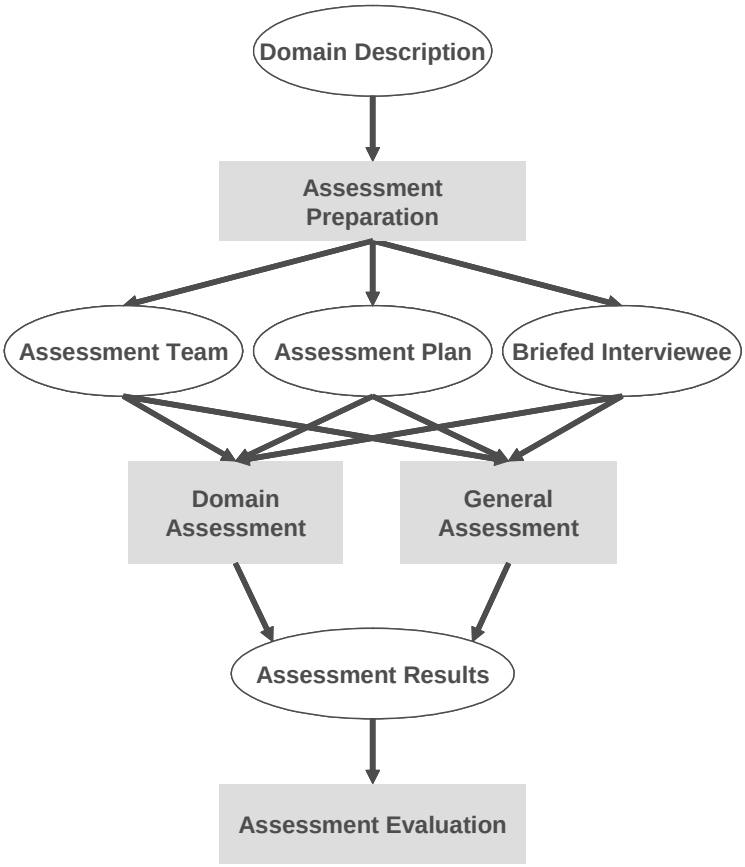

Abbildung 4: Product Line Assessment

Der erste Schritt der „Product Line Assessment" Aktivität (siehe Abbildung 4) sind Vorbereitungsaktivitäten. Sie beinhalten die Zusammenstellung der Interviewpartner, das Erstellen des Zeitplans der einzelnen Interviewpartner und die Einweisung der Partner in das Interviewvorgehen. Die Domänenbeschreibungen dienen als Input für die Vorbereitungen. Das Ergebnis ist ein für die Bewertung des Produktfamilienpotentials perfekt zusammengestellte Gruppe von Interviewpartnern. Außerdem wird mit ihnen das Interviewvorgehen besprochen und ein Zeitplan für die Interviews festgelegt.

Anschließend werden zwei verschiedene Arten von Interviews zur Bewertung des Produktlinienpotenzials durchgeführt. Das „Domain Assessment" hat zum Ziel das Produktfamilienpotential der einzelnen identifizierten Domänen zu bewerten. Ausgangspunkt ist ein Interviewtermin mit einem oder mehreren Interviewpartnern, die Experten in der Domäne bzw. für das Produkt sind und auf das Interview vorbereitet wurden, und ein Termin mit den Interviewpartnern. Das Ergebnis sind die Informationen, Einschätzungen, und Erfahrungen der Interviewpartner für die in Tabelle 2 beschriebenen Bereiche.

Tabelle 2: Assessment Dimensions

Bereiche	Fragen
Ressourcen	Welche Ressourcen sind der Organisation verfügbar um die Produktlinienentwicklung aufzusetzen?
Organisation	In welcher Beziehung steht die Domäne zur Organisationsstruktur und ermöglicht sie Wiederverwendung?
Externes Marktpotenzial	Wie stark bestimmt die Domäne das Marktpotential des Produktes?
Internes Marktpotenzial	Welche interne Strategie wird innerhalb der Organisation bezüglich der Domäne verfolgt?
Reife	How mature is the domain, i.e., how well understood is the domain and how well organized are the concepts in the domain?
Stabilität	Wie stabil und standardisiert sind Konzepte und Verhalten in der Domäne (z.B. Protokolle)?
Gemeinsamkeiten und Unterschiede	Wie verbreitet sind Gemeinsamkeiten in der Domäne und in welchem Ausmaß variierende Systeme in der Domäne?
Kopplung und Kohäsion	Hat die Domäne eine enge Kopplung mit anderen Domänen? Ist die Funktionalität in der Domäne wirklich geschlossen (d.h., ist es wirklich eine Domäne)?
Existierende Assets	Existieren bereits allgemein bzw. generische Assets in der Domäne

Das „General Assessment“ hat zum Ziel das Produktfamilienpotential der gesamten Produktfamilie zu bewerten. Ausgangspunkt für dieses Interview ist die Einführung des Produktlinienmanagers in das Interviewvorgehen bzw. –ziel und eine Terminvereinbarung. Das Ergebnis sind die Informationen, Einschätzungen, und Erfahrungen des Produktlinienmanagers für die in Tabelle 2 beschriebenen Bereiche.

Die „Assessment Evaluation“ hat zum Ziel das Produktfamilienpotenzial der einzelnen Domänen und der gesamten Produktfamilie zu identifizieren. Input für diese Evaluierung sind alle Informationen, Einschätzungen und Erfahrungen, die während den Interviews erfragt wurden. Das Ergebnis ist eine Bewertung des Produktfamilienpotenzials der einzelnen Domänen und der gesamten Produktfamilie, dabei werden die in der Produktfamilie bzw. Domäne identifizierten Vorteile, Nachteile, Risiken beschrieben und außerdem Vorschläge für Verbesserungsaktivitäten gegeben.

3.1.2 Entscheidungsmodellierung

Das Ziel der Entscheidungsmodellierung innerhalb PESOA ist die Identifizierung und Dokumentation der Beziehungen zwischen Variationspunkten und Eigenschaften der Produktfamilie. Input für die Entscheidungsmodellierung sind zum einen die variantenreiche Prozesse, die in der Aktivität „Design Processes" erstellt wurden (siehe [BBG05]), zum anderen die in Domain Scoping erstellte Produktkarte, die alle Eigenschaften der Prozessfamilie enthält. Das Ergebnis ist das Entscheidungsmodell, das zum einen das Wissen über Featureabhängigkeiten und zum anderen das Konfigurationwissen von Variationspunkten dokumentiert und wartet.

Wie aus Abbildung 5 hervorgeht, beinhaltet die Entscheidungsmodellierung eine iterative Sequenz von den Aktivitäten „Decision Specification" und „Constraint Identification". Ziel der Iteration ist es die Abhängigkeiten zwischen den einzelnen Variationspunkten bzw. Eigenschaften zu fassen, und damit das Application Engineering zu vereinfachen (siehe Abschnitt 4.1). In der folgenden Beschreibung werden die Iterationen getrennt behandelt.

Der erste Schritt hat zum Ziel die Auflösungsvarianten eines Variationspunktes abhängig von den Featuren zu definieren und als so genannte Entscheidungen zu dokumentieren.

Input für die Aktivität in der ersten Iteration sind einerseits die variantenreichen Prozesse, die die Variationspunkte beinhalten, andererseits die Produktkarte, die die Feature der Produktfamilie enthält. Das Ergebnis sind Entscheidungsspezifikationen für jeden einzelnen Variationspunkt. In der ersten Iteration von „Decision Specification" werden Entscheidungen definiert, die Variationspunkte mit Eigenschaften der Prozessfamilie in Verbindung setzt. Die Entscheidungen sollen die Auflösung der Variationspunkte unterstützen, in dem sie spezifizieren welche Eigenschaft(en) die Auflösung des Variationspunkt beeinflusst und wie sich die Auflösung auf die variantenreichen Prozesse auswirkt, das so genannte Konfigurationswissen.

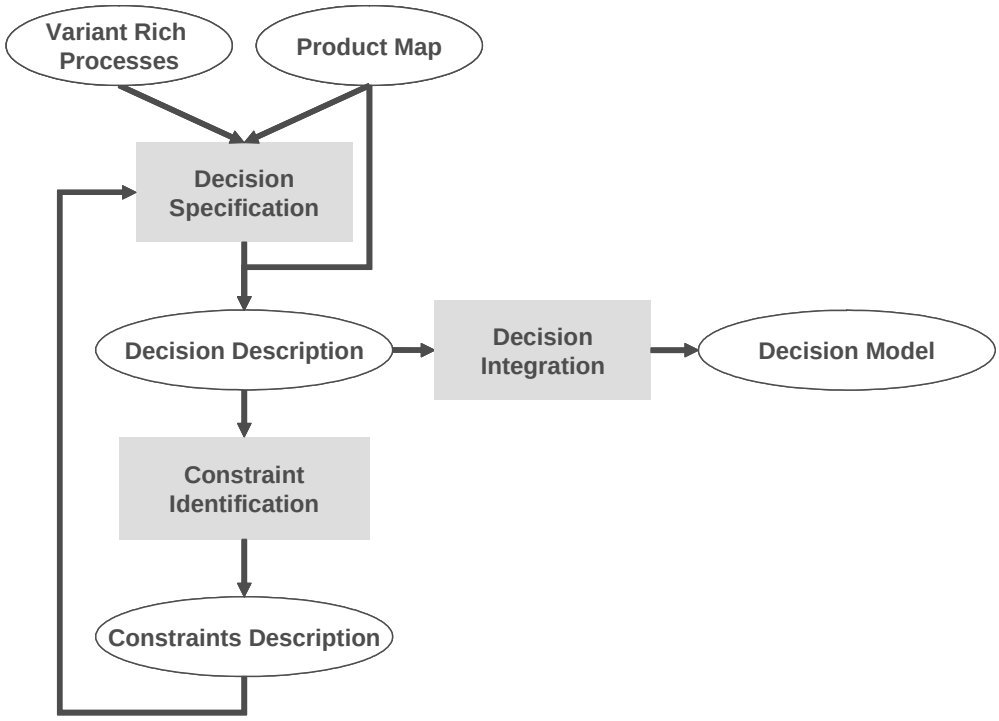

Abbildung 5: Decision Modeling

Das Ziel von „Constraints Identification" ist es Abhängigkeiten bzw. Bedingungen zwischen Entscheidungen zu identifizieren und zu dokumentieren. Als Input dienen dafür die Spezifikationen der Entscheidungen. Das Ergebnis sind dokumentierte Abhängigkeiten zwischen Entscheidungen. Abhängigkeiten definieren sich zum einen aus den Abhängigkeiten der Feature und zum anderen aus Überschneidungen der Entscheidungen bezüglich den Featureverweisen. Die identifizierten Featureabhängigkeiten sind mit dem Produkt „Feature Model" im PESOA Prozess vergleichbar.

Die identifizierten Abhängigkeiten dienen als Input für die „Decision Specification" Aktivität in der nächsten Iteration. Das Ergebnis dieser Aktivität sind komplexere Entscheidungen, die die Abhängigkeiten zwischen einfacheren Entscheidungen spezifizieren.

Die Iteration über „Decision Specifiation" und „Constaints Identification" resultiert in einer Hierarchie von Entscheidungen. Das Ziel von „Decision Integration" ist diese Hierarchie und jede einzelne Entscheidung zu dokumentieren und zu warten. Der Input für die Aktivität sind die Spezifikationen der Entscheidungen. Das Ergebnis ist ein Entscheidungsmodell, das die Hierarchie der Entscheidungen beinhaltet und damit die Auflösung der Variationspunkte entsprechend der ausgewählten Eigenschaft ermöglicht.

