Kaiser, Marco
Tälerweg 36
07646 Lippersdorf-Erdmannsdorf
Tel.: 036426 / 20286
E-Mail: lochigg@gmx.de
Studiengang: Informatik
Matrikelnummer: 22608

Ableitung einer komponentenbasierten Applikation aus
einem merkmalbasierten Systemfamilienmodell
im Rahmen des Digitalen Videoprojektes

Diplomarbeit

zur Erlangung des akademischen Grades
”Diplom-Informatiker”
an der Fakultät für Informatik und Automatisierung
der Technischen Universität Ilmenau

Fachbereich Prozessinformatik

Betreuende Hochschullehrerin:	Prof. Dr.-Ing. habil. Ilka Philippow
Weiterer Betreuer:	Dr.-Ing. Detlef Streitferdt
Starttermin:	05. Februar 2004
Abgabetermin:	05. August 2004
Inventarisierungsnummer:	2004-08-05/067/IN93/2232

Kurzfassung

Durch die steigende Nachfrage nach individueller Software wird die Forderung nach einem beschleunigten Entwicklungsprozess immer dringlicher. Die Anforderungen des Kunden müssen schnell in eine Anwendung überführbar sein. Einen Ausweg bietet die komponentenbasierte Softwareentwicklung, durch die mit vorgefertigten „Programmstücken“ komplexe Systeme in kurzer Zeit entwickelt werden können. Diese Komponenten enthalten häufig geforderte Funktionen, die in verschiedenen Projekten eingesetzt werden. Einmal entwickelt können die verschiedensten Anwendungen diese Komponenten enthalten.

Der Kunde versteht viele technische Begriffe der aktuellen Programmgenerationen nicht. Der Ansatz der Merkmalmodellierung versucht mit der Begriffswelt des Kunden die Anforderungen eines Systems zu abstrahieren. Die Abstraktion der Eigenschaften einer Anwendung mittels Merkmalen ermöglicht es sowohl Entwicklern als auch Kunden eine gemeinsame Basis der Kommunikation zu schaffen.

Die Verbindung der beiden Ansätze wird den Inhalt dieser Arbeit bilden. Die Anforderungen an ein zu entwickelndes System werden in einem Merkmalmodell festgehalten. Vorhandene Komponenten werden durch Merkmale beschrieben. Auf dieser Grundlage soll eine automatisierte Ableitung erfolgen, die aus bestehenden Komponenten eine den Anforderungen entsprechende Anwendung erzeugt.

Für die Merkmalmodelle wird das FORE-Datenmodell verwendet, ein Ansatz zur Ermittlung von Merkmalen basierend auf Anforderungen. Anhand einer Merkmalauswahl werden entsprechende Komponenten ausgesucht und konfiguriert.

Durch ein verbessertes, methodisches Vorgehen wird die Ableitung komponentenbasierter Applikationen unterstützt. Durch eine prototypische Implementierung werden die Grundzüge des eigenen Lösungsansatzes praktisch umgesetzt sowie die Tragfähigkeit der Lösung überprüft und bewertet.

Der Anspruch der Arbeit besteht darin, komponentenbasierte Anwendungen automatisiert aus den Anforderungen der Kunden zu erstellen.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einleitung

1.1. Motivation

Wo wäre die Softwareentwicklung heute, würde man nicht auf die Erkenntnisse unzähliger Forscher und Entwickler zurückgreifen können? Man stelle sich vor, man müsste für jede neue Anwendung die Algorithmen und Prozesse neu entwickeln. Neue Software wäre oft mit hohen Kosten, mit niedriger Qualität oder zu spät verfügbar. Als vielversprechendster Weg aus dieser allgemein als Software-Krise bezeichneten Situation gilt die Wiederverwendung. Die Verwendung bereits erprobter Bestandteile erhöht die Qualität ebenso wie es die Kosten verringert und die Entwicklungszeit verkürzt ([Sch01], S. 10). Allerdings ist Wiederverwendung nicht gleich Wiederverwendung. Sie reicht von Copy & Paste über Algorithmen, Entwurfmustern, Prozessdiagrammen bis hin zu Klassenbibliotheken und Frameworks. Als erfolgversprechendster Ansatz galt lange die Objektorientierung. Aber durch die technische Ausrichtung der Klassen und die Nichtbeachtung von Ökonomie und Märkten hat sie viele Erwartungen, die in diese Form der Wiederverwendung gesetzt wurden, nicht erfüllen können. Die oft prophezeiten Marktplätze für Objekte konnten sich aufgrund vieler Probleme nicht durchsetzen. Ein großes Problem ist die rein technische Definition der Objekte. Sie enthält ungenügende Hinweise zum Grad der Abhängigkeit zu anderen Objekten sowie zur Art und Weise, in der sie mit anderen Objekten zusammengefügt werden können. Aufgrund dieser Schwächen konnten Objekte nur schwer vermarktet werden; rein technische Beschreibungen reichen nicht aus, Märkte zu erschaffen. Die ursprüngliche Idee hinter diesen Marktplätzen kann man als gescheitert ansehen ([Szy02], S.11).

Als ein erfolgversprechender Ansatz, die Wiederverwendung im großem Maßstab zu etablieren, gilt die Verwendung von Komponenten. Doch weshalb sollte man Komponenten einsetzen? Eine oft zitierte Antwort, weshalb Komponenten der Ausweg aus der Softwarekrise seien, ist: Alle anderen Ingenieurdisziplinen, die Komponenten eingeführt haben, als sie ausgereift waren, tun dies auch noch heute ([Szy02], S. 4). Als Beispiele sind hier die Automobilindustrie und die Bauwirtschaft genannt. Doch ganz so einfach ist die Antwort nicht. Materielle Komponenten können nicht ohne weiteres kopiert werden, besitzen einen reellen Wert und unterliegen keiner schnellen und stetigen Veränderung wie Software.

Das Konzept der komponentenbasierten Software orientiert sich bereits stark am Kundenwunsch, d.h. einfacher Einsatz, schneller Austausch und Erweiterbarkeit von Software. Weiterhin soll sich ein Kunde eine Applikation nach seinen Wünschen zusammenstellen können. Um dies zu erlauben, könnte man eine Applikation aus „Bausteinen“ bzw. Komponenten herstellen, die dem Kunden eine Software ermöglicht, welche unter Berücksichtigung des Aufwandes seinen Ansprüchen so nahe wie möglich kommt. An diesem Punkt setzt die komponentenorientierte Softwarenentwicklung an. Dabei ergibt sich folgende Fragestellung: Woher weiß ein Kunde, welche Komponente am besten seinen Bedürfnissen entspricht bzw. welche Komponente soll ein Verkäufer seinen Kunden empfehlen?
Ein Kunde hat, wenn er eine Software einsetzen will, bereits eine Vorstellung, welchen Anforderungen sie genügen soll. Aus diesen Anforderungen lassen sich (standardisierte) Merkmale ableiten, die das Endprodukt besitzen soll. Anhand dieser Merkmale sollte es möglich sein, eine komponentenbasierte Software abzuleiten, die einerseits durch Wiederverwendung kostengünstig und qualitativ hochwertig ist, in einem kleinen Zeitrahmen erstellt wird sowie andererseits den individuellen Wünschen des Kunden entspricht. Diese Arbeit wird einen Ansatz zur automatisierten und kundenorientierten Ableitung von Applikationen liefern.

1.2. Zielsetzung

Unter Verwendung eines komponentenbasierten Ansatzes soll eine Applikation abgeleitet werden. Die Konfiguration der Ableitung soll sich am Vokabular und dem Verständnis des späteren Nutzers orientieren. Der Kunde wählt für ihn bedeutsame Merkmale aus, die seinen Anforderungen an die Applikation entsprechen. Anhand der Merkmale werden die Komponenten ausgewählt, die in das Endsystem integriert werden sollen.
Durch die Verbindung eines Merkmalmodells mit einem neu entwickelten Komponentenmodell soll eine automatisierte Ableitung ermöglicht werden. Dazu werden die Beziehungen zwischen Merkmalen und Komponenten untersucht. Das Komponentenmodell mit den zugeordneten Merkmalen bildet somit den Ausgangspunkt der Ableitung in eine komponentenbasierte Applikation.
Das Ergebnis ist ein verbesserter Entwicklungsprozess zur Systemfamilienentwicklung.

1.3. Vorgehensweise und Aufbau

Nach den einleitenden Worten im 1. Kapitel zur Motivation, Zielsetzung und Vorgehensweise wird im 2. Kapitel zunächst auf den derzeitigen technischen Stand eingegangen. Besondere Beachtung findet hier die Diskussion und Definition der zentralen Begriffe *Merkmal* und *Komponente*. Danach wird der FORE-Ansatz erläutert. Ein mit Hilfe des FORE-Ansatzes zur Anforderungsanalyse und Merkmalmodellierung erzeugtes Datenmodell bildet die Grundlage, aus der die Applikation abgeleitet wird. Im darauf folgenden Abschnitt werden verschiedene bestehende Ableitungskonzepte vorgestellt und analysiert. Den Abschluss des Kapitels bildet die präzisierte Problemstellung. Hier werden die Probleme zusammengefasst und daraus die Aufgabe für die vorliegende Arbeit erstellt.

Im 3. Kapitel wird das Digitale Videoprojekt vorgestellt. Dieses Projekt dient als Fallbeispiel. Es wird zur Überprüfung des Ansatzes im Rahmen der Beispielimplementierung verwendet.

Das 4. Kapitel – „Eigener Ansatz“ stellt eine Herangehensweise vor, mit der komponentenbasierte Applikationen auf Grundlage einer Merkmalauswahl automatisiert abgeleitet werden können. Dazu werden zunächst die Beziehungen zwischen den Komponenten und Merkmalen genauer untersucht. Im Anschluss daran stehen die Anforderungen an die Komponenten im Mittelpunkt. Daraus entsteht im nächsten Unterpunkt ein Modell, mit dem alle wichtigen Eigenschaften der Komponenten erfasst werden können. Das Hauptanliegen des Kapitels besteht jedoch in der Auswahl der Komponenten für die Applikation. Es wird eine Methode vorgestellt, die eine automatisierte Ableitung ermöglichen soll. Weiterhin wird ein Systemmodell entwickelt, dass die Konfiguration der Ableitung beinhalten soll. Zum Schluss wird noch betrachtet, wie man Anwendungen, die mit dieser Methode entwickelt wurden, in späteren Phasen modifizieren kann.

Das 5. Kapitel widmet sich der Beispielimplementierung. Hier wird für das Digitale Videoprojekt ein Werkzeug entwickelt, mit dessen Hilfe eine automatisierte Ableitung möglich ist.

Im 6. Kapitel wird aufgrund der Erfahrungen mit der Beispielimplementierung eine Wertung des Ansatzes vorgenommen und ein Ausblick auf die zukünftigen Forschungsmöglichkeiten gegeben.

Das 7. und letzte Kapitel gibt eine Zusammenfassung der Arbeit.

2. Stand der Technik

Im folgenden Kapitel „Begriffsklärung“ werden die Begriffe *Merkmal* und *Komponente* diskutiert und definiert. Sie sind die zentralen Punkte dieser Arbeit und bedürfen einer klaren Abgrenzung. Sowohl Merkmal als auch Komponente sind in vielen Branchen verbreitete Begriffe und jeweils abweichend definiert.
Daran anschließend folgt ein Überblick zu FORE. Dabei handelt es sich um eine auf Systemfamilien bezogene Methode zum Requirements Engineering. Neben der generellen Darstellung des Ansatzes wird das aus FORE erzeugte Datenmodell vorgestellt. Es bildet den Ausgangspunkt bei der Ableitung der Applikation. Mit seiner Hilfe lassen sich die Anforderungen eines Kunden in einem Modell erfassen und Varianten innerhalb der Systemfamilie zuordnen.
Im Anschluss daran werden Entscheidungsmethoden vorgestellt, die zur Auswahl von Komponenten für eine Applikation genutzt werden können.
Das nächste Kapitel beschäftigt sich mit den bestehenden Ableitungsformen. Die Vorstellung der verschiedenen Ableitungsprozesse soll verdeutlichen, worin die Probleme der herkömmlichen Ansätze bestehen.
Im letzten Kapitel dieses Abschnittes werden die aufgezeigten Probleme zu einer präzisierten Problemstellung zusammengefasst. Sie bildet die Grundlage dieser Arbeit.

2.1. Begriffsklärung

Um den Rahmen der Arbeit einzugrenzen und ein gemeinsames Verständnis für die verwendeten Begriffe zu schaffen, werden zunächst die zentralen Begriffe der Arbeit erklärt und definiert. Dabei wird im besonderen auf die Begriffe *Merkmal* und *Komponente* eingegangen, da dafür in der Literatur verschiedene Auslegungen existieren, die mit den hier verwendeten Auslegungen nicht immer übereinstimmen. In der Zusammenfassung dieses Kapitels werden die verwendeten Definitionen noch einmal kurz darlegt.

2.1.1. Definition: Merkmal

Das Merkmal ist ein in vielen Branchen verbreiteter Begriff. Allgemein bezeichnet es eine Eigenschaft eines Objektes (hier nicht im Zusammenhang mit der Objektorientierung), anhand der man ein Objekt von einem anderen unterscheiden kann. Was aber ist ein Merkmal bezogen auf komponentenbasierte Software?

In der UML existiert folgende Definition von einem Merkmal:

> „A feature is a property, like operation or attribute, which is encapsulated within another entity, such as an interface, a class, or a data type.” [OMG01]

Ein Merkmal ist demnach eine Eigenschaft, die in einem anderen Element gekapselt ist. Dabei wird keine Unterscheidung getroffen, für wen diese Eigenschaft von Relevanz ist. Auch werden anscheinend für Merkmale und Elemente ausschließlich technische Aspekte berücksichtigt. Die Beschreibung eines Merkmals in der UML ist zu allgemein. Ein Beispiel: Wenn sich Konsumenten über ihr neues Möbelstück unterhalten, wird nicht verglichen, welcher Leim, wie viele Dübel oder welcher Holztyp verwendet wurde. Eine solche Unterhaltung wird eher in Richtung von für den Konsumenten wichtigen Merkmalen gehen. Die Farbe, die Maße oder die Anzahl der Schubladen werden beim Käufer mehr im Vordergrund stehen, als die einzelnen technischen Bestandteile eines Schrankes. Ein Konsument wählt demnach Objekte nach Merkmalen aus, welche für ihn von Bedeutung sind und seinen Anforderungen entsprechen. Die Definition eines Merkmals sollte sich, wenn sie vom Konsumenten verstanden werden soll, an seinen Vorstellungen orientieren.

Zwei Definitionen, welche dies berücksichtigen, sind folgende:

> „A feature is a product characteristic that users and customers view as important in describing and distinguishing members of the product-line.” ([Hei01], S. 408)

oder

> „Anforderungen an das System werden von Merkmalen überblicksartig zusammengefasst, wobei durch Merkmale die Variabilität des Software Systems modelliert wird. Ein Merkmal repräsentiert einen Aspekt, der für einen Kunden wertvoll ist und durch ein einzelnes Wort ausgedrückt wird. [...]“ ([Str04], S. 98)

Ein Merkmal dient also dazu, die Anforderungen eines Kunden an ein Produkt zu beschreiben und um es von anderen Produkten zu unterscheiden. Wichtig dabei ist, das die Terminologie des Kunden verwendet wird. Er kann also mit jeden Merkmal etwas verbinden. Durch Untersuchungen von [Cza98], [Kan98] als auch [Str04] wurde gezeigt, dass die Beschreibung eines Systems mittels Merkmale durch den Kunden schneller und klarer verstanden wird. In den Definitionen werden auch die technischen Aspekte nicht ausgeschlossen. Es wird demnach vorausgesetzt, dass ein Kunde auch ein grundlegendes Domänenwissen besitzt. So sind technische Aspekte, wie zum Beispiel das Betriebssystem beim Einsatz von Software, als vorhandenes Wissen beim Kunden vorauszusetzen. Auch der umgekehrte Fall kann eintreten. Der Kunde kann über ein fundiertes

Wissen der Domäne verfügen. Wenn der Kunde im vorigen Beispiel aus der Möbelbranche kommt, kann er durchaus Interesse daran haben, welche Dübel im Schrank verwendet wurden. Der Verkäufer wird ihm allerdings für ein Serienprodukt kaum die Option einräumen, andere Dübel zu verwenden. Hier muss der Kunde sich für eine Maßanfertigung entscheiden. Es muss also bei den Merkmalen ein Grad der Verfeinerung gefunden werden, der eine möglichst große Kundenmenge anspricht und dabei weder zu grob noch zu detailliert wirkt.

In beiden Definitionen wird der „Merkmal"-Begriff noch weiter verfeinert. Bei [Hei01] findet eine Unterscheidung hinsichtlich Requirements (spezifisch, optional, alternativ), Product Characteristics und Implementation Characteristics statt. Diese Untergliederung hat sicherlich in anderen Betrachtungsweisen ihre Berechtigung, ist aber für diesen in einem später folgenden Kapitel erläuterten Ansatz der Komponentenbeschreibung ungeeignet. Der hier gewählte Ansatz baut auf der Unterscheidung zwischen Funktion, Parameter und Schnittstelle auf.

Eine genauere Differenzierung wird dagegen in [Str04] vorgenommen. Im Rahmen des *Family Oriented Requirements Engineering* (FORE) wird folgende Einteilung getroffen:

> „1. *Funktionale Merkmale* drücken ein Verhalten des Systems oder eine mögliche Interaktion der Benutzer mit dem System aus.
>
> *2. Schnittstellen-Merkmale* drücken die Integration von Standards oder Subsystemen in das Produkt aus.
>
> *3. Parameter-Merkmale* drücken Zahlenwerte, in Zahlenwerten darstellbare Umgebungseigenschaften oder nicht-funktionale Eigenschaften aus. Sie haben jeweils einen voreingestellten Wert.
>
> 4. *Strukturierungsmerkmale* dienen lediglich der Strukturierung des zu entwickelnden Modells. Sie können nicht ausgewählt werden und werden im Verlauf der Modellierung genutzt, um eine Reihe von Merkmalen in einem Teilbaum unterhalb des *Strukturierungsmerkmals* durch einen leicht verständlichen Oberbegriff logisch zusammenzufassen." ([Str04], S.99)

Diese Untergliederung kann gut für eine kundenorientierte Charakterisierung von Komponenten verwenden werden. Deshalb bauen die zukünftigen Betrachtungen auf der Merkmal-Definition von [Str04] auf.

Ein Merkmal lässt sich anhand verschiedener Attribute genauer spezifizieren. Im trivialen Fall sind dies der eindeutige Name, ein Wert, den das Merkmal annehmen kann und

natürlich noch der Typ dieses Wertes. Weiterhin können noch eine Reihe weiterer Attribute benannt werden, die hier aber nicht so sehr von Bedeutung sind. Als Anregung sei hier auf die Attribute des Merkmalmodells in [Str04] bzw. auf die Merkmal-Definition von eClass [eClass00] verwiesen.
Die Zusammenhänge zwischen den Komponenten und ihren Merkmalen werden im Kapitel „Komponenten und ihre Merkmale“ genauer untersucht.

2.1.2. Definition: Komponente

Zunächst wird die verbreitete Vorstellung einer Komponente umrissen und bezüglich anderer Formen der Softwarewiederverwendung abgrenzt. So ist nicht alles, was man wiederverwenden kann, eine Komponente. Was daher unter einer Komponente im weiteren und engeren Sinne verstanden wird, ist ausführlich in den folgenden Abschnitten dargelegt.

2.1.2.1. Die Softwarekomponente in der Literatur

Der Begriff der Komponente ist in der Softwarenentwicklung nicht klar definiert, es existieren eine Vielzahl von Definitionen und Abgrenzungen. Laut Duden handelt es sich bei einer Komponente um den Bestandteil eines Ganzen; die Verbindung mehrerer Komponenten heißt Komposition.
Es gibt eine Vielzahl von Komponentendefinitionen in der Literatur der Softwareentwicklung ([Szy02], S. 34 ff; [Atk02], S. 67ff; [Hei01], S. 6ff; [Sch01], S. 41ff; [Wal02] S. 10; s.a. [Szy02], S.195 ff.). Sie überschneiden sich, sind teils widersprüchlich oder die Komponente wird als nicht definierbar bezeichnet ([Cza00] in [Szy02], S. 201).

Trotz allem lassen sich aus diesen Definitionen eine Reihe von gemeinsamen Eigenschaften ableiten, die eine Komponente auszeichnen:

- Eine Softwarekomponente ist im Gegensatz zu den aus anderen Branchen (z.B. Automobilbau, Bauwirtschaft) bekannten Komponenten immateriell. Bei einer Softwarekomponente sind alle Instanzen einer Komponente identisch, d.h. sie besitzen das gleiche Verhalten aber auch die gleichen Fehler. Instanzen materieller Komponenten unterscheiden sich dagegen in vielen Bereichen. Beispielsweise kann sich deren Verhalten im Lauf der Zeit durch Verschleiß ändern oder ihre Schnittstellen variieren produktionsbedingt innerhalb festgelegter Toleranzgrenzen. Bei Software hingegen gibt es keine Probleme bei der Vervielfältigung von Komponenten.
- Eine Komponente ist ein Stück Software, das über eine klar definierte Schnittstelle verfügt, über die es mit anderen Komponenten kommunizieren kann. Die Komponente kann Schnittstellen zur Verfügung stellen oder von anderen Komponenten zur Verfügung gestellte Schnittstellen einbinden. Durch die Standardisierung von Schnittstellen können Komponenten aus verschiedenen Quellen die gleiche Funktionalität unter anderen Aspekten (z.B. Performance, Skalierbarkeit,...) anbieten.
- Weiterhin kann eine Softwarekomponente separat entwickelt werden. Dabei beinhaltet eine Komponente eine abgegrenzte anwendungsbezogene Funktionaltät, die eine eigene Wartung ermöglicht. Es ist somit möglich, Teile einer Applikation zu aktualisieren ohne die komplette Applikation auszutauschen (Evolution anstatt Revolution). Dies ist vor allem in komplexen Anwendungen von Vorteil, da der Komponentenaustausch es ermöglicht, technologisch auf aktuellem Stand zu bleiben.
- Die Verwendung von Komponenten ermöglicht die Erzeugung immer neuer unterschiedlicher Kompositionen. Durch den wiederholten Einsatz einer Komponente in verschiedenen Systemen werden Entwicklungszeit und Kosten gespart. Die gesparten Kosten können so in die Optimierung und die Qualitätssicherung investiert werden. Somit sind Komponenten beim Einsatz in verschiedenen Applikationen kostengünstiger und leichter zu warten als monolithische Systeme.

Ebenso wie eine Reihe von Übereinstimmungen in den zahlreichen Definitionen zu finden sind, lassen sich noch wesentlich mehr Unterschiede feststellen. Diese Differenzen zu untersuchen und zu bewerten würde aber den Rahmen dieser Arbeit sprengen. Im

weiteren Verlauf dieser Arbeit wird die Komponentendefinition von Szyperski verwendet, der in [Szy02] auch andere Definitionen untersucht und bewertet hat.

2.1.2.2. Die Komponente nach Szyperski

Die vorgenannten Punkte sind zwar Gemeinsamkeiten für alle Softwarekomponenten, beschreiben diese aber nicht präzise genug. Szyperski hat meiner Ansicht nach den Begriff der „Komponente“ sehr gründlich untersucht und bewertet. Seine Definition über Art und Wesen von Komponenten deckt sich am besten mit dem hier gezeigten Ansatz. Deshalb wird in dieser Arbeit der Begriff „Komponente“ in der Definition von Szyperski verwendet:

> „A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.” ([Szy02], S.41)

Diese Definition enthält sowohl technische als auch wirtschaftliche Kernaussagen, die das Wesen der Komponente beschreiben:

- Eine Komponente wurde speziell zum Einsatz in Kompositionen entwickelt. Dabei soll sie mit ihrer Funktionalität unterschiedliche Applikationen erweitern können. Sie ist also zur Wiederverwendung vorgesehen. Durch den wiederholten Einsatz von erprobten Komponenten kann so ein hohes Maß an Qualität bei einem geringen Zeit- und Kostenaufwand sichergestellt werden.
- Die Schnittstellen der Komponente, über die sie mit den anderen Teilen der Applikation kommuniziert, wurden verbindlich in einer Spezifikation festgelegt. Wichtig ist hierbei, dass sich verschiedene Komponenten, die die gleiche Schnittstelle nutzen, auch auf die gleiche Version der Spezifikation beziehen. Diese Spezifikation umfasst sowohl funktionale Aspekte (Syntax, Semantik) als auch „extra-funktionale“ Aspekte (z.B. QualityofService-Garantien). Auf diese Weise soll die Austauschbarkeit der Komponente gewährleistet bleiben. ([Szy02], S. 50ff)
- Die einzige Abhängigkeit einer Komponente zu einem System besteht in ihren Ressourcen. Die Implementation der Komponente ist unveränderlich, ihre Funktionalität kann aber mittels Ressourcen für ein System konfiguriert werden. Dazu muss die Komponente aber nicht neu kompiliert werden. Das bedeutet, dass veränderliche Ressourcen nicht Bestandteil einer Komponente sind, aber diese

ihr Verhalten modifizieren können. ([Szy02], S. 564)
Ihre Unabhängigkeit erstreckt sich von der Entwicklung über Testbarkeit und Beschaffung bis hin zur Installation. Die Funktionalität eines Gesamtsystems ist somit aus den Funktionen der verwendeten Komponenten ableitbar. Deshalb ist es wichtig, dass eine Komponente hinsichtlich ihrer Schnittstellendefinition genau analysiert und getestet wird, um ein reibungsloses Zusammenspiel mit anderen Teilen der Applikation zu gewährleisten ([Szy02], S. 555). Besonders bei komplexeren Applikationen ist das Zusammenspiel der Komponenten von entscheidender Bedeutung. Eine Komponente darf die Funktionalität einer anderen Komponente nur in soweit beeinflussen, wie es in ihrer Spezifikation festgeschrieben ist.

- Komponenten können unabhängig installiert werden, d.h. sie können auch nach der bereits abgeschlossenen Installation der Applikation hinzugefügt, aktualisiert oder entfernt werden. Eine Komponente wird stets komplett und nie partiell installiert. Das Einbinden der Komponenten können Dritte übernehmen, die keinerlei Wissen über den internen Aufbau und die Funktionsweise der Komponente verfügen. Somit ist es möglich, den Markt für wiederverwendbare Software zu etablieren, der schon durch die Einführung der Objektorientierung entstehen sollte. (Das Scheitern ist u.a. darauf zurückzuführen, dass die Objektorientierung Klassen rein technisch definierte und ökonomische Aspekte und deren technischen Konsequenzen weitgehend ignorierte ([Szy02], S.11)).

Eine bekannte Form der Komponente ist das Plug-In, wie man es von Internetbrowsern kennt, aber auch Hilfsprogramme und Werkzeuge von Betriebssystemen gehören dazu.

2.1.2.3. Abgrenzung

Es gibt viele Ansätze einmal entwickelte Software wiederzuverwenden, aber nicht alle Softwarestücke sind Komponenten. Deshalb wird hier der Begriff der Komponente entsprechend eingrenzt.

Klassen und *Objekte* sind keine Komponenten. Viel mehr sind Komponenten unabhängig von der Art der Implementierung, ob nun objektorientiert, funktional oder strukturiert. Objekte können in Komponenten erzeugt, modifiziert und gelöscht werden oder sogar die Komponente dauerhaft verlassen. ([Szy02], S. 38)

Frameworks (z. B. Sun's Java Development Kit oder Microsoft's .Net) und Class Libraries (z. B. Microsoft's MFC) sind ebenfalls keine Komponenten. In erster

Linie handelt es sich bei diesen Softwarestücken um Entwicklerwerkzeuge. Sie werden zur Kompilierzeit eingebunden und können damit nicht unabhängig von der Anwendung von Dritten installiert werden. Weiterhin ist die Interoperabilität von mehreren Frameworks schwierig. Sie sind darauf ausgerichtet, die Zusammenarbeit der einzelnen Teile einer Applikation zu koordinieren und zu regulieren. ([Szy02] S. 164f)
Module, wie sie z. B. in Oberon, Modula-3 und C# verwendet werden, haben mit den Komponenten die meisten Gemeinsamkeiten. Sie können separat entwickelt und getestet werden, können Klassen als auch Funktionen enthalten. Man kann sie schon fast als Komponenten betrachten. Allerdings fehlt ihnen eine wichtige Eigenschaft der Komponenten. Abgesehen von Konstanten im Sourcecode, gibt es keine dauerhaft unveränderbaren Ressourcen, die aber ein wesentliches Merkmal der Komponenten sind. Ohne die Möglichkeit, ein Modul entsprechend den Anforderungen des Systems zu konfigurieren, verringert sich die Wiederverwendbarkeit und die Reichweite im Anwendungsgebiet. Ebenso gibt es keinen Ansatz zur gezielten Vermarktung von Modulen. ([Szy02], S.39f)

2.1.2.4. Schnittstellen von Komponenten

Für das Zusammenstellen einer Anwendung sind die Schnittstellen von entscheidender Bedeutung. Sie ermöglichen einer Komponente die Kommunikation mit ihrer Umgebung. Das können z. B. andere Komponenten oder Anwendungsnutzer sein. Es können insgesamt drei verschiedene Typen von Schnittstellen unterschieden werden.
Die ersten beiden Typen dienen der Verbindung mit anderen Programmteilen. Je nachdem, ob eine Komponente eine Schnittstelle zur Verfügung stellt oder einbindet, werden diese nach *bereitstellen* (englisch: *provide*) und *benötigen* (englisch: *require*) unterschieden. Es ist zu beachten, dass nicht jede Komponente mit jeder anderen kommunizieren kann. Damit zwei Komponenten untereinander Daten austauschen können, müssen die verwendeten Schnittstellen zueinander passen. So muss die einbindende Komponente eine Schnittstelle bereitstellen, die von der eingebundenen Komponente unterstützt wird.
Eine Komponente kann beide Arten von Schnittstellen besitzen, die jeweils verschiedene Komponenten verbinden. Demzufolge kann es vorkommen, das eine Schnittstelle sowohl Dienste zur Verfügung stellt als auch nutzt. In einem solchen Fall sollte diese Schnittstelle in einer Überarbeitung der Komponente klar in *providing* und *requiring* geteilt werden ([Bos00], S. 223). Dies kann z. B. mittels Wrapper-Komponenten ge-

schehen. Sie stellen jeweils ein *providing* und *requiring* Interface für die zu zerlegende Schnittstelle bereit.

