Dokumentationsmethodik zur Rekonfiguration von Softwarekomponenten im Automobil-Service

Dissertation
der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Dipl.-Inform. (FH) Michael Köhler
aus Brandenburg a. d. Havel

Tübingen
2011

Tag der mündlichen Qualifikation:	04.10.2011
Dekan:	Prof. Dr. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. Wolfgang Rosenstiel
2. Berichterstatter:	Prof. Dr. Wolfgang Küchlin

Things that are different should look different.
Larry Wall

Vorwort

Die vorliegende Arbeit leistet einen wissenschaftlichen Beitrag zum Thema Rekonfigurieren im Automobil-Service. Die wesentlichen Grundlagen dazu und alle praktischen Arbeiten entstanden zwischen 2006 und 2008 im Forschungsbereich Offboard-Diagnose sowie im Bereich Flashprozess in Entwicklung, Produktion und Aftersales der Daimler AG. In den folgenden Jahren wurden die praktischen Arbeiten und theoretischen Vorüberlegungen zu der vorliegenden Arbeit vervollständigt, zusammengefasst und aufbereitet.

Mein Dank gilt allen, die zum Gelingen dieser Arbeit beigetragen haben. Ganz besonders danke ich Prof. Dr. Wolfgang Rosenstiel für seine offene und zielstrebige Betreuung. Bei Prof. Dr. Wolfgang Küchlin möchte ich herzlich für die Übernahme des Zweitgutachtens bedanken.
Darüber hinaus möchte ich allen Kollegen und Kolleginnen sowie Vorgesetzten danken, die mich während meiner Arbeit bei der Daimler AG unterstützt haben und ohne die die Konzeption und Umsetzung des entstandenen Systems nicht möglich gewesen wäre. Vor allem Stephan Steinhauer, Werner Preuschoff, Dr. Tim Schlüsener, Franz Bodensteiner, Dr. Jens Kohler und Thomas Weirauch, die mich sowohl organisatorisch als auch fachlich jederzeit unterstützt haben.
Mit einer starken Frau an der Seite geht vieles einfacher: Meinen ganz herzlichen Dank möchte ich auch an meine Lebensgefährtin Susan Griesbach richten, vor allem für ihre Geduld.

Inhaltsverzeichnis

1 Einleitung

1.1 Individualisierung der Produkte

Produkte für die Industrie wie Industrieanlagen oder medizinische Geräte sowie Autos, Bekleidung, Reisen oder sogar Müsli für private Kunden sind heute immer weniger Standardprodukte und werden zunehmend nach individuellen Anforderungen der Kunden hergestellt. Kunden können die Merkmale und Bestandteile ihres Wunschproduktes gezielt bestimmen und werden dabei durch Verkaufsberater oder Konfigurierungswerkzeuge unterstützt. Der durch Henry Ford geprägte Satz, dass seine Kunden ein Auto jeder Farbe bei ihm erhalten können, solange diese Farbe schwarz sei, ist längst überholt. Die technischen Möglichkeiten, ein Produkt ohne aufwändige Anpassungen an den Produktionsanlagen in vielen Varianten herzustellen, nehmen zu und Kunden können statt aus einem schmalen Produktangebot auszuwählen aus einem Angebot vieler Einzelkomponenten ein Produkt nach Maß zusammenstellen oder vorgegebene Produkte individuell anpassen. Diese Form der Massenproduktion führte zu dem Begriff „mass customization“[1] [Dav96] und sorgt für eine nahezu beliebige Vielfalt an Produkten, die auf den Markt gelangen können. Insbesondere die Automobilbranche bietet ihren Kunden selbst beim komplexen Produkt „Auto“ seit vielen Jahren Fahrzeuge, die sie nach ihren eigenen Vorstellungen konfigurieren können.

Technische Voraussetzungen

Diese starke Erhöhung der Vielzahl und Vielfalt möglicher Produkte stellt einen großen Nutzen für den Kunden dar und sichert damit den Unternehmenserfolg. Unternehmensintern müssen zur Bereitstellung dieser Produktpallette allerdings große Anstrengungen unternommen werden [Sch05].

Einerseits müssen die Produktionsanlagen die technischen Voraussetzungen erfüllen, um Produkte mit einer enormen Variantenvielfalt herstellen zu können. Andererseits müssen die internen Dokumentationssysteme die komplexen Produktstrukturen abbilden können, um den Umgang mit der Variantenvielfalt beim Bestellvorgang im Vertrieb, bei der Instandhaltung im Service sowie in Entwicklung und Produktion zu gewährleisten. Die Strategie der „mass customization“ muss also in allen genannten Bereichen durch ein entsprechendes Konfigurationsmanagement unterstützt werden.

[1] „mass customization“ = mass production + customization

Instandhaltung und Service

Besonders die Instandhaltung eines Produkts im Kundenservice stellt große zusätzliche Anforderungen an ein solches Konfigurationsmanagement, da Kunden hier mit unterschiedlichsten Wartungsaufträgen Einfluss auf die Weiterentwicklung und Anpassung ihres Produkts nehmen. Die Individualisierung der vielen, durch die Konfigurationsmöglichkeiten im Verkauf entstandenen Produktvarianten setzt sich hier fort und kann sich sogar verschärfen. Wegen dieser erheblichen Bedeutung der Instandhaltung im Produktlebenszyklus ist eine Weiterentwicklung des Konfigurierens zum Rekonfigurieren und eine Integration der Instandhaltung im Automobil-Service in die Produktentstehung eine aktuelle Herausforderung.

1.2 Allgemeine Einordnung des Problems

Ein Automobil ist ein hochgradig komplexes Produkt und besteht aus vielen mechanischen und elektrischen Teilen, Elektronikkomponenten und einem weiterhin stetig zunehmenden Anteil an Softwarekomponenten. Die Zusammensetzung eines konkreten Fahrzeugs aus diesen Bestandteilen wird Konfiguration genannt *(Konfigurieren)* und durch Veränderung beliebiger Elemente der Konfiguration entsteht eine neue Konfiguration *(Rekonfigurieren)*.
Grundlage für die Bildung einer neuen Konfiguration und die Umgestaltung einer bestehenden Konfiguration ist einerseits eine *Produktdokumentation*, die entsprechend der Komplexität des Produkts unterschiedlich umfangreich sein muss, und andererseits eine vorausschauende *Produktgestaltung*, die die möglichen Varianten beziehungsweise Konfigurations- und Rekonfigurationsmöglichkeiten eines Produkts festlegt.

1.2.1 Konfigurations- und Rekonfigurationsklassen

Für jedes Unternehmen bzw. Produkt muss das Konfigurations- und Rekonfigurationsproblem sehr spezifisch betrachtet werden, da jeweils unterschiedliche Gründe für die Komplexität beim Konfigurieren und Rekonfigurieren vorliegen. Kriterien für eine Klassifizierung sind die Anzahl der angebotenen Produkte und die Produktkomplexität (Quantität) sowie die Möglichkeiten der Konfigurier- und Rekonfigurierbarkeit (Qualität)[2]. Je höher die Komplexitätsklasse, desto umfangreicher sind die Anforderungen an die Produktdokumentation und -gestaltung. Die Zusammenhänge zwischen der Produktgestaltung, der Konfigurations- und Rekonfigurationskomplexität und der Produktdokumentation sind in Abb. 1.1 dargestellt.

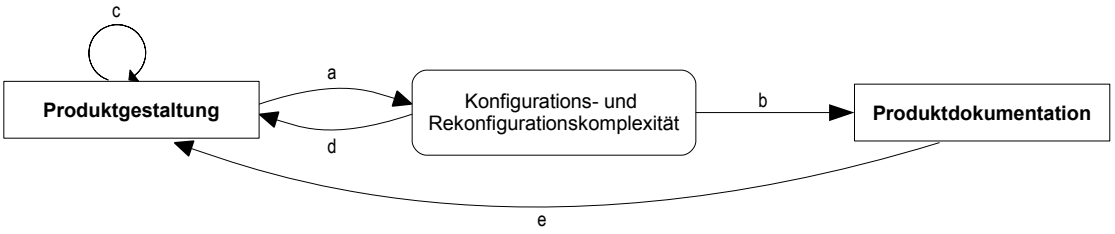
[2]Bezüglich der Qualität werden Unternehmen nach [Sch05] weiterhin in „Variantenkonfigurierer“ (agieren in Märkten, die kundenspzifische Lösungen, also individuelle Konfigurationen fordern) und „Variantenoptimierer“ (Produkte mit Seriencharakter und Planung der Variantenvielfalt der Endprodukte) eingeteilt. Automobilunternehmen sind theoretisch Variantenoptimierer, fallen aber durch die Individualisierung der Produkte, lange Laufzeiten der Fahrzeugmodelle und ihre Serviceaktivitäten mehr und mehr in die Kategorie der Variantenkonfigurierer.

Produktanzahl und -komplexität

Unternehmen mit einer sehr breiten Produktpalette oder komplexen Produkten haben in der Regel größere Schwierigkeiten beim Konfigurieren ihrer Produkte als Unternehmen, die nur mit einer geringen Anzahl von Produkten oder mit Produkten geringer Komplexität im Markt vertreten sind.

Automobilunternehmen bieten sehr viele verschiedene Produkte an, ebenso PC-Anbieter, Unternehmen der Unterhaltungselektronik, Möbelhersteller oder auch Reiseunternehmen. Anbieter speziellerer Produkte wie Schiffsbauer oder Hersteller spezieller Werkzeugmaschinen bieten weniger Produkte an und haben in der Regel geringere Schwierigkeiten beim Konfigurieren[3].

Der Grad der Produktkomplexität ist vor allem durch die Anzahl und Verschiedenheit der Teile in einem Produkt sowie deren Beziehungen untereinander bestimmt. Die Produktanzahl wird dagegen bestimmt durch die Anzahl verschiedener Produkte sowie die Anzahl der Varianten und Versionen eines Produkts. Dies kann sowohl auf *interne* als auch *externe* Ursachen zurückgeführt werden[4], wobei die interne Komplexität durch Unternehmensanforderungen und die externe Komplexität durch Kundenanforderungen verursacht wird [Sch05], [PBvdL05].



a) Die Produktgestaltung bestimmt die Komplexität
b) Die Komplexität muss durch die Produktdokumentation erfasst werden
c) Heutige strategische Entscheidungen zur Produktgestaltung beeinflussen zukünftige Entscheidungen zur Produktgestaltung
d) Ist die Komplexität nicht mehr handhabbar, kann dies Einfluss auf die Produktgestaltung haben
e) Kann die Produktdokumentation die Komplexität nicht abbilden, kann dies Einfluss auf die Produktgestaltung haben

Abbildung 1.1: Zusammenhang zwischen Produktgestaltung, Komplexität und Produktdokumentation

[3] Abhängig von der Definition des Konfigurierens können auch Anbieter weniger, spezieller Produkte einer großen Konfigurationskomplexität gegenüberstehen. Sie sind oft Vertreter des innovativen Konfigurierens [Gün95] und treffen (eine ebenfalls hohe Anzahl an) Konfigurationsentscheidungen auf einer anderen Ebene.

[4] [Sch05] spricht konkret von interner und externer Komplexität. [PBvdL05] spricht konkret von interner und externer Variabilität

Konfigurier- und Rekonfigurierbarkeit

Doch selbst zwischen solchen Unternehmen, die mit einer breiten Produktpalette oder komplexen Produkten am Markt vertreten sind, gibt es wesentliche Unterschiede bei den Konfigurationsschwierigkeiten.
Die bisher betrachtete Anzahl der angebotenen Produkte und die Produktkomplexität als Maß für die Konfigurationsschwierigkeiten ist nicht ausreichend und soll hier weiter in die Anzahl der

- Konfigurationsmöglichkeiten und die Anzahl der
- Rekonfigurationsmöglichkeiten

unterteilt werden. Entsprechend der angebotenen Konfigurationsmöglichkeiten können Unternehmen entweder Standardprodukte oder variable Produkte auf dem Markt anbieten [Wil99]. Viele Unternehmen bieten zwar viele Konfigurationen an, unterstützen aber nicht die Rekonfiguration[5]. Wurde ein Produkt an einen Kunden geliefert, kann ein Unternehmen eine Instandhaltung für das Produkt anbieten oder keine Wartung des Produkts im weiteren Lebenszyklus vornehmen (Wegwerfprodukte). Wird eine Instandhaltung angeboten, kann diese durch Wiederherstellungs- oder Aufrüstmaßnahmen umgesetzt werden. In Abb. 1.2 sind Automobilunternehmen im Vergleich zu anderen Branchen entsprechend ihrer Konfigurations- und Rekonfigurationsklasse eingeordnet[6]. Im Vergleich zu PC-Herstellern ist zu beachten, dass im Automobil durch fehlende, einheitliche Abstraktionsschichten weit mehr Restriktionen bezüglich der Konfigurations- und Rekonfigurationsmöglichkeiten bestehen und der benötigte Automatisierungsgrad wesentlich höher sein muss, um Fehlkonfigurationen und -rekonfigurationen zu verhindern.

1.2.2 Anforderungen im Automobil-Service

Das primäre Ziel im Automobil-Service ist die Durchführung von

- fehlerfreien,
- kundengerechten und
- wirtschaftlichen

Reparaturen bei Kundenfahrzeugen. Von einer Reparatur können beliebige Teile eines Fahrzeuges betroffen sein, wobei der Anteil an Software dabei stetig steigt, da die Menge an Software in den Fahrzeugen wie bereits erwähnt zunimmt und Software zudem schnelleren Änderungszyklen unterliegt.

[5] In [MSTS99] wurde eine Unterteilung in fünf Möglichkeiten der Rekonfiguration vorgestellt und 24 finnische Unternehmen in die entsprechenden Klassen eingeordnet.
[6] Die Abbildung stellt eine ausschließlich qualitative Einordnung der Unternehmen zur Verdeutlichung der Konfigurations- und Rekonfigurationsklassen dar.

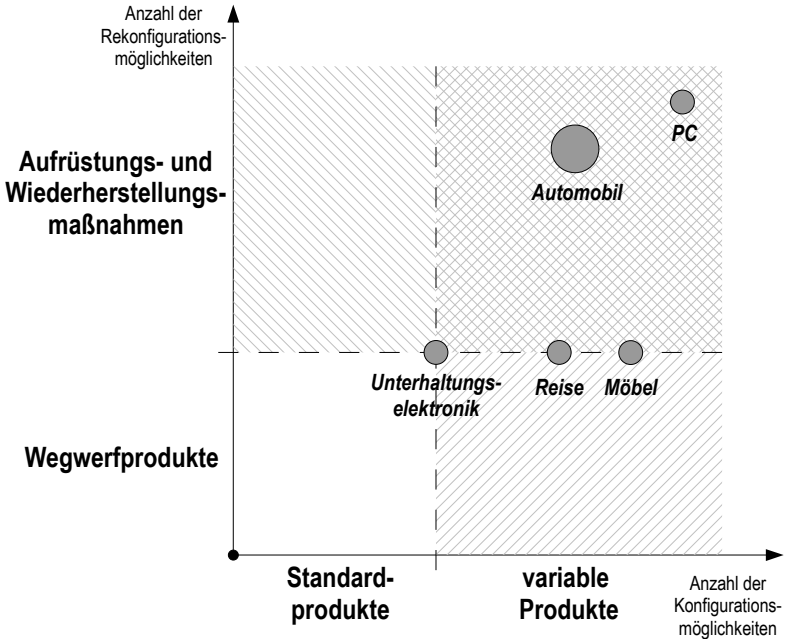


Abbildung 1.2: Einordnung des Automobilbaus nach seiner Konfigurations- und Rekonfigurationskomplexität

Unter einer fehlerfreien Reparatur wird hier verstanden, dass aus einer korrekten Konfiguration nach allen Rekonfigurationsschritten erneut eine korrekte Konfiguration entsteht. Durch die durchgeführten Rekonfigurationsschritte können sich kundenerlebbare funktionale Veränderungen am Fahrzeug ergeben. Sind diese vom Kunden gewünscht oder akzeptiert, war die Reparatur kundengerecht. Zu beachten ist hierbei, dass unterschiedliche Kunden durchaus unterschiedliche Erwartungen an das Ergebnis einer Reparatur desselben Fahrzeugs haben können. Wurden nicht mehr Teile als notwendig oder unnötig teure Teile für die Reparatur verwendet, war diese zudem wirtschaftlich.
Die Herausforderung bei der Erfüllung dieser Anforderungen besteht vor allem in der hohen Anzahl an Ausgangskonfigurationen und der damit verbundenen hohen Anzahl an Reparaturszenarien. Für jeden Kundenfall, also jede Ausgangskonfiguration in Verbindung mit jedem Kundensymptom[7], muss eine fehlerfreie, kundengerechte und wirtschaftliche Reparatur gewährleistet werden. Eine einheitliche Reparatur aller Ausgangskonfigurationen mit einer Zielkonfiguration ist somit nicht möglich. Diese Komplexität erfordert zudem eine möglichst weitgehend automatisierte Bereitstellung von Rekonfigurationsentscheidungen, um Fehler und uneinheitliche Vorgehensweisen bei manuellen Entscheidungen durch das Service-Personal zu vermeiden. Alle Daten, die für die automatisierte Entscheidung notwendig sind, müssen damit zum Zeitpunkt der Reparatur in der Werkstatt vorliegen. Im Unterschied zum Konfigurieren existiert beim Rekonfigurie-

[7] Unter Kundensymptom wird hier die Beanstandung durch den Kunden verstanden.

ren bereits ein Produkt, das neben den sonstigen (Re-)Konfigurationszielen als weitere Randbedingung im (Re-)Konfigurationsprozess berücksichtigt werden muss.
Grundlage für diese im Automobil-Service gestellten Anforderungen sind wie bereits erwähnt die Produktdokumentation und die Produktgestaltung.

Produktdokumentation

Die Produktdokumentation enthält alle zu einem Produkt gehörigen Daten, deren wesentlicher Bestandteil die Produktstruktur, also der Aufbau des Produkts und seiner Varianten, ist. Hierzu muss eine Dokumentationsmethodik festgelegt sein, die die Abbildung der vollständigen Produktkomplexität erlaubt.

Produktgestaltung

Die Produktgestaltung legt dagegen fest, welche Varianten eines Produkts zur Verfügung stehen und hergestellt werden sollen und wie die Fahrzeuge bei Reparaturen behandelt werden. In [San07] wird hierbei von einer expliziten Steuerung des Konfigurationsraums gesprochen. Sie stellt die strategische Komponente dar und baut auf der Produktdokumentation auf, um die Reparaturstrategien zu dokumentieren.

1.2.3 Rekonfigurationsstrategie

Durch die Rekonfigurationsstrategie werden die möglichen Rekonfigurationsszenarien festgelegt. Dabei sind die bereits erwähnten Aspekte aus Kundensicht sowie wirtschaftliche Aspekte aus Unternehmenssicht zu berücksichtigen.

Kundensicht

Zur Herstellung der *Kundenzufriedenheit* muss die Zielkonfiguration den vom Kunden beanstandeten Fehler beheben, darf aber gleichzeitig nur einen adäquaten Funktionsänderungsumfang mit sich bringen. D.h., die Funktionalität der Zielkonfiguration darf sich nur in begrenztem Maße von der der Ausgangskonfiguration unterscheiden, um gewohnte Verhaltensweisen des Fahrzeugs besonders bei sicherheitskritischen Funktionen beizubehalten. Gleichzeitig wird es Ziel der Unternehmensstrategie sein, ältere Fahrzeuge nicht im Rahmen einer Reparatur mit neuen, sonst kostenpflichtigen Funktionen auszustatten.
Um das Kriterium der Wirtschaftlichkeit zu erfüllen, sind die *Rekonfigurationskosten* gering zu halten. Diese setzen sich aus den Kosten für die Austauschteile, hier Steuergeräte, sowie aus der Reparaturzeit zusammen, die bei Softwareupdates den wesentlichen Kostenfaktor in der Werkstatt darstellen.
Beide Kriterien können umso besser erfüllt werden, je höher die Anzahl der möglichen Rekonfigurationsszenarien ist, da durch diese jede Rekonfigurationsanfrage individuell behandelt werden kann.

Unternehmenssicht

Aus Unternehmenssicht ist es sinnvoll, die *Entwicklungskosten* für die Herstellung von Ersatzteilen und die Absicherung von Konfigurationen gering zu halten. Gleichermaßen sollte der *Dokumentationsaufwand* für die Rekonfigurationsszenarien (Rekonfigurationswissen) gering ausfallen.
Diese Kriterien können dann gut erfüllt werden, wenn die Anzahl der möglichen Fahrzeugkonfigurationen und damit einhergehend die möglichen Rekonfigurationsszenarien reduziert werden. Besonders die Notwendigkeit, Abhängigkeiten zwischen Produktmerkmalen beim Rekonfigurieren zu berücksichtigen, erlaubt ein unterschiedliches Vorgehen bezüglich des Umfangs der Änderungen am Fahrzeug.

Der Zusammenhang zwischen der Anzahl der möglichen Rekonfigurationsszenarien und den eben erwähnten Parametern einer Rekonfigurationsstrategie ist in Abb. 1.3 dargestellt.

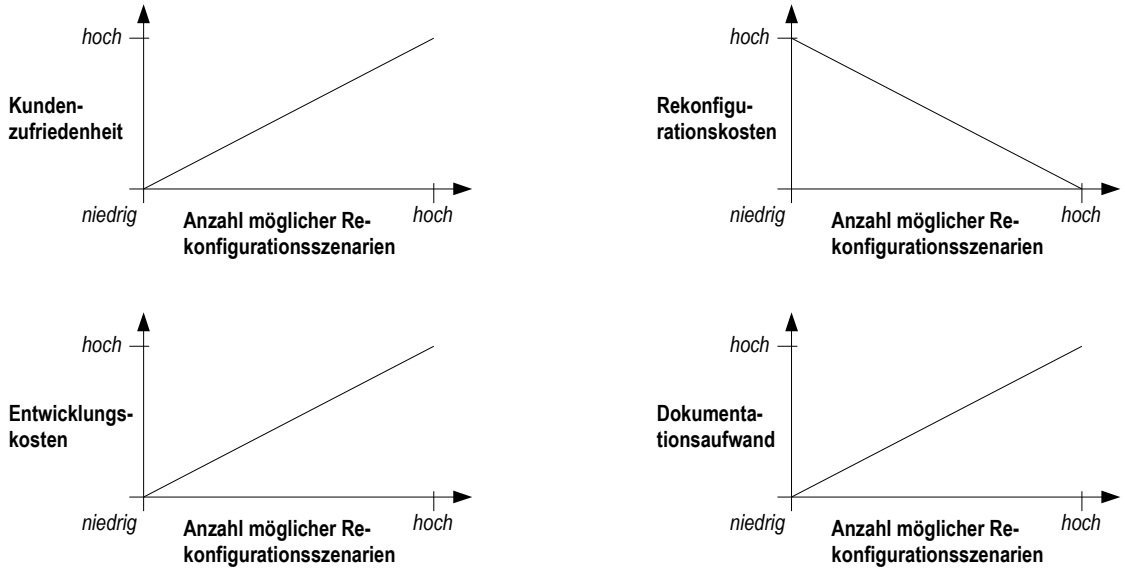


Abbildung 1.3: Einflussgrößen der Rekonfigurationsstrategie

1.3 Ziele der Arbeit und Methodik

Das Ziel dieser Arbeit ist die Beschreibung einer Methodik, die mehrere Aspekte des Rekonfigurationsproblems im Automobil-Service löst. Im Speziellen werden das Softwareupdate eines Steuergeräts eines Fahrzeugs und die dabei auftretenden Abhängigkeiten zu weiteren Fahrzeugkomponenten betrachtet. Die Methodik besteht aus der Dokumentation des dazu benötigten Rekonfigurationswissens, einem System zur Entscheidungsunterstützung, das die vorläufige Überprüfung des dokumentierten Rekonfigurationswissens

erlaubt sowie aus der automatisierten Auswertung des bereitgestellten Rekonfigurationswissens in der Werkstatt. Bei der Dokumentation des Rekonfigurationswissens sollen die Rekonfigurationsziele und -strategie des Kunden sowie des Unternehmens berücksichtigt werden. Zur allgemeinen Beschreibung des Rekonfigurationsproblems wird zuvor ein Domänenmodell erstellt.
Zur Erstellung des Domänenmodells wurde zunächst das Rekonfigurationsproblem bei der Daimler AG analysiert und verallgemeinert. Darauf aufbauend wurde ein konkretes Rekonfigurationskonzept als Erweiterung zu einem bereits bestehenden Softwareupdatemechanismus der Daimler AG erstellt und in der dort bestehenden Systemlandschaft implementiert. Die hier dargestellte Lösung abstrahiert an einigen Stellen von den realen Gegebenheiten im Unternehmen, um die wesentlichen Problemstellungen besser zu fokussieren.

2 Grundlagen der Rekonfiguration

Im vorliegenden Abschnitt werden Themengebiete im Überblick beschrieben, die sich heute mit der Entwicklung und Wartung komplexer Produkte befassen. Die in der Einleitung beschriebene, zunehmende Individualisierung von Produkten wird erst möglich durch eine modulare Produktentwicklung und erfordert aufwändige Konfigurations- und Rekonfigurationsmechanismen im Vertrieb und im Service.

2.1 Modulare Produktentwicklung

Unternehmen erzeugen heute Produkte nicht aus „einem Guss“, sondern streben in der Regel eine hohe Modularität ihrer Produkte an. Dementsprechend werden die einzelnen Bestandteile eines Produkts möglichst unabhängig voneinander entwickelt und anschließend über die Schnittstellen der Module miteinander so verbunden, dass das gewünschte Produkt entsteht. Ein modularer Aufbau eines Produkts bietet die Vorteile der verteilten Entwicklungsmöglichkeiten, der Wiederverwendbarkeit und Weiterentwicklungsmöglichkeit einzelner Module und der einfacheren Herstellung sowie Wartung des Produkts. Diese Vorteile werden jedoch erkauft durch eine höhere Komplexität bei der Abstimmung und Integration der verteilt entwickelten Module im Entwicklungsprozess, bei der Verwaltung der Produktdaten und bei der Steuerung des Wartungsprozesses. Als Module eines Produkts kommen z.B. physische Komponenten (z.B. Steuergerät), Datenkomponenten (z.B. Softwaremodule), Dokumente (z.B. Bedienungsanleitung) aber auch Dienstleistungen (z.B. Reiseveranstaltungen) in Frage [Sch96], [Her01]. Hier werden Steuergeräte und Softwarekomponenten im Fokus stehen.

2.2 Steuergerätesoftware im Automobil

Das Management von Softwarekomponenten im Produktlebenszyklus eines Fahrzeugs stellt eine große Herausforderung für Automobilunternehmen dar [Rai05], [Bro06], [PBKS07]. Die Software eines Steuergeräts wird heute separat von der Hardware entwickelt und ist im Service austauschbar[1], ohne das Steuergerät selbst tauschen zu müssen. Vorteilhaft ist dies vor allem, da Kosten bei Teileentwicklung und -tausch eingespart und neue Funktionen schneller entwickelt werden können. Die spürbar geringeren Kosten erlauben die Entwicklung zusätzlicher Varianten und Versionen, wodurch ein Anstieg der Kosten in der Dokumentation und im Test zu verzeichnen ist. Zusätzlich bedingt

[1] Bei der Daimler AG ist dieser Prozess seit 2001 im Einsatz [Kre06].

die fehlende physische Präsenz von Softwarekomponenten einen erhöhten Dokumentationsumfang, da diese bei sonst z.B. mechanischen Komponenten Informationen über die Verbaubarkeit in einer gegebenen Situation liefern kann.
Der Austausch der Software wird als *Flashen* bezeichnet und muss entsprechend der sonstigen Ersatzteilversorgung über einen Zeitraum von bis zu 15 Jahren nach Produktionsauslauf sichergestellt werden [Rai05], [KK03], [Her03], [FW03].

2.2.1 Aufbau eines Steuergeräts

Vereinfacht besteht ein Steuergerät aus einer Hardware- und einer Softwarekomponente. Eine Software kann weiterhin aus mehreren Modulen bestehen, so dass zur Reduktion der zum Flashen benötigten Zeit bei Bedarf nur ein Teil der Software ausgetauscht werden kann. Eine zusätzliche Anpassung der Software auf den Einsatzzweck (z.B. Software eines Türsteuergeräts in einem Links- oder Rechtslenker) wird durch eine Variantencodierung ermöglicht. Die Variantencodierung kann durch die manuelle Eingabe kurzer Zeichenketten oder die automatische Ermittlung und Übertragung umfangreicherer Zeichenketten in der Produktion und im Service erfolgen. Ein Fahrzeug besteht damit im Kontext dieser Arbeit aus Steuergeräten und sonstigen Komponenten (siehe Abb. 2.1).

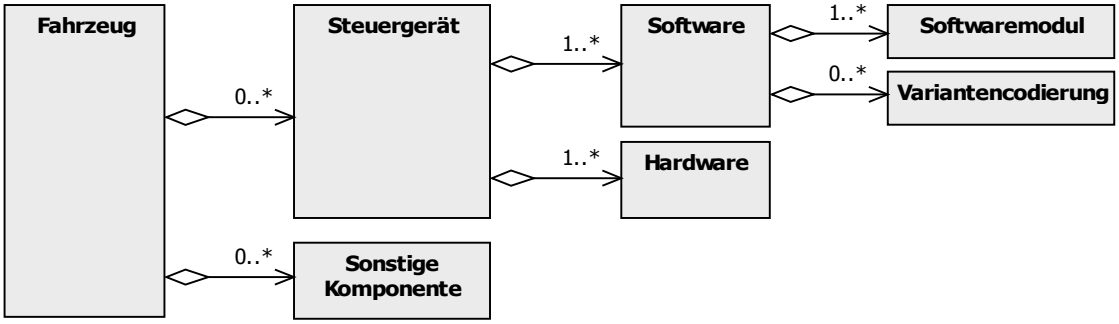


Abbildung 2.1: Steuergeräteaufbau in einem Fahrzeug

2.2.2 Konfiguration zur Laufzeit

Während die Variantencodierung die letzte Möglichkeit ist, die Steuergerätesoftware durch ein externes System (offboard) zu konfigurieren, kann die Software sich selbst noch zum Zeitpunkt der Laufzeit (onboard) durch die Auswertung von Eingangsgrößen auf ihre aktuelle Fahrzeugumgebung adaptieren. Dazu stehen ihr sämtliche durch ihre Schnittstelle zum Fahrzeug anliegenden Signale zur Verfügung; ein externes System kann dagegen ausschließlich die identifizierbaren und offboard dokumentierten Fahrzeugmerkmale in seine Konfigurationsentscheidung einbeziehen. Werden Entscheidungen bei der onboard-Konfiguration und damit das Verhalten einer Software durch die Software eines anderen Steuergeräts im Fahrzeug beeinflusst, muss dies bei der Auswahl der anderen

Software berücksichtigt werden[2].

Bevor eine solche Konfiguration oder Rekonfiguration von Steuergeräten oder Softwarekomponenten im Service möglich wird, sind umfangreiche Vorbereitungen, die als *Softwarelogistik* bezeichnet werden, zu treffen.

2.2.3 Softwarelogistik

Über die Softwarelogistik werden die Dokumentations- und Datenversorgungsprozesse bereitgestellt, die ab der Entwicklung einer neuen Software bis zu deren Einsatz im Service notwendig sind (siehe Abb. 2.2). Eine Software wird in der Entwicklung erstellt und anschließend auf einem Softwareserver zur Verfügung gestellt. Kunde des Softwareservers sind die Produktion und der Service, die für ihre Tätigkeiten über die Software jedes Steuergeräts verfügen müssen. Parallel dazu wird die Software durch den Entwickler selbst und einen unterstützenden Produktdatenmanager im Produktdatenmanagementsystem (PDM-System) dokumentiert. Insbesondere wird die Software in die Stückliste entsprechend ihrer geplanten Verwendung eingetragen und nach Abschluss aller nötigen Dokumentationsschritte und Tests freigegeben. Durch die Angabe der geplanten Verwendung stehen die benötigten Steuerdaten zur Verfügung, die in der Produktion und im Service zum Einsatz der korrekten Software für das jeweilige zu produzierende oder zu reparierende Fahrzeug benötigt werden. Neben dem Entwickler und dem Produktdatenmanager sind die Produktions- und Servicesteuerung wesentlich an der Erstellung der Steuerdaten für ihren jeweiligen Bereich beteiligt.

Daneben müssen Informationen über bereits bestehende Fahrzeugkonfigurationen gespeichert werden, die im späteren Rekonfigurationsprozess in die Ermittlung der korrekten Rekonfigurationsmaßnahme (z.B. Softwaretausch) eingehen. Diese Informationen enthalten Aussagen über die aktuellen Hard- und Softwareversionen der im Fahrzeug verbauten Steuergeräte sowie über weitere Ausstattungsmerkmale des Fahrzeugs. Sie werden aus der Fahrzeugdatenbank bezogen bzw. bei Bedarf direkt aus dem Fahrzeug ausgelesen. Wird eine Fahrzeugkonfiguration geändert, initial also in der Produktion, werden diese Änderungen wiederum in der Fahrzeugdatenbank abgelegt, um jederzeit Kenntnis über die aktuelle Fahrzeugkonfiguration zu besitzen. Durch die stetige Weiterentwicklung von Komponenten stellen die Automobilunternehmen den Werkstätten fortlaufend eine aktualisierte Wissensbasis mit Steuerdaten für die Rekonfiguration zur Verfügung [SS08], [Man07], [Rai05], [Bad04]. Relevant im Rahmen dieser Arbeit ist die Erstellung dieser Steuerdaten.

[2] Eine Beschreibung über den Zeitpunkt, zu dem variantenbildende Module zusammengesetzt werden (*Binding Time*) ist in [Sva00] zu finden.

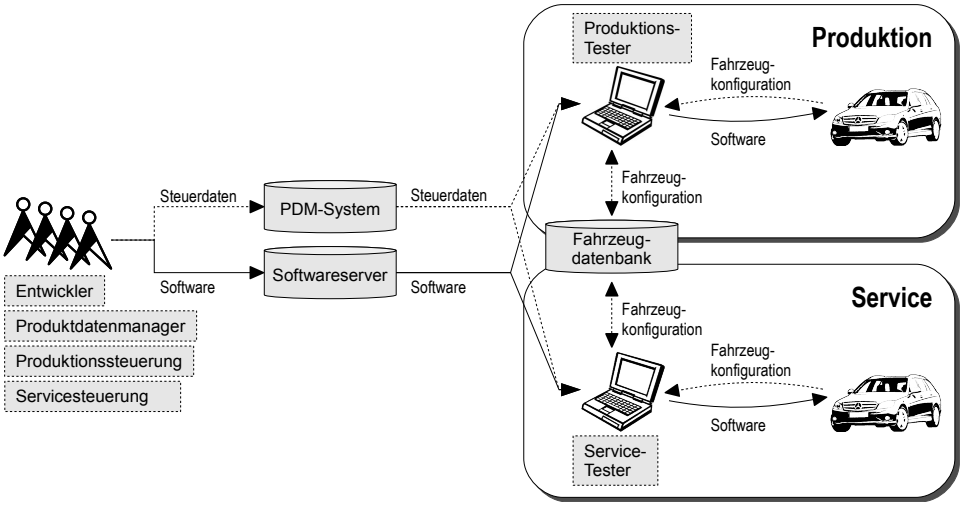


Abbildung 2.2: Softwarelogistikprozess

2.2.4 Ermittlung einer Steuergerätesoftware

Soll ein Fahrzeug in einer Werkstatt mit einer neuen Software versehen werden, werden dazu die zuvor im Softwarelogistikprozess zur Verfügung gestellten Steuerdaten ausgewertet und darauf basierend die Software für das konkret vorliegende Fahrzeug ermittelt. Dazu wählt der Werkstattmitarbeiter das Steuergerät aus, das eine neue Software erhalten soll. Der Algorithmus zur Ermittlung der Software befindet sich im Service-Tester. Mit dem aktuellen Stand des vorliegenden Systems werden Softwarekomponenten ermittelt, die kompatibel zum vorliegenden Fahrzeug sind. Das genaue Vorgehen dabei ist als *Passive Rekonfigurationsphase* in Abschnitt 4.4.2 beschrieben.
Um die möglichen Rekonfigurationsszenarien zu erweitern und zusätzlich zur ursprünglichen Softwareanpassung weitere Anpassungen vorzunehmen, müssen die Steuerdaten sowie der auswertende Algorithmus erweitert werden. Für dieses Problem gibt es bei verschiedenen Automobilherstellern bereits Ansätze oder es werden Lösungen entwickelt (siehe z.B. [MT08]).