Der Entscheidungsmodellierungsprozess wird im Folgenden anhand des Beispiels aus der Automotive Domäne illustriert. Die einzelnen Aktivitäten des Prozesses wurden mit Hilfe des Decision Modelling Tool, welches am Fraunhofer IESE entwickelt wird, ausgeführt. Zuerst wurden Entscheidungen definiert, die die identifizierten Variationspunkte mit den Eigenschaften definiert in der Produktkarte verbinden. Für unser Benzinmotorbeispiel wurden die Variationspunkte „MotorTyp“, „Einspritzung“ und „Klopfsensor“ mit Fragen verknüpft, die eine Verbindung zu den Eigenschaften in der Produktkarte (siehe Abbildung 3) darstellen (siehe Abbildung 6).

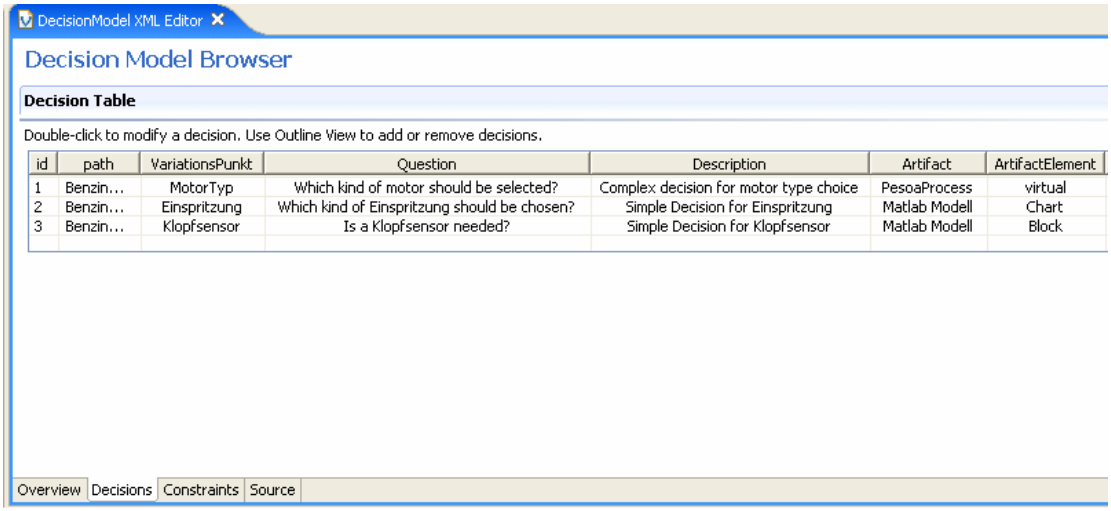

Abbildung 6: Beispiel Variationspunkte und Decisions in der „Bezinmotorsteuerung“

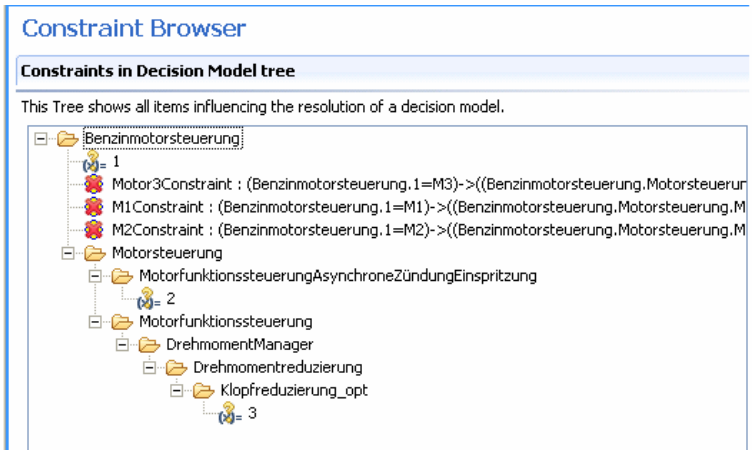

Abbildung 7: Constraint Browser für die “Benzinmotorsteuerung”

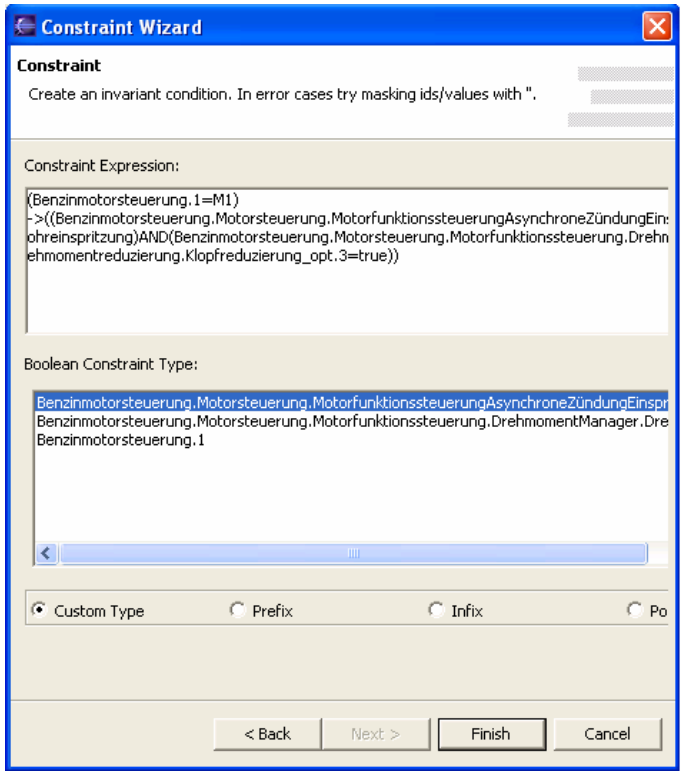

Abbildung 8: Beispiel Constraint Definition

Anschließend wurden Constraints zwischen den Variationspunkt Motortyp und seinen Ausprägungen „M1“, „M2“ und „M3“ identifiziert und im Constraint Browser, wie in Abbildung 6 beschrieben, dokumentiert. Abbildung 8 illustriert den Constraint, dass der Motortype M1 die Variante „Saugrohreinspritzung“ und den „Klopfsensor“ bedingt.

3.2 Feature-basiertes Domain Engineering

Die im folgenden Abschnitt vorgestellte Vorgehensweise beschreibt den Domain Engineering Prozess auf Basis von FODA und dem generativen Domänenmodell. Das generative Domänenmodell trennt anwendungsorientierte Konzepte, wie die Feature Modellierung, von Konzepten der Implementierung, d.h. den variantenreichen Prozessen.

3.2.1 FODA-basiertes Scoping

Wie beim Scoping in PuLSE, verfolgt das Scoping auch hier das Ziel relevante Funktionsbereiche für eine Wiederverwendung zu identifizieren und abzugrenzen. Ziel ist es ein konsistentes Bild der Domäne zu schaffen: Eine genaue Be-

schreibung der Domäne, um deren Umfang abzugrenzen und diese zu charakterisieren, unter anderem durch Beispiele anhand bereits existierender Systeme. Insbesondere ergeben sich im Rahmen des Scopings die folgenden Fragen:

- Wie grenzt sich die Domäne von Nachbardomänen ab?
- Was gehört zur Domäne? Was gehört nicht zur Domäne?
- Welcher Art sind die Beziehungen zu Konzepten außerhalb der Domäne?

Die möglichen Informationsquellen, die für das Scoping herangezogen werden können, sind weit reichend. Hierbei handelt es sich im Wesentlichen um:

- Technische Literatur: Lehrbücher, wissenschaftliche Journale, Handbücher
- Existierende Systeme, Ausführbare Systeme, Source Code, Architekturdokumente, Anforderungsspezifikationen, Benutzerhandbücher
- Expertenrat: Befragung von Domänenexperten und Entwickler bereits existierender Systeme in der betrachteten Domäne

In einer Studie haben wir eine abstrakte Benzinmotorsteuerung betrachtet. Im Rahmen des Scoping-Prozesses haben sich unsere Arbeiten an den technischen Aspketen in der Domäne Benzinmotorsteuerung orientiert. Auf die Stakeholder-Analyse – als Teil des Domain Scopings – haben wir verzichtet.

Hauptaufgabe der Benzinmotorsteuerung ist, das vom Motor erzeugte Drehmoment einzustellen. Dazu werden in den verschiedenen Teilsystemen der Motorsteuerung alle drehmomentbeeinflussenden Größen gesteuert – Füllungssteuerung, Gemischbildung und Zündung. In unserem Kontext werden der Benzinmotorsteuerung die folgenden Funktionsbereiche zugeordnet (siehe auch MED-Motronic aus [EL03]):

- Grundabstimmung über Kennfeld,
- Nachstart-, Volllast-, Beschleunigungsanreicherung,
- Startsteuerung,
- Schubabschaltung,
- Drehzahlbegrenzung,
- Lambda-Regelung,
- Leerlaufdrehzahlregelung,
- Abgasrückführung,
- Drehmomentregelung,
- E-Gas-Funktion,
- Fahrgeschwindigkeitsregelung,
- Lastwechselregelung,

- Wahl der Betriebsart,
- On Board Diagnose.

Die Domäne sollte so gewählt werden, dass viele Gemeinsamkeiten zwischen den Produktvarianten existieren. Dies stellt ein hohes Maß an Wiederverwendung sicher. Ein hoher Grad an Variationen in der Domäne lässt auf wenig Gemeinsamkeiten oder auf eine Domäne schließen, die aus verschiedenen Sub-Domänen besteht. In diesem Fall sollte die Domäne „re-scoped", d.h. der Umfang der Domäne neu betrachtet werden. Im Falle, dass viele große und komplexe Systeme in der betrachteten Domäne existieren, sollte berücksichtigt werden, dass eine detaillierte Analyse beachtliche Ressourcen – Zeit und Personal – in Anspruch nimmt [Ka03].

Über Kontextmodelle wird die Domäne, ihre Umgebung und die gegenseitige Wechselwirkung mit der Umgebung dargestellt. Kontextmodelle beschreiben die Position der Domäne relativ zu übergeordneten, untergeordneten und gleich geordneten Domänen.

Im Kontext-Strukturdiagramm werden die Domänen eingeordnet und dargestellt, die eine unmittelbare Beziehung zur betrachteten Domäne besitzen. Ergänzt wird die Beschreibung dieser Domänen durch konkrete Konzepte, die direkt die betrachtete Domäne beeinflussen oder von ihr beeinflusst werden. Abbildung 9 stellt das Kontext-Strukturdiagramm der Domäne Benzinmotorsteuerung dar. Das Kontext-Strukturdiagramm stellt die Position der Domäne Benzinmotorsteuerung relativ zu relevanten übergeordneten, untergeordneten und gleich geordneten Domänen dar. Die Benzinmotorsteuerung ist Teil der Domäne Antriebsstrangregelung. Untergeordnete Domänen sind Subdomänen der Benzinmotorsteuerung, wie Kommunikation, Gerätetreiber und Betriebssystem.