Als dritten Typ von Schnittstellen kann man die Konfigurationsschnittstellen nennen ([Bos00], S. 221). Sie dienen der Anpassung der Komponente an die Applikation, in der sie eingesetzt werden soll. Die Konfiguration wird im Normalfall durch den Entwickler oder den Anwender vorgenommen, der die Applikation zusammensetzt bzw. nutzt.

2.1.3. Zusammenfassung der Definitionen

Nach der Diskussion der Merkmale und Komponenten werden hier noch einmal die wichtigsten Punkte zusammenfasst.

In der Definition nach [Str04] verkörpert ein Merkmal eine für den Kunden wichtige Eigenschaft eines Systems. Das Merkmal wird durch einen für den Kunden verständlichen Ausdruck repräsentiert. Sie bilden die Anforderungen eines Kunden an ein System ab. Merkmale können in verschiedene Kategorien eingeteilt werden, die eine Modellierung und Verarbeitung vereinfachen. Durch die Unterteilung der Merkmale in Funktion, Schnittstelle und Parameter lassen sich Komponenten abbilden und spezifizieren. Die Zusammenhänge werden im Kapitel „Komponenten und ihre Merkmale“ konkreter erläutert.

Für die Komponenten wird die Definition nach Szyperski verwendet. Demnach sind Komponenten Bestandteile einer Applikation, die über festgeschriebene Schnittstellen mit den anderen Teilen der Software kommunizieren. Sie werden ausschließlich über ihre Ressourcen an ein System angepasst, die Codebasis bleibt unverändert. Komponenten können weiterhin separat installiert und gewartet werden. Das Zusammenfügen der Komponenten zu einem System kann ohne das Wissen über ihre interne Funktionsweise geschehen. Diese Definition einer Komponente entspricht der Vorstellung, dass Applikationen ohne großen Entwicklungsaufwand entsprechend den individuellen Vorstellungen des Kunden entwickelt werden können. Voraussetzung sind bereits bestehende Komponenten, welche die Kundenanforderungen erfüllen.

2.2. FORE

In diesem Kapitel wird das *Family Oriented Requirements Engineering* (FORE, [Str04]) erläutert. Zunächst wird der Umfang von FORE in einem kurzen Überblick dargestellt. In den beiden darauf folgenden Unterkapiteln wird auf das Anforderungsmodell und die erweiterte Merkmalmodellierung eingegangen. Im Anschluss wird der Entwicklungsprozess für Systemfamilien mit FORE dargelegt. Im letzten Unterkapitel wird das im FORE-Ansatz entwickelte Datenmodell genauer beleuchtet. Es bildet den Ausgangspunkt des Ansatzes dieser Arbeit.

2.2.1. Überblick

FORE basiert auf verschiedenen Ansätzen zum Requirements Engineering wie FAST ([Wei99]), FODA ([Kan90]), FeatuRSEB ([Gri98]) und KorbA ([Atk02]). Diese Methoden werden zu einem durchgängigen Prozess integriert. Weiterhin wird die für die Systemfamilienentwicklung wesentliche Merkmalmodellierung erweitert. FORE stellt einen zweigeteilten Lösungsansatz für die Requirements-Engineering-Phase von Systemfamilien vor. Der Ansatz beschreibt sowohl die methodische Vorgehensweise als auch ein Datenmodell der zu entwickelnden Systemfamilie.

2.2.2. Anforderungsmodell

Zunächst werden die für die Systemfamilie relevanten Anforderungen gesammelt und in einem Modell zusammengefasst. Zu den Anforderungen werden die beteiligten Personen mit ihren Rollen, die Dokumentenstruktur der zukünftigen Spezifikationsdokumente und die zwischen den Anforderungen existierende Beziehungen erfasst. Die Erhebung der Anforderungen kann mittels einer *Anforderungskarte* ([Str04], S. 90) erfolgen. Durch die gezielten Fragestellungen können alle für die Anforderung wichtigen Informationen ermittelt werden.

Beteiligte Personen werden in [Str04] als *Stakeholder* bezeichnet. Sie können in einer Hierarchie, die z. B. der Unternehmenshierarchie entspricht, angeordnet werden. Für jede Person sind die wichtigsten Daten als eine Art „Visitenkarte“ erfasst.

Die Dokumentenstruktur einer Spezifikation ist ein Ergebnis der Requirements-Engineering-Phase. Sie ist an die Verhältnisse im Unternehmen angepasst.

Alle Anforderungen werden in einem hierarchischen System weiter verfeinert. Die höchste Hierarchiestufe stellt dabei die ganz allgemeine Beschreibung dar, während mit

zunehmender Tiefe die Anforderungen weiter konkretisiert werden. Zusätzlich kann eine Anforderung von anderen Anforderungen über Parameter abhängig sein. Diese Beziehungen können sowohl verbal als auch formal beschrieben werden. Eine denkbare Beziehung wäre beispielsweise der Ausschluss einer Anforderung durch eine konkurrierende Anforderung. Das Einbringen solcher Anforderungen führt zu Varianten innerhalb der Systemfamilie. Weitere mögliche Beziehungen werden in [Str04] vorgestellt.
Zusätzlich wird zu jeder Anforderung eine Historie geführt. So soll sichergestellt werden, dass einerseits ein älterer Stand jederzeit wiederhergestellt und andererseits ein definierter Stand für weitere Entwicklungsarbeiten verwendetet werden kann. So kann ein System auf einen abgesprochenen Stand entwickelt werden, während das Anforderungsmodell weiterentwickelt wird.
Die Ableitung von Varianten basiert auf der erweiterten Merkmalmodellierung von FORE. Sowohl Anforderungen als auch Merkmale lassen sich einer Systemvariante zuordnen.
Weiterhin kann jeder Anforderung eine Komponente zugeordnet werden. Eine Beschreibung der Komponente ist im FORE-Datenmodell nicht vorgesehen. Es werden aber Ansatzpunkte gegeben, welche Informationen für eine Komponente erfasst werden müssen. Der Komponentenbegriff in FORE ist weiträumiger gefasst, als der in dieser Arbeit verwendete. Im Kapitel „Eigener Ansatz“ wird die Einbindung von Komponenten in das FORE-Datenmodell genauer beleuchtet.

2.2.3. Erweiterte Merkmalmodellierung

Die erweiterte Merkmalmodellierung stellt den Schwerpunkt von FORE dar. Sie erweitert den Modellierungsansatz nach [Kan90]. Nach diesem Ansatz beschreiben Merkmalmodelle die Variabilitäten innerhalb einer Systemfamilie. Die Definition der Merkmalmodelle besitzt jedoch Mängel. Bei der Integration in den Entwicklungsprozess treten somit Probleme auf. Die Beziehungen sind nicht immer eindeutig und lassen sich nur schwer automatisiert überprüfen.
An dieser Stelle setzt FORE an. Das Merkmalmodell bildet eine zusätzliche Abstraktionsschicht zwischen dem Anforderungsmodell und dem Systementwurf (Abb. 2-1). Es ist somit ein Ziel der Anforderungsmodellierung und stellt zugleich den Ausgangspunkt der Systementwicklung, beispielsweise eines UML-Modells, dar.
Wie der Name schon nahe legt, wird in der Merkmalmodellierung eine Systemfamilie anhand ihrer Merkmale modelliert. Es wird dabei zwischen Merkmalen des Systemfamilienkerns und der veränderlichen Anteile unterschieden. Die Kernmerkmale

milienkerns und der veränderlichen Anteile unterschieden. Die Kernmerkmale sind obligatorisch und die Merkmale der variablen Bestandteile optional. Zudem werden den Varianten bzw. dem Kern die enthaltenen Merkmale zugeordnet.

Ähnlich den Anforderungen bestehen zwischen einzelnen Merkmalen Beziehungen, die den Ausschluss oder die Einbeziehung weiterer Merkmale nach sich ziehen. Auch hier gibt es eine Hilfe zur Erhebung der Merkmale in Form einer *Merkmalkarte* ([Str04], S.102).

Die Merkmale werden entsprechend der Klassifizierung unterschieden, welche im Kapitel „Definition: Merkmal“ vorgenommen wurde. Funktionale Merkmale können auf Anforderungen verweisen, deren Funktionen durch die Merkmale abgedeckt werden.

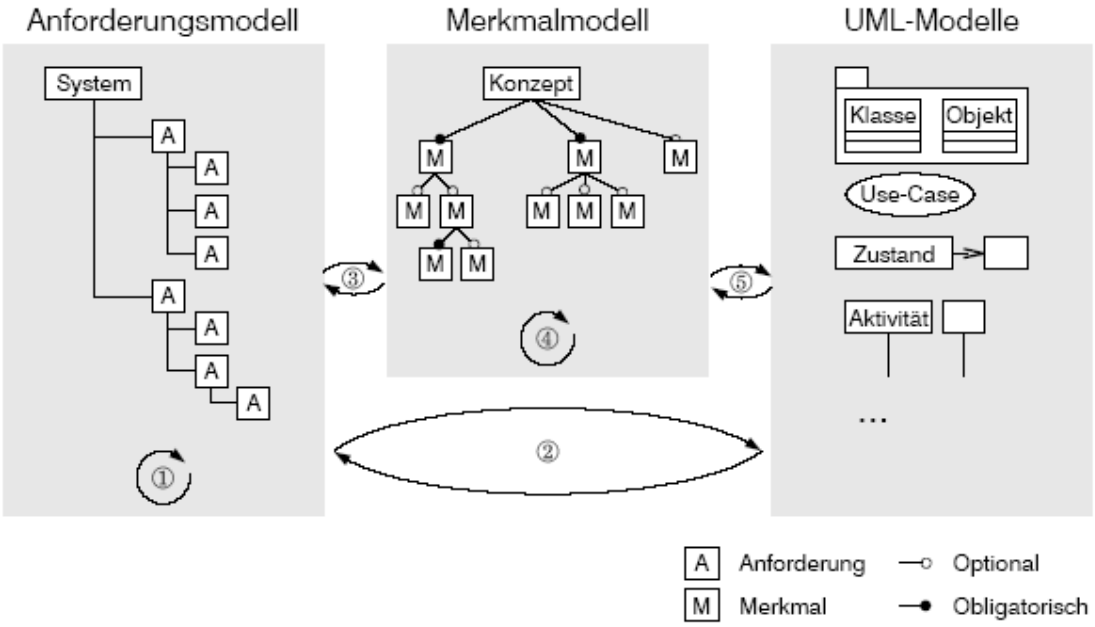

Abb. 2-1: Einordnung des Merkmalmodells ([Str04], S. 100)

Zum besseren Verständnis der Zusammenhänge zwischen den Merkmalen können diese mit einfach grafischen Elementen visualisiert werden. Alle Merkmale werden zudem im *Merkmalmodell* hierarchisch angeordnet. Nach [Str04] ist das Merkmalmodell wie folgt definiert:

> „Ein Merkmalmodell gibt die Merkmale in einer hierarchischen Struktur wieder. Zusätzlich zu den genannten Kategorien der Merkmale gibt es in einem Merkmalmodell auch noch abstrakte Merkmale, die jeweils Konzepte darstellen [...]. Die Wurzel des Baumes ist immer ein Konzept. Die Merkmale des Baumes werden über Beziehungen verbunden [...].“ ([Str04], S. 101)

Die Position eines Merkmals im Hierarchiebaum bestimmt, wie stark es die Architektur der Familie beeinflusst. Der Einfluss sinkt mit der größer werdenden Entfernung zur Wurzel des Baumes. Die Beziehungen zwischen den Merkmalen können nach [Str04] folgende Eigenschaften aufweisen:

1. Merkmal-Untermerkmal-Beziehungen, welche die Hierarchie abbilden. Sie zeigen auf obligatorische bzw. optionale Untermerkmale.
2. Verfeinerung (englisch: *refinement*) bzw. Notwendigkeit (englisch: *requires*) von Merkmalen.
3. Verfeinerung dient der weiteren Detaillierung von Untermerkmalen. Auf diese Weise kann man „ist-ein“ oder „ist-Teil-von“ Beziehungen darstellen.
4. Einschluss (englisch: *requires*) bzw. Ausschluss (englisch: *excludes*) von Merkmalen sowie Mehrfacheinbindung von Merkmalen bzgl. des Elternmerkmals.
5. Mathematische Beziehungen zwischen Merkmalen, insbesondere zwischen Parametermerkmalen
6. Anregungen (englisch: hints) für die Auswahl weiterer Merkmale, die im Auswahlprozess durch den Kunden mit einbezogen werden können.

Mit Hilfe des Merkmalmodells können die Anforderungen an die Systemfamilie durch Merkmale verallgemeinert werden. Es ermöglicht so die Modellierung von Varianten der Familie.

Die Konsistenz der Daten im Merkmalmodell ist für die Ableitung von Varianten von großer Bedeutung. Diese Informationen bestimmen die Architektur des zu erstellenden Systems. Eine automatisierte Prüfung dieser Informationen wird selbst bei kleinen Anwendungen schnell notwendig. Dazu ist eine eindeutige, klar strukturierte Beschreibung der Beziehungen zwischen den Merkmalen unumgänglich. Zu diesem Zweck wurde neben der Beschreibung einer Vielzahl von möglichen Beziehungen auch eine Beschreibungssprache entwickelt, mit der sich auch komplexe Zusammenhänge darstellen lassen. Die *Feature Contraint Language* (FCL) orientiert sich dabei an der in der UML verwendeten *Object Connstraint Language* (OCL). Die Ableitung einer Variante aus einem konsistenten Modell führt zu einem funktionstüchtigen System. Wenn die Konsistenz des Modells nicht gewährleistet ist, kann nicht garantiert werden, dass die Ableitung den Anforderungen eines Kunden entspricht. In FORE wird ein Verfahren zur Überprüfung der Konsistenz eines Modells vorgestellt. Darauf wird in diesem Kapitel aber nicht weiter eingegangen, da es den Rahmen der Arbeit sprengen würde. Der eige-

ne Ansatz basiert auf einem bereits validierten Modell, eine Konsistenzprüfung entfällt somit für diese Arbeit.

2.2.4. Entwicklungsprozess

FORE gliedert sich nahtlos in den Entwicklungsansatz für Systemfamilien ein. Die einzelnen Phasen der Entwicklung sind in Abb. 2-2 dargestellt. In diesem sogenannten *Six-Pack-Modell* sind alle Phasen zur Entwicklung einer Systemfamilie enthalten.
Die drei Phasen im oberen Teil der Abbildung stellen das *Domain Engineering* ([SEI00]) dar. Sie dienen der Entwicklung der eigentlichen Systemfamilie. Als Ergebnis dieser Phasen entstehen eine Reihe von Komponenten, welche die Anforderungen an die Systemfamilie erfüllen.
Die Resultate des Domain Engineering sind im Mittelteil der Abbildung zu sehen. Jede einzelne Phase bringt Ergebnisse hervor, welche den Ausgangspunkt für die Entwicklung einer Applikation bilden. Die daraus entstehende Applikation baut auf der Referenzarchitektur auf.
Der FORE-Ansatz deckt die Analysephase der Entwicklung von Systemfamilien ab (in Abb. 2-2 grau hinterlegt). Der Entwicklungsprozess gliedert sich ebenfalls in eine *Domänen-* und eine *Applikationsanalyse*. Das daraus entstehende Anforderungsmodell nimmt alle aus der Analyse gewonnen Informationen auf.

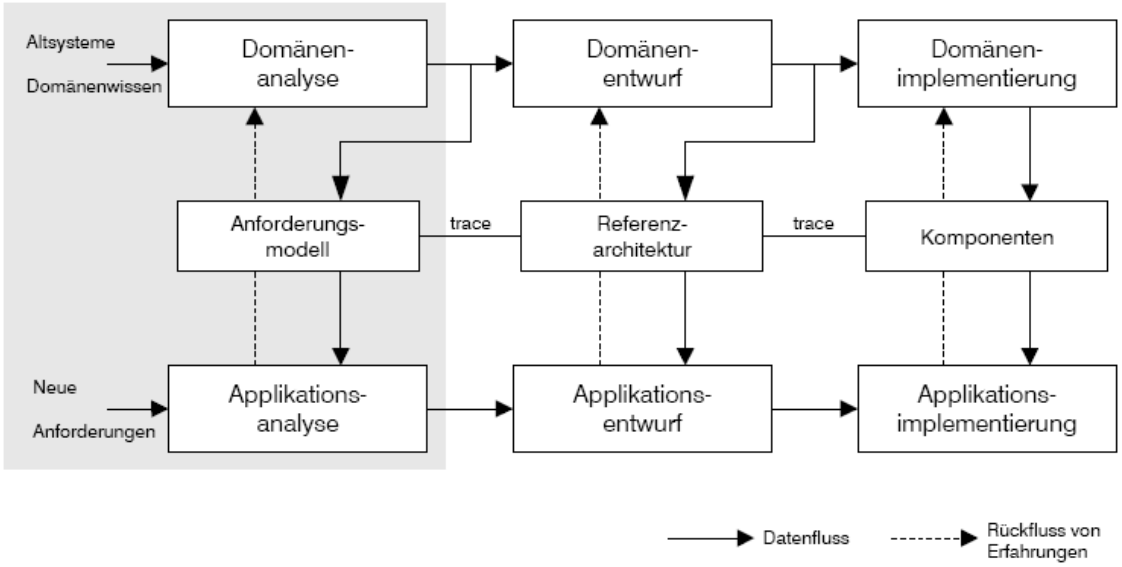

Abb. 2-2: FORE - Einordnung in die Systemfamilienentwicklung ([Str04], S. 125)

Die Domänenanalyse dient dem Requirements Engineering der Systemfamilie. Dazu werden bestehende Altanwendungen und vorhandenes Domänenwissen analysiert. Als

Endergebnis dieser Phase werden die Anforderungen und Merkmale der Systemfamilie im Anforderungsmodell erfasst.
In der Applikationsanalyse werden die Wünsche eines Kunden an eine zu entwickelnde Applikation festgeschrieben. Hier wird ermittelt, inwieweit die Systemfamilie die Anforderungen bereits erfüllt. Werden Anforderungen des Kunden nicht durch die Systemfamilie abgebildet, müssen sie separat entwickelt werden. Auch ein Rückfluss von neuen Anforderungen in die Systemfamilie, welche bis jetzt nicht von der Domänenanalyse erfasst worden, ist möglich.
Die eigentliche Neuerung im FORE-Ansatz bezieht sich auf die erweiterte Merkmalmodellierung im Anforderungsmodell. Für die Prozessschritte der Domänen- und der Applikationsanalyse wurden existierende Ansätze für das FORE-Datenmodell abgewandelt.

2.2.5. Datenmodell

Ziel dieser Arbeit ist die Ableitung einer Applikation aus vorhandenen Komponenten. Die Grundlage der Ableitung wird ein Merkmalmodell eines zu erstellenden Systems bilden. Das Modell wird alle Merkmale enthalten, die das zukünftige System aus Sicht des Endanwenders aufweisen soll. Als Modell wird das Datenmodell von FORE ([Str04], S. 148ff) verwendet. Es beinhaltet nicht nur die Merkmale der zu erstellenden Applikation, sondern auch Informationen zu den Anforderungen und den Merkmalen der gesamten Systemfamilie. Das Modell gliedert sich in mehrere Untermodelle. Im einzelnen besteht es aus folgenden Teilen:

- Anforderungsmodell
- Merkmalmodell
- Systemfamilienmodell
- Dokumentenmodell
- Personenmodell
- Relationen zwischen den einzelnen Elementen.

Im *Anforderungsmodell* werden alle Anforderungen an die Systemfamilie gesammelt und in einer hierarchischen Form weiter verfeinert. Jeder Anforderung sind eine eindeutige Identifikation, eine Dokumentation (Modell, verbal), der Ursprung und die Testvorschriften zugeordnet. Jede Anforderung kann dem Kern der Systemfamilie oder einer seiner Varianten zugeordnet werden. Dies geschieht über die im nachfolgenden Text erläuterten Relationen.

Das *Merkmalmodell* erfasst alle Merkmale einer Systemfamilie in hierarchischer Form. Die Wurzel des Baumes ist ein Konzept, das mit der Systemfamilie gleich zu setzen ist. Unterhalb dieses Konzeptes sind sowohl weitere Teilkonzepte (Strukturmerkmale) als auch die anderen Merkmaltypen (Funktion, Parameter, Schnittstelle) angeordnet.
Die Merkmale dieser Systemfamilie können mittels Beziehungen in Abhängigkeiten zueinander gesetzt werden. Auf diese Verbindungen wird im Absatz über die Relationen genauer eingegangen.
Im *Systemfamilienmodell* fließen die Merkmale der Familie zur Kernapplikation und den dazugehörigen Variationen zusammen. Die Kernapplikation stellt den Rumpf der Systemfamilie dar. In ihr sind ausschließlich die Merkmale enthalten, die für die minimale Konfiguration des Systems notwendig sind. In der minimalen Ausstattung befinden sich im System keine optionalen Merkmale. Unter einer Variation versteht man ein System, in dem optionale oder alternative Merkmale enthalten sind. Sie entsprechen der vom Endanwender gewünschten Applikation.
Alle im Rahmen der Analyse erstellten Spezifikationen sowie sonstige Dokumente werden im *Dokumentenmodell* referenziert. Anforderungen und Merkmale verweisen auf die hier vermerkten Dokumente.
Jedem dieser oben genannten Elemente ist eine *Historie* zugeordnet. So soll gewährleistet werden, das einerseits ältere Versionen des Modells wiederhergestellt werden können und andererseits die Entwicklung von Applikationen sich auf einen definierten Stand berufen kann.
Das *Personenmodell* enthält alle Personen oder Organisationen, die in irgendeiner Weise etwas zum Modell beitragen. Die für jedes Element gepflegte Historie enthält neben dem Datum auch eine Referenz auf die Person sowie den Grund, weshalb eine Änderung vorgenommen wurde.
Abschließend sind noch die *Relationen* zwischen den einzelnen Elementen zu nennen. Mit diesen Beziehungen werden Abhängigkeiten zwischen Merkmalen und Anforderungen modelliert. Jede Relation wird durch einen Ausdruck in der *Feature Constraint Language* kurz FCL ([Str04], S. 122ff) realisiert. Mit Hilfe der FCL kann eine Modellauswahl auf ihre Korrektheit überprüft werden. So kann z. B. kontrolliert werden, ob konkurrierende Merkmale, die einander ausschließen (englisch: *exclude*), nicht im Modell enthalten oder ob Merkmale, die von anderen im Modell aufgeführten Merkmalen benötigt werden (englisch: *require*), vorhanden sind. Auf diese Weise können selbst komplexe Abhängigkeiten dargestellt und überprüft werden.

Im Datenmodell sind alle Anforderungen enthalten, die ein zukünftiges System erfüllen muss. Es lässt jedoch offen, wie daraus eine Anwendung entwickelt wird. Für den komponentenbasierten Ansatz stellt sich hier insbesondere die Frage, wie Komponenten, welche die Anforderungen des Systems erfüllen, zu einer funktionierenden Applikation zusammengefügt werden können.
Ein Datenmodell mit einer Auswahl von Merkmalen, die der Nutzer mit der neu zu erstellenden Applikation verbindet, bildet den Ausgangspunkt der Ableitung. Alle notwendigen Erweiterungen oder Abänderungen am Modell werden im Kapitel „Eigener Ansatz“ erläutert.

2.3. Entscheidungsmethoden

Die in den nachfolgenden Unterkapiteln vorgestellten Entscheidungsmethoden können zur Auswahl von Komponenten genutzt werden. Welche der präsentierten Methoden sich besser für den vorgestellten Ansatz eignet wird im Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden.** „**Fehler! Verweisquelle konnte nicht gefunden werden.**“ untersucht und bewertet.

2.3.1. Entscheidungsmethode MAUT

Hinter der *Multi-Attribute Utility Technique* (MAUT, [Wal02], S. 115ff) verbirgt sich eine Entscheidungshilfe, mit der durch die Wichtung von Eigenschaften Gegenstände vergleichbar gemacht werden können. Im vorliegenden Fall soll sie den Nutzwert einer Komponente für den Kunden ermitteln. Das Wesen der Methode besteht in einer einfachen mathematischen Formel:

$$U_y = \sum_i^n w_i \cdot a_i$$

Die Variable U_y repräsentiert den Nutzen für eine Alternative y, im vorliegenden Fall der Nutzen des Merkmals y. U_y wird aus der Summe des Nutzens der einzelnen Eigenschaften mit ihren Wichtungen gebildet. Der Nutzwert setzt sich aus den Eigenschaften des betrachteten Merkmals (z. B. Wiedergabequalität) und den Eigenschaften der Komponente (z. B. Preis) zusammen.
Die Variable w_i ist die Wichtung der Eigenschaft i, die Summe aller Wichtungen ist genau 1 und jede einzelne Wichtung ist größer 0. Ingesamt werden n Eigenschaften betrachtet. Die Wichtungen müssen aus den Anforderungen des Kunden hergeleitet wer-

den. Werden keine Wichtungen ermittelt, sollte die Wichtung w_i dem Reziprok der Anzahl der Eigenschaften von *n* entsprechen. Ein Beispiel für eine Wichtungsverteilung ist in Abb. 2-3 zu sehen.

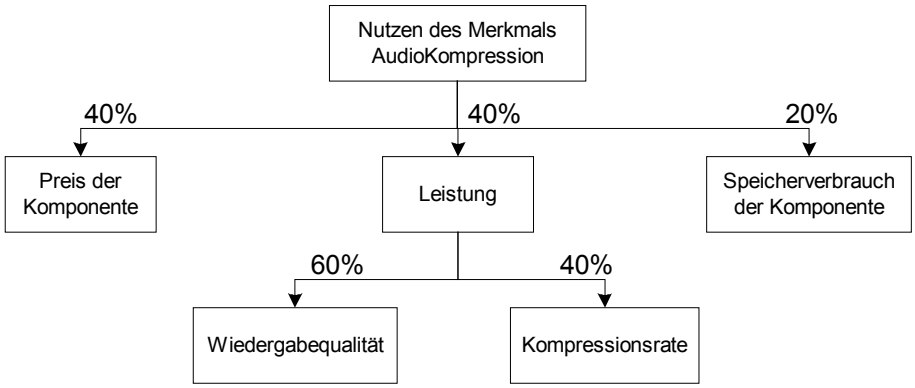

Abb. 2-3: Beispiel für eine Wichtungsverteilung

Abschließend gibt die Variable a_i den Nutzen der Eigenschaft *i* an. Für den Nutzen einer einzelnen Eigenschaft wird ein Wertebereich definiert, der für alle gleich groß ist. Im vorliegendem Fall gilt für alle *a*: $0 \le a \le 1$. Dieser Nutzen kann durch eine Nutzenfunktion bestimmt werden. Diese Funktion kann im einfachen Fall eine lineare Abbildung sein, aber auch kompliziertere Funktionen sind möglich. Hier einige Beispiele zum besseren Verständnis:

$$a_i = 1 - \frac{Speicher_i}{\max(Speicher)} \qquad a_i = \begin{bmatrix} Hersteller_i = "ABC", 1 \\ sonst, 0 \end{bmatrix}$$

$$a_i = \begin{bmatrix} Bitrate_i < 16KBps, 0 \\ 16KBps \le Bitrate_i \le 256KBps, (Bitrate_i - 16)/240 \\ Bitrate_i > 256KBps, 1 \end{bmatrix}$$

Auf diese Weise kann für jedes geforderte funktionale Merkmal der Nutzen innerhalb der Komponente bestimmt werden. Aus den einzelnen Nutzen der Merkmale einer Komponente lässt sich z. B. durch erneute Anwendung der MAUT-Methode oder durch Bildung des Mittelwertes aller Merkmalnutzen der Gesamtnutzen der Komponente ermitteln. Bei erneuter Anwendung der MAUT-Methode repräsentiert *U* dann den Gesamtnutzen der Komponente, *w* die Wichtung eines Merkmals innerhalb der Komponente und *a* den Nutzen eines Merkmals.

