Fraunhofer Institut
Experimentelles
Software Engineering

PESOA Guidebook
Methodik und Techniken

Autoren:
Joachim Bayer
Cord Giese
Theresa Lehner
Alexis Ocampo
Frank Puhlmann
Arnd Schnieders
Jens Weiland
Andrej Werner
Sebastian Kiebusch

PESOA
Process Family Engineering in Service-Oriented Applications

BMBF-Project

IESE-Report Nr. 129.06/D
Version 1.0
27. August 2006

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. 22/2005

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, das "Process Family Engineering". Dieses umfasst die Entwicklung der Produktlinie bis zur Instanziierung konkreter Produkte auf Basis variantenreicher Prozesse. Das folgende Dokument beschreibt die Aktivitäten und Artefakte innerhalb des PESOA-Software-Entwicklungsprozesses. Das PESOA-Guidebook gibt hierfür einen Überblick über den PESOA-Prozess.

Zielgruppe des PESOA-Guidebooks sind Projektmanager, die sich einen Überblick über das Management und die Modellierung variantenreicher Prozesse machen wollen.

Schlagworte: PESOA, Produktlinienentwicklung, Prozess, variantenreicher Prozess, Prozessvariabilität.

Inhaltsverzeichnis

1 Einleitung

1.1 Projektkontext

PESOA ist ein durch das Bundesministerium für Bildung und Forschung (BMBF) finanziertes Gemeinschaftsprojekt. Das Ziel des Förderprojektes ist die Entwicklung und Implementierung einer Plattform für PESOA-Prozessfamilien. PESOA-Prozessfamilien 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 Domänen-Engineering, Produktlinien-Engineering und Software-Generierung mit Methoden aus dem Bereich Prozess-Engineering integriert.

1.2 Motivation und Zielsetzung

Die Anwendungsdomänen E-Business und Automotive sind geprägt durch eine immense Produktvielfalt. Die Ursachen dieser Produktvielfalt sind vielschichtig:

- Unterschiedliche Absatzmärkte (Europa, NAFTA, etc.) mit ihren unterschiedlichen Regularien,
- Bereitstellung unterschiedlicher Funktionalität zur Individualisierung der Produkte,
- Anpassung der Software an unterschiedliche Hardware, insbesondere im Automobilbereich.

Software-Produktfamilien sind das probate Mittel, Gemeinsamkeiten der Software-Produkte einer Anwendungsdomäne zu verwerten, während ihre Variabilitäten explizit und systematisch verwaltet werden. Das Verlagern des Fokus von Einzelprodukten zu Produktfamilien ermöglicht die Entwicklung ganzheitlicher Software-Architekturen, die die Grundlage für die Architekturen aller Instanzen der Produktfamilie bilden.

Zentrale Software-Engineering-Artefakte sind in der Designphase in den Anwendungsdomänen E-Business und Automotive hierbei oftmals Prozesse – E-Business- und "embedded control"-Prozesse. Sie beschreiben die Verhaltenscharakteristiken der Software-basierten Systeme. Im Rahmen der Produktfamilienentwicklung wurde die prozessorientierte Software bislang vernachlässigt. Unser Ziel ist daher die Integration von Produktfamilienentwicklung und Pro-

zess-Engineering, insbesondere für die Modell-basierte Software-Entwicklung. Entscheidend sind hierbei Konzepte zum expliziten Management von Variabilität, zur Modellierung von Variabilität in Software-Prozessen / Architekturmodellen sowie zur Auflösung dieser Variabilität. Diese Konzepte werden hier unter dem Begriff "Process Family Engineering" zusammengefasst. Besonders betrachtet wird hierbei die Modellierung von Variabilität mit UML und Matlab/Simulink für die Automotive-Domäne sowie BPMN für die E-Business-Domäne.

Ziel dieses PESOA-Guidebooks ist die Beschreibung der Aktivitäten innerhalb des PESOA-Produktfamilienentwicklungsprozesses – wie in [BBG05] vorgestellt. Das PESOA-Guidebook soll in erster Linie einen Überblick über den PESOA-Prozess geben. Hierfür werden die Vorgehensweisen, die innerhalb des PESOA-Projektes erarbeitet wurden kompakt präsentiert. Technische Details werden bewusst ausgelassen. Hierfür wird in den Abschnitten auf die weiterführenden und detaillierten PESOA-Fachberichte, sowie die referenzierte Literatur verwiesen. Zielgruppe des PESOA-Guidebooks sind hauptsächlich Projektmanager, die sich einen Überblick über das Management und die Modellierung variantenreicher Prozesse machen wollen.

1.3 Übersicht

Kapitel 2) gibt einen Überblick über die "Process Family Engineering"-Methodik aus [BBG05]. An diesem Prozess orientiert sich anschließend das Kapitel 3). Hier werden Schritt für Schritt die einzelnen Entwicklungsabschnitte und deren Ergebnisse beschrieben. Das Kapitel schließt mit der Projektplanung und Metriken zur Umfangsmessung und Aufwandsabschätzung für den PESOA-Prozess.

2 PESOA – Überblick

PESOA – Process Family Engineering in Service-Oriented Applications – ist ein Ansatz zur systematischen Wiederverwendung Software-basierter Prozesse mit Hilfe von Produktfamilientechniken. Die Produktfamilientechniken werden in PESOA auf Prozesse angewandt um eine optimale Unterstützung für Anwendungsdomänen zu gewährleisten, in denen Prozesse das zentrale Software-Entwicklungsartefakt sind. Beispiele für Prozesse als zentrale Entwicklungsartefakte sind Workflows in der E-Business-Domäne oder "embedded control"-Prozesse in der Entwicklung von Steuergerätesoftware für die Automobilindustrie. Um den Fokus auf Prozesse zu verdeutlichen, sprechen wir in PESOA von Prozessfamilien.

PESOA besteht aus drei Engineering-Aktivitäten und dem Projektmanagement (siehe Abbildung 1). Die Engineering-Aktivitäten sind dabei Scoping, Domänen-Engineering (engl. "domain engineering") und Anwendungs-Engineering (engl. "application engineering"). Das Scoping dient der Definition der Prozessfamilie und der einzelnen von der Prozessfamilie abgedeckten Produkte. Das Domänen-Engineering baut eine Wiederverwendungsinfrastruktur auf (die so genannte Prozessfamilieninfrastruktur), die dann im Anwendungs-Engineering wieder verwendet wird, um konkrete Produkte zu erstellen.

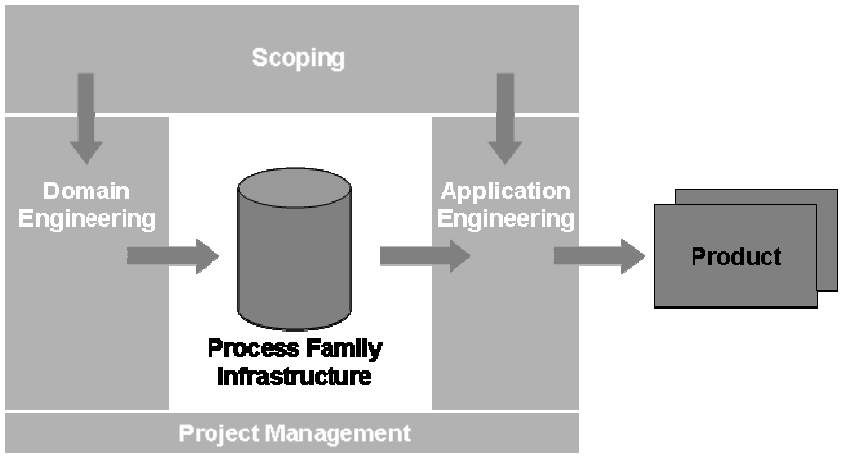

Abbildung 1: PESOA-Übersicht

Das Projekt-Management koordiniert die Engineering-Aktivitäten, insbesondere das Zusammenspiel zwischen dem Domänen-Engineering und den verschiedenen Anwendungsentwicklungsprojekten zur Erstellung einzelner konkreter Produkte.

Abbildung 2 stellt die PESOA-Engineering-Aktivitäten noch einmal im Detail dar.

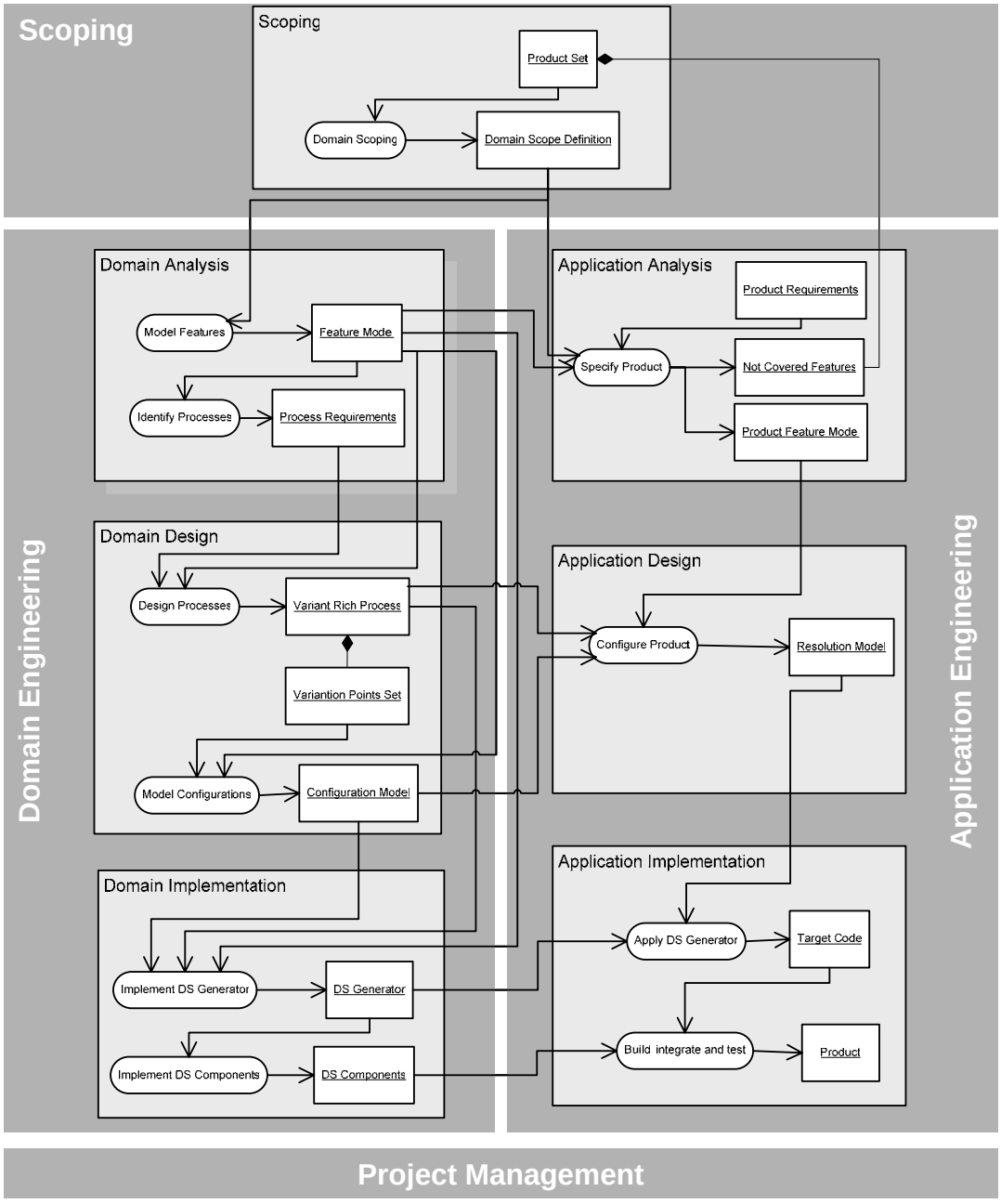

Abbildung 2: PESOA Engineering-Aktivitäten im Detail

In der Abbildung sind die Aktivitäten und die resultierenden Produkte zusammen dargestellt, das heißt dass die Prozessfamilien-Infrastruktur nicht einzeln

dargestellt ist, sondern gemeinsam mit dem Domänen-Engineering; sie besteht aus allen im Domänen-Engineering erstellten Produkten.

Das Scoping definiert die Prozessfamilie und ihre Grenzen. Dazu werden diejenigen Produkte identifiziert, die gemeinsam als Prozessfamilie entwickelt werden sollen. Das Ergebnis ist die Dokumentation der abgedeckten Produkte (im "Product Set") und der Merkmale dieser Produkte (in der "Domain Scope Definition").

Das Domänen-Engineering baut eine Prozessfamilieninfrastruktur auf, die generische, wieder verwendbare Artefakte enthält, die während des Anwendungs-Engineering an das jeweilige Produkt angepasst und wieder verwendet werden können. Domänen-Engineering besteht aus drei Schritten. Zunächst werden in der Domänenanalyse (engl. "domain analysis"), die Anforderungen an die Prozessfamilie ermittelt und dokumentiert. Der zweite Schritt ist der Domänenentwurf (engl. "domain design"), in dem generische Prozesse, so genannte variantenreiche Prozesse, entwickelt werden. Schließlich werden in der Domänenimplementierung (engl. "domain implementation") Komponenten und Generatoren entwickelt, die lauffähige Realisierungen der variantenreichen Prozesse darstellen.

Im Anwendungs-Engineering wird die Prozessfamilieninfrastruktur als wieder verwendbare Basis genommen, um konkrete Produkte zu erstellen. Dies geschieht wiederum in drei Schritten, die parallel zu den Schritten im Domänen-Engineering ablaufen. In der Anwendungsanalyse (engl. "application analysis") wird das konkrete Produkt spezifiziert, indem die generischen Anforderungen an die Prozessfamilie instantiiert werden. Im Anwendungsentwurf (engl. "application design") werden konkrete Prozesse dadurch gewonnen, dass die variantenreichen Prozesse konfiguriert werden. Schließlich werden in der Anwendungsimplementierung (engl. "application implementation") mit Hilfe domänenspezifischer Generatoren und Komponenten lauffähige Produkte erstellt.

Im folgenden Kapitel werden die einzelnen Aktivitäten, sowie die Projekt-Management-Aktivitäten Projektplanung (engl. "plan project") und Projektüberwachung (engl. "monitor project"), detailliert vorgestellt.

3 PESOA – Schritt für Schritt

In diesem Kapitel werden die in Kapitel 2 definierten PESOA-Aktivitäten detaillierter dargestellt. Es werden die einzelnen Aktivitäten mit ihren Zielen, Inputs und Outputs beschrieben. Basierend darauf werden unterschiedliche Techniken für die jeweiligen Aktivitäten vorgestellt. Diese Techniken wurden von den einzelnen Partnern des PESOA-Projektes entwickelt.

3.1 Aktivität "Domain Scoping"

"Domain Scoping" (oder kurz 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 Merkmalen (engl. "Features") in Beziehung gesetzt, die die Produkte kennzeichnen. Das Ergebnis des Scoping ist eine Liste von Produkten mit ihren identifizierten Merkmalen. In der Produktlinienentwicklung wird diese Scope-Definition oft in so genannten "Product Maps" dokumentiert.

3.1.1 Scoping mit PuLSE Eco

Zur Definition einer Produktfamilie und zur Bewertung ihres Wiederverwendungspotentials verfolgt das Fraunhofer IESE die in PuLSE™ Eco [Sch03] definierte Methode.

Ausgangspunkt für PuLSE Eco ist die Dokumentation von existierenden, geplanten und zukünftigen Produkten. Das Ergebnis der Scoping-Aktivität ist eine "Product Map", die die einzelnen Produkte der Prozessfamilie mit ihren entsprechenden Merkmalen auflistet, sowie eine Bewertung des Prozessfamilienpotenzials.

Die Erstellung der "Product Map" ("Product Line Mapping", siehe Abbildung 3) erfolgt in Zusammenarbeit mit verschiedenen Domänenexperten und Produktexperten bzgl. der Prozessfamilie. Um das Potenzial der einzelnen Domänen, d.h. der einzelnen technischen Bereiche in der Prozessfamilie, sowie der gesamten Prozessfamilien zu bewerten, werden die entsprechenden Experten interviewt ("Product Line Assessment").