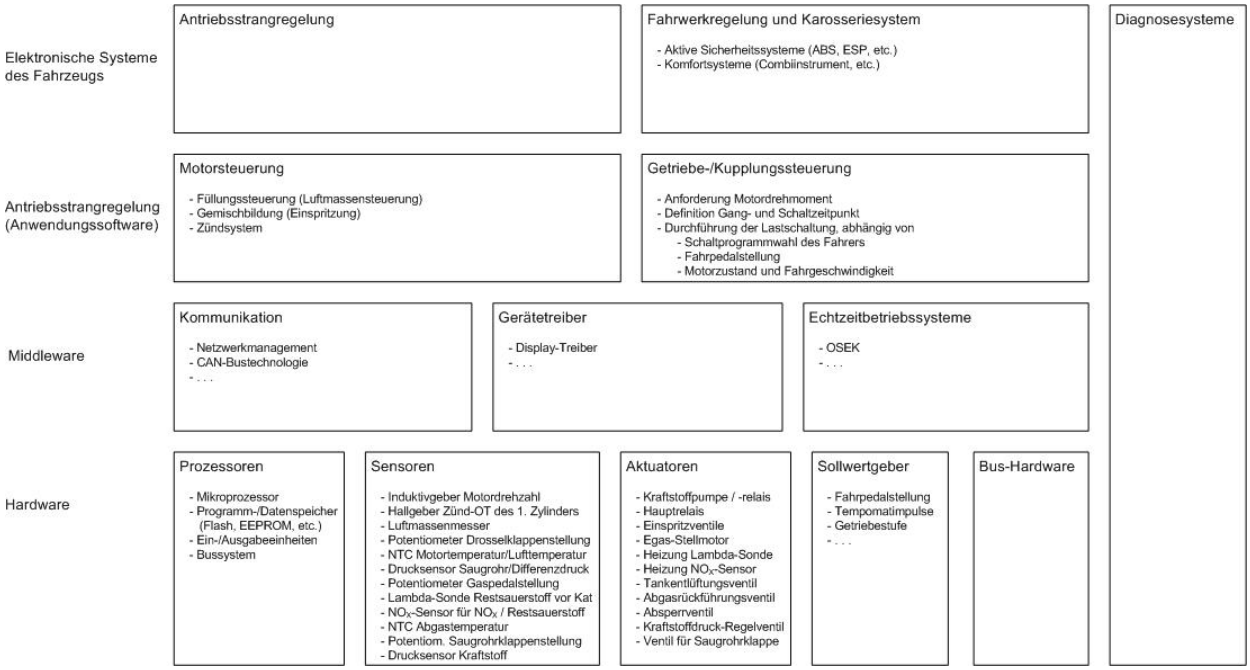

Abbildung 9: Kontext-Strukturdiagramm der Domäne Benzinmotorsteuerung

Auch wenn das Kontext-Strukturdiagramm Variabilitäten enthalten kann – z.B. über variable Konzepte angrenzender Domänen – ist es zur vollständigen Abbildung der Variabilität nicht geeignet. Es dient lediglich der Domänenübersicht, nicht aber einer konkreten Modellierung der Variabilität (allenfalls einer Grobübersicht). So ist beispielsweise im Kontext-Struturdiagramm der Benzinmotorsteuerung nicht explizit die Variabilität dargestellt, die sich durch unterschiedliche Typen eines Sensors oder Aktuators ergeben kann.

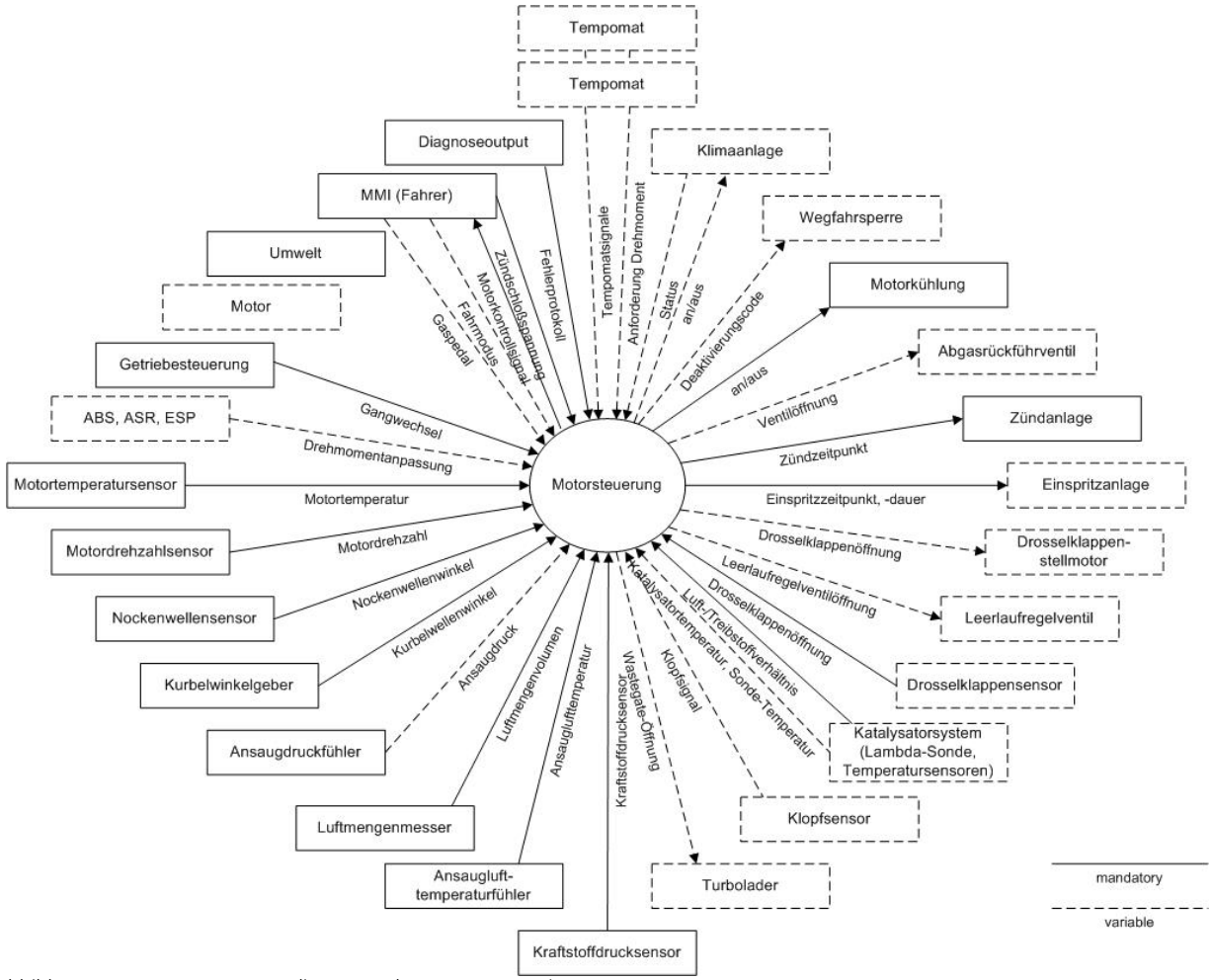

Abbildung 10: Kontextdiagramm der Domäne Benzinmotorsteuerung

Im Kontextdiagramm werden wichtige Konzepte außerhalb der Domäne, die die Domäne beeinflussen oder von ihr beeinflusst werden, betrachtet – z.B. Benutzer oder physikalisches Umfeld. Zudem werden Datenflüsse von und zur Domäne dargestellt. Variationen können gestrichelt dargestellt werden. Obligatorische externe Konzepte und Datenflüsse werden mit einer durchgezogenen Linie gekennzeichnet.

stellt das Kontextdiagramm der Domäne Benzinmotorsteuerung dar. Sensoren und Aktuatoren sind der Übersichtlichkeit wegen zu jeweils einem externen Block zusammengefasst. Die Kommunikation zwischen Motor und Motorsteuerung geschieht über die Sensoren und Aktuatoren, so dass der Motor im Beispiel selbst nicht explizit erfasst werden muss. Dennoch wird der Motor hier

abgebildet, da dieser die Motorensteuerung über Parameter (z.B. Kennlinien und -felder) beeinflusst. Dies gilt auch für den Block Umwelt.

3.2.2 Variabilitätsmanagement

Das im Folgenden dargestellte Variabilitätsmanagement orientiert sich am generativen Domänenmodell [CE00]. Dieses trennt Anwendungsorientierte Konzepte im Problemraum von Konzepten der Implementierung im Lösungsraum. Das Konfigurationswissen verbindet Problem- und Lösungsraum und beinhaltet das notwendige Regelwerk zur Instanziierung gültiger Produktvarianten. Im PESOA-Prozess beinhaltet der Problemraum das Feature Modell und der Lösungsraum die variantenreichen Prozesse und deren Variationspunkte. Das Konfigurationswissen wird durch das Konfigurationsmodell realisiert.

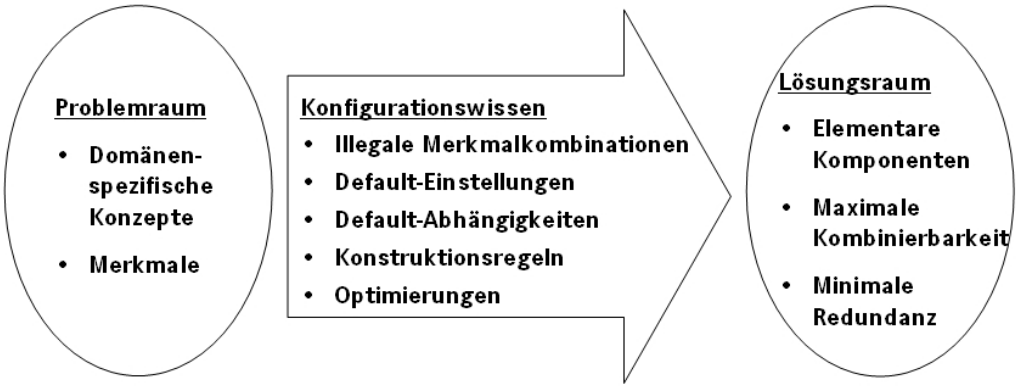

Abbildung 11: Elemente des generativen Domänenmodells aus [CE00]

Ein Werkzeug, welches das generative Domänenmodell unterstützt, ist pure::variants von pure-systems [PS04], welches mit dem Feature Modell und dem Familienmodell diese Trennung umsetzt. Im Folgenden werden die Konzepte zur Modellierung des Problemraums über das Feature Modell, die Modellierung des Lösungsraums über das Familienmodell und das Management des Konfigurationswissens am Beispiel von pure::variants beschrieben.

Der Prozess zur Auflösung der Variabilität, um gültige Produktvarianten zu instanziieren, wird in Kapitel 4.2 vorgestellt. Basis für die Auflösung der Variabilität sind geeignete Konzepte zur Modellierung von Variabilität in Matlab/ Simulink-Architekturmodellen. Diese Konzepte werden in [PS+05] vorgestellt.

3.2.2.1 Modellierung von Featuren

Im Feature Modell werden die Feature[1] der untersuchten Domäne verwaltet und hierarchisch strukturiert. Die Feature repräsentieren hierbei Konzepte der untersuchten Domäne. Ähnliche Feature, die dasselbe Konzept repräsentieren werden zusammengefasst. Hierbei wird das Konzept als abstraktes Feature modelliert und die Spezialisierungen als Child-Feature. Das Feature Modell stellt hierbei keine Hierarchie von Funktionsaufrufen dar. Es beschreibt lediglich welche Arten von Variabilität in der Domäne existieren. Das Ergebnis ist ein umfassendes Modell der gemeinsamen und variablen Feature sowie deren Abhängigkeiten zwischen den Produktvarianten.