Ein Vorteil dieser Methode ist die Erfassung verschiedenster Aspekte einer Komponente. Im Grunde nicht vergleichbare Eigenschaften können bewertet und so verschiedene Komponenten eingeschätzt werden. Auf diese Weise lassen sich komplexe Zusammen-

hänge vereinfachen. Bei einer geeigneten Bewertung sind die Ergebnisse genauer als bei der nachfolgend vorgestellten Prioritätenliste.
Der für diesen Ansatz wesentliche Nachteil liegt in der schwierigen Bewertung der Komponenteneigenschaften. Es müssen Wichtungen und Nutzenfunktionen von Eigenschaften festgelegt werden, die sich mit den Anforderungen des Kunden decken. Die Ermittlung der richtigen Nutzwerte gestaltet sich für den Kunden zeitaufwendig und schwierig. Eine Schwierigkeit besteht in der Ermittlung der individuellen Substitutionsraten. Wie viel würde z. B. ein Kunde mehr bezahlen wollen, um eine höhere Wiedergabequalität bei Videos zu erhalten? Diese Präferenzen unterscheiden sich von Kunde zu Kunde; viele Käufer könnten diese Frage gar nicht beantworten.

2.3.2. Entscheidungsmethode Prioritätenliste

Eine *Prioritätenliste* stellt eine Rangfolge von Eigenschaften dar. Dem Kunden werden die Eigenschaften des Systems präsentiert. Er ordnet sie danach, welche für ihn die größte Bedeutung besitzen. Dazu zählen nicht nur die Eigenschaften der Komponente, sondern auch die der funktionalen Merkmale. Zu jedem Punkt auf der Prioritätenliste muss vermerkt werden, wie dieser behandelt werden soll. Hier gibt es verschiedene Möglichkeiten: Die Eigenschaft soll in der Applikation identisch oder verschieden zu einer Vorgabe, minimal oder maximal sein. Beispiele für Prioritätenlisten könnten so aussehen:

Prioritätenliste A	**Prioritätenliste B**
1. Aufnahmequalität MP3 (KBps) → max	**1.** Hersteller→ = „ABC“
2. CPU-Belastung → min	**2.** Preis → min
3. Preis → min	**3.** Speicherverbrauch → min
[...]	[...]

Bei den Punkten 1 und 2 der Prioritätenliste A handelt es sich um die spezielle Eigenschaft eines Merkmals zur Aufzeichnung von mp3-Audiodaten. Sie bezeichnen somit Eigenschaften von einzelnen Funktionen einer Komponente. Die anderen Punkte sind jeweils allgemeine Eigenschaften einer Komponente. Anhand einer solchen Liste können die Komponenten Punkt für Punkt aussortiert werden bis eine gewünschte Komponente ermittelt wurde.

Die Prioritätenliste zeichnet sich vor allem durch ihre Einfachheit und Verständlichkeit aus. Jeder Kunde weiß bei einem Kauf, was ausschlaggebend für den Erwerb eines Produktes ist. Oft ist es einfach nur der Preis. Möglichst günstig sollen die geforderten Funktionen erworben werden. Bei anderen Käufern steht die Marke oder die Qualität im Vordergrund. Anhand von verständlichen Eigenschaften kann er hier seine Prioritäten setzen. Die Ermittlung dieser Prioritäten ist weder für den Kunden noch für den Anbieter aufwendig.
Der Nachteil dieser schnellen Anforderungserfassung und -verarbeitung liegt im Endergebnis der Auswahl. Die Kombination der einzelnen Prioritäten führt nicht immer zu einem optimalen Ergebnis. Weniger wichtige Eigenschaften können durch wichtigere Eigenschaften bei der Auswahl komplett verdrängt werden.

2.4. Vorstellung von Vorgehensmodellen

In diesem Kapitel werden verbreitete Vorgehensmodelle miteinander verglichen und bewertet. Jedes Modell hat seine Existenzberechtigung, aber nicht alle Modelle sind von gleicher Qualität und Eignung, den Wunsch der Kunden in kurzer Zeit umzusetzen. Je nachdem, ob eine Individualsoftware, Standardsoftware oder eine Produktlinie entwickelt werden soll, können die Vorteile einer Methode schnell zu Nachteilen werden. Hier werden zunächst einige Konzepte vorgestellt und im Fazit abschließend bewertet. Der Schwerpunkt liegt dabei auf der Erfassung und Berücksichtigung der Kundenanforderungen.

2.4.1. Wasserfallmodell

Das *Wasserfallmodell* (Abb. 2-4) basiert auf einer Veröffentlichung von W. W. Royce aus dem Jahre 1970 ([Roy70]). Es ist in mehrere Phasen untergliedert. Diese Phasen werden sequenziell abgearbeitet. Eine Rückkehr zu vorangegangen Phasen ist nur in Fehlerfällen möglich. Eine neue Phase kann nur nach Abschluss der vorangegangen Phase begonnen werden. Das Wasserfallmodell enthält alle für den Entwicklungsprozess wichtigen Phasen. Das Modell findet seine Anwendung sowohl für die Entwicklung von Standard- als auch Individualsoftware.
Die Kritik an diesem Modell besteht darin, dass die Softwareentwicklung keine sequentielle Abarbeitung der Phasen erlaubt, sondern entdeckte Fehler in früheren Phasen verbessert und somit weitere Phasen teilweise wiederholt werden müssen. Aber reale Projekte folgen keinem sequenziellen Ablauf. Zudem befindet sich der Kunde meist zu Pro-

jektbeginn noch im unklaren über den genauen Umfang der Anforderungen an die Applikation. Der Kundenwunsch kann in diesem Modell nur in frühen Stadien der Produktentwicklung berücksichtigt werden, spätere Änderungen sind mit hohen Kosten und Aufwand verbunden und erfordern ein wiederholtes Durchlaufen der vorangegangenen Phasen. Ein weiterer Nachteil besteht in der langen Wartezeit bis eine erste nutzbare Version zur Verfügung steht.

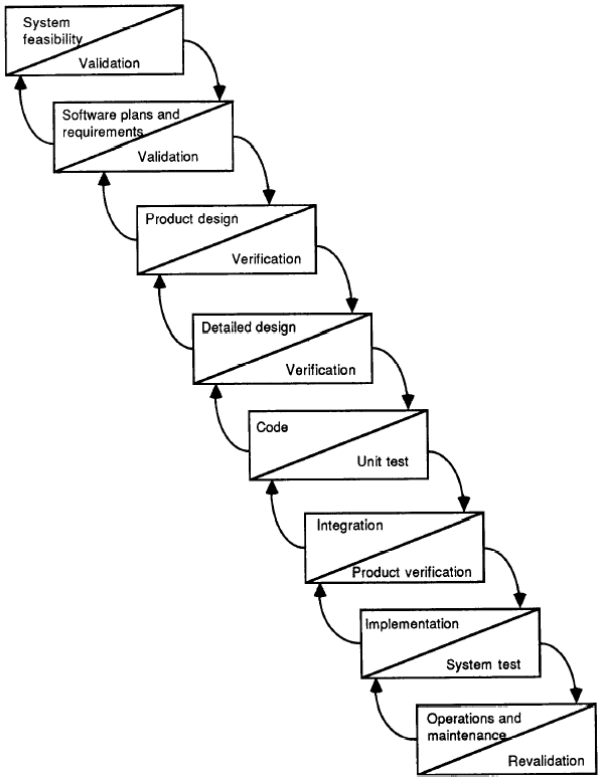

Abb. 2-4: Wasserfallmodell ([Boe88], S. 2)

2.4.2. Spiralmodell

Das *Spiralmodell* ([Boe88]), auch „Boehm Spiral Model“ genannt, beschreibt als eines der ersten Vorgehensmodelle einen iterativen-inkrementellen Prozess zur Entwicklung von Applikationen. Visualisiert wird der Prozess mit einer Spirale (Abb. 2-5), die wie auch der Prozess im Koordinatenursprung beginnt. Jeder Umlauf der Spirale entspricht einem kompletten Durchlauf durch das Wasserfallmodell. Großen Wert wird beim Spiralmodell auf die Risikoanalyse gelegt, die Probleme im Projektablauf rechtzeitig erkennen soll. Nachteilig erweist sich die schwierige Abschätzbarkeit der Dauer der einzelnen Prozessschritte. Die Abschätzung kann aber durch die Einführung von Meilensteinen für jede Iteration verbessert werden.

Das Spiralmodell wurde von Boehm zum „WinWin Spiral Model“ ([Boe98]) weiterentwickelt. Darin wird das Model ergänzt um die Erhebung und Behandlung von Änderungsanforderungen. Dazu wird zu jeder Anforderung die Person ermittelt, welche die erfolgreiche Umsetzung der Anforderung am besten bewerten kann. Für komplexe Applikationen sind das die Endanwender der betreffenden Bereiche des Unternehmens, in dem die Software eingesetzt wird.

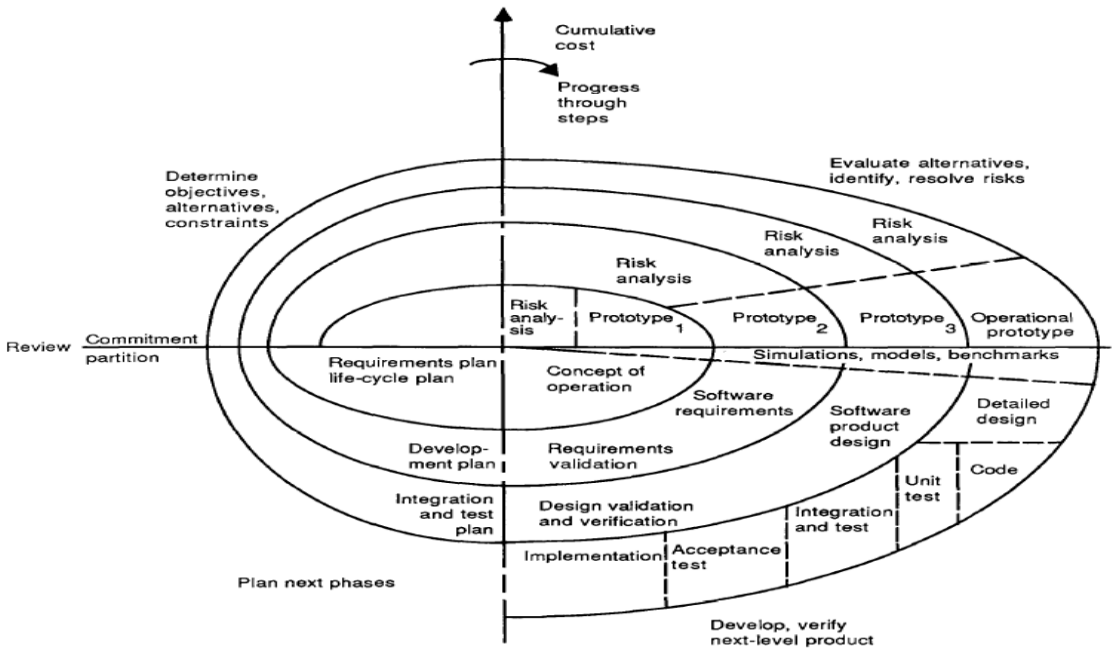

Abb. 2-5: Spiralmodell nach Boehm ([Boe88], S. 4)

Kundenanforderungen können innerhalb einer Iteration berücksichtigt werden. Dies stellt gegenüber dem Wasserfallmodell einen Fortschritt dar. Es kann aber nur schwer vorausgesagt werden, wann und zu welchen Kosten das zu entwickelnde System die Anforderung erfüllen wird.

2.4.3. Prototyping

Beim *Prototyping* [Ber98] findet man oft kein klares Vorgehensmodell. Es ist vielmehr so, dass eine Reihe von Modellen entwickelt wird, bei dem jedes weitere Modell mehr mit dem zu erwartenden Endergebnis übereinstimmt.

Nach einer eingehenden Anforderungsanalyse, gefolgt von einem Architekturdesign, wird ein erster Prototyp entwickelt. Dieser Prototyp enthält bereits zahlreiche Aspekte der zukünftigen Applikation. Bei der Erstellung eines Prototyps wird die Wartbarkeit und die Dokumentation des Sourcecodes vernachlässigt oder wie in [ID04] komplett auf Sourcecode verzichtet. Der Kunde testet, in wie weit dieser Prototyp seinen Anforde-

rungen entspricht. Wurden die Anforderungen nicht zur Zufriedenheit umgesetzt oder wurden für den Kunden neue Anforderungen an das Endprodukt ermittelt, muss ein weiterer Prototyp entwickelt werden. Dieses iterative Vorgehen wird wiederholt, bis man bezüglich der zu erwartenden Anforderungen einen Konsens gefunden hat. Basierend auf dem finalen Prototypen wird das Softwaresystem entwickelt.
Das Prototyping dient also dem Requirements Engineering, wenn nicht alle Anforderungen rechtzeitig bekannt sind. Der Kunde kann somit feststellen, ob die Software alle Anforderungen berücksichtigt. Das Risiko der Fehlentwicklung wird durch die Mitwirkung des Kunden an der Anforderungsanalyse minimiert. Weiterhin können unbeabsichtigte Nebenwirkungen frühzeitig ermittelt werden. Durch Prototyping kann so das Systemdesign festgelegt bzw. die Komplexität der Software verringert werden. Die Qualitätssicherung hat die Möglichkeit, frühzeitig in den Entwicklungsprozess eingebunden zu werden.
Nachteilig wirkt sich das Prototyping auf die Dokumentation der Anforderungen aus. Dies führt dazu, dass Anforderungen nicht korrekt oder unvollständig erhoben werden [PM04]. Auch lässt sich nicht abschätzen, wie viele Iterationen man benötigt, bis der finale Prototyp erstellt wird. Bei langen Prototypingphasen besteht die Gefahr, dass der Kunde den letzten Prototypen anstatt des eigentlichen Produktes einzusetzen gedenkt. Aufgrund der provisorischen Struktur des Prototypen ist später bei der Wartung mit erheblichen Problemen zu rechnen.
Ziel des Prototyping ist die vollständige Erfassung aller Anforderungen zu Beginn der Softwareentwicklung. Nach dem finalen Prototyp einfließende Anforderungen müssen wie in den anderen Prozessen nachträglich in die Architektur aufgenommen und eingebunden werden.

2.4.4. Extreme Programming

Das *Extreme Programming* (XP, [XP99]) ist eine Projektmanagementmethode, deren Ziel in einer möglichst zügigen Entwicklung von Sourcecode besteht. XP ist ein hochiterativer Prozess. Die Besonderheit am XP ist die enge Verbindung von Entwicklung und Qualitätssicherung sowie die in vielen Bereichen im Vergleich zu anderen Konzepten unkonventionelle Herangehensweise (Stand-Up-Meetings, Pair Programming, User Stories, ...). Der grobe Ablauf eines Projektes ist in Abb. 2-6 dargestellt.
Die Programmierung erfolgt nach dem Vier-Augen-Prinzip (englisch: Pair Programming) in kleinen dynamischen Teams mit sehr kurzen Releasezyklen. Parallel zur ei-

gentlichen Entwicklung des Sourcecodes werden Testszenarien erstellt und der Sourcecode automatisiert mit sogenannten Unit Tests überprüft.

Es besteht ein ständiger Kontakt zum Kunden, so dass Anforderungen jederzeit eingearbeitet bzw. überprüft werden können. Alle Anforderungen werden in User Stories erfasst. Eine User Story besitzt große Ähnlichkeit mit einem Use Case, dient in erster Linie jedoch der Release Planung. Sie sind von der Komplexität so gewählt, dass der Implementierungsaufwand gut abgeschätzt werden kann.

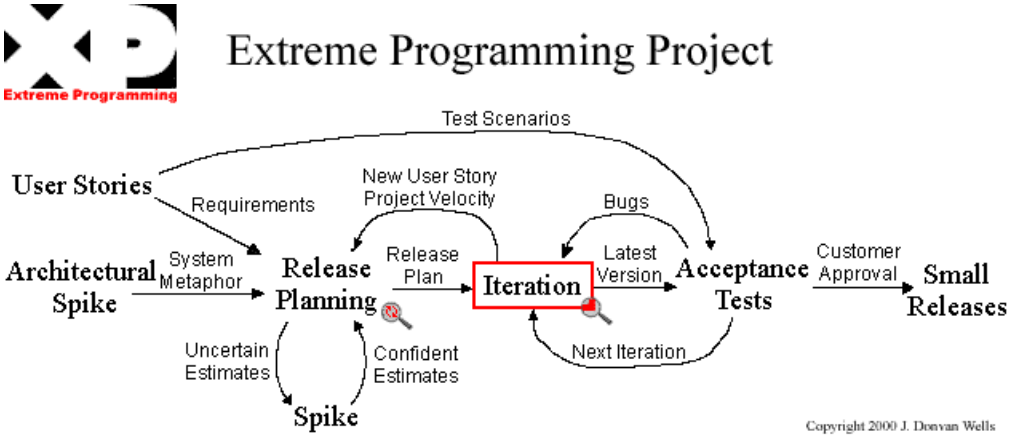

Abb. 2-6: Extreme Programming Projektablauf ([XP99])

XP ist besonders dann geeignet, wenn sich die Rahmenbedingungen des Projektes häufig ändern. Dies kann zum Beispiel bei der gleichzeitigen Entwicklung von Soft- und Hardware der Fall sein. Weiterhin eignet es sich bei einem hohen Innovationsgrad der Anwendung.

Der Projektablauf ist hochdynamisch, er setzt eine erfahrene Führung des Teams voraus. Das dadurch entstehende hohe Projektrisiko muss durch geeignete Controllingmaßnahmen vermindert werden.

2.4.5. Produktlinienentwicklung

Eine *Produktlinie* wird in der Literatur als „[...] a set of products that share a common set of requirements but also exhibit significant variability in requirements" ([Hei01], S. 405) beschrieben. Die Entwicklung von Produktlinien zielt im Gegensatz zu den bis jetzt genannten Vorgehensmodellen darauf ab, verschiedene Applikationen mit gleichen Kernfunktionen und variierendem Funktionsumfang zu entwickeln. Sie dient also der Serienfertigung von Software.

Produktlinien werden als komponentenbasierte Systeme entwickelt. Die variablen Bestandteile werden in Komponenten gegliedert, die vom Kern angesprochen werden kön-

nen. Die Verteilung der Funktionen auf Komponenten obliegt der Entscheidung des Softwarearchitekten. Der Kern selbst kann auch aus Komponenten bestehen, dies ist aber nicht zwingend notwendig. So benötigt man einen Kern und eine Auswahl von Komponenten, um eine auf den Kundenwunsch angepasste Applikation zu erstellen.

Zur Entwicklung einer neuen Produktfamilie gibt es zwei verschiedene Ansätze ([Bos00], S. 167). Der erste Ansatz wird als *evolutionäre Strategie* bezeichnet. Als Ausgangsbasis dient der Produktlinie eine bereits entwickelte Applikation, aus der weitere Systeme abgeleitet werden. Durch die Analyse und Restrukturierung des bestehenden Systems werden eine Produktlinien-Architektur und dazugehörige Komponenten abgeleitet.

Beim zweiten Ansatz, der *revolutionären Strategie*, existiert noch kein „Altsystem". Aus den Anforderungen der zu entwickelnden Systeme werden die gemeinsamen und variierenden Anforderungen herausgearbeitet, welche die Basis für die Entwicklung der Produktlinienarchitektur und der Komponenten bilden.

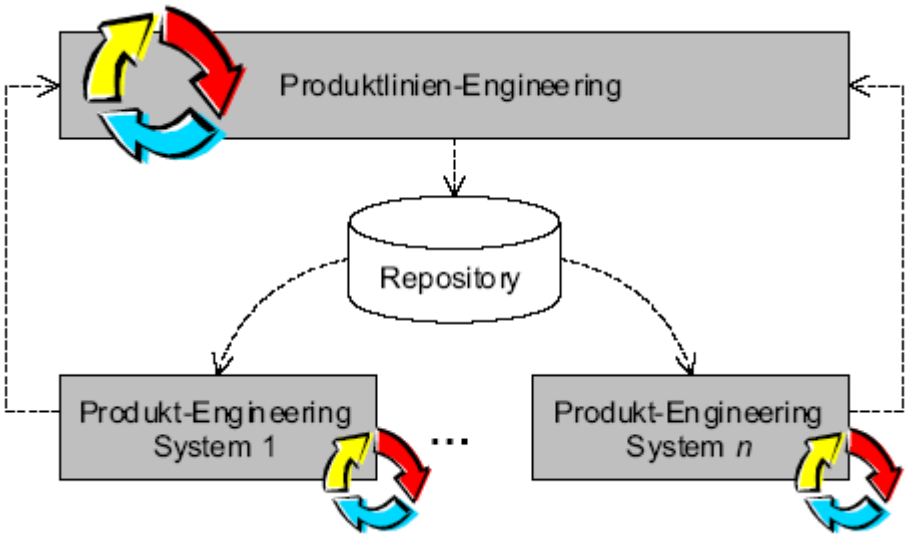

Abb. 2-7: Produktlinienentwicklung ([Böl02], S. 20)

Das Vorgehensmodell zur Entwicklung von Produktlinien (Abb. 2-7) untergliedert sich in zwei Teilprozesse ([Böl02], S. 17f). Zum einen gibt es das *Produktlinien-Engineering* (englisch: *product-line engineering*). In diesem Prozess werden Architektur und Komponenten der Produktlinie entwickelt. Im Unterschied zum *Domain Engineering* erfolgt keine Untersuchung des kompletten Anwendungsgebietes, sondern es werden nur die Anforderungen identifiziert, die bereits in Produkten implementiert oder geplant sind.

Auf diese Weise wird vermieden, die Domain zu groß oder zu klein zu wählen. Beides würde sich nachteilig auf die Produktlinienentwicklung auswirken ([Atk02], S. 281).
Im zweiten Teil, dem *Produkt-Engineering* (englisch: *application engineering*), wird aus der Produktlinie eine Applikation entsprechend der Kundenanforderungen abgeleitet.
Anforderungen für die Produktlinie werden in beiden Prozessschritten aufgenommen. Während im *Produktlinien-Engineering* die Anforderungen an die Produktlinie ermittelt werden (variable Bestandteile und Kernbestandteile), sind im *Produkt-Engineering* die konkreten Anforderungen der Kunden von Interesse. Es wird dabei verglichen, ob sie schon durch die Produktlinie abgedeckt werden. Nicht enthaltene Anforderungen können dann entweder in die Produktlinie aufgenommen oder individuell für diesen Kunden umgesetzt werden.

2.4.6. Fazit

Die in 2.4.1 bis 2.4.4 genannten Vorgehensmodelle dienen hauptsächlich der Entwicklung von Standard- bzw. Individualsoftware. Das in 2.4.5 genannte Vorgehensmodell findet in der Serienfertigung von Software Verwendung.
Das Wasserfall- bzw. das Spiralmodell ist hauptsächlich bei der Entwicklung von Standardsoftware zu finden. Bei dieser Software gibt es bereits zu Beginn eine relativ klare Vorstellung dessen, zu was das Endprodukt einmal fähig sein soll. Wünschen die Kunden neue Funktionalität, so können diese erst nach Einplanung in das nächste Release implementiert werden. Bei Standardsoftware wird der Kundenwunsch deshalb nur begrenzt Berücksichtigung finden. Um trotzdem einen großen Kundenkreis ansprechen zu können, ist es wichtig, zu Beginn so viele Anforderungen wie möglich zu erfassen und in das Produkt einzubauen. Dies führt zu komplexen Systemen, deren Funktionsumfang viele Kunden nur selten oder gar nicht verwenden. Der Hauptvorteil der Standardsoftware besteht in ihrer hohen Qualität und ihren im Vergleich niedrigen Entwicklungskosten.
Durch den engen Kundenkontakt und die schnelle Verarbeitung neuer Anforderungen eigenen sich vor allem Prototyping und Extreme Programming zur Entwicklung von Individualsoftware. Der Kunde erlebt die Entwicklung der zukünftigen Software und kann so jederzeit den Stand verfolgen und neue Anforderungen einfließen lassen. Der Vorteil ist zugleich auch ein Problem für die Entwicklung. Zwar kann der Kunde jederzeit Einfluss auf die Entwicklung nehmen, doch lässt sich aus diesem Grund auch ein

Projekt nur schwer planen. Wann alle Anforderungen zu seiner Zufriedenheit umgesetzt sind, weiss dann nur der Kunde selbst. Die Kosten und die Dauer sind deshalb schlecht zu ermitteln. Individualsoftware ist im Gegensatz zur Standardsoftware auf die Anforderungen des Kunden zugeschnitten. Vergleichsweise hohe Kosten und niedrigere Qualität gegenüber der Standardsoftware sind die Nachteile dieser Methoden.

Einen guten Mittelweg zu Standard- und Individualsoftware bildet die Entwicklung von Produktlinien. Produkte aus der Serienfertigung werden aus einzelnen Komponenten aufgebaut. Jede einzelne Komponente wird als ein Standardprodukt verstanden. Sie besitzt einen festen Funktionsumfang, der sich durch Qualität auszeichnet. Je nachdem wie viele Funktionen in eine Komponente implementiert werden, kann der Funktionsumfang der Applikation auf den Kundenwunsch abgestimmt werden. Fehlende Funktionen können in die Entwicklung neuer Komponenten einfließen, die dem Kunden separat angeboten werden.

Systeme aus der Serienfertigung sind schlanker als Standardsoftware, aber dennoch komplexer als Einzelanfertigungen. Solche komponentenbasierte Software bewegt sich in den Bereichen Qualität, Kosten und Funktionsumfang jeweils zwischen Standard- und Individualsoftware.

Der Vorteil des komponentenbasierten Ansatzes besteht in der schnellen Umsetzung von neuen Anforderungen in ein bestehendes Marktumfeld. So müssen Kunden nur einzelne Komponenten hinzukaufen, um den Funktionsumfang der eingesetzten Software zu erweitern.

Während dem Kunden bei Standardsoftware nicht viele Möglichkeiten auf Anpassung der Software an seine Anforderungen geboten werden, ist die Auswahl bei einer im Markt verbreiteten Systemfamilie um so größer. Ein ausgeprägter Markt, in dem ein Kunde für jeden seiner Wünsche Komponenten vorfinden kann, macht die Auswahl der richtigen Komponente für seine Zwecke nicht leichter. Oft verbergen sich die Funktionen hinter einem fantasievollen Komponentennamen. Ohne Hilfe eines Experten oder dem Studium der einschlägigen Literatur wird es ein Kunde schwer haben, die richtige Komponente für seine Anforderungen zu finden.

2.5. Präzisierte Problemstellung

Die Vorgehensmodelle im vorangegangenen Kapitel haben gezeigt, dass die Umsetzung des Kundenwunsches in eine qualitativ hochwertige und zugleich preisgünstige Applikation nicht ohne Probleme möglich ist. Entweder wird der Kundenwunsch nur ungenügend berücksichtigt oder er ist mit hohen Kosten und vergleichsweise geringer Qualität verbunden. Der komponentenbasierte Ansatz erhöht zwar die Flexibilität, mit der Kundenanforderungen berücksichtigt werden können, aber woher soll der Kunde wissen, welche Komponente seinen Ansprüchen am besten genügt? Der Weg, der in den folgenden Kapiteln vorgestellt wird, soll diese Probleme reduzieren oder gar gänzlich vermeiden.

In der Kommunikation zwischen Entwickler und Endanwender ist ein gemeinsamer Wortschatz von entscheidender Bedeutung. Mit Hilfe von Merkmalen kann ein gemeinschaftliches Verständnis für ein System geschaffen werden, bei dem Missverständnissen frühzeitig vorgebeugt wird.

Durch die Verwendung von Komponenten wird es möglich, eine für den Kunden maßgeschneiderte Applikation in kurzer Zeit zu erstellen. Wenn die Anforderungen an das System komplett durch bereits entwickelte Komponenten abgedeckt werden können, besteht die Möglichkeit nicht nur Zeit, sondern auch Entwicklungskosten und Ressourcen zu sparen. In einem solchem Fall kann die Applikation ableitet werden.

Eine Verknüpfung des merkmalbasierten Ansatzes mit komponentenbasierter Software sollte deshalb zu individuelleren Systemen führen, die eher den Anforderungen der Nutzer entsprechen als Programme aus der Massenfertigung. Im Vergleich zu Einzelanfertigungen sind komponentenbasierte Applikationen kostengünstiger. Das betrifft sowohl die Entwicklungs- als auch die Wartungskosten. Voraussetzung dafür ist allerdings, dass die verwendeten Komponenten nicht nur in einer, sondern in vielen Applikationen eingesetzt werden. Die Komponentensoftware wird so aus Sicht der Kosten wie auch der Anpassbarkeit zwischen Massenprodukt und individueller Entwicklung angesiedelt.

Das Ziel wird sein, auf der Grundlage eines unter Anwendung von FORE erstellten Merkmalmodells eine Applikation zu erzeugen. Dieses Modell wurde im Vorfeld erstellt und validiert. Es erfasst die Anforderungen des Kunden als eine Merkmalauswahl.

Zunächst werden die theoretischen Aspekte des Problems untersucht. Kernpunkte werden hier folgende Fragen sein:

- Inwieweit ist es möglich, das Merkmalmodell mit Hilfe eines zu definierenden Komponentenmodells in eine Applikation zu überführen?
- Welche zusätzlichen Informationen werden von den Komponenten benötigt?
- Nach welchen Aspekten werden die Komponenten ausgewählt und konfiguriert?

Das Resultat wird in einer Erweiterung der FORE-Methode bestehen, um deren Anwendbarkeit auf komponentenbasierte Software zu verbessern. Die Überführung der Merkmalauswahl in einen Satz von Komponenten wird automatisiert vorgenommen. Die Ableitung wird nur für die Softwarekomponenten genutzt. Die benötigten Hardwarekomponenten werden vorausgesetzt. Zur Demonstration des Ansatzes wird ein Prototyp entwickelt.