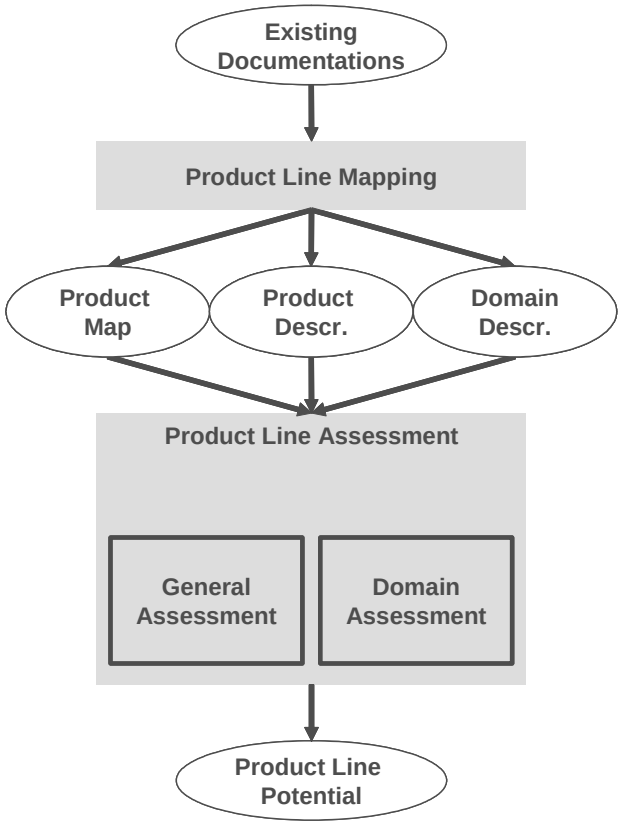

Abbildung 3: Scoping-Aktivitäten in PuLSE-Eco

Das Ziel der "Product Map"-Erstellung ist die Identifikation und Dokumentation der Produkte, deren Produktmerkmale und der (Teil-) Domänen der Prozessfamilie. Als Input werden zum einen existierende Dokumentationen der möglichen Produkte und zum anderen vorhandenes Expertenwissen verwendet. Zur Erstellung der "Product Map" bietet sich ein Workshop an, an dem sowohl die Domänen- und Produktexperten als auch das Management für die Prozessfamilie teilnehmen. In diesem Workshop werden dann die Produkte in der Prozessfamilie dokumentiert, indem deren Merkmale identifiziert und dokumentiert werden (in der Produktbeschreibung, engl. "product description"). Als nächstes werden die Merkmale zu (Teil-) Domänen gruppiert und dokumentiert (in der Domänenbeschreibung, engl. "domain description"). Die Domänen, Produkte und Merkmale dienen dann als Grundlage für die Erstellung einer "Product Map", die eine Zuordnung der Merkmale zu den Produkten und zu den Domänen darstellt.

Das Ziel des "Product Line Assessment" ist, das Prozessfamilienpotenzial der identifizierten Domänen und der gesamten Prozessfamilie zu bewerten. Ausgangspunkt dafür sind die "Product Map", die Domänenbeschreibungen und die Produktbeschreibungen. Das Ergebnis ist die Bewertung der Domänen bzw.

der Prozessfamilie hinsichtlich ihres Potentials auf einer Skala von „teilweise“ bis „fortgeschritten“.

3.1.2 Stakeholder-basiertes Scoping prozessorientierter Software-Produkte

Im Folgenden wird eine Technik für das Scoping für prozessorientierte Software-Produkte vorgestellt. Für weitere Details siehe [BBG05].

Ein wesentliches Konzeptelement der Software-Produktlinien-Ansätze, wie sie von SEI [SEI04] oder IESE [Baye+99] erarbeitet wurden, ist das Scoping. Es bezeichnet eine Tätigkeit, die systematisch die Fachleute bei der Scope-Definition unterstützt. Scope-Definition bedeutet, die Grenzen der Plattform (einer Produktfamilie) zu definieren, d.h. festzulegen welche Elemente zum Verbund gehören und welche Elemente außerhalb der Produktfamilie stehen. Um "Ecomonies of Scope" bei der Software-Entwicklung erzielen zu können, muss man systematisch die Elemente der Wiederverwendung identifizieren und planen.

Bei der Software-Entwicklung entscheidet die Software-Architektur über die Elemente eines Software-Produktes, und der Entwicklungsprozess über das wirtschaftliche Fertigungsverfahren. Das Fertigungsverfahren ist durch die Anwendung der Produktlinien-Entwicklung festgelegt. D.h. beim Scoping konzentriert man sich auf die Identifikation der für die Software-Architektur relevanten Funktionalität, die nachweislich die firmenspezifischen Geschäftsziele unterstützt, wenn man sie wieder verwendet [Schm02].

Scoping unterstützt die Planung der für einen definierten Zielmarkt gewünschten Produkte, welche dann effizient in einer Produktlinie technisch umgesetzt werden. Die Ziele des Scopings werden in drei Kategorien gegliedert, die Identifikation des Scoping-Raumes, die Entscheidung über die im Scope zugelassenen Produktmerkmale (Entwicklung einer Produkt-Merkmal-Matrix) und die Evaluierung dieser Entscheidung. Die Scoping-Ziele werden mit Hilfe der Scoping-Objekte operationalisiert.

Scoping-Aktivitäten

Im Rahmen des PESOA- Projektes wurden folgende drei Scoping-Objekte für Software-Produktlinien untersucht. Das sind die Geschäftsziele der Stakeholder, die Prozesse und die Software-Produkte des Zielmarktes. Im PESOA-Projekt stehen die Prozesse im Vordergrund. Daher wurde die Betrachtung der Geschäftsprozesse in die Scoping-Aktivitäten eingearbeitet. Die Planung der Produktlinie baut auf diesen Objekten auf. Die systematische Unterstützung bei der Identifikation der Geschäftsziele, der Prozesse sowie der Produktmerkmale und die anschließende Entwicklung der Produkt-Prozess- und der Produkt-Merkmal-Matrix sind die Ziele der Scoping- Aktivitäten, wie in Tabelle 1 vorgestellt. Diese Aktivitäten bauen auf den Arbeiten von *Schmid* [Schm02] auf. Das Scoping wurde

um die Betrachtung der Geschäftsziele der Stakeholder und der Prozesse im Zielmarkt ergänzt und in den PESOA- Fachberichten TR 13-2004 und TR 16-2005 ausführlich erläutert.

Tabelle 1: Scoping- Aktivitäten

Scoping- Aktivität:	Kurze Beschreibung der Aktivität
1. Stakeholderanalyse	- Identifikation der Geschäftsziele der Stakeholder
2. Prozessanalyse	- Identifikation potenzieller Produkte für diesen Zielmarkt - Identifikation charakteristischer Prozesse der Produkte im Zielmarkt - Entwicklung einer Produkt-Prozess-Matrix
3. Produktanalyse	- Identifikation der gemeinsamen und variablen Merkmale der Produkte - Entwicklung einer Produkt-Merkmal-Matrix
3. Produktlinienplanung	- Evaluieren der Produktmerkmale und der Prozesse gegenüber den Stakeholder-Zielen

Evaluierung der Produktmerkmale der Produktlinie

Bei der Planung der Software-Produktlinie erfolgt die Evaluierung der identifizierten Produktmerkmale gegen die Geschäftsziele der Stakeholder. Das Vorgehen nach *Schmid* wendet ein Näherungsverfahren zur Nutzenanalyse in Anlehnung an das Goal/Question/Metrik-Verfahren an. Im Fokus des PESOA-Projektes stehen die Prozesse und daher wurden die Geschäftsprozesse in diese Evaluierung eingearbeitet. Die Evaluierung erfolgt zweistufig. In der ersten Stufe werden die identifizierten Geschäftsprozesse der Software-Produktlinie gegenüber den Geschäftszielen der Stakeholder evaluiert. Anschließend werden die Merkmale der geplanten Produktlinie gegenüber den Prozessen, ob ein Merkmal einen messbaren Anteil zum Geschäftsprozess beiträgt, evaluiert. In dem Fall, dass kein messbarer Beitrag eines Merkmals zu einem Prozess gemessen werden konnte, kann das Merkmal gegenüber den Geschäftszielen evaluiert werden. Wird hier ebenfalls ein negativer Beitrag ermittelt, so kann das Merkmal aus der Software-Produktlinie ausgesondert werden.

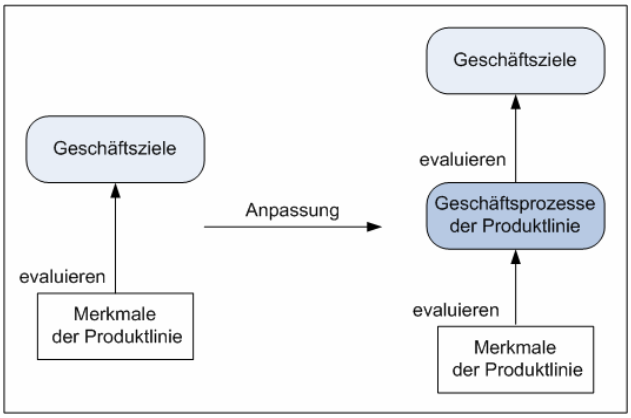

Abbildung 4: Evaluierung der Merkmale der Produktlinie

3.2 Aktivität "Model Features"

Das Ziel dieser Aktivität ist, die Merkmale ("Features") der Produkte, die für die Prozessfamilieninfrastruktur identifiziert wurden, zu modellieren und zu dokumentieren. Als Input dient die Scope-Definition, die die Merkmale enthält. Das Resultat ist ein Modell, das die Merkmale und ihre Beziehungen darstellt und in dem variable Merkmale identifiziert und gekennzeichnet werden.

Im Folgenden werden zwei Techniken zur Modellierung von Merkmalen dargestellt.

3.2.1 Merkmalbasierte Modellierung

Ziel der Merkmalmodellierung ist die Darstellung und Modellierung von Variabilitäten und Abhängigkeiten in der betrachteten Produktfamilie. Die variablen Eigenschaften der Produktfamilie werden über Merkmale beschrieben; Abhängigkeiten zwischen Merkmalen werden durch Constraints dargestellt.

Die Merkmalmodellierung beeinflusst die Konzeptanalyse und das Anwendungs-Engineering. Zu wissen, welche Merkmale innerhalb der Produktfamilie verfügbar sind, kann den Kunden bei seiner Entscheidung, welche Merkmale in seinem Produkt verfügbar sein sollen, unterstützen. Das Wissen, welche der gewünschten Merkmale die Produktfamilie zur Verfügung stellt und welche Merkmale kundenspezifisch entwickelt werden müssen, hilft zudem eine bessere Abschätzung bezüglich Zeit und Entwicklungskosten durchzuführen.

Bei der Merkmalmodellierung eignet sich die hierarchische Anordnung der Merkmale [BFK05]. Ähnliche Merkmale, die dasselbe Konzept repräsentieren, werden zusammengefasst. Hierbei wird das Konzept als abstraktes Merkmal

modelliert und die Spezialisierungen als Child-Merkmale. Das Merkmalmodell (engl. "feature model") stellt hierbei keine Hierarchie von Funktionsaufrufen dar. Es beschreibt lediglich welche Arten von Variabilität in der Domäne existieren. Die folgenden Schritte helfen Merkmale zu organisieren:

a) Modellierung der Merkmale derselben Spezialisierung in hierarchischer Form auf derselben Ebene.

b) Festlegen des Merkmal-Typs – "mandatory", "optional", oder "alternative".

c) Definition der Zwangsbedingungen – wie "requires", oder "mutual exclude" – zwischen Merkmalen.

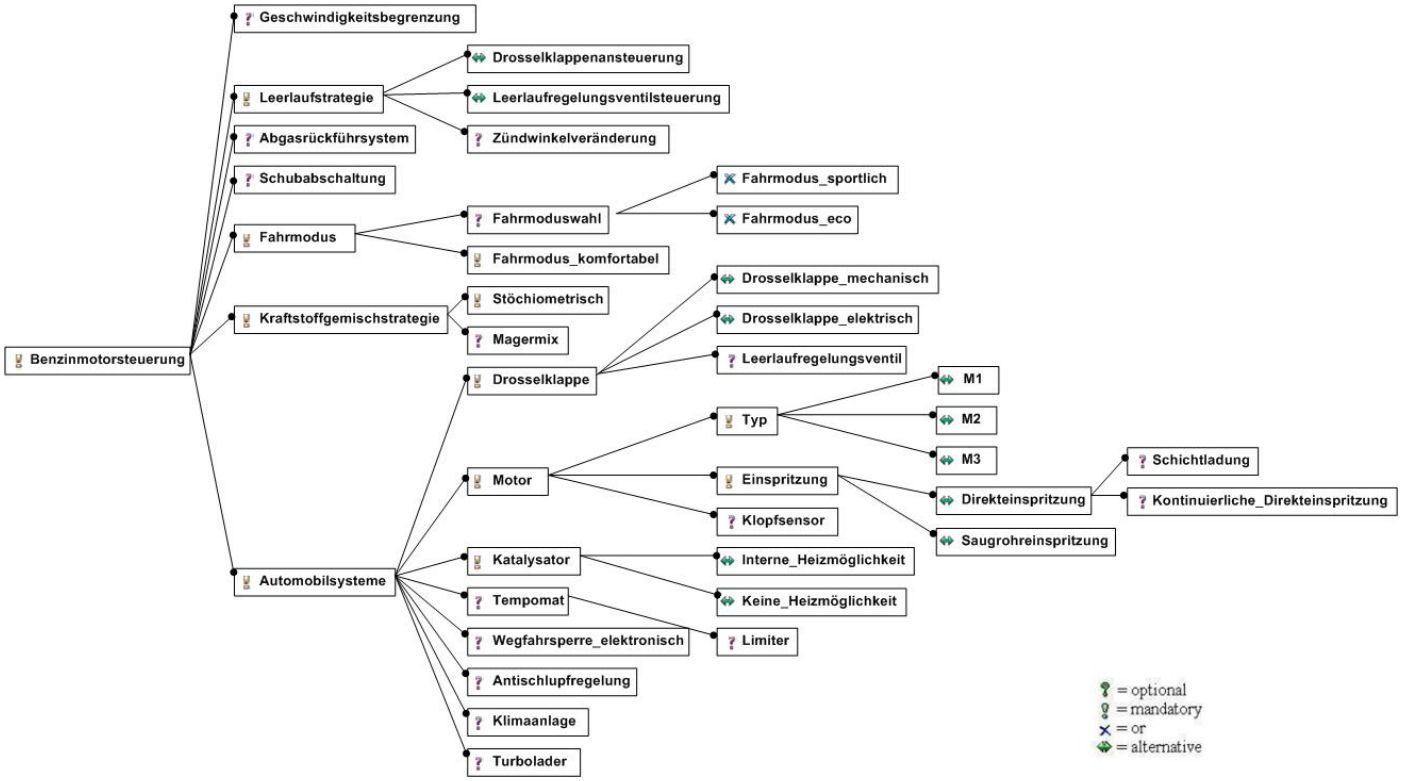

Abbildung 5: Beispiel des Merkmaldiagramms einer Benzinmotorsteuerung

Neben dem eindeutigen Identifier und dem Merkmal-Typ können Merkmale durch zusätzliche Informationen, wie eine Beschreibung des Merkmales, Attribut-Werte-Paare, Bindezeiten und -modi, Prioritäten, etc. charakterisiert werden [CE00].

Es existieren verschiedene Möglichkeiten Merkmale zu gruppieren, z.B. beschrieben in [CE00], [Ka03]. Nach [CE00] werden Merkmale in die folgenden Kategorien eingeteilt: *Context features* beschreiben im Wesentlichen nicht-

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

3.2.2 Merkmalmodellierung im Entscheidungsmodell

Merkmalmodelle werden in PuLSE nicht als eigenständige Modelle verwendet. Vielmehr werden die Merkmale von Produkten und die Beziehungen zwischen diesen Merkmalen in Entscheidungsmodellen dokumentiert, die außerdem noch das Konfigurationsmodell (engl. "Configuration Model") beinhalten. Die Produkte und ihre Merkmale werden während der Scoping-Aktivität identifiziert und in einer "Product Map" dokumentiert (siehe Abschnitt 3.1.1). Die Information in der "Product Map" dient als Ausgangspunkt zur Merkmalmodellierung im Entscheidungsmodell (engl. "Decision Model"). Ein Entscheidungsmodell beinhaltet auf der einen Seite die Produktmerkmale und auf der anderen Seite deren Auswirkungen auf die Prozessfamilieninfrastruktur (dieser Aspekt von Entscheidungsmodellen wird im Abschnitt 3.5 erläutert).

Entscheidungsmodelle erfassen verschiedene Beziehungen zwischen Produktmerkmalen, um zum Beispiel Hierarchien von Merkmalen aufzubauen oder um auszudrücken, dass ein Merkmal ein anderes bedingt bzw. ausschließt. Es können aber auch allgemein logische Verknüpfungen verwendet werden, um die Beziehungen zwischen Merkmalen zu dokumentieren. Die dokumentierten Beziehungen in Entscheidungsmodellen dienen zur Identifikation gültiger Konfigurationen der Prozessfamilieninfrastruktur. Eine ausführlichere Beschreibung der Merkmalmodellierung findet sich in [BFK05].