2.3 Entwicklungsmodelle

Im oben gezeigten Softwarelogistikprozess treten Probleme des Konfigurationsmanagements zu Tage, die in allen Unternehmensbranchen eine Rolle spielen, bei denen komplexe Produkte hergestellt und über den gesamten Produktlebenszyklus verwaltet werden müssen. Modelle zur Unterstützung der System- und Softwareentwicklung wie CMMI (Capability Maturity Model Integration) und SPICE (Software Process Improvement and Capability dEtermination, ISO/IEC 15504) [Kne07], [Ins03b], [Ins03c] sind besonders in der Automobilindustrie verbreitet und machen, neben vielen weiteren im Entwick-

lungsprozess relevanten Aspekten, Vorgaben darüber, wie ein Konfigurationsmanagement angelegt sein sollte. Auch im Software-Produktmanagement [SHT05], der ITIL (IT Infrastructure Library) [Kne07], [Ins03a] und dem PLM (Product Lifecycle Management) [ES01], [ADEK05], [KSR07] werden Methoden zur Verwaltung von Produkten über deren Lebenszyklus hinweg beschrieben und das Konfigurationsmanagement als zentraler Bestandteil integriert.

2.4 Konfigurationsmanagement und Produktdokumentation

Standards zum Konfigurationsmanagement existieren bereits seit den 1950er-Jahren. Sie unterstützen Unternehmen und Behörden im Umgang mit komplexen Produkten. Durch die Zunahme des Softwareanteils in Industrieprodukten wurde Software später explizit als Aspekt im Konfigurationsmanagement berücksichtigt (1971, MIL-STD 483) [Wei00], [Pop06]. Heute wird das (Software-)Konfigurationsmanagement z.B. in den Standards MIL-STD-973, ANSI/EIA-649-1998 und ISO 10007:2003 definiert, die weitgehend die Grundlage für die Umsetzung eines Konfigurationsmanagements in Industriebetrieben liefern [Wei00]. Insgesamt existierten 2002 nach [BA02] über 200 Standards für diese Disziplin, was die Vielfalt der Anforderungen in diesem Bereich deutlich macht. In [Est00] wird das Software Configuration Management (SCM) wie folgt definiert:

> *SCM is the control of the evolution of complex systems.*[3]

Bis heute haben sich die Anforderungen an das Konfigurationsmanagement stetig verändert und eine klare Abgrenzung der zum Konfigurationsmanagement gehörigen Konzepte ist nicht möglich. Dennoch können einige Kernbestandteile des Konfigurationsmanagements herausgehoben werden. In allen der oben genannten Standards werden die vier folgenden Aktivitäten des Konfigurationsmanagements definiert [SHT05]:

1. Identifikation (*identification*)
2. Steuerung (*control*)
3. Statusbericht (*status accounting*)
4. Audit (*auditing*)

Eine aus Sicht dieser Arbeit wichtige Erweiterung der oben genannten Aktivitäten um unter anderem

5. Produktion (*manufacture*)

[3] SCM dient der Steuerung der Weiterentwicklung komplexer Systeme.

wurde durch [Dar91] vorgenommen.
Obwohl das Konfigurationsmanagement vor allem wichtig für den Entwicklungsprozess ist, darf dessen Bedeutung für die nachgelagerte Wartungsphase nicht vernachlässigt werden. Innerhalb der Aktivitäten Identifikation (1.) und Steuerung (2.) entstehen bereits während der Entwicklung Produktinformationen, die für die Rekonfiguration im Service genutzt werden können. Bei der Identifikation werden die Bestandteile einer Konfiguration festgelegt und die Struktur und Beziehungen dazwischen dokumentiert[4] (siehe hierzu Abschnitt 2.5). Durch die Steuerung werden Änderungsanforderungen an einem bestehenden Produkt erfasst und damit die für die Rekonfiguration wichtige Komponenten- und Produkthistorie dokumentiert (siehe hierzu Abschnitt 2.6). In der Produktion (5.) werden konkrete Produkte, basierend auf den zuvor erstellten Produktdaten, abgeleitet. Die genannten Aktivitäten des bereits in vielen Unternehmen etablierten Konfigurationsmanagements bilden eine Grundlage für die Rekonfiguration im Service, müssen für deren vollständige Unterstützung aber erweitert werden.
Neben dem Konfigurationsmanagement beinhaltet die Disziplin des Produktdatenmanagements (PDM) [ES01] die Dokumentation einer Produktstruktur. Ein Vergleich beider Disziplinen ist in [EFM98] und [EV07] zu finden. Darüberhinaus wurden und werden unter den Begriffen *Konfigurieren* und *(Software-)Produktlinien* Methoden zur Beschreibung von Produkten in Konfigurationmodellen und deren Ableitung basierend auf einer konkreten Kundenanforderung bereitgestellt. Die wichtigsten Grundlagen dieser Produktdokumentation werden im Folgenden vorgestellt.

2.5 Modellierung von Produkten

Die Produktstruktur hat als wesentliches Ziel, die in einem Produkt enthaltenen Komponenten strukturell zu gliedern. Damit ist sie ein wesentlicher Teil der Produktdokumentation und kann bei durchdachter Wahl viele Sichten eines Unternehmens abbilden, z.B. die Fertigungs-, Vertriebs- und Ersatzteilsicht [Sch05].

2.5.1 Kompositions- und Taxonomiebeziehung

Grundsätzlich können die einzelnen Konfigurationseinheiten einer Konfiguration durch eine Kompositionsbeziehung (*has-parts-Relation*) oder eine taxonomische Beziehung (*is-a-Relation*) zu einer Produktstruktur zusammengesetzt werden [EFM98]. [Gün95] spricht bei der Produktstruktur auch von einer Begriffshierarchie, die sich entsprechend aus der Zerlegungs- oder Spezialisierungshierarchie zusammensetzt. Einzelne Konfigurationseinheiten werden als Konzepte und speziell als Ober- bzw. Unterkonzepte bezeichnet.

[4] In [Dar91] wurden neben den oben genannten Aktivitäten acht *Aufgabengebiete* innerhalb des Konfigurationsmanagements abgegrenzt. Das Aufgabengebiet *Structure* umfasst dabei die Modellierung der Produktstruktur einschließlich Abhängigkeitsbeziehungen zur Sicherstellung der Konsistenz aller ableitbaren Konfigurationen.

Die Kompositionsbeziehung besteht zwischen einer Konfigurationseinheit und ihren Einzelbestandteilen, wie z.B. zwischen einem Motor und seinen Einzelteilen Motorblock und Motorsteuergerät. Das Motorsteuergerät kann weiterhin zerlegt werden in die Steuergerätehardware, die Motorsoftware usw. Ein Produkt kann durch die Kompositionsbeziehung beliebig und abhängig von der benötigten Sicht in seine Bestandteile zerlegt werden (siehe Abb. 2.3[5]). Nach [Sch05] und [Sch04] ist bei komplexen Produkten eine Strukturtiefe von 7 optimal. Eine ausführliche Diskussion der Kompositionsbeziehung ist in [Män98] zu finden.
Durch die Taxonomiebeziehung kann dagegen ausgedrückt werden, welche Möglichkeiten zur Realisierung einer Konfigurationseinheit existieren. Eine is-a-Relation zwischen einem Ober- und seinen Unterkonzepten sagt aus, dass das Oberkonzept wahlweise durch eines seiner Unterkonzepte realisiert werden kann. Die Unterkonzepte können als Varianten des Oberkonzeptes verstanden werden. Beispielhaft hierfür sind zwei unterschiedliche Motorvarianten mit sechs und acht Zylindern, für die eine jeweils eigene Spezialisierung des Parametermoduls der Motor-Software zur Verfügung stehen muss (siehe Abb. 2.3).
Im rechten Teil von Abb. 2.3 ist eine andere mögliche Strukturierung desselben Produkts dargestellt. Wie die Produktstruktur aussieht, hängt von der Sicht auf das Produkt und den Variationspunkten im Produkt ab.

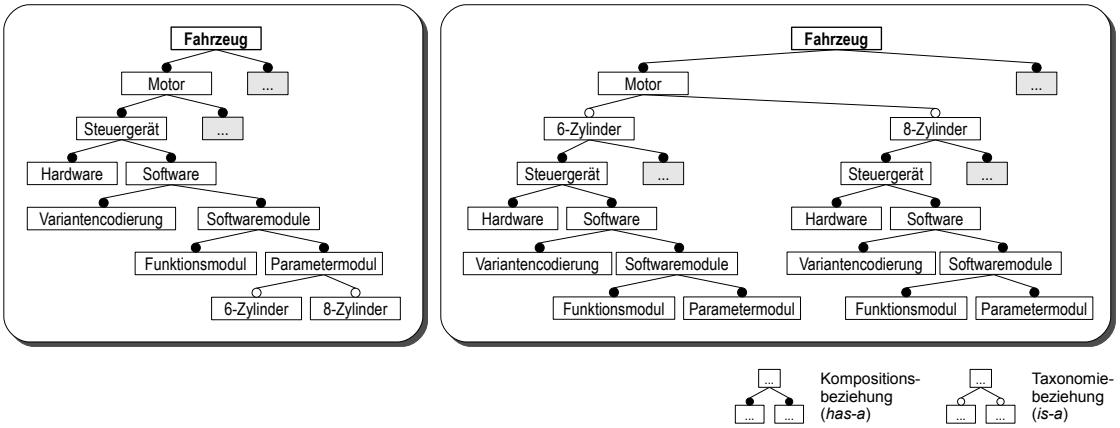


Abbildung 2.3: Produktstrukturen mit Kompositions- und Taxonomiebeziehungen

2.5.2 Merkmalmodelle

Auch im Bereich der Softwareproduktlinien besteht während der Phase der Domänenanalyse [PBvdL05] die Notwendigkeit, die Variabilität innerhalb einer Produktfamilie zu beschreiben und damit die möglichen Produkte zu modellieren. Ziel ist es, eine Familie von Produkten zu entwickeln, die auf gemeinsamen, wiederverwendbaren Kernkompo-

[5] Die Abbildung zeigt nur einen Ausschnitt der gesamten Produktstruktur. Komponenten, die mit „...“ bezeichnet sind, stehen stellvertretend für alle nicht explizit dargestellten Komponenten.

nenten basieren und diese Basis durch zusätzliche Entwicklungen und Anpassungen zu einem konkreten Produkt auszuprägen. Häufig wird eine Produktfamilie während der Domänenanalyse durch Merkmale beschrieben und es entsteht ein Merkmalmodell basierend auf der FODA-Notation (Feature Oriented Domain Analysis) [KCH+90] oder Erweiterungen dazu, z.B. in [Cza98], [GBS01], [Beu03], [PBvdL05]. Die vorab beschriebenen Kompositions- und Taxonomiebeziehungen werden um semantisch mächtigere Beziehungstypen erweitert und dadurch eine kompaktere Darstellung variantenreicher Produkte unterstützt. Allen bekannten Erweiterungen der ursprünglichen FODA-Notation ist aber gemein, dass sie zwar kompaktere Darstellungsformen bereitstellen, die Mächtigkeit der Beschreibungsmöglichkteiten aber nicht vergrößern [BHST04].
Den meisten Notationen gemeinsam sind die folgenden Beziehungstypen (Beschreibung nach [Beu03]):

obligatorisch: Entspricht der Kompositionsbeziehung. Wird das Obermerkmal (bzw. -konzept) ausgewählt, muss auch das Untermerkmal (bzw. -konzept) eingefügt werden.

optional: Wird das Obermerkmal eingefügt, kann das Untermerkmal ebenfalls eingefügt werden.

alternativ: Alternative mit einwertiger Auswahl. Entspricht der Taxonomiebeziehung. Alternative Merkmale werden gruppiert und genau ein Merkmal der Gruppe wird ausgewählt, wenn das Obermerkmal ausgewählt wurde.

or: Alternative mit mehrwertiger Auswahl. Or-Merkmale werden gruppiert und mindestens ein Merkmal der Gruppe wird ausgewählt, wenn das Obermerkmal ausgwählt wurde.

Ein Beispiel dieser Notation ist in Abb. 2.4 zu sehen.

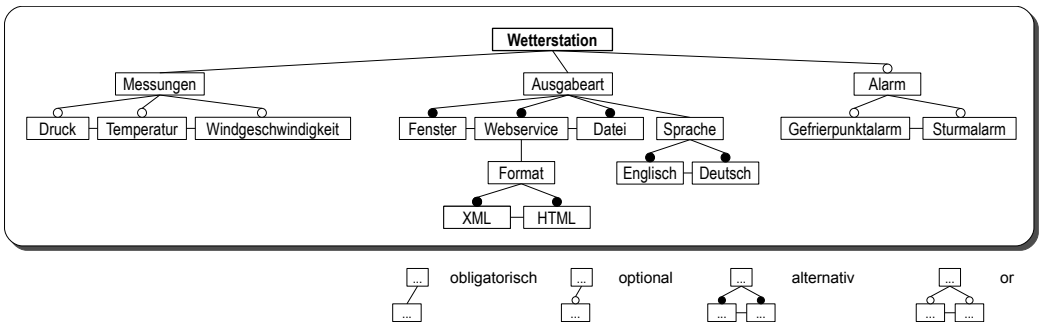


Abbildung 2.4: Merkmalmodell nach [Beu03]

In [Cla01] wurde den Beziehungstypen aus der Produktlinienmodellierung entsprechende UML-Beziehungstypen zugewiesen, so dass eine Modellierung einer Produktlinie auch mittels der UML-Notation möglich ist. Eine Modellierung von Konfigurationsmodellen durch UML wird ebenso in [FFSS08] und [FFJ00] gezeigt.

2.5.3 Einschränkungen im Produktmodell

Grundsätzlich kann eine Produktstruktur ein einzelnes konkretes Produkt oder eine Menge von möglichen Produkten abbilden. [Zel97] spricht in diesem Zusammenhang von abstrakten (*abstract*), generischen (*generic*) und vollständig aufgelösten (*bound*) Konfigurationen. Enthält eine Produktstruktur Variationspunkte, also Taxonomiebeziehungen, ist immer eine Menge von Produkten dargestellt. In Abb. 2.3 werden genau zwei mögliche Produkte beschrieben. Eine Einschränkung der möglichen Produkte kann z.B. durch Abhängigkeits- oder Ausschlussbeziehungen vorgenommen werden, die in der FODA-Notation *requires* und *mutually exclusive with* genannt werden. Weitere Beziehungstypen zur Einschränkung der möglichen Produktmenge werden durch [Beu03] vorgeschlagen: *discouraged* und *recommended*. Beide sind Abschwächungen von *requires* und *mutually exclusive with*, die durch eine nicht explizit modellierte Ursache begründet sind. In [PF89] wird zudem eine *consistency*-Beziehung verwendet, die, sobald sie zwischen allen in einer Konfiguration enthaltenen, abhängigen Elementen existiert, die gesamte Konfiguration als konsistent definiert. Weiterhin wird dort eine *compatibility*-Beziehung definiert, die zwei Untermerkmale als austauschbar bezüglich der von ihnen abhängenden Konfigurationselemente kennzeichnet. In [Pul02] werden die Verträglichkeiten zwischen Produktmerkmalen und -bestandteilen binär durch eine Verträglichkeitsmatrix dokumentiert.

2.5.4 Stücklisten

Stücklisten sind weitere Darstellungsformen, um Produktstrukturen abzubilden. Man unterscheidet zwischen einfachen Stücklisten und Variantenstücklisten. Einfache Stücklisten enthalten die Struktur genau eines Produktes während Variantenstücklisten mehrere verschiedene Produktausprägungen in kompakter Form enthalten. Weiterhin wird zwischen Mengen- und Strukturstücklisten unterschieden, von denen die Mengenstückliste eine Aussage darüber trifft, welche Konfigurationseinheiten Bestandteil eines Produktes sind und wie häufig das jeweilige Element im Produkt vorkommt. Strukturstücklisten enthalten dagegen die gesamte hierarchische Struktur des Produktes. Eine kompaktere Darstellungsform sind Baukastenstücklisten, die nur Konfigurationseinheiten einer Strukturstufe enthalten. Setzt sich ein Element in dieser Strukturstufe aus weiteren Konfigurationseinheiten zusammen, wird die Struktur dieses „Platzhalters“ in einer eigenen Stückliste beschrieben. Variantenstücklisten bilden die Variantenstruktur des Produktes z.B. durch Plus-Minus-Stücklisten oder Variantenstücklisten mit Variantenleisten ab, wobei die Variantenleisten die alternativen Varianten für einen Variantenplatzhalter enthalten [EHS91].

Offensichtlich existieren durch die vorgestellten Notationen verschiedene Möglichkeiten, eine Menge möglicher Produkte kompakt zu beschreiben. Statt jedes einzelne, mögliche Produkt mit all seinen Bestandteilen explizit in einer Liste zu hinterlegen, können diese so anschaulich und angereichert um die Produktstruktur dargestellt

werden.

2.6 Änderungsmanagement

Ein wesentlicher Teil im Konfigurationsmanagement ist das Änderungsmanagement (siehe *Steuerung* in Abschnitt 2.4), welches Änderungen an Produktbestandteilen erfasst und steuert. Jede Weiterentwicklung eines Produktbestandteils wird durch einen Änderungsantrag beschrieben, durchläuft einen definierten Freigabeprozess, wird implementiert und zusammen mit allen zusätzlich damit verbundenen Änderungen am Produkt dokumentiert. Die Dokumentation, um welche Art von Änderung es sich handelt und unter welchen Bedingungen eine Änderung gültig ist, ist neben der Dokumentation der Produktstruktur der für das Rekonfigurieren im Service wesentliche Bestandteil jedes Produktentwicklungsprozesses. Teil dieser Dokumentation ist die Information, in welchen Produkten die neue Komponente unter welchen Bedingungen eingesetzt werden soll. Dazu zählt unter anderem die zeitliche Gültigkeit der Komponente, also z.B. ein Einsatz der neuen Version der Software des Klimasteuergeräts ab dem 01.09.2008. Weiterhin wird die Austauschbarkeit der neuen Komponente gegenüber ihrer oder ihren Vorgängerkomponente(n) dokumentiert [ES01]. In der Norm DIN 6789-3 wird die Austauschbarkeit, wie in Tab. 2.1 angegeben, definiert.
Zur detaillierten Bewertung der Austauschbarkeit von Hard- und Softwareänderungen haben deutsche Automobilhersteller gemeinsam in der Herstellerinitiative Software (HIS) im Arbeitskreis Kompatibilitätsmanagement eine Kriterienliste entwickelt [HIS09]. Anhand dieser genaueren Änderungsangaben können qualifiziertere Aussagen über mögliche Ersetzungen im Service und Auswirkungen der Änderungen auf das Restfahrzeug getroffen werden (*Auswirkungsanalyse*).
Obwohl die Änderungsangaben zu einem Produkt oder einer Komponente nach der aktuellen Definition des Änderungsmanagements für die Rekonfiguration grundsätzlich wiederverwendet werden können, haben sie zunächst nur das aktuell entwickelte Produkt im Fokus. Für die Rekonfiguration muss dieser Fokus erweitert werden und im Änderungsmanagement auch Entscheidungen getroffen werden, wie die durch die Änderung neu entstandene Komponente in bereits bestehenden Produkten bei der Wartung eingesetzt werden kann.

2.7 Rekonfigurieren

Rekonfigurieren ist die Methode zur Überführung einer bestehenden Konfiguration in eine neue Konfiguration unter Berücksichtigung der geänderten Anforderungen. Für die bestehende Aufgabenstellung dieser Arbeit, also die Rekonfiguration von Fahrzeugen, wird die Rekonfiguration durchgeführt, nachdem die Konfiguration als Produkt an einen Kunden ausgeliefert wurde.

Benennung und Definition	Bemerkung
Austauschbarkeit Die Austauschbarkeit ist die Eignung eines Gegenstandes, einen anderen Gegenstand zu ersetzen.	
Eingeschränkte Austauschbarkeit Die eingeschränkte Austauschbarkeit ist die nur bedingte Eignung eines neuen Gegenstandes, einen bisherigen zu ersetzen oder durch einen bisherigen Gegenstand ersetzt werden zu können.	Die zusätzlichen Maßnahmen für Anpassen, Umarbeiten usw. zum Erreichen der Austauschbarkeit müssen auf geeignete Weise dokumentiert werden.
Vollaustauschbarkeit Die Vollaustauschbarkeit ist die Eignung eines Gegenstandes, einen bisherigen Gegenstand zu ersetzen oder durch diesen ersetzt werden zu können.	
Vorwärtsaustauschbarkeit Die Vorwärtsaustauschbarkeit ist die Eignung eines Gegenstandes, einen bisherigen Gegenstand zu ersetzen, ohne jedoch durch diesen in jedem Fall ersetzt werden zu können.	Zum Sicherstellen der Vorwärts-Austauschbarkeit gehört das Beibehalten aller spezifischen Merkmale des bisherigen Gegenstandes. Der neue Gegenstand kann darüber hinaus erweiterte oder neu hinzugefügte Merkmale aufweisen.
Nichtaustauschbarkeit Nichtaustauschbarkeit ist gegeben, wenn ein neuer Gegenstand einen bisherigen nicht ersetzen kann.	Die fehlende Austauschbarkeit von Neukonstruktionen erfordert unterschiedliche Sachnummern. Auch wenn die fehlende Austauschbarkeit eine Folge einer Änderungsmaßnahme ist, wird der neue Gegenstand wie eine Neukonstruktion (Variante) behandelt.

Tabelle 2.1: Definition der Austauschbarkeit nach DIN 6789-3 (zitiert nach [ES01])

2.7.1 Parallele Weiterentwicklung

Nach der Auslieferung des Produkts findet im Unternehmen eine Weiterentwicklung der Produktpalette und damit des Konfigurationsmodells (Produktstruktur) statt. Parallel dazu werden die ausgelieferten Produkte durch Wartungsarbeiten verändert. Diese Weiterentwicklungen der existierenden Produkte können durch den Kunden selbst, Serviceunternehmen oder das Herstellerunternehmen (siehe auch Abschnitt 1.2.1) durchgeführt werden. Die dann durchgeführte Weiterentwicklung eines Produkts kann losgelöst vom Konfigurationsmodell (*Independent Maintenance*), abhängig vom aktuellen Konfigurationsmodell (*Disciplined Maintenance* und *Upgrade or Modernisation*) oder mit entsprechender Anpassung des Konfigurationsmodells (*Advanced Reconfiguration*) stattfinden [MPS96], [Män98].

2.7.2 Rekonfigurationsstrategien

Für die Rekonfiguration von Fahrzeugen muss die *Independent Maintenance* berücksichtigt werden, da die willkürliche Wartung und Aufrüstung von Fahrzeugen besonders in sicherheitskritischen Funktionen verhindert werden muss. Weil eine vollständige Verhinderung nicht möglich ist, müssen solche Fahrzeuge dennoch in die Abschätzung der möglichen Rekonfigurationsszenarien und die Entwicklung der Rekonfigurationsstrategie einfließen. In Nutzfahrzeugsparten (z.B. Lastkraftwagen, Reisebusse), in denen Produkte auftragsbasiert individuell hergestellt werden, werden diese auch auftragsbasiert rekonfiguriert (*Advanced Reconfiguration*). Die Rekonfigurationsaufgabe ist damit in diesem Bereich eine Entwicklungsaufgabe, sollte aber soweit machbar, durch bestehende Komponenten gelöst werden. Dazu ist eine umfangreiche Dokumentation des Wissens über die bestehenden Komponenten und deren Kompatibilitäten notwendig [Man05]. In dieser Arbeit werden Lösungen zur Umsetzung der Strategien *Disciplined Maintenance* und *Upgrade or Modernisation* beschrieben, da hier eine strenge Steuerung der Produktweiterentwicklung durch ein vorhandenes Rekonfigurationsmodell notwendig ist. Dabei ist immer die Optimierung des Rekonfigurationsvorgangs hinsichtlich der entstehenden Kosten einzubeziehen.

2.7.3 Rekonfigurationsbeschreibungen

Explizites Rekonfigurationsmodell

Wie oben bereits erwähnt wurde, überführt eine Rekonfiguration ein bestehendes Produkt in ein neues Produkt. In [MSTS99] wurde ein konzeptionelles Modell für diesen Vorgang entwickelt. Die Beschreibung der Änderungen, die an den existierenden Produkten vorgenommen werden sollen, geschieht in einem *Rekonfigurationsmodell*, das aus Rekonfigurationsoperationen und Rekonfigurationsinvarianten besteht. Die Rekonfigurationsoperationen sind logische Regeln, bestehend aus einer Vorbedingung und einer Aktion. Durch Aktionen werden entweder Komponenten selbst oder Bezie-

hungen zwischen ihnen modifiziert. Rekonfigurationsinvarianten stellen Bedingungen (*constraints*) dar, die durch die Rekonfigurationsoperationen nicht verletzt werden dürfen (z.B. „Komponente A benötigt mindestens Komponente B“).

Während im eben genannten Modell die Änderungen am Produkt explizit für die Rekonfiguration beschrieben werden, gibt es Ansätze, die die bestehenden Methoden zur modellbasierten Diagnose ([CR91], [CR94], [SW98], [FFJZ01], [FFJ01]) oder zur Konfiguration ([KR99]) weitestgehend auf dem Gebiet der Rekonfiguration wiederverwenden.

Rekonfiguration als modellbasierte Diagnose

Beim Ansatz der modellbasierten Diagnose zur Rekonfiguration ist das Zielsystem als Modell beschrieben und der Unterschied zwischen dem Modell und dem bestehenden Produkt sowie die notwendigen Rekonfigurationsmaßnahmen werden ermittelt. Zum Teil ([SW98]) werden bei diesem Vorgehen nur Hinweise erzeugt, an welchen Stellen des Produkts Änderungen vorgenommen werden müssen, um die neuen Anforderungen zu erfüllen, wodurch das Rekonfigurationsproblem nicht vollständig gelöst wird.
Ein ähnlicher Ansatz wird in [Hei04] beschrieben, indem das Zielprodukt der Rekonfiguration explizit vorgegeben wird und anschließend Aktualisierungsanweisungen generiert werden, durch die die Zielkonfiguration ausgehend von der vorliegenden Konfiguration erreicht werden kann.

Rekonfiguration als Konfigurationsproblem

Beim Konfigurieren wird ein konkretes Produkt aus einem Konfigurationsmodell abgeleitet. Die dazu notwendigen Entscheidungen über die Auswahl der gewünschten Varianten einzelner Komponenten werden üblicherweise durch den Benutzer getroffen. Können Benutzervorgaben auf Grund von Einschränkungen im Konfigurationsmodell nicht mehr erfüllt werden, wird durch Backtracking nach Alternativen gesucht (siehe z.B. [Gün95]). Der Rekonfigurationsprozess kann nun auf dieses Vorgehen abgebildet werden [KR99], indem

- fehlerhafte Komponenten entfernt oder
- gewünschte Funktionalitäten hinzugefügt

werden und der Konfigurationsprozess basierend auf diesen neuen Anforderungen neu gestartet wird. Nach dem Einfügen der Ersatz- oder Zusatzkomponente(n) unter Minimierung der Konflikte mit dem bestehenden System werden die verbleibenden Konflikte durch strategisch gesteuertes Backtracking gelöst.
Gleichermaßen kann das Rekonfigurationsproblem durch Konfigurationstechniken gelöst werden, wenn das Konfigurationsmodell vollständig durch Constraints beschrieben ist. In [JG99], [Joh02] werden Rekonfigurationsoperationen durch *Improvement Constraints*

ausgedrückt, die z.B. Anforderungen an das Entfernen oder Hinzufügen von Komponenten beinhalten. Unter Einhaltung von kontextunabhängigen Einschränkungen (*constraints*) wird dann ein neues Produkt aus dem bestehenden abgeleitet.
In [Wil99] werden Ansätze zur nachträglichen Anpassung von Produkten beschrieben, welche durch fallbasierte Suche (*case-based reasoning*) ermittelt werden, aber die Kundenanforderungen nicht vollständig erfüllen. Hierbei wird gleichzeitig auf das *Negotiation Problem* hingewiesen. Verschiedene Kunden formulieren danach ihren Rekonfigurationswunsch unterschiedlich, weshalb eine Zuordnung einer Kundenanfrage auf ein unterstütztes Rekonfigurationsziel vorgenommen werden muss.

2.7.4 Offene Anforderungen

Offensichtlich existieren unterschiedliche Ansätze zur Lösung des Rekonfigurationsproblems. Die meisten Ansätze unterstützen neben der einfachen Wiederherstellung einer Konfiguration (Reparatur) auch eine Aufrüstung. Die Steuerbarkeit der einzelnen Rekonfigurationsszenarien ist aber oftmals nicht detailliert genug möglich, da global gültige Rekonfigurationsstrategien wie z.B. die Optimierung nach der Anzahl der notwendigen Änderungsmaßnahmen angewendet werden. Wird eine exakte Beeinflussung eines Rekonfigurationsszenarios durch z.B. explizite Rekonfigurationsoperationen unterstützt, ist kein Konzept zur Minimierung des Dokumentationsaufwands für diese Regeln definiert.
Neben der expliziten Steuerung einzelner Rekonfigurationsszenarien muss weiterhin ein Konzept zur Unterstützung der automatisierten Rekonfiguration für eine sehr hohe Zahl an Rekonfigurationsszenarien abhängig vom jeweiligen Ausgangsprodukt definiert werden. Viele der oben erwähnten Ansätze gehen dagegen bei der Rekonfiguration von einem interaktiven Prozess für ein Einzelprodukt aus, in dem der Kunde Entscheidungen zur Rekonfiguration durch vorhandenes Domänenwissen treffen kann und auch muss. Die automatisierte Rekonfiguration benötigt zusätzlich ein Rekonfigurationsmodell, in dem eine Kundenanfrage als Auslöser für die Rekonfiguration berücksichtigt wird. Das Verfolgen einer einheitlichen Unternehmensstrategie bei der automatisieren Rekonfiguration und das fehlende technische Wissen des Kunden erfordert zudem ein Konzept für die Verteilung des Rekonfigurationswissens auf mehrere Beteiligte.
Teile des Rekonfigurationsproblems können damit immer noch als offene Probleme betrachtet werden [Man05], [Stu08]. Besonders die Dokumentation von Rekonfigurationswissen und Abhängigkeiten zwischen Produktmerkmalen zur automatisierten Auswertung ist zu lösen [Man05].

2.8 Praktische Beispielsysteme zur Rekonfiguration

2.8.1 Automobilbranche

Wie bereits erwähnt sind Rekonfigurationswerkzeuge bereits bei Automobilkonzernen im Einsatz. Neben den Anwendungen im Service-Tester, die neue Softwarekomponenten für

ein Fahrzeug ermitteln, stellen Teilekataloge ebenfalls Austauschinformationen zu Fahrzeugkomponenten, hier speziell physische Komponenten, bereit. Diese enthalten jedoch, zumindest für das vorliegende Beispielsystem, keine Möglichkeiten zur Ermittlung abhängiger Komponenten beim Flashen von Softwaremodulen.

2.8.2 PC-Branche

Dagegen werden Abhängigkeiten seit langem durch Programme wie z.B. verschiedene Linux Paketmanager in der PC-Branche automatisch aufgelöst. Weitere Beispiele sind die Plugin-Verwaltung der Eclipse-Plattform (The Eclipse Foundation) oder die automatische Updatefunktion in Microsoft Windows Betriebssystemen, Mozilla Firefox (The Mozilla Foundation) und vielen anderen PC-Applikationen. Grundsätzlich sind bei diesen Applikationen zwei Kategorien des Rekonfigurationsablaufs zu unterscheiden. Bei Rekonfigurationen der ersten Kategorie gibt der Anwender den Auftrag, eine bestimmte Zielsoftware zu seinem bereits bestehenden System hinzuzufügen. Bei Rekonfigurationen der zweiten Kategorie lässt der Anwender nach Updates für eine bestehende Applikation suchen. Systeme wie der Linux Paketmanager oder die Eclipse-Plattform unterstützen explizit beide Kategorien, Microsoft Windows Automatische Updates und Mozilla Firefox unterstützen explizit[6] nur die zweite Kategorie.
Im Fall eines Linux Paketmanagers werden die Abhängigkeiten der zu installierenden Software zu anderen Softwarepaketen üblicherweise in Form exakter Beziehungen (z.B. Software A, v1.3.1 benötigt Software B, v2.0.0) oder durch mathematische Operatoren angegeben, die sich auf die Version der Zielsoftware beziehen und dadurch eine Menge möglicher benötigter Softwarepakete definiert (z.B. Software A, v1.3.1 benötigt Software B in der Version $\geq$ v2.0.0). Die zur Anwendung dieser durch den mathematischen Operator angegebenen Relation benötigte Ordnung in der Menge der Zielsoftwarepakete wird hier implizit durch die Versionsnummerierung festgelegt.
Sucht der Anwender nach einem Update eines kommerziellen Produktes, werden ihm, abhängig vom Lizenzmodell des Herstellers, ggf. nur solche Versionen zur Aktualisierung angeboten, die in der von ihm erworbenen Lizenz enthalten sind. Spätere Produktversionen existieren evtl. bereits, stehen ihm aber nicht zur Verfügung. Dieses Vorgehen entspricht üblicherweise auch der Strategie in der Automobilbranche.

2.8.3 Vergleich und Diskussion

Obwohl Automobil- und PC-Branche in dieselbe Rekonfigurationsklasse eingeordnet werden können, müssen beim Rekonfigurieren im Automobilbereich besondere Randbedingungen berücksichtigt werden. Eine Rekonfigurationsstrategie im Automobilbereich muss aus den folgenden Gründen erheblich differenzierter ausgelegt werden:

[6] Durch spezielle Installationsroutinen im Fall von Microsoft Windows Automatische Updates oder Betriebssystemfunktionen im Fall von Mozilla Firefox kann der Anwender dennoch eine Zielsoftware auswählen und installieren.

1. **Ressourcen:** Ressourcen wie Speicher und Rechenleistung in einem Fahrzeugsteuergerät sind stark begrenzt, so dass kaum neuere, ressourcenintensivere Software im gleichen Steuergerät eingesetzt werden kann.

2. **Abstraktion:** Durch die sehr spezifische Entwicklung von Steuergeräten für einen Einsatzzweck, z.B. bezüglich Ein- und Ausgängen, ist die Einsatzfähigkeit neuerer Software auf älteren Steuergeräten oft nicht mehr möglich.

3. **Flashzeit:** Wegen des vergleichsweise langsamen Datenkanals zwischen Service-Tester und Fahrzeug bzw. Steuergerät entstehen beim Flashen vieler Softwarekomponenten hohe Werkstattkosten.

4. **Teiletausch:** Müssen aufgrund eines Softwareupdates zusätzliche Teile oder das Steuergerät im Fahrzeug getauscht werden, entstehen ebenfalls hohe Werkstattkosten.

In der PC-Branche stehen zusätzliche Ressourcen für neuere Softwarekomponenten in der Regel in hohem Maße zur Verfügung. Ist dennoch eine Aufrüstung der Ressourcen notwendig, ist diese meist günstiger und der Anwender ist eher dazu bereit, die Kosten dafür zu tragen als in der Automobilbranche, obwohl durch die Aufrüstung nicht zwangsläufig die Funktionalität der Software erweitert wird, sondern Fehler behoben werden. Diese Bereitschaft resultiert auch aus der Wahrnehmung des Anwenders, der sich selbst verantwortlich für die Wartung seines PCs fühlt. Ein Automobil wird aus verschiedenen Gründen aber als Gesamtsystem eines Herstellers betrachtet, das zu 100% zuverlässig funktionieren muss und dessen Wartung Aufgabe des Herstellers ist. Grundlage dafür ist zudem die höhere Kompetenz des Anwenders innerhalb der PC-Domäne. Bezüglich der Dauer eines Softwareupdates kann durch die häufige Verbindung eines PCs mit dem Internet ein regelmäßiges und gleichzeitig schnelles Update zum Teil unbemerkt durchgeführt werden, wodurch lange und teure Warte- oder Werkstattzeiten vermieden werden. Die Zeit, die ein Anwender bereit ist, vor dem PC zu verbringen, ist wesentlich höher, als die Zeit, die er sich in der Werkstatt aufhalten würde.

3 Domänenmodell zum Rekonfigurieren mit mehreren Operationsrealisierern

In diesem Kapitel wird das Problem des Rekonfigurierens im Automobil-Service in ein Domänenmodell übertragen. Das Modell beschreibt die Interaktionen zwischen einem Kunden und einem Unternehmen beim Rekonfigurieren eines Produkts, das aus einer Menge von Produktmerkmalen besteht. Die Produktmerkmale werden dabei durch Rekonfigurationsaufträge verändert, die durch Rekonfigurationsoperationen umgesetzt werden. Zum Beispiel kann ein Kunde einer Werkstatt den Auftrag geben, sein Fahrzeug zu reparieren, da er ein Schaltrucken zwischen dem ersten und zweiten Gang festgestellt hat. Um die Reparatur durchzuführen müssen die fehlerverursachenden Produktmerkmale verändert werden. Das Update einer Steuergerätesoftware wäre eine solche Aktion. Hierbei müssen alle notwendigen Produktmerkmale entsprechend angepasst werden.

3.1 Produkt

Ein `Produkt`[1] ist eine Konfiguration, die den Auftrag eines Kunden (z.B. `RekonfigurationsAuftrag`) erfüllt. Es besitzt ein oder mehrere `ProduktMerkmale`, durch die es beschrieben wird (siehe Abb. 3.1). Bereits durch die Nennung einiger Produktmerkmale ist die Identifikation eines Produkts bzw. die Unterscheidung eines Produkts von einem anderen möglich. Zwei Produkte sind dann unterschiedlich, wenn sie sich in mindestens einem Produktmerkmal unterscheiden, das relevant für den Kundenauftrag ist.