Durch die Anwendung dieses Ansatzes sollen Personen, die nicht direkt mit dem Entwicklungsprozess der Applikation verbunden sind (z. B. Verkäufer oder Kunden), befähigt werden, aus bestehenden Komponenten mit minimalem Aufwand eine den Anforderungen entsprechende Applikation zu erzeugen.

3. Fallbeispiel Digitales Videoprojekt (DVP)

Zur Demonstration des Lösungsansatzes wird eine Beispielimplementierung im Rahmen des digitalen Videoprojektes vorgenommen, erprobt und anschließend bewertet. Dazu wird im Kapitel „Eigener Ansatz“ der Lösungsansatz vorgestellt und erläutert. Das DVP-System wurde bereits für FORE in [Str04] als Beispiel verwendet. Aufbauend auf den dort entstandenen Merkmalmodellen wird in dieser Arbeit daraus eine komponentenbasierte Applikation abgeleitet. In diesem Kapitel wird zunächst das DVP-System vorgestellt.

Um die Forschungen im Rahmen der Systemfamilienentwicklung und von FORE mit einer praktischen Anwendung zu untermauern, wurde DVP ins Leben gerufen. Als Kern des Projektes kommt die Open Source Software *vdr* („Video-Disk-Recorder“, [Schm02]) zur Anwendung. Den Autoren des *vdr* ist der Funktionsumfang handelsüblicher Videorekorder zu eingeschränkt. Viele Geräte besitzen ein oder mehrere technische Highlights, aber das Traumgerät, in dem alle Wünsche vereinigt sind, existiert nicht. Um ein flexibles System zu erhalten, welches den Ansprüchen von jedermann gerecht werden sollte, entwickelten sie ein neues System. Dazu wählten sie einen komponentenbasierten Ansatz, bei dem es möglich ist, in das *vdr*-Grundsystem viele verschiedene Plug-Ins einzubinden. Zahlreiche Entwickler nutzen die Gelegenheit und programmieren Erweiterungen mit den von ihnen gewünschten Funktionalitäten. Bis heute sind etwa 70 Plug-Ins für *vdr* erschienen, ihre Anzahl wächst ständig weiter.

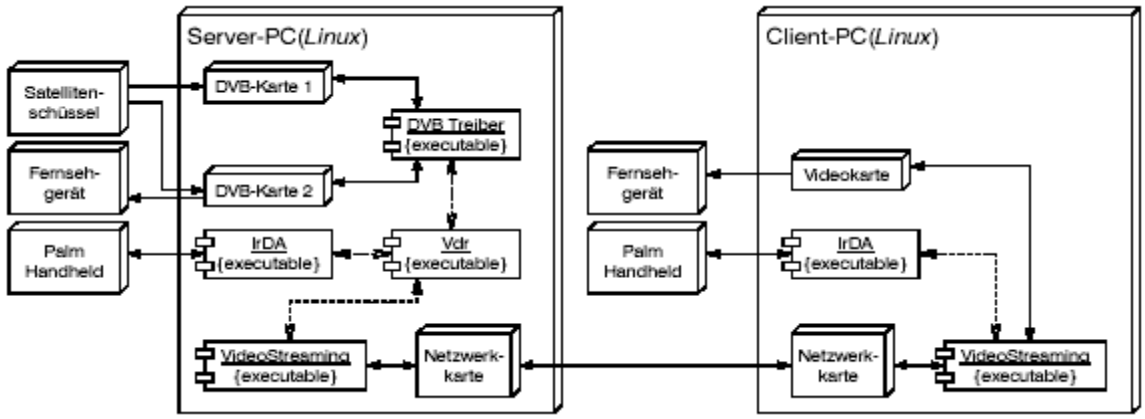

Abb. 3-1: Architekturüberblick Digitales Videoprojekt ([Str04], S. 20)

Die Hardwaregrundlage des DVP bilden handelsübliche PCs. Die derzeitige Ausbaustufe des Systems ist in Abb. 3-1 dargestellt. Es besteht aus einem Server-PC und einem

Client-PC, die mit je 2 DVB-Karten („Digital Video Broadcast“) ausgerüstet sind. Der Server dient der kompletten Verarbeitung der Videosignale, die er über die beiden an einen Satellitenempfänger angeschlossenen DVB-Karten erhält. Die Karten werden über den DVB-Treiber von [Met02] angesteuert. Der Client kann vom Server die verarbeiteten Videosignale nach dem Video-On-Demand-Prinzip anfordern und darstellen. Die Verteilung der Videosignale vom Server zum Client erfolgt über das von [Mef03] in das DVP eingebrachte Streaming-Plug-In für den *vdr*. Der Datentransfer basiert auf dem Video-Streaming-Projekt von [Lat02] unter Verwendung von MPEG2 Streaming-Typen. Die Daten können daher über ein Standard-Netzwerkinterface (Ethernet o. ä.) übertragen werden. Weiterhin stellt der Server die Videosignale über den TV-Ausgang der DVB-Karte bzw. der Client über den TV-Out einer Videokarte einem Fernsehgerät zur Verfügung. Der Funktionsumfang der verwendeten DVB-Karten ermöglicht es, Zusatzinformationen in das laufende Bild einzublenden. Auf diesem Weg kann man den Nutzer zusätzlich mit Programminformationen versorgen oder ein Feedback zur Gerätebedienung liefern. Das System kann zusätzlich mit einem Handheld-Computer bedient werden. Dazu wurden die Ergebnisse des IrDA-Projektes ([Bar02]) von [Die03] integriert.

Als Ergebnis des DVP entstand somit ein System, an dem die unterschiedlichen Ansätze der Systemfamilienentwicklung getestet werden konnten. Dieses System wird im weiteren Verlauf der Arbeit für die beispielhafte Umsetzung des Ansatzes genutzt.

4. Eigener Ansatz

In diesem Kapitel wird ein Verfahren vorgestellt, mit dem eine automatisierte Ableitung vorgenommen werden kann. Dazu werden die theoretischen Aspekte des Ansatzes untersucht und erläutert. Im nächsten Kapitel wird dieser Ansatz anhand eines Beispiels demonstriert.

Zunächst werden die Beziehungen zwischen einer Komponente und ihren Merkmalen genauer untersucht. Merkmale können einerseits eine Komponente charakterisieren und sind andererseits für den Kunden verständlich. Im ersten Unterkapitel werden deshalb diese Zusammenhänge erklärt und dargestellt, mit welchen Merkmalen man eine Komponente am besten beschreibt.

Im darauf folgenden Kapitel werden die Besonderheiten einer komponentenbasierten Architektur erläutert, die durch die Ableitung aus einem Merkmalbaum zu beachten sind. Nicht jede Komponente kann ohne weiteres in einer Anwendung genutzt werden, nur weil sie das richtige Interface und die benötigten Funktionen bereitstellt.

Das nächste Kapitel beschäftigt sich mit dem Komponentenmodell. Hier wird der Aufbau und der Zweck dieses Modells dargelegt.

Darauf folgt der Kern des eigenen Ansatzes, die Auswahl der Komponenten für die zukünftige Applikation. Es wird das Verfahren zunächst erläutert und später anhand eines Beispiels weiter vertieft.

Als Resultat der Komponentenwahl entsteht ein Systemmodell, das im darauffolgenden Kapitel beschrieben wird. Dort wird die Struktur des entstehenden Systems für die Ableitung hinterlegt.

Der vorletzte Abschnitt widmet sich der Modifikation einer bestehenden Anwendung. Er befasst sich mit dem Thema, wie man eine Anwendung, die mit dem gezeigten Ansatz erstellt wurde, verändern kann.

Den Abschluss bildet eine Zusammenfassung des eigenen Ansatzes.

4.1. Komponenten und ihre Merkmale

Komponenten stellen eine Form der Abstraktion dar, denn sie fassen Funktionalität zusammen. Dabei wird nicht willkürlich Funktionalität gebündelt. Es handelt sich vielmehr um Funktionen eines autonomen Konzeptes oder Prozesses aus dem gleichen Geschäftsumfeld. Diese Bündelung dient der Sammlung von Wissen innerhalb des abgedeckten Anwendungsfeldes ([And03], S. 19f). Dieses Wissen wird über wohl definierte Schnittstellen anderen Bestandteilen einer Software zugänglich gemacht.

Auf dem Markt erfolgreiche Komponenten besitzen eine direkte begriffliche Bedeutung für den Endanwender ([Szy02], S. 13). Allerdings sind für einen Kunden auf den ersten Blick zwei Komponenten für z. B. eine Videowiedergabe nicht zu unterscheiden. Er wird sich also genauer informieren und die Fähigkeiten der einen mit der anderen Komponente vergleichen und abwägen. Die Unterscheidung geschieht anhand von Funktionen, Preisen, Lizenzen, Ressourcenverbrauch und vielem mehr. Die Kunden unterscheiden die Komponenten also nicht nach dem Namen, sondern nach den Fähigkeiten bzw. anhand von Merkmalen. Der Name einer Komponente mag einen ersten Hinweis auf deren Funktionsumfang geben, dieser ist aber häufig nicht aussagekräftig genug, um eine Entscheidung fällen zu können. Als Beispiele mögen hier zwei Plug-Ins aus dem vdr-Projekt reichen. Ein Plug-In trägt den Namen *Commander* [Com03]. Diese Plug-In unterstützt die Dateiverwaltung über die Anzeige (OSD) des Videosystems im Stile des *Norton Commanders*. Aus dem Namen kann man nur mit Kenntnis der Commander-Programme [Orn04] auf den Verwendungszweck schließen. Als weiteres Beispiel sei hier *Femon* [Ahr04] genannt. Femon ist eine Kurzform für *DVB Frontend Status Monitor*. Die Aufgabe dieses Plug-Ins ist die Anzeige von Informationen zur Signalqualität des Videosignals.

Die Komponenten werden anhand ihrer Merkmale klassifiziert. Merkmale beschreiben eine Komponente präziser als nur ihr Name. Sie lässt sich damit besser differenzieren. Der Name gibt dem Kunden wenige Rückschlüsse, welche Anforderungen des Systems durch die Komponente erfüllt werden.

Was für Merkmale kann man nun zur Beschreibung einer Komponente heranziehen? Im Kapitel „Definition: Merkmal“ wurde eine Untergliederung der Merkmale in vier Kategorien vorgenommen. Es wird unterschieden in funktionale Merkmale (FM), Schnittstellenmerkmale (SSM), Parametermerkmale (PM) und Strukturmerkmale (SM). Mit

dieser Einteilung kann man eine Komponente gut, aber noch nicht vollständig beschreiben.

Zusätzlich wird in dieser Arbeit das Beschaffenheitsmerkmal (BM) eingeführt. Es handelt sich dabei um eine Differenzierung des Parametermerkmals. In dessen Definition wird nicht darauf eingegangen, wie die Werte des Merkmals zustande kommen bzw. wie sie behandelt werden. Parametermerkmale enthalten im weiteren Verlauf der Arbeit Werte, die durch Anforderungen verändert werden können, während Beschaffenheitsmerkmale Werte einer Komponente bezeichnen, die durch den Hersteller/Anbieter vorgegeben und nicht veränderbar sind.

In den folgenden Unterkapiteln wird konkreter auf die Beziehungen der einzelnen Merkmaltypen zu den Komponenten eingegangen. Nicht jeder Typ ist hierbei für die Komponenten von Belang bzw. kann in seiner anfangs eingeführten Definition verwendet werden. Weiterhin wird erläutert, welche Bedeutung die Merkmalkategorien bezogen auf eine Komponente haben und wie sie von der Definition der Merkmale in FORE abweichen.

4.1.1. Funktionale Merkmale einer Komponente

Dieser Ansatz geht davon aus, dass der Kunde die Komponenten anhand ihrer *funktionellen Merkmale* auswählt. Sie spiegeln die Konzepte bzw. Prozesse wieder, welche in die Komponenten eingebettet sind. Jede Komponente besitzt mindestens ein, im allgemeinen jedoch mehrere funktionale Merkmale. Komponenten ohne funktionale Merkmale existieren nicht. Man könnte dagegenhalten, das Wrapper-Komponenten keine Funktion besitzen, dies ist jedoch nicht richtig. Die Funktion einer solchen Komponente besteht in der Koppelung anderer Komponenten.

Parametermerkmale können zur Konfiguration von Funktionen verwendet werden. Sie können funktionalen Merkmalen zugeordnet werden und verändern somit deren Verhalten. Ein Beispiel wäre die Aufzeichnungskapazität für ein Video. Dem funktionalen Merkmal „Video aufzeichnen“ wäre das Parametermerkmal „Kapazität“ mit dem Wert „4 Stunden“ zugeordnet.

Funktionale Merkmale sind also das wichtigste Charakterisierungsmerkmal der Komponente. Durch eine Auswahl funktionaler Merkmale können Komponenten bestimmt werden, die letztendlich durch ihr Zusammenwirken die Applikation bilden.

4.1.2. Schnittstellenmerkmale einer Komponente

Für *Schnittstellenmerkmale* wird in dieser Arbeit eine andere Definition verwendet als die in [Str04] dargestellte. Dort wird definiert, dass sie „die Integration von Standards oder Subsystemen“ wiedergeben. Schnittstellenmerkmale dienen im vorliegenden Ansatz der Spezifikation, welche Schnittstellen von einer Komponente erwartet bzw. zur Verfügung gestellt werden. Insofern trifft der zweite Teil der Definition teilweise auch hier zu, da man angeschlossene Komponenten durchaus als Subsystem ansehen kann.
Bei der Einordnung von Standards in die Schnittstellenmerkmale weicht die Definition gänzlich ab. Die Frage, ob eine Komponente beispielsweise den Audiostandard MP3 unterstützt, wird hier durch ein funktionales Merkmal ausgedrückt. Ein Beispiel: Angenommen die fiktive Komponente *audioConvert* würde die in Abb. 4-1 dargestellten Merkmale besitzen.

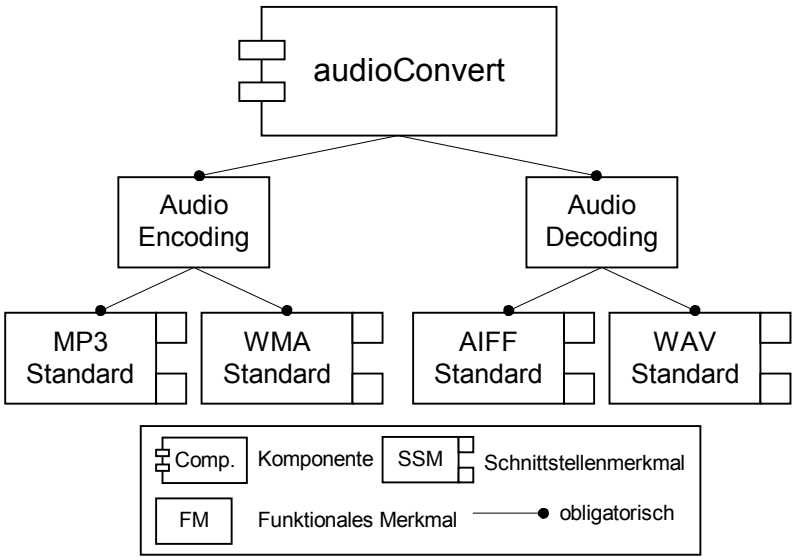

Abb. 4-1: Komponente audioConvert (Interpretation nach [Str04])

Kann die Komponente MP3-Dateien in das WAV-Format oder WAV-Dateien nach MP3 und WMA konvertieren? Man kann nicht sagen, welchen Funktionsumfang diese Beispielkomponente wirklich bereitstellt. Beispielweise wäre hieraus nicht ersichtlich, dass die Komponente eine Konvertierung von AIFF nach WMA nicht unterstützt. Eine Integration der Standards in die funktionalen Merkmale verschafft mehr Klarheit. Genauer könnte man die Komponente wie in Abb. 4-2 dargestellt beschreiben.
Die Komponente wird jetzt nur noch über diese drei funktionalen Merkmale beschrieben. Die ursprünglichen Schnittstellenmerkmale führten zur Fehlinterpretation des Funktionsumfangs der Komponente. Erst jetzt wird wirklich erkennbar, welche Aufgaben diese Komponente verrichten kann.

In der Definition der Merkmale heißt es, dass die sie einen Aspekt wiedergeben, der für den Kunden von Bedeutung ist. Schnittstellendefinitionen sind hier keine Ausnahme, auch wenn sie den Anschein erwecken, rein technische Merkmale zu sein. Für einen Kunden macht es durchaus einen Unterschied, welche Komponente in seiner Applikation eingesetzt werden kann.

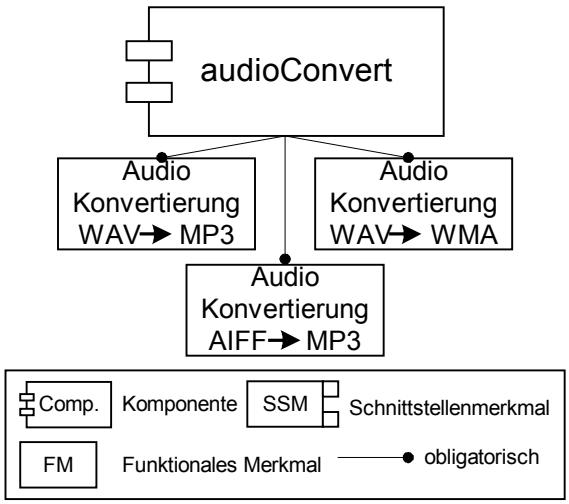

Abb. 4-2: Komponente audioConvert (Integration von Standards)

Ein kurzes Beispiel soll dies verdeutlichen. Im Browsermarkt sind Komponenten, oder wie dort bezeichnet Plug-Ins, weit verbreitet. Wer den *Shockwave Player* von *Macromedia* [Mac04] einsetzen will muss wissen, für welchen Browser das Plug-In sein soll. Ein *Mozilla* Plug-In unterscheidet sich von der *Internet Explorer* [Mic04] Variante in der Schnittstelle. Die Plug-Ins würden also jeweils ein Schnittstellenmerkmal besitzen, dass entweder *Mozilla* oder *Internet Explorer* heißt.

Bei der Erstellung einer Applikation sind Schnittstellenmerkmale von keiner bis geringer Bedeutung. Beim Neukauf eines Systems interessieren sich Kunden hauptsächlich für die funktionalen Aspekte. Wichtig wird die Schnittstelle erst dann, wenn der Kunde sein bestehendes System erweitern möchte. Eine Erweiterung liegt beispielsweise auch dann schon vor, wenn der Kunde die Anwendung für das Betriebssystem Windows benötigt.

Die Schnittstellen von Komponenten lassen sich aber noch weiter beschreiben. Angenommen beim ersten Beispiel *audioConvert* handelt es sich um ein Plug-In für *vdr*. Es unterstützt die Schnittstelle des *vdr*-Systems. Es ist bekannt, dass *audioConvert* die von *vdr* zur Verfügung gestellte Schnittstelle nutzt, um seinen Funktionsumfang an *vdr* weiterzugeben. Was würde passieren, wenn nicht bekannt wäre, dass *audioConvert* das Plug-In für *vdr* ist? Es könnte angenommen werden, *vdr* ist das Plug-In! Man kann also

die Schnittstellen einer Komponente nochmals in „bereitgestellt“ (englisch: *provided*) und „genutzt“ (englisch: *required*) unterteilen. Die Unterscheidung ist für einen Menschen trivial, für eine automatisierte Ableitung aber von entscheidender Bedeutung. Andernfalls könnte ein Programm versuchen eine Kombination umzusetzen, wie sie in Abb. 4-3dargestellt ist.

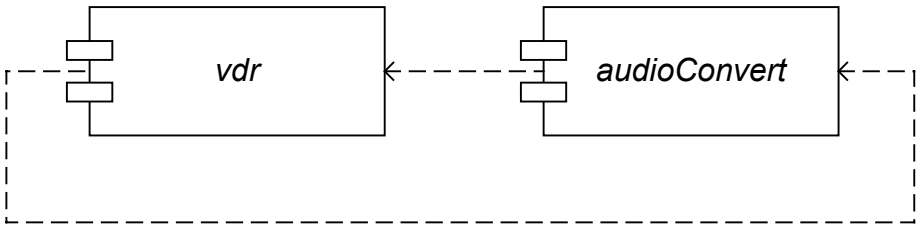

Abb. 4-3: Fehlerhafte Verknüpfung von Komponenten

Schnittstellen unterteilen die Welt der Komponenten in Klassen von Komponenten. Jeder Komponente, die über die *vdr*–Schnittstelle angesprochen werden kann, ist damit eine *vdr*-Komponente und kann nur dort eingesetzt werden. Es ist möglich, dass eine Schnittstelle von mehreren Komponenten gleichzeitig genutzt wird. Natürlich kann man eine Komponente auch so gestalten, dass sie außer *vdr* auch andere Programme unterstützt. Dazu benötigt sie aber jeweils eine zusätzliche Schnittstelle für jedes weitere Programm.

Die Standardisierung von Schnittstellen und die Gliederung nach Anwendungsgebieten würde das Einsatzgebiet einer Komponente erweitern. Zu den Vorteilen zählen unter anderem die Versionssicherheit. Auf diese Weise kann eine Komponente über mehrere Programmversionen hinweg genutzt werden. Eine Anpassung an die neue Version würde entfallen. Weiterhin besteht die Chance, dass eine Standardisierung zur Erweiterung des Einsatzgebietes oder gar zu einem Markt für Komponenten führt. Erweiterungen durch Dritte gestatten eine Erhöhung der Vielfalt und die Steigerung des Wettbewerbes.

4.1.3. Parametermerkmale einer Komponente

Komponenten besitzen keine direkten *Parametermerkmale*. Solche Merkmale dienen der Konfiguration von funktionalen Merkmalen. Damit haben sie indirekt einen Einfluss auf die Konfiguration der Komponente. Parametermerkmale sind durch Anforderungen im Wert veränderliche Parameter von Funktionen. Beispiele hierfür können die Anzahl der CPU’s eines Systems, die Speicherkapazität von Medien oder die Anzahl der Threads einer Applikation sein.

4.1.4. Beschaffenheitsmerkmale einer Komponente

Für die automatische Auswahl sind die konstanten Eigenschaften der Komponenten, die *Beschaffenheitsmerkmale*, von entscheidender Bedeutung. Sie können ebenso wie die Parametermerkmale aus quantifizierbaren oder nicht quantifizierbaren Werten eines bestimmten Typs bestehen. Beispiele für quantifizierbare Eigenschaften einer Komponente sind die Komponentengröße (Speicherverbrauch oder Festplattenplatz in Bytes), die Ausfallsicherheit (in Prozent) oder auch der Preis. Nicht quantifizierbare Eigenschaften können das Lizenzmodell, die Verfügbarkeit des Sourcecodes oder der Hersteller sein.
Die funktionalen Merkmale einer Komponenten können ebenfalls Beschaffenheitsmerkmale besitzen. Sie beschreiben Eigenschaften, die nicht für die gesamte Komponente gelten und meist rein technischer Natur sind. Die Wiedergabequalität oder die Kompressionsrate für Videos sind hierfür Beispiele.
Das Betriebssystem zählt nicht zu den Beschaffenheitsmerkmalen. Das Betriebssystem gehört in den Bereich der Schnittstellen. Nicht jede Komponente benötigt eine Schnittstelle zum Betriebssystem, sie kann auch durch die einbindende Komponente interpretiert werden (z. B. Makros). Weiterhin können die Komponenten auf verschiedene physische Systeme verteilt sein (z. B. Applikation unter Windows mit zugehörigem Datenbanksystem auf Unix).
Die Beschaffenheitsmerkmale können als zusätzliche Entscheidungshilfe sowohl bei der automatischen als auch bei der manuellen Auswahl dienen. Nicht alle Eigenschaften sind gleich gut für die automatische Selektion geeignet. Während es bei quantifizierbaren Eigenschaften weniger Probleme gibt, ist die Behandlung nicht quantifizierbarer Eigenschaften ungleich komplizierter. Anhand des Lizenzmodells lässt sich das Problem verdeutlichen. Eine OpenSource oder GNU Lizenz lässt sich noch relativ einfach einordnen. Ein Verständnis für deren Inhalte ist weit verbreitet. Nimmt man allerdings die Lizenz einer Firma, so erfordert deren Inhalt meist eine rechtliche Prüfung. Auch können durch eine Lizenz andere Eigenschaften einer Komponente beeinflusst werden. Eine Preisstaffelung je nach Anzahl der verwendeten CPUs oder der Kapazität kann darin verankert sein.
Nicht automatisiert verwertbare Eigenschaften werden im hier vorgestellten Ansatz nicht beachtet. Auch auf das Lizenzmodell wird nicht weiter eingegangen, sie werden zukünftigen Forschungsvorhaben überlassen. Alle im Beispielszenario verwendeten Komponenten besitzen eine OpenSource-Lizenz.

4.1.5. Strukturmerkmale einer Komponente

Die Strukturmerkmale haben keinen direkten Einfluss auf die Auswahl der Komponenten. Sie dienen lediglich der Strukturierung des Merkmalmodells. In einem Merkmalbaum fassen sie verwandte Merkmale mit einem verständlichen Oberbegriff zusammen und gestatten so einen schnelleren Überblick auf die Möglichkeiten der Systemfamilie. Man kann sagen, dass sobald ein funktionales Merkmal durch untergeordnete funktionale Merkmale weiter verfeinert wird, daraus ein Strukturmerkmal entsteht.

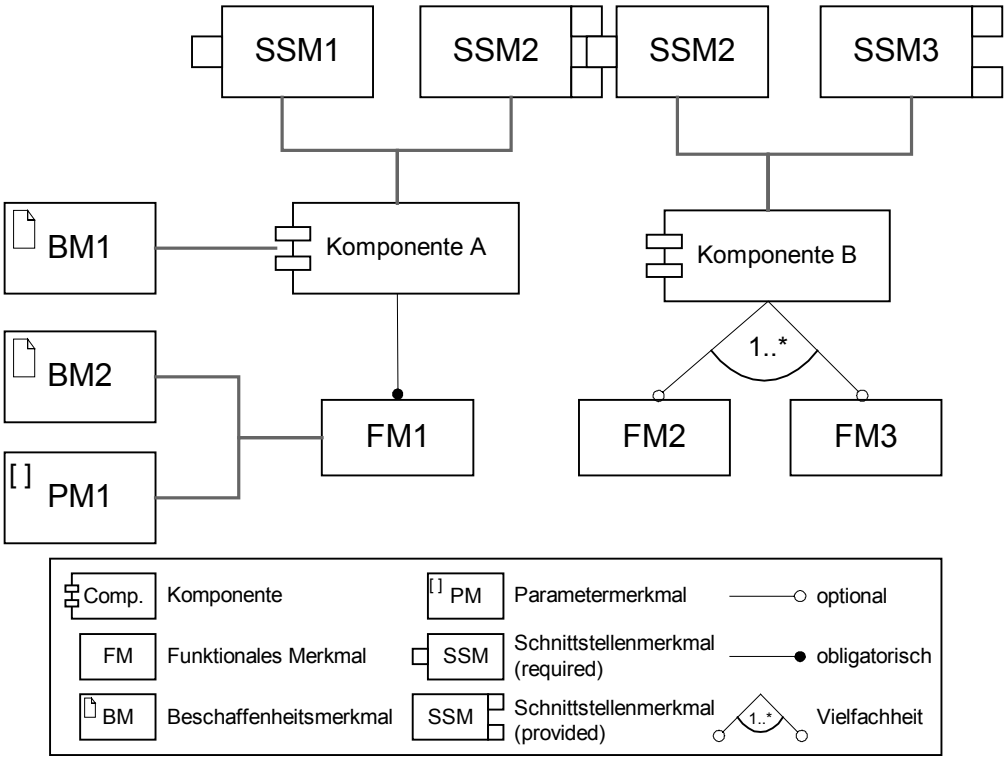

Abb. 4-4: Zusammenhänge zwischen Merkmalen und Komponenten

Strukturmerkmale können demgegenüber einen Einfluss auf die Entwicklung der Systemfamilie haben. Sie liefern einen guten Ansatzpunkt, welche Merkmale in einer Komponente zusammengefasst werden könnten bzw. welche Schnittstellen diese Komponente ihrer untergeordneten Komponenten zur Verfügung stellen sollte. Dieser Ansatz wird hier jedoch nicht weiter verfolgt und sollte in einer separaten Arbeit untersucht werden. Im Rahmen dieser Arbeit werden nur bereits fertiggestellte Komponenten berücksichtigt.

4.1.6. Fazit

Die Untersuchung der einzelnen Typen zeigt, wie man eine Komponente für einen Anwender verständlich mit Merkmalen beschreiben kann. In Abb. 4-4 sind die Zusammenhänge zwischen Merkmalen und Komponenten dargestellt. Für die automatisierte Auswahl von Komponenten werden vier Merkmaltypen benutzt:

- *Funktionale Merkmale* spiegeln die primären Anforderungen des Kunden wieder. Die Entscheidung des Kunden wird vorrangig durch die funktionalen Aspekte beeinflusst.
- *Schnittstellenmerkmal*e bestimmen, welche Komponenten in die Auswahl aufgenommen werden können. In der zukünftigen Applikation muss eine kompatible Schnittstelle zum Andocken der weiteren Komponenten vorhanden sein.
- *Parametermerkmale* beschreiben im Wert veränderliche Konfigurationen der funktionalen Merkmale einer Komponente.
- *Beschaffenheitsmerkmale* sind Randbedingungen, die bei einer Komponentenauswahl Verwendung finden. Sie lassen sich automatisiert oder manuell auswerten und dienen der Eingrenzung bzw. Wertung der Komponentenauswahl.