3.3 Aktivität "Identify Processes"

Das Ziel dieser Aktivität ist die Anforderungen der Prozesse, die innerhalb der identifizierten Produkte ablaufen, zu definieren. Als Input dienen die Merkmalmodelle. Das Resultat ist eine Beschreibung der variablen und gemeinsamen Prozessanforderungen.

3.3.1 Von Anwendungsfalldiagrammen zu abstrakten Prozessen (E-Business)

E-Business-Systeme bestehen aus Stakeholdern und Geschäftsprozessen. Aktiv an den Geschäftsprozessen beteiligte Stakeholder werden als Aktoren bezeichnet. Die Modellierung von E-Business-Systemen beginnt dann mit der Modellierung der Anwendungsfalldiagramme (engl. "use case diagrams"), welche Ziele von Geschäftsprozessen (Anwendungsfälle) auf bestimmte Aktoren verteilt. Die

Anwendungsfälle werden in einem weiteren Schritt zu abstrakten, d.h. Rahmen-, Geschäftsprozessen erweitert.

Basierend auf der Scoping-Phase werden zur Erstellung von Anwendungsfalldiagrammen zunächst diejenigen Stakeholder ermittelt, welche aktiv an den Prozessen beteiligt sind. Diese werden gemäß der UML-Spezifikation als Aktoren bezeichnet. Sodann müssen, basierend auf den enthaltenen Merkmalen einer Software-Produktlinie, Anwendungsfälle definiert werden welche die Merkmale implementieren. Die Erstellung der Anwendungsfalldiagramme wird durch die Zuordnung von Aktoren zu Anwendungsfällen abgeschlossen. Es kann für jede Produktvariante ein eigenes Anwendungsfalldiagramm erstellt werden, oder mehrere Produktvarianten können, entsprechend gekennzeichnet, in einem Anwendungsfalldiagramm dargestellt werden. Das Ergebnis dieses Arbeitschrittes ist in Abbildung 6 abstrakt dargestellt.

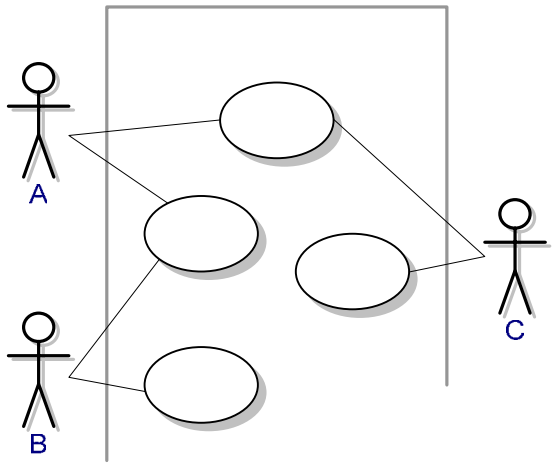

Abbildung 6: Anwendungsfalldiagramm

Im nächsten Arbeitsschritt werden die Anwendungsfalldiagramme auf abstrakte Prozesse abgebildet. Die Abbildung für die Domäne E-Business erfolgt manuell in folgenden Schritten:

1. Alle Aktoren des Anwendungsfalldiagrammes werden als Pools oder Lanes gemäß der BPMN-Spezifikation dargestellt.
2. Anwendungsfälle, welche nur von einem Aktor implementiert werden, werden als einfache Prozessteile in dem zum Aktor gehörigen Pool/ der Lane dargestellt.
3. Anwendungsfälle, welche von mehreren Aktoren desselben Pools implementiert werden, werden als Lane-übergreifende Prozessteile in dem zu den Aktoren gehörigen Pool dargestellt.

4. Anwendungsfälle, welche von mehreren Aktoren verschiedener Pools implementiert werden, werden als verteilte Prozessteile in den jeweiligen Pools der beteiligten Aktoren dargestellt. Die Prozessteile müssen durch Nachrichtenfluss verbunden werden.

Eine schematische Darstellung ist in Abbildung 7 enthalten.

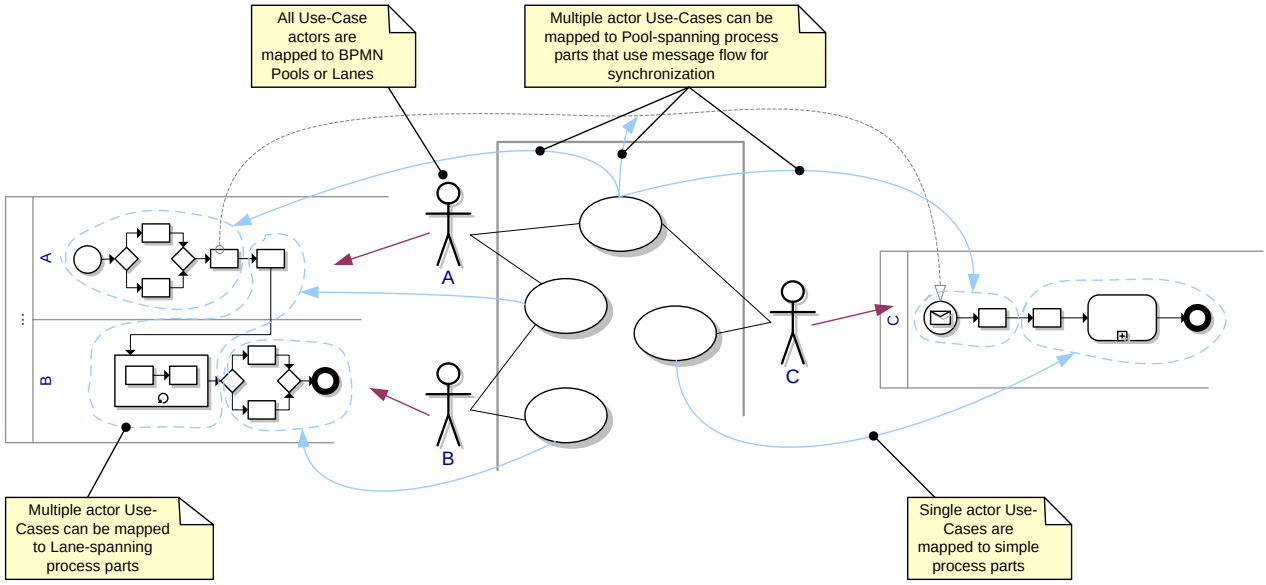

Abbildung 7: Erstellung von abstrakten Prozessmodellen für die Domäne E-Business

Die Integration der nunmehr vorliegenden abstrakten Prozesse zu einem variantenreichen Prozessmodell wird im Abschnitt 3.4 betrachtet.

3.3.2 Modellierung der Systemumgebung (Automotive)

Jedes System kann als Netzwerk aus Prozessen und gespeicherten Daten beschrieben werden. Während der Analyse werden diese Prozesse und gespeicherten Daten aus Sicht der späteren Anwendung strukturiert. Eine mögliche Vorgehensweise ist hierbei die „Outside-In"-Modellierung: Alles was im Modell / im System geschieht beruht auf Einflüssen außerhalb des Modells. Daher beginnen wir mit der Modellierung der Systemumgebung. Auf dieser beruht die anschließende Verhaltensmodellierung unseres Systems.

Die hier beschriebene Vorgehensweise ist Teil der strukturierten Analyse. Diese eignet sich besonders bei der funktional-orientierten Modellierung von "embedded"-Software.

Das Umgebungsmodell definiert den Zweck und die Schnittstellen des Systems. Es besteht aus Kontextmodellen und der Ereignisliste (engl. "event list"). Kontextmodelle beschreiben das System, die Umgebung und die gegenseitigen Wechselwirkungen mit der Umgebung. Sie positionieren das System relativ zu übergeordneten, untergeordneten und gleichgeordneten Domänen. Es werden wichtige Konzepte außerhalb des Systems, die das System beeinflussen oder vom System beeinflusst werden, betrachtet. Verbindungen zum System beschreiben mögliche Datenflüsse. Kontextmodelle werden in [RSW05] am Beispiel der Benzinmotorsteuerung dargestellt (vgl. auch Abbildung 8).

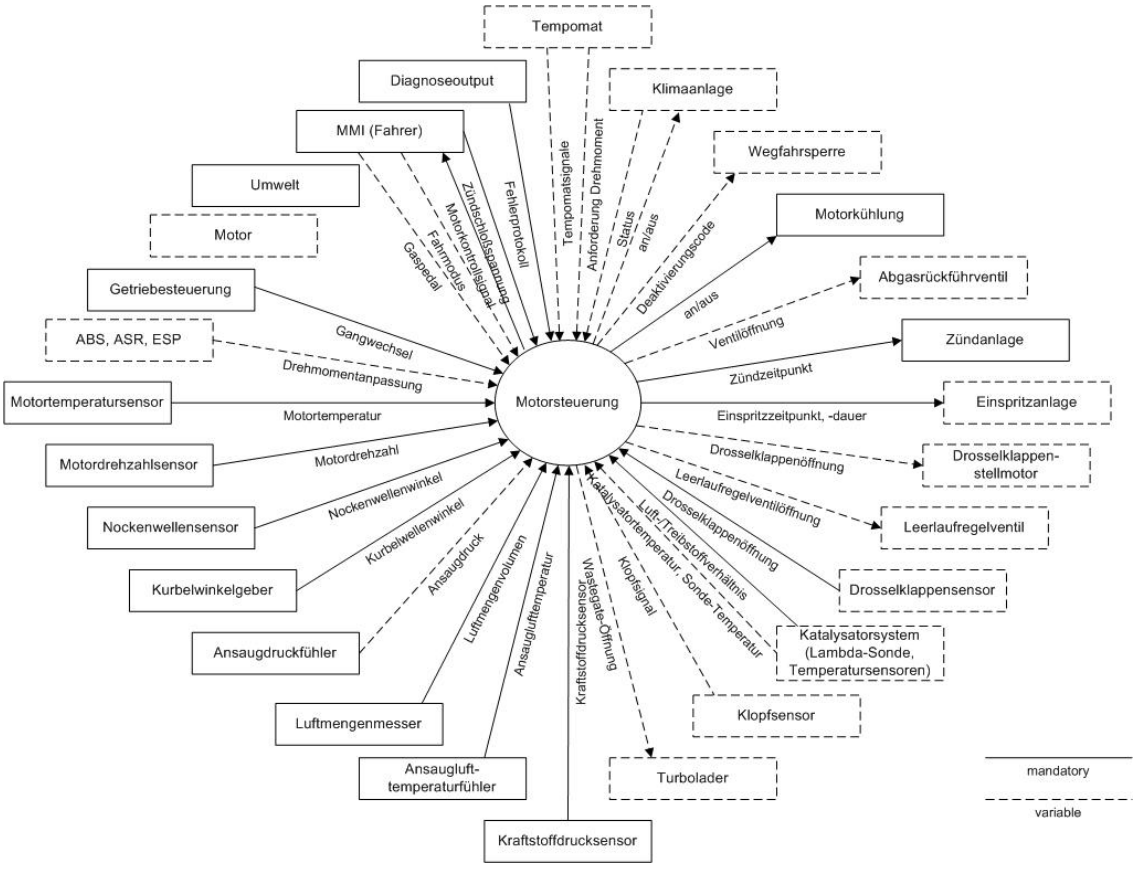

Abbildung 8: Kontextdiagramm der Benzinmotorsteuerung

Über die Ereignisliste – vergleichbar mit "Use Cases" aus der objekt-orientierten Analyse – werden die funktionalen Anforderungen an das System erfasst. Die Ereignisliste dokumentiert das System als "Black Box" mit einer Reihe von Inputs (Ereignisse) und Outputs (Antworten). Sie ist eine Tabelle, welche Ereignisse, Antworten (die Reaktionen auf ein Ereignis) und die Klassifikation eines Ereignisses in informeller textueller Form enthält (siehe auch Abbildung 9). Antworten des Systems werden nur durch Impulse aus der Systemumgebung veranlasst.

Bei den Ereignissen handelt es sich um externe Ereignisse aus der Systemumgebung, die bestimmten Bedingungen genügen, wie

- das Ereignis muss in der Systemumgebung auftreten,
- das System muss auf das Ereignis reagieren und
- das System muss in der Lage sein zu erkennen, wann exakt das Ereignis auftrat.

Nr.	2
Ereignis	Tempomat-Hebel „Pull“ betätigt
Antwort	- Aktivieren des Tempomaten
Klassifikation	- – / C
V(in)	-
V(out)	- Tempomat-Signal (= on - Drehmomentregelung über Tempomat-Sensor)
Zustand(in)	- Tempomat off
Zustand(out)	- Tempomat on
Bedingungen	- Geschwindigkeitssollwert > $v_{MinTempomat}$
Beschreibung	- Gespeicherter Geschwindigkeitssollwert wird abgerufen

Abbildung 9: Beispiel eines Ereignisses aus einer Ereignisliste

Die Antwort beschreibt, was das System machen soll, wenn ein spezifisches Ereignis auftritt. Wenn die Ereignisliste vollständig ausgefüllt ist, beschreibt die Spalte mit den Systemantworten die Gesamtfunktionalität des Systems. Die Klassifikation von Ereignissen hilft bei der Entscheidung, wie das System letztendlich die Antwort zur Verfügung stellt, d.h. letztendlich auch, wie es modelliert wird (d.h. im Datenflussdiagramm, Zustandsautomaten, ER-Diagramm). Unterschieden werden hier temporale (zyklische) und normale Ereignisse, sowie Daten-, Kontroll- bzw. Daten-/Kontroll-Ereignisse.
Für die Systemfamilie müssen nun die externen Ereignisse, die sämtliche Varianten der Systemfamilie umfassen, erfasst werden. Um die Ereignisliste übersichtlich zu halten ist es sinnvoll durch Gruppieren von Ereignissen die Ereignisliste in funktionale Bereiche zu partitionieren (siehe auch [RSW05]).

3.4 Aktivität "Design Processes"

Das Ziel dieser Aktivität ist die Erstellung von Prozessmodellen, die die Produkte der Prozessfamilie beschreiben. Ausgangspunkt für diese Aktivität sind zum einen das Anforderungsdokument und zum anderen das Merkmalmodell. Das Ergebnis ist ein generisches Prozessmodell, das variantenreiche Prozesse beinhaltet, in denen Variabilitäten mit Hilfe von Variationspunkten explizit dokumentiert sind.
Im Folgenden werden Techniken zur Modellierung variantenreicher Prozesse in UML-Aktivitätsdiagrammen (engl. "Activity Diagrams") und –Zustandsautomaten (engl. "State Machines"), BPMN und Matlab/Simulink vorgestellt.

3.4.1 Modellierung variantenreicher Prozesse in UML und BPMN

Abbildung 10 zeigt wie die verschiedenen Konzepte des konzeptionellen Modells auf die entsprechenden Elemente der Prozessmodellierungssprachen der PESOA-Anwendungsdomänen abgebildet werden. Die Abbildung wird im Detail in [BBG05] beschrieben.

	PESOA Conceptual Model	**UML Activity Diagrams**	**UML State Machines**	**BPMN**
Process	Process	Activity	State machine	Process, Pool (for abstract processes)
	Composite Activity	Activity	Composite states, submachine states	Sub-Process
	Activity	Action	Action	Task
Control Flow	Sequence	Control flow edge	Transition	Sequence Flow
	Parallel	Fork node / join node	Fork pseudo state, join pseudo state	AND Gateway
	Choice	Decision node /merge node	Choice pseudo state, junction pseudo state	XOR/OR Gateway
	Iteration	Reentering arc, loop node, expansion region	Reentering transitions	Iteration Marker / Sequence Flow
Data Flow	Input	Input pin, activity parameter node	Only for Activities	InputSets, Inputs Attributes
	Output	Output pin, activity parameter node	Only for Activities	OutputSets, Output Attributes
	DataSource	Action, activity, central buffer node, data store node	Only for Activities	(Abstract) Processes
	DataSink	Action, activity, central buffer node, data store node	Only for Activities	(Abstract) Processes
Environment	State	Token distribution during execution + variable values	Simple state, composite state, submachine state	Status Attribute of Process
	Variable	In guards, decision nodes, selection behaviors, local preconditions, local postconditions, parameter names	Input and output parameters of activities and transition guards	Data Object
	Scope	Classifier	Classifier	Process, Sub-Process, Task, Group
	Event	Send signal actions, accept event actions	Several types of triggers	Event
	Exception	Exception handlers, is-exception pins	Not supported	Error Event
NFP	Realtime	Comments, accept time event actions	Comments, time triggers	-
	Costs	Comments	Comments	Associations, Text Annotations
	Security, etc.	Represented by the process structure	Represented by the process structure	Represented by the process structure

Abbildung 10: Abbildung der PESOA-Konzepte auf domänenspezifische Sprachen

Die UML Modellelemente, auf die die Konzepte des konzeptionellen PESOA-Modells abgebildet werden, sind in UML-Büchern [Pen03, RJB04] sowie der UML-Spezifikation [UML03] dokumentiert und illustriert.