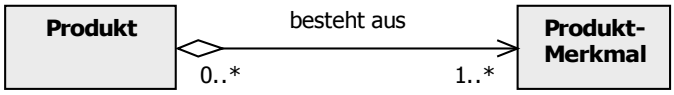


Abbildung 3.1: UML-Ausschnitt: Beziehung zwischen einem Produkt und seinen Produktmerkmalen

In der Realität besteht ein Produkt aus physischen Komponenten, Datenkomponenten, Dokumenten oder Dienstleistungen, hat bestimmte Eigenschaften (z.B. Abmessungen in Länge und Breite) und realisiert kundenerlebbare Funktionen (z.B. Höchstgeschwindigkeit = 232 km/h). Sämtliche dieser Komponenten werden einheitlich als Produkt-

[1] Alle Bestandteile des Domänenmodells werden im Text einmalig durch eine andere Schriftart hervorgehoben.

merkmale aufgefasst. Somit ist das Modell unabhängig von den Komponententypen und Eigenschaften, die rekonfiguriert werden sollen und beschreibt explizit die zur Rekonfiguration nötigen Schritte.

3.2 Produktmerkmale

Ein Produkt wird demnach als Menge ungeordneter und unstrukturierter Produktmerkmale aufgefasst:

$$\mathbf{P_x} = \{pm_1, \ldots, pm_n\}$$

mit n = Anzahl aller Produktmerkmale des Produkts $\mathbf{P_x}$. Die Variablen $pm_1, \ldots, pm_n$ dienen hier als Bezeichner der Produktmerkmale. Weiter unten werden noch detailliertere Aussagen zur Identität zweier Produktmerkmale getroffen, da ein Produkt durchaus Produktmerkmale besitzen kann wie z.B. pm_2 = „hat Motor“ und pm_{32} = „hat Antriebsmaschine“. Das ist nicht sinnvoll, aber möglich, da unterschiedliche Beteiligte des Rekonfigurationsprozesses dasselbe Produktmerkmal unterschiedlich formulieren können. Produktmerkmale können ebenso überbestimmt sein, d.h. ein Produktmerkmal ist ggf. mehrfach in der Beschreibung weiterer Produktmerkmale enthalten. Die Beziehungen zwischen den verschiedenen Produktmerkmalen (Produktstruktur) stecken implizit in deren Beschreibung und werden in diesem Modell nicht explizit formuliert.

In der Beschreibung eines Produkts müssen nicht alle, sondern nur die für die Rekonfiguration relevanten Produktmerkmale aufgeführt werden. Es muss aber bekannt sein, welche Produktmerkmale Ziel einer Rekonfiguration sein können, da entsprechende Folgemaßnahmen bei einer Rekonfiguration notwendig werden können.

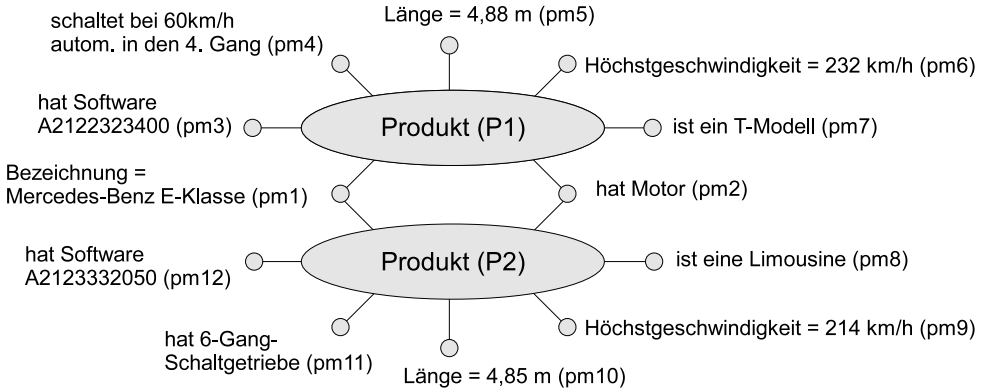


Abbildung 3.2: Zwei Beispielprodukte mit Produktmerkmalen

Im Beispiel in Abb. 3.2 sind zwei unterschiedliche Produkte mit einem Teil ihrer Pro-

duktmerkmale dargestellt. Im Folgenden sollen Produkte wie folgt beschrieben werden:

$$\begin{aligned} \mathbf{P_1} &= \{pm_1, pm_2, pm_3, \ldots, pm_7\} \\ \mathbf{P_2} &= \{pm_1, pm_2, pm_8, \ldots, pm_{12}\} \end{aligned}$$

Außer in zwei Merkmalen sind die Produkte nach dieser Beschreibung vollständig verschieden. Zur Unterscheidung der beiden Produkte ist im Beispiel z.B. nur die Nennung des Produktmerkmals „T-Modell“ (pm7) oder „Limousine“ (pm8) notwendig.

Abstrakte und konkrete Produktmerkmale

Ohne die Struktur zwischen Produktmerkmalen explizit zu kennen, kann man sie in abstrakte und konkrete Merkmale unterteilen, wobei zur Rekonfiguration abstrakter Produktmerkmale immer weniger abstrakte oder konkrete Produktmerkmale rekonfiguriert werden müssen. Abstrakte Produktmerkmale im Beispiel in Abb. 3.2 sind z.B. „Länge = 4,88 m“ oder „ist eine Limousine“, da zur Rekonfiguration dieser Merkmale eine Menge weiterer Produktmerkmale angepasst werden muss. Generell sind vor allem kundenerlebbare Funktionen abstrakte Produktmerkmale. Komponentenbezogene Produktmerkmale wie z.B. „hat Software A2123332050“ sind in diesem Fall konkrete Produktmerkmale.

3.3 Kunde

Ein Produkt ist immer in Besitz eines Kunden. Jeder Kunde hat Zugriff auf die Produkte, die er besitzt und ist die Person, die eine Rekonfiguration ihres Produkts veranlasst.

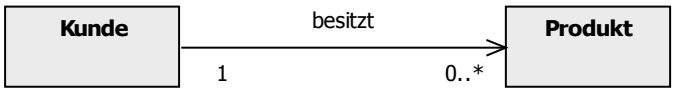


Abbildung 3.3: UML-Ausschnitt: Beziehung zwischen Kunde und Produkt

Da ein Kunde nur begrenztes Wissen über das Produkt und dessen Produktmerkmale besitzt, ist er auf Unterstützung z.B. durch die Werkstätten eines Unternehmens angewiesen (siehe Abschnitt 3.6). Verschiedene Kunden, die das gleiche Produkt besitzen und dasselbe Rekonfigurationsziel (siehe Abschnitt 3.5) verfolgen, können unterschiedliche Ansprüche an die Durchführung und das Ergebnis der Rekonfiguration haben. Nicht alle Ansprüche, die ein Kunde an das Resultat eines Rekonfigurationsvorgangs erhebt, werden vorher explizit von ihm formuliert. Die diversen, teilweise nicht explizit benannten aber dennoch möglichen Kundenansprüche müssen in der Reparaturstrategie des Unternehmens berücksichtigt werden.

3.4 Einführung Rekonfigurieren

Rekonfigurieren ist eine Maßnahme zur Anpassung eines bestehenden Produkts unter Berücksichtigung eines vorgegebenen `RekonfigurationsZiel`s, wodurch ein vom Ausgangsprodukt verschiedenes Produkt entsteht. Die konkrete Umsetzung des Rekonfigurationsziels wird in einem Rekonfigurationsauftrag vereinbart. Damit ist das Rekonfigurieren, unter Einhaltung eines konkreten Rekonfigurationsauftrags, eine Funktion f zur Überführung eines Produkts in ein anderes Produkt:

$$f : \mathbf{P} \mapsto \mathbf{P}$$

mit $\mathbf{P}$ als Menge aller möglichen Produkte. In der Regel wird ein Rekonfigurationsauftrag zwischen einem Kunden und dem Unternehmen abgeschlossen, dessen Produkt der Kunde erworben hat. Neben diesem Unternehmen kann es weitere Unternehmen geben, die Rekonfigurationen für bestimmte Produkte anbieten oder Kunden, die ein Produkt ohne Unterstützung durch ein Unternehmen rekonfigurieren. Außer dem Kunden verfolgen weitere Parteien Rekonfigurationsziele an dem Produkt, so z.B. das verkaufende Unternehmen, das ausschließlich (auch nach der Rekonfiguration) „funktionierende“ Produkte anbieten möchte oder gesetzliche Vorgaben, die z.B. Einschränkungen bei der Rekonfiguration sicherheitskritischer Produktmerkmale definieren. Die Möglichkeit zur Unterstützung von Kunden bei der Rekonfiguration sowie das Verfolgen von Zielen mehrerer Beteiligter wird im Abschnitt 3.6 durch das Konzept der Operationsrealisierung beschrieben.

3.5 Rekonfigurationsziele

Ein Rekonfigurationsziel besteht aus einem Produkt, einer `RekonfigurationsOperation` und einer `RekonfigurationsInvariante`:

$$rz = (\mathbf{P}, ro, ri)$$

Während die Rekonfigurationsoperation ro beschreibt, welche Änderungen an dem Produkt vorgenommen werden sollen, wird durch die Rekonfigurationsinvariante ri beschrieben, welche bestehenden Produktmerkmale des Produkts nicht verändert werden dürfen. Letztlich stellt jedes Rekonfigurationsziel eine Regel dar, deren Prämisse aus dem zu rekonfigurierenden Produkt und deren Aktionsteil aus der Rekonfigurationsoperation besteht.
Soll dieselbe Rekonfigurationsoperation auf mehrere, verschiedene Produkte angewendet werden, muss die Menge dieser Produkte beschrieben werden. Die Beschreibung kann durch bereits vorhandene, gemeinsame Produktmerkmale vorgenommen werden, die diese Produktmenge eindeutig kennzeichnen (z.B. „hat KLA-SW 2“). Sie kann ebenso durch die Zusammenfassung von bereits vorhandenen Produktmerkmalen (z.B. Fahrzeugidentifikationsnummer 1 bis 9999 oder („hat SAM-SW 3“ oder „hat SAM-SW 4“)) geschehen.

3.5.1 Rekonfigurationsoperationen

Durch Rekonfigurationsoperationen wird ein Produkt bzw. seine Produktmerkmale verändert. Da Produkte als Menge unstrukturierter Produktmerkmale definiert sind, werden sämtliche Rekonfigurationsoperationen durch das Einfügen und Entfernen von Produktmerkmalen in oder aus diese(r) Menge realisiert. Damit ist eine Rekonfigurationsoperation eine Struktur, die aus einer Menge einzufügender und einer Menge zu entfernender Produktmerkmale besteht. Die dadurch erreichte Veränderung von Produktmerkmalen bezieht sich immer auf das im Rekonfigurationsziel angegebene Produkt.
Die folgende Struktur

$$ro = (\mathbf{PM}^+, \mathbf{PM}^-)$$

zeigt eine Rekonfigurationsoperation ro, die einem im Rekonfigurationsziel angegebenen Produkt die Produktmerkmale der Menge $\mathbf{PM}^+$ hinzufügen und die Produktmerkmale der Menge $\mathbf{PM}^-$ entfernen soll. Die Ausführung einer Rekonfigurationsoperation auf einem Produkt $\mathbf{P_x}$ ist damit wie folgt definiert:

$$\mathbf{P_y} = (\mathbf{P_x} \cup ro.\mathbf{PM}^+) \setminus ro.\mathbf{PM}^-$$

mit . als Zugriffsoperator auf die Mengen der Rekonfigurationsoperation ro. Hierbei wird von einer bereits vervollständigten und konkretisierten Rekonfigurationsoperation ausgegangen (siehe Abschnitt 3.7).
Eine Vereinigung mehrerer Rekonfigurationsoperationen kann dann stattfinden, wenn sich alle zu vereinigenden Rekonfigurationsoperationen auf das gleiche Produkt beziehen. Dann müssen alle Mengen der einzufügenden Produktmerkmale und alle Mengen der zu entfernenden Produktmerkmale vereinigt werden:

$$ro_x = ((ro_1.\mathbf{PM}^+ \cup \ldots \cup ro_n.\mathbf{PM}^+), (ro_1.\mathbf{PM}^- \cup \ldots \cup ro_n.\mathbf{PM}^-))$$

mit n = Anzahl der zu vereinigenden Rekonfigurationsoperationen.

Ungültige Rekonfigurationsoperationen

Da das Hinzufügen und Entfernen von Produktmerkmalen gleichberechtigt sind, ist eine Rekonfigurationsoperation ro dann ungültig, wenn es ein oder mehrere Produktmerkmale gibt, die sowohl in der Menge der einzufügenden als auch in der Menge der zu entfernenden Produktmerkmale vorkommen, also wenn gilt:

$$ro.\mathbf{PM}^+ \cap ro.\mathbf{PM}^- \neq \{\}$$

Hierbei wird davon ausgegangen, dass die Identität zweier Produktmerkmale nur dann gegeben ist, wenn sie dieselbe Bezeichnung haben, wie z.B. „pm1“ und „pm1“. Eine Rekonfigurationsoperation $ro = (\{pm2\}, \{pm32\})$ mit pm_2 = „hat Motor“ sowie pm_{32} = „hat Antriebsmaschine“ wäre nach diesem Test gültig. Da (neben der Identitätsfunktion) auch keine direkte Negation eines Produktmerkmals definiert ist, kann keine Regel aufgestellt

werden, die eine Rekonfigurationsoperation ungültig macht, wenn ein Produktmerkmal und dessen Negation in $\mathbf{PM}^+$ oder in $\mathbf{PM}^-$ vorkommen. Beide Widersprüche innerhalb einer Rekonfigurationsoperation können also ausschließlich über das Rekonfigurationswissen bei der Realisierung der Rekonfigurationsoperation aufgedeckt werden.
Da wir bei allen Mengen von einer Zusammenfassung wohlunterscheidbarer Objekte ausgehen, kann dasselbe Produktmerkmal nie mehrmals in $\mathbf{PM}^+$ oder $\mathbf{PM}^-$ vorkommen. Ein erneutes Hinzufügen eines Produktmerkmals zur Menge $\mathbf{PM}^+$ oder $\mathbf{PM}^-$ würde zu keiner Änderung der entsprechenden Rekonfigurationsoperation führen.

3.5.2 Rekonfigurationsinvarianten

Neben der Beschreibung der gewünschten Veränderungen eines Produkts müssen zusätzlich alle Produktmerkmale erfasst werden, die nicht verändert werden dürfen. Grund für diese Angaben ist die Vervollständigung und Konkretisierung eines Rekonfigurationsziels bei seiner Realisierung. Die ursprünglich im Rekonfigurationsziel angegebenen, zu verändernden Produktmerkmale werden dadurch erweitert oder eingeschränkt. Eine Begrenzung dieser Erweiterungen und Einschränkungen kann durch die Angabe einer Rekonfigurationsinvariante realisiert werden. Sie ist eine Struktur, die aus einer Menge von Produktmerkmalen besteht, die im Rahmen der Realisierung der Rekonfigurationsoperation nicht eingefügt werden dürfen und einer weiteren Menge, die nicht entfernt werden dürfen.
Die folgende Struktur

$$ri = (\mathbf{PM^{n+}}, \mathbf{PM^{n-}})$$

zeigt eine Rekonfigurationsinvariante ri, die verhindert, dass einem Produkt die Produktmerkmale der Menge $\mathbf{PM^{n+}}$ hinzugefügt und die Produktmerkmale der Menge $\mathbf{PM^{n-}}$ entfernt werden. Ein Rekonfigurationsziel rz ist damit ungültig, sobald gilt:

$$\begin{aligned} rz.ro.\mathbf{PM}^+ \cap rz.ri.\mathbf{PM^{n+}} &\neq \{\} \qquad \text{oder} \\ rz.ro.\mathbf{PM}^- \cap rz.ri.\mathbf{PM^{n-}} &\neq \{\} \end{aligned}$$

Auch hierbei gelten die Aussagen zur Identität und zur Negation von Produktmerkmalen wie bei den Rekonfigurationsoperationen. Die Vereinigung zweier Rekonfigurationsinvarianten erfolgt analog zur Vereinigung von Rekonfigurationsoperationen:

$$ri_x \cup ri_y = (ri_x.\mathbf{PM^{n+}} \cup ri_y.\mathbf{PM^{n+}}, ri_x.\mathbf{PM^{n-}} \cup ri_y.\mathbf{PM^{n-}})$$

Beziehen sich zwei Rekonfigurationsziele auf dasselbe Produkt, können sie vereinigt werden, indem ihre Rekonfigurationsoperationen und -invarianten vereinigt werden.
Alle bisher beschriebenen Zusammenhänge beim Rekonfigurieren werden in Abb. 3.4 als UML-Klassendiagramm dargestellt. Die Umsetzung von Rekonfigurationszielen durch einen `OperationsRealisierer` wird in den folgenden Abschnitten vorgestellt.

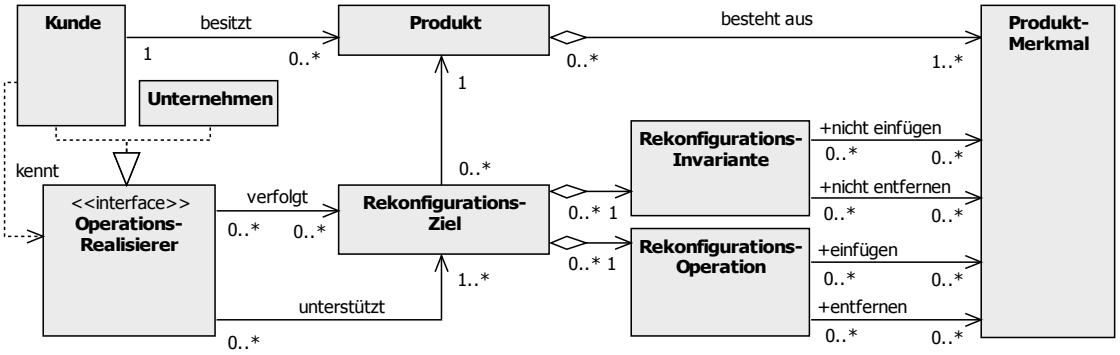


Abbildung 3.4: UML-Modell mit den wesentlichen Bestandteilen des Domänenmodells

3.6 Operationsrealisierer

Da der Kunde in der Regel nur begrenzte Fähigkeiten und kein vollständiges Wissen über die Merkmale und die Struktur des zu rekonfigurierenden Produkts hat, benötigt er in vielen Fällen Unterstützung. Weiterhin haben neben dem Kunden andere Beteiligte Einfluss auf die Rekonfiguration des Produkts, so dass der Rekonfigurationsprozess eine Menge von Rekonfigurationszielen aller Beteiligten berücksichtigen muss. Eine Entität, die einem Kunden Rekonfigurationsziele anbietet und selbst Rekonfigurationsziele verfolgt, ist ein Operationsrealisierer. Ein Kunde selbst ist ebenfalls, für die in seinem Kompetenzbereich liegenden Rekonfigurationsoperationen, ein Operationsrealisierer. Teilweise müssen Operationsrealisierer ihre Rekonfigurationsziele auch in Vertretung anderer definieren, insbesondere im Fall von Service-Unternehmen, die für die Einhaltung von Gesetzesvorgaben beim Rekonfigurieren sorgen. Für die Umsetzung einer Rekonfigurationsoperation kann ein Operationsrealisierer dem beauftragenden Kunden Kosten berechnen. Diese Kosten können beispielsweise aus Kosten für die Beschaffung von Ersatzteilen und das bereitgestellte Rekonfigurationswissen bestehen.

Um eine Rekonfiguration zu beauftragen, stellt ein Kunde zunächst eine `RekonfigurationsAnfrage` an einen Operationsrealisierer. Der Operationsrealisierer erstellt auf Grundlage dieser Anfrage ein `RekonfigurationsKonzept`, das dem Kunden zurückgemeldet wird. Das Rekonfigurationskonzept enthält alle durchzuführenden Rekonfigurationsoperationen und die entstehenden Kosten. Da Operationsrealisierer Produktmerkmale oft anders formulieren als der Kunde, ist der Operationsrealisierer ebenfalls für die Erläuterung des Rekonfigurationskonzepts gegenüber dem Kunden zuständig. Es folgt eine Bewertung des erstellten Rekonfigurationskonzepts durch den Kunden, woraufhin es entweder akzeptiert oder abgelehnt werden kann. Im Fall einer Ablehnung kann erneut ein (nachgebessertes) Rekonfigurationskonzept durch den Operationsrealisierer vorgeschlagen werden, das dann analog wie im ersten Schritt bewertet wird und angenommen oder abgelehnt werden kann. Während dieser Evaluationsschleife muss der Kunde seine Rekonfigurationsziele gegebenenfalls neu definieren, um das

tatsächlich gewünschte Produkt zu erhalten (siehe Abb. 3.5). Es ist hierbei insbesondere

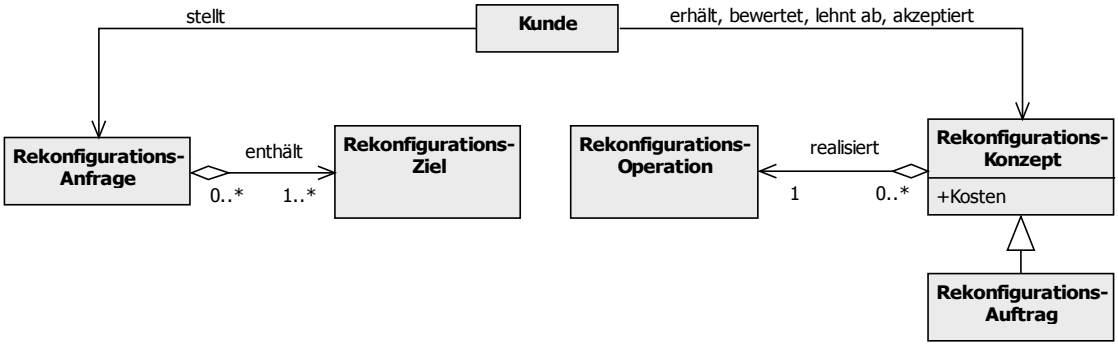


Abbildung 3.5: UML-Diagramm mit Rekonfigurationsanfrage und -konzept

im Sinne des Kunden, dass die Erstellung des Rekonfigurationskonzepts keine oder wenig Kosten verursacht, d.h., dass ein möglichst vollständiges Rekonfigurationskonzept entsteht, bevor tatsächlich Änderungen am Produkt vorgenommen werden. Hierzu ist umfangreiches Rekonfigurationswissen nötig (siehe Abschnitt 3.6.1).
Wird ein Rekonfigurationskonzept akzeptiert, gilt der Rekonfigurationsauftrag als vereinbart und die im Rekonfigurationskonzept enthaltenen Rekonfigurationsoperationen werden ausgeführt. Der Kunde erhält anschließend sein rekonfiguriertes Produkt zurück.

3.6.1 Wissens- und Änderungsgranularität

Im folgenden sollen drei Messgrößen eingeführt werden, die angeben, wie präzise ein Rekonfigurationskonzept dem Kunden gegenüber formuliert werden kann (*Dokumentationsgrad*) und wie feingranular Änderungen am Produkt durch einen Operationsrealisierer vorgenommen werden können (*Kompositions-* und *Aggregationsgrad*).

Dokumentationsgrad

Der Dokumentationsgrad beschreibt die Übereinstimmung zwischen den dokumentierten Produktmerkmalen und den Produktmerkmalen, die bei der entsprechenden Rekonfigurationsoperation tatsächlich verändert werden. Er sollte möglichst hoch sein. Andernfalls wird der Kunde das Ergebnis der Rekonfiguration erst nach Erhalt des rekonfigurierten Produkts evaluieren können und gegebenenfalls weitere Veränderungen von Produktmerkmalen entdecken, die nicht seinem ursprünglichen Rekonfigurationsziel entsprechen. Bei einem geringen Dokumentationsgrad ist der Informationsgehalt eines Rekonfigurationskonzepts demnach niedrig, bei großem Dokumentationsgrad ist er hoch.

Kompositionsgrad

Der Kompositionsgrad eines Produktmerkmals beschreibt, wie viele Produktmerkmale sich zwangsläufig zusätzlich verändern werden, wenn dieses Produktmerkmal einem

Produkt hinzugefügt oder daraus entfernt wird. Ist der Kompositionsgrad > 0, dann existiert kein Operationsrealisierer, der eine Rekonfigurationsoperation zum Hinzufügen oder Entfernen der einzelnen Produktmerkmale anbietet, die Bestandteil der Komposition sind. Zum Beispiel ist der Kompositionsgrad einer Steuergerätesoftware > 0, wenn beim Hinzufügen dieser Software ein Fehler behoben wird, diese aber gleichzeitig auch eine funktionale Anpassung enthält. Ein Beheben des Fehlers ohne die funktionelle Anpassung ist nicht möglich.
Der Kompositionsgrad ist abhängig vom anzupassenden Produkt, da durch das Verändern desselben Produktmerkmals in unterschiedlichen Ausgangsprodukten eine unterschiedliche Menge zusätzlicher Änderungen stattfinden kann.

Aggregationsgrad

Der Aggregationsgrad eines Produktmerkmals beschreibt ebenso, wie viele Produktmerkmale sich zwangsläufig zusätzlich verändern werden, wenn dieses Produktmerkmal einem Produkt hinzugefügt oder daraus entfernt wird. Im Gegensatz zum Kompositionsgrad wurden die zusätzlichen Änderungen hierbei aber bewusst festgelegt, um mehrere Produktmerkmale gebündelt zu verändern. Eine einzelne Änderung der Produktmerkmale wäre ebenso möglich.

Sowohl bei einem hohen Kompositions- als auch Aggregationsgrad hat der Kunde wenige Möglichkeiten zur Rekonfiguration, da Produktmerkmale nur stark gebündelt verändert werden können. In diesem Fall können die wenigen möglichen Produkte umfangreicher durch das Unternehmen getestet werden, wobei der Kunde Möglichkeiten zur Individualisierung seines Produkts verliert.

3.6.2 Kundenerwartungen

Offensichtlich formulieren Kunden aufgrund ihres begrenzten Produktwissens und dem fehlenden Überblick über die tatsächliche Realisierung durch den beauftragten Operationsrealisierer initial nur einen Teil ihres Rekonfigurationsziels. Eine vollständige Formulierung eines Rekonfigurationsziels durch den Kunden ist auf Grund der theoretisch unendlichen Möglichkeiten der Operationsrealisierer kaum möglich. Zusätzlich zu diesem explizit formulierten Teil des Rekonfigurationsziels gibt es daher einen impliziten Teil, der erst durch die Erstellung gegebenenfalls mehrerer Rekonfigurationskonzepte innerhalb der Evaluationsschleife herausgefunden werden muss. Nachdem der Kunde nun zusätzliches Wissen über die Realisierung seines Rekonfigurationsziels erhalten hat, kann er bei Bedarf Rekonfigurationsinvarianten formulieren, um die Realisierung zusätzlich zu steuern und Veränderungen bestimmter Produktmerkmale zu verhindern.
Seitens des Unternehmens bzw. aller Rekonfigurationsteilnehmer, die die Rekonfigurationsziele des Kunden umsetzen, muss es daher das Ziel sein, die Erwartungen des Kunden möglichst nicht zu verletzen, wobei der Kunde diese Erwartungen hierzu nicht explizit

angeben muss. Das Fehlen der expliziten Angabe von Invarianten zwingt das Unternehmen dazu, diese für die Gesamtmenge der Kunden zwar pauschal, aber mit möglichst hoher Genauigkeit im Mittel zu definieren. Hier sind die Kosten zur Schaffung einer Methode zur expliziten Angabe von Kundenerwartungen abzuwägen gegen die Kosten für die Pauschalisierung und damit nicht optimale Erfüllung von Kundenerwartungen.

3.6.3 Verkettete Operationsrealisierer

Operationsrealisierer können auch verkettet arbeiten, wobei jeder Operationsrealisierer eine Teilmenge des benötigten Rekonfigurationswissens besitzt (siehe Abb. 3.6). Jeder Operationsrealisierer trägt dabei die volle Verantwortung für die von ihm umgesetzten Rekonfigurationsziele. Ist der Kunde selbst der Operationsrealisierer, gilt diese Verantwortung ebenso für ihn. Für manche Rekonfigurationsziele, insbesondere sicherheitskritische Rekonfigurationen, ist der Kunde verpflichtet, auf die ihm zur Verfügung gestellten Operationsrealisierer zurückzugreifen. Der direkte Eingriff von Kunden auf bestimmte Produktmerkmale muss daher durch Sicherheitsmechanismen wie z.B. Authentifizierungsverfahren eingeschränkt werden, so dass für diesen Vorgang nur authorisierte Operationsrealisierer eingesetzt werden.

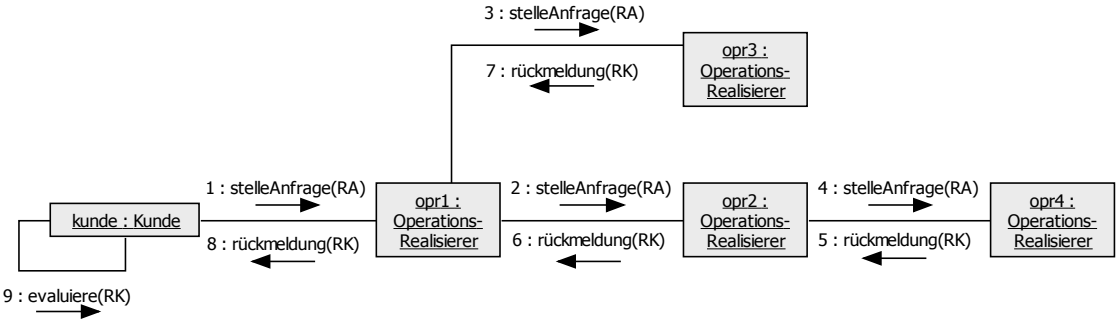


Abbildung 3.6: Erstellung eines Rekonfigurationskonzepts durch mehrere, verkettete Operationsrealisierer

Abhängig davon, wie viele Rekonfigurationsziele ein Operationsrealisierer unterstützt, hat der Kunde sehr zahlreiche oder nur wenige Möglichkeiten zur Rekonfiguration seines Produkts.

3.7 Operationsrealisierung

Eine Rekonfigurationsanfrage wird von einem Operationsrealisierer durch

- Konkretisierung,
- Vervollständigung und

- Konfliktauflösung

der in der Anfrage enthaltenen Rekonfigurationsziele schrittweise bearbeitet. Dabei wird die Rekonfigurationsanfrage durch die Funktion g auf ein Rekonfigurationskonzept abgebildet:

$$g : \mathbf{RA} \mapsto \mathbf{RK}$$

mit **RA** als Menge aller möglichen Rekonfigurationsanfragen und **RK** als Menge aller Rekonfigurationskonzepte. Ist die Funktion nicht eindeutig definiert, kann **RA** auch auf eine Menge von Rekonfigurationskonzepten abgebildet werden.
Teil dieser Funktion ist die Abbildung einer Rekonfigurationsoperation auf eine andere Rekonfigurationsoperation:

$$h : \mathbf{RO} \mapsto \mathbf{RO}$$

mit **RO** als Menge aller möglichen Rekonfigurationsoperationen. Hierbei müssen die im Rekonfigurationsziel angegebenen Rekonfigurationsinvarianten eingehalten werden.

3.7.1 Konkretisierung von Rekonfigurationszielen

Bei der Konkretisierung eines Rekonfigurationsziels wird die dazugehörige, abstrakte Rekonfigurationsoperation durch eine oder mehrere andere, konkretere Rekonfigurationsoperationen ersetzt:

$$h(ro_x) = h(ro_y) \cup h(ro_z)$$

wobei die Rekonfigurationsoperation ro_x durch die beiden Rekonfigurationsoperationen ro_y und ro_z ersetzt, also konkretisiert wird. Zuvor werden ro_y und ro_z ebenfalls durch h konkretisiert und vervollständigt.

3.7.2 Vervollständigung von Rekonfigurationszielen

Bei der Vervollständigung eines Rekonfigurationsziels wird die dazugehörige Rekonfigurationsoperation mit weiteren Rekonfigurationsoperationen vereinigt:

$$h(ro_x) = ro_x \cup h(ro_y)$$

wobei die Rekonfigurationsoperation ro_x um die Rekonfigurationsoperation ro_y vervollständigt wird. Zuvor wird ro_y ebenfalls durch h konkretisiert und vervollständigt.
Sowohl das Konkretisieren als auch Vervollständigen von Rekonfigurationsoperationen haben in den obigen Beispielen mehrere neue Rekonfigurationsoperationen zum Ergebnis, die durch den Operationsrealisierer realisiert werden. Sollen die entstandenen Rekonfigurationsoperationen vollständig durch diesen Operationsrealisierer umgesetzt werden, ist eine Vereinigung der entstandenen Rekonfigurationsoperationen möglich, da diese sich alle auf dasselbe Produkt beziehen. Eine Aufteilung auf mehrere Rekonfigurationsoperationen ist notwendig, wenn weitere Operationsrealisierer mit der Umsetzung von Teilen

der Rekonfigurationsoperation beauftragt werden sollen. Die Zerlegung in Einzeloperationen erfolgt dann entsprechend der Aufteilung auf die Operationsrealisierer (wie z.B. ro_y und ro_z weiter oben, die nun durch einen jeweils anderen Operationsrealisierer umgesetzt werden können).
Operationsrealisierer bieten unterschiedlich formulierte Rekonfigurationsziele an, die aber inhaltlich identisch sind, um das fehlende Produktwissen des beauftragenden Kunden zu kompensieren und deren formulierte Rekonfigurationsoperationen in Standard-Produktmerkmale zu übersetzen.

3.7.3 Berücksichtigung von Rekonfigurationsinvarianten

Bei der Ausführung von h zur Konkretisierung und Vervollständigung einer Rekonfigurationsoperation müssen sämtliche Rekonfigurationsinvarianten, die in der Rekonfigurationsanfrage formuliert wurden, berücksichtigt werden. Bestimmte Möglichkeiten zur Realisierung einer Rekonfigurationsoperation werden dadurch ausgeschlossen und es muss eine der Realisierungsmöglichkeiten durch den Operationsrealisierer gefunden werden, die die vorgegebenen Rekonfigurationsinvarianten nicht verletzen. Existiert keine solche Realisierungsmöglichkeit, kann der Operationsrealisierer die Rekonfigurationsanfrage nicht bearbeiten und gibt dem beauftragenden Kunden oder Operationsrealisierer eine entsprechende Rückmeldung mit der nicht umsetzbaren Rekonfigurationsoperation und der verletzten Rekonfigurationsinvariante.
Beauftragt der Operationsrealisierer weitere Realisierer, muss er bei der Formulierung der Rekonfigurationsanfragen die Rekonfigurationsinvarianten entsprechend befüllen. Es werden hierbei alle Rekonfigurationsinvarianten übernommen, die bereits in der ursprünglichen Rekonfigurationsanfrage vorgegeben wurden und weitere Invarianten ergänzt, die den Rekonfigurationszielen des Operationsrealisierers entsprechen. Dabei ist zu beachten, dass keine abstrakten Produktmerkmale als Invariante formuliert werden, die durch den beauftragten Operationsrealisierer konkretisiert und dadurch entfernt werden müssen. In der Regel beziehen sich die Rekonfigurationsinvarianten also ausschließlich auf konkrete Produktmerkmale.