Die *Strukturmerkmale* sind für die Komponentenauswahl ohne Bedeutung.

4.2. Komponente, Merkmalbaum, Architektur

In diesem Kapitel werden wichtige Aspekte der Kern- und Komponentenstruktur erläutert. Nachdem im vorangegangen Kapitel die Beziehungen zwischen Komponenten und einzelnen Merkmalen untersucht wurden, erfolgt jetzt die Betrachtung von Komponenten und Merkmalbaum. Es werden sowohl an den Kern als auch an die Komponenten Anforderungen gestellt, ohne die eine automatisierte Ableitung nicht möglich ist. Anhand von zwei Beispielen werden die Probleme der Ableitung erläutert.

Neben dem Kunden wird ein weiterer Akteur eingeführt. Bisher ist er teilweise als Verkäufer in Erscheinung getreten. Die Zusammenstellung der notwendigen Komponenten wird durch eine Fachkraft vorgenommen, welche die Anforderungen des Kunden interpretieren und das Werkzeug zur Ableitung bedienen kann. Szyperski ([Szy03], S. 497f) hat eine Person mit diesem Tätigkeitsfeld als *Component Assembler* oder frei übersetzt *Komponentenmonteur* bezeichnet. Im weiteren Verlauf der Arbeit wird der Begriff des *Komponentenmonteurs* verwendet.

4.2.1. Architektur einer merkmalbasierten Produktlinie I

Die automatische Ableitung stellt hohe Anforderungen an die Architektur des Komponentensystems. Der Systemkern muss Schnittstellen bereitstellen, über welche die optionalen Merkmale eingebunden werden können. Weitere Voraussetzung für die Auswahl ist die komplette Abdeckung der ermittelten Anforderungen durch fertige Komponenten und vor allem deren problemloses Zusammenspiel.

Ein Beispielsystem namens Alpha ist in Abb. 4-5 dargestellt. Man kann es sich als Produktlinie vorstellen, die von einem einzelnen Hersteller entwickelt und betreut wird. Der Funktionsumfang und dessen Verteilung auf Komponenten ist klar definiert.

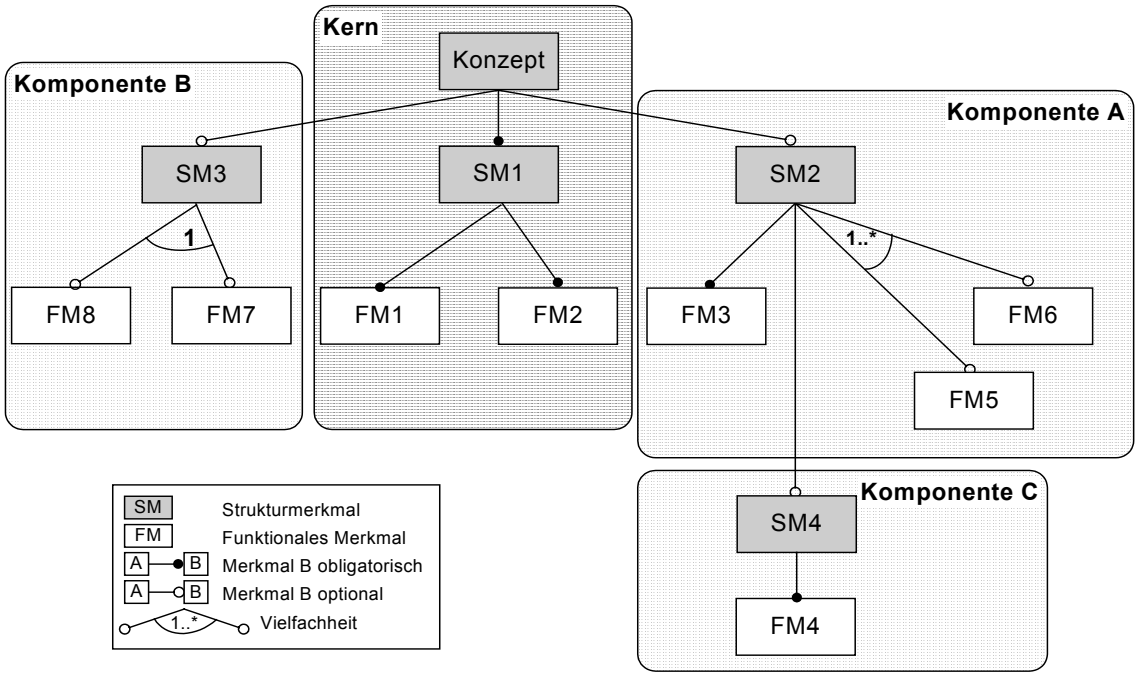

Abb. 4-5: Merkmalbaum Beispielsystem Alpha

Der Systemkern implementiert die über das Strukturmerkmal SM1 zusammengefassten funktionalen Merkmale FM1 und FM2. Diese Merkmale sind obligatorisch und müssen ausgewählt werden. Der Kern repräsentiert die Basis der Produktlinie, die alle obligatorischen Merkmale des Konzeptes einbindet. Der Kern kann über zwei Schnittstellen erweitert werden. Die Schnittstellen ermöglichen die Erweiterung mit Komponenten des Typs A und Typs B.

Komponente A stellt eine optionale Erweiterung des Systems dar; das Strukturmerkmal SM2 ist optional. Es besteht aus dem in dieser Erweiterung obligatorischen Merkmal FM3 sowie den optionalen Merkmalen FM5 und FM6. Mindestens eines der beiden Merkmale muss laut Merkmalmodell ausgewählt sein. Weiterhin stellt Komponente A

eine Schnittstelle für Komponenten des Typs C bereit, um die eigenen Funktionen ergänzen zu können.

Was sind die technischen Konsequenzen? Komponenten des A-Typs müssen eine Möglichkeit besitzen, optionale Merkmale abschalten zu können. Besteht diese Konfigurationsmöglichkeit nicht, kann die Komponente die Anforderungen der Produktlinie nicht voll erfüllen. Sie dürfte also kein Teil der durch diesen Merkmalbaum repräsentierten Produktlinie sein und darf nicht mit zur Auswahl stehen.

Man könnte nun sagen, sie solle nur dann berücksichtigt werden, wenn beide Merkmale FM5 und FM6 genutzt werden sollen. Eine nachträgliche Entfernung eines der Merkmale wäre dann mit der eingesetzten Komponente nicht möglich.

Kurzes Szenario zur Erklärung: Ein Softwareadministrator hat eine Konfiguration bestehend aus Kern und Komponente Super-A gekauft. Die verwendete Komponente des Typs A kann FM5 und FM6 nicht abschalten. Der Verkäufer hatte ihm erklärt, dass in dieser Produktlinie die Merkmale FM5 und FM6 optional sind und jederzeit eines der beiden Merkmale abgeschaltet werden kann. Aufgrund von weiteren Rahmenbedingungen (Speicherverbrauch, Performance, Preis) entscheidet sich der Softwareadministrator letztendlich für Komponente Super-A. Die Applikation wird in der Firma eingesetzt und auf verschiedenen Systeme installiert. Als ein neuer Mitarbeiter in der Firma eingestellt wird, soll während der Anlernphase das kritische Merkmal FM6 (z. B. Lösch-Funktion) abgeschaltet werden. Die Konfiguration der Komponente Super-A lässt dies aber nicht zu.

Eine Komponente kann also nur dann in einer Produktlinie eingesetzt werden, wenn sie die gleichen Fähigkeiten besitzt, wie es der Merkmalbaum vorgibt. Zusätzliche Merkmale der Komponente, die nicht im Merkmalbaum enthalten sind, bleiben unberücksichtigt. Wenn ein Merkmal optional einsetzbar ist, muss dies auch in der Komponente möglich sein. Ein optionales Merkmal kann auch dadurch abgeschaltet werden, indem die komplette Komponente aus dem System entfernt wird. Sie implementiert also ausschließlich dieses optionale Merkmal.

Weiterhin können im Merkmalbaum obligatorische Merkmale in der Komponente optional abschaltbar sein. Dies erhöht ihre Flexibilität für den Einsatz in anderen Produktlinien (falls geplant). Man muss also unterscheiden, ob ein Merkmal im Baum oder in der Komponente obligatorisch ist. In Tab. 4-1 sind alle Kombinationen dargestellt.

Merkmalbaum	Komponente	
optional	optional	erlaubt
obligatorisch	optional	erlaubt
optional	obligatorisch	nur erlaubt, wenn Komponente entfernt werden kann, ohne andere Merkmale zu beeinflussen
obligatorisch	obligatorisch	erlaubt

Tab. 4-1: Erlaubte Kombinationen für die Umsetzung eines Merkmals

Um im Einsatz Probleme zu vermeiden, ist es von Vorteil entweder nur ein Merkmal pro Komponente zu implementieren oder jedes Merkmal einzeln abschaltbar zu gestalten.

Komponente C enthält nur ein obligatorisches Merkmal FM4. Das optionale Strukturmerkmal SM4 könnte entfallen. Damit kann das Merkmal FM4 in ein optionales Merkmal umgewandelt werden. Es wurde jedoch eingefügt, damit die Komponenten im Alpha-System durch Strukturmerkmale repräsentiert werden können. Komponente C kann nicht vom Kern direkt benutzt werden. Der Kern kann C nur indirekt unter Nutzung der Komponente A ansprechen. Unter der Annahme, dass entgegen der Vorgabe im Beispiel die Komponente A nur optionale Merkmale enthält und der Kunde nur das Merkmal FM4 einsetzen möchte, muss automatisch auch Komponente A installiert werden, obwohl keinerlei Merkmale dieser Komponente genutzt werden sollen. Sie würde dann als Schnittstellenadapter dienen.

Die letzte Komponente im Beispiel ist vom Typ B. Sie beinhaltet die alternativen Merkmale FM7 und FM8. Genau eines dieser beiden Merkmale muss ausgewählt werden, beide Merkmale gleichzeitig sind nicht möglich. Beispielsweise kann es eine Komponente für die Steuerung eines Prozesses sein, die sowohl manuelle als auch automatische Steuerung zulässt. Diese beiden funktionalen Merkmale schließen sich immer aus. Mittels Konfiguration der Komponente B muss dann die automatische Steuerung ein- oder ausgeschaltet werden können.

In Abb. 4-6 ist der Merkmalbaum zusammen mit Komponenten dargestellt. Die Komponenten implementieren dort den kompletten darunterliegenden Merkmalzweig; eine explizite Kennzeichnung an jedem Merkmal ist nicht notwendig. Komponente A imp-

lementiert den kompletten Zweig unterhalb von SM2 mit Ausnahme von SM4, das in Komponente C eingebaut wurde.

Anhand des Alpha-Beispiels wird deutlich, wie sehr das Merkmalmodell die Architektur einer Produktlinie beeinflusst. Komponenten, die mehrere funktionale Merkmale implementieren, müssen auf die Beziehungen zwischen den Merkmalen Rücksicht nehmen. Das Vorhandensein von Strukturmerkmalen gibt bereits Aufschluss, an welchen Stellen sich der Einsatz von Komponenten lohnen kann. Das besondere am Alpha-Beispiel ist der Einsatz von Komponenten auf Zweigebene. Keine Komponente implementiert Merkmale eines anderen Zweiges des Merkmalbaumes. Durch gezieltes Einfügen von Strukturmerkmalen können Verzweigungen erreicht werden. An diesen Punkten lässt sich der Baum in eine Komponentenstruktur aufgliedern (Abb. 4-6).

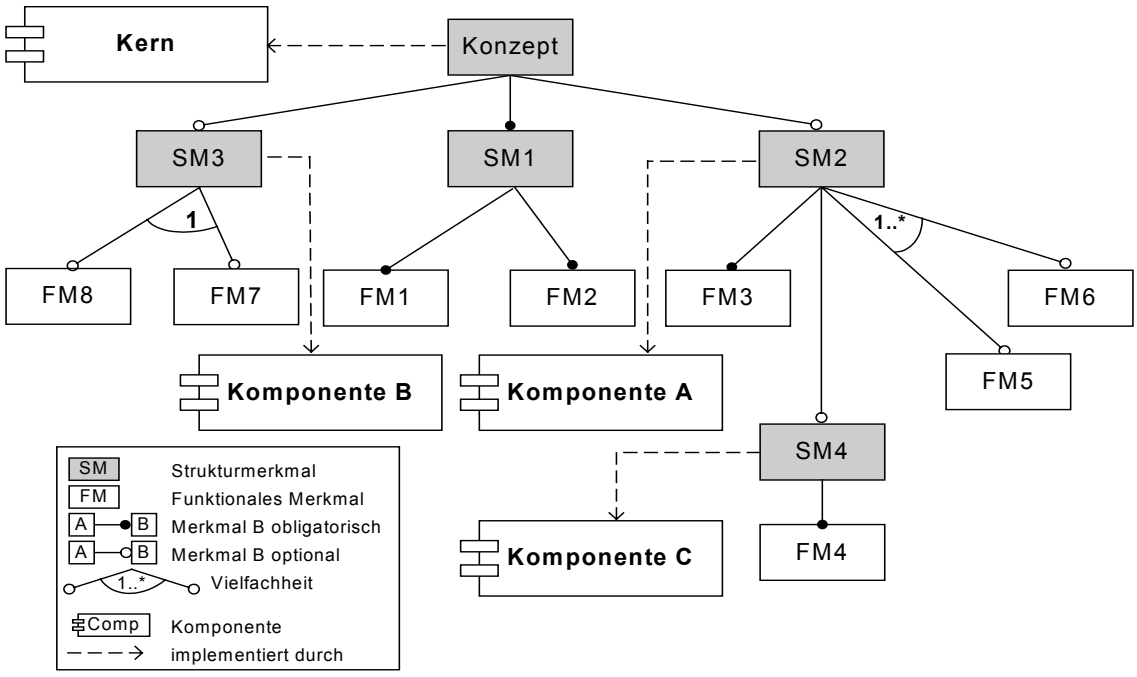

Abb. 4-6: Alpha-System: Merkmalbaum mit Komponenten

Das Alpha-System zeigt, wie ein Merkmalbaum sich mit einer Komponentenstruktur im Idealfall verknüpfen lässt. In der Praxis ist eine derartige Struktur, wenn sie nicht von Anfang an darauf konzipiert wurde, eher unwahrscheinlich. Es gibt sowohl innerhalb als auch außerhalb von Merkmalzweigen Überschneidungen. Das Beispiel im nächsten Kapitel wird sich dieser Problematik annehmen.

4.2.2. Architektur einer merkmalbasierten Produktlinie II

Die Produktlinie Tonstudio Abb. 4-7 soll als weiteres Beispiel für die komplexen Zusammenhänge zwischen den Komponenten dienen. Nach dem Idealfall des ersten abstrakten Beispiels soll dieses die Probleme in einer realeren Umgebung aufdecken. Beim Tonstudio handelt es sich um eine fiktive frei verfügbare Software, deren Schnittstellen wohl definiert offengelegt sind. Jeder, der den Funktionsumfang erweitern möchte, kann Komponenten dafür entwickeln. Auf die Darstellung der Merkmale des Kerns im Baum wurde verzichtet.

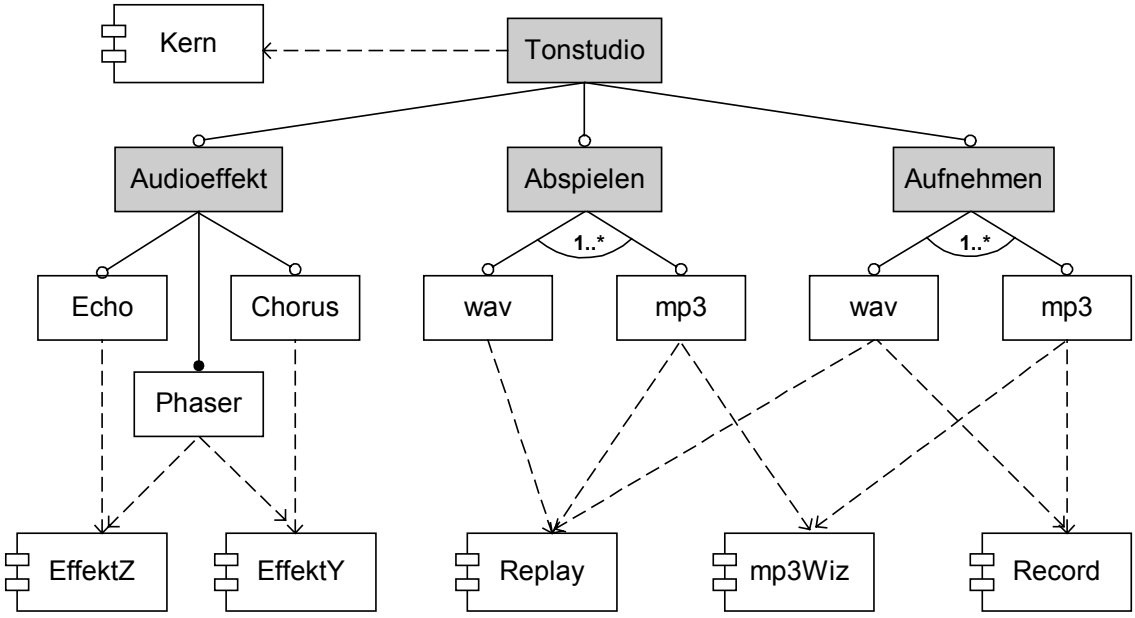

Abb. 4-7: Beispiel – Produktlinie Tonstudio

Das Tonstudio ermöglicht optional die Aufnahme und das Abspielen von MP3 und WAV-Dateien. Weiterhin können die Musikdaten mit Effekten gefiltert werden. Wie man sieht, gibt es hier keine klare Komponentenstruktur. Eine Komponente kann beliebige Anforderungen erfüllen. Es wurde nicht festgelegt, welche Komponente welche Merkmale zu implementieren hat. Auch könnten die Komponenten wiederum Schnittstellen für eigene Erweiterungen bereitstellen.

Für die Merkmalzweige *Abspielen* und *Aufnehmen* sind die Probleme auf den ersten Blick nicht gravierend. Ein Kunde, der MP3 und WAV sowohl aufnehmen als auch abspielen möchte, wird hier immer ein funktionierendes System geliefert bekommen. Wie sich aus dem vorigen Kapitel schlussfolgern lässt, muss in diesem Beispiel jede Komponente ihre Funktionen optional anbieten können. Das eigentliche Problem ist hier, mit welcher Komponente welche Anforderung erfüllt werden soll. Ohne zusätzliche Ent-

scheidungshilfe unter Verwendung von Eigenschaften der Komponente oder deren einzelner Merkmale kann hier eine Auswahl nur nach dem Zufallsprinzip erfolgen.
Das bedeutet, der Kunde muss seine Präferenzen in die Auswahl einfließen lassen. Dies kann im Notfall auch durch den Komponentenmonteur erfolgen, wenn dem Kunden hier das Wissen fehlt. Hier sind die Beschaffenheitsmerkmale von entscheidender Bedeutung. Es kann eine Untergliederung nach globalen Beschaffenheitsmerkmalen, die für jede Komponente erfasst werden können, und speziellen Beschaffenheitsmerkmalen, die nur einzelne Funktionen betreffen, vorgenommen werden. Ein spezielles Beschaffenheitsmerkmal für das Tonstudio wäre z. B. die Aufnahmequalität des Merkmals *Aufnehmen – MP3*.
Ein weiterer wichtiger Punkt ist die Auswertbarkeit der Beschaffenheitsmerkmale. Das Lizenzmodell lässt sich zweifelsfrei komplizierter auswerten als der Preis. Entweder sollten die Präferenzen eindeutig zuordenbar (z. B. Hersteller) oder vergleichbar (kleiner als, größer als) sein.

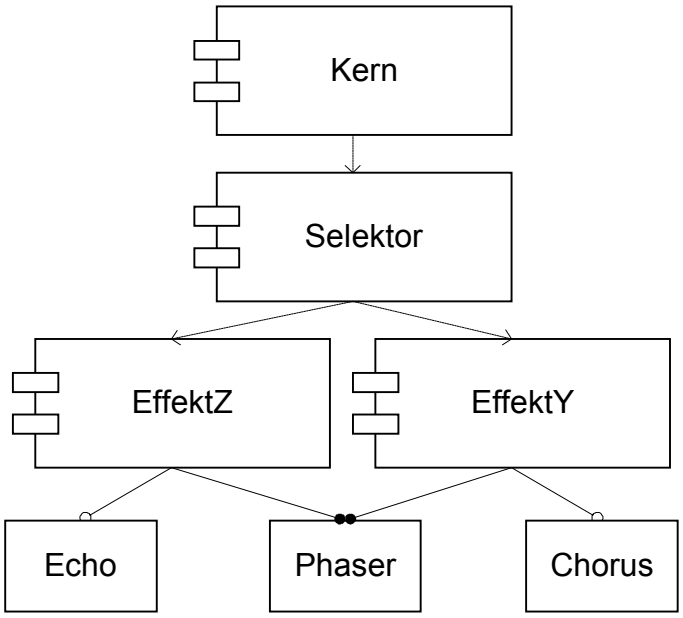

Abb. 4-8: Selektive Komponente

Nun zu den Audioeffekten. Die Produktlinie schreibt vor: Wenn ein Audioeffekt installiert wird, muss auf jeden Fall der *Phaser* zur Verfügung gestellt werden. Er ist also obligatorisch. Zwei Anbieter entwickelten Effektkomponenten für das Tonstudio. *EffektY* kann neben dem Phaser auch noch einen *Echo*-Effekt bieten, *EffektZ* enthält zusätzlich einen *Chorus*-Effekt.
Entscheidet sich ein Kunde dafür, dass er sowohl *Echo* als auch *Chorus* zusätzlich zum *Phaser*-Effekt in seiner Applikation haben möchte, können Schwierigkeiten auftreten.

Wäre der *Phaser* in der Produktlinie optional, könnte man ihn in einer Komponente abschalten (siehe Diskussion im vorangegangenen Kapitel). Den einzigen Ausweg kann hier nur noch eine übergeordnete Komponente bieten, indem sie die Auswahl des *Phasers* aus einer der beiden Komponenten ermöglicht und den anderen abschaltet. Eine Komponente, die gleichzeitig mehrere Komponenten über eine Schnittstelle ansteuern kann, sollte somit auch eine Möglichkeit besitzen, einzelne Funktionen explizit aus untergeordneten Komponente zu nutzen.
Dies kann z. B. über die Konfiguration oder über komponentenspezifische Bedienelemente erfolgen. In der Abb. 4-8 wurde eine zusätzliche Komponente *Selektor* eingeführt, die es gestattet, für das Merkmal *Phaser* aus einer der beiden implementierenden Komponenten auszuwählen. Ist dies nicht der Fall, sind die Anforderungen des Kunden nicht erfüllbar.

4.2.3. Fazit

Diese beiden als Beispiel vorgestellten Produktlinien haben eine Reihe von Problemen und Anforderungen offenbart, die eine automatisierte Ableitung berücksichtigen muss.
Komponenten müssen die Anforderungen, die im Merkmalmodell wiedergespiegelt werden, rückhaltlos erfüllen. Sind Merkmale im Modell optional, muss entweder das Merkmal in der Komponente abschaltbar sein oder die Komponente als Gesamtes entfernt werden können. Überschneiden sich Komponenten in ihrer Funktionalität, muss eine ungestörte nebeneinander funktionierende Arbeitsweise gewährleistet sein. Überschneidende Merkmale müssen dann entweder abschaltbar oder explizit ansprechbar sein. Um die Auswahl der richtigen Komponente aus einer Vielzahl passender Kandidaten zu ermöglichen, sollte sie durch Beschaffenheitsmerkmale beschrieben werden, sonst muss der Zufall die Entscheidung übernehmen. Weiterhin müssen auch Komponenten in das System eingebunden werden, die keinen direkten Bezug zur Merkmalauswahl vorweisen. Das können Komponenten sein, die als Schnittstellenadapter eingesetzt oder durch die ausgewählten Komponenten eingebunden werden.
Ein Merkmalbaum kann bereits in der Anforderungsanalyse gute Hinweise geben, wie die Struktur der Komponenten in der Produktlinie aussehen kann. Er deutet auf deren Schnittstellen und Funktionsumfang hin.

4.3. Komponentenmodell

Das FORE-Datenmodell wird um das Komponentenmodell erweitert. In Abb. 4-9 sind die Bestandteile des ursprünglichen FORE-Datenmodells grau hinterlegt. Die übrigen Bestandteile stellen das neue Komponentenmodell dar. Es enthält eine Auflistung sämtlicher für diese Produktlinie verfügbaren Komponenten. Das allgemeinere UML-Komponentenmodell wird den hier gestellten Anforderungen nicht gerecht. Die Möglichkeiten zur Modellierung der Abhängigkeiten zwischen Merkmalen und Komponenten sind nicht ausreichend. Das präsentierte Modell integriert sich nahtlos in das FORE-Datenmodell.

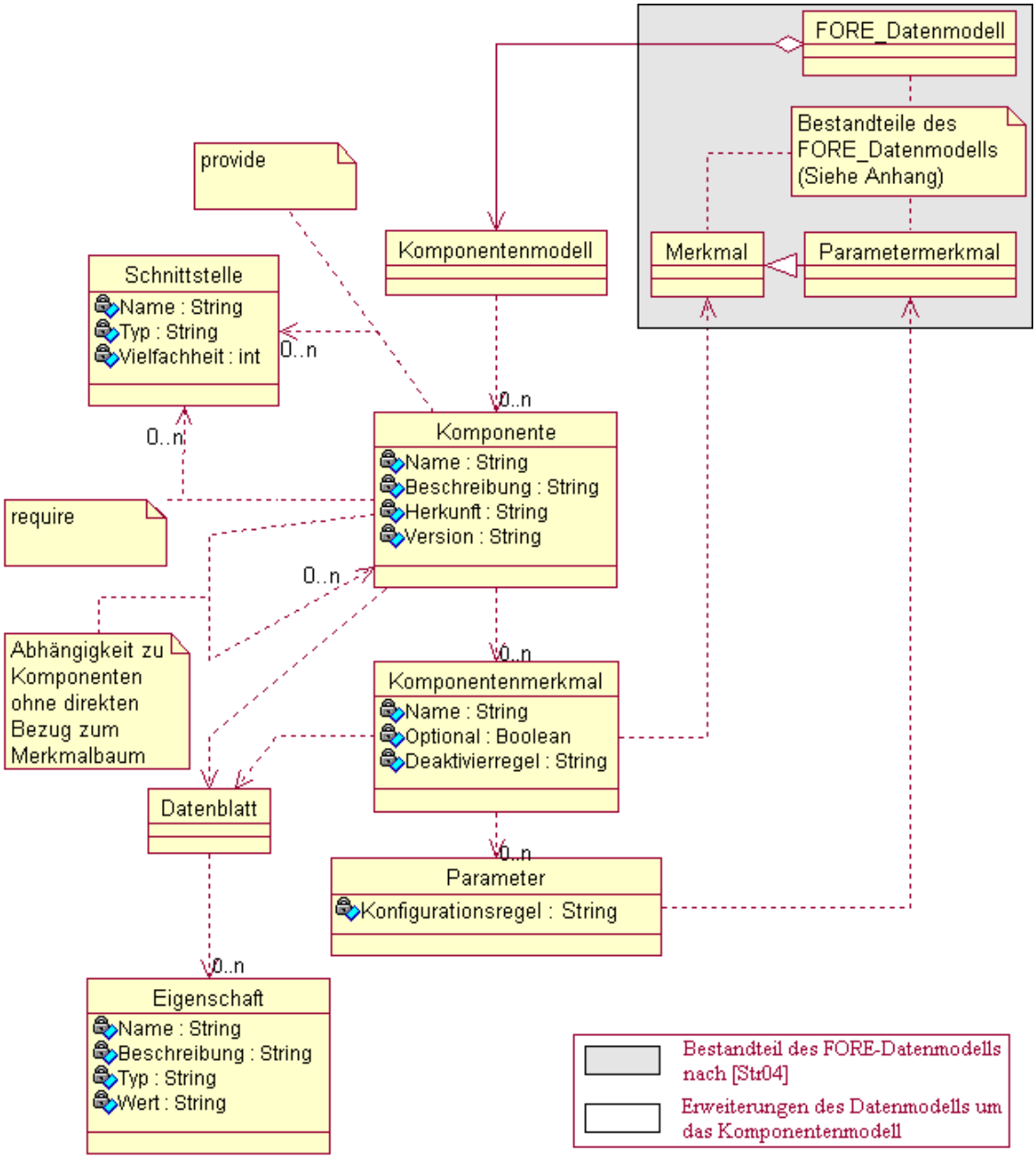

Abb. 4-9: Komponente im erweiterten FORE-Datenmodell

Jeder Komponente ist ein Datenblatt zugeordnet, das alle für die automatische Auswahl relevanten Beschaffenheitsmerkmale der Komponente enthält. Für jede dieser Eigenschaften ist der Name, eine Beschreibung, der Wertetyp und der Wertebereich hinterlegt. Solche Beschaffenheitsmerkmale können beispielsweise Preis, Hersteller, Ressourcenverbrauch oder Lizenzmodell sein.
Alle in der Komponente implementierten funktionalen Merkmale sind mit für die Ableitung relevanten Informationen, wie Parametern und Eigenschaften, versehen. Eine Referenz zu im FORE-Merkmalmodell enthaltenen Merkmalen stellt eine eindeutige Beziehung her. Es wird gekennzeichnet, ob ein funktionales Merkmal in der Komponente optional oder obligatorisch ist. Wenn das Merkmal optional ist, wird eine Regel definiert, mit der es abgeschaltet werden kann. Zugewiesene Parametermerkmale ändern ebenfalls über eine Regel die Konfiguration der Komponente.
In welcher Sprache eine Regel implementiert ist, wird nicht näher spezifiziert. Die Regel ist abhängig vom Programm, das dieses Modell auswertet. Im Beispielprojekt besitzt jede Komponente eine ANT-Projektdatei. Hier sind alle Prozesse, die zum Abschalten eines Merkmals oder der Änderung eines Komponentenparameters notwendig sind, in der ANT-Skriptsprache hinterlegt. Das Programm zur Ableitung der Applikation ruft jeweils den entsprechenden Teil des ANT-Skriptes auf, der zur Änderung der Komponentenkonfiguration notwendig ist. So besteht eine Regel zum Abschalten eines Merkmals einer Komponente nur aus dem Namen eines ANT-Target (z. B. „disableFeature2“). Der Aufruf des Targets ändert die Konfiguration der Komponente dahingehend, dass die unerwünschte Funktion abgeschaltet wird. Die Art und Weise, wie ein Merkmal durch das ANT-Skript abgeschaltet wird, ist von Komponente zu Komponente verschieden. Äquivalent funktioniert eine Regel für die Änderung eines Parameters einer Komponente. Der Unterschied zur vorhergehenden Regel besteht in der zusätzlichen Übergabe des Parameters durch das Programm zur Ableitung an das ANT-Skript.
Ebenso wie die Komponente besitzt auch ein funktionales Merkmal ein Datenblatt. Hier sind Beschaffenheitsmerkmale wie Performance (z. B. als Performanceindex) oder Flexibilität hinterlegt.
Schnittstellenmerkmale finden sich im Merkmalbaum nicht wieder, da der Baum keine Aussage über die der Produktlinie zugrundeliegenden Architektur darstellt. Im vorliegenden Modell werden den Komponenten Interfaces zugeordnet. Sie besitzen einen Namen, einen Typ und eine Vielfachheit.