Abbildung 11 zeigt ein Beispiel für die Umsetzung einiger der wichtigsten Konzepte des konzeptionellen Modells in UML-Aktivitätsdiagramme. Der dargestellte Flug-Prozess wird durch einen "Initial Node" gestartet. Zunächst wird die "action" "Buy Tickets" ausgeführt. Danach, in Abhängigkeit vom Tickettyp über den der Passagier verfügt, kann einer von zwei möglichen alternativen Prozesspfaden gewählt werden. Die Entscheidung darüber welcher der beiden Pfade genommen wird, wird am "Decision Node" getroffen und hängt vom Wert der "Ticket.class"-Variablen ab. Die beiden alternativen Pfade werden im "Merge Node" wieder zusammengeführt. Eine "Accept Time Event" "action" initiiert das nachfolgende "Boarding". Nach dem "Boarding" kann der Passagier gleichzeitig essen und Filme ansehen. Die beiden parallelen Aktivitäten sind durch einen "Fork" und "Join Node" umschlossen. Nachdem der Passagier ausgestiegen ist, wird der Flug-Prozess durch erreichen des "Final Node" terminiert.

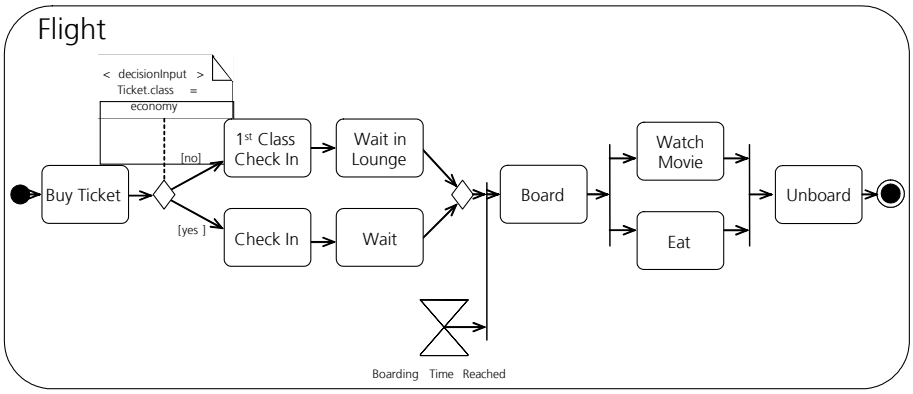

Abbildung 11: Flug-Prozess der die Umsetzung einiger wichtiger PESOA-Konzepte in UML-Aktivitätsdiagrammen zusammenfasst

Abbildung 12 zeigt ein Beispiel für die Darstellung einiger wichtiger PESOA-Konzepte in UML-Zustandsautomaten. Der Zustandsautomat enthält einige "simple states" wie zum Beispiel "Error" und "Startup" sowie einen "submachine state" "Stop".

Ein "Choice" "pseudo state" erlaubt alternativ das Erreichen des Zustands "Error" oder des Zustands "Running" nach dem Zustand "Start". Welcher dieser beiden Zustände erreicht wird, hängt von der Auswertung der Variablen "Rotation Speed" ab, die in dem Guard-Ausdruck der Transition abgefragt wird. Der Zustand "Shutdown" zeigt ein Beispiel für die Integration von "actions" in Zustandsautomaten. Die "action" "prepare engine to stop" wird ausgeführt, sobald der Zustand "Startup" betreten wird, während die "actions" "stop engine" und "controlled termination of processes" bei Verlassen des Zustands "Startup" ausgeführt werden.

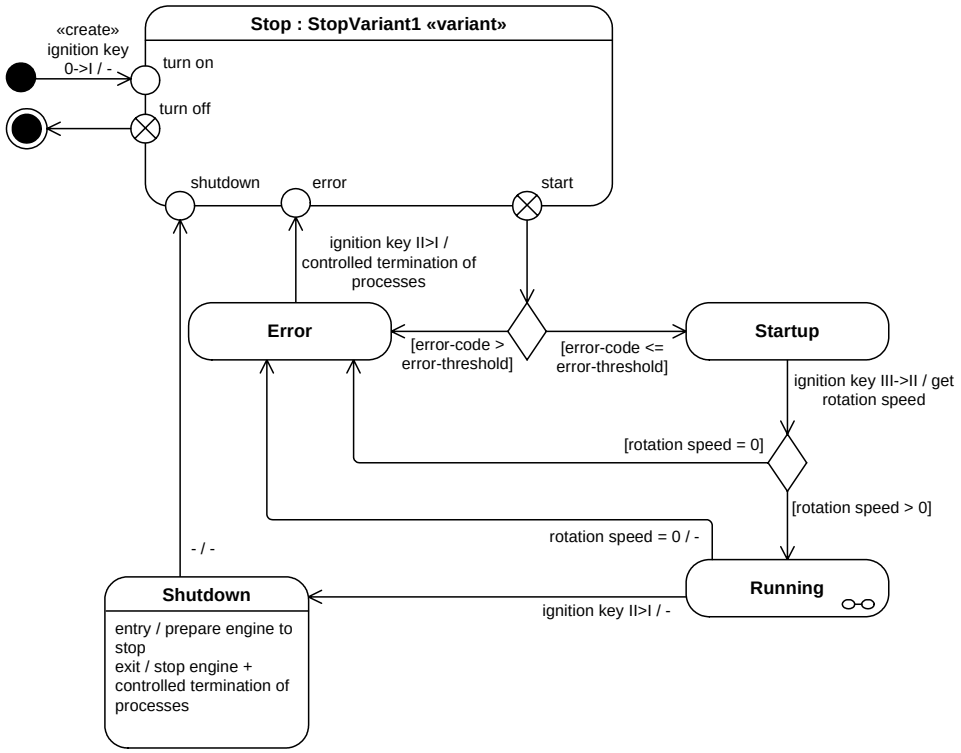

Abbildung 12: Motorsteuerungsprozess der die Umsetzung einiger wichtiger PESOA-Konzepte in UML-Zustandsautomaten zusammenfasst

Abbildung 13 zeigt ein Beispiel für die Abbildung der PESOA-Konzepte in der "Business Process Modeling Notation" (BPMN). Der Beispielprozess beschreibt den Workflow einer kleinen Software-Firma namens SoftMax Inc. SoftMax erhält Bestellungen für ihre Produkte über HTTP-Web-Formulare.

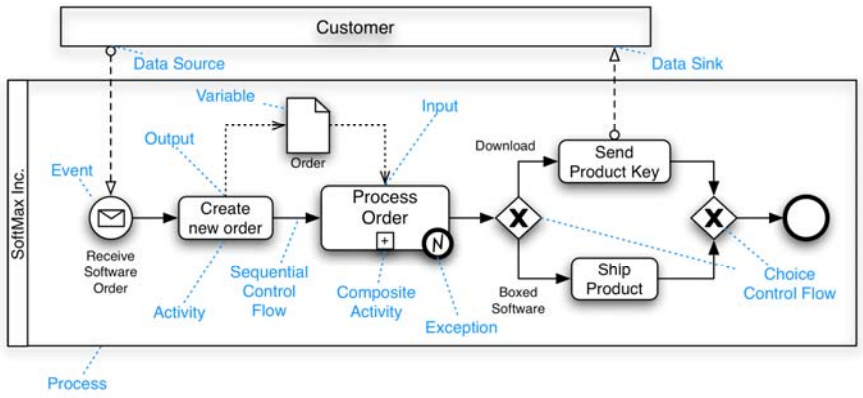

Abbildung 13: Ein einfacher Online-Bestellprozess der die Umsetzung der wichtigsten PESOA-Konzepte in BPMN zusammenfasst

Nachdem eine neue Bestellung empfangen wurde, wird zunächst ein interner Auftrag erzeugt. Der Auftrag wird dann bearbeitet, was u.a. die Erstellung einer Rechnung beinhaltet. Abschließend wird geprüft, ob der Kunde eine CD-Version mit gedruckter Anleitung oder einen Software-Download erhält. Im Falle der CD-Version wird das Produkt versandt, während bei der Download-Version ein Produkt-Schlüssel per E-Mail geschickt wird. Die entsprechenden PESOA-Konzepte sind im Beispiel annotiert.

3.4.2 Konzepte zur Modellierung von Variabilität in UML und BPMN

Für die Darstellung von Varianten in einem Prozessmodell müssen die folgenden Informationen berücksichtigt werden:

- Die Punkte im Prozessmodell, wo Variabilität auftaucht, müssen als Variationspunkte identifiziert und gekennzeichnet werden.
- Es muss eindeutig ersichtlich werden welche Varianten, unter Berücksichtigung der Produkteigenschaften, an einen bestimmten Variationspunkt gebunden werden können.
- Die Variabilitätsmechanismen, welche zur Bindung von Varianten an Variationspunkten verwendet werden, müssen ersichtlich sein.

Variationspunkte in UML-Zustandsautomaten und UML-Aktivitätsdiagrammen werden durch den Stereotyp «VarPoint» identifiziert. Varianten sind mit einem Variationspunkt durch UML-"Dependency Relations" verknüpft, die als Stereotypen den Namen des Variabilitätsmechanismus enthalten, der als Technik zur Realisierung der Variabilität verwendet werden soll. Eine Zuordnung von Stereotypnamen zu den jeweiligen Variabilitätsmechanismen findet sich in [PSW05]. Jede Variante verfügt über einen Stereotyp «Variant» und einen "Tagged Value" "feature", der eine (1-n elementige) Liste von Produktmerkmalen enthält, der die Variante den Produktmerkmalen zuordnet die durch die Variante (teilweise) umgesetzt werden. Der Bindezeitpunkt kann dem Stereotyp für den Variabilitätsmechanismus mittels eines "Tagged Value" (Schlüssel "*bt*") zugeordnet werden. Des weiteren kann ein Variabilitätsmechanismus über einen zusätzlichen "Tagged Value" (Schlüssel "*id*") verfügen, der zur eindeutigen Identifikation eines implementierenden Variabilitätsmechanismus dienen kann. Des weiteren führen wir einen Stereotyp «Variable» ein, der verwendet werden kann, um auszudrücken, dass ein Modellelement von Variabilität betroffen ist, ohne dass es sich dabei um einen konkreten Variationspunkt handelt.

Variationspunkte und Varianten in BPMN-Prozessmodellen werden durch eine Adaptierung des Stereotyp-Konzeptes der UML dargestellt. Zum Zwecke der Darstellung eines variantenreichen Geschäftsprozessmodells genügt die Einführung von zwei Stereotypen: «VarPoint» und «Variant». Der zweite Stereotyp kann ggf. grafisch als Puzzleteilmarkierung am unteren Rand eines Arbeits-

schrittes dargestellt werden. Beide Stereotypen können, im Einklang mit der UML-Spezifikation, mit Attributen verfeinert werden. Der Stereotyp «VarPoint» besitzt das vordefinierte Attribut "type". Das "type"-Attribut kennzeichnet, welche Art von Variationspunkt dargestellt wird. Die Werte des "type"-Attributes sind "optional", "alternative", "abstract" sowie "null". Die vier Ausprägungen des Stereotyps «VarPoint», welche als Attribute gekennzeichnet werden, können zur Einsparung von Bildschirm- und Papierfläche, gleichfalls als eigenständige Stereotypen dargestellt werden. Die korrespondierenden Stereotypen sind «Optional», «Alternative», «Abstract» sowie «Null». Der Stereotype «Variant» besitzt ein Attribut "feature", welches die Variante mit der Produkteigenschaft verbindet die (teilweise) durch die Variante implementiert wird.

Für UML-Zustandsautomaten, -Aktivitätsdiagramme und BPMN wurden Variabilitätsmechanismen definiert [PSW05], die eine flexible Modellierung von Variabilität in Prozessmodellen erlauben.

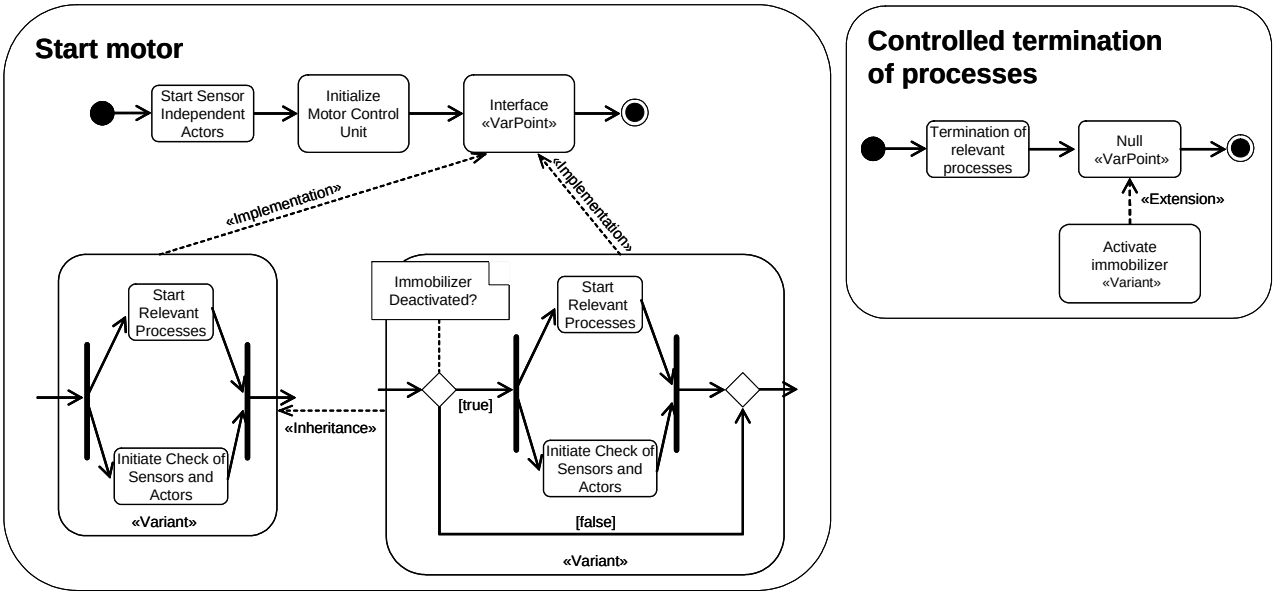

Abbildung 14: Motorsteuerungsprozess mit Variabilität, die abhängig von dem Vorhandensein einer elektronischen Wegfahrsperre ist

Abbildung 14 zeigt beispielhaft einen Ausschnitt eines Motorsteuerungsprozesses, der die Repräsentierung von Variabilität in Aktivitätsdiagrammen darstellt. Um die zur Steuerung einer elektronischen Wegfahrsperre benötigten Prozesse zu integrieren, müssen die "actions" "start motor" und "controlled termination of processes" entsprechend konfiguriert werden. Zur optionalen Reaktivierung der elektronischen Wegfahrsperre während des Abschaltens der Motorsteuerungseinheit in der "controlled termination of processes" "action" wird der Variabilitätsmechanismus "Extension" (Stereotyp «Extension») verwendet.

Um die benötigte Variante des "start motor"-Subprozesses zu erhalten, werden die Variabilitätsmechanismen "Activity Diagram Inheritance" (Stereotyp «Inheritance») und "Encapsulation of Varying Subprocesses" (Stereotyp «Implementation») verwendet.

Abbildung 16 zeigt anhand eines Motorsteuerungsprozesses mit optionaler Wegfahrsperre die Darstellung von Variabilität in UML-Zustandsautomaten. Es gibt eine "Submachine"-Variante für einen Motorsteuerungsprozess ohne Wegfahrsperre (StopVariant1) und eine "Submachine"-Variante für einen Motorsteuerungsprozess mit Wegfahrsperre (StopVariant2). Je nachdem welche Variante umgesetzt werden soll, wird vom "Submachine"-Zustand "Stop" die eine oder andere "Submachine" aufgerufen. Es handelt sich hierbei um eine Demonstration der Anwendung des Variabilitätsmechanismus "Encapsulation of Varying Subprocesses".

Abbildung 15 zeigt einen einfachen Online-Shop in BPMN, wobei optionale Komponenten als Erweiterungspunkte gekennzeichnet sind.