3.7.4 Konfliktauflösung

Bei der Realisierung von Rekonfigurationsoperationen darf der Operationsrealisierer nur gültige Rekonfigurationskonzepte an seinen Kunden zurückmelden und bei der Beauftragung weiterer Operationsrealisierer nur gültige Rekonfigurationsanfragen formulieren.
In der gesamten Kette von Operationsrealisierern, die eine Rekonfigurationsanfrage bearbeiten, muss es demnach mindestens einen Realisierer geben, der die inhaltliche Identität und den inhaltlichen Widerspruch zweier Produktmerkmale erkennt. Für das obige Beispiel mit pm_2 = „hat Motor“ und pm_{32} = „hat Antriebsmaschine“ und $ro = (\{pm_2\}, \{pm_{32}\})$ würde spätestens der Operationsrealisierer, der ro ausführt einen Widerspruch feststellen, da nach seinem Wissen pm_2 und pm_{32} äquivalent bezüglich des Hin-

zufügens und Entfernens sind. Ist stattdessen $ro = (\{pm_2, pm_{32}\}, \{\})$, würde der Operationsrealisierer diese Überbestimmung durch ein Löschen eines der Produktmerkmale aus $\mathbf{PM}^+$ oder eine Überführung beider Produktmerkmale in ein Standard-Produktmerkmal (z.B. pm_{90} = „hat A2125558090“) auflösen. Der Operationsrealisierer führt die Erkennung inhaltlich identischer Produktmerkmale innerhalb der Rekonfigurationsoperationen, innerhalb der Rekonfigurationsinvarianten und zwischen Rekonfigurationsoperationen und -invarianten durch, um Überbestimmungen und Widersprüche aufzulösen.
Erhält ein Operationsrealisierer oder ein Kunde ein Rekonfigurationskonzept von den von ihm beauftragten Operationsrealisierer zurück, müssen diese in einem Rekonfigurationskonzept zusammengeführt und dieses auf Gültigkeit und Überbestimmung geprüft werden. Da unabhängig voneinander beauftragte Operationsrealisierer nichts über die Rekonfigurationsoperationen der jeweils anderen Realisierer wissen, können rückgemeldete Rekonfigurationskonzepte eine Rekonfigurationsoperation mehrfach enthalten. Da eine Rekonfigurationsoperation jedoch nur einmalig ausgeführt werden muss, löst der Operationsrealisierer diese Überbestimmung in seinem Rekonfigurationskonzept auf und vermindert die Kosten entsprechend. Dazu muss ein Operationsrealisierer eine Aussage zu den von ihm veranschlagten Kosten für eine spezielle Rekonfigurationsoperation treffen können. Bei der späteren Umsetzung des Rekonfigurationskonzepts wird davon ausgegangen, dass die betroffenen Operationsrealisierer eine Rekonfigurationsoperation nicht erneut ausführen, wenn sie bereits von einem anderen Realisierer ausgeführt wurde. Wird die eigene Umsetzung einer Rekonfigurationsoperation durch die vorherige Arbeit eines anderen Operationsrealisierers beeinflusst, muss dies in der Aussage über die eingesparten Kosten berücksichtigt werden, wobei die Reihenfolge, in der die unterschiedlichen Operationsrealisierer beauftragt werden, festgehalten werden muss.
Entsteht beim Zusammenführen der rückgemeldeten Rekonfigurationskonzepte ein Rekonfigurationskonzept, das eine ungültige Rekonfigurationsoperation enthält, kann der Operationsrealisierer eine neue Rekonfigurationsanfrage mit einer angepassten Rekonfigurationsinvariante stellen. Dabei wird bei einem beauftragten Realisierer durch die Rekonfigurationsinvariante das Entfernen oder Hinzufügen genau der Produktmerkmale eingeschränkt, die zur ungültigen Rekonfigurationsoperation geführt haben. Kann der Realisierer auch durch mehrfaches Anpassen der Rekonfigurationsanfrage den Konflikt nicht lösen, gibt er eine Fehlermeldung mit der betroffenen Rekonfigurationsoperation zurück.
Durch das Umformulieren von Produktmerkmalen durch die Operationsrealisierer kann der Kunde am Ende ein Rekonfigurationskonzept erhalten, dessen Produktmerkmale ihm nicht bekannt sind. Daher muss ein Operationsrealisierer den Inhalt seines zurückgemeldeten Rekonfigurationskonzepts gegenüber dem beauftragenden Kunden erklären können.

3.7.5 Rekonfigurationsszenario

Nachdem ein Rekonfigurationskonzept basierend auf dem Wissen der einzelnen Operationsrealisierer erstellt und nach Zustimmung des Kunden umgesetzt wurde (siehe nächster

Abschnitt 3.8), ist ein `RekonfigurationsSzenario` entstanden. Das Rekonfigurationsszenario rs besteht aus einem Ausgangs- ($\mathbf{P_{vorher}}$) und einem Endprodukt ($\mathbf{P_{nachher}}$):

$$rs = (\mathbf{P_{vorher}}, \mathbf{P_{nachher}})$$

3.8 Umsetzung eines Rekonfigurationskonzepts

Nachdem eine Kette von Operationsrealisierern ein Rekonfigurationskonzept an einen Kunden zurückgemeldet hat und der Rekonfigurationsauftrag zu Stande gekommen ist, erfolgt die eigentliche Realisierung des Konzepts. Dazu wird jedoch nicht das erstellte konkrete Rekonfigurationskonzept, sondern erneut die abstrakten Rekonfigurationsziele beauftragt, da jeder beauftragte Operationsrealisierer nur die von ihm unterstützten Rekonfigurationsoperationen, und nicht zwangsläufig die konkreten Rekonfigurationsoperationen anderer Realisierer kennt. Da sich zwischen der Erstellung des Rekonfigurationskonzepts und der Umsetzung des Konzepts einige Randbedingungen für die Rekonfigurationsentscheidungen einzelner Operationsrealisierer ändern können, muss ein Operationsrealisierer bei der Erstellung des Konzepts seine Entscheidungen abspeichern, um sie bei der Umsetzung des Konzepts exakt wiederholen zu können. Kann ein Operationsrealisierer das von ihm erstellte Rekonfigurationskonzept nicht mehr umsetzen, meldet er diesen Fehler mit der betroffenen Rekonfigurationsoperation an den beauftragenden Realisierer zurück. Dieser muss dann erneut eine Rekonfigurationsanfrage mit geändertem Rekonfigurationziel stellen.

3.9 Rekonfigurationsbeispiel

Werkstätten bieten für eine begrenzte Menge an Fahrzeugen das Rekonfigurationsziel „Repariere Fahrzeug“ an. In diesem Beispiel soll ein Tausch der Software für die Klimaautomatik Abhilfe für das Problem schaffen. Da der Mitarbeiter der Werkstatt Hilfsmittel für diese Aktion benötigt, wird ein Service-Tester als Operationsrealisierer eingesetzt. Um die Kontrolle über die Auswahl der entsprechenden Software im Service-Tester zu belassen, bietet dieser nur das abstrakte Rekonfigurationsziel „Ersetze die Software des KLA-SG“ an. Eine konkrete Auswahl der einzusetzenden Software durch den Werkstattmitarbeiter ist nicht möglich. Hier findet eine Aufteilung der Diagnose zwischen dem Werkstattmitarbeiter und dem Service-Tester statt.
Der Kunde stellt im ersten Schritt eine Rekonfigurationsanfrage an einen Werkstattmitarbeiter, die das o.g. Rekonfigurationsziel „Repariere Fahrzeug“ enthält. Konkret wird dieses Rekonfigurationsziel wie folgt formuliert:

$$\begin{aligned} \mathbf{P} &= \{pm_1, pm_2, pm_3, pm_4, pm_5, pm_6, pm_7, pm_{30}, pm_{31}, \ldots\} \\ ro &= (\{\}, \{pm_7\}) \\ ri &= (\{\}, \{\}) \end{aligned}$$

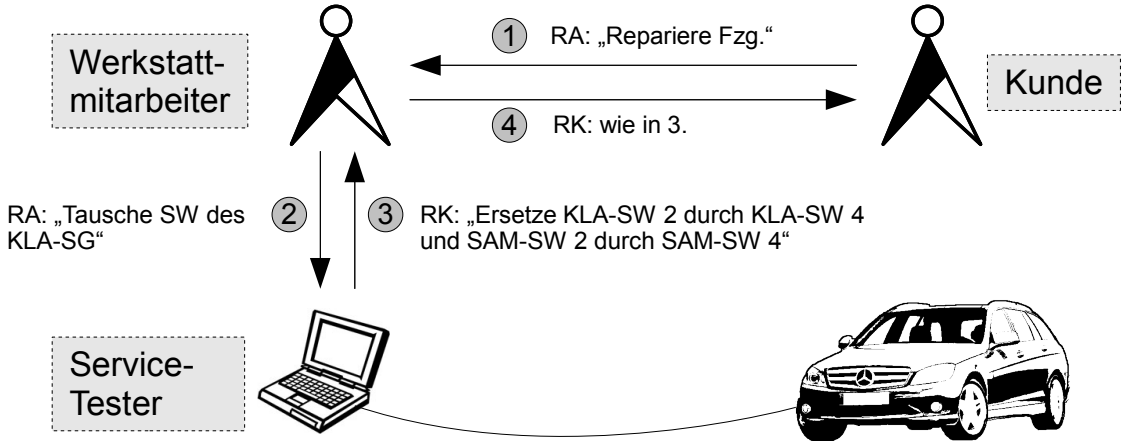


Abbildung 3.7: Beispiel einer Operationsrealisierung

mit

$$
\begin{aligned}
pm_1 &= \text{„hat 3-Zonen Klimaautomatik (KLA)“} \\
pm_2 &= \text{„hat KLA-SG“} \\
pm_3 &= \text{„hat KLA-SW 2“} \\
pm_4 &= \text{„hat Signalerfass- und Ansteuermodul (SAM)“} \\
pm_5 &= \text{„hat SAM-SW 2“} \\
pm_6 &= \text{„ist USA-Variante“} \\
pm_7 &= \text{„ist defekt“} \\
pm_{30} &= \text{„hat 6-Zylinder-Motor“} \\
pm_{31} &= \text{„hat Innenraumschutz“}
\end{aligned}
$$

Um pm_7 aus dem gegebenen Produkt gemäß der Anfrage des Kunden zu entfernen, erstellt der Werkstattmitarbeiter eine Diagnose indem er die gewünschte Rekonfigurationsoperation konkretisiert und die Funktion h darauf anwendet, also $h(ro)$. Als Ergebnis seiner Diagnose erhält er das folgende Rekonfigurationsziel, das er im Schritt 2 als Rekonfigurationsanfrage an den Service-Tester übergibt:

$$
\begin{aligned}
\mathbf{P} &= \{pm_1, pm_2, pm_3, pm_4, pm_5, pm_6, pm_7, pm_{30}, pm_{31}, \ldots\} \\
ro &= (\{\}, \{pm_3\}) \\
ri &= (\{\}, \{\})
\end{aligned}
$$

Zur Umsetzung dieser Anfrage wendet der Service-Tester die Funktion h wiederum auf die gegebene Rekonfigurationsoperation an, also $h(ro)$. Dabei ensteht ein Rekonfigurationskonzept mit folgender Rekonfigurationsoperation, die dem Werkstattmitarbeiter in

Schritt 3 zurückgemeldet wird:

$$ro \quad = \quad (\{pm_8, pm_9\}, \{pm_3, pm_5\})$$

mit

$$pm_8 \quad = \quad \text{„hat KLA-SW 4“}$$
$$pm_9 \quad = \quad \text{„hat SAM-SW 4“}$$

Dieses Rekonfigurationskonzept meldet der Werkstattmitarbeiter schließlich in Schritt 4 auch dem Kunden zurück.
Der Service-Tester hat die Rekonfigurationsoperation $(\{\}, \{pm_3\})$ zu $(\{pm_8, pm_9\}, \{pm_3, pm_5\})$ konkretisiert und vervollständigt. Das Wissen dazu könnte in einer Tabelle wie nachfolgend beschrieben dokumentiert werden:

Nr.	**RA**	**RK**
1	$(P_1, (\{\}, \{pm_3\}), (\{\}, \{\}))$	$(\{pm_8, pm_9\}, \{pm_3, pm_5\})$
2	$(P_2, (\{\}, \{pm_3\}), (\{\}, \{\}))$	$(\{pm_8, pm_9\}, \{pm_3, pm_5\})$
...	...	...
n	$(P_n, (\{\}, \{pm_3\}), (\{\}, \{\}))$	$(\{pm_8, pm_9\}, \{pm_3, pm_5\})$
...	...	...

Tabelle 3.1: Tabelle mit Funktionsbeschreibungen

In der Tabelle wurde Zeile 1 vom Service-Tester ausgewählt, da diese die Funktion für die entsprechende Rekonfigurationsanfrage beschreibt. Neben dieser Zeile ist in weiteren Zeilen dieselbe Funktionsbeschreibung für andere Rekonfigurationsanfragen angegeben, die sich auf die Produkte $P_2, \ldots, P_n$ beziehen. Es existieren parallel dazu weitere Funktionsbeschreibungen für den Austausch der Software jedes anderen Steuergeräts.
Eine explizite Angabe aller Zuordnungen zwischen Rekonfigurationsanfrage und Rekonfigurationskonzept wie in Tabelle 3.1 steht in der Praxis jedoch außer Frage, da die Anzahl der Zuordnungen zu hoch wäre. Es muss daher eine Funktionsbeschreibung gefunden werden, die mit weniger Aufwand zu erstellen ist. Dennoch muss jede Zuordnung durch die Beschreibung abgedeckt sein.
Gleichzeitig kann bei einer verteilten Funktionsbeschreibung die Dokumentation der Rekonfigurationsfunktion auf die für ein Produktmerkmal verantwortlichen Personen verteilt werden. Andererseits muss dabei das Problem möglicher Widersprüche zwischen den verteilt entstandenen Dokumentationen gelöst werden.

3.10 Zusammenfassung

Im hier dargestellten Domänenmodell zum Rekonfigurieren wurden die grundlegenden Teilnehmer und der Ablauf einer Rekonfiguration beschrieben. Dabei wurde herausgearbeitet, welches Wissen durch welchen Teilnehmer zur Verfügung gestellt werden muss

und welche Regeln beim Rekonfigurieren eingehalten werden müssen. Es wurde nicht beschrieben, wie das benötigte Wissen, speziell das Wissen des Operationsrealisierers, erstellt wird, das seinem Vorgehen beim Vervollständigen und Konkretisieren von Rekonfigurationsoperationen zu Grunde liegt.
Durch das hier eingeführte Konzept der Operationsrealisierer ist es möglich, die Rekonfigurationsziele mehrerer Parteien zu berücksichtigen, denn sie sind die Schnittstellen im Rekonfigurationsprozess, an denen zusätzliches Wissen oder Rekonfigurationsziele verarbeitet werden können. Wird ein Produkt als Menge von Produktmerkmalen beschrieben, kann es mit den Methoden dieses Modells beliebig rekonfiguriert werden.
Aufbauend auf dem erstellten Modell können weitere Betrachtungen zum Rekonfigurieren durchgeführt werden, insbesondere über die Erstellung des Rekonfigurationswissens.

4 Erzeugen und Dokumentieren von Rekonfigurationswissen

Nachdem im Domänenmodell die wichtigen allgemeinen Zusammenhänge beim Rekonfigurieren dargestellt wurden, wird im Folgenden erläutert, wie das dazu benötigte Rekonfigurationswissen dokumentiert werden kann. Zur Realisierung von Rekonfigurationsoperationen wurden Funktionen zur Abbildung einer Rekonfigurationsanfrage auf ein Rekonfigurationskonzept und zur Vervollständigung und Konkretisierung einer Rekonfigurationsoperation eingeführt. Das Rekonfigurationswissen steckt damit in der Beschreibung dieser Funktionen. Bei dieser Überführung eines vorhandenen Produkts in ein anderes Produkt werden die Ziele des Kunden und des Unternehmens berücksichtigt.
Zur Abbildung dieser Rekonfigurationsszenarien müssen die Beziehungen, die beim Austausch von Komponenten ausgewertet werden, entsprechend definiert sein und operativ korrekt eingesetzt werden.

4.1 Aufteilung des Rekonfigurationswissens

Abhängig vom Abstraktionsgrad der Rekonfigurationsziele, die durch einen Operationsrealisierer zur Verfügung gestellt werden, ist der Umfang des bereitzustellenden Rekonfigurationswissens höher oder geringer. Bietet ein Realisierer sehr abstrakte Rekonfigurationsziele an, muss er selbst über viel Rekonfigurationswissen verfügen, um eine Rekonfigurationsanfrage und damit ein Rekonfigurationsziel zu konkretisieren. Abstraktere Rekonfigurationsziele bieten dafür auch umfangreichere Möglichkeiten eigene Rekonfigurationsziele einzubringen. Werden dagegen sehr konkrete Rekonfigurationsziele angeboten, ist nur geringes Rekonfigurationswissen zur Umsetzung einer Anfrage notwendig.
Da die Aktualisierung einer Fahrzeugsoftware automatisiert in der Werkstatt ablaufen soll, soll der Service-Tester also möglichst abstrakte Rekonfigurationsziele anbieten und das dafür notwendige Realisierungswissen vollständig im Service-Tester zur Verfügung stehen. In die darin modellierten Entscheidungen über die auszuwählende Zielkonfiguration müssen die weiter oben bereits geforderten Kriterien Kundenzufriedenheit und Wirtschaftlichkeit eingehen[1]. Die in [Hei04] gewählte Trennung zwischen dem Konfigura-

[1] Die grundsätzliche Fehlerfreiheit der zum Einsatz kommenden Konfigurationen muss durch die korrekte Dokumentation des Konfigurationswissens sichergestellt werden. Durch diese Fehlerfreiheit ist nicht ausgeschlossen, dass innerhalb einer Komponente der Konfiguration Fehler enthalten sind. Die Konfiguration an sich enthält aber nur Komponenten, die korrekt zusammen arbeiten.

tionswissen im Service-Tester (Baureihen-Releases) und dem Realisierungswissen[2] beim Werkstattmitarbeiter (Auswahl der Wunschkonfiguration) ist im Rahmen dieser Arbeit daher nur bedingt sinnvoll, da hier von einer sehr hohen Produktvarianz als Ausgangsmenge für die Rekonfiguration ausgegangen wird und die Unternehmensstrategie beim Rekonfigurieren nur durch einen hohen Automatisierungsgrad zuverlässig und einheitlich umgesezt werden kann. Statt der expliziten Auswahl der Wunschkonfiguration durch den Werkstattmitarbeiter[3] soll dieser lediglich eine Diagnose auf der Abstraktionsebene *Update* oder *Defekt im Steuergerät XY* stellen und das richtige vom Service-Tester angebotene Rekonfigurationsziel zum Tausch der entsprechenden Software auswählen[4].

4.2 Fahrzeugbeispiel als Produktbaum

Im Abschnitt 3.9 wurde bereits ein Beispiel für die Rekonfiguration eingeführt, das hier erneut aufgegriffen und erweitert werden soll. Die grundlegende Strukturierung der Fahrzeugkomponenten wird in einem Produktbaum (UND-ODER-Graph) in Anlehnung an die Stücklistenstruktur des PDM-Systems DIALOG (siehe auch [Sin03]) der Daimler AG dargestellt (Abb. 4.1). Zur Verdeutlichung der einzelnen Varianten wird die Struktur auf der Ebene der Positionsvarianten um gruppierende ODER-Knoten erweitert, statt alle Varianten auf der gleichen Hierarchieebene einzuordnen. Die entsprechenden aussagenlogischen Coderegeln zur Auswahl der Varianten sind dann im folgenden Abschnitt zu finden.

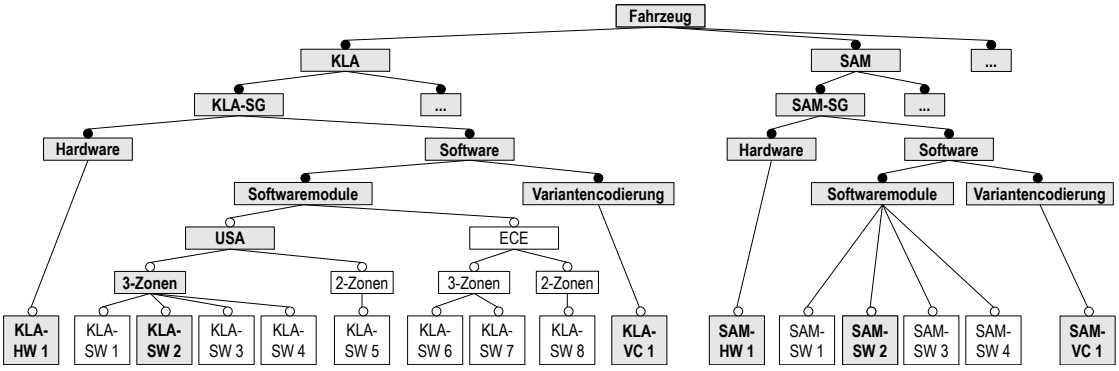


Abbildung 4.1: Produktbaum mit Konfigurationswissen und einem Beispielfahrzeug

Jedes im Produktbaum dargestellte Produktmerkmal, das tatsächlich Teil des Beispielfahrzeugs ist, ist grau hinterlegt. Neben diesem so markierten Beispielfahrzeug sind 31

[2] Zu den Begriffen Konfigurations- und Realisierungswissen siehe Abschnitt 5.2.2

[3] Dem Werkstattmitarbeiter müsste analog zum Service-Tester das Wissen über adäquate, also kundengerechte und wirtschaftliche, Ersatzkonfigurationen für jeden Reparaturfall bereitgestellt werden, um eine qualifizierte Entscheidung treffen zu können.

[4] Eine Unterstützung bei dieser Diagnose kann zwar ebenfalls durch den Service-Tester geliefert werden, diese Funktion soll aber kein Bestandteil dieser Arbeit sein.

weitere Konfigurationen möglich, wenn alle Knoten mit der Bezeichnung „. . .“ jeweils als eine Variante gezählt werden. Zur Vereinfachung des Beispiels wurde für einige Merkmale nur eine mögliche Variante angegeben.
Alle im Produktbaum dargestellten Produktmerkmale können in *Fahrzeugkomponenten* und *Codes* (siehe folgender Abschnitt) unterteilt werden. Fahrzeugkomponenten sind Produktmerkmale der Form *hat KLA-SW 2* oder *hat SAM-HW 1*. Zur Einführung der Beziehungen zwischen den Produktmerkmalen und des Algorithmus zur Ermittlung des Rekonfigurationskonzepts werden hier jedoch sämtliche Produktmerkmale, die sich auf die Blätter im Produktbaum beziehen, vereinfachend direkt mit dem Namen der entsprechenden Komponente bezeichnet. Also statt *hat KLA-SW 2* nur *KLA-SW 2*.

4.3 Grundlegende Beziehungstypen und Begriffe

Zusätzlich zu den Informationen aus dem Produktbaum im vorigen Abschnitt sind weitere Begriffe und Beziehungen zwischen den Produktmerkmalen zur Abbildung des vollständigen Konfigurations- und Realisierungswissens notwendig. Diese sollen hier allgemein eingeführt und am Beispiel gezeigt werden.

4.3.1 Steuergerät

Allgemein

Der Begriff des Steuergeräts wurde bereits in der Einführung in Abschnitt 2.2 als Zusammenfassung mehrerer Hard- und Softwarekomponenten und im Beispiel in Abschnitt 3.9 sowie in Abschnitt 4.1 als Einstiegspunkt für die Rekonfiguration einer Software verwendet. Dementsprechend soll das Steuergerät als Oberbegriff für eine Menge von Hard- und Softwarekomponenten definiert werden, das wie folgt dargestellt wird:

$$\mathbf{SG} = \{teil_1, \ldots, teil_n\}$$

Dabei gilt

$$teil_i \in \mathbf{HW} \vee teil_i \in \mathbf{SW},\ 1 \leq i \leq n$$

wobei **HW** und **SW** die Mengen aller möglichen Hard- und Softwarekomponenten sind. $teil_1$ bis $teil_n$ sind dabei unterschiedliche Varianten oder Versionen der jeweiligen Hard- oder Software, die alternativ zueinander verbaut werden können[5]. Die Menge aller Steuergeräte soll $\mathbf{SG_{gesamt}}$ heißen.

[5] Handelt es sich bei den Softwarekomponenten um Softwaremodule derselben Hardware, werden diese nicht alternativ, sondern gleichzeitig verbaut und gehören dennoch zum gleichen Steuergerät.

Beispiel

Im Produktbaum in Abb. 4.1 sind bereits die beiden Steuergeräte *KLA-SG* und *SAM-SG* mit den folgenden Hard- und Softwarekomponenten dargestellt:

$$\begin{aligned} \textbf{KLA-SG} &= \{\textit{KLA-HW 1, KLA-SW 1, ..., KLA-SW 8}\} \\ \textbf{SAM-SG} &= \{\textit{SAM-HW 1, SAM-SW 1, ..., KLA-SW 4}\} \end{aligned}$$

4.3.2 Codes

Allgemein

Die Entscheidungen zur Bildung der Varianten im Produktbaum werden anhand von Codes getroffen. Über die Struktur *code_komp* der Form

$$code\_komp = (teil, C)$$

ist jeder Fahrzeugkomponente *teil* eine Codebedingung C zugewiesen. Ist C erfüllt, kommt die Komponente zum Einsatz, d.h. es gilt:

$$C \Rightarrow teil$$

Die Menge aller Codebedingungen soll **CODE_KOMP** heißen. Je nach Verwendung eines Codes in der aussagenlogischen Formel C kann so die Abhängigkeit einer Fahrzeugkomponente von einem Code (und damit einem Produktmerkmal) oder die Inkompatibilität einer Fahrzeugkomponente zu dem Code ausgedrückt werden.

Beispiel

Einige Codes des Beispielfahrzeugs sind bereits im Produktbaum ersichtlich. Alle weiteren sind im Folgenden zusätzlich aufgelistet. Zur Vereinfachung gegenüber dem Beispiel aus Abschnitt 3.9 werden auch diese Produktmerkmale umbenannt:

$$\begin{aligned} \textit{3kla} &= \text{„hat 3-Zonen Klimaautomatik (KLA)“} \\ \textit{usa} &= \text{„ist USA-Variante“} \\ \textit{defekt} &= \text{„ist defekt“} \\ \textit{6zylinder} &= \text{„hat 6-Zylinder-Motor“} \\ \textit{irs} &= \text{„hat Innenraumschutz“} \end{aligned}$$

Weiterhin existieren die folgenden Codes zur Bildung der Varianten im Produktbaum:

$$\begin{aligned} \textit{2kla} &= \text{„hat 2-Zonen Klimaautomatik (KLA)“} \\ \textit{ece} &= \text{„ist ECE-Variante“} \end{aligned}$$

Die für das Beispiel geltenden Abhängigkeiten und Inkompatibilitäten sind durch die folgenden Formeln gegeben:

$$\begin{aligned} usa \wedge 3kla &\Rightarrow KLA\text{-}SW\ 1\ bis\ KLA\text{-}SW\ 4 \\ usa \wedge 2kla &\Rightarrow KLA\text{-}SW\ 5 \\ ece \wedge 3kla &\Rightarrow KLA\text{-}SW\ 6\ bis\ KLA\text{-}SW\ 7 \\ ece \wedge 2kla &\Rightarrow KLA\text{-}SW\ 8 \end{aligned}$$

Neben den oben genannten Codes gehen auch die Ausführungs- und Lenkungsart des Fahrzeugs in die Variantenbildung ein, also z.B. *Limousine* bzw. *Rechtslenker*. Beide wirken daher wie Codes, sind jedoch in ihrer Auswirkung auf die Fahrzeugkonfiguration so stark, dass sie gesondert behandelt werden. In diesem Beispiel sollen beide aber keine Rolle spielen.

4.3.3 Austauschbarkeiten

Allgemein

Zusätzlich zu der in Abb. 4.1 dargestellten Komponentenhierarchie sind Austauschbarkeiten zwischen den Softwarekomponenten angegeben. Durch die Austauschbarkeitsbeziehung wird angegeben, dass eine Komponente anstelle einer anderen Komponente in einer Konfiguration eingesetzt werden kann (Konfigurationswissen) und auch eingesetzt werden soll (Realisierungswissen). Dies entspricht dem allgemeinen Bild einer Versionierung von Softwarekomponenten, wobei die Ordnung innerhalb der Versionsstände in diesem Fall explizit durch eine Beziehung hergestellt statt durch ein Nummerierungsschema vorgegeben ist. Eine Austauschbarkeitsbeziehung soll als Struktur aus der Form

$$aus = (sw_{vor}, sw_{nach})$$

dargestellt werden, wobei sw_{nach} die Software ist, die anstelle von sw_{vor} eingesetzt werden kann und soll. Die Menge aller Austauschbarkeitsbeziehungen soll als **AUS** bezeichnet werden.
Weiterhin soll der Begriff der Austauschkette definiert werden, um mehrere Komponenten, zwischen denen jeweils eine Austauschbarkeit angegeben ist, zusammenzufassen. Die Austauschkette beginnt mit einer Ausgangskomponente und endet mit einer Endkomponente, wobei die Endkomponente die letzte Komponente entlang des Pfades ist, der durch die Austauschbarkeiten ausgehend von der Ausgangskomponente gebildet wird. Dabei werden die Nachfolgerkette und die Vorgängerkette als Spezialform der Austauschkette unterschieden. Für die Nachfolgerkette wird der Pfad zwischen Ausgangs- und Endkomponente in Richtung der Austauschbarkeiten gebildet, für die Vorgängerkette genau umgekehrt.

Eine Austauschkette soll als Struktur *kette* der Form

$$kette = (sw_{ausgang}, sw_{ende}, \mathbf{T})$$

dargestellt werden, wobei $sw_{ausgang}$ die Ausgangs- und sw_{ende} die Endkomponente bezeichnet. **T** enthält alle Komponenten, die Teil der Austauschkette sind einschließlich $sw_{ausgang}$ und sw_{ende}. Die Menge aller Austauschketten einer Softwarekomponente soll als **KETTE_N** (Nachfolgerketten) bzw. **KETTE_V** (Vorgängerketten) bezeichnet werden.

Beispiel

Die Austauschbarkeiten der für das Beispiel relevanten Softwarekomponenten sind entsprechend Abb. 4.2 dokumentiert. Die Austauschbarkeit zwischen der *KLA-SW 1* und der *KLA-SW 2* wird beispielsweise als $aus_1 = (KLA\text{-}SW\ 1, KLA\text{-}SW\ 2)$ notiert. Eine

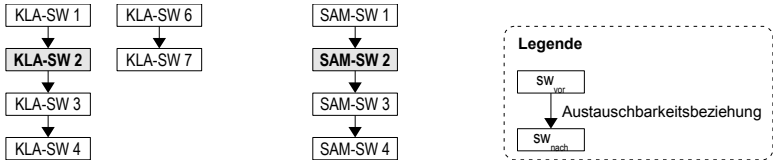


Abbildung 4.2: Austauschbarkeiten zwischen Softwarekomponenten

beispielhafte Nachfolgerkette mit der Ausgangssoftware *KLA-SW 2* ist hierbei

$$kette = (KLA\text{-}SW\ 2, KLA\text{-}SW\ 4, \{KLA\text{-}SW\ 2,\ KLA\text{-}SW\ 3,\ KLA\text{-}SW\ 4\})$$

4.3.4 Kompatibilität zwischen Hard- und Software

Allgemein

Im Produktbaum sind durch die Strukturierung über Positionen und Positionsvarianten sowie die Gültigkeitsbedingungen durch Code bereits Aussagen zu den Verwendungsmöglichkeiten der Komponenten und damit implizit zur Kompatibilität zwischen den Hard- und Softwarekomponenten enthalten, die jedoch nicht vollständig sind. Daher wird das Wissen über die Kompatibilität zwischen Hard- und Software in Form einer expliziten Kompatibilitätsbeziehung als Struktur *hw_sw_komp* der Form

$$hw\_sw\_komp = (hw, sw)$$

bereitgestellt. Dabei ist *sw* die Software, die kompatibel zur Hardware *hw* ist. Die Menge aller Kompatibilitätsbeziehungen soll **HW_SW_KOMP** heißen.

Beispiel

Für das Beispiel sollen die Kompatibilitätsbeziehungen entsprechend Abb. 4.3 gelten. Die Kompatibilität zwischen der *KLA-HW 1* und der *KLA-SW 1* wird beispielsweise als $hw\_sw\_komp_1 = (KLA\text{-}HW\ 1, KLA\text{-}SW\ 1)$ notiert.

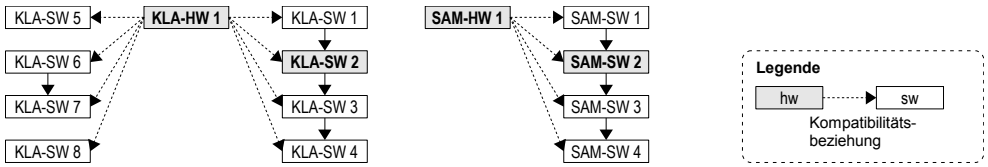


Abbildung 4.3: Kompatibilität zwischen Hard- und Software

4.4 Funktionsbeschreibung mit Abhängigkeiten

Im folgenden Abschnitt wird der Algorithmus zur Ermittlung eines Rekonfigurationskonzepts für ein Softwareupdate beschrieben. Dabei werden die benötigten Beziehungstypen aus dem vorherigen Abschnitt verwendet und der Algorithmus am Beispiel verdeutlicht. Zur Dokumentation der Abhängigkeiten zwischen verschiedenen Steuergerätesoftwaren werden die beiden Dokumentationsarten *Einzelabhängigkeit* und *Teilrelease* eingeführt. Beide bauen auf den bereits vorhandenen Beziehungstypen auf und erweitern diese, so dass zusätzliche Rekonfigurationsstrategien möglich werden.

4.4.1 Passive und aktive Rekonfigurationsphase

Die Ermittlung der Steuergerätesoftware findet in den beiden Phasen *Passive Rekonfigurationsphase* und *Aktive Rekonfigurationsphase* statt (Abb. 4.4). Als Vorbedingung für

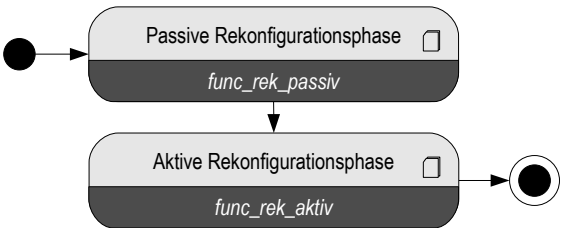


Abbildung 4.4: 2-stufige Rekonfigurationsfunktion (UML-Aktivitätsdiagramm)

diesen Algorithmus hat der Werkstattmitarbeiter basierend auf seinem Diagnosewissen ein Steuergerät ausgewählt, für das eine neue Software ermittelt werden soll.

Passive Rekonfigurationsphase ($func\_rek\_passiv$) In der ersten, passiven Phase wird die zum aktuellen Fahrzeug und dem ausgewählten Steuergerät passende Software ermittelt, ohne sonstige Produktmerkmale des Fahrzeugs anzupassen (siehe

Abschnitt 4.4.2). Dieser Algorithmus ist nicht Bestandteil dieser Arbeit, sondern wird hier als Grundlage für die folgende Phase vorgestellt.

Aktive Rekonfigurationsphase (*func_rek_aktiv*) In der zweiten, aktiven Phase werden die Abhängigkeiten der zuvor ermittelten Software aufgelöst. Hierbei werden alle solchen Änderungen von Produktmerkmalen im Fahrzeug vorgenommen, durch die ein funktionsfähiges Produkt entsteht (siehe Abschnitt 4.4.4).

4.4.2 Umsetzung der passiven Rekonfigurationsphase

Allgemein

Zur Ermittlung der Softwarekomponente, die zum aktuellen Fahrzeug passt, ohne das Fahrzeug selbst zusätzlich anpassen zu müssen, werden die in Abb. 4.5 dargestellten Schritte durchgeführt. Im Folgenden werden die Schritte näher erläutert:

Abbildung 4.5: Ermittlung einer Softwarekomponente in der passiven Rekonfigurationsphase

func_hw_sw_komp Die Funktion zur Ermittlung aller Softwarekomponenten, die kompatibel zur verbauten Hardware sind, ist wie folgt definiert:

$$\begin{aligned}\mathbf{SW_{hw}} \quad := \quad & \text{func\_hw\_sw\_komp}(hw) = \{hw\_sw\_komp_1.sw, \ldots, hw\_sw\_komp_n.sw\} \\ & \text{mit } hw\_sw\_komp_i.hw = hw,\ hw\_sw\_komp_i \in \mathbf{HW\_SW\_KOMP}, \\ & 1 \leq i \leq n,\ n = |\mathbf{HW\_SW\_KOMP}|\end{aligned}$$

Die als Eingangsparameter einzusetzende Hardware *hw* entspricht der für das vom Werkstattmitarbeiter ausgewählte Steuergerät verbauten Hardware im Fahrzeug.

func_code_komp Die Funktion zur Einschränkung der Softwarekomponenten anhand der sonstigen Produktmerkmale des Fahrzeugs ist wie folgt definiert:

$$\begin{aligned}\mathbf{SW_{code}} \quad := \quad & \text{func\_code\_komp}(\mathbf{SW_{hw}}) = \{code\_komp_1.teil, \ldots, code\_komp_n.teil\} \\ & \text{mit } code\_komp_i.teil \in \mathbf{SW},\ code\_komp_i.C = true, \\ & code\_komp_i \in \mathbf{CODE\_KOMP},\ 1 \leq i \leq n,\ n = |\mathbf{CODE\_KOMP}|\end{aligned}$$

Als Eingangsparameter geht die im Schritt $func\_hw\_sw\_komp$ ermittelte Menge $\mathbf{SW_{hw}}$ ein.

func_austausch Die Funktion zur Ermittlung der aktuellsten Softwarekomponente anhand der Austauschbarkeitsbeziehung ist wie folgt definiert:

$$\begin{aligned}sw \quad := \quad & \text{func\_austausch}(\mathbf{SW}) = kette_i.sw_{ende} \\ & \text{mit } kette_i.sw_{ende} \in \mathbf{SW},\ kette_i.sw_{ausgang} \text{ die im Fahrzeug verbaute} \\ & \text{Software}, kette_i \in \mathbf{KETTE\_N}, \mathbf{KETTE\_N} \text{ enthält die Nachfolgerketten} \\ & \text{jeder } sw \in \mathbf{SW}\end{aligned}$$

Als Eingangsparameter geht die im Schritt $func\_code\_komp$ ermittelte Menge $\mathbf{SW_{code}}$ ein. Nach dieser Funktion muss eine eindeutige Software sw gefunden worden sein. D.h., es dürfen nach dem vorherigen Schritt $func\_code\_komp$ nur Softwarekomponenten mit genau einer Nachfolgerkette in der Menge **SW** verblieben sein. Dazu müssen die Codebedingungen entsprechend restriktiv formuliert worden sein.