Diese Schnittstellen werden durch den Typ in zwei Gruppen unterteilt: *provide* und *require*. Zwei Komponenten, die je ein Interface mit gleichem Namen aber unterschiedlichen Typ besitzen, können miteinander gekoppelt werden. Somit kann eine Komponente, die eine Schnittstelle mit dem Namen „abc“ und dem Typ *require* besitzt, mit einer Komponente verbunden werden, die eine Schnittstelle mit dem gleichem Namen und dem Typ *provide* enthält.

Die Vielfachheit gibt an, wie viele Komponenten gleichzeitig über eine Schnittstelle angesprochen werden können. Kann mehr als eine Komponente parallel benutzt werden, dann wird angenommen, dass die einbindende Komponente eine Managementfunktion besitzt, um eine Komponente gezielt anzusprechen. Auf diese Weise können Merkmale, welche die gleiche Funktionalität verkörpern, gezielt aus verschiedenen Komponenten benutzt werden.

Weiterhin existiert für jede Komponente eine Liste von Komponenten, die außerhalb der ausgewählten Merkmale benötigt werden. Das können z. B. Bibliotheken sein, die nicht durch die Merkmalauswahl repräsentiert werden, aber für die Implementierung unabdinglich sind.

Es wird davon ausgegangen, dass alle Funktionen über eine Schnittstelle zur Verfügung gestellt werden. Die Möglichkeit, dass über verschiedene Schnittstellen unterschiedliche Funktionen bereitgestellt werden, wird im Rahmen der Arbeit nicht berücksichtigt. Eine Erweiterung des Ansatzes in diese Richtung ist möglich, würde aber am Grundprinzip der Methode nichts ändern.

4.4. Komponentenwahl

In diesem Kapitel erfolgt der wichtigste Schritt der Applikationsableitung: die Auswahl und Zusammenstellung der benötigten Komponenten. Die Applikationsableitung besteht aus zwei Teilprozessen (Abb. 4-10). Im ersten Schritt werden die Komponenten ausgewählt und im zweiten Schritt zu einer installierbaren Applikation zusammengefügt. Zuvor wurde aus dem Merkmalmodell eine Merkmalauswahl erzeugt, validiert und im Systemfamilienmodell hinterlegt

Bei der Komponentenwahl werden zwei verschiedene Szenarien betrachtet. Das erste Szenario besteht in der Erstellung einer komplett neuen Installation. Die Merkmalauswahl dient hier als alleinige Basis der Auswahl.

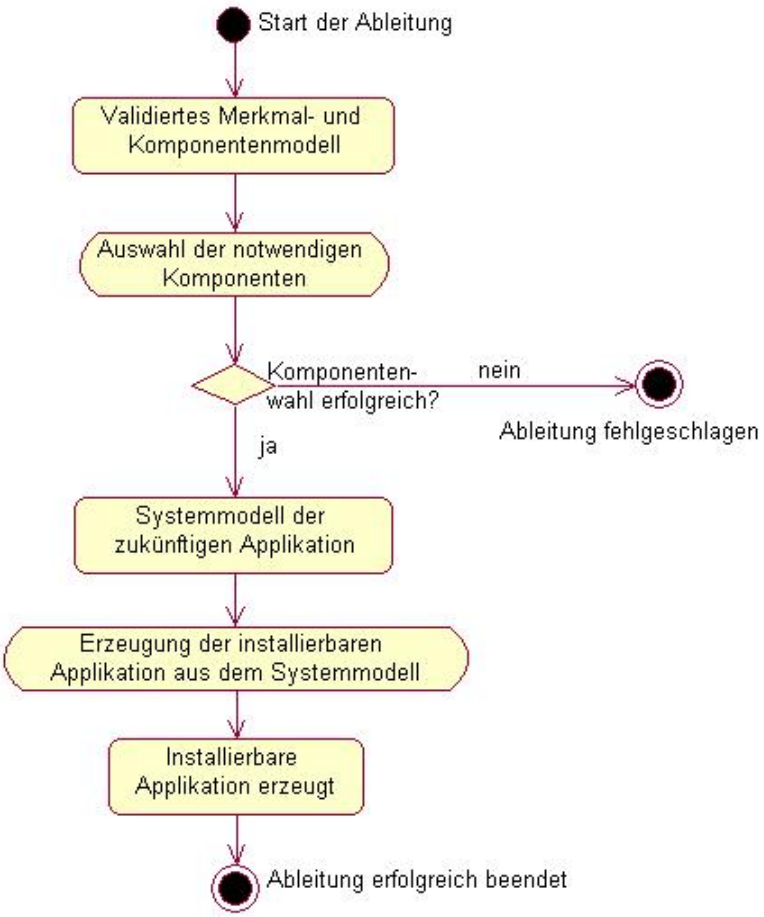

Abb. 4-10: Übersicht des Ableitungsprozesses

Das zweite Szenario befasst sich mit der Änderung einer bestehenden Applikation. Hier werden neben der Merkmalauswahl auch die bereits eingesetzten Komponenten einbezogen. Dieses Szenario wird im Kapitel 4.6 untersucht.

Im vorliegenden Kapitel wird eine Methode zur Erstellung einer Anwendung vorgestellt. Anhand eines kleinen Beispiels wird sie anschließend erläutert.

4.4.1. Prozess der Komponentenwahl

Die Erstellung einer Applikation ist die Überführung der Merkmale einer Variante der Produktlinie in ein Paket, bestehend aus dem Systemkern und den für die ausgewählten Merkmale notwendigen Komponenten. Es beinhaltet nicht die Installation der Komponenten auf ein System und die Anpassung an dessen Gegebenheiten. Diesen Prozess nennt man Deployment, es folgt auf die Ableitung. Der Prozess der Komponentenwahl ist in Abb. 4-11 dargestellt.

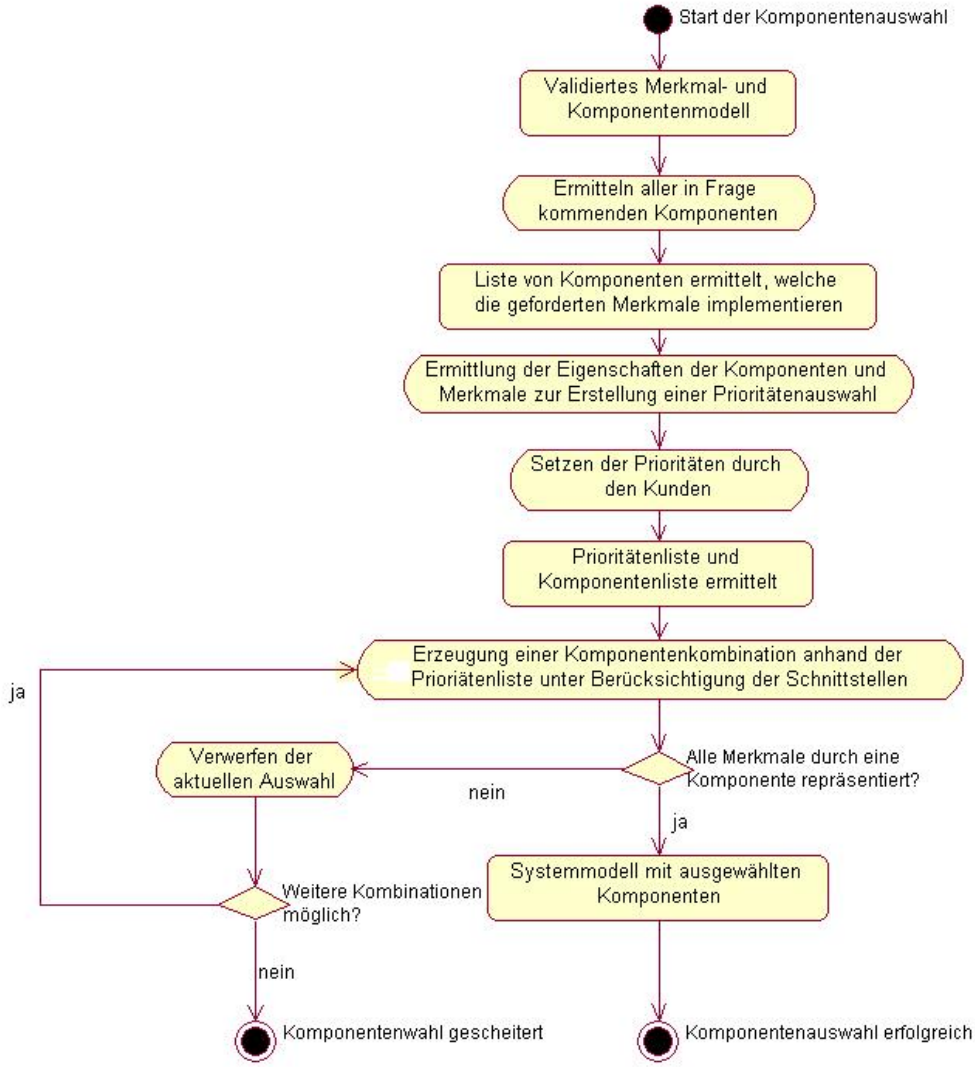

Abb. 4-11: Prozess der Komponentenwahl

Die Auswahl der Komponenten erfolgt zunächst durch die direkte Zuordnung der funktionalen Merkmale der Systemvariante zu den funktionalen Merkmalen der Komponenten. Ausgehend vom Konzept, der Wurzel des Merkmalbaumes, werden die benötigten Komponenten ermittelt. Das Konzept stellt den Systemkern der Applikation dar, welcher durch die Komponenten erweitert werden kann. Er bildet den ersten Teil der zukünftigen Applikation und wird selbst als Komponente angesehen. Dabei ist zu beachten, dass bereits der Kern funktionale Merkmale der Auswahl repräsentieren kann. Es wird davon ausgegangen, dass der Kern keine optionalen Merkmale enthält. Alle optionalen Merkmale werden durch zusätzliche Komponenten repräsentiert. Die Beschaffenheit des Systemkerns ist hier nicht von Bedeutung. Er kann selbst weitere Komponenten benötigen, um alle obligatorischen Merkmale bereitstellen zu können.

Ausgehend von der Wurzel des Merkmalbaumes werden jedem Merkmal mögliche Komponenten zugeordnet. Es spielt hier zunächst keine Rolle, ob ein Merkmal bereits durch eine Komponente repräsentiert wird. Jede Komponente, die ein Merkmal der Auswahl implementiert, kommt in den engeren Kreis der Auswahl. Kann ein Merkmal keiner Komponente zugeordnet werden, ist die Ableitung als gescheitert anzusehen.
Im nächsten Schritt gilt es unerwünschte Überschneidungen zu beseitigen. Hier spielen die Beschaffenheitsmerkmale der Komponenten und Merkmale eine entscheidende Rolle. Sind keine Präferenzen gegeben oder besitzen alle Komponenten keine der zur Auswertung benötigten Eigenschaften, muss der Zufall entscheiden. Die Auswahl beschränkt sich dann darauf, jedes Merkmal durch eine Komponente repräsentiert zu wissen.
Für obligatorische Merkmale gilt es einen Sonderfall zu beachten. Merkmale, die im Merkmalbaum als obligatorisch ausgelegt wurden, können in einer Komponente auch optional implementiert sein, d.h. die Komponente könnte auch dann verwendet werden, wenn im Baum dieses Merkmal optional sein würde. Man muss also unterscheiden, ob ein Merkmal im Baum oder in der Komponente obligatorisch ist (s.a. Tab. 4-1). Wird eine Komponente gewählt, deren andere ebenfalls in der Auswahl enthaltenen Merkmale obligatorisch umgesetzt wurden, muss sie für diese Merkmale verwendet werden.
Die Auswahl muss durch eine Entscheidungsmethode gefällt werden. Sowohl MAUT als auch die Prioritätenliste haben Vor- als auch Nachteile, die in Kapitel 2.3 dargelegt wurden.
Bei der Anwendung von MAUT ist es für einen Kunden, der eine Applikation zu einem erschwinglichen Preis erwerben will, schwer nachzuvollziehen, warum so viele Rahmenbedingungen vorgegeben werden müssen. Hinzu kommt, dass die Substitutionsrate sich von Kunde zu Kunde unterscheidet. Ein Kunde würde für 10% bessere Kompressionsrate einen 20 € höheren Preis akzeptieren, ein anderer vielleicht nur 5 €. Die Mehrzahl der Kunden kann diese Frage gar nicht beantworten.
Anders betrachtet würden Kunden, die eine Anwendung für strategische Ziele erwerben und dafür auch eine große Summe investieren wollen, sich nicht mit einer einfachen Prioritätenliste zufrieden geben. Für eine Software, welche die nächsten Jahre eine Stütze des Unternehmens darstellt, ist das Beste gut genug. Wenn sich das Unternehmen auf Standardsoftware festlegt und eine Individualentwicklung ausschließt, dann ist in diesem Bereich eher die MAUT-Methode zu finden. In [Wal02] wird ein kompletter

MAUT-Prozess bestehend aus mehreren teilweise iterativen Teilprozessen vorgestellt, der für eine gute Evaluierung notwendig ist.

Welche Methode eingesetzt werden soll, hängt gänzlich von den Rahmenbedingungen des Kaufes ab. Die Zielgruppe der automatisierten Ableitung liegt eher im Bereich der Heimanwender und weniger bei den Enterpriselösungen. Jede noch so gut konfigurierbare Softwarelösung kommt bei strategischen Anwendungen schnell an ihre Grenzen und muss manuell angepasst und erweitert werden.

Der Vorteil der Prioritätenliste besteht in ihrer Einfachheit. Sie ist für einen Kunden leicht verständlich. Er kann mit ihrer Hilfe schnell zu einer guten Auswahl kommen. Ihr Nachteil besteht darin, dass das Ergebnis nicht unbedingt dem Optimum entsprechen muss. Aber um dieses zu ermitteln, ist eine wesentlich aufwendigere Informationserfassung zu betreiben. Die komplexen Zusammenhänge müssten dem Kunden begreiflich sein und würden ein umfangreiches Hintergrundwissen voraussetzen. Im vorliegenden Ansatz wird deshalb die Prioritätenliste angewendet.

Werden durch den Kunden keine Prioritäten gesetzt, dann wird die Auswahl der Komponenten so getroffen, dass die Anforderungen durch eine möglichst kleine Anzahl umsetzt werden. Dadurch werden die Risiken für deren fehlerfreies Zusammenspiel minimiert. Stehen am Ende des Entscheidungsprozesses mehrere gleichwertige Komponenten, entscheidet der Zufall, welche eingesetzt wird.

Die Auswertung von Beschaffenheitsmerkmalen der Komponenten geschieht ganzheitlich, d.h. für die Minimierung bzw. Maximierung werden alle zulässigen Kombinationen der Komponenten überprüft. Dabei werden auch die Abhängigkeiten zu Komponenten berücksichtigt, die nicht durch die Merkmalauswahl selektiert wurden, aber für das Zusammenspiel der ausgewählten Komponenten unbedingt notwendig sind. Das sind zum einen die Komponenten, die von einer ausgewählten Komponente benötigt werden. Zum anderen werden die Komponenten einkalkuliert, die es dem Kern erst ermöglichen, auf die gewählte Komponente zuzugreifen. Das können sowohl reine Adapterkomponenten sein als auch Komponenten, deren direkte Funktionen für die abzuleitende Applikation nicht benötigt werden und deshalb abgeschaltet sein müssen.

Bei den Beschaffenheitsmerkmalen eines funktionalen Merkmals werden nur die Komponenten in die Kombination einbezogen, die dieses Merkmal implementieren.

Die Auswahl ist beendet, sobald für jedes funktionale Merkmal eine Komponente ausgewählt wurde. Das Endergebnis wird in einem Systemmodell gespeichert, dessen Aufbau im Kapitel 4.5 erläutert wird.

4.4.2. Beispiel - Tonstudio II

Anhand eines Beispiels soll die Vorgehensweise bei der Komponentenauswahl besser verständlich gemacht werden. Als Entscheidungsmethode wird aus den eben erläuterten Gründen die Prioritätenliste verwendet.

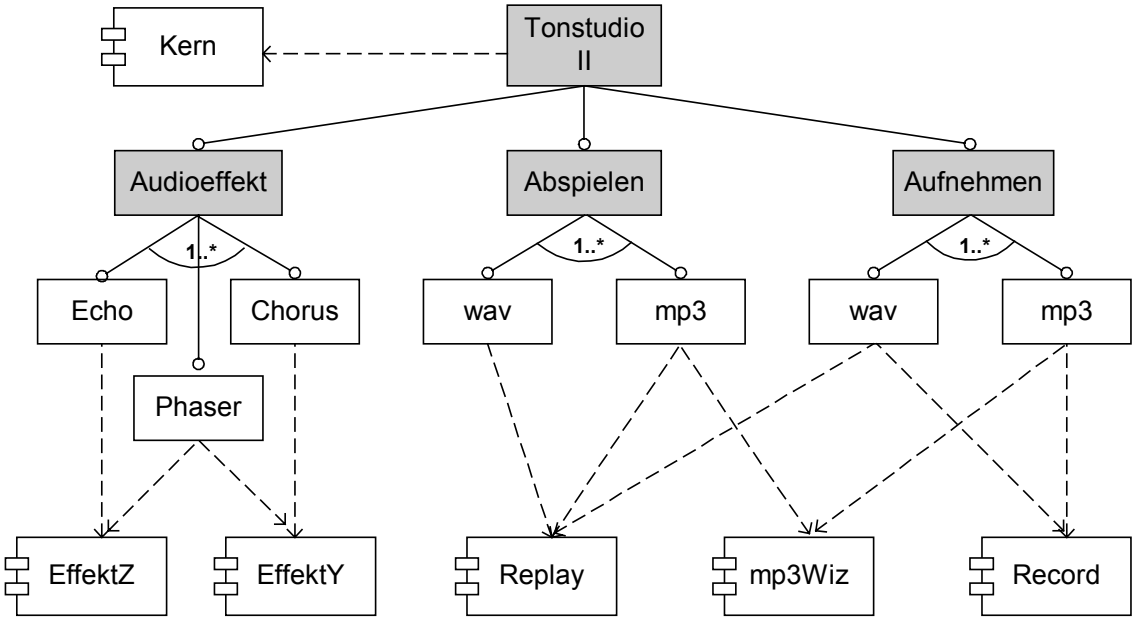

Abb. 4-12: Beispielsystem Tonstudio II

Beim Beispiel Tonstudio II (Abb. 4-12) handelt es sich um eine abgeänderte Variante des weiter oben eingeführten Beispiels Tonstudio. Das obligatorische Merkmal *Phaser* wurde durch ein gleichnamiges optionales Merkmal ersetzt. Die implementierenden Komponenten *EffektY* und *EffektZ* können dieses Merkmal jetzt abschalten. Damit ist der Konflikt, der im Tonstudio-Beispiel existierte, behoben.

In diesem Beispiel werden alle Komponenten vom Kern direkt eingebunden. Keine Komponente ist von einer anderen abhängig. Es müssen also keine Adapter oder Zusatzkomponenten berücksichtigt werden.

Die Beschaffenheitsmerkmale der zur Auswahl stehenden Komponenten sind in Tab. 4-2 aufgelistet. Auch der Kern wird als eine Komponente angesehen.

Komponente	Preis	Speicherplatz	Hersteller
Kern	50 €	200 KB	Basic
EffektY	50 €	50 KB	Mannhof
EffektZ	60 €	50 KB	Sicromoft
Replay	90 €	100 KB	Sicromoft
mp3Wiz	60 €	60 KB	Mannhof
Record	60 €	50 KB	Nosy

Tab. 4-2: Eigenschaften der Tonstudio II - Komponenten

Weiterhin wurden für die Merkmale einiger Komponenten zusätzliche Beschaffenheitsmerkmale ermittelt. Diese sind in Tab. 4-3 dargestellt.

Merkmal	Komponente	Kompressionsrate	CPU-Last auf einem Referenzsystem
Aufnahme–mp3	Record	128 KBps	
	mp3Wiz	192 KBps	
Abspielen-mp3	Replay		10%
	mp3Wiz		8%

Tab. 4-3: Eigenschaften ausgewählter Tonstudio II - Merkmale

Ein Kunde möchte eine Tonstudio II-Applikation erwerben. Er geht zu einem *Komponentenmonteur* und teilt ihm seine Anforderungen mit. Dieser sucht folgende Merkmale aus:

- Audioeffekt – Phaser
- Audioeffekt – Chorus
- Abspielen – wav
- Abspielen – mp3
- Aufnahme – mp3

Als erster Schritt der Komponentenwahl steht die Zuordnung von Merkmalen zu Komponenten, ihre Eigenschaften bleiben jetzt noch unberücksichtigt. Ausgehend von der Wurzel werden die notwendigen Komponenten bestimmt. Zunächst wird der Systemkern ausgewählt. Die Komponente Kern, die den Systemkern repräsentiert, implemen-

tiert die Strukturmerkmale *Abspielen*, *Aufnehmen* und *Audioeffekt* als Schnittstelle für weitere Komponenten.
Beim weiteren Hinabsteigen in die Zweige des Merkmalbaumes werden Komponenten ermittelt, welche die Anforderungen an das System erfüllen können. Die erfolgte Auswahl ist in Tab. 4-4 zu sehen.

Merkmal	Komponenten
Audioeffekt – Phaser	→ EffektY → EffektZ
Audioeffekt – Chorus	→ EffektY
Abspielen – wav	→ Replay
Abspielen – mp3	→ Replay → mp3Wiz
Aufnahme – mp3	→ mp3Wiz → Record

Tab. 4-4: Ermittelte Komponenten für Tonstudio II

Damit ist der erste Schritt zu Auswahl vollendet. Wie man sieht, gibt es für zwei Merkmale keine Alternativen. Sowohl *EffektY* (Merkmal *Chorus*) als auch *Replay* (Merkmal *Abspielen-wav*) müssen in der Applikation enthalten sein. Sie können also schon jetzt verbindlich für das System eingeplant werden. Für die drei anderen Merkmale ist die Entscheidung noch nicht gefallen, welche Komponente hier eingesetzt werden soll.
Für die Überschneidungen müssen die Prioritäten des Kunden ermittelt werden. Die Prioritätenliste kann aus allen zur Verfügung stehenden Beschaffenheitsmerkmalen aufbaut werden, sowohl aus denen der Komponenten als auch der funktionalen Merkmale.
Die Prioritäten des Kunden sehen so aus:

1. Abspielen-mp3 CPU-Last → minimal
2. Preis → minimal
3. Hersteller = „Mannhof“

Im nächsten Schritt werden die Prioritäten des Kunden ausgewertet. Im ersten Punkt scheidet die Replay-Komponente für das Merkmal *Abspielen-mp3* aus. Die CPU-Last des Merkmals vom *mp3Wiz* ist mit 8% niedriger als die 10% der *Replay*-Komponente. Damit wird dieses Merkmal allein durch den *mp3Wiz* repräsentiert.

Der zweite Schritt dient der Preisminimierung. Dazu können noch vier Kombinationen ausprobiert werden. Dies sind jeweils zwei Möglichkeiten beim Merkmal *Phaser* und zwei beim Merkmal *Aufnahme-mp3*:

1. Gesamtpreis = Kern + EffektY + Replay + mp3Wiz
   = 50 € + 50 € + 90 € + 60 €
   = 250 €
   (Phaser, Chorus → EffektY
   Abspielen-wav → Replay
   Abspielen-mp3, Aufnahme-mp3 → mp3Wiz)

2. Gesamtpreis = Kern + EffektZ + EffektY + Replay + mp3Wiz
   = 50 € + 50 € + 60 € + 90 € + 60 €
   = 310 €
   (Phaser → EffektZ
   Chorus → EffektY
   Abspielen-wav → Replay
   Abspielen-mp3, Aufnahme-mp3 → mp3Wiz)

3. Gesamtpreis = Kern + EffektY + Replay + mp3Wiz + Record
   = 50 € + 50 € + 90 € + 60 € + 60 €
   = 310 €
   (Phaser, Chorus → EffektY
   Abspielen-wav → Replay
   Abspielen-mp3 → mp3Wiz
   Aufnahme-mp3 → Record)

4. Gesamtpreis = Kern + EffektY + EffektZ + Replay + mp3Wiz + Record
   = 50 € + 50 € + 60 € + 90 € + 60 € + 60 €
   = 370 €
   (Phaser → EffektZ
   Chorus → EffektY
   Abspielen-wav → Replay
   Abspielen-mp3 → mp3Wiz
   Aufnahme-mp3 → Record)

Die Komponenten *mp3Wiz* und *EffektY* sind bereits durch andere Merkmale fest in der Applikation verankert, während *EffektZ* und *Record* jeweils für die strittigen Merkmale explizit eingebunden werden müssten. Die preisgünstigste Variante wäre demnach Variante 1 mit 250 €.

Jetzt ist jedem Merkmal exakt eine Komponente zugeordnet. Punkt 3 der Prioritätenliste kann somit unberücksichtigt bleiben. Die finale Auswahl der Komponenten sieht demnach so aus: Neben dem Kern gehören die Komponenten *EffektY*, *Replay* und *mp3Wiz* zur zukünftigen Applikation.

4.5. Systemmodell

Die ermittelte Komponentenkombination wird im FORE-Datenmodell gespeichert. Dazu wird das *Systemmodell* eingeführt (Abb. 4-13). Es enthält die Konfiguration aller Komponenten, die im System eingesetzt werden. Unter Verwendung des System- und Komponentenmodells kann durch ein externes Werkzeug ein Installationsprogramm erzeugt werden.

Im Systemmodell wird neben einer Änderungshistorie der Name, eine Beschreibung und die Herkunft der Systemkonfiguration beschrieben. Kernpunkt des Modells ist aber eine Auflistung aller verwendeten Komponenten und deren Konfigurationen. Für jede Komponente werden die Abhängigkeiten, die genutzten funktionalen Merkmale und deren Parameter gespeichert. Es werden jeweils nur die Referenzen zum Komponenten- bzw. Merkmalmodell gespeichert. Weiterhin kann es einen Verweis auf die ursprüngliche Konfiguration bzgl. der Merkmalauswahl geben. Um die Auswahl später noch nachvollziehen zu können, wird auch die Prioritätenliste im Systemmodell gespeichert. Somit kann gewährleistet werden, dass die in der Komponentenauswahl gefällten Entscheidungen reproduzierbar sind.

Nach der Ableitung wird eine Kopie dieses Systemmodells der Applikation beigefügt. Auf diese Weise ist eine Erweiterung bzw. Aktualisierung des Systems möglich, ohne zuvor den kompletten Aufbau neu erfassen zu müssen.