Jedes Feature ist gekennzeichnet durch einen eindeutigen Identifier und zusätzliche Informationen, wie Featuretyp – Mandatory, Alternative, Optional, oder Or –, eine Beschreibung des Features, Attribut-Werte-Paare, Bindezeiten und -modi, etc. [CE00]. Attribut-Werte-Paare eignen sich an den Stellen, an denen Konzepte nicht weiter verfeinert werden und die Variabilität zu keiner weiteren Auswirkung auf die Software führt. Ein Beispiel hierfür sind die Motoreneigenschaften, ausgedrückt durch Attribut-Werte-Paare, wie Anzahl Zylinder, maximale Geschwindigkeit, Drehmomentkennfeld, etc.

[1] Wir verwenden den Begriff Feature für Eigenschaften bzw. Charakteristika eines Systems, die für User, Entwickler oder andere Entitäten, die mit dem System interagieren, von Bedeutung sind oder diese direkt beeinflussen. In unserem Kontext umfaßt dies sowohl funktionale als auch non-funktionale Aspekte.

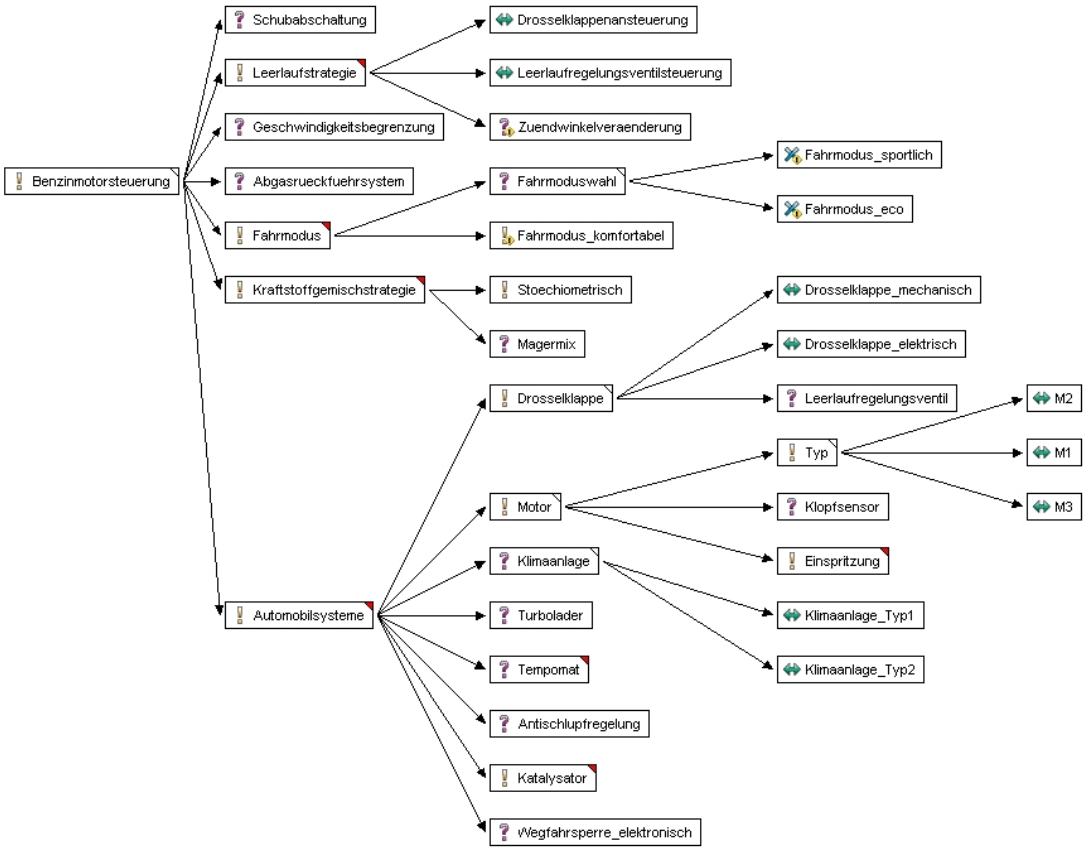

Abbildung 12: Beispiel eines Feature Diagramms einer Benzinmotorsteuerung

Als Einstieg in die Feature Modellierung eignen sich Startersets. Diese unterstützen die Identifizierung von Feature durch das Bestimmen von Featurekategorien. Sie stellen unterschiedliche Perspektiven zur Modellierung von Konzepten der untersuchten Domäne dar. Die Erstellung des Startersets für die Benzinmotorsteuerung wurde geleitet durch die Fragestellung nach typischen Quellen von Variabilität für embedded Steuerungsprozesse. Als Teil des Startersets der Benzinmotorsteuerung wurden beispielsweise die folgenden Aspekte identifiziert:

- Speicherformate: Kennfelder, Kennlinien, Parameter
- Sensoren und Aktuatoren
- Charakteristika von embedded Steuerungsprozessen, wie Echtzeitanforderungen, Verteilung/Vernetzung, etc.
- . . .

Es existieren verschiedene Möglichkeiten Feature zu gruppieren [CE00, Ka03]. Nach [CE00] werden Feature in die folgenden Kategorien eingeteilt: *Context*

features beschreiben im wesentlichen nicht-funktionale Charakteristiken eines Systems. *Representation features* sind Feature, die beschreiben, wie Informationen für den Nutzer oder ein anderes System aufbereitet werden. *Operational features* beschreiben die funktionalen Charakteristiken eines Systems.

3.2.2.2 Verwaltung von Implementierungskomponenten

Im Familienmodell werden die Implementierungskomponenten, in unserem Fall die variantenreichen Prozesse, verwaltet und – vergleichbar den Feature im Feature Modell – hierarchisch strukturiert abgelegt. Implementierungskomponenten können neben Source- und Objektdateien auch Dokumente, Variablen, etc. zur Erstellung von Produktinstanzen sein.

Im Rahmen der Modellierung der Benzinmotorsteuerung umfaßt das Familienmodell das Simulink-Architekturmodell, sowie eine Anzahl von Artefakten, denen Konfigurationswissen zugeordnet ist[2]. Lediglich die Artefakte werden in pure::variants verwaltet. Das Simulink-Modell der Benzinmotorsteuerung wird unter Matlab verwaltet. Im Rahmen der Instanziierung gültiger Produktvarianten werden die Artefakte gemäß der selektierten Produktvariante konfiguriert (siehe Kapitel 4.2).

3.2.2.3 Management des Konfigurationswissens

Über das Konfigurationswissen werden gültige Softwareprodukte einer Familie spezifiziert. Im Rahmen der Modellierung variantenreicher Simulink-Modelle war hierbei das Ziel das Konfigurationswissen dediziert an einer Stelle zu verwalten. Dies wurde über die Verwendung Werkzeug-intrinsischer Konfigurationssprachen – *Configuration Domain Specific Languages* (Configuration DSLs) und *Implementation Component Configuration Languages* (ICCLs) – in pure::variants realisiert. Diese können Domänen-übergreifend verwendet werden.

Über Configuration DSLs lassen sich im Feature Modell gültige Feature Konfigurationen spezifizieren. Configuration DSLs beschreiben sowohl Abhängigkeiten zwischen Featuren, z.B. „requires“ und „conflicts“, als auch Restriktionen zwischen Featuren und/oder Attributen. Beispielsweise ergibt die Restriktion

```
alternative_child('Typ', Motor),
hasAttribute (Motor, 'Kennlinie_Fahrmodus_Eco')
```

nur dann eine gültige Konfiguration, falls der selektierte Motor aus den Alternativen *M1*, *M2*, oder *M3* (siehe **Error! Reference source not found.**) auch eine Kennlinie mit der Bezeichnung *Kennlinie_Fahrmodus_Eco* besitzt. Im obi-

[2] Bei Verwendung von pure-variants gibt es kein explizites Konfigurationsmodell. Das Konfigurationswissen wird sowohl im Feature Modell als auch in den Implementierungskomponenten verwaltet (siehe auch Abschnitt 3.2.2.3).

gen Beispiel ist *Motor* ein Platzhalter für einen selektierten Motor und *Kennlinie_Fahrmodus _Eco* ein Attribut.

ICCLs definieren Konfigurationssprachen im Lösungsraum. Sie werden verwendet,

a) um Abhängigkeiten zwischen Implementierungskomponenten bzw. Attributen von Implementierungskomponenten zu definieren. Dies ist vergleichbar mit den Configuration DSLs im Problemraum.

b) um Implementierungskomponenten konkreten Features aus dem Feature Modell zuzuordnen. Beispielsweise erzeugt die folgende Regel

```
alternative_child('Klimaanlage', Typ),
getAttribute(Typ, 'Additional_Load', Load),
swritef( Value,
         'set_parameterIB( "Klimaanlage",
                           "Additional_Load",
                           %w);',
         [Load])
```

einen Output-String abhängig vom selektierten Subfeaturen des Features *Klimaanlage* und des Wertes des Attributes *Additional_Load* des selektierten Subfeatures. Das Subfeature kann entweder die Klimaanlage *Klimaanlage_Typ1* oder *Klimaanlage_Typ2* sein (siehe Abbildung 12).

c) um Implementierungskomponenten der Variabilität im variantenreichen Simulink-Modell zuzuordnen. Zum Auflösen der Variabilität im variantenreichen Simulink-Modell wurden Matlab-basierte Funktionen definiert (zur Modellierung der Variabilität siehe auch [PS+05]). Beispiele für diese Funktionen sind:

- set_confsub: Ersetzen eines Template-Blockes durch einen Member-Block.
- set_parameterIW: Zuweisen eines Wertes zu einem Parameter im Workspace.
- set_lookup2dIB: Zuweisen eines zweidimensionalen Look-Up-Tabes zu einer Variablen in einem Block.
- . . .

Diese Funktionen bestehen aus einer Folge von Matlab-Befehlen, so genannten Model Construction Commands. Beispielsweise weist die Funktion *set_parameterIB* einem in einem Simulink-Block definierten Parameter einen Wert zu. Auf diese Weise wird die Variabilität dieses Parameters aufgelöst:

```
set_parameterIB(Blockname, VarName, VarValue)
  % --- Blockname = name of the block
  % --- VarName   = name of the parameter
  % --- VarValue  = value of the parameter
```

```
  %search for block "Blockname"
  %search in block for parameters with variability
   (= VariationPointIB::ParameterName)
  %check parameter "VarName" is part of collection
  %load parametername-value-pair
  %assign value "VarValue" to parameter "VarName"
   in block "Blockname"
end;
```

Die parametrierten Funktionsaufrufe sind Implementierungskomponenten zugeordnet. Während des Konfigurationsprozesses werden die Funktionsaufrufe konfiguriert und in einem Matlab-Skript zusammengestellt. Dieses Matlab-Skript kann anschließend in Matlab ausgeführt werden.

4 Application Engineering

Application Engineering dient der Entwicklung eines konkreten Produktes auf der Basis einer Prozessfamilien-Infrastruktur.

Ausgangspunkt für das Application Engineering sind die Anforderungen an die Produktfamilie, die im Rahmen des Domain Scopings aufgenommen wurden, und die Prozessfamilien-Infrastruktur. Das Ergebnis von Application Engineering bzw. der Instanziierung der Prozessfamilien-Infrastruktur ist zum einen ein konkretes Produkt, welches aus aufgelösten variantenreichen Prozessen besteht, die dann keine Variabilität mehr beinhalten und zum anderen aus dem Auflösungs-Modell (engl. resolution model), das die Konfigurationsinformation für das entsprechende Produkt erfasst.