Nach dieser Softwareermittlung steht fest, welche Software auf das betreffende Steuergerät geflasht werden soll.

Beispiel

Für das Beispiel würde dieser Algorithmus die in Tab. 4.1 gezeigten Ergebnisse liefern. Das Fahrzeug besteht dann aktuell aus diesen Produktmerkmalen:

$$\begin{aligned}\mathbf{P} \quad = \quad & \{3kla,\ usa,\ 6zylinder,\ irs,\ \textit{KLA-HW 1},\ \textit{KLA-SW 4}, \\ & \textit{SAM-HW 1},\ \textit{SAM-SW 2},\ \ldots\}\end{aligned}$$

4.4.3 Beziehungstyp Einzelabhängigkeit

Allgemein

Eine Einzelabhängigkeit ist eine gerichtete Beziehung zwischen zwei Softwarekomponenten, die dann ausgewertet wird, wenn die Softwarekomponente am Startpunkt der Beziehung einem Fahrzeug während einer Rekonfiguration hinzugefügt wird. Die Beziehung

Funktionsschritt	**Ergebnis**
Ermittlung aller Softwarekomponenten, die kompatibel zur verbauten Hardware sind ($func\_hw\_sw\_komp$)	$\mathbf{SW_{hw}} = \{$ *KLA-SW 1, KLA-SW 2, KLA-SW 3, KLA-SW 4, KLA-SW 5, KLA-SW 6, KLA-SW 7, KLA-SW 8*$\}$
Einschränkung der gefundenen Softwarekomponenten anhand der sonstigen existierenden Poduktmerkmale des Fahrzeugs ($func\_code\_komp$)	$\mathbf{SW_{code}} = \{$ *KLA-SW 1, KLA-SW 2, KLA-SW 3, KLA-SW 4*$\}$
Ermittlung der aktuellsten Softwarekomponente anhand der Austauschbarkeitsbeziehung ($func\_austausch$)	$sw =$ *KLA-SW 4*

Tabelle 4.1: Beispielergebnisse für die passive Rekonfigurationsphase

ist nur dann gültig und führt nur dann zu einer weiteren Veränderung des Fahrzeugs, wenn die Softwarekomponente am Endpunkt der Beziehung oder eine ihrer Nachfolgerkomponenten im betreffenden Fahrzeug gültig ist und die Gültigkeitsbedingung der Beziehung selbst wahr ist. Durch diese Angabe eines mindestens benötigten Softwarestands am Endpunkt der Beziehung wird eine zuvor dokumentierte Austauschbarkeitsbeziehung bzw. eine Nachfolgerkette vorausgesetzt. Weiterhin wird so der Dokumentationsaufwand minimiert, ohne jedoch die Mächtigkeit des Ansatzes einzuschränken. Entgegen der Dokumentation eines Mindeststands bei der benötigten Software ist die Einzelabhängigkeitsbeziehung ausgehend von einer Softwarekomponente für jede Software erneut zu dokumentieren, wenn die Abhängigkeit weiterhin besteht. Es können nur Software-, aber keine Hardwareänderungen durch diese Beziehung ausgelöst werden. Ebenso sorgt die Richtung der Beziehung für eine höhere Anzahl möglicher Rekonfigurationsszenarien.
Mit diesen Eigenschaften wurde die Beziehung so definiert, dass der Algorithmus der passiven Rekonfigurationsphase möglichst vollständig wiederverwendet werden kann.
Eine Einzelabhängigkeit wird als eah der Form

$$eah = (sw_{ab}, sw_{be}, C)$$

dargestellt, wobei sw_{ab} die abhängige Software und sw_{be} die benötigte Software ist. C stellt eine aussagenlogische Codebedingung dar, die festlegt, in welchen Fahrzeugen die Einzelabhängigkeitsbeziehung gültig ist:

$$C \Rightarrow eah$$

Die Menge aller Einzelabhängigkeiten soll **EAH** heißen.

Beispiel

Für das Beispiel sind die in Abb. 4.6 dargestellten Einzelabhängigkeiten dokumentiert.

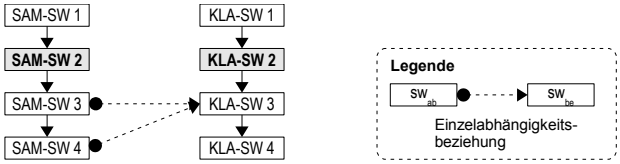


Abbildung 4.6: Einzelabhängigkeiten im Beispiel

Dabei gelten folgende Codebedingungen:

$$eah_1 = (SAM\text{-}SW\ 3, KLA\text{-}SW\ 3, irs)$$
$$eah_2 = (SAM\text{-}SW\ 4, KLA\text{-}SW\ 3, irs)$$

Beide Einzelabhängigkeitsbeziehungen sind also nur gültig, wenn der Innenraumschutz (irs) im Fahrzeug vorhanden ist.

4.4.4 Umsetzung der aktiven Rekonfigurationsphase mit Einzelabhängigkeiten

Allgemein

Nachdem in der passiven Rekonfigurationsphase eine zum Fahrzeug passende Software ermittelt wurde, die offensichtlich den Rekonfigurationsauftrag des Kunden erfüllt, wird nun überprüft, ob weitere Produktmerkmale des Fahrzeugs verändert werden müssen, um es gemeinsam mit der zuvor gefundenen Software wieder in ein funktionsfähiges Produkt zu überführen. Dazu werden die Einzelabhängigkeiten ausgehend von der zuvor ermittelten Software ausgewertet. Anschließend wird für jede der so erhaltenen Softwarekomponenten eine Kompatibilitätsprüfung zu den Produktmerkmalen des Fahrzeugs durchgeführt (siehe Abb. 4.7). Die einzelnen Funktionsschritte sind im Folgenden erläutert:

func_eah_ermitteln Die Funktion zur Ermittlung aller Einzelabhängigkeiten ausgehend von der abhängigen Software ist wie folgt definiert:

$$\begin{aligned} \mathbf{EAH_{pruefen}} \quad := \quad & \text{func\_eah\_ermitteln}(sw) = \{eah_1, \ldots, eah_n\} \\ & \text{mit } eah_i.sw_{ab} = sw,\ eah_i \in \mathbf{EAH}, \\ & 1 \leq i \leq n,\ n = |\mathbf{EAH}| \end{aligned}$$

Die als Eingangsparameter einzusetzende Software sw entspricht der zuvor in der passiven Rekonfigurationsphase ermittelten Software. Aus der Funktion resultiert

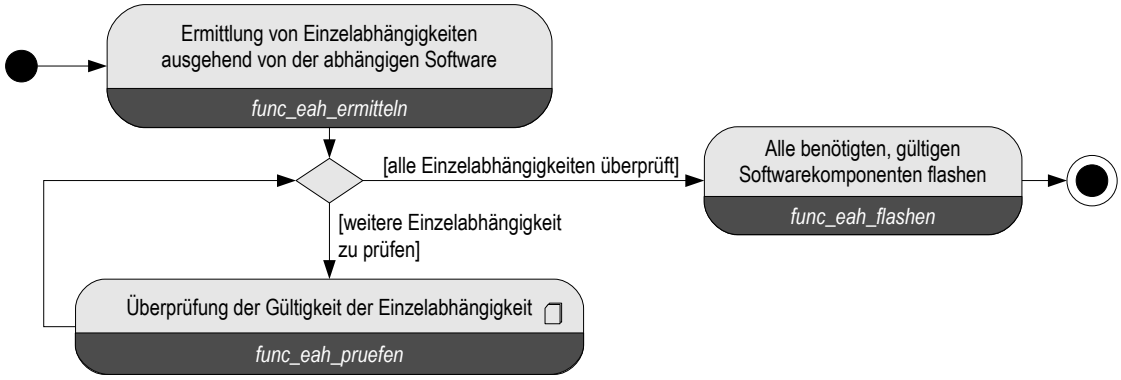


Abbildung 4.7: Auswertung der Einzelabhängigkeiten

die Menge $\mathbf{EAH_{pruefen}}$, die alle Einzelabhängigkeiten mit der Software sw als abhängiger Software enthält.

func_eah_pruefen Die Funktion zur Überprüfung der Gültigkeit der Einzelabhängigkeit ist in Abb. 4.8 dargestellt. Das Resultat ist die Menge aller benötigten Softwarekomponenten, die auf Grund von gültigen Einzelabhängigkeiten geflasht werden sollen.

func_eah_flashen Alle zuvor ermittelten benötigten Softwarkomponenten werden geflasht.

Im Folgenden sind die einzelnen Funktionsschritte der Funktion $func\_eah\_pruefen$ näher erläutert:

func_eah_sw_kette Die Funktion zur Ermittlung der Nachfolgerkette, ausgehend von der mindestens benötigten Softwarekomponente, ist wie folgt definiert:

$$\begin{aligned} \mathbf{SW\_KETTEN} \quad := \quad & \text{func\_eah\_sw\_kette}(eah) = \{kette_1, \ldots, kette_n\} \\ & \text{mit } kette_i.sw_{ausgang} = eah.sw_{be},\ kette_i \in \mathbf{KETTE\_N}, \\ & 1 \leq i \leq n,\ n = |\mathbf{KETTE\_N}| \end{aligned}$$

Die als Eingangsparameter einzusetzende Einzelabhängigkeit eah entspricht der zuvor ermittelten Einzelabhängigkeit. Aus der Funktion resultiert die Menge **SW_KETTEN**, die alle Nachfolgerketten der mindestens benötigten Software aus eah enthält.

func_eah_sw_im_fahrzeug Die Funktion zur Überprüfung, ob die Softwarekomponente oder einer ihrer Nachfolger bereits im Fahrzeug enthalten ist, ist wie folgt

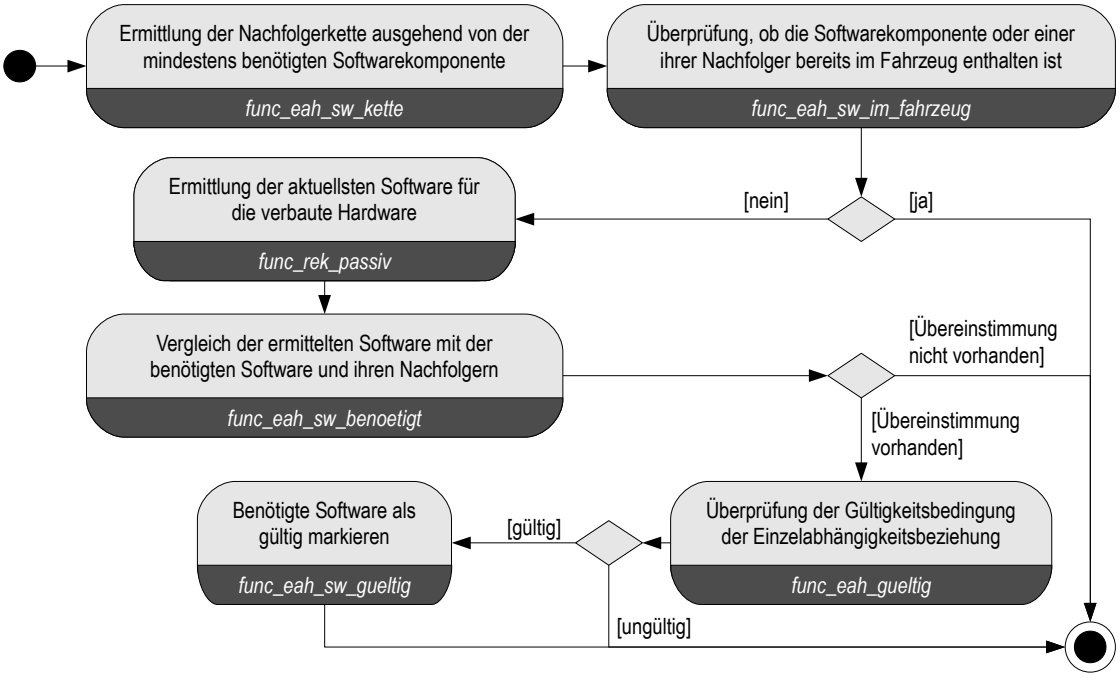


Abbildung 4.8: Überprüfung der Gültigkeit einer Einzelabhängigkeit

definiert:

$$bool := \text{func\_eah\_sw\_im\_fzg}(\mathbf{SW\_KETTEN}) = \begin{cases} \text{ja:} & \mathbf{P} \cap kette_i.\mathbf{T} \neq \{\} \\ \text{nein:} & \text{sonst} \end{cases}$$
$$\text{mit } kette_i \in \mathbf{SW\_KETTEN}, 1 \leq i \leq n,\ n = |\mathbf{SW\_KETTEN}|$$

Die als Eingangsparameter einzusetzende Menge von Nachfolgerketten **SW_KETTEN** entspricht den zuvor ermittelten Softwareketten. Aus der Funktion resultiert die Aussage *bool* darüber, ob die Einzelabhängigkeit bereits im Fahrzeug aufgelöst ist.

func_rek_passiv Die Funktion zur Ermittlung der aktuellsten Software für die verbaute Hardware entspricht der in Abschnitt 4.4.2 bereits beschriebenen Funktion. Dort wird die Hardware *hw* als Eingangsparameter eingesetzt, die für das Steuergerät der in Schritt $func\_eah\_sw\_kette$ ermittelen Softwarekomponenten im Fahrzeug verbaut ist. Aus der Funktion resultiert die Softwarekomponente, die für die Hardware des betreffenden Steuergeräts unter Berücksichtigung der Rekonfigurationsstrategie zum Fahrzeug passen würde.

func_eah_sw_benoetigt Die Funktion zum Vergleich der ermittelten Software mit der

benötigten Software und ihren Nachfolgern ist wie folgt definiert:

$$bool := \text{func\_eah\_sw\_benoetigt}(sw) = \begin{cases} \text{Übereinst. vorh.:} & sw \in kette_i.\mathbf{T} \\ \text{Übereinst. nicht vorh.:} & \text{sonst} \end{cases}$$
$$\text{mit } kette_i \in \mathbf{SW\_KETTEN}, 1 \leq i \leq n,\ n = |\mathbf{SW\_KETTEN}|$$

Die als Eingangsparameter einzusetzende Software *sw* entspricht der zuvor in Schritt *func_rek_passiv* ermittelten Software. Aus der Funktion resultiert die Aussage *bool* darüber, ob die Einzelabhängigkeit durch die zuvor ermittelte Software aufgelöst werden kann. Wird keine Übereinstimmung gefunden, ist die Einzelabhängigkeit für dieses Fahrzeug ungültig.

func_eah_gueltig Die Funktion zur Überprüfung der Gültigkeit der Einzelabhängigkeitsbeziehung ist wie folgt definiert:

$$bool := \text{func\_eah\_gueltig}(eah) = \begin{cases} \text{gültig:} & eah.C = true \\ \text{ungültig:} & \text{sonst} \end{cases}$$

Die als Eingangsparameter einzusetzende Einzelabhängigkeit *eah* entspricht der zuvor in Schritt *func_eah_sw_kette* eingesetzten Einzelabhängigkeit. Aus der Funktion resultiert die Aussage *bool* darüber, ob die Einzelabhängigkeit für dieses Fahrzeug gültig ist.

Beispiel

Für das Beispiel soll nun angenommen werden, dass die Software *SAM-SW 4* statt der oben im Beispiel ermittelten Software *KLA-SW 4* das Resultat der passiven Rekonfigurationsphase ist und die Einzelabhängigkeitsbeziehungen ausgehend von dieser Software ausgewertet werden. Das Fahrzeug sieht vor der Auswertung der Einzelabhängigkeiten damit wie folgt aus:

$$\mathbf{P} = \{3kla,\ usa,\ 6zylinder,\ irs,\ \textit{KLA-HW 1},\ \textit{KLA-SW 2},\ \textit{SAM-HW 1},\ \textit{SAM-SW 4},\ \ldots\}$$

Im Schritt *func_eah_ermitteln* resultieren so aus der Ermittlung der Einzelabhängigkeiten die in Tab. 4.2 dargestellten Ergebnisse.
Für die Überprüfung der Gültigkeit der Einzelabhängigkeiten sind die Ergebnisse in Tab. 4.3 dargestellt.
Das Produkt sieht damit nach der Rekonfiguration wie folgt aus:

$$\mathbf{P} = \{3kla,\ usa,\ 6zylinder,\ irs,\ \textit{KLA-HW 1},\ \textit{KLA-SW 4},\ \textit{SAM-HW 1},\ \textit{SAM-SW 4},\ \ldots\}$$

Funktionsschritt	**Ergebnis**
Ermittlung von Einzelabhängigkeiten ausgehend von der ermittelten Software ($func\_eah\_ermitteln$)	**EAH** = $\{eah_2\}$

Tabelle 4.2: Beispielergebnisse für die aktive Rekonfigurationsphase mit Einzelabhängigkeiten

Funktionsschritt	**Ergebnis**
Ermittlung der Nachfolgerkette ausgehend von der mindestens benötigten Softwarekomponente ($func\_eah\_sw\_kette$)	$kette_1$ = ($KLA\text{-}SW\ 3, KLA\text{-}SW\ 4$, $\{KLA\text{-}SW\ 3, KLA\text{-}SW\ 4\}$)
Überprüfung, ob die Softwarekomponente oder einer ihrer Nachfolger bereits im Fahrzeug enthalten ist ($func\_eah\_sw\_im\_fahrzeug$)	$\{KLA\text{-}SW\ 2, \dots\} \cap \{KLA\text{-}SW\ 3, \ KLA\text{-}SW\ 4\} = \{\} \rightarrow$ nein
Ermittlung der aktuellsten Software für die verbaute Hardware ($func\_rek\_passiv$)	$KLA\text{-}SW\ 4$
Vergleich der ermittelten Software mit der benötigten Software und ihren Nachfolgern ($func\_eah\_sw\_benoetigt$)	$KLA\text{-}SW\ 4 \in \{KLA\text{-}SW\ 3, KLA\text{-}SW\ 4\}$ $\rightarrow$ Übereinstimmung vorhanden
Überprüfung der Gültigkeit der Einzelabhängigkeitsbeziehung ($func\_eah\_gueltig$)	$irs = true \rightarrow$ gültig

Tabelle 4.3: Beispielergebnisse für die aktive Rekonfigurationsphase mit Einzelabhängigkeiten

4.4.5 Beziehungstyp Teilrelease

Allgemein

Ein Teilrelease stellt eine ungeordnete Menge mehrerer Softwarekomponenten dar, die gemeinsam eine baubare und abgesicherte Konfiguration für eine Menge von Steuergeräten ergeben. Dabei können Softwarekomponenten mehrerer Varianten eines Steuergeräts im gleichen Teilrelease im Sinne einer generischen (Teil-)Konfiguration enthalten sein. Die Auswahl der tatsächlich benötigten Komponenten wird gleichzeitig mit der Überprüfung der Gültigkeit des Teilreleases vorgenommen. Entgegen der Einzelabhängigkeit wird durch das Teilrelease eine explizite Festlegung der abhängigen Softwarekomponenten vorgenommen, statt einer indirekten Festlegung über eine mögliche Vererbung von Abhängigkeitsinformationen über die Austauschbarkeitsbeziehung. Neben der Rekonfiguration von Softwarekomponenten kann unter bestimmten Bedingungen auch ein Tausch von Hardwarekomponenten durch Teilreleases erzwungen werden, was eine Erweiterung der möglichen Rekonfigurationsszenarien gegenüber den Einzelabhängigkeiten bedeutet. In diesem Zusammenhang ist eine weitere Veränderung der Bedeutung der Austauschbarkeitsbeziehung und damit eine Erweiterung des Algorithmus der passiven Rekonfigurationsphase notwendig. Zusätzlich muss dazu eine Austauschbarkeitsbeziehung zwischen solchen Hardwarekomponenten dokumentiert werden, die für einen Austausch in Frage kommen sollen.
Ein Teilrelease wird als Menge $\mathbf{TR_x}$ der Form

$$\mathbf{TR_x} = \{sw_1, \ldots, sw_n\}$$

dargestellt, wobei sw_1 bis sw_n die im Teilrelease enthaltenen Softwarekomponenten sind. Die Menge aller Teilreleases soll $\mathbf{TR_{gesamt}}$ heißen.
Obwohl nicht expliziter Bestandteil des Teilreleases, sollen *Steuergeräte* (wie in Abschnitt 4.3.1 definiert) als Teilnehmer eines Teilreleases bezeichnet werden. Ein Steuergerät ist dann Teilnehmer, wenn mindestens eine Softwarekomponente dieses Steuergeräts Teilnehmer des Teilreleases ist. Nur so ist eine Bündelung der gemeinsam im Teilrelease enthaltenen Varianten und deren Unterscheidung von anderen Komponenten bei der Gültigkeitsprüfung des Teilreleases (siehe Abschnitt 4.4.6) möglich. Dagegen darf nur eine Softwarekomponente einer Softwarekette an einem Teilrelease teilnehmen. Andersherum darf eine Softwarekomponente nur dann Teilnehmer mehrerer Teilreleases sein, wenn diese alternativ verbaut werden. Kommt eine Software in mehreren Teilreleases vor, die gleichzeitig in einem Fahrzeug verbaut werden können, müssen diese Teilreleases zu einem Teilrelease zusammengeführt werden.

Beispiel

Für das Beispiel sollen die folgenden Teilreleases dokumentiert sein (siehe auch Abb. 4.9):

$$\mathbf{TR_1} = \{\textit{SAM-SW 3}, \textit{KLA-SW 3}, \textit{KLA-SW 6}\}$$
$$\mathbf{TR_2} = \{\textit{SAM-SW 4}, \textit{KLA-SW 4}, \textit{KLA-SW 7}\}$$

Zur Vereinfachung wurden die dadurch möglichen Rekonfigurationsszenarien gegenüber

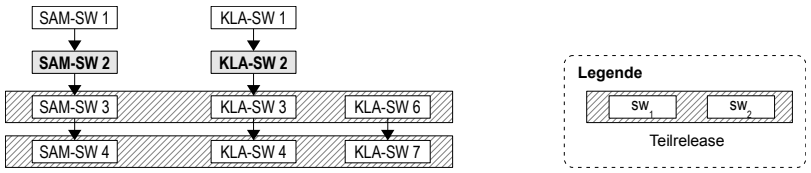


Abbildung 4.9: Teilreleases im Beispiel

der Dokumentation mittels Einzelabhängigkeiten eingeschränkt. Um den generischen Inhalt eines Teilreleases zu verdeutlichen, wurden für das KLA-Steuergerät noch die beiden Softwarekomponenten *KLA-SW 6* und *KLA-SW 7* als Teilnehmer dokumentiert[6]. Damit sind die beiden Steuergeräte *KLA-SG* und *SAM-SG* nach der Definition oben Teilnehmer des Teilreleases.

4.4.6 Umsetzung der aktiven Rekonfigurationsphase mit Teilreleases

Allgemein

Wie auch bei den Einzelabhängigkeiten werden die dokumentierten Teilreleases zunächst ausgehend von der zuvor ermittelten Softwarekomponente ausgewertet. Auf Grund der zusätzlich möglichen Rekonfigurationsszenarien werden aber weitere, alternative Softwarekomponenten ermittelt. Die dazu notwendigen Schritte werden im Folgenden beschrieben sowie in Abb. 4.10 dargestellt.

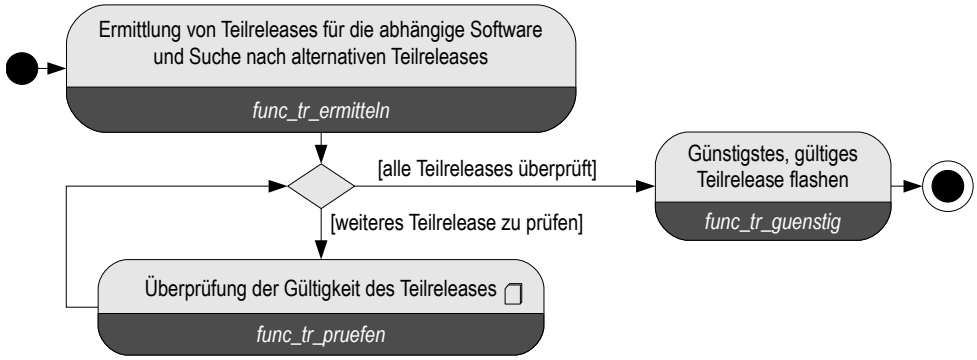


Abbildung 4.10: Auswertung der Teilreleases

func_tr_ermitteln Die Funktion zur Ermittlung aller Teilreleases für die abhängige Software und zur Suche nach alternativen Teilreleases wendet drei unterschiedliche Strategien bei ihrer Suche an. Alle so ermittelten Teilreleases **TR** aus der Menge

[6] Diese Abhängigkeit wurde im Beispiel zu den Einzelabhängigkeiten zur Vereinfachung nicht dokumentiert

$\mathbf{TR_{pruefen}}$ werden anschließend in der Funktion $func\_tr\_pruefen$ auf Gültigkeit überprüft. Dabei wird die Information über die Reihenfolge der Teilreleases in der Menge $\mathbf{TR_{pruefen}}$ gespeichert, um diese später als Auswahlkriterium in der Funktion $func\_tr\_guenstig$ verwenden zu können. Die Reihenfolge entspricht der Reihenfolge der Abarbeitung der Strategien 1 bis 3.

Mittels der Strategie 1 werden alle Teilreleases für die zuvor in der passiven Rekonfigurationsphase ermittelten, abhängigen Software sw bestimmt:

$$\begin{aligned} \mathbf{TR_{pruefen}} \quad := \quad & \text{func\_tr\_ermitteln}(sw) = \{\mathbf{TR_1}, \ldots, \mathbf{TR_n}\} \\ & \text{mit } sw \in \mathbf{TR_i},\ 1 \leq i \leq n,\ n = |\mathbf{TR_{gesamt}}| \end{aligned}$$

In der Strategie 2 werden alternative Teilreleases entlang der Vorgängerkette **KETTE_V** der betroffenen Software sw gesucht. Dabei wird für jede Software entlang dieser Kette, beginnend mit der jüngsten, einzeln rückwärts nach Teilreleases gesucht.

In der Strategie 3 wird nach solchen Teilreleases gesucht, die durch den Tausch einer Hardware gültig werden könnten. Dazu wird die Austauschbarkeitsbeziehung ausgehend von der aktuell verbauten Hardware ausgewertet und es werden die dann zur neueren Hardware kompatiblen Softwarekomponenten entsprechend der Strategien 1 und 2 zur Suche nach Teilreleases verwendet.

func_tr_pruefen Die Funktion zur Überprüfung der Gültigkeit eines Teilreleases ist in Abb. 4.11 dargestellt und wird weiter unten näher beschrieben. Das Resultat dieser Funktion ist die Menge aller gültigen Teilreleases.

func_tr_guenstig Die Funktion zur Ermittlung des günstigsten, gültigen Teilreleases bestimmt anhand der Kriterien

- Anzahl der benötigten Hardwaretausche
- Reihenfolge der gültigen Teilreleases

das zu flashende Teilrelease. Es wird das Teilrelease mit der geringsten Anzahl an Hardwaretauschen geflasht. Existieren mehrere Teilreleases mit einer minimalen Anzahl an Hardwaretauschen, wird das erste Teilrelease in der Reihenfolge, wie in $func\_tr\_ermitteln$ festgelegt, geflasht. Dieses entspricht in der Regel dem neusten Teilrelease.

Im Folgenden sind die einzelnen Funktionsschritte der Funktion $func\_tr\_pruefen$ näher erläutert:

func_tr_sg_ermitteln In dieser Funktion wird ein Steuergerät sg aus der Menge der Steuergeräte im Teilrelease $\mathbf{TR_x}$ ausgewählt. Dabei werden nur solche Steuergeräte berücksichtigt, die zuvor noch nicht innerhalb dieses Teilreleases überprüft wurden. Wurden alle Steuergeräte überprüft, wird das aktuell geprüfte Teilrelease

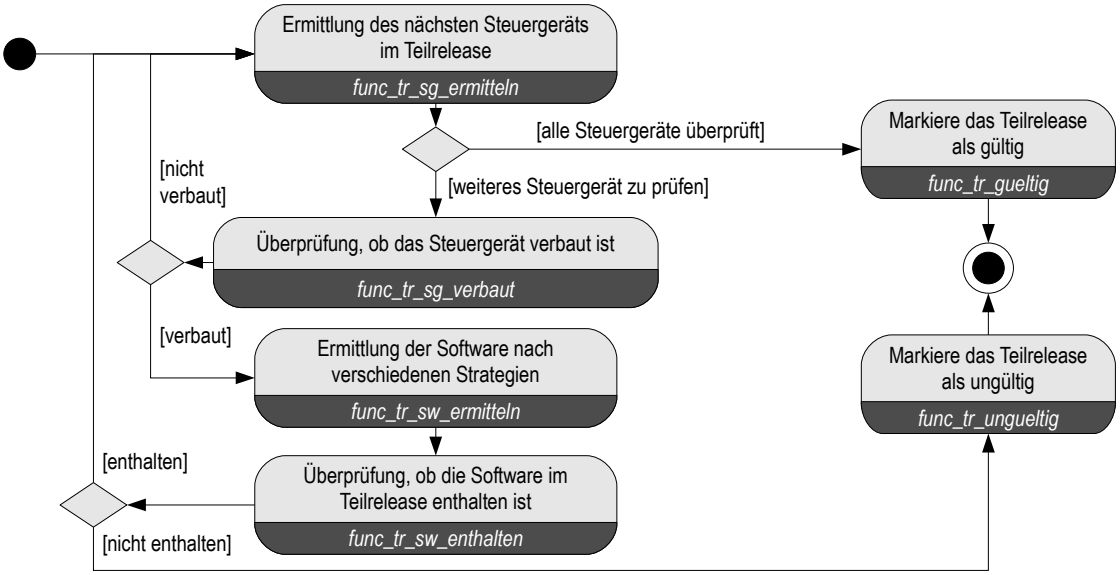


Abbildung 4.11: Überprüfung der Gültigkeit eines Teilreleases

in der Funktion $func\_tr\_gueltig$ als gültig markiert. Ist ein weiteres Steuergerät zu prüfen, wird dieses der Funktion der Funktion $func\_tr\_sg\_verbaut$ übergeben.

func_tr_sg_verbaut Die Funktion zur Überprüfung, ob das untersuchte Steuergerät im Fahrzeug verbaut ist, ist wie folgt definiert:

$$bool \quad := \quad \text{func\_eah\_sw\_benoetigt}(sg) = \begin{cases} \text{verbaut:} & sg \in \mathbf{P} \\ \text{nicht verbaut:} & \text{sonst} \end{cases}$$

Wird ein Steuergerät als nicht verbaut erkannt, ist es für die Gültigkeitsprüfung irrelevant.

func_tr_sw_ermitteln Die Funktion zur Ermittlung der Steuergerätesoftware wendet dieselben, oben in der Funktion $func\_tr\_ermitteln$ bereits beschriebenen Strategien an, um eine Software zu finden, die im aktuell untersuchten Teilrelease enthalten und damit gültig sein könnte. Die oben genannten Strategien 1 bis 3 werden zusätzlich um die Strategie 0 erweitert, die die bereits für das aktuell untersuchte Steuergerät im Fahrzeug enthaltene Software ermittelt. Alle so gefundenen Softwaren sollen als $\mathbf{SW_{pruefen}}$ bezeichnet werden.

func_tr_sw_enthalten Die Funktion zur Überprüfung, ob eine der zuvor ermittelten Softwarekomponenten im Teilrelease enthalten ist, ist wie folgt definiert:

$$bool \quad := \quad \text{func\_tr\_sw\_enthalten}(\mathbf{SW_{pruefen}}) = \begin{cases} \text{enthalten:} & sw_i \in \mathbf{TR} \\ \text{nicht enthalten:} & \text{sonst} \end{cases}$$
$$\text{mit } sw \in \mathbf{SW_{pruefen}}, 1 \leq i \leq n,\ n = |\mathbf{SW_{pruefen}}|$$

Wurden im vorherigen Schritt *func_tr_sw_ermitteln* mehrere Softwaremodule für das Steuergerät ermittelt, müssen alle Module vollständig im Teilrelease enthalten sein. Wurde eine Softwarekomponente oder mehrere Softwaremodule für das Steuergerät im Teilrelease gefunden, müssen diese auf Grund des generischen Aufbaus des Teilreleases als gültig markiert werden (nicht in Abb. 4.11 dargestellt), da nur so beim Flashen des Teilreleases die für das konkrete Fahrzeug gültigen Softwarekomponenten verwendet werden können. Alle sonstigen im Teilrelease enthaltenen Varianten werden dann ignoriert.

func_tr_gueltig, func_tr_ungueltig In diesen beiden Funktionen wird das gerade geprüfte Teilrelease entweder als gültig oder ungültig markiert. Diese Information wird später im Schritt *func_tr_guenstig* (siehe Abb. 4.10) ausgewertet, um das tatsächlich zu flashende Teilrelease auszuwählen.

Beispiel

Im Beispiel resultieren mit diesem Algorithmus die in Tab. 4.4 dargestellten Ergebnisse für die Funktion *func_tr_ermitteln*. Die Ergebnisse der einzelnen Funktionsschritte aus

Funktionsschritt	**Ergebnis**
Ermittlung von Teilreleases für die abhängige Software und Suche nach alternativen Teilreleases (*func_tr_ermitteln*)	*sw* nach Strat. 1 = {*KLA-SW 4*} *sw* nach Strat. 2 = {*KLA-SW 3*} *sw* nach Strat. 3 = {} $\mathbf{TR_{pruefen}} = \{\mathbf{TR_1}, \mathbf{TR_2}\}$

Tabelle 4.4: Beispielergebnisse für die aktive Rekonfigurationsphase mit Teilreleases

func_tr_pruefen sind in Tab. 4.5 dargestellt. Hierbei wird nur auf die Überprüfung des Teilreleases $\mathbf{TR_2}$ im Detail eingegangen. Die Gültigkeitsprüfung für $\mathbf{TR_1}$ wird im Anschluss daran durchgeführt und ist ebenfalls positiv. Durch die vorgegebene Reihenfolge

1. $\mathbf{TR_2}$
2. $\mathbf{TR_1}$

und die gleiche minimale Anzahl benötigter Hardwaretauschen von 0 wird $\mathbf{TR_2}$ in der Funktion *func_tr_guenstig* als günstigstes Teilrelease erkannt und geflasht. Das Produkt sieht damit nach der Rekonfiguration wie folgt aus:

$$\mathbf{P} = \{3kla, usa, 6zylinder, irs, \textit{KLA-HW 1}, \textit{KLA-SW 4}, \textit{SAM-HW 1}, \textit{SAM-SW 4}, \ldots\}$$

Funktionsschritt	**Ergebnis**
Ermittlung des nächsten Steuergeräts im Teilrelease ($func\_tr\_sg\_ermitteln$)	$sg = KLA\text{-}SG$
Überprüfung, ob das Steuergerät verbaut ist ($func\_tr\_sg\_verbaut$)	$KLA\text{-}SG \in \{\ldots, KLA\text{-}SG, KLA\text{-}HW\ 1\} \rightarrow$ verbaut
Ermittlung der Software nach verschiedenen Strategien ($func\_tr\_sw\_ermitteln$)	$\mathbf{SW_{pruefen}} = \{KLA\text{-}SW\ 3, KLA\text{-}SW\ 4\}$
Überprüfung, ob die Software im Teilrelease enthalten ist ($func\_tr\_sw\_enthalten$)	$KLA\text{-}SW\ 4 \in \{SAM\text{-}SW\ 4, KLA\text{-}SW\ 4, KLA\text{-}SW\ 7\} \rightarrow$ enthalten
Ermittlung des nächsten Steuergeräts im Teilrelease ($func\_tr\_sg\_ermitteln$)	alle Steuergeräte überprüft
Markiere das Teilrelease als gültig ($func\_tr\_gueltig$)	$\mathbf{TR_2}$ ist gültig

Tabelle 4.5: Beispielergebnisse für die aktive Rekonfigurationsphase mit Teilreleases

5 Diskussion der Lösung

Im vorigen Abschnitt wurden zwei Lösungen für die Dokumentation und Auswertung von Abhängigkeiten zwischen Softwarekomponenten dargestellt. Im Folgenden werden nun beide Methoden bewertet und die offenen Probleme aufgezeigt. Weiterhin werden die verwendeten Beziehungstypen mit bestehenden Definitionen verglichen und Aussagen zu deren Verwendbarkeit im allgemeinen Kontext getroffen.

5.1 Bewertung der Dokumentationsmethodik

Die Dokumentationsmethodik wird nach den im Folgenden aufgeführten Kriterien bewertet. Zum besseren Verständnis sind einige Kriterien untergliedert. Dabei hat ein Kriterium der tieferen Strukturebene Einfluss auf das jeweils übergeordnete Kriterium. Wiederholen sich die Einflussgrößen, werden sie nur einmal beschrieben. An den sonstigen Bezugsstellen wird auf diese Beschreibungen verwiesen.