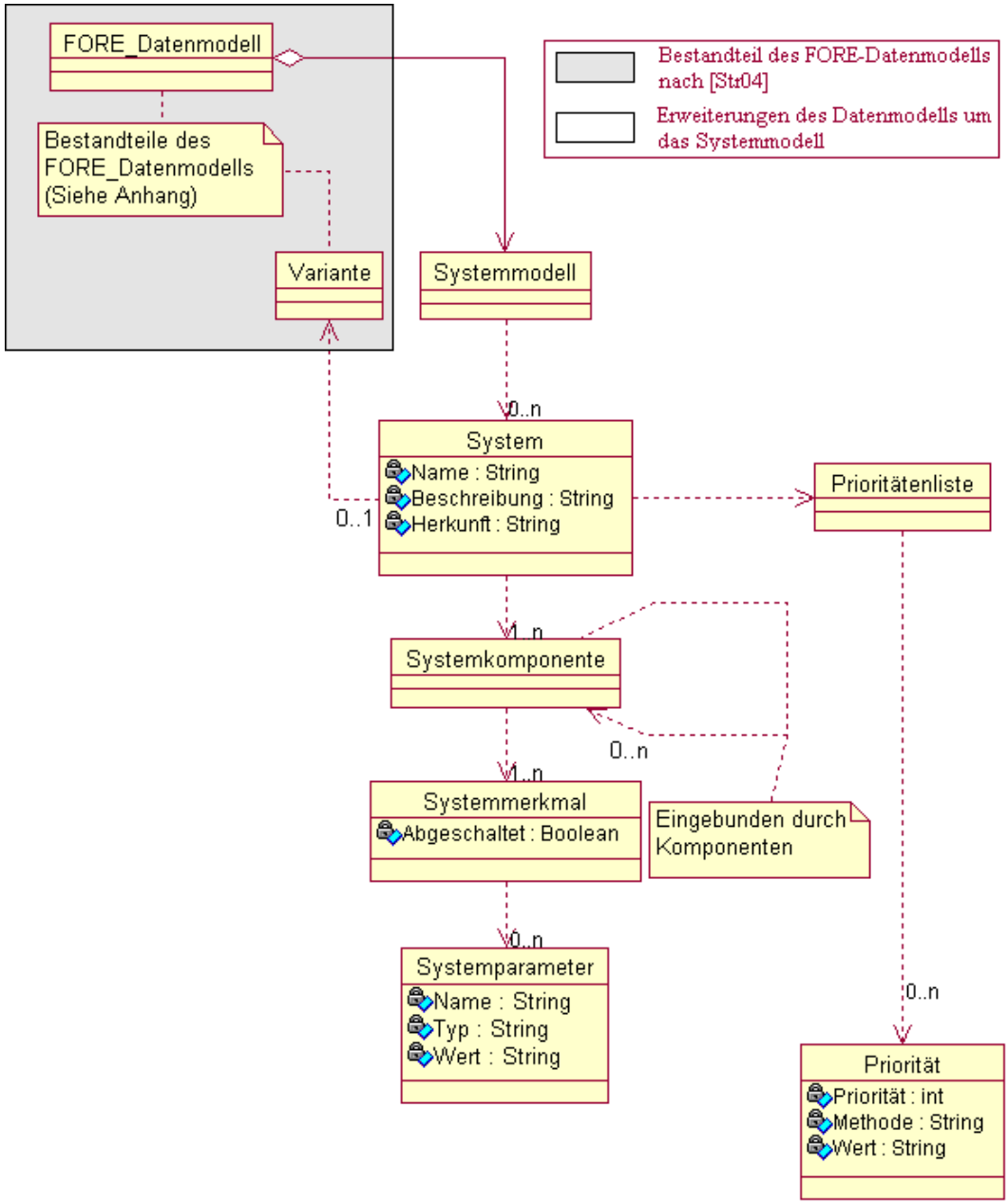

Abb. 4-13: System im erweiterten FORE-Datenmodell

4.6. Änderung einer Applikation

Applikationen werden nicht nur einfach verkauft, sondern unterliegen einer ständigen Aktualisierung und Erweiterung. Viele namhafte Hersteller bieten Möglichkeiten zum Upgrade der vorhandenen Software an. Für den Kunden äußert sich eine Aktualisierung meist in fehlerbereinigten Funktionen und performanteren Verhalten. Neue Funktionen sind entweder kostenfrei oder gegen Aufpreis käuflich zu erwerben. Komponentenbasierte Software bietet durch ihren modularen Aufbau gute Erweiterungsmöglichkeiten und einen einfachen Austausch fehlerhafter Funktionen, ohne die komplette Applikation ersetzen zu müssen.

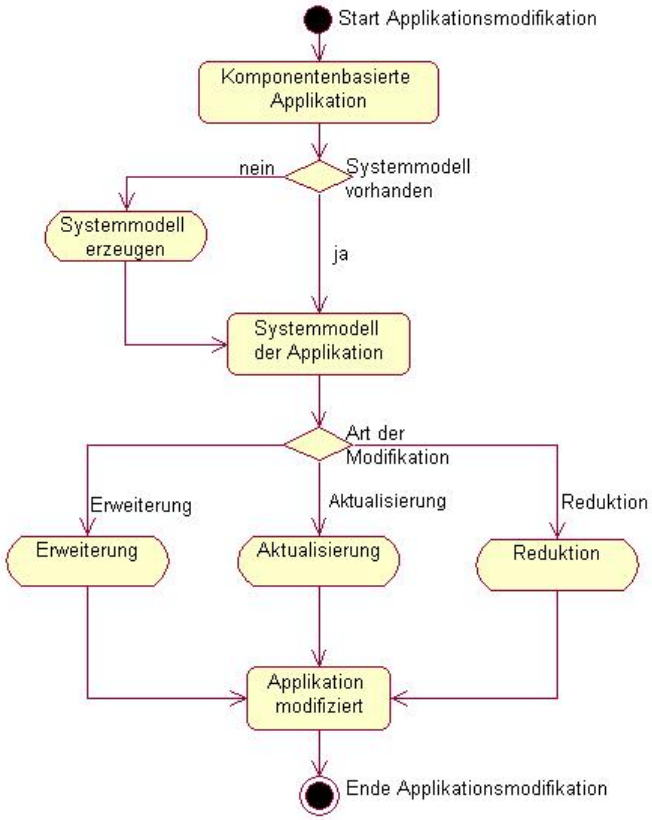

Abb. 4-14: Prozess der Applikationsmodifikation

Bei einer Änderung der bestehenden Applikation können wie in Abb. 4-14 dargestellt drei Fälle unterschieden werden. Zunächst gibt es die einfache Aktualisierung. Hier werden keine Veränderungen im Funktionsumfang vorgenommen. Eine Aktualisierung dient der Fehlerbehebung, Optimierung der Funktionalität, Verbesserung von Ressourcennutzung oder auch der Bedienbarkeit. Eine solche Änderung erfolgt im allgemeinen durch den Austausch von Komponenten.

Das zweite Szenario besteht in der Erweiterung des Funktionsumfanges einer Anwendung. Im Rahmen der Arbeit mit der Applikation hat der Kunde festgestellt, dass einige zusätzliche Funktionen die Arbeit erleichtern würden oder neue Funktionen benötigt werden. Die Funktionserweiterung erfolgt durch die nachträgliche Installation oder den Austausch von Komponenten.
Das dritte und letzte Szenario ist die Entfernung von Funktionen. Dafür kann es verschiedene Gründe geben. Einerseits können nicht mehr benötigte Funktionen entfernt werden, um Ressourcen zu sparen und um so die Applikation schlanker zu gestalten. Andererseits kann es auch rechtliche Beweggründe geben, z. B. bei Verwendung von Mietsoftware. Eine Reduzierung des Funktionsumfanges kann durch die Deinstallation von Komponenten erfolgen.
Im vorliegenden Ansatz wird jede abgeleitete Applikation in einem Systemmodell gespeichert. Dieses Modell kann der Anwendung beigefügt werden oder auch beim Händler verbleiben. Möchte der Kunde seine Anwendung modifizieren, kann der *Komponentenmonteur* präzise feststellen, welche Bestandteile bereits enthalten sind und welche Anforderungen zur bestehenden Applikation geführt haben. Das Modell kann von einem Werkzeug zur Änderung der Anwendung geladen werden. In Verbindung mit einem dazu gehörigen FORE-Datenmodell kann dieses Programm aus dem vorliegenden Systemmodell die bereits umgesetzten Merkmale analysieren. Es stellt damit den Ausgangpunkt der Modifikation dar. Als Ergebnis der Ableitung wird ein neues Systemmodell erzeugt. Durch einen Vergleich mit dem Vorgängermodell kann ein externes Werkzeug ein Patchprogramm ableiten, mit dem alle Installationen an die neuen Anforderungen angepasst werden können.

4.6.1. Aktualisierung

Für die Aktualisierung der Anwendung stehen dem Kunden zwei Möglichkeiten zur Verfügung. Der triviale Fall besteht darin, alle eingesetzten Komponenten durch die jeweils aktuellste Version zu ersetzen. Hier kann dem Kunden aber keine Garantie gegeben werden, dass seine Anforderungen aus der Komponentenwahl, speziell die Prioritätenliste, wieder Berücksichtigung finden. In den neuen Komponenten können sich beispielsweise Ressourcenaspekte zum Nachteil geändert haben.
Im zweiten Fall kann eine neue Komponentenwahl durchgeführt werden. Dadurch können neu entwickelte Komponenten, die in der ersten Auswahl noch nicht zur Verfügung standen, berücksichtigt werden. Der Vorteil dieses Vorgehens kann auch zum Nachteil

werden. Durch den Austausch von Komponenten mit denen anderer Anbieter können sich Nebeneffekte ergeben, die den Kunden nicht befriedigen. Zum Beispiel verschwinden gewohnte Bedienoberflächen, Arbeitsabläufe müssen neu erlernt werden. Auch die Kosten für solch eine Aktualisierung durch die Anschaffung neuer Komponenten sind zu beachten. Abhilfe kann geschaffen werden, indem bei bestimmten kritischen Funktionen auf einer Beibehaltung der Komponenten bestanden wird. So kann man erlauben, nur neue Versionen der Komponenten einzusetzen und keinen Wechsel zu einem anderen Anbieter zu billigen.

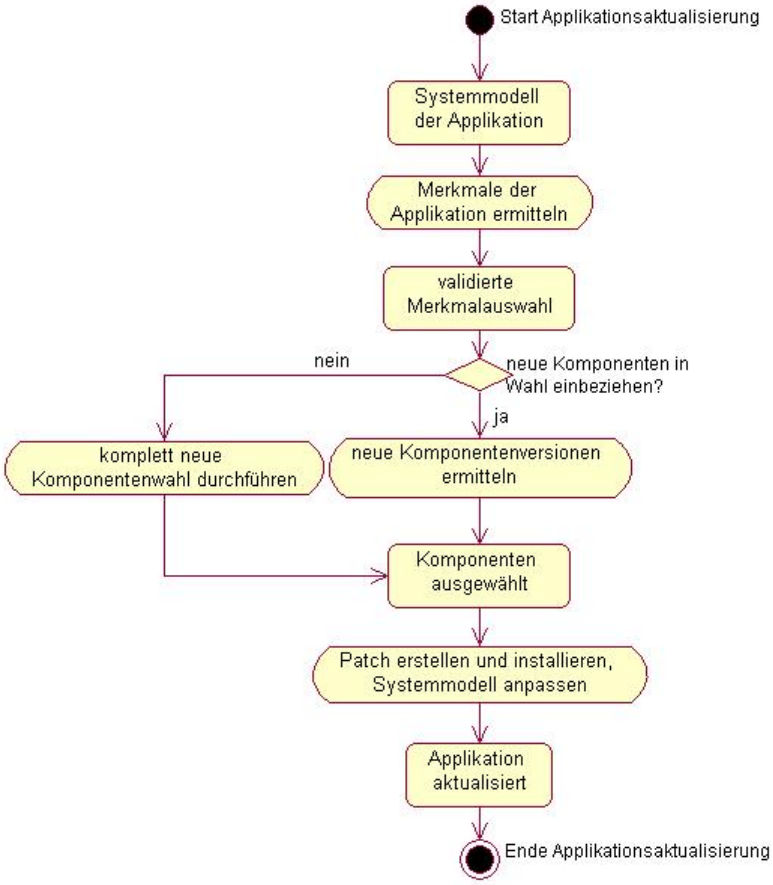

Abb. 4-15: Prozess der Applikationsaktualisierung

4.6.2. Erweiterung

Die Erweiterung der Applikation (Abb. 4-16) betrifft ausschließlich das Hinzufügen neuer Funktionen. Oft wird bei solch einer Erweiterung auch gleich eine Aktualisierung durchgeführt. Die Vorgehensweise bei der Aktualisierung wurde bereits im vorangegangenen Kapitel angesprochen.

Zunächst wird dem Kunden die Merkmalauswahl der bestehenden Anwendung zusammen mit den möglichen zusätzlichen Funktionen präsentiert. Aus dieser Palette kann er neue Funktionen auswählen bzw. die Parameter bestehender Funktionen modifizieren.

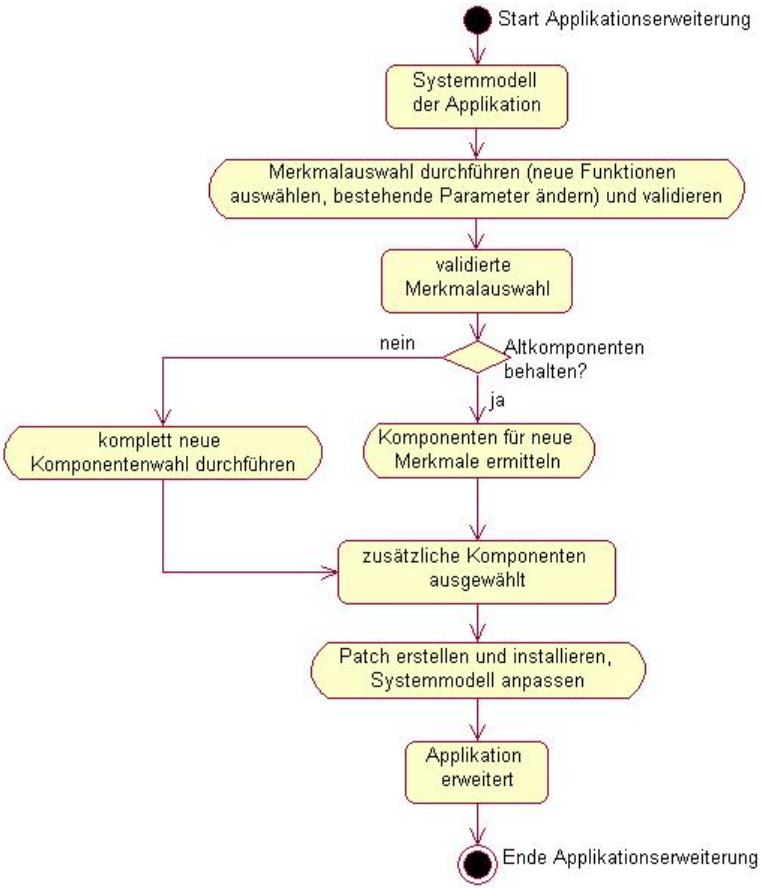

Abb. 4-16: Prozess der Applikationserweiterung

Äquivalent zur Aktualisierung gibt es ebenfalls zwei Verfahren. Der erste Weg behält sämtliche bereits eingesetzte Komponenten für die Funktionen der Altanwendung. Neue Komponenten kommen ausschließlich für neue Funktionen in Frage. Die zusätzlichen Funktionen können auch durch deren Freischaltung in bereits eingesetzten Komponenten zur Applikation hinzugefügt werden. Unter diesen Voraussetzungen kann eine neue Komponentenwahl analog der ersten Ableitung durchgeführt werden. Der einzige Unterschied besteht in der Bindung der bereits vorhandenen Funktionen an die bestehenden Komponenten.

Das zweite Verfahren besteht ebenso wie bei der Aktualisierung in einer neuen Komponentenwahl. Bestimmte Funktionen können auf bereits erworbene Komponenten festgelegt werden, so dass hierfür keine neue Komponente ausgewählt wird. Dies vermindert die Kosten und den Lernaufwand, der hier durch den Einsatz anderer Komponenten entstehen würde.

4.6.3. Reduktion

Bei der Reduktion (Abb. 4-17) werden Funktionen aus einer bestehenden Anwendung entfernt. Der Kunde erhält eine Übersicht aller in der Applikation enthaltenen Merkmale und kann dediziert einzelne Merkmale entfernen. Dies kann einerseits durch Abschalten der Funktion in einer Komponente oder durch die komplette Entfernung der Komponente geschehen. Auch hier ist wiederum die Durchführung einer neuen Komponentenwahl mit den in den vorherigen Abschnitten genannten Konsequenzen möglich.

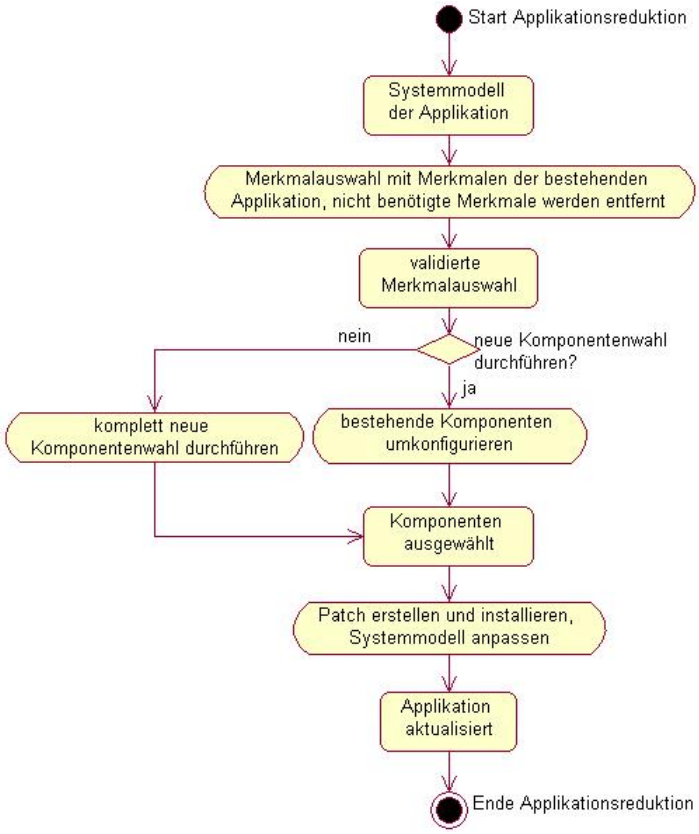

Abb. 4-17: Prozess der Applikationsreduktion

4.7. Zusammenfassung

Zu Beginn des Kapitels wurden die Beziehungen zwischen Komponenten und Merkmale sorgfältig untersucht. Dabei hat sich gezeigt, dass man mit für den Anwender verständlichen Merkmalen Komponenten präzise genug beschreiben kann, um eine Ableitung vornehmen zu können. Für die Ableitung sind die funktionalen Merkmale einer Komponente von entscheidender Bedeutung. Sie repräsentieren den wichtigsten Teil der Anforderungen des Kunden an die zukünftige Applikation, den Funktionsumfang. Parametermerkmale dienen der Anpassung von funktionalen Merkmalen an die Anforderungen einer Applikation. Sie haben keine direkte Beziehung zur Komponente, sondern

beeinflussen indirekt über die funktionalen Merkmale die Komponente. Als neue Merkmalkategorie wurden die Beschaffenheitsmerkmale eingeführt. Sie beschreiben die unveränderlichen Eigenschaften von Komponenten und Merkmalen; damit ermöglichen sie z. B. die Berücksichtigung von qualitativen Anforderungen an eine Anwendung. Welche Komponenten im Verbund zusammenarbeiten können, bestimmen die Schnittstellenmerkmale. Strukturmerkmale besitzen für die Ableitung keine Bedeutung.

Im nächsten Unterkapitel wurde festgestellt, dass die Komponenten die im Merkmalmodell definierten Variabilitäten rückhaltlos unterstützen müssen. Merkmale, die im Modell optional hinterlegt sind, müssen entweder durch Weglassen einer Komponente entfernbar oder innerhalb einer Komponente abschaltbar sein. Komponenten und ihre Merkmale können durch Beschaffenheitsmerkmale genauer beschrieben werden. So kann die Komponente ausgewählt werden, die mit den Ansprüchen des Kunden am ehesten übereinstimmt.

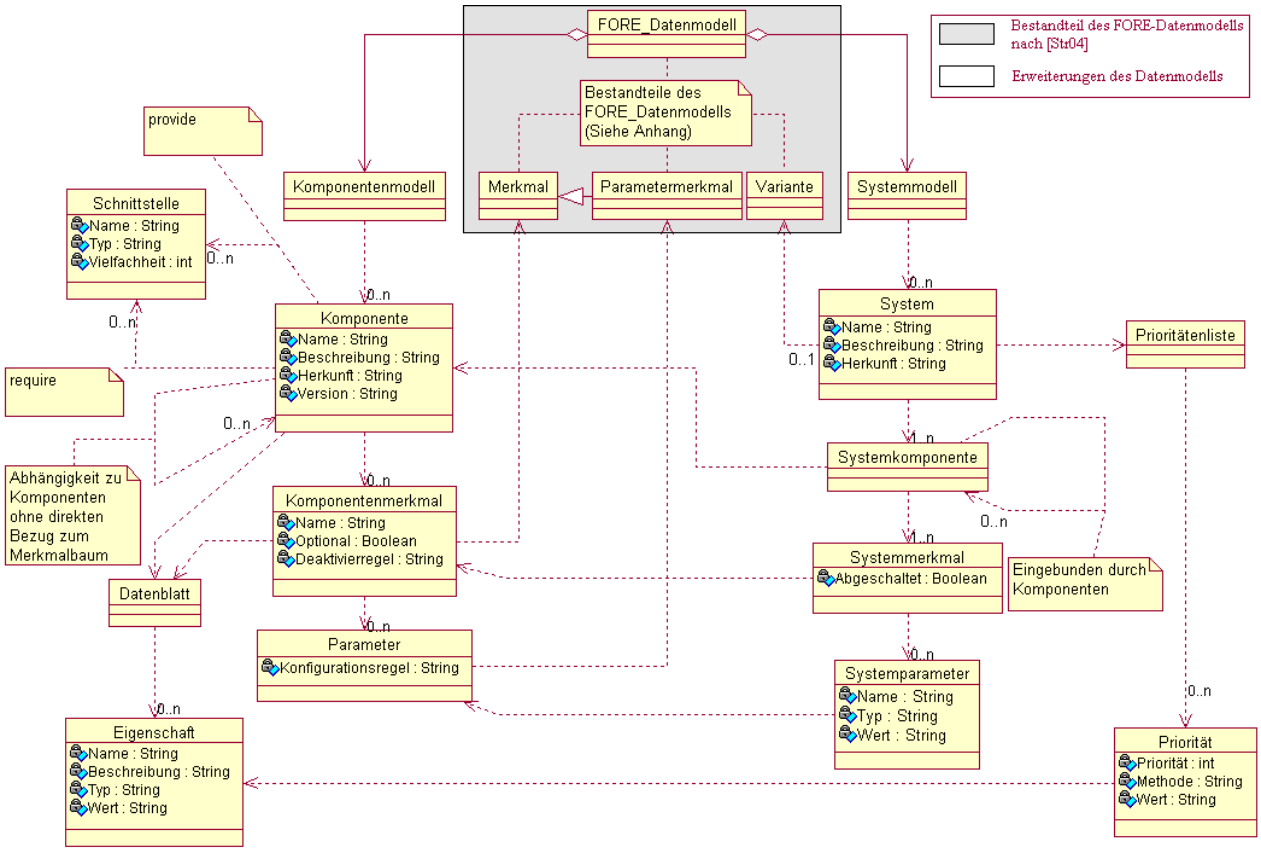

Abb. 4-18: Zusätze zum FORE-Datenmodell – Komponenten- und Systemmodell

Daraus ging die Formulierung eines Komponentenmodells hervor. Es dient der Erfassung aller Komponenten, die in der Systemfamilie eingesetzt werden können. Jeder Komponente werden die implementierten Merkmale mit ihren Parametern zugeordnet

und deren Abhängigkeiten zu anderen Komponenten erfasst. Weiterhin werden Datenblätter mit den Eigenschaften der Komponente und deren Merkmalen gespeichert.
Der eigentliche Kernpunkt des Kapitels war die Komponentenwahl. In einer Vorselektion werden alle Komponenten ausgewählt, welche die geforderten Merkmale implementieren. Anschließend werden mit Hilfe einer Entscheidungsmethode daraus die den Anforderungen am ehesten entsprechenden Komponenten ausgewählt. Als Verfahren wurde die Verwendung einer Prioritätenliste vorgeschlagen. Im Gegensatz zur ebenfalls vorgestellten MAUT-Methode zeichnet sich die Prioritätenliste durch einfache Anforderungserfassung und schnelle Verständlichkeit für den Kunden aus. Anhand eines Beispiels wurde die Vorgehensweise untermauert.
Das Resultat der Komponentenwahl wird in einem Systemmodell gespeichert. Aus dem Systemmodell mit dem dazugehörigen Komponentenmodell (Abb. 4-18) lässt sich durch ein Werkzeug eine Applikation ableiten. Ebenso dient es der Archivierung für spätere Änderungen an der Anwendung. Der Gesamtprozess der Ableitung ist in Abb. 4-19 dargestellt.
Im letzten Unterkapitel wurde die Vorgehensweise zur Modifikation einer bestehenden Applikation angeschnitten. Dabei wurden Szenarien für die Aktualisierung, die Erweiterung oder die Reduktion des Funktionsumfanges einer Applikation näher erläutert. Durch ein vorhandenes Systemmodell wird eine Änderung der Anwendung erheblich erleichtert. Es besteht jeweils die Möglichkeit einzelne Komponenten beizubehalten oder eine neue Komponentenwahl durchzuführen.

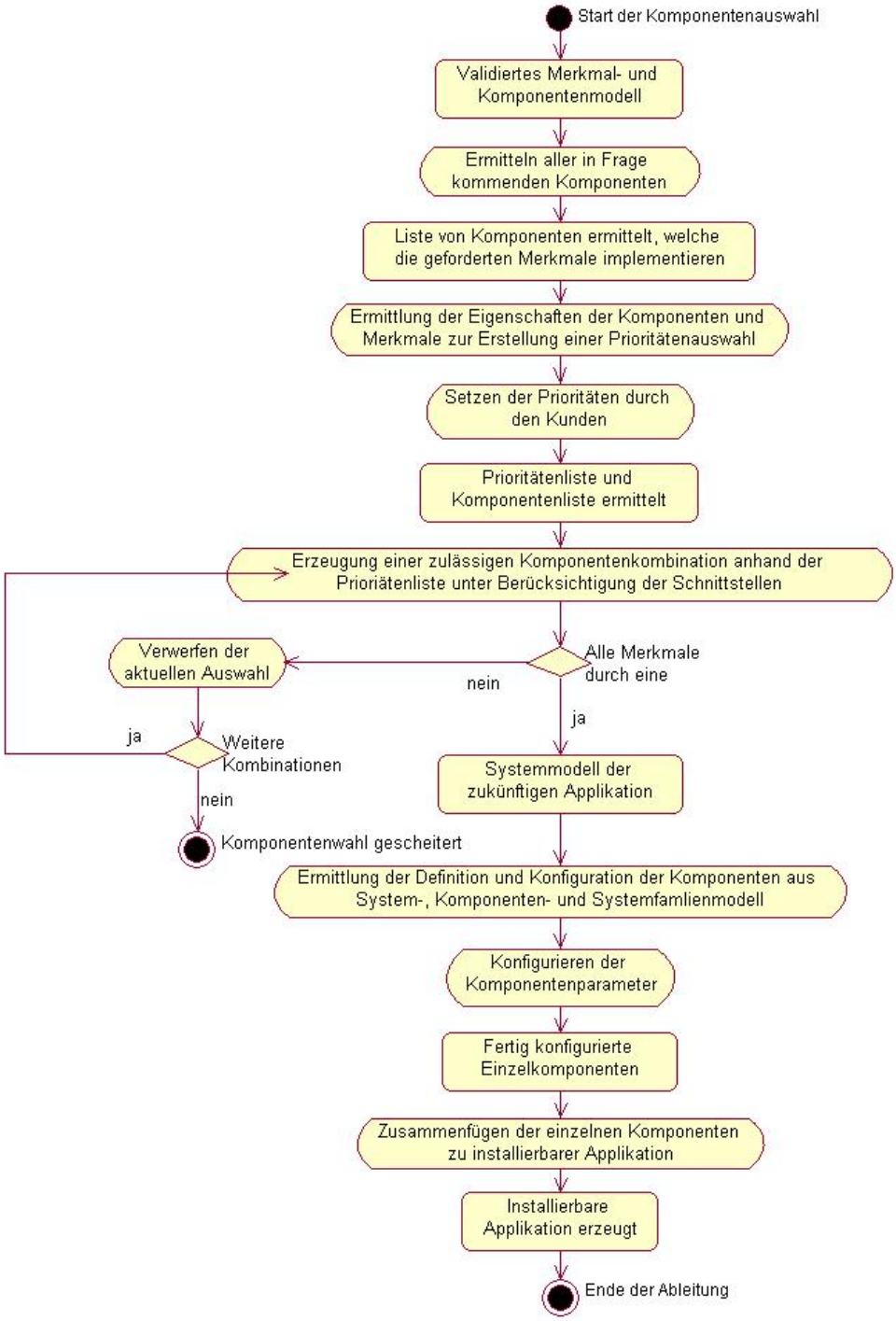

Abb. 4-19: Gesamtprozess der Ableitung

5. Beispielimplementierung

Im Rahmen der Arbeit wird ein Programm als Prototyp entwickelt. Dieser wird unter Verwendung des OpenSource-Programms ANT ([Ant04]) realisiert. Zunächst besteht die Aufgabe des Prototyps darin, eine Zuordnung von durch den Nutzer selektierten Merkmalen zu verfügbaren Komponenten zu treffen. Im zweiten Schritt werden die ausgewählten Komponenten gesammelt, konfiguriert und abschließend zu einer installierbaren Applikation zusammengesetzt. Basis der Ableitung wird neben dem Systemmodell auch das Komponentenmodell mit Detailinformationen zu den einsetzbaren Komponenten sein.