Im Folgenden werden zwei mögliche Vorgehensweisen für das Application Engineering beschrieben: In Abschnitt 4.1 der vom Fraunhofer IESE entwickelte Ansatz der Produktinstantiierung auf der Basis von Entscheidungsmodellen und in Abschnitt 4.2 der von DaimlerChrysler entwickelte Ansatz zur Variantenkonfiguration.

4.1 Instantiierung mittels Entscheidungsmodellen

Der in [BBG05] dokumentierte PESOA Prozess unterscheidet innerhalb des Application Engineerings wie beim Domain Engineering zwischen Analysis, Design und Implementation. Dieser Abschnitt präsentiert eine Vorgehensweise für die in den beiden ersten Phasen definierten Aktivitäten „Specify Product“ und „Configure Product“. Diese Vorgehensweise beschreibt die Instanziierung eines konkreten Produktes mittels Entscheidungsmodellen, die in Abschnitt 3.1.2 definiert sind.

Wie in Abbildung 13 beschrieben ist der erste Schritt („Specify Product“ [BBG05]) des Application Engineerings die Analyse des Produktes. Ausgangspunkt ist die Liste von Eigenschaften („Feature Model“ [BBG05]), die in der Prozessfamilie bzw. in den variantenreichen Prozessen umgesetzt sind. Das Ergebnis ist eine Spezifikation der Eigenschaften des neuen Produktes. Die Produkteigenschaften sind einerseits eine Selektion aus der Liste („Product Feature Model“ [BBG05]), und andererseits auch neue Eigenschaften, die noch nicht in der Prozessfamilie vorgesehen sind („Not Covered Features“ [BBG05]).

Der zweite Schritt („Configure Product“ [BBG05]) hat zum Ziel die Variationspunkte innerhalb des variantenreichen Prozess entsprechend der analysierten Produkteigenschaften aufzulösen. Input für diese Instantiierung sind die identi-

fizierten Produkteigenschaften, das Entscheidungsmodell, und der variantenreiche Prozess. Ergebnis ist das konkrete Produkt und ein so genanntes Auflösungsmodell („Resolution Model“ [BBG05]), das die Konfigurationen des Entscheidungsmodells für die Auflösung der Variationspunkte entsprechend der Produkteigenschaften dokumentiert.

Für die Auflösung der Variationspunkte mittels Entscheidungsmodell gibt es zwei Vorgehensweisen, bottom-up und top-down. Die bottom-up Methode durchläuft den variantenreichen Prozess und löst jeden einzelnen Variationspunkt mit Hilfe des Entscheidungsmodells auf, d.h. bottom-up im Entscheidungsmodell. Die zweite Methode löst den variantenreichen Prozess für jede einzelne identifizierte Eigenschaft auf, dabei werden Entscheidungen im Entscheidungsmodell top-down ausgeführt. Bei beiden Vorgehensweisen werden die Schritte und die Auflösungsschritte mittels Entscheidungsmodell im Auflösungsmodell dokumentiert.

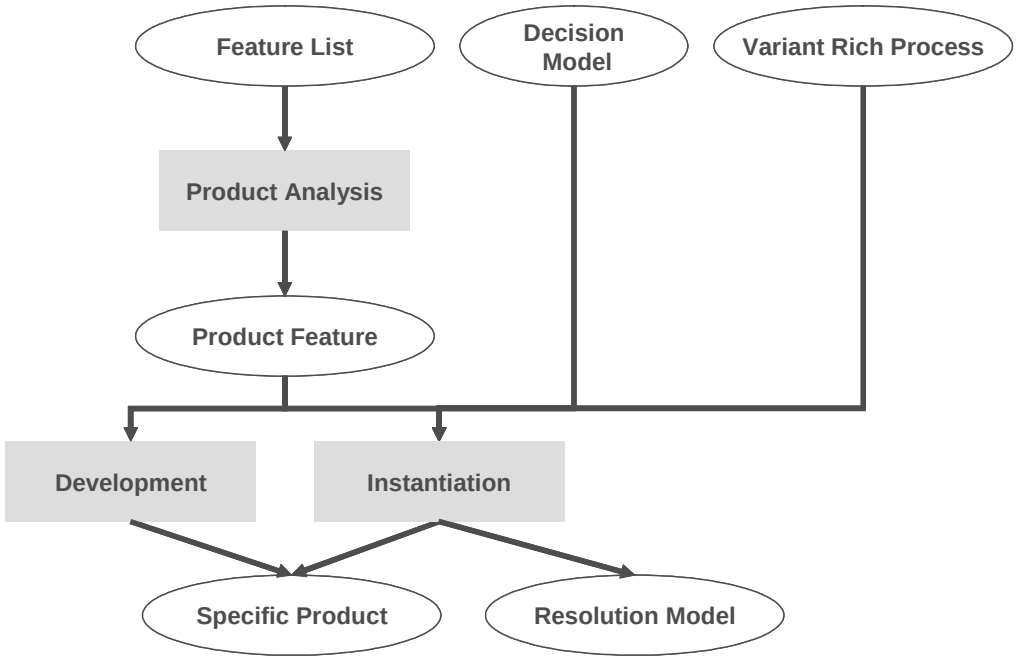

Abbildung 13: Application Engineering

Das Ziel von „Development“ ist es Produkteigenschaften („Not Covered Features“ [BBG05], die nicht durch eine Instantiierung realisiert werden können, umzusetzen. Input ist die Liste von Eigenschaften, die nicht in der Prozessfamilie vorzufinden sind. Das Ergebnis ist dann das Produkt mit allen spezifizierten Eigenschaften. Die Realisierung der Produkteigenschaften erfolgt wie im PESOA Prozess beschrieben.

Im Folgenden wird die Instantiierung mittels Decision Models anhand der Benzinmotorsteuerung exemplarisch erklärt. Abbildung 14 zeigt die möglichen Varianten „M1“, „M2“ und „M3“ für den Motortyp an. Wählt man Motortyp „M1“ aus, dann lösen sich entsprechend der in Abbildung 8 definierten Constraints die Variationspunkte „Einspritzung“ und „Klopfsensor“ auf, wie in Abbildung 15 dargestellt.

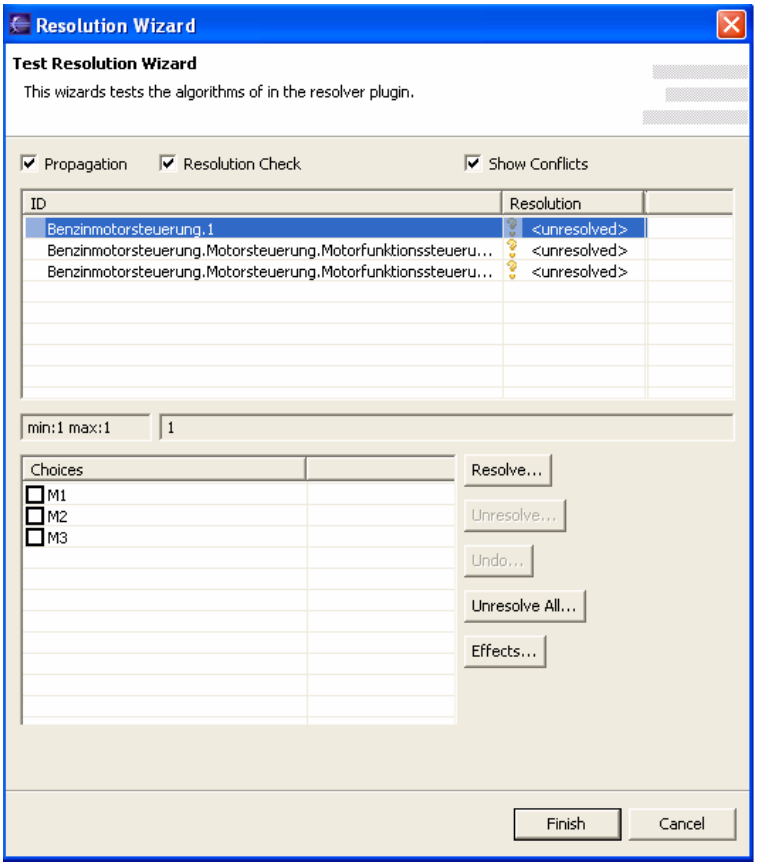

Abbildung 14: Beispiel – unaufgelöster Variationspunkt „Motortyp“

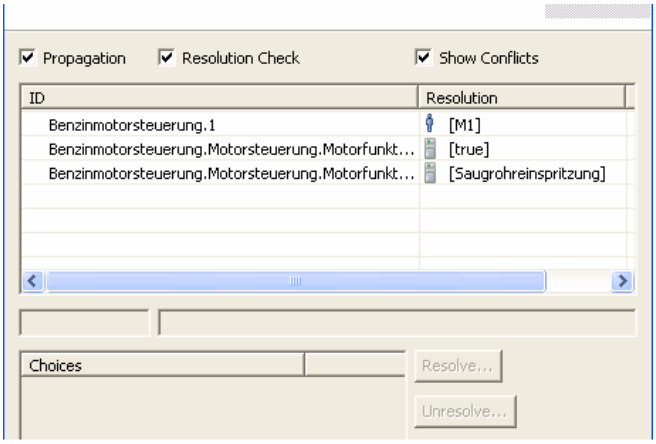

Abbildung 15: Beispiel – Aufgelöster Variationspunkt „Motortyp"

4.2 Variantenkonfiguration

Ziel der Variantenkonfiguration ist es aus dem variantenreichen Simulink-Modell im Rahmen des Application Engineerings das Modell einer konkreten Produktinstanz, in der die Variabilität aufgelöst ist, abzuleiten. Die Verbindung zwischen dem Variabilitätsmanagement und dem variantenreichen Simulink-Modell wird über ein Matlab-Skript hergestellt. In Abbildung 16 wird der Prozess zur Instanziierung variantenreicher Simulink-Modelle dargestellt.

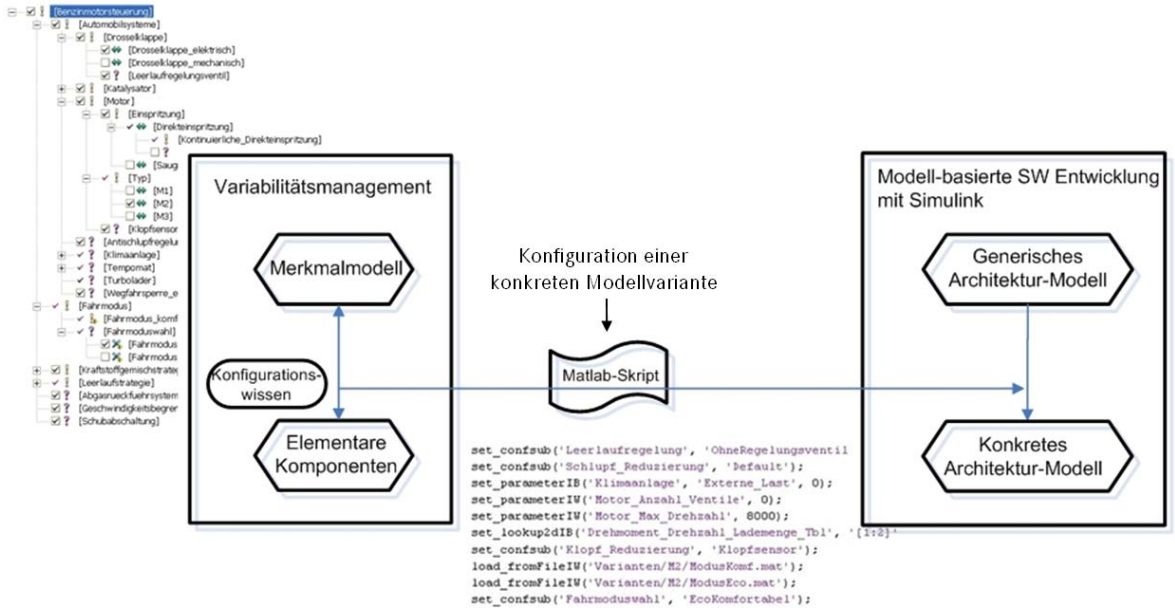

Abbildung 16: Instanziierung variantenreicher Simulink-Modelle

Während der Variantenkonfiguration wird aus dem Feature Modell eine gültige Feature Konfiguration selektiert. Durch einen XSL-Transformator werden die parametrierten Matlab-basierten Funktionsaufrufe mittels der Implementierungskomponenten konfiguriert (siehe auch Abschnitt 3.2.2.3). Während des Transformationsprozesses werden diese Funktionsaufrufe zu einem ausführbaren Matlab-Skript zusammengestellt.