1. **Kundenzufriedenheit**
   1.1. **Steuerbarkeit von Rekonfigurationsszenarien** *Die Unterkriterien hierfür sind in Punkt 5.1. unten angegeben.*
2. **Wirtschaftlichkeit (Kundensicht)**
   2.1. **Steuerbarkeit von Rekonfigurationsszenarien** *Die Unterkriterien hierfür sind in Punkt 5.1. unten angegeben.*
3. **Wirtschaftlichkeit (Unternehmenssicht)**
   3.1. **Dokumentationsaufwand**
      3.1.1. **Abstraktionen** Für die Einzelabhängigkeit wurde eine *mindestens-Beziehung* verwendet, um mehrere Teilnehmer einer Softwarekette durch eine Beziehung zu referenzieren. Eine weitere Abstraktion über mehrere Varianten hinweg (Steuergeräteebene) ist im vorgestellten Konzept nicht vorgesehen, könnte aber als Eingabeunterstützung mit einfachen Mitteln bereitgestellt werden.

      Für die Teilreleases wurde als Abstraktion eine variantenübergreifende Dokumentation eingeführt, so dass mehrere Varianten derselben Software eines Steuergeräts im selben Teilrelease enthalten sein können. Je zusätzlicher Variante im Teilrelease wird die Dokumentation eines eigenen Teilreleases eingespart.

3.2. **Entwicklungskosten**

3.2.1. **Reduktion der Rekonfigurationsszenarien** *Siehe dazu den Aggregationsgrad in Punkt 5.1.3.* Steigt der Aggregationsgrad, sinkt die Anzahl der Rekonfigurationsszenarien.

4. **Korrektheit**

4.1. **Syntaktische Korrektheit**

4.1.1. **Widerspruchsfreiheit** Bei den Einzelabhängigkeiten werden unerfüllbare Abhängigkeiten als ungültig betrachtet und es existiert keine Ausschlussbeziehung, die eine Einzelabhängigkeit ungültig werden lässt und damit einen Widerspruch erzeugt. Wurden mehrere Einzelabhängigkeitsbeziehungen ausgehend von einer Softwarekomponente dokumentiert, die an unterschiedlichen Zielsoftwarekomponenten derselben Ersatzkette enden, wird dabei nur die Beziehung auf die neuste Softwarekomponente als gültig betrachtet und somit ein möglicher Widerspruch ausgeschlossen. Für Teilreleases werden mögliche Widersprüche bei der Ermittlung mehrerer Ergebnis-Teilreleases durch eine Festlegung ihrer Reihenfolge ausgeschlossen. Anhand der Reihenfolge der Ermittlung der Softwarekomponenten und der Anzahl der benötigten Hardwaretausche kann immer ein eindeutiges Ergebnis erzielt werden. Weiterhin gilt die Festlegung, dass dieselbe Softwarekomponente nur dann in verschiedenen Teilreleases vorkommen darf, wenn diese nur alternativ, also nicht im selben Fahrzeug verbaubar sind. Hierdurch wird ebenfalls die Eindeutigkeit des Ergebnisses sichergestellt.

Insgesamt wird bei der Ermittlung der Softwarekomponente eine monotone Einschränkung der möglichen Rekonfigurationsszenarien vorgenommen, so dass Widersprüche ausgeschlossen sind (siehe Abb. 5.1).

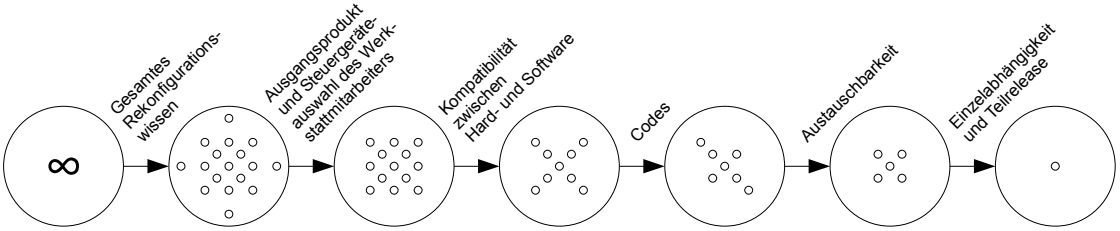


Abbildung 5.1: Monotone Einschränkung des Lösungsraums

4.2. **Semantische Korrektheit** Die semantische Korrektheit kann nicht direkt durch die Dokumentationsmethoden unterstützt werden. Eine Unterstützung in diesem Kriterium ist nur durch eine hohe Intuitivität der Methoden möglich, da so das Verständnis für die Methode und damit die inhaltlich korrekte

Dokumentation vereinfacht wird. Siehe auch Intuitivität unten sowie die Dokumentationsunterstützung in Abschnitt 6.

5. **Lösungsumfang**

5.1. **Steuerbarkeit von Rekonfigurationsszenarien**

5.1.1. **Einstiegsmöglichkeiten in die Rekonfiguration** Der Einstieg in die Rekonfiguration ist bei beiden Methoden nur über den Auftrag zur Ermittlung einer Software für ein vorgegebenes Steuergerät möglich. Eine explizite Vorgabe einer Zielsoftware oder -konfiguration oder eines Zielproduktmerkmals wird nicht unterstützt. Eine solche Erweiterung z.B. zur Vorgabe eines gewünschten Produktmerkmals erfordert in praktischer Hinsicht eine umfangreiche Anpassung der bestehenden Lösung[1].

5.1.2. **Einflussgrößen zur Steuerung der Rekonfiguration** Für beide Lösungen wird das Ausgangsprodukt und das ausgewählte Steuergerät als Eingangsparameter für die Rekonfigurationsfunktion berücksichtigt. Innerhalb des Ausgangsprodukts werden die Steuergerätehardware, die auf dem Steuergerät existierende Software und alle sonstigen in der Fahrzeugdatenbank hinterlegten Produktmerkmale betrachtet. Im Fall der Teilreleases werden weiterhin die verbauten Hardwarekomponenten aller am Teilrelease teilnehmenden Steuergeräte in die Softwareermittlung einbezogen. Bei den Einzelabhängigkeiten wird die Steuerbarkeit der Szenarien dadurch erhöht, dass neben der Gültigkeit der betroffenen Softwarekomponenten im Fahrzeug eine eigene Gültigkeitsbedingung für die Beziehung angegeben werden kann.

Eine explizite Vorgabe von Rekonfigurationsinvarianten durch den Kunden ist nicht vorgesehen. Dieser kann jedoch anhand des erstellten Rekonfigurationskonzepts entscheiden, ob er dieses akzeptiert oder nicht.

5.1.3. **Aggregationsgrad** Der Aggregationsgrad kann durch beide Lösungen nach Bedarf erhöht und dadurch die Anzahl der Rekonfigurationsszenarien verringert werden. Durch die Angabe von Abhängigkeiten können bisher unterschiedliche Fahrzeuge so rekonfiguriert werden, dass sie nach der Rekonfiguration identisch sind. Zudem können so Inkonsistenzen während der Rekonfiguration eingeschränkt werden, da funktional eng interagierende Komponenten gemeinsam verändert werden.

5.1.4. **Kompositionsgrad** Der Kompositionsgrad wird durch keine der beiden Lösungen verändert, da sowohl Einzelabhängigkeiten als auch Teilreleases

[1] Eine Abhängigkeit zwischen dem hinzufügbaren Produktmerkmal und allen anzupassenden Softwarekomponenten usw. müsste dann dokumentiert werden können. Gleichzeitig muss auch die Unternehmensstrategie berücksichtigt bleiben, d.h. das Ausgangsprodukt muss weiterhin als Kriterium in die Ermittlung des Rekonfigurationskonzepts eingehen.

nur zur Rekonfiguration der bestehenden Komponenten eingesetzt werden, diese aber nicht in feingranularere Komponenten zerlegen und damit steuerbar machen. Ebenso werden alle Variantenausprägungen unterhalb der Softwarekomponente durch die Variantencodierung oder die Konfiguration zur Laufzeit als kompatibel bezüglich der auf sie zeigenden Abhängigkeitsbeziehung betrachtet.

5.2. **Automatisierungsgrad** Für beide Lösungen wird die Diagnose durch den Werkstattmitarbeiter erstellt und anschließend eine Steuergeräteauswahl durch ihn getroffen. Ab diesem Zeitpunkt findet die Ermittlung des Rekonfigurationskonzepts vollständig automatisiert statt. Im Fall der Teilreleases ist je nach konkreter Implementierung noch eine manuelle Auswahl des gewünschten Teilreleases durch den Werkstattmitarbeiter notwendig. Hierbei wird er in der Regel die notwendige zu tauschende Hardware als Kriterium ansetzen.

5.3. **Dokumentierbare Komponenten** Beide Lösungen bieten ausschließlich die Möglichkeit, Abhängigkeiten zwischen Softwarekomponenten zu dokumentieren.

5.4. **Berücksichtigung praktischer Probleme**

5.4.1. **Keine Software im Steuergerät** Befindet sich zu Beginn der Softwareermittlung keine Softwarekomponente in der verbauten Hardware, ist das Ausgangsprodukt nicht vollständig und es muss basierend auf anderen Produktmerkmalen auf das ursprüngliche Ausgangsprodukt geschlossen werden können. Dieses Problem muss in der Basislösung gelöst werden und ist nicht Bestandteil des Einzelabhängigkeits- oder Teilreleasekonzepts.

5.4.2. **Integration in externe Lösungen** Zur Integration in externe Lösungen zur Ermittlung von Ersatzteilen (z.B. Teilekatalog) existiert im vorgestellten Konzept keine explizite Unterstützung, da sich beide Lösungen auf die Ermittlung von Software beschränken. Externe Lösungen, die ebenfalls die Rekonfiguration unterstützten (z.B. Tausch einer mechanischen Komponente), müssen ähnliche Abhängigkeitsdokumentationen bereitstellen oder die vorgestellten Lösungen auf die in diesem System verwalteten Komponenten übertragen. Eine solche Integration ist unverzichtbar, da sonst inkonsistente Fahrzeugzustände durch unqualifizierte Tauschvorgänge entstehen können.

6. **Hohe Anzahl unterschiedlicher Rekonfigurationsszenarien** Eine hohe Anzahl unterschiedlicher Rekonfigurationsszenarien wird durch beide Lösungen durch die flexible Steuerbarkeit verschiedener Rekonfigurationsszenarien unterstützt. Gleichzeitig ist der Dokumentationsaufwand durch Abstraktionen verringert und die Ermittlung des Rekonfigurationskonzepts automatisiert.

7. **Dokumentationsgrad** Beide Lösungen erlauben, dass die Auswirkungen der Soft-

wareermittlung bei der Anzeige des Rekonfigurationskonzepts berücksichtigt werden. Diese Anforderung muss in der Implementierung des Konzepts berücksichtigt werden. Bezüglich der feineren Dokumentation von veränderlichen Produktmerkmalen unterhalb einer Softwarkomponente leisten beide Lösungen wie bereits erwähnt keinen Beitrag.

8. **Beeinflussung der Basislösung** Grundsätzlich wird der bestehende Rekonfigurationsalgorithmus (wie in Abschnitt 4.4.2 beschrieben) weitestgehend wiederverwendet und die vorgestellten Lösungen als zusätzliche Ebene über ihn gelegt. Das führt dazu, dass nun mehrere abhängige Softwarekomponenten gleichzeitig ermittelt werden können. Darüber hinaus wird die Austauschbarkeitsbeziehung durch die Teilreleaselösung in ihrer Bedeutung verändert. Softwarekomponenten können nun auch dann Ziel einer Rekonfiguration sein, wenn sie nicht das Ende einer Softwarekette darstellen, aber dennoch Teilnehmer eines Teilreleases sind.

9. **Intuitivität** Die Einzelabhängigkeitsbeziehung entspricht weitgehend dem bekannten Beziehungstyp *requires*, was zu einem höheren Verständnis dieser Beziehung beim Dokumentierenden führt. Beeinträchtigend für die Intuitivität ist ihre stark kontextabhängige Gültigkeit. Zum Beispiel werden keine Hardwaretausche durch die Einzelabhängigkeit erzwungen, da diese bei unpassender Hardware als nicht gültig definiert ist.

   Die Anwendung von Teilreleases ist komplexer und die Intuitivität bei der Verwendung daher geringer als bei der Einzelabhängigkeit. Die Gründe dafür sind die Möglichkeit zur Bildung von variantenübergreifenden Teilreleases sowie die Einbeziehung von Softwarekomponenten, die nicht das Ende einer Softwarekette darstellen.

10. **Wartbarkeit** Die Wartbarkeit wird in Abschnitt 5.1.1 untersucht.

11. **Wissenstiefe** Die Wissenstiefe wird in Abschnitt 5.2 untersucht.

5.1.1 Dokumentationsszenarien

In diesem Abschnitt sollen ausgewählte Szenarien im Entwicklungs- und Dokumentationsprozess gezeigt werden, die die Anwendung von Einzelabhängigkeiten und Teilreleases verdeutlichen. Anhand der Szenarien wird gleichzeitig die Wartbarkeit der Wissensbasis diskutiert.

Entstehung einer neuen Softwarekomponente (1)

In der Ausgangssituation existieren vier Softwarekomponenten *SW A1, SW A2, SW B1* und *SW B2* zwischen denen die beiden Austauschbarkeiten $aus_1 = (SW\ A1, SW\ A2)$ und $aus_2 = (SW\ B1, SW\ B2)$ dokumentiert sind. Zwischen *SW A2* und *SW B2* ist die Einzelabhängigkeit $eah_1 = (SW\ A2, SW\ B2, -)$ ohne Codebedingung angegeben (Abb.

5.2, links). Wird nun eine neue Softwarekomponente *SW A3* entwickelt und durch $aus_3 =$

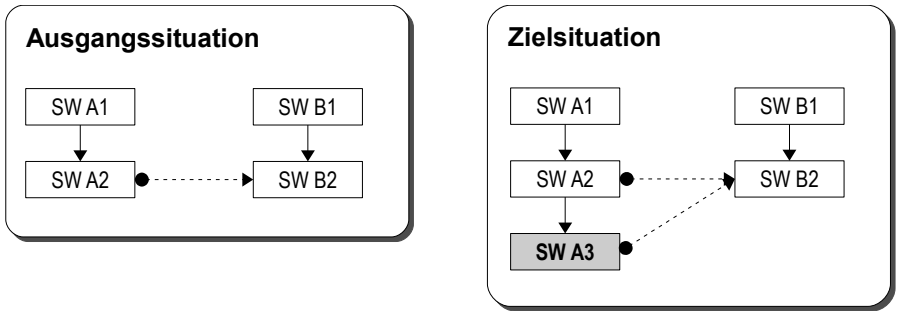


Abbildung 5.2: Entstehung einer neuen Softwarekomponente innerhalb der Softwarekette der abhängigen Software

(*SW A2*, *SW A3*) als austauschbar zu *SW A2* dokumentiert, muss geprüft werden, ob die Einzelabhängigkeit zu *SW B2* weiterhin besteht. Ist diese weiterhin nötig, muss sie entsprechend durch $eah_2 = (SW\ A3, SW\ B2, -)$ angegeben werden (Abb. 5.2, rechts).

Entstehung einer neuen Softwarekomponente (2)

Bei gleicher Ausgangsituation wie oben in Abb. 5.2 kann ebenfalls eine neue Softwarekomponente *SW B3* innerhalb der Softwarekette der benötigten Software entwickelt werden. Abhängig davon, ob dieser Nachfolger durch $aus_3 = (SW\ B2, SW\ B3)$ als austauschbar oder nicht als austauschbar angegeben wird, muss die Dokumentation einer Einzelabhängigkeit $eah_2 = (SW\ A2, SW\ B3, -)$ geprüft werden. Sie wird nur dann nötig, wenn *SW B3* nicht austauschbar zu *SW B2* ist, die Abhängigkeit aber weiterhin vorhanden ist (Abb. 5.3, rechts). Ist *SW B3* dagegen austauschbar zu *SW B2*, muss keine Dokumentation einer zusätzlichen Einzelabhängigkeitsbeziehung vorgenommen werden (Abb. 5.3, links). Dann wird diese über die Austauschbarkeitsbeziehung an den Nachfolger *SW B3* vererbt.

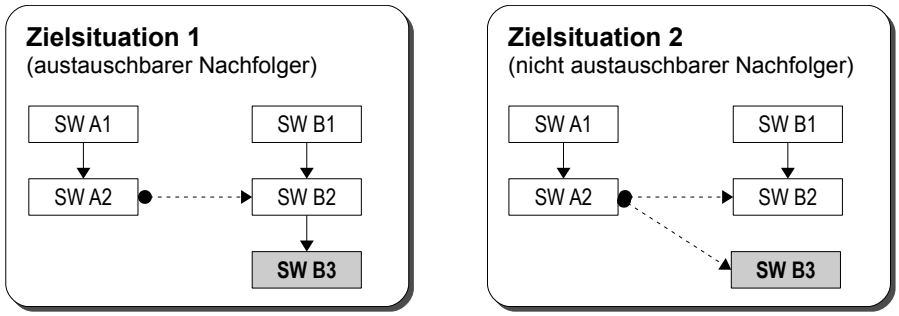


Abbildung 5.3: Entstehung einer neuen Softwarekomponente innerhalb der Softwarekette der benötigten Software

Entstehung einer neuen Softwarekomponente (3)

In der Ausgangssituation existieren acht Softwarekomponenten *SW A1, SW A2, SW B1, SW B2, SW C1, SW C2, SW D1* und *SW D2* zwischen denen die vier Austauschbarkeiten aus_1 = (*SW A1*, *SW A2*), aus_2 = (*SW B1*, *SW B2*), aus_3 = (*SW C1*, *SW C2*) und aus_4 = (*SW D1*, *SW D2*) dokumentiert sind. Zusätzlich ist das Teilrelease $\mathbf{TR_1}$ = {*SW A2*, *SW B2*, *SW C2*, *SW D2*} angegeben (Abb. 5.4, oben). Werden nun die bei-

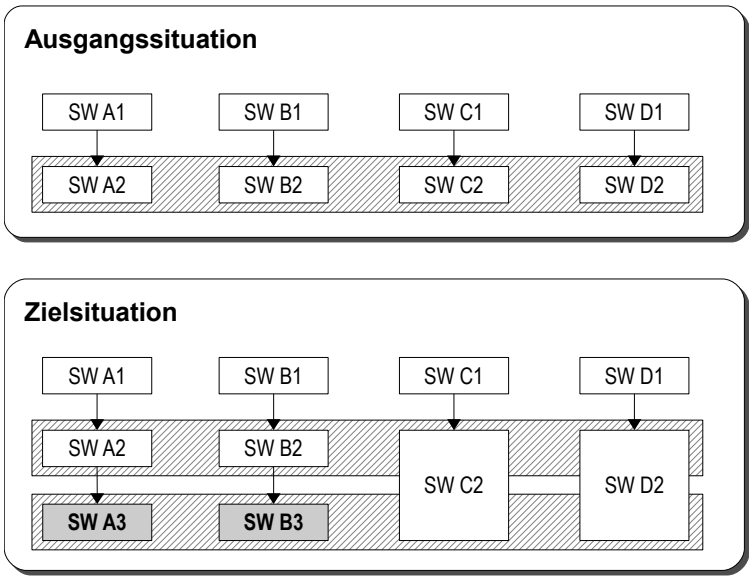


Abbildung 5.4: Entstehung von zwei neuen Softwarekomponenten bei einem Teilrelease

den neuen Softwarekomponenten *SW A3* und *SW B3* entwickelt und durch aus_5 = (*SW A2*, *SW A3*) und aus_6 = (*SW B2*, *SW B3*) als austauschbar zu ihren Vorgängern dokumentiert, muss geprüft werden, ob ein neues Teilrelease erstellt werden muss und welche Softwarekomponenten der anderen Steuergeräte darin enthalten sein müssen. Sollen auch *SW A3* und *SW B3* gemeinsam mit *SW C2* und *SW D2* geflasht werden, muss das Teilrelease $\mathbf{TR_2}$ = {*SW A3*, *SW B3*, *SW C2*, *SW D2*} angelegt werden (Abb. 5.4, unten). Eine Vererbung des Teilreleases auf austauschbare Nachfolger findet nicht statt.

Verwendung einer Einzelabhängigkeit bei vielen Varianten

In Abb. 5.5 wird eine Produktstruktur gezeigt, die zwei Steuergeräte A und B enthält, wobei Steuergerät B eine sehr hohe Variantenvielfalt aufweist. Beispielhaft sind dort die Softwarekomponenten der Variante B.1 dargestellt. Gleichzeitig existiert eine Einzelabhängigkeit eah_1 = (*SW A2*, *SW B.1.1.1.2*, −) zwischen *SW A2* und einer der zahlreichen Softwarevarianten von Steuergerät B. Durch die Art der Einzelabhängigkeitsbeziehung ist eine solche explizite Abhängigkeitsdokumentation möglich. Würde *SW A2* dagegen von allen Softwarevarianten des Steuergeräts B abhängig sein, müsste eine Einzelabhängig-

keit zu jeder dieser Softwarekomponenten dokumentiert werden. Eine Verringerung des Dokumentationsaufwands wäre dann durch die Verwendung eines Teilreleases möglich, das *SW A2* sowie alle Softwarevarianten von Steuergerät B enthält.

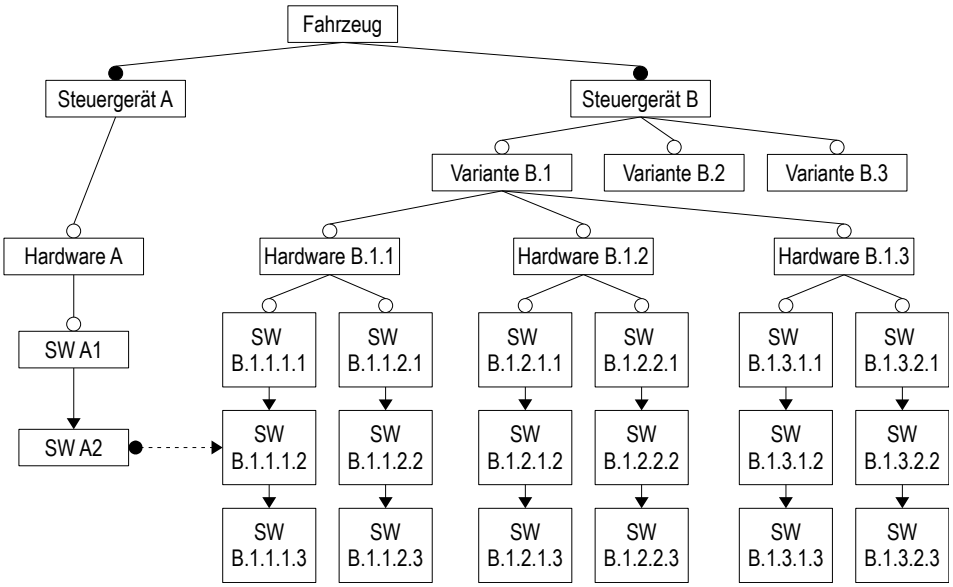


Abbildung 5.5: Verwendung einer Einzelabhängigkeit bei vielen Varianten

5.2 Wissenstiefe

Die Bedeutung der Beziehungen *Kompatibilität* und *Austauschbarkeit* sowie der *Abhängigkeit* als Grundlage der Einzelabhängigkeit und des Teilreleases sind bisher innerhalb der vorgestellten Lösung intuitiv klar geworden. Bei der Auswertung der Beziehungen wurde dabei davon ausgegangen, dass der Service alleiniger Nutzer dieses Wissens ist. Darüber hinaus verwenden und erzeugen jedoch auch die Produktions- und Entwicklungsbereiche Teile dieses Wissens.
Wichtig für alle Bereiche ist, den Dokumentationsaufwand zu minimieren und eine korrekte operative Verwendung der Beziehungen sicherzustellen. Zur Minimierung des Dokumentationsaufwands muss abgeschätzt werden können, wann eine Beziehung tatsächlich dokumentiert werden sollte und in welchem Fall sie unnötiges Wissen darstellt. Bei der Rekonfiguration ergibt sich zusätzlich die Anforderung, Wissen über das Verhalten von Komponenten innerhalb eines Produkts zu einem gegebenen Zeitpunkt so zu dokumentieren bzw. zu *konservieren*, dass darauf später bei der Diskussion neuer Rekonfigurationsszenarien mit neu entwickelten Komponenten zuverlässig zurückgegriffen werden kann. Im Folgenden werden die Bedeutungen dieser Beziehungen daher eingehender auf ihre *Wissenstiefe* hin untersucht. Es wird insbesondere darauf eingegangen, dass die Bedeutung jeder Beziehung abhängig vom Kontext ist, in dem sie verwendet wird.

5.2.1 Klassifikation von Konfigurationen

Um Ziel einer Produktion oder Rekonfiguration zu sein, muss ein Produkt vorher entwickelt und getestet worden sein. Dazu wurde(n) aus einer anfangs unendlich großen Menge möglicher Konfigurationen durch das Unternehmen ein oder mehrere Produkte verwirklicht, indem zunächst die Bestandteile der Konfiguration entwickelt und dann integriert wurden. Diese nun baubaren Konfigurationen stehen nach den entsprechenden Tests als abgesicherte Konfigurationen zur Verfügung und können vom Kunden bestellt werden oder Ziel einer Rekonfiguration sein. In diesem Fall wurde ein konkretes Produkt realisiert. Damit gibt es die vier Klassen

- theoretisch mögliche Konfiguration,
- baubare Konfiguration,
- abgesicherte Konfiguration und
- realisiertes Produkt.

Die Klasse der theoretisch möglichen Konfigurationen ist für das Rekonfigurieren dann interessant, wenn ein Rekonfigurationsziel durch die bereits bestehenden abgesicherten Konfigurationen nicht verwirklicht werden kann. Dann muss zunächst eine theoretisch mögliche Konfiguration durch eine Neuentwicklung in eine baubare und schließlich eine abgesicherte Konfiguration überführt werden. Existiert bereits eine passende baubare Konfiguration, muss diese nur noch abgesichert werden.

5.2.2 Konfigurations- und Realisierungswissen

Es existiert offenbar eine Hierarchie innerhalb des Wissens, das zur Realisierung von Produkten herangezogen wird und somit unterschiedlichen Wissensebenen zugeordnet werden kann. Es soll hier zwischen den folgenden Wissensebenen unterschieden werden:

1. Konfigurationswissen
2. Realisierungswissen
    a) Produktionswissen
    b) Rekonfigurationswissen

Das Konfigurationswissen (vgl. *configuration model knowlegde* in [STMS98]) beschreibt sämtliche baubaren und abgesicherten Konfigurationen, die anschließend unter Anwendung des Realisierungswissens in der Produktion produziert werden können (Produktionswissen) oder im Service Ziel einer Rekonfiguration sein können (Rekonfigurationswissen). Mit dem Realisierungswissen werden allgemein die Bedingungen beschrieben, unter denen Produkte basierend auf dem Konfigurationswissen realisiert werden können. Im Fall einer Rekonfiguration besteht diese Bedingung im Rekonfigurationsziel des Kunden,

in der Produktion z.B. aus den im Kundenauftrag enthaltenen Produktmerkmalen. Es können also jeweils nur Fakten im Realisierungswissen berücksichtigt werden, die Teil des Realisierungsauftrags sind.

Die Fakten aller Wissensebenen werden im Schritt *Modellieren und Steuern von Produkten* durch Entwickler, Produktdatenmanager, Servicesteuerung und Produktionssteuerung erzeugt. Anschließend folgt der Schritt *Realisieren von Produkten* (Abb. 5.6).

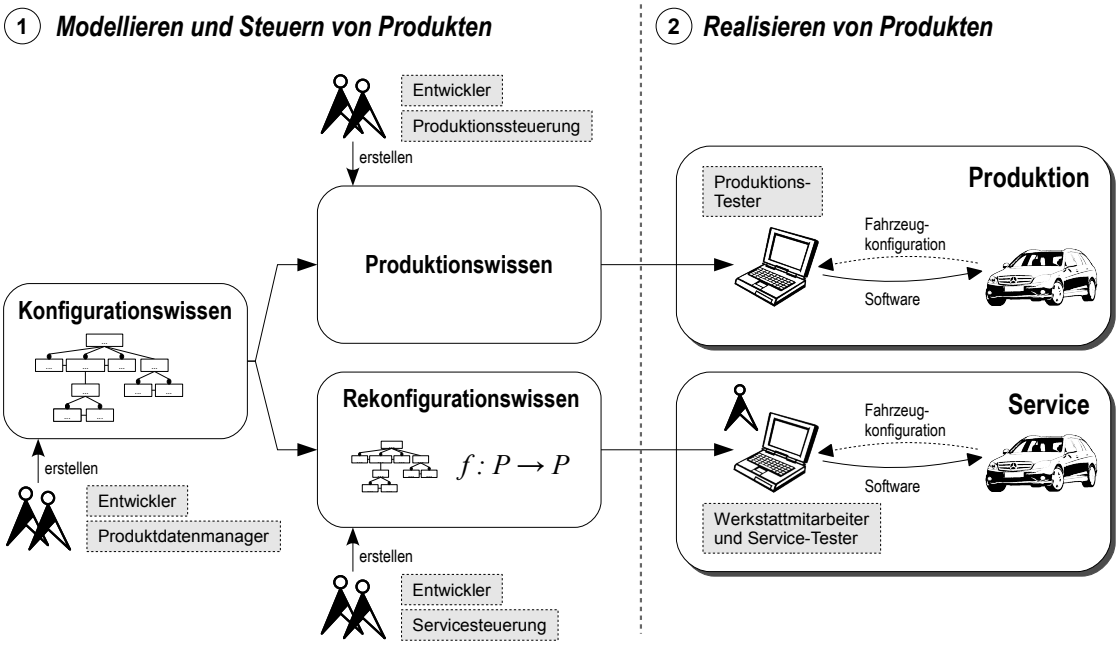


Abbildung 5.6: Konfigurations- und Realisierungswissen

Konservierung von Wissen

Die durch das Konfigurationswissen beschriebene Menge von baubaren und abgesicherten Konfigurationen kann über die Anzahl der Produkte hinausgehen, die tatsächlich im Service oder der Produktion realisiert werden. Die Menge der im Realisierungswissen verwendeten Konfigurationen ist damit immer eine Untermenge dieser Konfigurationen. Ist die Menge baubarer und abgesicherter Konfigurationen im Konfigurationswissen tatsächlich höher als im Realisierungswissen, kann dieses Wissen über diese zusätzlichen Konfigurationen verwendet werden, sobald neue Realisierungsszenarien dokumentiert werden sollen. Das ist insbesondere dann sinnvoll, wenn das Wissen über diese zusätzliche Konfiguration zum aktuellen Zeitpunkt nicht mehr oder nur noch mit sehr hohem Aufwand erzeugt werden kann, z.B. die nachträgliche Überprüfung der Kompatibilität zwischen zwei Komponenten.

Wissensnutzer

Die Bereiche Entwicklung, Produktion und Service sind damit als Nutzer des Konfigurations- und Realisierungswissens zu bezeichnen. Da jeder Nutzer andere Produkte in seinen Realisierungsszenarien anstrebt, muss bei der Dokumentation von Beziehungen zwischen Produktmerkmalen bekannt sein, welches Produkt dadurch entstehen kann und in welchem Kontext es verwendet werden soll.

Werden mehrere Produktmerkmale in einem Produktions- oder Rekonfigurationsprozess miteinander kombiniert, entsteht ein Produkt, das wiederum Produktmerkmale aufweist. In der Regel entstehen dabei vollständig neue Produktmerkmale, die erst durch das Zusammensetzen der ursprünglichen Produktmerkmale realisiert werden konnten. In Abschnitt 3.2 wurde dazu bereits die Bezeichnung der abstrakten und konkreten Produktmerkmale eingeführt. Anhand dieser neuen Eigenschaften kann bewertet werden, welchen Nutzen das Produkt hat und wie es in ein bestehendes Produkt integriert werden kann, also wie es erneut mit weiteren Produktmerkmalen zusammengesetzt werden kann und welche Produktmerkmale dabei entstehen. So wird z.B. durch die Zusammensetzung einer Steuergerätehardware mit einer Softwarekomponente eine bestimmte Funktionalität realisiert. Hard- und Software sind dabei konkrete Produktmerkmale, die eine abstrakte Funktionalität liefern.

Statt der Dokumentation der entstehenden Produktmerkmale wurde in der vorgestellten Lösung eine schlussfolgernde (kompilierte) Aussage bezüglich der bekannten Nutzungsszenarien des entstehenden Produkts in Form einer Kompatibilitäts-, Inkompatibilitäts-, Austauschbarkeits- oder Abhängigkeitsaussage getroffen.

Parallele Dokumentation

Abhängig davon, wie das Konfigurations- und Realisierungswissen dokumentiert wird, kann mittels derselben Dokumentation gleichzeitig ein neues baubares, abgesichertes Produkt und ein neues Realisierungsszenario beschrieben werden. Konfigurations- und Realisierungswissen können also im selben Modell abgebildet werden. Vor allem die Austauschbarkeit zwischen zwei Produktmerkmalen definiert zwei in ihren Produktmerkmalen sehr ähnliche Produkte inklusive einer Weiterentwicklungsrichtung, so dass bei ihrer Nutzung zur Rekonfiguration ein Produkt abgeleitet wird, dass dem alten noch ähnlich genug ist, um weiterhin dem Kundenwunsch zu entsprechen, aber dennoch das Rekonfigurationsziel (z.B. Reparatur) erfüllt. Wird das Realisierungsszenario nicht mehr unterstützt und die entsprechende Dokumentation gelöscht, kann das entsprechende Konfigurationswissen dennoch erhalten oder ebenfalls entfernt werden, da die damit bereitgestellten Konfigurationen technisch doch nicht realisierbar sind.

5.2.3 Kompatibilitätsbeziehung

Die Kompatibilitätsbeziehung wird in dieser Arbeit im Sinne der *Verträglichkeit* zweier Komponenten verstanden[2]. Allgemein sind zwei Komponenten dann kompatibel, wenn sie gemeinsam zu einem Produkt zusammengefügt werden können und dabei eine vorgegebene Spezifikation oder Kundenanforderung erfüllen. Sie sind kompatibel bezüglich dieser Spezifikation (siehe auch [BBDS06]). Andersherum bedeutet dies, dass mindestens ein Produkt existiert, in dem beide Produktmerkmale vorkommen.
In der oben beschriebenen Lösung wird die Kompatibilitätsbeziehung zwischen Hard- und Softwarekomponenten eingesetzt und dabei binär dokumentiert. Sie wird nicht zwischen Produktmerkmalen dokumentiert, bei denen die Kompatibilität bereits aus der Produktstruktur implizit hervorgeht und die während einer Rekonfiguration nicht durch eine Rekonfigurationsoperation verändert werden sollen. Eine explizite Überprüfung der Kompatibilität im Rekonfigurationsalgorithmus ist dann nicht notwendig, da davon ausgegangen werden kann, dass keine Kompatibilität innerhalb des Produkts verletzt wird. Z.B. ist eine Dokumentation der Kompatibilität zwischen einer Steuergerätesoftware und einem durch diese Software angesteuerten Aktor überflüssig, wenn der Aktor in keinem Realisierungsalgorithmus passend zur verbauten Software ausgewählt werden muss.
Werden eine Steuergerätehardware und eine Softwarekomponente als kompatibel zueinander dokumentiert, bilden sie ein Produkt, das die Anforderungen eines Nutzers erfüllt. Da unterschiedliche Nutzer unterschiedliche Anforderungen haben, gilt die Kompatibilitätsbeziehung damit nicht uneingeschränkt. Um entsprechend der oben eingeführten Hierarchie von Konfigurationen eine baubare Kombination von Hard- und Software zu erhalten, reicht es aus, wenn die Software prozessorkonform kompiliert wurde und in den Speicher der Hardware passt. Diese Zusammensetzung ist aus Sicht einer Entwicklungsabteilung kompatibel und die entsprechende Kompatibilitätsbeziehung würde ins Konfigurationswissen übernommen werden. Anhand dieser Information könnte z.B. ein Prototypfahrzeug aufgebaut werden. Betrachtet man nun die Anforderungen des Services, kann anhand dieser Kompatibilitätsbeziehung nicht beurteilt werden, ob diese Hard- und Software auch in einem Produkt eingesetzt werden kann, das bei einer Rekonfiguration entsteht.
Da Kundenaufträge immer bezogen auf das Gesamtfahrzeug formuliert werden, die Kompatibilitätsbeziehung aber binär dokumentiert wird, muss eine Übertragung dieser Beziehung auf das Gesamtfahrzeug und den dazugehörigen Realisierungsauftrag stattfinden. Hierzu muss tieferes Wissen über die Produktmerkmale vorliegen, die entstehen, wenn diese Zusammensetzung aus Hard- und Software als Teilprodukt in ein Gesamtprodukt integriert wird. In der vorgestellten Lösung werden keine entstehenden Produktmerkmale, sondern nur die Kompatibilitätsbeziehung dokumentiert. Die Einschränkung auf die tatsächlich in der Produktion oder im Service angestrebten Realisierungsszenarien findet durch die Coderegeln der Hard- und Softwarekomponenten statt.