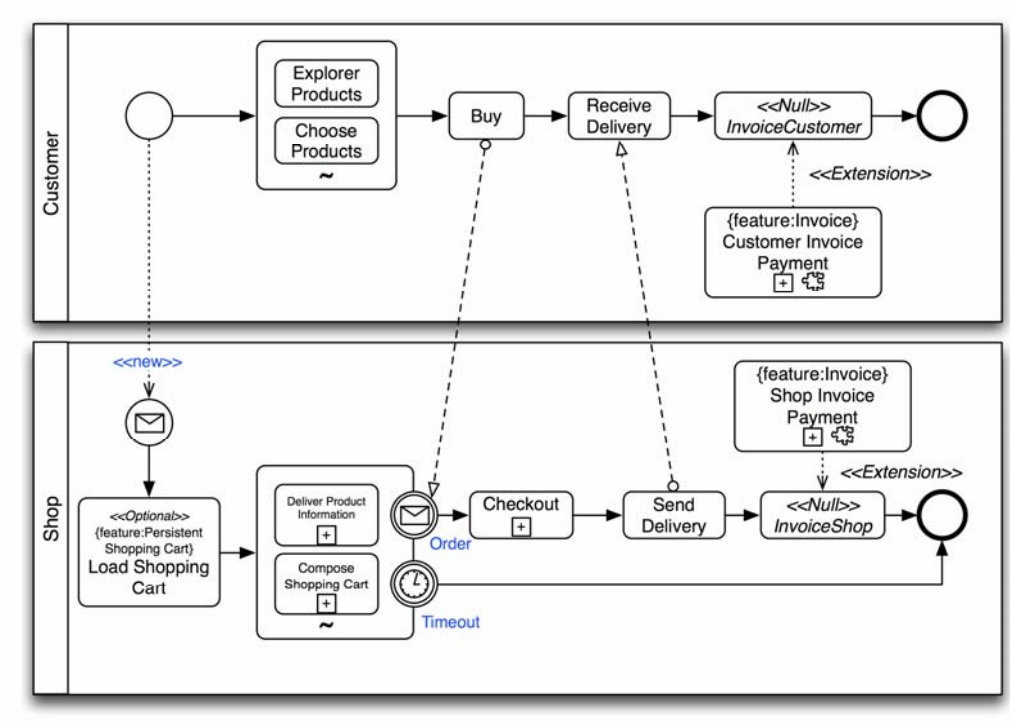

Abbildung 15: Einfacher Online-Shop mit Varianten, abgeleitet durch Erweiterungspunkte

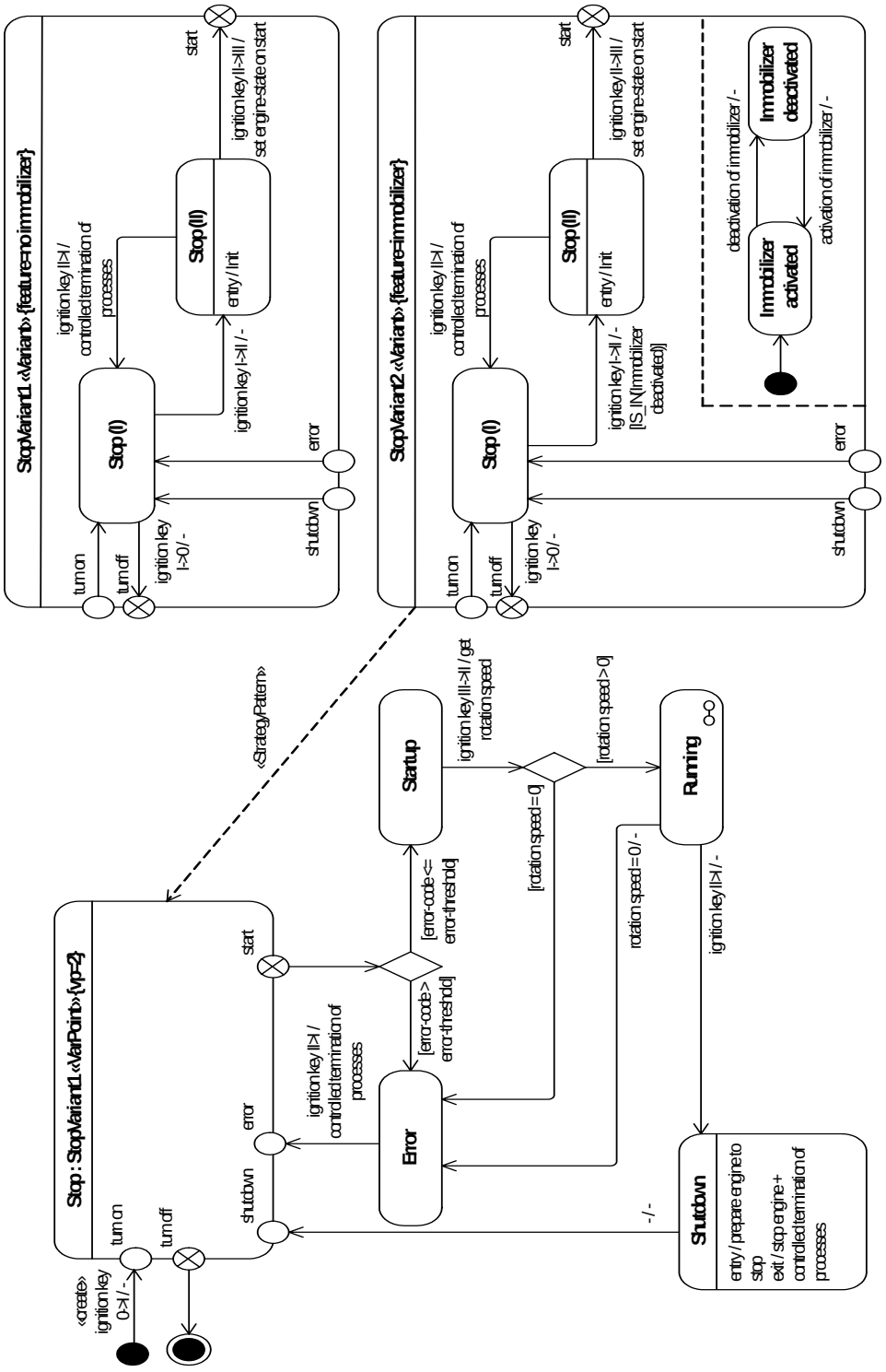

Abbildung 16: Variantenreicher Motorsteuerungsprozess für Variante mit und ohne elektronische Wegfahrsperre

3.4.3 Modellierung variantenreicher Prozesse in Simulink

Zur Modellierung von Variabilität stehen in Matlab/Simulink zwei Modellierungselemente zur Verfügung; beide Modellierungselemente sind Simulink-intrinsisch:

- Template-Blöcke ermöglichen das Ersetzen ganzer Simulink-Blöcke entsprechend der auftretenden (lokalen) Variabilität. Blockvarianten werden hierbei einem Template-Block als „Member" zugeordnet. Bei Verwendung von Template-Blöcken im Modell kann eine Auswahl einer gewünschten Variante erfolgen.
- Einsatz von (Block-)Parametern (direkt oder – für umfangreiche Datenmengen – über .mat-Dateien). Anwendungsfälle für Parameter sind z.B. die Einbindung von Varianten-spezifischen Kennlinien oder die statische Festlegung des Verzweigungsverhaltens in Kontrollflüssen (und somit innerhalb von Zustandsdiagrammen), d.h. eine Varianten-spezifische Steuerung.

Template-Blöcke und Parameter repräsentieren Variationspunkte und finden ihre Entsprechung implizit oder explizit im Merkmalmodell wieder.

Um die Kopplung zwischen dem Management der Variabilität und der Modellierung mit Simulink durchführen zu können, aber auch um ein vereinfachtes Suchen nach derartigen Blöcken zu ermöglichen, müssen sie in Simulink besonders gekennzeichnet werden. Die von uns gewählte Möglichkeit, beispielsweise um Template-Blöcke als Variationspunkte zu kennzeichnen, ist, diesen als "Tag" die Bezeichnung "VariationPoint::Name" anzufügen. Die Blockvarianten erhalten als "Tag" den Namen des korrespondierenden Merkmals in der Form "Feature::Featurename".

Unter Anwendung dieser Mechanismen zur Modellierung von Variabilität entsteht ein variantenreiches Simulink-Modell, welches auf die einzelnen Produktvarianten angewendet werden kann.

3.4.4 Entwurf von Prozessvarianten

Die grundlegende Idee in der Prozessfamilienentwicklung liegt in der Betrachtung von Gemeinsamkeiten und Unterschieden zwischen verwandten Produkten. Die Gemeinsamkeiten beschreiben die Merkmale, die alle Produkte einer Prozessfamilie besitzen; diese können für alle Prozessfamilienmitglieder wieder verwendet werden. Die Unterschiede zwischen den Prozessfamilienmitgliedern werden explizit dokumentiert und beschreiben, inwieweit sich die einzelnen Prozessfamilienmitglieder voneinander unterscheiden.

In PESOA werden Produkte mit Hilfe von Prozessen definiert. Es ist daher notwendig Gemeinsamkeiten und Unterschiede in Prozessen dokumentieren zu können. Wir verwenden einen Ansatz zur Einführung von Produktlinien in Organisationen, der es ermöglicht, beliebige Software-Engineering-Artefakte in generische Artefakte zu überführen, um Gemeinsamkeiten und Unterschiede explizit modellieren zu können [Mut02]. Das Ergebnis ist ein Modell für variantenreiche Prozesse, das es erlaubt Gemeinsamkeiten und Unterschiede in den jeweiligen Prozessen explizit zu machen und zu dokumentieren. Die Dokumentation von Unterschieden in den Prozessen wird dabei mit so genannten Variationspunkten realisiert, die Prozesselemente markieren, die für verschiedene Produkte verschiedene Eigenschaften haben.

Eine detaillierte Beschreibung des verwendeten Ansatzes, des resultierenden Modells für variantenreiche Prozesse, sowie Beispiele für die PESOA-Domänen finden sich in [BBG05].

3.5 Aktivität "Model Configurations"

Das Ziel dieser Aktivität ist es, Konfigurationen einer Prozessfamilie zu modellieren, indem die Verbindung zwischen den im Prozessmodell definierten Variationspunkten und den identifizierten Produktmerkmalen hergestellt wird. Dadurch ist es dann möglich, Aussagen über die Auswirkungen der Auswahl von Merkmalen auf die variantenreichen Prozessmodelle zu machen. Dies ist notwendig, um die Variationspunkte in den variantenreichen Prozessen auflösen zu können, um somit spezifische Prozesse für konkrete Produkte abzuleiten.

Im Folgenden werden zwei Techniken zur Modellierung von Konfigurationen vorgestellt.

3.5.1 Management des Konfigurationswissens

Über das Konfigurationswissen werden gültige Software-Produkte einer Familie spezifiziert. Das Konfigurationswissen definiert sowohl die Abhängigkeiten zwischen Merkmalen und variantenreichen Prozessen untereinander, als auch die Zuordnung von Merkmalen zu variantenreichen Prozessen.

Wie zur Merkmalmodellierung auch haben wir zur Definition und Verwaltung des Konfigurationswissens das Werkzeug pure::variants von pure-systems verwendet. Das Konfigurationswissen selbst haben wir mit Werkzeug-intrinsischen Konfigurationssprachen – "Configuration Domain Specific Languages" (Configuration DSLs) und "Implementation Component Configuration Languages" (ICCLs) – realisiert.

Über "Configuration DSLs" lassen sich im Merkmalmodell gültige Merkmalkombinationen definieren. "Configuration DSLs" beschreiben sowohl Abhängigkeiten zwischen Merkmalen, z.B. "requires" und "conflicts", als auch Restriktionen zwischen Merkmalen und/oder Attributen. ICCLs definieren Konfigurationssprachen zur Beschreibung von Abhängigkeiten zwischen implementierungs-orientierten Konzepten. Sie werden verwendet,

a) um Abhängigkeiten zwischen Implementierungskomponenten[1] bzw. Attributen von Implementierungskomponenten zu definieren; vergleichbar der "Configuration DSL" für das Merkmalmodell.

b) um Implementierungskomponenten konkreten Merkmalen aus dem Merkmalmodell zuzuordnen.

c) um Implementierungskomponenten der Variabilität im variantenreichen Simulink-Modell zuzuordnen. Zum Auflösen der Variabilität im variantenreichen Simulink-Modell wurden zu diesem Zweck parametrisierte Matlab-basierte Funktionen implementiert. Diese Funktionen bestehen aus einer Folge von Matlab-Befehlen, sogenannten "Model Construction Commands".
   Die parametrisierten Funktionen 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. Die Ausführung bewirkt, dass die Variabilität im variantenreichen Simulink-Modell aufgelöst wird.

[BFK05] beschreibt ausführlich die Definition des Konfigurationswissens als Bestandteil des Merkmal-basierten Domänen-Engineerings.

3.5.2 Konfigurationsmodellierung mit Entscheidungsmodellen

Das Ziel der Konfigurationsmodellierung innerhalb PESOA ist die Identifizierung und Dokumentation der Beziehungen zwischen Variationspunkten und Merkmalen der Prozessfamilie. Ausgangspunkt für die Entscheidungsmodellierung sind zum einen die variantenreiche Prozesse, die im Prozessentwurf erstellt wurden und zum anderen die während der Merkmalmodellierung entwickelte "Product Map", die alle Merkmale der Prozessfamilie, sowie deren Beziehungen untereinander, dokumentiert.

Die Beziehungen zwischen den Variationspunkten aus den variantenreichen Prozessen und den Prozessfamilienmerkmalen werden in PuLSE in den Ent-

[1] Wir haben Implementierungskomponenten lediglich als Proxy zur Speicherung von Konfigurationswissen verwendet.

scheidungsmodellen dokumentiert. Entscheidungsmodelle enthalten somit sowohl das Eigenschaftsmodell, als auch das Konfigurationsmodell, und somit die gesamte Information, die im Anwendungs-Engineering notwendig ist, um aus der Prozessfamilieninfrastruktur ein konkretes Produkt abzuleiten.

Abbildung 17 zeigt die vom IESE verfolgte Vorgehensweise zur Konfigurationsmodellierung.

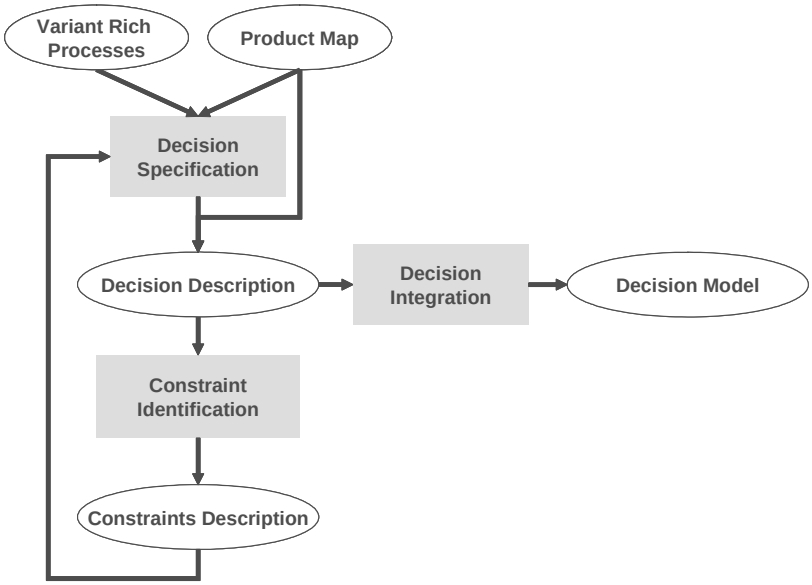

Abbildung 17: Konfigurationsmodellierung mit Entscheidungsmodellen

Wie aus Abbildung 17 hervorgeht, liegt der Konfigurationsmodellierung eine iterative Vorgehensweise zu Grunde. Das Ziel der Konfigurationsmodellierung ist es, die Abhängigkeiten zwischen den Variationspunkten und den Merkmalen zu erfassen, und damit das Anwendungs-Engineering zu ermöglichen.

Im ersten Schritt werden dabei die Auswirkungen von Merkmalen auf Variationspunkte identifiziert (Entscheidungsspezifikation, engl. "decision specificiation") und in einem zweiten Schritt als Entscheidungen dokumentiert (Entscheidungsbeschreibung, engl. "decision description"). Das Ergebnis sind Entscheidungsspezifikationen für jeden einzelnen Variationspunkt aus den vorhandenen variantenreichen Prozessen, die dann in das vorhandene Entscheidungsmodell integriert werden (Entscheidungsintegration, engl. "decision integration").

Abbildung 18 zeigt einen Ausschnitt aus einem Entscheidungsmodell zur Benzinmotorsteuerung. In dem Entscheidungsmodell werden Entscheidungen als Zeilen repräsentiert.