Hat beispielsweise der Parameter *Additional_Load* der Klimaanlage *Klimaanlage_Typ1* den Wert 250, so wir bei Auswahl der Klimaanlage *Klimaanlage_Typ1* die folgende Zeile mittels der Implementierungskomponente *Klimaanlage* (siehe Abschnitt 3.2.2.3) in das Matlab-Skript geschrieben:

```
set_parameterIB("Klimaanlage", "Additional_Load", 250);
```

Das zusammengestellte Matlab-Skript kann abschließend in Matlab gestartet werden. Hierbei werden die parametrierten Funktionen ausgeführt und die Variabilität im variantenreichen Simulink-Modell aufgelöst. Das Ergebnis ist das Modell einer konkreten Produktvariante aus der nun in Matlab, z.B. mit dem Real-Time Workshop, Sourcecode generiert werden kann.

5 Bindezeitmodelle

Der Zeitpunkt, an dem eine Variabilität an einen Variationspunkt gebunden wird, ist als Bindezeitpunkt zu bezeichnen [vgl. Brag+04]. Ein Bindezeitpunkt repräsentiert also den Augenblick, an dem die Entscheidung für eine bestimmte Option endgültig getroffen wird.

Bindezeitpunkte können, abhängig von der modellierenden Person, aus verschiedenen Sichtweisen betrachtet werden [vgl. Clea01]:

- Aus Perspektive des Domänenentwicklers kann eine Bindung während der Domain Analyse und deren Implementierung erfolgen. Darüber hinaus ist jedoch auch beim Domain Design die Festlegung auf bestimmte Varianten zweckmäßig.
- Aus der Sichtweise des Produktentwicklers erfolgt eine Bindung während der Spezifikation, vor Beginn der Generierung bzw. bei der Kompilierung und Verknüpfung.
- Der Endbenutzer schließlich hat die Möglichkeit, bei der Installation, der Initialisierung und zur Laufzeit über die verschiedenen Varianten zu entscheiden.

Die Bindezeiten im Rahmen der Softwaresystemfamilien- Entwicklung können in product architecture derivation, buildtime, activation time und runtime klassifiziert werden [vgl. Svah+01]. Eine Entscheidung, die in einer früheren Phase getroffen wird, gilt automatisch für sämtliche späteren Entwicklungsphasen. Je früher eine Entscheidung erfolgt, desto höher ist die Effizienz und umso niedriger sind die Kosten. Andererseits wird eine umso größere Flexibilität der Systemfamilie erreicht, wenn die Entscheidungen für eine bestimmte Variante so spät wie möglich getroffen werden [vgl. Svah+02].

Eine weitere Einteilung von Bindezeitpunkten, ist die der internen bzw. externen Bindung [vgl. Svah+02]. Die interne Bindung setzt voraus, dass das System die Funktionalitäten zur Bindung sowie die einzelnen Varianten enthält. Diese Art ist charakteristisch für Varianten, die zur Laufzeit gebunden werden. Externe Bindung hingegen setzt voraus, dass eine Person oder ein Software- Tool die Bindung vornimmt. Somit ist diese Bindeart typisch für die Phasen in denen Werkzeuge (Konfigurations- Management- Tools bzw. Compiler und Linker) zum Einsatz kommen. Das Linken nimmt dabei eine Sonderstellung ein, da es dynamisch zur Laufzeit sowie unter der Kontrolle des Systems erfolgen kann. Somit ist das Linken sowohl als intern, wie auch als extern zu klassifizieren. Der Vorteil einer externen Bindung ist, dass sie keine Rückstände im Quellcode hin-

terlässt, so wie es bei einer internen Bindung der Fall ist. Dies verringert die Komplexität des Quellcodes und wird deshalb in der Regel bevorzugt.

Konventionelle Realisierungstechniken

Um Variationspunkte zu implementieren, kommen verschiedene Realisierungstechniken zum Einsatz [vgl. Svah+02]. Die Entscheidung, welche Technik verwendet wird, ist abhängig von der Bindezeit und den verschiedenen Softwareentitäten, welche in den unterschiedlichen Entwicklungsstufen im Vordergrund stehen [vgl. Tabelle 3].

Tabelle 3: Entitäten während der Entwicklungsstufen

Entwicklungsstufe	Software- Entitäten
Architekturdesign	Komponenten Frameworks
Detaildesign	Framework Implementierungen Sammlung von Klassen
Implementierung	Individuelle Klassen Quellcodezeilen
Kompilieren	Quellcodezeilen
Linken	Komponenten

Kennzeichnend für eine Systemfamilienarchitektur ist, dass viele offene Variationspunkte vorhanden sind. Durch Bindung der Variabilitäten in diesen Variationspunkten wird eine Produktarchitektur geschaffen. In der Regel werden für diesen Prozess Konfigurationswerkzeuge verwendet. Die meisten dieser Mechanismen nutzen Software- Entitäten, deren Einführung während des Architekturdesigns erfolgte [vgl. Svah+02; Tabelle 4].

Zu Beginn der Produkterzeugung wird der Quellcode durch Kompilierung fertig gestellt und mithilfe von Compiler- Direktiven auf die spezifischen Bedürfnisse gekürzt. Ungeachtet dieser möglichen Reduktion kann sich der Quellcode auch vergrößern, indem z. B. Makros aufgelöst werden [vgl. Svah+02]. Die letztendliche Bindung der Varianten erfolgt dabei vor der Generierung des Endprodukts, aber nicht währenddessen.

Der kompilierte Code wird während der Linkphase zum lauffähigen Endprodukt zusammengefügt. Wann diese beginnt bzw. endet, hängt sehr stark von der genutzten Entwicklungs- und Laufzeitumgebung ab. Teilweise geschieht das Linken direkt nach dem Kompilieren und ist in der Regel unumkehrbar [vgl. Svah+02]. In anderen Fällen erfolgt das Linken erst beim Start der Applikation, also bei deren Aktivierung. Bestimmte Systeme erlauben ein flexibles Linken zur Laufzeit. Diese Aspekte beeinflussen maßgeblich, wie spät neue Varianten in das System eingepflegt werden können.

Nicht alle Entscheidungen sind in der Entwicklungsphase des Produktes zu treffen. Einige müssen auf die Zeit nach der Auslieferung an den Kunden verschoben werden. Diese können jedoch eine Betrachtung als verzögerter Entwicklungsschritt erfahren. In der Regel werden diese Varianten durch Startparameter festgelegt, die z.B. in Form von Konfigurationsdateien vorliegen. Eine Entscheidung dieser Art kann auch während der Laufzeit erneut getroffen werden. Darüber hinaus ist es möglich, dass Softwaresysteme nach der Auslieferung aktualisiert werden müssen, wenn eine nachträgliche Einbringung von Varianten gewünscht ist, aber die Variationspunkte an der benötigten Stelle nicht existieren.

Eine maximale Flexibilität wird für eine Softwareproduktlinie erreicht, wenn die Bindung der Variabilitäten erst zur Laufzeit erfolgt. Dies bedeutet aber, dass das System Funktionalitäten zur Bindung vorhalten muss, was in der Regel zu Performanceverlusten durch erhöhten Speicherbedarf oder auch durch die erhöhte Kommunikation zwischen den Softwarekomponenten führt [vgl. Svah+02]. Ferner müssen sämtliche Varianten des betreffenden Variationspunktes in der Anwendung enthalten sein [vgl. Brag+04].

Tabelle 5 fasst die Realisierungstechniken für die soeben diskutierten Entwicklungsstufen stichpunktartig zusammen.

Tabelle 5: Realisierungstechniken für Bindezeiten

Realisierungstechnik	**Bindezeitpunkt**	**Kategorisierung**
Architecture Reorganization	Produktarchitektur- Ableitung	Extern
Variant Architecture Component		
Optional Architecture Component		
Binary Replacement – Linker Directives	Linken	Intern oder Extern
Binary Replacement – Physical	Aktivierung	Extern
Infrastructure-Centered Architecture	Kompilierung und Laufzeit	Intern
Variant Component Implementations	Laufzeit	
Variant Component Specialization	Produktarchitektur- Ableitung	Extern
Optional Component Specialization		
Runtime Variant Component Special.	Laufzeit	Intern
Condition on Constant	Kompilierung	Intern oder Extern
Condition on Variable	Laufzeit	Intern
Code Fragment Superimposition	Kompilierung oder Laufzeit	Extern

Nähere Erläuterungen zu den Detailaspekten der in Tabelle 5 aufgezeigten Realisierungstechniken sind [Svah+02], [Kicz+97] und [Brag+04] zu entnehmen.

Zeitlinienvariabilität

Sollen Variationspunkte zu mehreren Zeitpunkten des Lebenszyklus gebunden werden, handelt es sich um den Ansatz der Zeitlinienvariabilität. Dadurch wird die Konfiguration eines Softwaresystems sehr flexibel, da die Entscheidung, wann ein Feature gebunden wird, zunächst offen bleibt [vgl. Dols+03a]. Zeitlinienvariabilität ist eine zusätzliche Variabilitätsdimension, die oft unbeachtet bleibt. In der Praxis werden die Themen Zeitlinien und Zeitlinienvariabilität weder betrachtet, noch explizit modelliert. Variable assets sind dadurch oft nicht orthogonal zur Zeitlinie und das Design einer Systemvariabilität erscheint ad hoc zu entstehen [vgl. Dols+03b].

Zeitlinienvariabilität erzeugt jedoch einige Grundprobleme in der Implementierung und Konfiguration. Das Hauptproblem im Rahmen der Implementierung ist, dass zeitvariable Variationspunkte in existierenden Entwicklungsumgebungen und Programmiersprachen nicht unterstützt werden. Ein zentrales Problem der Konfiguration ist, dass jede Phase im Lebenszyklus ein anderes Konfigurationsinterface benutzt, durch welches die Entscheidungen für Varianten getroffen werden. Dies wirkt sich nachteilig aus, wenn verschiedene Bindezeiten in einem Variationspunkt verwendet werden sollen.

Zeitlinienvariabilität bedingt schließlich ein Framework, welches eine einheitliche Schnittstelle zu einer Vielzahl von darunter liegenden Konfigurationsmechanismen repräsentiert. Im Rahmen des TraCE- Forschungsprojektes[3] entsteht hierfür ein generisches Konfigurationsinterface, welches durch ein formales Feature Modell parametrisiert wird [vgl. TRAC05].