[2] Im Gegensatz dazu wird dieser Begriff an anderen Stellen auch im Sinne der Austauschbarkeit zwischen zwei Komponenten gebraucht, die hier aber explizit als Austauschbarkeit bezeichnet wird.

Im Beispiel in Abb. 5.7 ist eine Produktstruktur zu sehen, die zwei unterschiedliche Baureihen zeigt. In beiden Baureihen wird dieselbe Hardwarekomponente *KLA-HW 1*

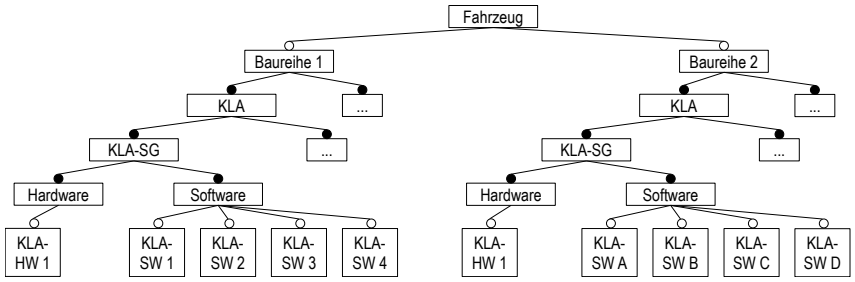


Abbildung 5.7: Erweiterter Produktbaum mit zwei Fahrzeugbaureihen

verwendet. Für jede Baureihe existieren aber jeweils vier unterschiedliche Softwarekomponenten. Zwischen *KLA-HW 1* und allen acht Softwarekomponenten besteht eine Kompatibilitätsbeziehung, die nicht im Bild dargestellt ist. Obwohl also *KLA-HW 1* und *KLA-SW 1* damit kompatibel sind, können beide nicht gemeinsam in Baureihe 2 eingesetzt werden. Die korrekte Verwendung beider Komponenten wird erst durch die zusätzlichen Informationen aus der Produktstruktur sichergestellt.

5.2.4 Austauschbarkeitsbeziehung

Die Austauschbarkeitsbeziehung wird in dieser Arbeit zwischen einzelnen Softwarekomponenten und zwischen einzelnen Hardwarekomponenten verwendet und stellt eine explizite Ordnung zwischen den jeweiligen Komponenten her. Eine austauschbare Komponente ist bezüglich festgelegter Eigenschaften äquivalent zur der Komponente, die durch sie ersetzt wird. Sie kann also entweder überall dort eingesetzt werden, wo die Vorgängerkomponente eingesetzt wird oder eingesetzt werden könnte (Austauschbarkeit am Stamm) oder sie ersetzt die Vorgängerkomponente nur in der Verwendung, für die die Austauschbarkeit angegeben wurde (Austauschbarkeit ist verwendungsbezogen). Damit existiert eine Annahme über die Produkte, in denen das Vorgängerteil eingesetzt wird. Wird eine Komponente als austauschbar deklariert, muss für jede Konfiguration oder jedes Produkt, in denen eine der Vorgängerkomponenten vorkommt überprüft werden, ob die neue Komponente dort verwendet werden kann.
Die Austauschbarkeit wird jedoch nicht als alleinige Einsatzbedingung für eine Komponente verstanden. Einer Komponente werden zusätzlich immer weitere Gültigkeitsbedingungen zugeordnet, die aus der Codebedingung und der Kompatibilität zwischen Hard- und Software bestehen, ohne deren Erfüllung sie im aktuellen Produkt nicht eingesetzt werden darf. Offenbar beinhalten die Gültigkeitsbedingung und die Austauschbarkeitsbeziehung redundante Informationen. Durch die Gültigkeitsbedingung wird eine Aussage über die Verbaubarkeit einer Softwarekomponente in einem Fahrzeug getroffen. Dieselbe Aussage wird auch durch die Austauschbarkeit basierend auf einer Vorgängerkomponen-

te getroffen. Dennoch sind beide Informationen nötig. Die Austauschbarkeitsbeziehung kann kein vollständiger Ersatz für die Gültigkeitsbedingung sein, da

1. eine Aufspaltung der Softwarekette möglich sein muss. Das ist insbesondere wichtig für die Verwendung derselben Software in verschiedenen Fahrzeugen, wenn es unterschiedliche Nachfolger für die verschiedenen Fahrzeuge gibt. Die durch die Aufspaltung entstehende Lösungsmenge muss durch weitere Gültigkeitsbedingungen eingegrenzt werden.

2. Zudem ist bei neuen, sich noch im Ursprungszustand befindlichen Steuergeräten nicht immer eine Software enthalten, auf deren Basis eine Ersatzsoftware ermittelt werden kann.

Andersherum kann auf die Austauschbarkeitsbeziehung nicht verzichtet werden, da durch sie die bestehende Software in die Ermittlung einer neuen Software für ein Fahrzeug einfließen kann. Die bestehende Software steht stellvertretend für viele Produktmerkmale, die durch sie realisiert werden, die aber nicht durch Codes repräsentiert werden. Diese Berücksichtigung der bestehenden Software ist zur Steuerung des adäquaten Funktionsänderungsumfangs notwendig. Weiterhin kann durch sie, anstelle der Einbeziehung von Zeitangaben in eine Gültigkeitsbedingung, eine historische Ordnung innerhalb der Komponenten hergestellt werden.
Wie oben bereits erwähnt, gilt die Austauschbarkeit nur für festgelegte Eigenschaften. Wird eine Softwarekette sehr lang, kann somit mitunter nur noch sehr schwer festgestellt werden, ob der erste Teilnehmer der Kette noch durch den letzten Teilnehmer ausgetauscht werden kann. Zur Minimierung des Risikos tragen kürzere Softwareketten bei, so dass keine umfangreichen Änderungen durch eine neue Software in ein Fahrzeug gelangen können. Das führt wiederum zu einer starken Diversifizierung der bestehenden Produkte, da z.B. ältere Fahrzeuge nicht immer mit der neusten Software versorgt werden, sondern eigene, ältere Softwarestände erhalten. Durch die Einzelabhängigkeit und das Teilrelease können Kompromisslösungen für dieses Dilemma definiert werden.
In der vorgestellten Lösung wird die Austauschbarkeit ausschließlich aus der Sicht des Servicebereichs verwendet. Eine Austauschbarkeit aus Servicesicht bedeutet, dass eine Software eine bestehende Software ersetzen kann und soll. Wie auch die Kompatibilitätsbeziehung kann diese aber unterschiedlich für z.B. verschiedene Märkte oder Baureihen dokumentiert werden. Während *KLA-SW 5a* in Abb. 5.7 als austauschbarer Nachfolger für *KLA-SW 4* in der Linkslenkervariante der Baureihe 1 angegeben werden könnte, könnte *KLA-SW 5b* als Ersatz für die gleiche Software in der Rechtslenkervariante gültig gemacht werden. Weiterhin können unterschiedliche austauschbare Nachfolger für Produktion und Service oder in der Entwicklung vereinbart werden. Damit ist auch die Austauschbarkeitsbeziehung immer im Kontext ihrer Verwendung zu betrachten. Eine für einen bestimmten Kontext dokumentierte Austauschbarkeit heißt nicht zwangsläufig, dass diese in allen Umständen gilt. Genauso bedeutet eine nicht vorhandene Austauschbarkeit nicht zwangsläufig, dass diese nicht technisch möglich wäre.

5.2.5 Abhängigkeitsbeziehung

Die Abhängigkeitsbeziehung wird in dieser Arbeit zwischen zwei oder mehr Softwarekomponenten als Einzelabhängigkeit oder Teilrelease dokumentiert. Zusätzlich werden Abhängigkeiten in der dieser Arbeit zu Grunde liegenden Lösung auch durch Codes ausgedrückt, die eine Beziehung zwischen beliebigen Produktmerkmalen herstellen können. In Anlehnung an die allgemeine Definition der Kompatibilität kann die Abhängigkeit allgemein auf zwei Arten beschrieben werden:

1. Ein Produktmerkmal B ist abhängig von den Produktmerkmalen $A_1 \dots A_n$, wenn diese kompatibel zueinander sind und gemeinsam eine Spezifikation erfüllen, die Produktmerkmal B enthält.

2. Ebenfalls ist das Produktmerkmal A_1 zur Erfüllung von Produktmerkmal B abhängig von den Produktmerkmalen $A_2 \dots A_n$, wenn A_1 bereits im Produkt enthalten ist.

Für die erste Beschreibung wurde weiter oben bereits der Begriff der abstrakten und konkreten Produktmerkmale eingeführt. Ein abstraktes Produktmerkmal ist also abhängig von den konkreten Produktmerkmalen, durch die es realisiert wird. Produktmerkmale, die nach der zweiten Beschreibung abhängig voneinander sind, sind immer kompatibel zueinander.
Anhand der Definition, die sich auf ein entstehendes Produktmerkmal bezieht, wenn zwei Produktmerkmale zusammengefügt werden, kann abgeleitet werden, dass auch die Abhängigkeit kontextbezogen betrachtet werden muss. Eine globale Aussage darüber, ob ein Teil abhängig von einem anderen ist, ist nicht möglich. Nur unter der Betrachtung, ob das entstehende Produkt oder Produktmerkmal im aktuellen Kontext gewünscht beziehungsweise gefordert ist, kann eine Aussage über die Abhängigkeit getroffen werden. In unserem Beispiel besteht eine Einzelabhängigkeit zwischen dem SAM-Steuergerät (*SAM-SW 3* und *SAM-SW 4*) und dem KLA-Steuergerät (*KLA-SW 3*), wenn der Innenraumschutz aktiviert ist. Diese Abhängigkeit wurde für den Servicebereich dokumentiert, da dort für alle zu rekonfigurierenden Fahrzeuge sichergestellt werden soll, dass die Klimaanlage den Betrieb des Innenraumschutzes nicht beeinflusst. Dieses Produktmerkmal *Innenraumschutz wird nicht durch Klimaanlage beeinflusst* ist im Servicebereich gewünscht, könnte aber im Fall von Testfahrzeugen in der Entwicklung unerwünscht sein, um technische Details des Zusammenspiels zwischen dem SAM-Steuergerät und dem KLA-Steuergerät untersuchen zu können.
Noch genauer auf den Kontext abgestimmt werden kann ein Rekonfigurationsszenario durch die Unidirektionalität der Einzelabhängigkeitsbeziehung. Die Auswahl des zu flashenden Steuergeräts geht dadurch in die Ermittlung der Abhängigkeit ein. Würde im Beispiel das KLA-Steuergerät zum Flashen ausgewählt werden, würde keine Abhängigkeitsbeziehung gelten. Offensichtlich ist das Produktmerkmal *Innenraumschutz wird nicht durch Klimaanlage beeinflusst* auch immer dann erfüllt, wenn sich die Software

KLA-SW 3 oder *KLA-SW 4* im Fahrzeug befinden; unabhängig davon, welche Software sich im SAM-Steuergerät befindet. In diesem Fall kann also Flashzeit und damit Kosten gespart werden. Der Kompositionsgrad ist hierbei so hoch, dass das Produktmerkmal *Innenraumschutz wird nicht durch Klimaanlage beeinflusst* nicht separat rekonfiguriert werden kann, sondern in Abhängigkeit der verbauten Softwarekomponenten im SAM- und KLA-Steuergerät erfüllt oder nicht erfüllt ist.
Weiterhin kann die Abhängigkeitsinformation auch durch andere Dokumentationsmittel ausgedrückt werden. Bei der Produktion von Fahrzeugen kann die Ermittlung der benötigten Teile zum Beispiel anhand einer zeitlichen Gültigkeit (vgl. *Effectivity*, [ES01]) gesteuert werden. Alle Komponenten, die zum gleichen Zeitpunkt gültig sind und die sonstigen Gültigkeitsbedingungen (Codes) erfüllen, werden in die Stückliste für das zu produzierende Fahrzeug aufgenommen. Eine explizite Abhängigkeitsinformation über Einzelabhängigkeiten oder Teilreleases liegt hier nicht vor (vgl. [Man05]). Diese müssen dann nachträglich dokumentiert werden, wenn einzelne Komponenten im Service ausgetauscht werden sollen und weitere Veränderungen des bestehenden Fahrzeugs dazu notwendig sind. Abhängigkeitsinformationen müssen auch dann nachträglich dokumentiert werden, wenn Komponenten, die zuvor nur im Verbund, später aber getrennt voneinander getauscht werden sollen und sich dadurch die Produktstruktur ändert. Die Abhängigkeit war anfangs implizit zum Beispiel durch die physikalische Verbindung zwischen den einzelnen Komponenten erfüllt und muss nun, nach dem *Aufbrechen* dieser Verbindung durch eine entsprechende Dokumentation ersetzt werden.

Zusammenfassend kann gesagt werden, dass zur Definition der Allgemeingültigkeit einer Aussage beziehungsweise eines Beziehungstyps die Allgemeinheit definiert werden muss. Dazu muss jeder Kontext bekannt sein, in dem eine Aussage eingesetzt werden kann. Ein Kontext ist immer gleichzusetzen mit einem Realisierungsszenario, das wiederum von dem Realisierer abhängt (Entwicklung, Produktion, Service) bzw. den Realisierungsaufträgen, die innerhalb jedes Realisierers existieren (z.B. Rekonfigurationauftrag im Service, Produktionsauftrag in der Produktion oder Entwicklungsauftrag in der Entwicklung). Ändern sich die Kontexte, in denen eine Aussage zum Produkt verwendet wird z.B. durch einen neuen Auswertealgorithmus im Service, kann eine vorher allgemeingültige Aussage zu einer verwendungsbezogenen Aussage werden. Dann müssen die Regelungen, unter denen diese Aussage getroffen werden soll und kann, angepasst werden und der Aufwand bei der Dokumentation steigt, da zusätzliche Auswirkungen der Aussage berücksichtigt werden müssen.
Die allgemeine Bedeutung der Kompatibilität, Austauschbarkeit und Abhängigkeit kann hier nicht angewendet werden. Stattdessen sind diese Informationen Teil der Gesamtinformation (Rekonfigurationswissen), die zur Ermittlung eines Rekonfigurationsszenarios ausgewertet werden.

6 Rekonfigurationstestsystem

Die Dokumentation des Rekonfigurationswissens wird durch verschiedene Unternehmensbereiche wie der Entwicklung und der Servicesteuerung vorgenommen. Durch die modularisierte Produktentwicklung werden Produktdaten innerhalb der Entwicklung oder Servicesteuerung dabei weitgehend unabhängig durch die zuständigen Fachbereiche für die verschiedenen Produktmodule erstellt. Anschließend an diese Dokumentation wird das Rekonfigurationswissen über den Softwarelogistikprozess den Werkstätten zur Verfügung gestellt. Erst gemeinsam und unter Berücksichtigung der möglichen Rekonfigurationsanforderungen kann die Korrektheit der eingegebenen Daten evaluiert werden. Insbesondere die semantische Fehlerfreiheit kann nur unter Einbeziehung aller Daten und in einem interaktiven Prozess mit dem Dokumentierenden überprüft werden.

In Abschnitt 4 wurden zwei Lösungen zur Dokumentation von Abhängigkeiten zwischen mehreren Produktmerkmalen vorgestellt, die die Komplexität bei der Abstimmung des zu dokumentierenden Rekofigurationswissens zwischen den Fachbereichen zusätzlich steigert. Weiterhin sind nun neue Rekonfigurationsszenarien möglich, die bei falschem Einsatz zu unnötigen, kostenintensiven Reparaturen führen können.

Zur Unterstützung des Dokumentierenden ist daher ein System notwendig, das für Transparenz über die Auswirkung seiner Eingaben zum Rekonfigurationswissen sorgt. Im folgenden Abschnitt soll ein Konzept für ein solches *Rekonfigurationstestsystem (RTS)* vorgestellt werden.

6.1 Anforderungen und Ziele

Das Ziel des RTS ist die Überprüfung von vorhandenem Rekonfigurationswissen. Bei der Überprüfung müssen Aussagen zur *syntaktischen* und *semantischen Korrektheit* getroffen werden können.

Durch die direkten Ergebnisse des Systems oder die Schlussfolgerungen, die ein Entwickler oder Servicesteuerer basierend auf diesen Ergebnissen ziehen kann, können sowohl die Produktdokumentation als auch die Produktgestaltung beeinflusst werden (vgl. Abb. 1.1 in Abschnitt 1.2.1). Die Produktdokumentation wird dann beeinflusst, wenn zuvor eingegebenes Rekonfigurationswissen korrigiert wird, um die verfolgte Rekonfigurationsstrategie richtig zu implementieren. Die Produktgestaltung wird angepasst, indem die bestehende *Rekonfigurationsstrategie*, *Entwicklungsstrategie* oder *Absicherungsstrategie* basierend auf den Ergebnissen des RTS verändert wird.

6.1.1 Syntaktische Korrektheit

Die syntaktische Korrektheit wird durch Tests gewährleistet, die die Regeln zur Dokumentation des Rekonfigurationswissens überprüfen. Sie sollte bereits während der Eingabe der Daten überprüft werden. Das RTS soll nur in solchen Fällen zur Überprüfung eingesetzt werden, in denen das gesamte Rekonfigurationswissens zur Überprüfung benötigt wird. Die syntaktischen Regeln sind aus den Beschreibungen der Einzelabhängigkeiten und Teilreleases ableitbar und werden im Weiteren nicht weiter beschrieben.

6.1.2 Semantische Korrektheit

Die semantische Korrektheit stellt den umfangreicheren und wichtigeren Baustein des RTS dar. Als Grundlage für ihre Überprüfung muss für alle an der Dokumentation des Rekonfigurationswissens Beteiligten Transparenz geschaffen werden. Da die semantische Korrektheit vor allem durch manuelle Auswertungen evaluiert werden kann, müssen die Rekonfigurationsergebnisse für verschiedene Rekonfigurationsszenarien den Dokumentierenden zugänglich gemacht werden, bevor die entsprechende Dokumentation freigegeben wird. Dabei muss besonders die Auswirkung von Abhängigkeitsbeziehungen sichtbar werden.
Die möglichen Aussagen, die das RTS zur Überprüfung der semantischen Korrektheit treffen kann, werden im Folgenden in *einfache Aussagen* und *Optimierungsvorschläge* unterteilt.

Einfache Aussagen

Die einfachen Aussagen werden in Form von Fragen formuliert, bei denen ein Anwender des RTS Unterstützung erhält. Die ersten beiden Fragen werden durch weitere Fragen zusätzlich detailliert:

1. Wird das richtige Rekonfigurationskonzept für eine Rekonfigurationsanfrage erstellt?
   1.1. Wird die richtige Softwarekomponente für eine Rekonfigurationsanfrage gefunden?
   1.2. Werden alle Abhängigkeiten korrekt aufgelöst?
2. Wie hoch sind die Rekonfigurationskosten für ein vorgegebenes Rekonfigurationsszenario?
   2.1. Wie hoch sind die Teilekosten für Hardwaretausche?
   2.2. Wie hoch sind die Reparaturzeiten und die Kosten dafür?
3. Welche Anzahl an Fahrzeugkonfigurationen entsteht durch das dokumentierte Rekonfigurationswissen?

Optimierungsvorschläge

Optimierungsvorschläge können durch das RTS basierend auf den einfachen Aussagen 1 - 3 von oben erstellt werden. Sie dienen der Optimierung der Rekonfigurationsstrategie bezüglich der entstehenden Rekonfigurationskosten (siehe Aussage 2) und der Anzahl der entstehenden Fahrzeugkonfigurationen (siehe Aussage 3)[1]. Eine automatische Optimierung hinsichtlich der Ermittlung des korrekten Rekonfigurationskonzepts (siehe Aussage 1) wird an dieser Stelle ausgeschlossen, da über das bestehende Rekonfigurationswissen hinaus kein Redundanzmodell existiert, das zur Bewertung eines entstandenen Rekonfigurationsszenarios herangezogen werden kann. Insbesondere hinsichtlich des Funktionsänderungsumfangs scheint eine adäquate Modellierung derzeit nicht möglich.
Zur Erstellung von Optimierungsvorschlägen müssen die existierenden Rekonfigurationsszenarien simulativ angepasst werden und anschließend eine Bewertung des dadurch neu entstandenen Rekonfigurationsszenarios bezüglich der einfachen Aussagen (Optimierungskriterien) aus dem vorherigen Absatz durchgeführt werden. Eine Anpassung der Rekonfigurationsszenarien entspricht der Veränderung der Produktgestaltung und kann daher durch die Anpassung der Entwicklungs-, Absicherungs- oder Rekonfigurationsstrategie vorgenommen werden. Grundsätzlich können dazu die Beziehungen zwischen den im Rekonfigurationswissen enthaltenen Komponenten und die beteiligten Komponenten selbst verändert werden.
Eine Veränderung der Rekonfigurationsstrategie ist im einfachsten Fall durch die Dokumentation anderer Rekonfigurationsszenarien basierend auf den baubaren und abgesicherten Konfigurationen möglich. Dies kann durch die Änderung der Beziehungen zwischen den im Rekonfigurationswissen bereits existierenden Komponenten simuliert werden. Sind bestimmte Rekonfigurationsszenarien so nicht implementierbar, können zusätzliche Konfigurationen zielgerichtet durch die Neuentwicklung von Komponenten (Anpassung der Entwicklungsstrategie) bereitgestellt werden. Um das zu simulieren, kann eine neue, fiktive Komponente zum Rekonfigurationswissen hinzugefügt werden, deren Eigenschaften sich durch ihre Kompatibilitäten, Austauschbarkeit und Abhängigkeiten, also durch die zuvor vorgestellten Beziehungstypen, modellieren lassen. Dabei können Anforderungen an die Kompatibilität der neuen Komponenten aus den Ergebnissen des RTS abgeleitet werden. Die neuen Komponenten sind dann in zusätzlichen Rekonfigurationsszenarien verwendbar. Eine Anpassung der Absicherungsstrategie und damit zusätzliche Absicherungen müssen immer durchgeführt werden, wenn sich die Entwicklungsstrategie ändert und neue Komponenten entwickelt werden. Sie muss ebenfalls angepasst werden, wenn durch eine neue Rekonfigurationsstrategie eine Menge neuer Konfigurationen zum Einsatz kommt, die zuvor noch nicht abgesichert wurde. Dazu können, wie auch bei der Anpassung der Entwicklungsstrategie, neue Konfigurationen im Rekonfigurationswissen bereitgestellt werden.

[1] Für die Berechnung der Rekonfigurationskosten müssen in diesem Fall zu den Kosten in der Werkstatt alle Kosten gerechnet werden, die durch die zusätzliche Absicherung einer neuen Konfiguration entstehen, um neue Rekonfigurationsszenarien zu ermöglichen.

Alle genannten Möglichkeiten zur simulativen Veränderung der drei Strategien spannen einen unendlich großen Suchraum für das Finden neuer Rekonfigurationsszenarien auf. Viele der so entstehenden Rekonfigurationsszenarien enthalten mit sehr hoher Wahrscheinlichkeit zu viele Szenarien, die allein aus Sicht der Korrektheit nicht zielführend sind. Denn in jedem Fall muss das entstandene Rekonfigurationsszenrio neben den Optimierungskriterien Kosten und Anzahl der Fahrzeugkonfigurationen das Kriterium der Korrektheit erfüllen, für das wie oben erwähnt kein redundantes Modell zusätzlich zum Rekonfigurationswissen besteht. Daher ist eine Simulation nur eingeschränkt unter Einbeziehung des Anwenders möglich, der die entstandenen Rekonfigurationsszenarien auf ihre Korrektheit hin überprüfen kann.
Auf Grund dieser Einschränkungen soll im Folgenden ein stark vereinfachtes Konzept der Optimierungsvorschläge vorgestellt werden, das sich auf die Rekonfigurationskosten als Optimierungskriterium beschränkt. Dabei soll die Aussage getroffen werden können, welche Konfigurationen freigegeben werden müssen, um die Gesamtkosten zu minimieren. Diese setzen sich zusammen aus den Werkstattkosten und den Kosten für die Entwicklung und Absicherung neuer Komponenten (Entwicklungskosten).

6.2 Konzept

6.2.1 Systemarchitektur

Zur Umsetzung der oben aufgeführten Anforderungen und Ziele soll das RTS an einem Punkt in den Softwarelogistikprozess integriert werden, der sich möglichst am Ende der Datenbereitstellung und -versorgung befindet. So können Fehler in der Datenerstellung als auch im Versorgungsprozess berücksichtigt werden (siehe Abb. 6.1). Das RTS benötigt

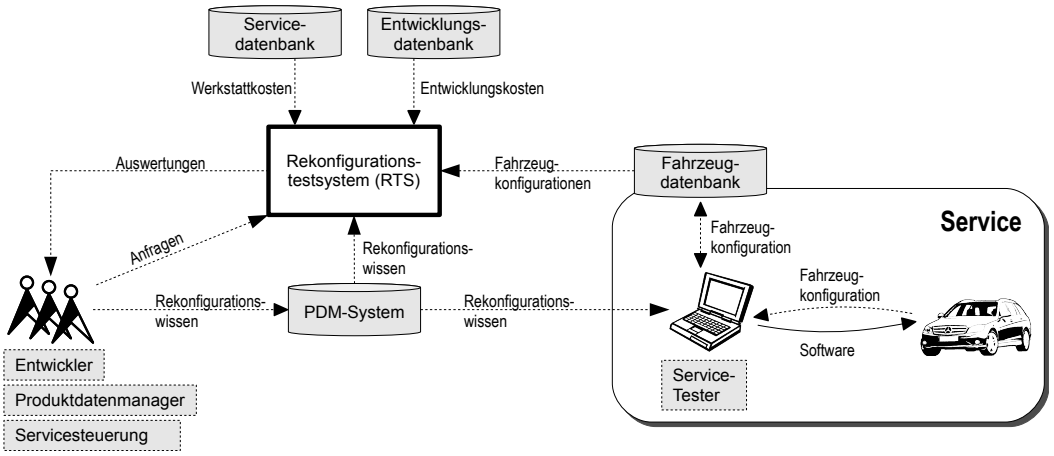


Abbildung 6.1: Anordnung eines Rekonfigurationstestsystems im Softwarelogistikprozess

die folgenden Eingangsschnittstellen:

PDM-System: Versorgung des RTS mit dem letzten Stand des *Rekonfigurationswissens*, das für die Versorgung in die Werkstatt vorbereitet und freigegeben wurde

Fahrzeugdatenbank: Versorgung des RTS mit den aktuell im Feld vorhandenen *Fahrzeugkonfigurationen*

Servicedatenbank: Versorgung des RTS mit *Werkstattkosten* für Hardwaretausche und Flashvorgänge

Entwicklungsdatenbank: Versorgung des RTS mit *Entwicklungskosten* für die Entwicklung und Absicherung zusätzlicher Zielkonfigurationen

Anwender: Anwender können Entwickler, Produktdatenmanager und die Servicesteuerung sein. Durch sie erfolgt die Vorgabe und Verwaltung der benötigten *Anfragen* zur Bearbeitung durch das RTS.

Basierend auf den Daten der Eingangsschnittstellen werden die Auswertungen des RTS an der Ausgangsschnittstelle bereitgestellt:

Anwender: Ausgabe von *Auswertungen* zur syntaktischen und semantischen Korrektheit basierend auf den gestellten Anfragen

6.2.2 Erstellung von Rekonfigurationsszenarien

Die Generierung der Rekonfigurationsszenarien kann entweder *manuell* oder *automatisch* erfolgen und auf *realem* oder *fiktivem* Rekonfigurationswissen basieren.
Für die einfachen Aussagen kommt meist die manuelle Generierung mit realem oder fiktivem Rekonfigurationswissen zum Einsatz (Aussagen 1 - 2) und für die Aussagen 2 - 3 zusätzlich die automatische Generierung mit realem Wissen. Für die Optimierungsvorschläge erfolgt die Generierung automatisch mit realem oder fiktivem Rekonfigurationswissen.

Manuelle Generierung

Erfolgt die Generierung manuell, bestimmt der Anwender das Szenario, indem er ein Rekonfigurationsziel, also das Ausgangsprodukt und ein zu flashendes Steuergerät auswählt. Weiterhin kann er das reale Rekonfigurationswissen auswählen oder fiktives Rekonfigurationswissen bereitstellen, um die Veränderung einer der Strategien zu simulieren.

Automatische Generierung

Bei der automatischen Generierung wird das Ausgangsprodukt vom Anwender ausgewählt und anschließend automatisch alle möglichen Rekonfigurationsziele durch Kombination des gewählten Produkts mit allen möglichen flashbaren Steuergeräten erzeugt. Als Erweiterung können alle existierenden Ausgangsprodukte automatisch in die Auswertung einbezogen werden, statt eine Auswahl durch den Anwender vorauszusetzen.

Hierbei können Daten über Flashhäufigkeiten eingesetzt werden, um nur die wichtigsten Rekonfigurationsszenarien zu generieren.

6.2.3 Anwenderschnittstellen

Auf Grund der verschiedenen Aussagen, die durch das RTS erzeugt werden können, muss die Anwenderschnittstelle (siehe Abb. 6.1) sehr unterschiedlich ausgelegt werden. Zu diesem Zweck soll die Funktionalität des RTS in zwei separaten Systemen implementiert werden.

Regeltest

Einfache Aussagen, die auf einer automatischen Generierung der Rekonfigurationsszenarien mit realem Rekonfigurationswissen beruhen (Aussagen 2 - 3), können in einem *Regeltest* implementiert werden. Dieser wird automatisch dann durchgeführt, sobald neues Rekonfigurationswissen freigegeben werden soll. Die Ergebnisse werden dem Anwender dann in Form eines Berichts zur Verfügung gestellt. Bei den Aussagen zu den Rekonfigurationskosten (Aussagen 2.1 und 2.2) können zusätzlich Schwellwerte definiert werden, die bei Überschreitung zu einer Information an den Anwender führen. Dadurch können unbeabsichtigte Hardwaretausche oder zu lange Flashzeiten zuverlässig entdeckt werden. Neben diesen semantischen Tests sollen auch alle syntaktischen Tests im Regeltest implementiert werden.

Interaktiver Test

Einfache Aussagen, die auf einer manuellen Generierung der Rekonfigurationsszenarien mit realem oder fiktivem Rekonfigurationswissen beruhen oder beruhen können (vor allem die Aussagen 1.1 und 1.2, aber auch 2.1 und 2.2) sowie die Optimierungsvorschläge benötigen eine umfangreichere Anwenderschnittstelle als die Regeltests und sollen in einem *interaktiven Test* implementiert werden.
Zur Erzeugung der einfachen Aussagen muss der Anwender die folgenden Möglichkeiten zur Eingabe haben:

- Fahrzeuge müssen aus der Fahrzeugdatenbank geladen und verändert werden können. Zusätzlich müssen eigene Fahrzeuge erstellt und gespeichert werden können. Für ein Fahrzeug muss dabei die Hard- und Software aller Steuergeräte sowie die Codes angepasst werden können.

- Das zu verwendende Rekonfigurationswissen muss ausgewählt werden können. Dabei muss die Auswahl zwischen dem aktuellen Rekonfigurationswissen und selbst erstelltem, fiktivem Rekonfigurationswissen möglich sein. Das vorhandene Rekonfigurationswissen muss testweise verändert werden können, um Iterationen bei der Optimierung des Wissens zu ermöglichen.

- Ein gesamtes Rekonfigurationsszenario muss als Testfall gespeichert und geladen werden können, um Testergebnisse zu einem späteren Zeitpunkt erneut betrachten zu können.

Nach diesen Eingaben müssen alle Aussagen 1 - 3 basierend auf den oben beschriebenen Eingangsdaten vom RTS getroffen werden. Dadurch kann der Anwender das von ihm oder anderen Anwendern erstellte Rekonfigurationswissen iterativ optimieren und auf seine Korrektheit überprüfen.
Für die Optimierungsvorschläge müssen die folgenden Möglichkeiten zur Eingabe bereitgestellt werden:

- Laden der relevanten Fahrzeugkonfigurationen (Fahrzeuge) aus der Fahrzeugdatenbank. Zusätzlich muss eine Eingrenzung auf die relevanten Steuergeräte im Fahrzeug vorgenommen werden können, um die anschließende Vorgabe der Zielkonfiguration zu vereinfachen.

- Vorgabe einer Zielkonfiguration für jede bestehende Fahrzeugkonfiguration. Dabei werden je Steuergerät die Softwarekomponenten angegeben, die zwingend in der Zielkonfiguration enthalten sein müssen. Wird für ein Steuergerät keine Zielsoftware angegeben, kann an dieser Stelle jede Softwarekomponente des Steuergeräts in der Zielkonfiguration enthalten sein.

- Explizite Vorgabe aller (theoretisch) möglichen Fahrzeugkonfigurationen

- Manuelle, selektive Freigabe von baubaren und abgesicherten Konfigurationen

- Manueller Start der Suche nach den kostengünstigsten Freigaben (Optimierungsvorschlag)

- Vergleichsmöglichkeit für die Ergebnisse verschiedener Rekonfigurationsvorgänge

Basierend auf diesem Konzept kann das RTS die Gesamtkosten, die sich aus den Werkstatt- und Entwicklungskosten zusammensetzen, minimieren. Dazu berechnet es die kleinste Freigabemenge für die vorgegebenen möglichen Konfigurationen basierend auf einer einfachen Suche, die alle Freigabekombinationen testet. Das optimale Ergebnis enthält die Freigabemenge, für die neben dem Kriterium der minimalen Kosten auch die vorgegebenen Ziel-Fahrzeugkonfigurationen (Rekonfigurationsziele) erfüllt sind.
Dieses Vorgehen entspricht grundlegend der Rekonfiguration als modellbasierter Diagnose, wie sie in Abschnitt 2.7.3 beschrieben wurde. Es stellt damit eine Vereinfachung gegenüber dem zuvor vorgestellten Softwareermittlungsalgorithmus dar, da das Rekonfigurationsergebnis durch den Anwender vorgegeben werden muss.
Die Ergebnisse der Implementierung dieses Systems sind in Abschnitt 7.2 beschrieben.

7 Ergebnisse

In den Kapiteln 3 - 5 wurde ein Konzept zur Dokumentation von Abhängigkeiten zwischen Softwarekomponenten vorgestellt, das während der Rekonfiguration von Fahrzeugen in einer Werkstatt eingesetzt wird. Basierend auf diesem Konzept wurde eine Implementierung in der bestehenden Systemlandschaft der Daimler AG vorgenommen (siehe Abschnitt 7.1). Zur Unterstützung eines Anwenders bei der Dokumentation des Rekonfigurationswissens wurde das Rekonfigurationstestsystem aus Kapitel 6 in reduziertem Umfang implementiert (siehe Abschnitt 7.2).

7.1 Rekonfigurationswissen

Durch die Implementierung der Abhängigkeitsmethodik können Abhängigkeiten zwischen Softwarekomponenten durch den Entwickler, Produktdatenmanager und Servicesteuerer eingegeben werden. Zu definierten Zeitpunkten erfolgt die Freigabe aller für die Rekonfiguration benötigten Daten, woraufhin die Daten in die Werkstatt versorgt werden. Diese Daten werden für jede Fahrzeugreparatur, die eine Softwareaktualisierung eines Steuergeräts beinhaltet, durch den Service-Tester ausgewertet und die so ermittelte(n) Softwarekomponente(n) in das Fahrzeug übertragen.
Die benötigten Voraussetzungen zur Eingabe, Weiterverarbeitung und Auswertung von Einzelabhängigkeiten und Teilreleases wurden als Teil eines umfangreichen Projekts zur Dokumentation von Produktdaten (DeepC, siehe [Sto07]) geschaffen, getestet und erfolgreich pilotiert. Sie stehen damit für den Einsatz im Service der Daimler AG zur Verfügung. Bei der Umsetzung des Konzepts haben Anpassungen des hier vorgestellten Konzepts an die bestehenden Rahmenbedingungen stattgefunden, z.B. Verhalten bei unbekannten Softwarekomponenten im Fahrzeug; die Nutzung vorhandener Steuergerätebezeichnungen zur Bildung eines adäquaten Steuergerätebegriffs; das Zusammenspiel von Einzelabhängigkeiten und Teilreleases; die Versorgung zusätzlicher Softwarekomponenten, die erst durch Teilreleases gültig werden; Asynchronitäten zwischen der Steuer- und Nutzdatenversorgung in die Werkstätten und Gateway-Steuergeräte als Teilreleaseteilnehmer, die zum Zeitpunkt der Softwareermittlung nicht in Betrieb genommen sind und somit den Zugriff auf dahinterliegende Steuergeräte verhindern.
Die Eingabe der Abhängigkeitsinformationen wurde in den Standardfreigabeprozess von Softwarekomponenten integriert. Damit erfolgt die Dokumentation von Einzelabhängigkeiten und Teilreleases über die Konstruktionseinsatzmeldung der abhängigen Softwarekomponente(n). Diese Daten können im Fall der Einzelabhängigkeiten über eine neu geschaffene Benutzerschnittstelle in das PDM-System übernommen werden. Im Fall der

Teilreleases wurde auf Grund unterschiedlicher Randbedingungen zunächst ein separates System zur Erzeugung von Teilreleaseinformationen geschaffen und ein Stufenkonzept zur Integration der Lösung in die bestehende Systemlandschaft erstellt.
In beiden Fällen finden zum Zeitpunkt der Eingabe Überprüfungen zur syntaktischen Korrektheit statt und die Daten werden für den Versorgungsprozess in die Werkstatt gespeichert. Die Auswertung beider Abhängigkeitsdokumentationen wurde in den bestehenden Rekonfigurationsalgorithmus im Service-Tester integriert, so dass eine transparente Nutzung beider Lösungen möglich ist. In Abb. 7.1 ist die Anzeige des Ergebnisses einer Softwareermittlung für das SAM-Steuergerät basierend auf Einzelabhängigkeiten zu sehen. Für den Fall eines notwendigen Hardwaretauschs wurden entsprechende Interaktionen zwischen dem Werkstattmitarbeiter und dem Service-Tester definiert.