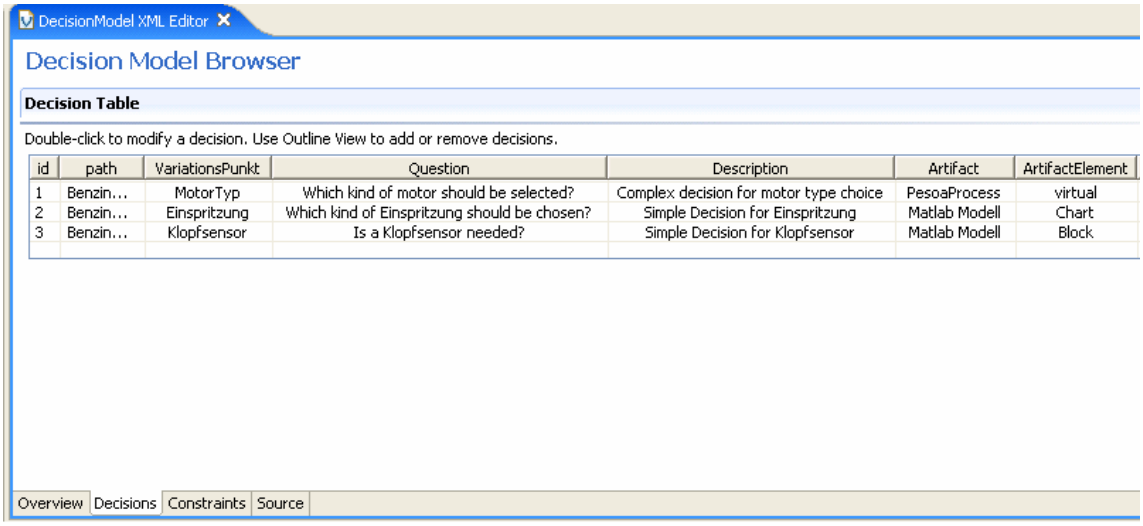

Abbildung 18: Beispiel: Variationspunkte und Entscheidungen

Die Entscheidungen werden außerdem miteinander in Beziehung gesetzt, indem Abhängigkeiten und Bedingungen zwischen ihnen identifiziert und dokumentiert werden (Abhängigkeitsidentifikation, engl. "constraints identification"). Das Ergebnis dieser Aktivität sind dokumentierte Abhängigkeiten zwischen Entscheidungen. Um die Abhängigkeiten zu identifizieren, werden die Entscheidungen untersucht, ob sie auf gleiche Merkmale verweisen.

Die Iteration über Entscheidungsspezifikation und Abhängigkeitsidentifikation resultiert in einer Hierarchie von Entscheidungen. Das Ziel der Entscheidungsintegration ist, diese Hierarchie und jede einzelne Entscheidung zu dokumentieren und zu warten. Das Ergebnis ist ein Entscheidungsmodell, das die Hierarchie der Entscheidungen beinhaltet und damit die Auflösung der Variationspunkte entsprechend der ausgewählten Merkmale ermöglicht.

Der Entscheidungsmodellierungsprozess wird im Folgenden anhand des Beispiels aus der Automotive-Domäne illustriert. Die einzelnen Aktivitäten des Prozesses wurden mit Hilfe des Entscheidungsmodellierungswerkzeugs, das am Fraunhofer IESE entwickelt wurde, ausgeführt. Zuerst wurden Entscheidungen definiert, die die identifizierten Variationspunkte mit den in der "Product Map" definierten Merkmalen verbinden. Für das Benzinmotorbeispiel wurden die Variationspunkte "Motor-Typ", "Einspritzung" und "Klopfsensor" mit Fragen verknüpft, die eine Verbindung zu den Merkmalen in der "Product Map" darstellen (siehe Abbildung 18).

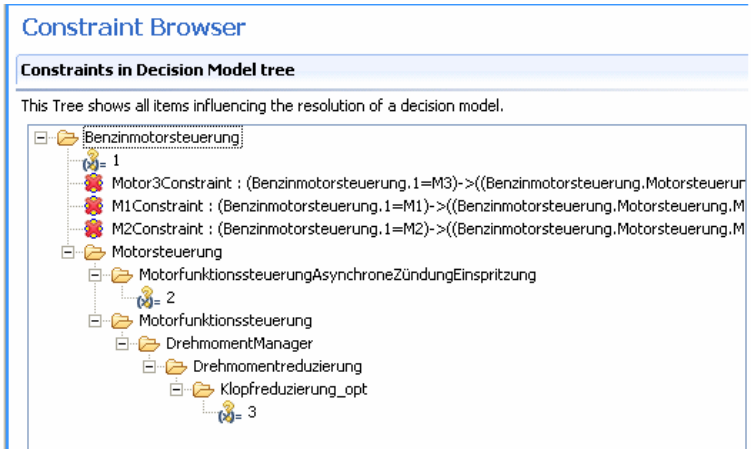

Abbildung 19: Beispiel: Constraint Browser

Anschließend wurden Abhängigkeiten zwischen dem Variationspunkt "Motor-Typ" und seinen Ausprägungen "M1", "M2" und "M3" identifiziert und im Constraint Browser (siehe Abbildung 19) beschrieben und dokumentiert. Abbildung 20 illustriert die Abhängigkeit, dass der Motor-Typ M1 die Variante "Saugrohreinspritzung" und den "Klopfsensor" bedingt.

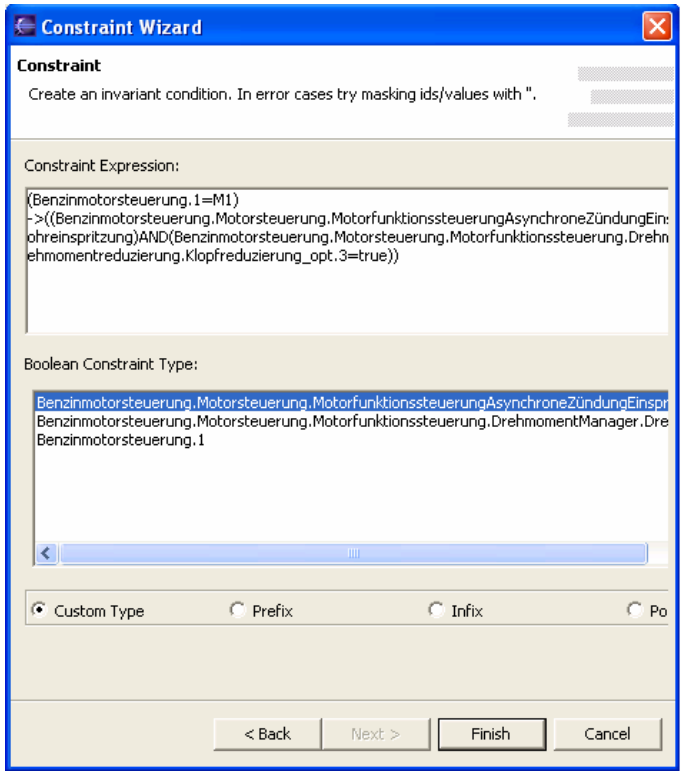

Abbildung 20: Beispiel: Constraint Definition

3.6 Aktivität "Implement DS (Domain-specific) Generator"

Das Ziel dieser Aktivität ist die Implementierung einer Abbildung, die die Konfigurationen variantenreicher Prozesse in ihre entsprechende Implementierung überführt. Dies soll möglichst automatisch, d.h. durch den Einsatz von Software-Generatoren, ablaufen.

Allgemein dienen Generatoren sowohl der Automation der Software-Entwicklung als auch der Realisierung höherer Abstraktionsebenen. Während die Automation mit Verbesserungen bezüglich Qualität und Produktivität einhergeht, führt die Realisierung höherer Abstraktionsebenen zu einem Effizienzgewinn beim Einsatz des Generators [Gie04a, Kap. 5.1]. Auf Generatoren, die genau dies tun, beziehen wir uns im Folgenden, nicht etwa auf einfache Filter oder Konverter. Solche Generatoren implementieren immer Gemeinsamkeiten und Variabilitäten einer Domäne. Da in PESOA variantenreiche Prozesse implementiert werden sollen, stellen domänenspezifische Generatoren die wichtigsten Komponenten für ihre Implementierung dar.

Im Folgenden wird die im Rahmen von PESOA gewählte Technik zur Entwicklung eines domänenspezifischen Generators vorgestellt. Das Konfigurationsmodell, das Merkmalmodell und der variantenreiche Prozess mit seinen Gemeinsamkeiten und Varianten sind Input für diese Aktivität. Das Ergebnis ist ein domänenspezifischer Generator.

Bei der Entwicklung von Generatoren wird in PESOA die von Delta Software Technology entwickelte HyperSenses-Technologie eingesetzt. Charakteristisch für HyperSenses sind die Aufteilung der Implementierung eines Generators in eine Modellebene und eine zielplattformspezifische Ebene, die beide werkzeuggestützt implementiert werden. Für beide Teilaufgaben steht hier HyperSenses Meta Composer™ als Werkzeug zur Verfügung [Gie04b, Kap. 2.1, 2.2]. Wir skizzieren im Folgenden diese Teilaufgaben genauer.

3.6.1 Variabilitätsmodelle

Dreh- und Angelpunkt der Entwicklung eines Generators sind die zu implementierenden Variabilitäten. In der Domänenanalyse wurden die Variabilitäten ermittelt und als Merkmalmodell erfasst (siehe Abschnitt 3.2). Entscheidend für die Implementierung eines domänenspezifischen Generators sind alle Variabilitäten, die sich auf die Erzeugung des Codes für eine Zielplattform auswirken. Dies sind alle Prozessvariabilitäten, erweitert um Variabilitäten, die nicht auf Prozessebene sichtbar sind, aber für die Generierung von Zielcode benötigt werden. Letztere sind Variabilitäten aus dem Lösungsraum, entsprechend dem generativen Domänenmodell aus [CE00]. Einzelne Variabilitäten des Lösungsraums können Variabilitäten des Domänenmodells entsprechen – wir beziehen uns hier auf solche Variabilitäten, für die dies nicht gilt. Die Existenz solcher zu-

sätzlicher Variabilitäten im Lösungsraum ist nicht zwingend, sondern eine Eigenschaft der spezifischen Prozesstransformation. Sind sie vorhanden, müssen sie im Variabilitätsmodell des Generators berücksichtigt werden.

Das Merkmalmodell dient als Vorlage für die interaktive Erfassung eines entsprechenden HyperSenses-Metamodells mit HyperSenses Meta Composer. Die dort verfügbaren Strukturen entsprechen dem OMG-Standard MOF (Meta Object Facility). Mit der Abbildung der Variabilitäten auf ein HyperSenses-Metamodell ist der erste Schritt zur Implementierung eines Generators abgeschlossen.

3.6.2 Konfigurations-Renderings

Im Zusammenhang mit HyperSenses bezeichnet man die konkreten Konfigurationsdaten für eine Generierung als Modell. Für einen Code-Generator kann es sinnvoll sein, Test-Modelle zu erzeugen, um die Konsistenz des Metamodells und der Produktions-Renderings (siehe Abschnitt 3.6.3) zu überprüfen. Die Erstellung von Modellen geschieht in HyperSenses durch so genannte Konfigurations-Renderings. Sie verknüpfen Variabilitäten des Metamodells mit einer Beschreibung in Textform und einer Wertzuordnung [Gie04b, Kap. 3.3]. Das zuvor in der Phase Domänenentwurf erstellte Konfigurationsmodell stellt eine hilfreiche Vorlage zur Definition der Konfigurations-Renderings dar (siehe Abschnitt 3.5).

Außer zu Testzwecken können Konfigurationen in HyperSenses auch erforderlich sein, wenn die Produktkonfiguration (siehe Abschnitt 3.9) die oben genannten zusätzlichen Variabilitäten aus dem Lösungsraum nicht abdeckt. Dann sind diese in der Phase Anwendungsimplementierung noch zu konfigurieren (siehe Abschnitt 3.10).

Ein Konfigurations-Rendering realisiert, zusammen mit dem entsprechenden Visualisierungswerkzeug, in HyperSenses eine domänenspezifische Sprache. Diese stellt eine Mischung aus Text- und graphischen Elementen dar und erhält durch das eingesetzte Werkzeug HyperSenses Active Intent™ interaktiven Charakter.

3.6.3 Renderings für Zielplattformen

Auf der Basis des jeweiligen Metamodells ist zu definieren, wie einzelne Variabilitäten auf Zielcode abgebildet werden. Mit der HyperSenses-Technologie wird diese Abbildung durch so genannte Produktions-Renderings realisiert. Sie annotieren einzelne Variabilitäten mit Text, der zudem mit logischen Bedingungen und Abhängigkeiten versehen werden kann [Gie04b, Kap. 3.5]. Möchte man, wie bei Produktions-Renderings, mit einem Rendering Zielcode erzeugen, bestehen diese Annotationen aus Zielcode. Die dafür notwendigen Zielcode-

Fragmente besitzt man bereits aus einer vorangegangenen Entwicklung von Prototypen oder man leitet sie aus den in der Domänenanalyse ermittelten Gemeinsamkeiten der Domäne ab. Ersteres entspricht der "Pattern By Example"-Methode [Gie04a, Kap. 4]. Im zweiten Fall gehen die gesuchten Gemeinsamkeiten aus dem variantenreichen Prozess hervor (siehe Abschnitt 3.4.1). Die Produktions-Renderings werden mit HyperSenses Meta Composer erstellt. Sie sind an das zuvor erfasste Metamodell gekoppelt.

Die Erstellung der Produktions-Renderings schließt die Implementierung des Code-Generators ab.

3.7 Aktivität "Implement DS (Domain-specific) Components"

Das Ziel dieser Aktivität besteht in der Implementierung generischer Komponenten. Zusammen mit domänenspezifischen Generatoren (siehe Abschnitt 3.6) machen sie die Implementierung einer Domäne aus. Sie lassen sich nach dem Zeitpunkt ihrer Verwendung in Komponenten zur Erstellung der Applikation (Infrastrukturkomponenten, engl. "Infrastructure Components") und in Laufzeitkomponenten (engl. "Runtime Components") unterteilen.

Infrastrukturkomponenten werden zur Erstellung der ausführbaren Prozessimplementierung aus dem Resultat des Generators eingesetzt. Laufzeitkomponenten enthalten Teile der Funktionalität des variantenreichen Prozesses. Als Input dient der domänenspezifische Generator. Das Resultat sind die generischen Komponenten.

Im Folgenden werden technische Aspekte bei der Erstellung von Infrastrukturkomponenten und Laufzeitkomponenten getrennt voneinander skizziert.

3.7.1 Infrastrukturkomponenten

Der durch den Generator erzeugte Zielcode stellt in der Regel Quellcode dar, der im Rahmen der Anwendungsimplementierung noch weiter verarbeitet werden muss, um eine ausführbare Applikation (das Produkt) zu erhalten. Eine solche Verarbeitung kann z.B. im Kompilieren, Binden, Packaging oder Deployment bestehen. Sowohl die dafür notwendigen Werkzeuge als auch verwendete Frameworks und Bibliotheken stellen Infrastrukturkomponenten dar. Sofern diese spezifisch für die Domäne entwickelt werden müssen, sind diese Entwicklungsarbeiten Bestandteil der Domänenimplementierung.

Die Entwicklung des domänenspezifischen Generators stellt die Basis für die Entwicklung der Infrastrukturkomponenten dar (siehe Abschnitt 3.6), da sich dabei herausstellt, wie der zu verarbeitende Zielcode aussehen wird.

3.7.2 Laufzeitkomponenten

Die Zielplattform für zu implementierende Prozessfamilien kann in der Laufzeitumgebung einer konventionellen Programmiersprache oder in einer speziell für Prozesse entworfenen Simulations- oder Ablaufumgebung bestehen. In jedem Fall ist es denkbar, dass Funktionalitäten, die domänenspezifische Gemeinsamkeiten bündeln, als eigene Laufzeitkomponenten, beispielsweise Laufzeitbibliotheken, realisiert werden.

Ist die Zielplattform eine spezielle Simulations- oder Ablaufumgebung, d.h. ist diese domänenspezifisch, so fällt die Implementierung der entsprechenden Laufzeitkomponenten ebenfalls in die Projektphase Domänenimplementierung. Ein Beispiel dafür ist ein Laufzeitsystem für eine eigene domänenspezifische Prozessausführungssprache, etwa nach dem Vorbild von BPEL4WS ("Business Process Execution Language for Web Services"), auf welche variantenreiche Prozessmodelle durch den Code-Generator abgebildet werden.

Die durch Laufzeitkomponenten implementierten Funktionalitäten ergeben sich aus dem Funktionsumfang des durch den domänenspezifischen Generator zu erzeugenden Zielcodes (siehe Abschnitt 3.6).

3.8 Aktivität "Specify Product"

Das Ziel dieser Aktivität ist die Definition eines neuen Produktes, das unter Wiederverwendung der Prozessfamilieninfrastruktur erstellt werden soll. Die Scope-Definition und das Merkmalmodell sind die Ausgangspunkte zur Produktspezifikation. Das Resultat ist ein Merkmalmodell für das spezifische Produkt und eine Liste von Merkmalen, die in die Prozessfamilie aufgenommen werden sollten, da sie für das aktuelle Produkt relevant sind, aber nicht von der Prozessfamilie abgedeckt sind. Diese Merkmale dienen als Input für das Domänen-Engineering, wo sie verwendet werden um die Scope-Definition der Prozessfamilie zu erweitern.