Um Zeitlinienvariabilität einbinden zu können, werden weitere Implementierungstechniken benötigt, die eine bessere Bindung zu verschiedenen Zeitpunkten ermöglichen. Eine Realisierung dieses Ansatzes geht jedoch über eine Plattformentwicklung für Prozessfamilien hinaus und sollte demnach von den weiteren Betrachtungen ausgeschlossen werden.

Angewandte Bindezeitmodelle

Die folgenden Ansätze wurden hinsichtlich der vorgeschlagenen Bindezeitpunkte untersucht und in Tabelle 6 vergleichend gegenübergestellt:

- **Feature- Oriented Domain Analysis (FODA):** Modell zur featuresorientierten Domain Analyse [vgl. Kang+90].
- **Product Line Software Engineering (PuLSE):** Methode zur Bereitstellung eines anpassbaren Rahmenwerkes für die Entwicklung von Softwaresystemfamilien [vgl. Anas+00].
- **KobrA:** Objektorientierte und gebrauchsfertige Ableitung von PuLSE, die das PuLSE- Vorgehensmodell übernimmt [vgl. Atki+00; Atki+01].

[3] Transparent Configuration Environments (TraCE): http://www.cs.uu.nl/wiki/Trace/WebHome

- **FeatuRSEB:** Verknüpfung der Wiederverwendungsmechanismen von Komponenten aus dem Reuse- Driven Software Engineering Business (RSEB) mit der Featuremodellierung von FODA [vgl. Gris+98; Jaco+97].
- **Catalysis:** Objektorientiertes Vorgehensmodell, welches insbesondere die komponentenorientierte Entwicklung fokussiert [vgl. Souz+99].
- **Bosch- Automotive:** Bindezeitenmodell zur Unterstützung von Softwaresystemfamilien in eingebetteten Steuergeräten des Automobilbereiches [vgl. Frit+02].

Tabelle 6: Angewandte Bindezeitpunkte

<table>
<tr><th>Zeitpunkt</th><th>FODA</th><th>PuLSE</th><th>KobrA</th><th>Catalysis</th><th>Featu RSEB</th><th>Bosch</th></tr>
<tr><td rowspan="2">Ableitung Produkt-architektur</td><td rowspan="2">-</td><td rowspan="2">-</td><td rowspan="2">Implementation</td><td rowspan="2">Coding</td><td rowspan="4">Reuse</td><td>Domain Engineering</td></tr>
<tr><td>Application Engineering</td></tr>
<tr><td rowspan="3">Produkt-erzeugung</td><td rowspan="3">Compile</td><td>Compile</td><td rowspan="3">Build</td><td>Compile</td><td>Compilation</td></tr>
<tr><td rowspan="2">Source Selection</td></tr>
<tr><td>Link</td><td>Link</td><td rowspan="6">Use</td></tr>
<tr><td rowspan="2">Produkt-aktivierung</td><td rowspan="2">Activation</td><td rowspan="2">-</td><td>Installation</td><td rowspan="2">-</td><td>Image Build</td></tr>
<tr><td>Start-up</td><td>Supplier EOL</td></tr>
<tr><td rowspan="4">Laufzeit</td><td rowspan="3">Runtime</td><td rowspan="3">Runtime</td><td rowspan="3">Runtime</td><td>Dynamic</td><td>Automotive EOL</td></tr>
<tr><td>Runtime</td><td rowspan="2">First Startup</td></tr>
<tr><td>Reflective</td><td rowspan="2">Every Startup</td></tr>
<tr><td>-</td><td>Update</td><td>-</td><td>-</td><td>Dynamic</td></tr>
</table>

Besonders Hervorzuheben ist das von der Robert Bosch GmbH entwickelte Bindezeitmodell, welches die eingebetteten Softwaresysteme des Automobilbereiches fokussiert [vgl. Frit+02]. Hierbei werden vier Hauptphasen der Implementierung von embedded Softwaresystemen unterschieden, in denen jeweils mehrere Bindezeiten vorgeschlagen werden. Im einzelnen sind dies die, in Abbildung 17 dargestellten, Bindezeitpunkte der Programmierung, Integration, Montage und Laufzeit.

Die durch Abbildung 17 visualisierte Phase der Programmierung unterteilt sich in das Domain Engineering (siehe Kapitel 3), in dem vor allem die Entwicklung der core assets im Vordergrund steht, sowie in das Application Engineering (siehe Kapitel 4), in welchem einzelne Artefakte zu einem Endprodukt zusam-

mengefügt werden. In beiden Entwicklungsphasen ist es möglich, Varianten zu binden, so dass zwischen verschiedenen Bindezeitpunkten unterschieden werden muss.

Die sich anschließende Integration der Systemsoftware umfasst die Erstellung des ausführbaren Produktes aus verschiedenen Quellen. Variabilitäten können in dieser Phase binnen verschiedener Teilprozesse gebunden werden. Während der Kompilierung resultieren Entscheidungen vor allem aus Präprozessor- Direktiven. Die Auswahl von bestimmten ausführbaren Softwareteilen ist ein weiterer Bindezeitpunkt. Dies können Codes der benötigten Produktartefakte sein, aber auch Daten, Ressourcendateien und Komponenten von Dritten. Ein weiterer Bindezeitpunkt ist in der Herstellung verschiedener Images zu identifizieren, die im nächsten Entwicklungsschritt auf den Flashspeicher des Gerätes übertragen werden.

Das Beschreiben des Flashspeichers kann im Rahmen des Montagevorgehens vonseiten des Zulieferers (Supplier) oder des Herstellers (Automotive) umgesetzt werden. Hierbei besteht die Möglichkeit, bestimmte Teile des Speichers zu überschreiben und neue Variabilitäten zu binden.

Eine nachgelagerte Bindung kann schließlich während der Aktivierung bzw. zur Laufzeit erfolgen. Dies bedeutet, dass sich die Software entweder nur beim ersten Start oder bei jeder Startphase auf die spezifische Hardware des Gerätes einstellt. Optional ist auch eine spätere Bindung möglich, wenn z. B. Hardware-Elemente bei Reparaturen oder planmäßigen Wartungsarbeiten ausgetauscht werden.

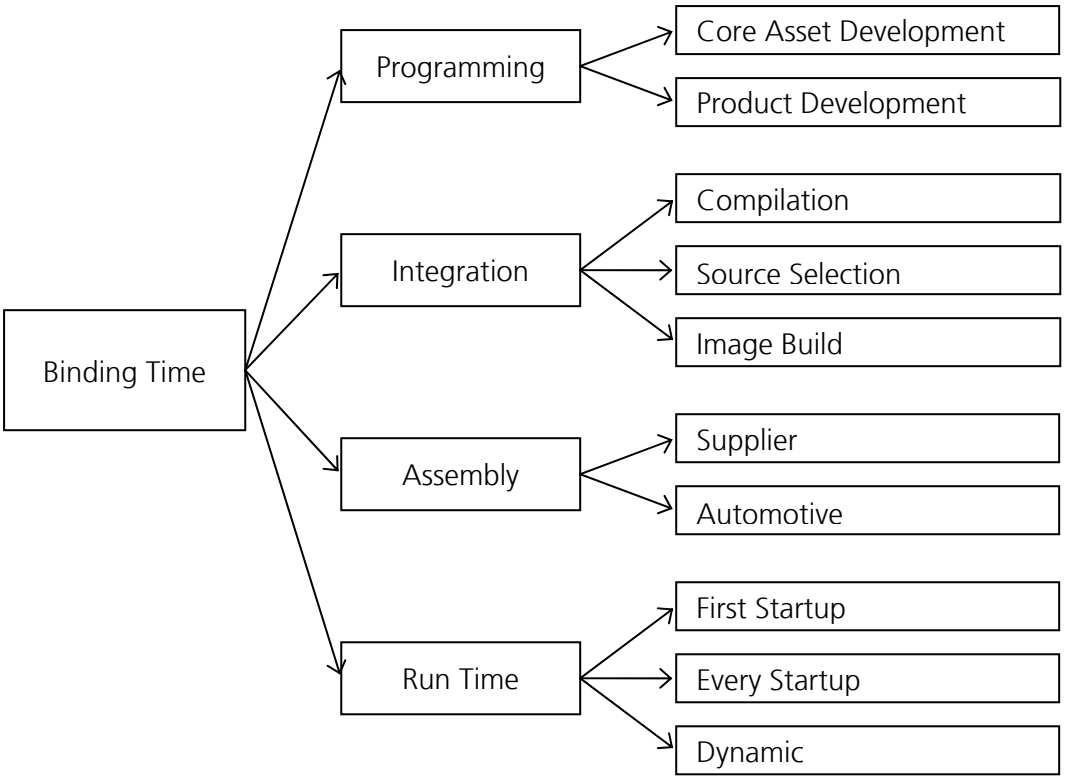

Abbildung 17: Bindezeiten in eingebetteten Automotive- Systemen [vgl. Frit+02]

Abschließend ist zu erkennen, dass nur KobrA, FeatuRSEB und der Ansatz der Robert Bosch GmbH, von den in Tabelle 6 dargestellten Modellen, Bindezeiten für den gesamten Entwicklungszyklus anbieten.

Bei der Gegenüberstellung der verwendeten Bindezeiten in den untersuchten Modellen fällt deren unterschiedliche Anzahl auf [vgl. Tabelle 6]. Das Spektrum reicht dabei von zwei Bindezeiten bei FeatuRSEB bis hin zu zehn Phasen der fein untergliederten Bosch- Methode. Die beiden Bindezeiten des erstgenannten Ansatzes sind allerdings eher als Zeiträume zu verstehen und decken ausdrücklich mehrere, nicht näher spezifizierte Zeitpunkte ab. Im Mittel werden etwa vier bis fünf Bindezeiten unterschieden, da sich die Komplexität der Produktlinie mit jedem weiteren Bindezeitpunkt vergrößert.

PESOA- Bindezeitraummodell

Unter Bezugnahme auf die bisherigen Analysen wird im Folgenden ein Bindezeitmodell für das PESOA- Forschungsprojekt vorgeschlagen. Hierfür erfolgt eine Definition der nachstehenden vier Zeitabschnitte, die wiederum unterschiedliche Bindezeitpunkte enthalten können:

1. **Prozess- Modellierung:** Schon zum Zeitpunkt der frühen Phasen des Domain Engineerings ist es möglich, sich auf bestimmte Varianten festzulegen. Dies betrifft vor allem Entscheidungen, deren spätere Bindung keinen Mehrwert ergibt. Beispielsweise kommen, durch mögliche Benutzer zu charakterisierende, Login- Prozesse in beiden PESOA- Domänen zur Anwendung. In der Automotive- Domäne kommen z. B. Fahrzeugführer und Fahrzeugtechniker in Frage, die über jeweils unterschiedliche Kompetenzen im Zugriff auf die Funktionalitäten einzelner Steuergeräte besitzen. Ein Fahrzeugtechniker könnte somit per speziellem Login auf gesonderte Funktionen eines eingebetteten Steuergerätes zugreifen bzw. diese modifizieren. Darüber hinaus kann auch die fokussierte Sicherheitsstufe für diese Login- Prozesse schon während der Prozessmodellierung festgelegt werden. Ein weiteres Beispiel ist in der Unterstützung von heterogenen Absatzmärkten zu sehen, wonach ein Steuergerät bereits auf Ebene der Prozess- Modellierung an die Spezifika einer bestimmten Region/ Straßenverkehrsordnung angepasst werden kann.
2. **Prozess- Instantiierung:** Innerhalb des Application Engineerings werden die variantenreichen Prozessmodelle importiert, konvertiert und erfahren eine generative Transformation (Instanziierung) auf die jeweilige Zielplattform. Ein häufig zitiertes Beispiel für das Auftreten einer Featurevariabilität der Automotive- Domäne, ist die potentielle Integration einer elektronischen Wegfahrsperre. Diese muss nicht nur im Domain sondern auch im Application Engineering beachtet werden, da sich Änderungen in den Zuständen und Ereignissen des Systems ergeben.
3. **Prozess- Aktivierung:** Die Prozess- Aktivierung ist der letzte Bindezeitpunkt vor der planmäßigen Laufzeit der Software. In den obigen Analysen wurde bereits herausgestellt, dass diese Phase vor allem in der Automotive- Domäne eine wichtige Rolle spielt. Hier können Softwarevarianten an die spezifischen Wünsche des Kunden angepasst werden, was vor allem durch Überschreiben des Geräteflashspeichers vonseiten des Zulieferers bzw. durch den Hersteller realisiert wird. Diese Art der Anpassung ist während des laufenden Betriebes nicht immer möglich. Darüber hinaus kann eine Variantenbindung durch Parametrierung erfolgen, die eine Anpassung der eingebetteten Software an deren Umgebung (Aktuatoren und Sensoren) ermöglicht.
4. **Prozess- Ausführung:** Die maximale Flexibilität einer Systemfamilie wird erreicht, wenn Variabilitäten erst zur Laufzeit gebunden werden, da hier die Entscheidung so spät wie möglich erfolgt. Die Bindung zur Laufzeit ist jedoch als sehr kritisch für den Ressourcenbedarf einzustufen. Eine Variantenbindung zur Laufzeit ist daher in der Automotive- Domäne nur bedingt empfehlenswert. Hinsichtlich einer Bindung zur Updatezeit sind im Automobilbe-

reich verschiedene Szenarien denkbar, die eine Notwendigkeit eines Bindezeitpunktes während der Wartung begründen.[4]

Die hinsichtlich ihrer zeitlichen Einstufung in den Entwicklungsprozess hierarchisierten Bindezeiträume werden durch Tabelle 7 erneut aufgegriffen und beschreiben das allgemeine PESOA- Bindezeitmodell.

Tabelle 7: Generische PESOA- Bindezeiträume

Bezeichnung Bindezeitraum	Klassische Einordnung	Flexibilität	Aufwand	Bindezeitpunkt- Beispiele (Automotive)
Prozess- Modellierung	Domain Engineering	Gering	Gering	Techniker- Login
Prozess- Instantiierung	Application Engineering	□	□	Elektronische Wegfahrsperre
Prozess- Aktivierung	Installation und Start- Up	□	□	Flashspeicher aktualisieren
Prozess- Ausführung	Lauf- und Updatezeit	Hoch	Hoch	Wartungsarbeiten

Abschließend bleibt festzuhalten, dass für PESOA die in Tabelle 7 dargestellten Bindezeiträume der Prozess- Modellierung, Prozess- Instantiierung, Prozess- Aktivierung und Prozess- Ausführung vorgeschlagen werden. In diesem Kontext ist hervorzuheben, dass eine offene Ausgestaltung der Bindezeiträume mit nicht näher definierten Bindezeitpunkten zur Erhöhung der Flexibilität des PESOA-Vorgehens beiträgt.

[4] Die Nachrüstung einer Wegfahrsperre erfordert deren Berücksichtigung in der Motorsteuerung. Eine neue Softwareversion des Motorsteuergerätes kann darüber hinaus beispielsweise den Kraftstoffverbrauch senken, aber auch durch den Austausch eines defekten Gerätes notwendig sein.

6 Zusammenfassung

Auf der Basis des PESOA Prozesses [BBG05] werden in diesem Fachbericht zwei Vorgehensweisen zur Modellierung, Verwaltung und Instanziierung einer Prozessfamilie vorgestellt. Die Vorgehensweisen werden anhand eines Beispiels aus der Domäne Benzinmotorsteuerung erläutert.

Das Hauptartifakt der Prozessfamilie sind die variantenreichen Software-basierten Prozesse. Diese werden während des Domain Designs basierend auf den analysierten Features modelliert. Basierend auf diesen variantenreichen Prozessen werden während des Application Engineerings im Rahmen eines Konfigurationsprozesses konkrete Produkte instanziiert.

Process-Family Engineering auf Basis des Entscheidungsmodell-basierten Ansatzes verfolgt den beim Fraunhofer IESE entwickelten PuLSE-Ansatz. Im Rahmen des Scoping wird die Produktfamilie und ihre Grenzen definiert. Das Ergebnis ist eine Liste der Produkte einer Produktlinie mit ihren entsprechenden Eigenschaften und eine Bewertung des Produktlinienpotentials. Im Rahmen der Entscheidungsmodellierung wird die Verbindung zwischen den Eigenschaften der Produkte und den Variationspunkten in den variantenreichen Prozessen hergestellt. Während des Application Engineerings werden zur Instanziierung eines konkreten Produktes Entscheidungen im Entscheidungsmodell auf Basis selektierter Eigenschaften, die das Produkt haben soll, getroffen.

Process-Family Engineering auf Basis des Feature-basierten Ansatzes beruht auf dem FODA-Ansatz und dem generativen Domänenmodell. Scoping bedeutet hier relevante Funktionsbereiche für eine Wiederverwendung zu identifiziert und abzugrenzen. Die gemeinsamen und variablen Eigenschaften der Produkte der Produktfamilie werden im Rahmen der Featuremodellierung erfaßt. Das Konfigurationswissen, d.h. die Verbindung zwischen Featuren und den Variationspunkten in der gemeinsamen Architektur der Produktfamilie, wird über Domain-spezifische Sprachen (DSLs) und Konfigurationssprachen für die Implementierungskomponenten (ICCLs) definiert und implementiert. Im Application Engineering dient das Konfigurationswissen der Validierung und Konfiguration eines konkreten Produktes auf Basis selektierter Feature.

Im Rahmen des Process Family Engineerings kann die Variabilität in den Prozessen zu verschiedenen Zeitpunkten aufgelöst werden. Mögliche Bindezeiträume werden vorgestellt und eine Empfehlung für ein PESOA-Bindezeitmodell gegeben.

Referenzen

[Anas+00] Anastasopoulos, M., Gacek, C. *Implementing Product Line Variabilities.* In: Proceedings of the 2001 Symposium on Software Reusability (SSR'01), Toronto 2001.

[Atki+00] Atkinson, C., Bayer, J., Muthig, D. *Component-Based Product Line Development: The KobrA Approach*. In: Proceedings of the First International Software Product-Line Conference (SPLC-1), Denver 2000.

[Atki+01] Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., Muthig, D., Paech, B., Wüst, J., Zettel, J. *Component-based Product Line Engineering with UML*. Addison-Wesley, London 2001.

[BBG05] Bayer, Buhl. Giese, Lehner, Ocampo, Puhlmann, Richter, Schnieders, Weiland, Weske: „Process Family EngineeringModeling variant-rich processes" PESOA-Report No. 18/2005
July 14, 2005

[BO02] BOSCH, Kraftfahrtechnisches Taschenbuch, Vieweg, April 2002

[Brag+04] Bragança, A., Machado, R., J. *A Methodological Approach to Domain Engineering for Software Variability Enhancement.* In: Proceedings of the Second Workshop on Method Engineering for Object-Oriented and Component-Based Development (OOPSLA'2004), Vancouver 2004.

[CE00] K. Czarnecki, U. Eisenecker, Generative Programming – Methods, Tools, and Applications, Addison-Wesley, Boston, MA, 2000

[Clea01] Cleaveland, J., C. *Program Generators with XML and Java.* Prentice Hall PTR, Upper Saddle River 2001.

[Dols+03a] Dolstra, E., Florijn, G., Visser, E. *Timeline Variability: The Variability of Binding Time of Variation Points.* In: Proceedings of Workshop on Software Variability Management (SVM'03), Groningen 2003.

[Dols+03b] Dolstra, E., Florijn, G., de Jonge, M., Visser, E. *Capturing Timeline Variability with Transparent Configuration Environments.* In: Proceedings of International Workshop on Software Variability Management, Portland 2003.

[EL03] Europa Lehrmittel, *Tabellenbuch Kraftfahrzeugtechnik*, Europa-Lehrmittel, 2003

[Frit+02] Fritsch, C., Lehn, A., Strohm, T. *Evaluating Variability Implementation Mechanisms.* In: Proceedings of the Second International Workshop on Product Line Engineering - The Early Steps: Planning, Modeling, and Managing (PLEES'02), Seattle 2002.

[Gris+98] Griss, M. L., Favaro, J., d'Alessandro, M. *Integrating Feature Modeling with the RSEB*. In: Proceedings of Fifth International Conference on Software Reuse, Victoria 1998.

[Jaco+97] Jacobson, I., Griss, M., Jonsson, P. *Software Reuse: Architecture, Process and Organization for Business Success*. Addison Wesley, Reading 1997.

[Ka03] [Kang Kyo C.. *Feature-Oriented Product Line Software Engineering with Market Analysis*. POSTECH, 2003

[Kang+90] Kang, K., C., Cohen, S., G., Hess, J., A., Novak, W., E. Peterson, A., S. *Feature-Oriented Domain Analysis (FODA) Feasibility Study.* Technical Report CMU/SEI-90-TR-21, Pittsburgh 1990.

[Kicz+97] Kiczales, G., Irwin, J., Lamping J., Loingtier, J., M., Lopes, C., V., Maeda, C., Mendhekar, A. *Aspect Oriented Programming*. In: Proceedings of 11th European Conference on Object-Oriented Programming, Berlin 1997.

[Mut02] D. Muthig: A Light-weight Approach Facilitating an Evolutionary Transition Towards Software Product Lines. Stuttgart: Fraunhofer IRB Verlag, 2002 (PhD Theses in Experimental Software Engineering Vol. 11). Kaiserslautern, Univ., Diss., 2002

[MW04] The MathWorks, Simulink – Simulation and Model-based Design, Natick, MA, 2004

[PS04] pure-systems, pure::variants Eclipse Plugin User Guide, 2004

[PS+05] Puhlmann, Schnieders, Weiland, Weske: „Variability Mechanisms for Process Family Engineering", PESOA-Report No. 17/2005, June 30, 2005

[Sch03] Klaus Schmid, „Planning Software Reuse – A Discipline Scoping Approach for Software Product Lines", Fraunhofer IRB Verlag, 2003

[Souz+99] D'Souza, D., Wills, A. *Objects, Components, and Frameworks with UML - The Catalysis™ Approach*. Boston u. a. 1999.

[Svah+01] Svahnberg, M., van Gurp, J., Bosch, J. *On the Notion of Variability in Software Product Lines.* In: Proceedings of The Working IEEE/ IFIP Conference on Software Architecture (WICSA 2001), Amsterdam 2001.

[Svah+02] Svahnberg, M., van Gurp, J., Bosch, J. *A Taxonomy of Variability Realization Techniques, technical paper*. ISSN: 1103-1581, Blekinge Institute of Technology, Karlskrona 2002.

[TRAC05] TraCE – Forschungsprojekt (Hrsg.) *Transparent Configuration Environments.* http://www.cs.uu.nl/wiki/Trace/WebHome, Abruf am: 05.07.2005.

Dokumenten Information

Titel:	Feature- und Entscheidungsmodell-basierte Varianteninstanziierung im PESOA-Prozess
Datum:	8. September 2005
Report:	IESE-128.06/D
Status:	Final
Klassifikation:	Öffentlich