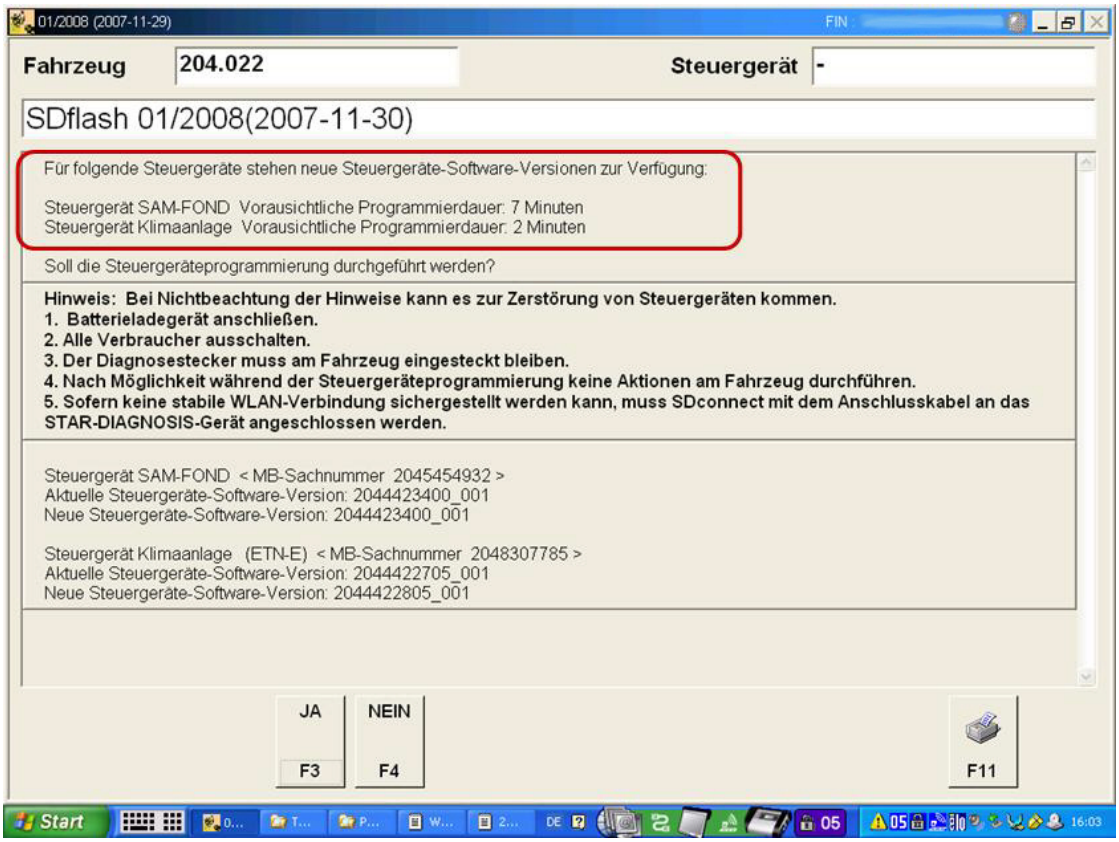


Abbildung 7.1: Anzeige einer Einzelabhängigkeit im Service-Tester

In einer Pilotierung der Einzelabhängigkeiten wurden im ersten Halbjahr 2008 608 Flashvorgänge durchgeführt, an denen das SAM- und Klimasteuergerät der aktuellen Mercedes-Benz C-Klasse beteiligt waren. In 427 Fällen davon führten die dokumentierten Einzelabhängigkeitsbeziehungen zu einer gemeinsamen Rekonfiguration des SAM- und Klimasteuergeräts, da ursprünglich eine Software für das SAM-Steuergerät ermittelt

werden sollte und eine Einzelabhängigkeit zwischen der entsprechenden Software dieses Steuergeräts und einer passenden Software des Klimasteuergeräts dokumentiert war. In 107 Fällen ging die Softwareermittlung ebenfalls vom SAM-Steuergerät aus, das Klimasteuergerät wurde aber nicht angepasst, da dort bereits die benötigte Software vorhanden war. In 74 Fällen wurde nur das Klimasteuergerät geflasht, da die Softwareermittlung von diesem Steuergerät ausging, die Einzelabhängigkeit aber nur unidirektional vom SAM-Steuergerät auf das Klimasteuergerät zeigte.
Im Rahmen der gesamten bisherigen Laufzeit seit Anfang 2008 wurden die Einzelabhängigkeiten für sieben Fahrzeugmodelle der Marken Mercedes-Benz und smart produktiv eingesetzt. Teilreleases werden in 18 Mercedes-Benz Fahrzeugmodellen verwendet.
Mit Stand Oktober 2010 sind insgesamt 35 Einzelabhängigkeiten für 4 - 15 Softwarekomponenten und 2 - 4 Steuergeräte je Fahrzeugmodell dokumentiert, 13 davon mit einer eigenen Gültigkeitsangabe. Zum selben Zeitpunkt wurden 29 Teilreleases eingesetzt, wobei einige davon in mehreren Fahrzeugmodellen verwendet werden. Insgesamt sind somit 15 verschiedene Teilreleases dokumentiert. Je Fahrzeugmodell kommen dabei 1 - 5 Teilreleases zum Einsatz, die jeweils zwischen zwei und neun Steuergeräte und 5 - 40 verschiedene Softwarekomponenten enthalten. Ein Hardwaretausch wird zu diesem Zeitpunkt durch keines der Teilreleases erzwungen. Seit Oktober 2008 werden monatlich zwischen ca. 3.000 und 17.000 Softwarerekonfigurationen durchgeführt, bei denen Einzelabhängigkeiten ausgewertet werden. Davon werden ca. 100 - 6000 monatlich als gültig erkannt und führen zur Rekonfiguration eines zusätzlichen Steuergeräts. Teilreleases wurden im gleichen Zeitraum monatlich ca. 2.000 - 13.500 mal ausgewertet, ca. 250 - 3.500 mal davon mit positivem Ergebnis und entsprechenden zusätzlichen Rekonfigurationsvorgängen.

7.2 Rekonfigurationstestsystem

Die Funktionen des Rekonfigurationstestsystems wurden in einem bereits bestehenden und zwei neuen Testsystemen umgesetzt. Hierbei konnten nur Teilumfänge des in Kapitel 6 vorgestellten Konzepts umgesetzt werden.
Im Regeltest wurde die Erkennung von Hardwaretauschen (Werkstattkosten) und alle syntaktischen Tests implementiert. Der Test wird produktiv regelmäßig vor der Freigabe von neuem Rekonfigurationswissen durchlaufen.
Die interaktiven Tests wurden zum Teil in einem Onlinetest sowie zum Teil als eigenständige Anwendung realisiert.
Der Onlinetest wurde als Java EE-Anwendung implementiert, die produktiv im Einsatz ist und von verschiedenen Anwendern regelmäßig benutzt wird. Die Anwendung ist an die Fahrzeugdatenbank angebunden und verwendet den Softwareermittlungsalgorithmus, der auch im Service-Tester eingesetzt wird. So kann ein vollständig identisches Ergebnis einer Rekonfiguration in der Werkstatt simuliert werden. In Abb. 7.2 ist zu sehen, wie innerhalb des Onlinetests ein neues Fahrzeug erstellt werden kann, um dieses anschließend in einem fiktiven Rekonfigurationsszenario zu verwenden.

Abbildung 7.2: Erstellen eines fiktiven Fahrzeugs im Onlinetest

Durch die gute Nachvollziehbarkeit der Softwareermittlung wird der Onlinetest auch in Hotline-Fällen eingesetzt, wo er bei der Suche nach aktuellen Fehlern in einem vorliegenden Werkstattfall unterstützt.
Die Optimierungsvorschläge wurden in einer Java-Anwendung umgesetzt. Diese Anwendung wurde an bekannten Rekonfigurationsszenarien getestet und liefert korrekte Ergebnisse. In Abb. 7.3 ist ein Beispielergebnis für den Vergleich von vier Rekonfigurationsstrategien dargestellt. Die Säulen des Diagramms setzen sich aus den Werkstattkosten („min. Werkkosten“, unterer Säulenteil) und den Entwicklungskosten („Q-Kosten“, oberer Säulenteil) zusammen.

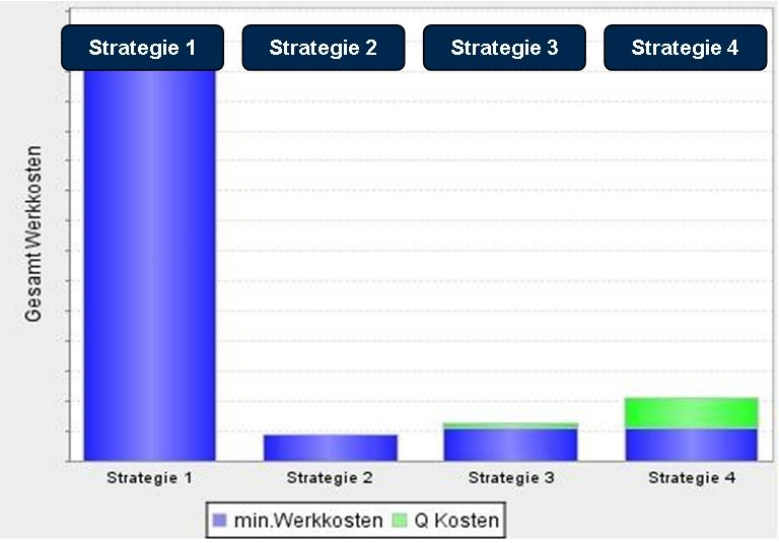


Abbildung 7.3: Darstellung der Reparaturkosten für vier Rekonfigurationsstrategien

8 Zusammenfassung und Ausblick

8.1 Zusammenfassung

Es gelangen immer vielfältigere Produkte auf den Markt und die Möglichkeiten zur individuellen Konfiguration durch den Kunden steigen. Diese Entwicklung gilt neben vielen anderen Produktbereichen auch für den Automobilmarkt und wird durch den Begriff „mass customization“ beschrieben.
Zur Realisierung dieses Angebots müssen in den Produktionsunternehmen einige technische Voraussetzungen geschaffen werden, wozu auch die Dokumentation der komplexen Produktstrukturen sowie die Abbildung möglicher Wartungsaufträge im Service zählen. Gerade hier entstehen durch eine große Produktvielfalt eine hohe Anzahl möglicher Reparaturszenarien und die Individualisierung der Produkte kann sich fortsetzen. Man spricht von der Rekonfiguration des Produkts.
Bei der Einordnung dieses Problems können, abhängig von der Komplexität im jeweiligen Problemfeld, unterschiedliche Konfigurations- und Rekonfigurationsklassen gebildet werden. Der Automobilbereich ist dabei durch seine hohe Anzahl variabler Produkte sowie die in Werkstätten angebotenen Aufrüst- und Wiederherstellungsmaßnahmen in die Unternehmensklasse mit den größten Herausforderungen beim Konfigurieren und Rekonfigurieren einzuordnen.
Das Ziel der Rekonfiguration eines Fahrzeugs im Service besteht in einer fehlerfreien, kundengerechten und wirtschaftlichen Reparatur. Grundlage für diese Zielsetzung ist eine möglichst weitgehend automatisierte Bereitstellung von Rekonfigurationsentscheidungen, um Fehler zu vermeiden. Diese Entscheidungen werden in der Produktdokumentation festgehalten und den Werkstätten zur Verfügung gestellt. Darin enthalten ist die Rekonfigurationsstrategie, die sowohl die Unternehmenssicht als auch die Kundensicht berücksichtigen muss. Für den Kunden sind die Kosten der Reparatur sowie ein adäquater Funktionsänderungsumfang wichtige Kriterien. Dieser Anspruch ist durch die Bereitstellung einer möglichst hohen Anzahl möglicher Rekonfigurationsszenarien gut erfüllbar. Dagegen werden die Ansprüche des Unternehmens, also die Begrenzung der Entwicklungskosten und des Dokumentationsaufwands, durch eine geringe Anzahl möglicher Rekonfigurationsszenarien unterstützt. Hier ist ein Optimum zu finden. In jedem Fall müssen aber alle angestrebten Rekonfigurationsszenarien dokumentierbar sein.
Aus diesem Grund werden in dieser Arbeit zwei Möglichkeiten zur Dokumentation von Abhängigkeitsbeziehungen zwischen Softwarekomponenten in Fahrzeugen vorgestellt, die auf einer bestehenden Dokumentationsmethodik für Rekonfigurationswissen basieren. Zuvor wird das Rekonfigurationsproblem im Automobil-Service in einem allgemeinen Do-

mänenmodell beschrieben. Anschließend wird die erstellte Lösung diskutiert sowie ein Testsystem zur Überprüfung des Rekonfigurationswissens vorgestellt.

Produkte werden heute modular entwickelt. Im Automobilbereich werden seit vielen Jahren Software- und Hardwarekomponenten eines Steuergeräts separat entwickelt, so dass verschiedene Softwarekomponenten auf derselben Hardwarekomponente zum Einsatz kommen können. Mit der Softwarelogistik werden die Dokumentations- und Datenversorgungsprozesse bereitgestellt, die ab der Entwicklung einer neuen Software bis zu deren Einsatz im Service notwendig sind. Dabei werden Steuerdaten, die das Wissen zur korrekten Verwendung der Softwarekomponenten enthalten, in PDM-Systemen gespeichert und für Produktions- und Servicebereiche zur Verfügung gestellt. Die Auswertung dieser Daten geschieht im Service durch den Service-Tester, der das Werkstattpersonal bei der Ermittlung einer passenden Software für ein vorliegendes Fahrzeug unterstützt.

Die dabei auftretenden Probleme können in die Problemklasse des Software Configuration Managements (SCM) eingeordnet werden. Teil dieses Problems ist die Modellierung variabler Produkte, die heute durch verschiedene Beschreibungsmodelle ermöglicht wird. Besonders die FODA-Notation oder deren Erweiterungen zur Beschreibung von Merkmalmodellen in Softwareproduktlinien hat sich zu diesem Zweck durchgesetzt. Daneben existieren Stücklisten in verschiedenen Ausprägungen, mit denen Produktausprägungen dokumentiert werden können.

Für die Rekonfiguration, die den Kern des in dieser Arbeit beschriebenen Problems im Automobil-Service bildet, ist zusätzlich zur Beschreibung von Produkten die Beschreibung ihrer Änderbarkeit erforderlich. Anhand dieser Beschreibung wird festgelegt, welche bestehende Ausgangskonfiguration unter welchen Bedingungen (Anforderungen) in welche neue Konfiguration überführt wird (Rekonfigurieren).

Im Domänenmodell des Rekonfigurierens wird diese Operation als Funktion $f : \mathbf{P} \mapsto \mathbf{P}$ definiert, wobei $\mathbf{P}$ die Menge aller Produkte ist, die jeweils aus beliebigen Produktmerkmalen besteht. Kunden, die ein Produkt besitzen, können eine Rekonfigurationsanfrage an den Servicebereich (Werkstatt) eines Unternehmens stellen, um die Rekonfiguration ihres Produkts einzuleiten. Eine Rekonfigurationsanfrage enthält ein Rekonfigurationsziel, das wiederum eine Rekonfigurationsoperation und eine Rekonfigurationsinvariante enthält. Diese legen fest, welche Produktmerkmale dem bestehenden Produkt hinzugefügt beziehungsweise welche daraus entfernt werden sollen und welche Produktmerkmale im Verlauf der Rekonfiguration nicht hinzugefügt oder entfernt werden dürfen. Die Werkstatt tritt dabei als Operationsrealisierer auf, der, basierend auf dem Rekonfigurationsziel des Kunden, für die Erstellung eines Rekonfigurationskonzepts und für die Umsetzung des Rekonfigurationsziels verantwortlich ist. Ein Operationsrealisierer besitzt das für diese Umsetzung notwendige Rekonfigurationswissen und kann, sofern er selbst nicht über ausreichendes Wissen verfügt, weitere Operationsrealisierer beauftragen Teile der Rekonfiguration zu übernehmen. Während der Realisierung, die aus der Konkretisierung und Vervollständigung des ursprünglichen Rekonfigurationsziels besteht, muss darauf geachtet werden, dass das Produkt nur soweit verändert wird, wie es den Erwartungen des

Kunden entspricht. Die konkrete Ausprägung eines Rekonfigurationsvorgangs wird als Rekonfigurationsszenario bezeichnet.

Das benötigte Wissen zur Umsetzung einer Rekonfiguration im Automobil-Service kann verteilt sein auf Werkstattmitarbeiter und den Service-Tester. Für eine automatisierte Rekonfiguration wird der größte Teil jedoch im Service-Tester vorliegen, so dass der Werkstattmitarbeiter lediglich Diagnosen auf der Ebene *Update* oder *Defekt im Steuergerät XY* stellen muss.

Die zentrale Komponente, über die Wissen im Rekonfigurationswissen abgelegt ist, ist das Steuergerät, das wiederum aus mehreren Hard- und Softwarekomponenten besteht. Die Verwendung eines Steuergeräts beziehungsweise einer Hard- oder Softwarekomponente in einem Produkt wird über Codes geregelt, die in aussagenlogischen Formeln als Variablen verwendet werden. Ergibt die einer Komponente zugeordnete Formel den Wert *wahr*, wird diese in der aktuellen Produktvariante verwendet. Weiterhin wird über Austauschbarkeiten die Fähigkeit einer Komponente ausgedrückt, anstelle einer anderen Komponente eingesetzt zu werden. Über die Kompatibilität zwischen Hard- und Software wird die Fähigkeit einer Softwarekomponente beschrieben, auf einer Hardwarekomponente lauffähig zu sein. Diese Beziehungen werden in der sogenannten passiven Rekonfigurationsphase über einen Algorithmus so ausgewertet, dass eine passende Softwarekomponente für ein vom Werkstattmitarbeiter ausgewähltes Steuergerät ermittelt wird. Aufbauend auf diesem bestehenden Algorithmus werden mit der *Einzelabhängigkeit* und dem *Teilrelease* zwei Beziehungstypen definiert, die die Möglichkeiten zur Rekonfiguration erweitern. Sie werden in der sogenannten aktiven Rekonfigurationsphase ausgewertet, deren Algorithmus beschrieben wird. Bei der Einzelabhängigkeit wird eine Softwarekomponente mit einer weiteren Softwarekomponente verknüpft, so dass beim Flashen der ersten Softwarekomponente auch die zweite Softwarekomponente ins Fahrzeug verbracht wird, wenn diese für das aktuelle Fahrzeug gültig ist. Beim Teilrelease werden mehrere Softwarekomponenten miteinander in Beziehung gesetzt, die gleichzeitig in ein Fahrzeug geflasht werden, sobald ein Teilnehmer dieses Teilreleases ins Fahrzeug gelangen soll.

Die vorgestellten Methoden zur Dokumentation von Abhängigkeiten bei der Softwareermittlung werden anhand verschiedener Kriterien, z.B. Kundenzufriedenheit, Wirtschaftlichkeit und Korrektheit positiv bewertet und ihre Einsatzfähigkeit in vier Dokumentationsszenarien verdeutlicht. Anschließend wird die Wissenstiefe der verwendeten Beziehungen diskutiert, wobei dargestellt wird, dass jegliches Rekonfigurationswissen ausschließlich im Kontext seiner Verwendung betrachtet werden kann. Durch die Bildung der Wissensebenen Konfigurations- und Realisierungswissen kann allerdings eine Konservierung von Wissen vorgenommen werden.

In einem Rekonfigurationstestsystem können Tests des erstellten Rekonfigurationswissens durchgeführt werden, die für mehr Transparenz über die Auswirkungen der dokumentierten Beziehungstypen sorgen. Basierend auf den Testergebnissen kann die Produktgestaltung oder -dokumentation angepasst werden. Der Schwerpunkt des Testsystems liegt auf der Überprüfung der semantischen Korrektheit, wozu dem Benutzer Rekonfigurati-

onsergebnisse für verschiedene Rekonfigurationsszenarien zur Verfügung gestellt werden. Hierbei können Fragen beantwortet werden, wie z.B. *Wird die richtige Softwarekomponente für eine Rekonfigurationsanfrage gefunden?* oder *Werden alle Abhängigkeiten korrekt aufgelöst?*. Neben diesen einfachen Testfällen kann das System Vorschläge zur Optimierung der Rekonfigurationsstrategie machen. Die Optimierung wird jedoch nur bezüglich der Werkstattkosten ermöglicht. Im System kann der Anwender die gewünschten Rekonfigurationsszenarien manuell erstellen oder automatisch erstellen lassen und reales oder fiktives Rekonfigurationswissen zu Grunde legen. Die Tests können als Regeltests oder interaktive Tests ablaufen.
Die gezeigte Lösung wurde erfolgreich im DeepC-Projekt der Daimler AG umgesetzt und pilotiert und steht für den produktiven Einsatz zur Verfügung. Dabei wurde eine Möglichkeit zur Eingabe des Rekonfigurationswissens und dessen Auswertung und Überprüfung geschaffen.

8.2 Ausblick

Aufbauend auf der in dieser Arbeit vorgestellten Lösung sind einige Erweiterungen vorstellbar. Die wichtigsten werden im Folgenden beschrieben:

1. Mit der hier vorgestellten Lösung kann kompiliertes Rekonfigurationswissen dokumentiert werden. Es ist nicht möglich, die Eigenschaften (Produktmerkmale) von Produkten zu dokumentieren, die beim Rekonfigurieren entstehen. Mit einer solchen Dokumentationsmöglichkeit würden Entwicklungsbereiche tieferes Wissen über ihre Produktbestandteile anlegen können, das später bei der Dokumentation neuer Rekonfigurationsszenarien genutzt werden könnte. Wie weiter oben bereits diskutiert, sind hierbei aber Aufwand und Nutzen gegeneinander abzuwägen.

2. Dieses so dokumentierte Wissen über abstrakte Produktmerkmale, die beim Zusammenwirken verschiedener Produktbestandteile entstehen, könnte zusätzlich zur Steuerung von Rekonfigurationsszenarien in der Werkstatt genutzt werden. Dazu könnte ein Kunde ein abstraktes Produktmerkmal in seinem Rekonfigurationsziel angeben, woraufhin die Werkstatt dieses bei der Rekonfiguration berücksichtigen würde. Dieser Fall ist grundsätzlich durch das Domänenmodell abgebildet, wurde aber mit den konkret vorgestellten Methoden zur Abhängigkeitsdokumenation nicht ermöglicht. Da so aber Kundenwünsche detaillierter berücksichtigt werden könnten und Funktionsupdates wie aus der PC-Branche bekannt möglich würden, erscheinen weitere Arbeiten an dieser Stelle sinnvoll.

3. Die Optimierungsvorschläge im Rekonfigurationstestsystem wurden mit der vereinfachenden Annahme der Rekonfiguration als modellbasierter Diagnose erstellt. Dazu muss die Zielkonfiguration explizit durch den Anwender angegeben werden. Eine mögliche Erweiterung hier könnte diese Annahme eliminieren und eine Optimierung allein basierend auf der Vorgabe des Ausgangsprodukts und des zu flas-

henden Steuergeräts vornehmen. Dabei sollten die statistischen Daten über Flashhäufigkeiten genutzt werden. Dazu müsste zunächst ein Redundanzmodell für die Überprüfung der Korrektheit der enstehenden Konfigurationen geschaffen werden. Weiterhin müsste die Suche nach potenziellen Zielkonfigurationen durch Heuristiken eingeschränkt werden, wenn nicht nur im Bereich der bekannten baubaren und abgesicherten Konfigurationen gesucht werden soll. Diese Heuristiken könnten durch die Modellierung sinnvoller Weiterentwicklungsmöglichkeiten für Hard- und Softwarekomponenten bereitgestellt werden.

Literaturverzeichnis

[ADEK05] Volker Arnold, Hendrik Dettmering, Torsten Engel und Andreas Karcher. *Product Lifecycle Management beherrschen: Ein Anwenderhandbuch für den Mittelstand*. Springer, 2005.

[BA02] Stephen P. Berczuk und Brad Appleton. *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*. Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0201741172.

[Bad04] Jens Badstübner. Immer up to date - Volkswagen aktualisiert die Fahrzeugelektronik online. *kfz-betrieb*, 37, 2004.

[BBDS06] M. Bechter, M. Blum, H. Dettmering und B. Stützel. Compatibility models. In *SEAS '06: Proceedings of the 2006 international workshop on Software engineering for automotive systems*, Seiten 5–12. ACM, New York, NY, USA, 2006. ISBN 1-59593-402-2.

[Beu03] Danilo Beuche. *Composition and Construction of Embedded Software Families*. Dissertation, Otto-von-Guericke-Universität Magdeburg, 2003.

[BHST04] Yves Bontemps, Patrick Heymans, Pierre-Yves Schobbens und Jean-Christophe Trigaux. Semantics of FODA feature diagrams. In Tomi Männistö und Jan Bosch (Herausgeber), *Workshop on Software Variability Management for Product Derivation*. 2004.

[Bro06] Manfred Broy. Challenges in automotive software engineering. In *ICSE '06: Proceeding of the 28th international conference on Software engineering*, Seiten 33–42. ACM, New York, NY, USA, 2006. ISBN 1-59593-375-1.

[Cla01] Matthias Clauß. Modeling variability with UML. In *Online Proceedings of the GCSE 2001 Young Researchers Workshop*. 2001.

[CR91] Judy Crow und John Rushby. Model-Based Reconfiguration: Toward an Integration with Diagnosis. In *In Proceedings of the National Conference on Artificial Intelligence*, Seiten 836–841. The MIT Press, 1991.

[CR94] Judy Crow und John Rushby. Model-Based Reconfiguration: Diagnosis and Recovery. NASA Contractor Report 4596, NASA Langley Research Center, Hampton, VA, may 1994. (Work performed by SRI International).

[Cza98] Krzysztof Czarnecki. *Generative Programming - Principles and Techniques of Software Engineering Based On Automated Configuration and Fragment-Based Component Models.* Dissertation, Technical University of Ilmenau, 1998.

[Dar91] Susan Dart. Concepts in configuration management systems. In *Proceedings of the 3rd international workshop on Software configuration management*, Seiten 1–18. ACM Press, New York, NY, USA, 1991. ISBN 0-897914-429-5.

[Dav96] Stanley M. Davis. *Future Perfect.* Addison-Wesley Longman, 10. Auflage, 1996.

[EFM98] Jacky Estublier, Jean-Marie Favre und Philippe Morat. Toward SCM / PDM Integration? In *ECOOP '98: Proceedings of the SCM-8 Symposium on System Configuration Management*, Seiten 75–94. Springer-Verlag, London, UK, 1998. ISBN 3-540-64733-3.

[EHS91] Martin Eigner, Christine Hiller und Stephan Schindewolf. *Engingeering Database. Strategische Komponente in CIM-Konzepten.* Hanser Fachbuchverlag, 1991.

[ES01] Martin Eigner und Ralph Stelzer. *Produktdatenmanagement-Systeme. Ein Leitfaden für Product Development und Life Cycle Management.* Springer, 2001.

[Est00] Jacky Estublier. Software configuration management: a roadmap. In *ICSE - Future of SE Track*, Seiten 279 – 287. 2000.

[EV07] Jacky Estublier und German Vega. Reconciling software configuration management and product data management. In *ESEC-FSE '07: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, Seiten 265–274. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-811-4.

[FFJ00] Alexander Felfernig, Gerhard E. Friedrich und Dietmar Jannach. UML As Domain Specific Language For The Construction Of Knowledge-Based Configuration Systems. *International Journal of Software Engineering and Knowledge Engineering (IJSEKE*, 10:449–469, 2000.

[FFJ01] Alexander Felfernig, Gerhard Friedrich und Dietmar Jannach. Conceptual modeling for configuration of mass-customizable products. *Artificial Intelligence in Engineering*, 15(2):165 – 176, 2001.

[FFJZ01] A. Felfernig, G. Friedrich, D. Jannach und M. Zanker. Intelligent support for interactive configuration of mass-customized products. In *Lecture Notes in Computer Science*, Seiten 746–756. 2001.

[FFSS08] Andreas Falkner, Ingo Feinerer, Gernot Salzer und Gottfried Schenner. Solving Practical Configuration Problems Using UML. In *Workshop on Configuration Systems at ECAI 2008*. 2008.

[FW03] Michael Faulbacher und Gerhard Wagner. Updateprogrammierung entlang der Prozesskette aus Sicht eines Automobilherstellers. In *IIR: Software-Flashen im Kfz*. 2003.

[GBS01] Jilles Van Gurp, Jan Bosch und Mikael Svahnberg. On the Notion of Variability in Software Product Lines. In *WICSA '01: Proceedings of the Working IEEE/IFIP Conference on Software Architecture*. IEEE Computer Society, Washington, DC, USA, 2001. ISBN 0-7695-1360-3.

[Gün95] Andreas Günter. Das Projekt PROKON im Überblick. In Andreas Günter (Herausgeber), *Wissensbasiertes Konfigurieren - Ergebnisse aus dem Projekt PROKON*, Seiten 3–10. infix Verlag, 1995.

[Hei04] Cornelia Heinisch. *Konfigurationsmodell und Architektur für eine automatisierte Software-Aktualisierung von Steuergeräten im Automobil*. Dissertation, Eberhard-Karls-Universität Tübingen, 2004.

[Her01] Martin Hermsen. *Ein Modell zur kundenindividuellen Konfiguration produktnaher Dienstleistungen - Ein Ansatz auf Basis modularer Dienstleistungsobjekte*. Dissertation, Ruhr-Universität Bochum, 2001.

[Her03] Werner Herrmann. Software-Update für BMW-Fahrzeuge. In *IIR: Software-Flashen im Kfz*. 2003.

[HIS09] Herstellerinitiative Software HIS. Kompatibilitätsmanagement. http://www.automotive-his.de/, Januar 2009.

[Ins03a] Institute of Configuration Management. Information Technology Infrastructure Library (ITIL) Relative to CMII. White Paper, 2003.

[Ins03b] Institute of Configuration Management. SEI's Capability Maturity Model Integrated (CMMI) Relative to ICM's CMII. White Paper, 2003.

[Ins03c] Institute of Configuration Management. Software Process Improvement and Capability dEtermination (SPICE) Relative to CMII. White Paper, 2003.

[JG99] U. John und U. Geske. Reconfiguration of technical products using ConBaCon. Technischer Bericht, In Configuration Papers from the AAAI Workshop, AAAI, 1999.

[Joh02] Ulrich John. *Konfiguration und Rekonfiguration mittels Constraint-basierter Modellierung*. Dissertation, Technische Universität Berlin, 2002.

[KCH+90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak und A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technischer Bericht, Carnegie-Mellon University Software Engineering Institute, November 1990.

[KK03] Thilo Kühner und Stefan Kreuser. Prozeßsicheres Flashen in Entwicklung, Produktion und Aftersales. In *IIR: Software-Flashen im Kfz*. 2003.

[Kne07] Ralf Kneuper. *CMMI - Verbesserung von Software- und Systementwicklungsprozessen mit Capability Maturity Model Integration (CMMI-DEV)*. dpunkt.verlag, dritte Auflage, 2007.

[KR99] Ingo Kreuz und Dieter Roller. Knowledge growing old in reconfiguration context. In *Papers from the AAAI Workshop*. 1999.

[Kre06] Carsten Krebs. Eine gut durchdachte Flash-Strategie ist unumgänglich. *ELEKTRONIKPRAXIS*, 9, 2006.

[KSR07] Detlef Kuttig, Christian Scholz und Christiane Richters. Konfigurationsmanagement - Der rote Faden im PLM. *IT&Production*, Seiten 56 – 57, 2007.

[Man05] Peter Manhart. Reconfiguration - A Problem in Search of Solutions. In *Papers from the Configuration Workshop at IJCAI'05*. 2005.

[Man07] Oliver Manicke. Flash-Programmierung von Steuergeräten in Prototypenfahrzeugen. In *27. Tagung - Elektronik im Kraftfahrzeug*. 2007.

[Män98] Tomi Männistö. Towards Management of Evolution in Product Configuration Data Models. Licentiate Thesis, 1998. Helsinki University of Technology.

[MPS96] Tomi Männistö, Hannu Peltonen und Reijo Sulonen. View to product configuration knowledge modelling and evolution. In *Proceedings of the AAAI 1996 Fall Symposium on Configuration*, Seiten 111–118. AAAI Press, 1996.

[MSTS99] T. Männistö, T. Soininen, J. Tiihonen und R. Sulonen. Framework and conceptual model for reconfiguration. In *Configuration Papers from the AAAI Workshop, AAAI Technical Report WS-9905*, Seiten 59 – 64. AAAI Press, 1999.

[MT08] Thomas Müller und Boubacar Traoré. Softwarelogistik - Bereitstellung von konsistenten Softwareständen in der Fahrzeugentwicklung. In *ProSTEP iViP Symposium 2008*. 2008.

[PBKS07] Alexander Pretschner, Manfred Broy, Ingolf H. Krüger und Thomas Stauner. Software Engineering for Automotive Systems: A Roadmap. In *Future of Software Engineering (FOSE '07)*, Seiten 55–71. 2007.

[PBvdL05] Klaus Pohl, Günter Böckle und Frank van der Linden. *Software Product Line Engineering*. Springer, 2005.

[PF89] Erhard Ploedereder und Adel Fergany. The data model of the configuration management assistant (CMA). *SIGSOFT Softw. Eng. Notes*, 14(7):5–14, 1989.

[Pop06] Gunther Popp. *Konfigurationsmanagement mit Subversion, Ant und Maven*. dpunkt.verlag, 2006.

[Pul02] Christoph Puls. *Die Konfigurations- & Verträglichkeitsmatrix als Beitrag zum Management von Konfigurationswissen in KMU*. Dissertation, Eidgenössische Technische Hochschule Zürich, 2002.

[Rai05] Thomas Raith. Diagnose und Flashen im Produktlifecycle. In *6. EUROFORUM-Jahrestagung: Software in Automobil*. 2005.

[San07] Jürgen Sandhop. Strukturierung macht produktiv. *CAD CAM*, 3-4:47 – 49, 2007.

[Sch96] Stephan Schwarze. *Configuration of Multiple-Variant Products - Application Orientation and Vagueness in Customer Requirements*. Dissertation, ETH Zürich, 1996.

[Sch04] Paul Schönsleben. *Integrales Logistikmanagement: Planung und Steuerung der umfassenden Supply Chain*. Springer, vierte Auflage, 2004.

[Sch05] Günther Schuh. *Produktkomplexität managen: Strategien - Methoden - Tools*. Carl Hanser Verlag, zweite Auflage, 2005.

[SHT05] Harry M. Sneed, Martin Hasitschka und Maria-Therese Teichmann. *Software-Produktmanagement*. dpunkt.verlag, erste Auflage, 2005.

[Sin03] Carsten Sinz. *Verifikation regelbasierter Konfigurationssysteme*. Dissertation, Eberhard-Karls-Universität Tübingen, 2003.

[SS08] Thorsten Schulz und Gerhard Schmitt. Diagnose als Kernprozess der Fahrzeugentwicklung. In *2. Tagung Diagnose in mechatronischen Fahrzeugsystemen*. 2008.

[STMS98] Timo Soininen, Juha Tiihonen, Tomi Männistö und Reijo Sulonen. Towards a general ontology of configuration. *Artif. Intell. Eng. Des. Anal. Manuf.*, 12(4):357–372, 1998.

[Sto07] Kai Storjohann. Seamless E/E Product Documentation and Change Management. In *EDM CAE Forum*. 2007.

[Stu08] Markus Stumptner. Reconfiguration from First Principles with a fair bit of pragmatism in the mix. ECAI Configuration Workshop, 2008.

[Sva00] Mikael Svahnberg. *Variability in Evolving Software Product Lines*. Dissertation, Blekinge Institute of Technology, 2000.

[SW98] Markus Stumptner und Franz Wotawa. Model-Based Reconfiguration. In *In Proceedings Artificial Intelligence in Design*, Seiten 45–64. Kluwer Academic Publishers, 1998.

[Wei00] Ingenieurbüro Wolfgang Weiss. *Configuration Management Training*. ESA, 2000.

[Wil99] Wolfgang Wilke. *Knowledge Management for Intelligent Sales Support in Electronic Commerce*. Dissertation, Universität Kaiserslautern, 1999.

[Zel97] Andreas Zeller. *Configuration Management with Version Sets*. Dissertation, Technische Universität Braunschweig, 1997.