3.8.1 Merkmalmodell-basierte Produktspezifikation

Das Merkmal-basierte Konfigurationswerkzeug pure::variants erlaubt dem Anwendungsentwickler, durch Auswahl gewünschter Merkmale das Merkmalmodell einer Produktfamilie zu laden. Aus dem Merkmalmodell kann der Entwickler anschließend konkrete Produktvarianten spezifizieren. Über Transformatoren werden die Merkmalkonfigurationen geprüft. Auf diese Weise wird sichergestellt, dass nur eine gültige Auswahl von Merkmalen erzeugt werden kann.

3.8.2 Produktspezifikation auf der Basis von Entscheidungsmodellen

Entscheidungsmodelle bestehen aus einem Merkmalmodell und aus einem Konfigurationsmodell. Zur Produktspezifikation werden das Merkmalmodell, das die von der Prozessfamilie abgedeckten Merkmale dokumentiert, und die Scope-Definition verwendet. Auf der Basis der Anforderungen an ein neues Produkt werden nun diejenigen Merkmale der Prozessfamilie ausgewählt, die benötigt werden um die Produktanforderungen zu erfüllen. Das Ergebnis ist eine Instanz des Merkmalmodells, in dem diejenigen Merkmale, die das neue Produkt erfüllen soll, gekennzeichnet sind. Zusätzlich werden noch die Merkmale erfasst, die von der Prozessfamilieninfrastruktur nicht abgedeckt werden können.

3.9 Aktivität "Configure Product"

Das Ziel dieser Aktivität ist die Erstellung einer Konfiguration anhand der Spezifikation bzw. des Produktmerkmalmodells, so dass das Produkt von der Prozessfamilieninfrastruktur instantiiert werden kann. Als Input dienen das Produktmerkmalmodell, das Konfigurationsmodell, und der variantenreiche Prozess. Das Konfigurationsmodell ermöglicht die Auflösung der Variationspunkte im variantenreichen Prozess, basierend auf den spezifizierten Produktmerkmalen. Das Resultat ist die Dokumentation der Auflösung, das so genannte Auflösungsmodell (engl. "resolution model").

Im Folgenden werden drei Techniken zur Konfiguration eines Produktes vorgestellt.

3.9.1 UML / BPMN-basierte Produktkonfiguration

Bei der Konfiguration eines variantenreichen Prozessmodells werden entsprechend den Informationen aus dem Konfigurationsmodell die Varianten aus dem Prozessmodell entfernt, die für die Implementierung der Applikation nicht relevant sind.

3.9.2 Matlab-basierte Produktkonfiguration

Im Rahmen der Instanziierung variantenreicher Simulink-Modelle wird über das Merkmalmodell eine Auswahl der Merkmale für eine Produktvariante durchgeführt (siehe Abschnitt 3.8.2 und [BFK05]).

Auf Basis des Konfigurationswissens kann anschließend ein Modellkonfigurator automatisiert die Merkmalauswahl validieren und das Auflösungsmodell erzeugen. In unserem Fall ist dies ein so genanntes Matlab-Skript. Dieses Matlab-

Skript umfasst eine Folge parametrisierter Funktionsaufrufe, entsprechend der ausgewählten Merkmale (siehe Abschnitt 3.5.1).

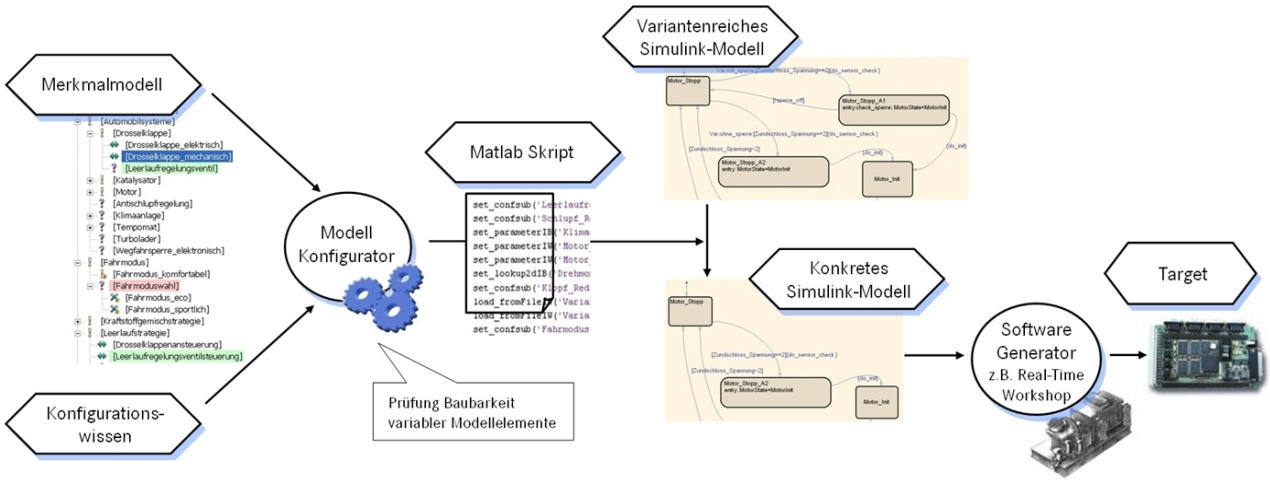

Abbildung 21: Matlab-Skript-basierte Produktkonfiguration

Das Matlab-Skript kann anschließend in Matlab importiert und ausgeführt werden. Gemäß der spezifizierten Funktionsaufrufe wird die Variabilität im variantenreichen Simulink-Modell aufgelöst. Das Ergebnis ist ein konkretes Simulink-Modell, aus dem über einen Software-Generator – beispielsweise den Real-Time Workshop – C-Code generiert werden kann. Dieser kann anschließend übersetzt, gebunden und auf die Zielplattform – das Steuergerät – aufgespielt werden.

Die Auflösung der Variabilität im variantenreichen Simulink-Modell auf Basis des Matlab-Skripts und das anschließende Erzeugen von Quellcode in Matlab entspricht hierbei den PESOA-Prozessschritten der Phasen Domänen- und Anwendungsimplementierung.

3.9.3 Produktkonfiguration mit Entscheidungsmodellen

Die Produktkonfiguration hat zum Ziel, die Variationspunkte innerhalb des variantenreichen Prozesses entsprechend der identifizierten Produktmerkmale aufzulösen. Ausgangspunkt für diese Instantiierung sind die identifizierten Produktmerkmale, das Entscheidungsmodell, und die variantenreichen Prozesse. Ergebnis ist das so genannte Auflösungsmodell (engl. "resolution model"), das die Konfigurationen des Entscheidungsmodells für die Auflösung der Variationspunkte entsprechend der Produktmerkmale 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. Die zweite Methode löst den variantenreichen Prozess für jedes einzelne identifizierte Merkmal auf, dabei werden Entscheidungen im Entscheidungsmodell "top-down" ausgeführt. Bei beiden Vorgehensweisen werden die Schritte und die entsprechenden Auflösungen im Auflösungsmodell dokumentiert.

Gibt es Merkmale, die zwar für das Produkt gefordert sind, die aber nicht von der Prozessfamilieninfrastruktur abgedeckt sind, müssen diese produktspezifisch zur aufgelösten Prozessfamilieninfrastruktur hinzugefügt werden. Input dafür ist eine Liste von Merkmalen, die nicht in der Prozessfamilie vorzufinden sind. Das Ergebnis der produktspezifischen Entwicklung ist dann das Produkt mit allen spezifizierten Merkmalen.

Im Folgenden wird die Konfiguration auf der Basis von Entscheidungsmodellen anhand des Beispiels der Benzinmotorsteuerung erklärt (vergleiche Abschnitt 3.5.2). Abbildung 22 zeigt die möglichen Varianten für den Motor-Typ an. Wählt man Motor-Typ "M1" aus, dann lösen sich entsprechend der in Abschnitt 3.5.2 definierten Abhängigkeiten die Variationspunkte "Einspritzung" und "Klopfsensor" auf, wie in Abbildung 23 dargestellt.

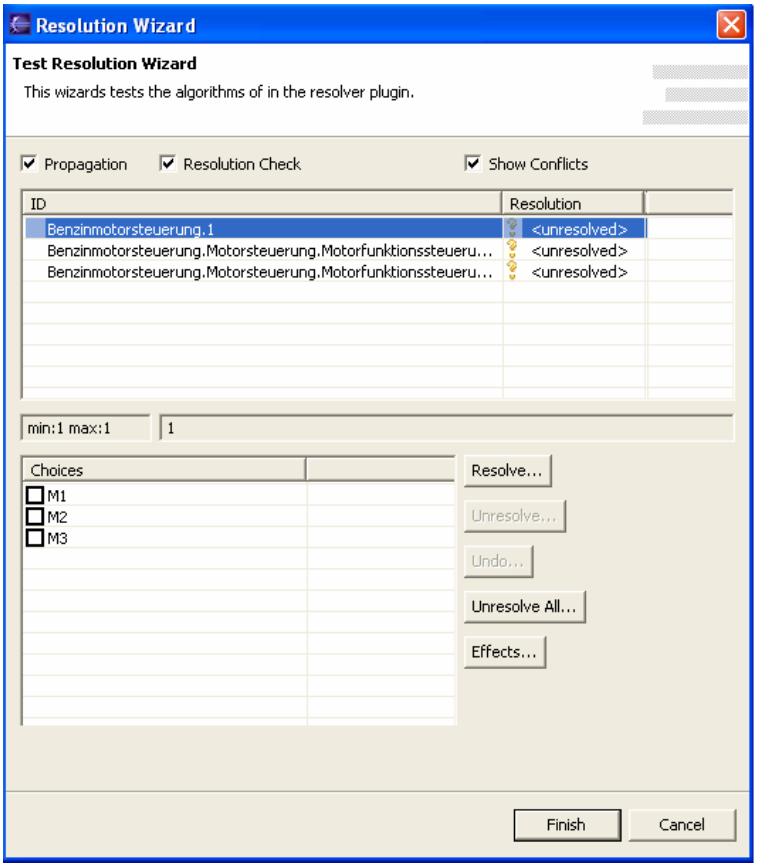

Abbildung 22: Beispiel: Nicht aufgelöst

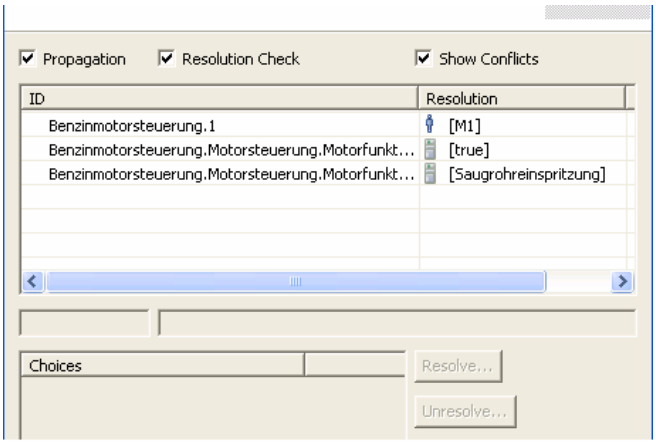

Abbildung 23: Beispiel: Instantiierung

3.10 Aktivität "Apply DS (Domain-specific) Generator"

Das Ziel dieser Aktivität ist die Erzeugung von konkretem Code mit Hilfe des domänenspezifischen Generators, basierend auf den Auflösungen der Variationspunkte im variantenreichen Prozess.

Als Input dienen der domänenspezifische Generator und das Auflösungsmodell (engl. "resolution model"). Das Resultat ist der Zielcode (engl. "target code"). Im Folgenden wird eine Technik zur Erzeugung des Zielcodes vorgestellt.

Die Anwendung eines domänenspezifischen Generators lässt sich auf einer technischen Ebene in zwei wesentliche Teilvorgänge aufteilen, die in den folgenden beiden Abschnitten erläutert werden.

3.10.1 Modell-Import

Dieser erste Arbeitsschritt kennzeichnet die Anbindung an die vorangegangene Phase im Anwendungs-Engineering, die Produktkonfiguration ("Configure Product", siehe Abschnitt 3.9). Die dort erhaltenen Konfigurationsdaten sind in ein HyperSenses-Modell zu überführen, welches zu dem erstellten Metamodell (siehe Abschnitt 3.6.1) konsistent ist. Einzelheiten zum Modell-Import finden sich in [GOB05, Kap. 4.2].

3.10.2 Code-Generierung

Ist ein HyperSenses-Modell vorhanden, kann eine Konfiguration noch nicht aufgelöster Variabilitäten erforderlich sein. Dies setzt ein entsprechendes Konfigurations-Rendering voraus (siehe Abschnitt 3.6.2).

Die eigentliche Code-Generierung geschieht durch die Auswahl der gewünschten Zielplattform, d.h. des gewünschten Produktions-Renderings [Gie04b, Kap. 3.6]. In HyperSenses besteht eine Generierung immer lediglich im Umschalten auf ein anderes Rendering.

Zur Anwendung des Code-Generators wird das Werkzeug HyperSenses Active Intent™ eingesetzt.

3.11 Aktivität "Build, integrate and test"

Das Ziel dieser Aktivität besteht darin, aus dem erzeugten Zielcode die entsprechenden ausführbaren Module zu erstellen, diese zu integrieren und zu testen.

Input sind der zuvor generierte Zielcode und die domänenspezifischen Komponenten. Das Resultat ist die lauffähige Applikation (das Produkt).

Die aus der Code-Generierung erhaltenen Quellcode-Dateien werden, sofern sie nicht unmittelbar ausführbar sind (etwa durch ein spezielles Laufzeitsystem), kompiliert und zusammen mit möglichen generischen Komponenten gebunden. Dieser Schritt schließt insbesondere die Verwendung domänenspezifischer Infrastrukturkomponenten ein (siehe Abschnitt 3.7.1).

Anschließend werden alle Komponenten einschließlich möglicher Laufzeitkomponenten in eine Testumgebung integriert. Auch Teile der Testumgebung selbst können als Laufzeitkomponenten in der Domänenimplementierung entwickelt worden sein (siehe Abschnitt 3.7.2).

Den Abschluss der Implementierung einer Variante eines variantenreichen Prozesses bildet die Testphase.

4 Management

Das Management einer Produktlinie umfasst den kompletten Lebenszyklus einer Produktlinie – Entwicklungs- und Wartungsphasen – wie in Abbildung 24 dargestellt. Innerhalb dieses Lebenszyklus werden verschiedenartige Projekte durchgeführt. In der Entwicklungsphase einer Produktlinie unterscheidet man zwischen Domänen- und Anwendungs-Engineering-Projekten. In der Wartungs- und Weiterentwicklungsphase können Projekte zum Beispiel mit dem Ziel der Erweiterung der Funktionalität innerhalb der Produktlinieninfrastruktur bzw. in ausgelieferten Produkten als so genannte "Extension"-Projekte aufgesetzt werden. Des weiteren unterscheidet man zwischen Servicing-, Reengineering-, Integrations- und Migrationsprojekten. Eine Definition der einzelnen Projektarten findet sich in [BLM05].

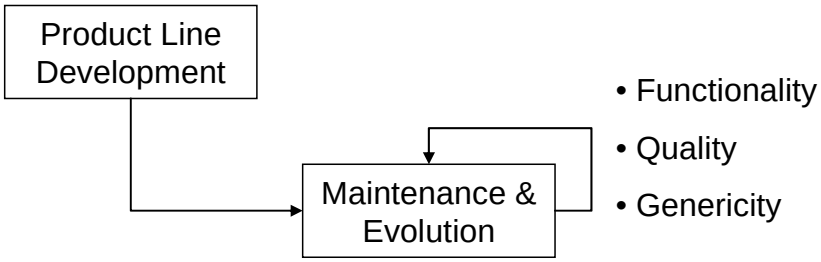

Abbildung 24: Allgemeiner Produktlinien-Lebenszyklus

Die Aufgabe des Produktlinien-Managers ist das Management des Produktlinien-Lebenszyklus, d.h. die Produktlinien zu entwickeln, weiter zu entwickeln, und zu warten, indem er die entsprechenden Projekte plant. Das "Lebensziel" einer Produktlinie und damit das Ziel des Managers ist es, mindestens die geforderte Funktionalität und Qualität, sowie das mögliche Wiederverwendungspotential der Produktlinien zu gewährleisten bzw. zu steigern.

Der in PESOA definierte Managementansatz basiert darauf, das Profil, d.h. die Funktionalität, Qualität und das Wiederverwendungspotential einer Produktlinie zu erhalten und zu verbessern. Der Produktlinienmanager definiert sich ein Zielprofil, das er dann durch die Auswahl von entsprechenden Projekten ausgehend vom aktuellen Profil der Produktlinie fokussiert und anstrebt. Zur Auswahl der Projekte dient die Spezifikation von Projekten und deren Beeinflussung des Produktlinienprofils [BLM05].

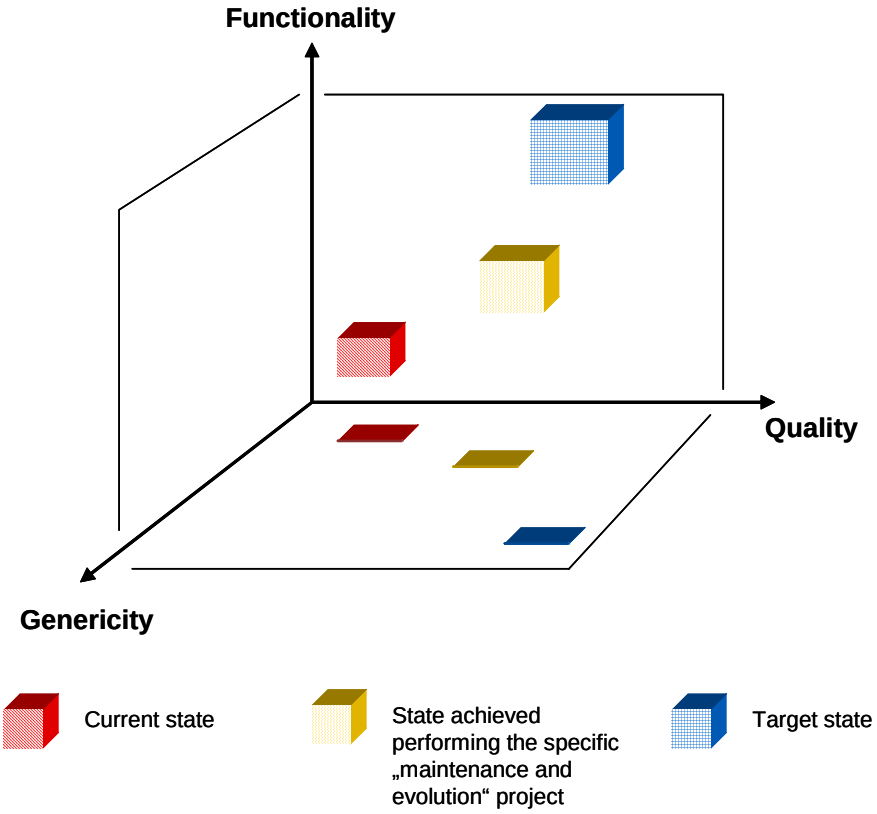

Abbildung 25: Managementansatz

Abbildung 25 stellt den PESOA-Managementansatz graphisch dar; ausgehend vom aktuellen Zustand der Produktlinie werden Projekte ausgewählt um das angestrebte Zielprofil zu fokussieren.

Neben der Auswahl der einzelnen Projekte innerhalb eines Produktlinienlebenszyklus hat das Management die Aufgabe, die ausgewählten Projekte im Detail zu planen und zu kontrollieren. Eine Voraussetzung für eine gute Projektplanung ist unter anderem die Abschätzung des Umfangs und damit des resultierenden Aufwands für das jeweilige Projekt.

Der Planungsprozess für Domänen-Engineering und Anwendungs-Engineering-Projekte, sowie eine Technik zu Abschätzung von deren Umfang und resultierendem Aufwand werden im Folgenden detaillierter erklärt.

4.1 Projektplanung

Das Ziel der Projektplanung ist die Definition eines Projektplans mit definierten Aktivitäten und deren zeitlichen Rahmen. Ausgangspunkt der Planung ist die

Definition des Projektrahmens, d.h. das Ziel des Projektes wie in Abbildung 25 beschrieben.

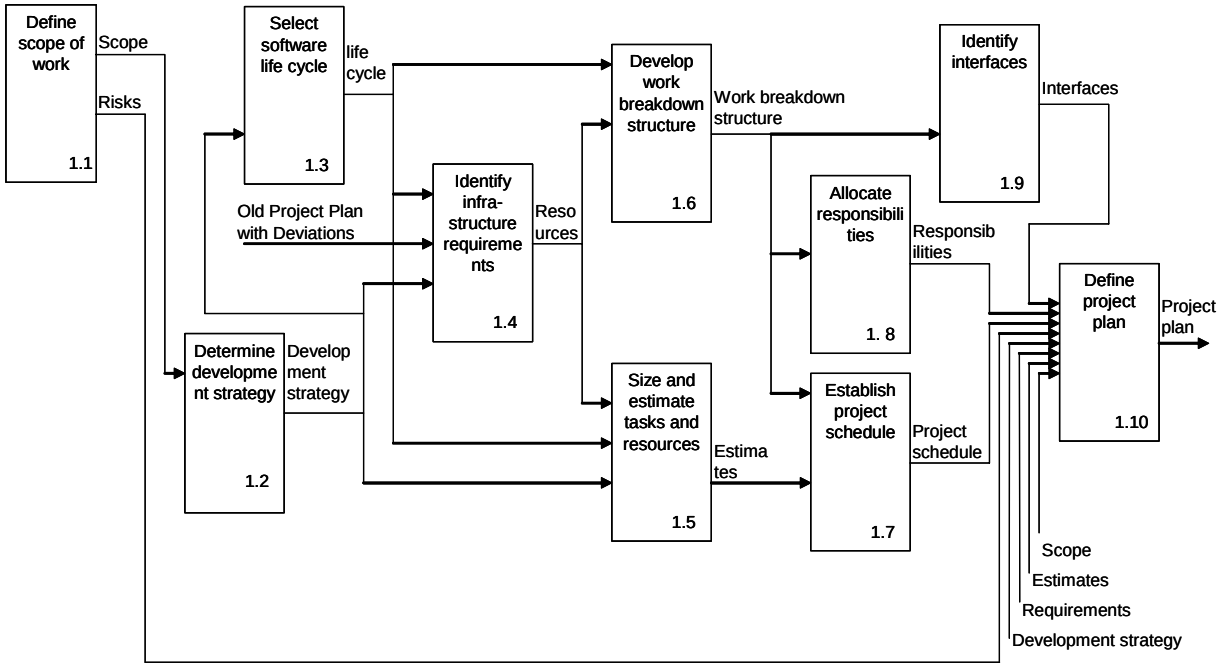

Abbildung 26: Planungsprozess

Der Rahmen für ein Domänen-Engineering-Projekt ergibt sich aus den im "Domain Scoping" definierten Merkmalen und einer ersten konzeptionellen Sicht des variantenreichen Prozesses. Dagegen definiert sich ein Anwendungs-Engineering-Projekt aus den identifizierten Anwendungsanforderungen. Auf der Basis dieser Rahmendefinition werden eine Entwicklungsstrategie und ein Software-Lebenszyklus für das jeweilige Projekt ausgewählt. Dann werden die Anforderungen für die Infrastruktur, der Strukturplan der einzelnen notwendigen Aktivitäten und mögliche Schnittstellen spezifiziert. Anschließend wird der Aufwand für das Projekt berechnet, und die benötigten Personen den definierten Rollen zugeordnet. Eine detaillierte Definition des Planungsprozesses wie in Abbildung 26 dargestellt findet sich in [BLM04].

4.2 Metriken der Umfangsmessung und Aufwandsprognose

Der PESOA-Ansatz beinhaltet ein Framework aus Software-Metriken, um die Größe einer prozessorientierten Software-System-Familie zu messen. Die Ergebnisse dieser Umfangsbestimmung können anschließend als Indikatoren zur Durchführung einer Aufwandsprognose genutzt werden.

4.2.1 Metriken der Umfangsmessung

Die Software-Metriken der Prozess-Familien-Punkte (PFP)-Umfangsmessung setzen die Determination eines Zählmodells voraus. Diese Spezifikation des Zähltyps beeinflusst die Berechnung des justierten Größenmaßes und stellt eine akzeptanzfördernde Vergleichbarkeit zu anderen Metriken sicher.

In einem weiteren Schritt erfolgt die Demarkation des Umfangs der Zählung sowie die Festlegung der Software-System-Familien (SSF)-Systemgrenzen. Hierbei werden die dynamischen Begrenzungen zwischen den variablen sowie gemeinsamen SSF-Bausteinen identifiziert und einzelne Varianten umrissen. Eine mehrmalige, zeitlich versetzte Durchführung des Vorgehens der Demarkation ermöglicht die Erfassung und Berücksichtigung der Evolution von SSF.

Die darauf folgende PFP-Mikroanalyse ist als ein Konglomerat aus Metriken zur Kalkulation eines unjustierten Umfangsmasses für SSF zu kennzeichnen. Aufgrund der derzeit eingeschränkten Nutzung von SSF, welche sich auf die Domäne des *Electronic Business* und den Automobilbereich beschränkt, erfährt die PFP-Mikroanalyse eine duale Untergliederung:

- *Electronic Business:* Kategorisierung, Komplexitätsgewichtung sowie Transformation der umfangswirksamen Aspekte einer Daten- und Prozessperspektive;
- *Automotive:* Kategorisierung, Komplexitätsgewichtung sowie Transformation der den Umfang beeinflussenden Elemente einer echtzeit- und prozessorientierten Sichtweise.

Im Anschluss an die Transformationsphase der PFP-Mikroanalyse werden die berechneten Größenmaße summiert und auf Basis des Zähltyps einem Projekt oder Produkt zugeordnet. Das hierbei ausgewiesene Maß der unjustierten PFP kann bereits als frühzeitiger Aufwandsindikator genutzt werden und ermöglicht einen Vergleich mit klassischen Größenmaßen.

Die Durchführung der PFP-Umfangsmessung beschließend, erfolgt eine Dokumentation sämtlicher Aktivitäten und Annahmen mit einem potentiellen Einfluss auf das Messergebnis.

4.2.2 Metriken der Aufwandsprognose

In einer ersten Evaluationsphase erfassen die Metriken der PFP-Aufwandsprognose vier allgemeine SSF-Rahmenbedingungen, die in jeweils fünf beispielhafte Subkategorien untergliedert werden. Die Bewertung der Aufwandswirkung dieser 20 Einzelkriterien resultiert in dem numerischen Einflussgrad der domänenunabhängigen Faktoren und erfolgt obligatorisch.

Nach einer Beurteilung der generellen Einflussfaktoren fokussieren die Softwaremetriken der PFP-Aufwandsvorhersage die domänenspezifischen Kostentreiber im *Electronic Business* oder im Automobilbereich. Für jede dieser Applikationsdomänen werden drei exemplarische Einflussfaktoren mit jeweils fünf Subkategorien hinsichtlich ihrer potentiellen Aufwandswirkung untersucht. Die Evaluation von 15 domänenabhängigen Einzelkriterien ist verbindlich durchzuführen und ermöglicht die Kalkulation eines domänenspezifischen Einflussgrades.

Im Gegensatz zu den obigen Metriken der PFP-Aufwandsprognose erfolgt die Bewertung von 27 qualitativen Aufwandsgeneratoren nach ISO/IEC 9126 ausschließlich optional. Die zusätzliche Berücksichtigung dieser Kriterien resultiert in einem qualitativen Einflussgrad.

Auf Basis des generellen und domänenspezifischen Einflussgrades erfolgt in einer weiteren Phase die hochflexible Justierung des Ergebnisses der PFP-Umfangsmessung. Neben dem prozentualen Maximaleinfluss der Justierung sind die Menge der generellen sowie domänenabhängigen Faktoren und die Anzahl der jeweiligen Subkriterien variabel änderbar. Ferner ist eine zählmodellspezifische Justierung durchzuführen, um die Kompatibilität zwischen den justierten Größenmaßen der PFP-Analyse und der *Function Point Analysis* sicherzustellen.

Unter Bezugnahme auf den optional kalkulierten Einflussgrad der qualitativen Kostentreiber kann eine Berechnung des Größenmaßes der qualitätsjustierten PFP erfolgen, welches eine erhöhte Korrelation zum SSF-Entwicklungsaufwand aufweist. Gleichzeitig wird jedoch die Vergleichbarkeit zu den etablierten Ansätzen der Aufwandsvorhersage beeinträchtigt, welche diese qualitativen Einflüsse nicht ausreichend berücksichtigen. Ferner ist der prozentuale Maximaleinfluss der Qualitätsjustierung frei wählbar, wohingegen die Anzahl der qualitativen Einflussfaktoren mit dem Ziel einer ISO/IEC 9126-Kompatibilität statisch vorgegeben wird.

Die finale Prognose der künftigen Aufwände zur Entwicklung oder Modifikation von SSF basiert auf empirisch hergeleiteten Schätzgleichungen. Für die Umfangsmasse der unjustierten, justierten und qualitätsjustierten PFP werden domänenspezifische Gleichungssysteme zur Kalkulation der künftig anfallenden Aufwendungen in Personenstunden bereitgestellt.

Abschließend erfolgt die Dokumentation der Annahmen und Aktivitäten, die einen potentiellen Einfluss auf das Schätzergebnis der PFP-Aufwandsprognose ausüben.

Referenzen

[Baye+99] Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T., DeBaud, J.-M. PuLSE – A Methodology to Develop Software Product Lines. Symposium on Software Reus-ability, Los Angeles, USA (SSR'99), 1999, S. 122-131.

[BBG05] Joachim Bayer, Winfried Buhl, Cord Giese, Theresa Lehner, Alexis Ocampo, Frank Puhlmann, Ernst Richter, Arnd Schnieders, Jens Weiland. *Process Family Engineering: Modeling variant-rich processes*. PESOA Report No. 18/2005, Fraunhofer IESE, Delta Software Technology, Hasso-Plattner-Institute, DaimlerChrysler Research and Technology, 2005.

[BFK05] Joachim Bayer, Thomas Forster, Sebastian Kiebusch, Theresa Lehner, Alexis Ocampo, Jens Weiland. *Feature- und Entscheidungsmodellbasierte Varianteninstanziierung im PESOA-Prozess*. PESOA Report No. 21/2005, Fraunhofer IESE, Universität Leipzig, DaimlerChrysler Forschung und Technologie, 2005.

[BLM04] Joachim Bayer, Theresa Lehner, Dirk Muthig. *Product Line Engineering and Software Project Management*. PESOA Report No. 11/2004, Fraunhofer IESE, 2005.

[BLM05] Joachim Bayer, Theresa Lehner, Dirk Muthig. Product Line Management in Practice. PESO Report No.19/2005, Fraunhofer IESE, 2005.

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

[Gie04a] C. Giese, W. Buhl. *Software-Generatoren*. PESOA-Report Nr. 04/2004, Delta Software Technology, Februar 2004, siehe: http://www.PESOA.org

[Gie04b] C. Giese, W. Buhl. *Modell-basierte Prozesstransformationen*, PESOA-Report Nr. 10/2004, Delta Software Technology, Oktober 2004, siehe: http://www.PESOA.org

[GOB05] C. Giese, H. Overdick, W. Buhl: *Realisierungsstrategien für Prozessfamilien*, PESOA-Report Nr. 15/2005, Delta Software Technology, Hasso-Plattner-Institut, Juni 2005, siehe: http://www.PESOA.org

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

[Mut02] Dirk Muthig, *A light-weight Approach Facilitating an Evolutionary Transition Towards Software Product Lines.* PhD Thesis, Fraunhofer IRB Verlag, 2002.

[Pen03] T. Pender, *UML Bible*. Indianapolis, Indiana: Wiley Publishing Inc., 2003.

[PuLSE] Fraunhofer IESE: PuLSE™ Websites; http://www.iese.fraunhofer.de/PuLSE/

[PSW05] F. Puhlmann, A. Schnieders, J. Weiland, M. Weske. *Variability Mechanisms for Process Family Engineering.* PESOA Report No. 17/2005, Hasso-Plattner-Institute, DaimlerChrysler Research and Technology, 2005.

[RJB04] J. Rumbaugh, I. Jacobson, G. Booch. The Unified Modeling Language Reference Manual, Addison-Wesley Professional, 2004.

[RSW05] Ernst Richter, Arnd Schnieders, Jens Weiland. *Prozessanalyse und -modellierung in der Domäne Automotive*. PESOA Report No. 07/2004, Hasso-Plattner-Institute, DaimlerChrysler Research and Technology, 2005.

[Schm02] Schmid, K. Planning Software Reuse – A Disciplined Scoping Approach for Software Product Lines. Dissertation, 2002

[SEI04] Software Engineering Institute, A Framework for Software ProductLine Practice Version 4.2. 2004, http://www.sei.cmu.edu/plp/framework.html, Abruf am: 2004-12-10.

[UML03] UML 2.0 Superstructure Specification, OMG, August 2003.

Dokumenten Information

Titel:	PESOA Guidebook Methodik und Techniken
Datum:	27. August 2005
Report:	IESE-129.06/D
Status:	Final
Klassifikation:	Öffentlich