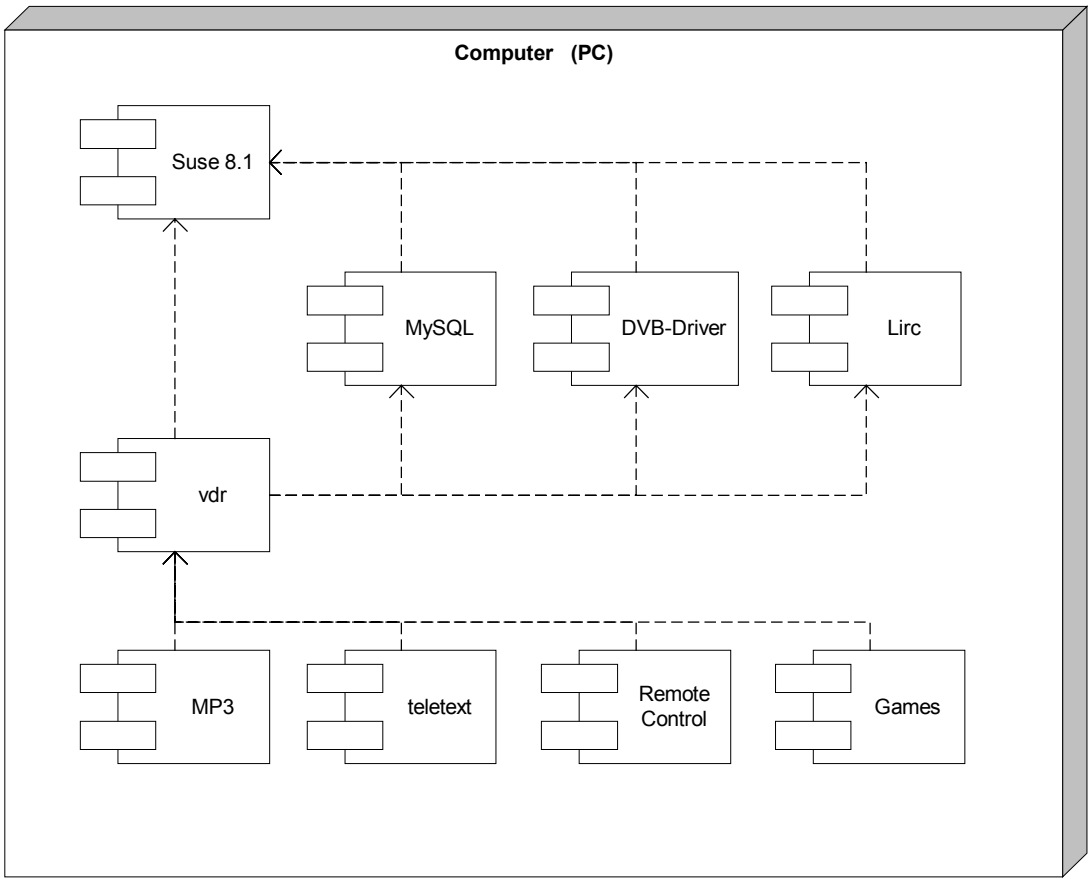

Abb. 5-1: Beispielarchitektur eines DVP-Systems

Auf diese Weise soll es Personen, die nicht direkt mit dem Entwicklungsprozess der Applikation verbunden sind (z. B. ein Verkäufer oder Kunde), ermöglicht werden, aus bestehenden Komponenten mit minimalem Aufwand eine den Anforderungen entsprechende Applikation zu erzeugen.

5.1. Systemvoraussetzungen

Die durch den Systemkern erforderte Komponente „Linux-Rechner“ wird vorausgesetzt. Es ist ein handelsüblicher vdr-fähiger PC (DVB-Karte, Soundkarte, Festplatte) mit einer SuSe 8.1 verfügbar. Im Rahmen der Ableitung werden somit nur die Softwarekomponenten berücksichtigt.

Als Kern des Systems wird das Release 1.3.11 des vdr Projektes verwendet. Durch eine Änderung der Plug-In-Schnittstelle von Release 1.2.6 zu 1.3.11 sind eine Reihe von Komponenten nicht für beide Releases verfügbar. Das hat zur Folge, das nicht alle Merkmale der Produktfamilie von [Str04] zur Auswahl stehen. Der Merkmalbaum wurde auf eine Auswahl der Fähigkeiten der für das aktuellere Release 1.3.11 zur Verfügung stehenden Komponenten beschränkt. Es wurden dabei alle Merkmale des Kerns unberücksichtigt gelassen, da diese für die Ableitung der Beispielapplikation nicht von Bedeutung sind.

5.2. Datenmodell

Das Datenmodell wird durch eine XML-Datei dargestellt, dessen Struktur durch ein XSD-Schema festgelegt wurde. Dazu wurde das in [Str04] entwickelte Schema um das Komponentenmodell (Abb. 5-3) und das Systemmodell (Abb. 5-2) erweitert.

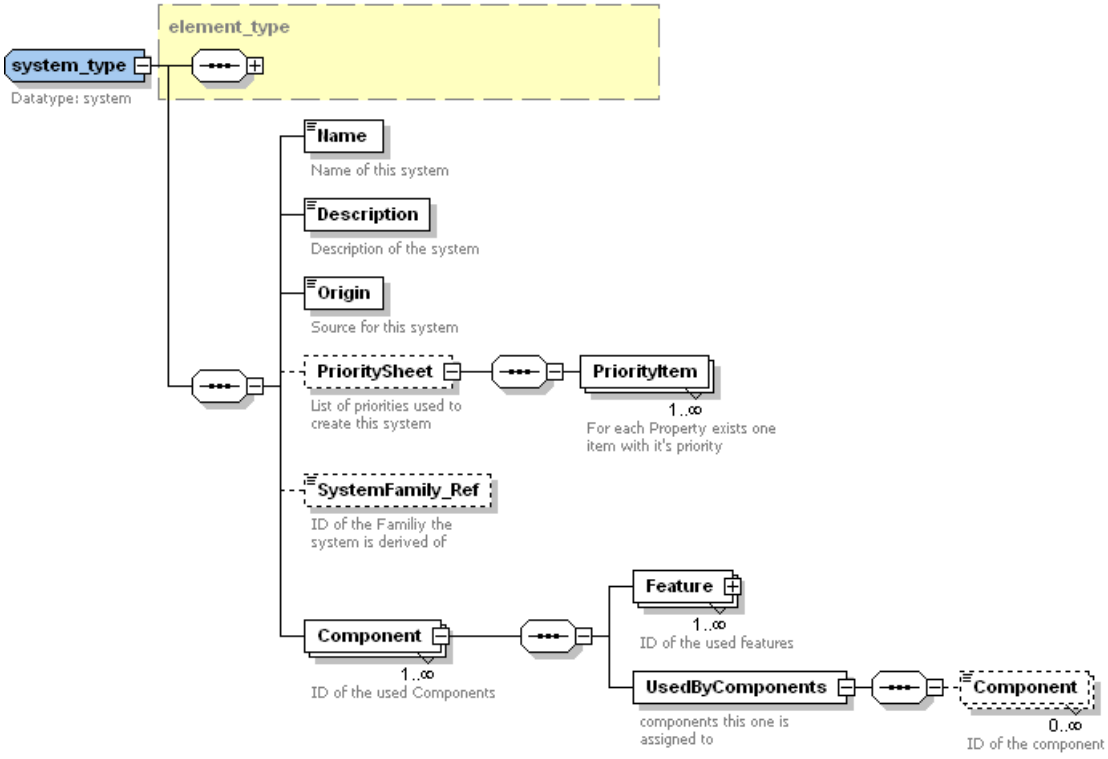

Abb. 5-2: Systemmodell (XML-Sicht)

Die Systemfamilie wird durch die in Abb. 7-2 dargestellten Merkmale repräsentiert. Dieser Merkmalbaum ist im XML-Modell hinterlegt. Im Komponentenmodell sind alle bekannten Komponenten enthalten, welche die vom Merkmalbaum geforderten Merkmale und Strukturen realisieren können. Eine Auswahl der erfassten Komponenten ist in Abb. 7-3 zu sehen.

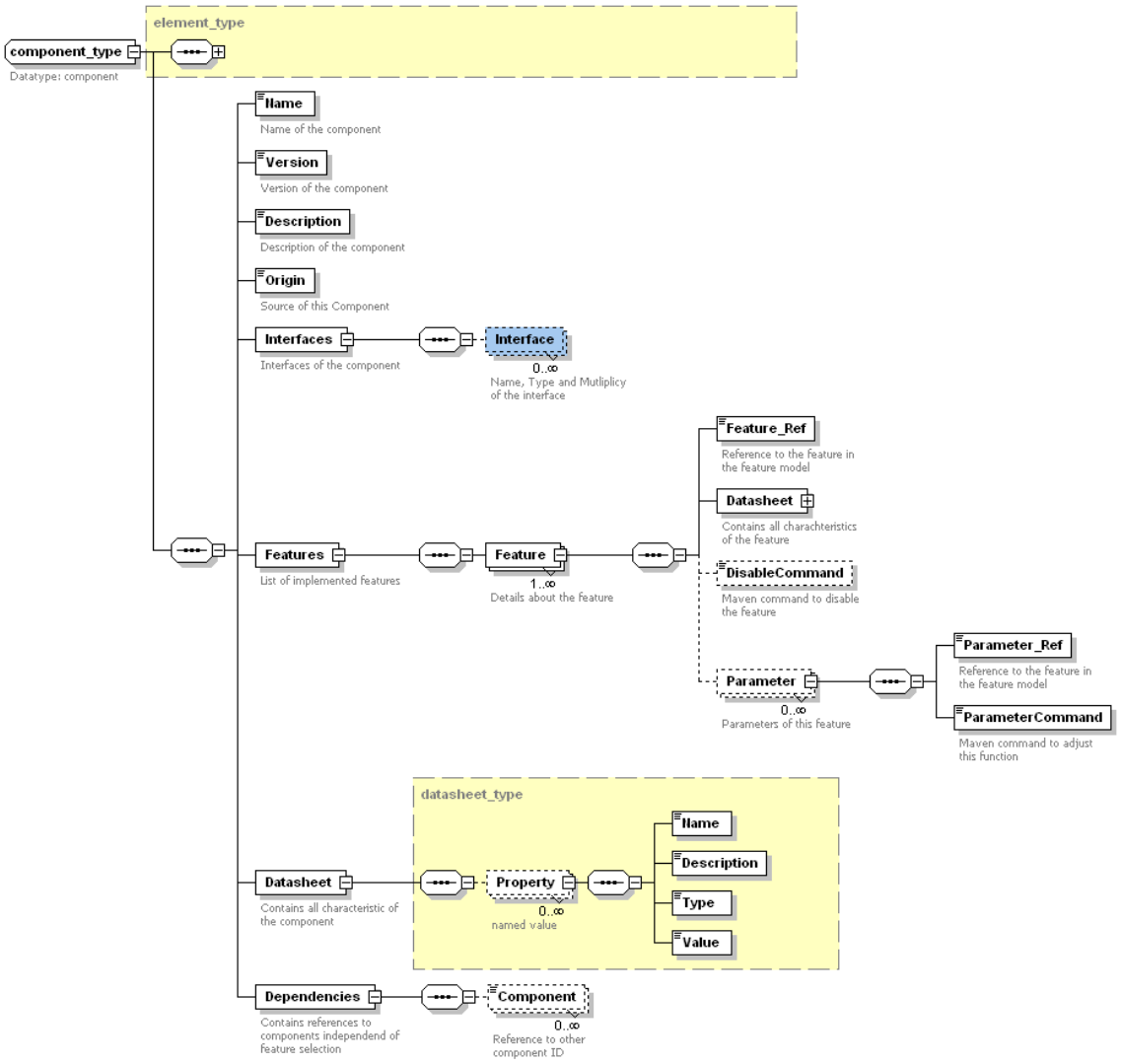

Abb. 5-3: Komponentenmodell (XML-Sicht)

Weiterhin sind bereits einige Mitglieder der Systemfamilie als Varianten enthalten. Sie stellen eine Auswahl dar, die ein Kunde getroffen haben könnte. Eine Variante enthält zum Beispiel alle Merkmale, die eine Nutzung des Internets ermöglichen. Eine andere erweitert die Standardfunktionen des *vdr*-Kerns um eine Vielzahl von Spielen.

Der Bereich der fertigen Systemmodelle im XML-Modell ist anfänglich leer. Hier werden die ermittelten Konfigurationen gespeichert.

5.3. Komponentenwahl

Für die Auswahl der Komponenten wurde ein Java-Programm entwickelt. Im ersten Schritt präsentiert es eine Auswahl der verfügbaren Varianten im Datenmodell. Hier muss sich für eine Variante entschieden werden.

Zur Kontrolle werden alle Merkmale der Variante aufgelistet. Entspricht diese Auflistung den Vorstellungen des Kunden, werden im nächsten Schritt alle Komponenten ermittelt, welche die Merkmale implementieren. Diese Liste stellt eine Vorauswahl der benötigten Komponenten dar.

Die Kernkomponente bildet den Ausgangspunkt der nun folgenden ersten Schnittstellenüberprüfung. Sie dient der Bestimmung der Komponenten, die einerseits die geforderten Merkmale besitzen und andererseits auch vom System genutzt werden können. Von der Kernkomponente ausgehend wird zunächst untersucht, ob alle Komponenten der Vorauswahl über ihre Schnittstellen angesprochen werden können. Falls eine Komponente eine Schnittstelle benötigt, die nicht von der Anwendung zur Verfügung gestellt werden kann, muss entweder eine passende Adapterkomponente gefunden oder sie aus der Vorauswahl entfernt werden. Den Abschluss des Schnittstellentests bildet ein erneuter Überdeckungstest. Sind noch alle Merkmale durch Komponenten repräsentiert oder gibt es eine Komponente, die ein Merkmal implementierte aber nicht durch den Kern angesprochen werden kann? Sind nach einem Schnittstellentest nicht mehr alle Merkmale Komponenten zugeordnet, ist die Ableitung fehlgeschlagen.

Im nächsten Schritt wird nach Überschneidungen gesucht. Eine solche liegt vor, wenn mehrere Komponenten das gleiche Merkmal implementieren können. Diese Konflikte gilt es jetzt zu lösen.

Eine Prioritätenliste soll helfen, die Konflikte zu lösen und die am besten passenden Komponenten zu ermitteln. Bevor eine solche Liste erstellt werden kann, muss herausgefunden werden, welche Prioritäten gesetzt werden können. Hierfür werden die Beschaffenheitsmerkmale der funktionalen Merkmale und der daran beteiligten Komponenten erfasst. Jedes Beschaffenheitsmerkmal kann einen Punkt der Prioritätenliste repräsentieren. Diese muss jedoch zunächst vom Kunden entsprechend seinen Anforderungen aufgebaut werden. Dazu werden ihm alle Beschaffenheitsmerkmale aufgelistet. Er muss nun entscheiden, welches dieser Merkmale für ihn am wichtigsten ist und wie es ausgeprägt sein soll. Hier besteht die Wahl, ob der Wert des Merkmals minimal oder maximal sein soll bzw. ob er einem vom Kunden vorgegebenen Wert gleicht oder dazu

verschieden ist. Setzt der Kunde keine Prioritäten, wird eine zufällige Komponente für jeden Konflikt ausgewählt.

Anhand der jetzt gegebenen Prioritäten werden für die Konfliktpunkte die entsprechenden Komponenten ausgewählt. Zunächst wird ein Überdeckungstest durchgeführt, der kontrolliert, ob die vorliegende Kombination alle Merkmale bereitstellt. Es kann vorkommen, dass Komponenten auch von Merkmalen entfernt werden müssen, an denen kein Konflikt existierte. Das ist insbesondere bei Merkmalen der Fall, die in einer Komponente obligatorisch implementiert sind. In Abb. 5-4 ist solch ein Problemfall dargestellt. Für FM2 existiert ein Konflikt, sowohl „Komponente 1“ als auch „Komponente 2“ bieten das Merkmal FM2 an. Für welche Komponente man sich auch entscheidet, es wird immer ein Merkmal fehlen.

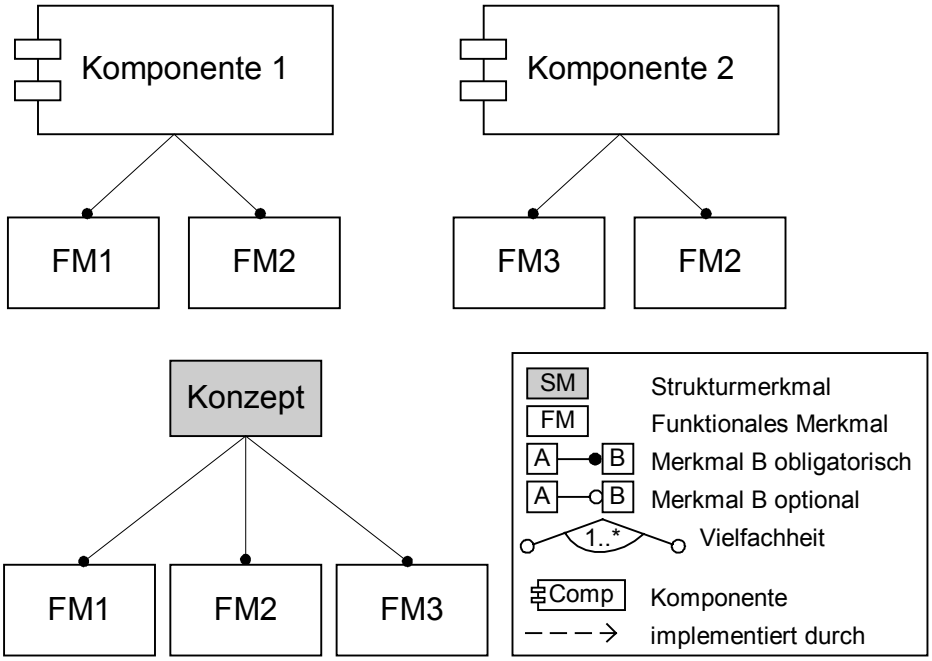

Abb. 5-4: Problemfall

Weiterhin muss überprüft werden, ob alle Komponenten vom Komponentenkern aus erreichbar sind, da eine im ersten Schnittstellentest enthaltene Komponente jetzt nicht mehr verfügbar sein kann. Dazu wird ein erneuter Schnittstellentest mit der bestimmten Komponentenkombination durchgeführt.

Erfüllt eine Kombination nicht alle Anforderungen, wird sie verworfen. Die Konfliktpunkte werden erneut mit der Prioritätenliste aufgelöst. Im Unterschied zum ersten Durchlauf wird jetzt die nächst beste Kombination ermittelt und getestet.

Führt keine Kombination zum geforderten Ergebnis, ist die Ableitung als gescheitert anzusehen. Besteht die Auswahl sowohl den Überdeckungs- als auch den Schnittstellen-

test, ist eine Komponentenkombination gefunden worden, welche die Anforderungen an das System erfüllen kann.
Die erfolgreiche Kombination wird mit der ermittelten Konfiguration der Komponenten als Systemmodell in der XML-Datei gespeichert. Damit ist die Komponentenwahl erfolgreich beendet.

5.4. Ableitung

Für die Ableitung wurde ein weiteres Java-Programm entwickelt. Aufgabe dieses Programms ist die Zusammenstellung eines installierbaren Pakets mit allen benötigten Komponenten. Dazu koordiniert es die Ausführung von ANT-Skripten, von denen jeder Komponente jeweils eins zugeordnet ist. Der prinzipielle Ablauf ist in Abb. 5-5 dargestellt.
Nach dem Start wird dem Nutzer eine Liste aller verfügbaren Systeme präsentiert, die im Datenmodell enthalten sind. Mit der Auswahl eines dieser Systeme und der Angabe der Stelle im Filesystem, an welcher das Installationsprogramm abgelegt werden soll, beginnt die Ableitung.

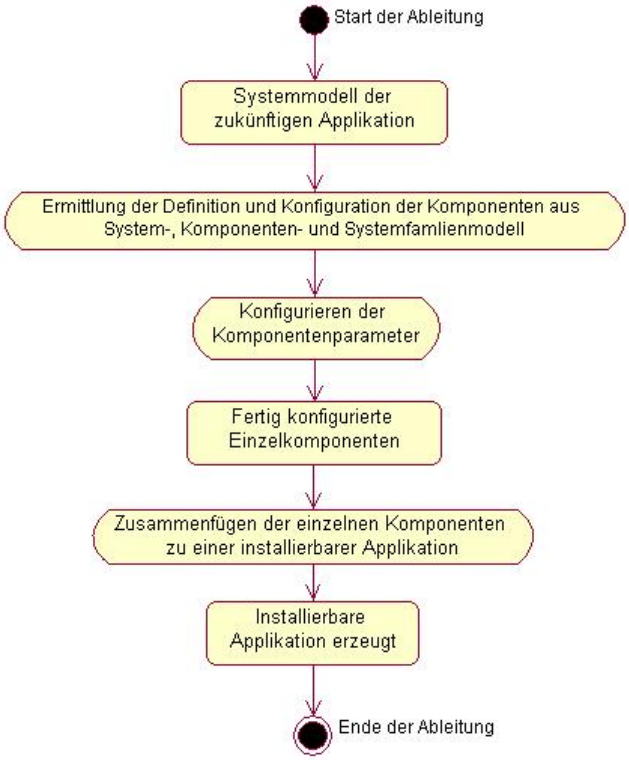

Abb. 5-5: Ableitungsprozess

Zunächst werden alle benötigten Komponenten des Systems ermittelt. Für jede Komponente wird die dazugehörige Definition im Komponentenmodell ermittelt. Hier sind die

Informationen hinterlegt, die benötigt werden, um die Komponente zu konfigurieren. Die wichtigste Information ist hier die Position des für diese Komponente verantwortlichen ANT-Skriptes. Es enthält alle notwendigen Prozesse, um eine Komponente zu konfigurieren und zusammenzupacken. Diese Prozesse sind in sogenannten Targets organisiert. Der Aufruf des Targets „disableSnake“ sorgt beispielsweise beim dortigen ANT-Skript der Komponente „Games“ dafür, dass im Makefile der Komponente das Spiel „Snake“ ausgetragen wird.
In der Komponentendefinition sind die Namen der Targets hinterlegt, mit denen Parameter verändert bzw. Merkmale abgeschaltet werden können. Zunächst werden für alle abgeschalteten Merkmale einer Komponente die entsprechenden Targets aufgerufen. Der Aufruf des externen ANT-Kommandos für das oben kurz erwähnte Beispiel sieht dann so aus:

ant –f vdr/games-0.6.1/project.xml disableSnake

Bei den Parametern wird ähnlich verfahren. Der im Systemmodell hinterlegte Wert des Parameters wird als zusätzlicher Parameter an das Skript übergeben:

ant –f vdf/clock-0.0.4/project.xml setDisplayKey –Dparam=”User1”

Nachdem alle im Systemmodell enthaltenen Komponenten entsprechend den Vorgaben konfiguriert wurden, wird mit der Erstellung des installierbaren Pakets begonnen. Zuerst wird für den Systemkern das *make*-Target aus dessen ANT-Skript aufgerufen. Dann erfolgt der gleiche Aufruf für alle Komponenten, die direkt vom Kern eingebunden werden. Dieses Vorgehen wird für jede weitere Schicht von Komponenten fortgesetzt, bis alle Komponenten des zukünftigen Systems erfasst wurden. Das *make*-Target sorgt dafür, dass alle benötigten Teile der Komponente in ein temporäres Verzeichnis transferiert werden. Sind alle Komponenten dort hinterlegt, wird mittels eines Packprogramms ein Archiv erstellt, das installiert werden kann. Mit diesem Schritt ist die Ableitung vollständig.

6. Bewertung und Ausblick

In diesem Kapitel wird sowohl eine Bewertung des Ansatzes als auch ein Ausblick auf weitere Forschungsansätze gegeben. Da ohne Retrospektive eine Perspektive nur unvollständig möglich ist, wird zunächst betrachtet, in welchem Umfang die Ziele der Arbeit erreicht wurden. Daran anschließend wird ein Ausblick gewährt, wie der Ansatz weiter vorfolgt und verbessert werden kann.

6.1. Bewertung des Ansatzes

Das Ziel der Arbeit besteht darin, einen Ansatz zu entwickeln, mit dem auf Basis einer Merkmalauswahl eine komponentenbasierte Applikation aus vorhandenen Komponenten abgeleitet werden kann. Damit verbunden waren folgende Fragen, die im Rahmen der Arbeit untersucht worden:

- Inwieweit ist es möglich, das Merkmalmodell mit Hilfe eines zu definierenden Komponentenmodells in eine Applikation zu überführen?
- Welche zusätzlichen Informationen werden von den Komponenten benötigt?
- Nach welchen Aspekten werden die Komponenten ausgewählt und konfiguriert?

In der Frage, inwieweit eine Applikation aus einem Merkmalmodell überführt werden kann, wurde festgestellt, dass eine Reihe von Randbedingungen zu beachten sind. So stellt die Struktur des Merkmalbaumes Anforderungen sowohl an die Architektur der Applikation als auch an die einzelnen einzusetzenden Komponenten. Es reicht nicht, dass eine Komponente die Merkmale des Baumes implementiert, sondern sie muss auch die Beziehungen zwischen den Merkmalen des Baumes unterstützen. Im Idealfall wird das Merkmalmodell der Produktlinie entwickelt, bevor das Komponentenmodell bzw. die Komponenten selbst erstellt werden. Sind die Komponenten bereits vorhanden, muss sichergestellt werden, dass sie den Merkmalbaum der Produktlinie wiedergeben können. Es wurde gezeigt, welche Bedingungen erfüllt sein müssen, um eine komponentenbasierte Applikation erfolgreich aus einem Merkmalmodell ableiten zu können.
Für eine erfolgreiche Ableitung werden eine Reihe von Zusatzinformationen über die Komponenten benötigt. Die wichtigsten Informationen sind die Beziehungen zwischen einer Komponente und den Merkmalbaum. Für jede Komponente wird erfasst, welche Merkmale wie in der Komponente implementiert werden. Für jede Komponente müssen Regeln hinterlegt sein, mit der ihre Konfiguration entsprechend den Erfordernissen an-

gepasst werden kann. Jeder Komponente werden Schnittstellen zugeordnet, über die sie in die Applikation eingebunden werden können. Das entwickelte Komponentenmodell unterscheidet sich vom UML-Komponentenmodell vor allem durch die Einbindung von Merkmalen. Das Komponentenmodell genügt den Ansprüchen einer automatisierten Ableitung,

Für die Auswahl der Komponenten wurde ein Prozess vorgestellt, der eine Überführung einer Merkmalauswahl in eine Applikation ermöglicht. Zunächst erfolgt eine Zuordnung der Merkmale zu den Komponenten. Bei Überlappungen wurde mittels einer Prioritätenliste entschieden, welche Komponente besser den Anforderungen des Kunden entspricht. Dazu wurden die Beschaffenheitsmerkmale eingeführt, die sowohl Komponenteneigenschaften als auch Eigenschaften von funktionalen Merkmalen zur Differenzierung von Komponenten mit überschneidenden funktionalen Merkmalen bereitstellen. Zur Konfiguration der Merkmale und deren Parameter werden Regeln verwendet, die durch das ableitende Programm interpretiert werden können. Er wurde gezeigt, nach welchen Aspekten Komponenten für die Applikation ausgewählt und konfiguriert werden können.

Durch die Verwendung dieses Absatzes kann eine Applikation für einen Kunden in kurzer Zeit erstellt werden. Durch die bereits vorgefertigten Komponenten wird die Zeitspanne von der Erfassung der Anforderungen bis zur Erstellung der Anwendung verkürzt. Solange alle Anforderungen des Kunden durch diese Komponenten erfüllt werden können, lässt sich die Applikation durch den vorgestellten Ansatz direkt erstellen. Grundlage dazu ist ein ausgereiftes Datenmodell und eine große Anzahl von Komponenten, so dass möglichst viele Anforderungen erfüllen werden können.

Die Kosten für die Erstellung einer komponentenbasierten Applikation sind anfänglich höher als bei einer Individualsoftware. Der zusätzliche Aufwand muss in die Wiederverwendbarkeit investiert werden. Laut [Pou97] kann man diese Zusatzkosten je nach Einsatzgebiet auf 0% bis 120% der Entwicklungskosten für die entsprechende Individualsoftware schätzen ([Pou97, S. 26f]). Eine komponentenbasierte Software wird somit erst dann kostengünstiger, wenn die verwendeten Komponenten vielfach eingesetzt werden.

Das Ziel der Arbeit, die *Ableitung einer komponentenbasierten Applikation aus einem merkmalbasierten Systemfamilienmodell* vorzunehmen, wurde unter Berücksichtigung von Randbedingungen erreicht. Es wurde ein Ableitungsprozess vorgestellt und anhand einer Beispielimplementierung nachgewiesen.

6.2. Ausblick

Abschließend sollen verschiedene Möglichkeiten der Weiterentwicklung präsentiert werden, die im Rahmen dieser Arbeit aufgedeckt wurden.

Der vorliegende Ansatz wurde nur auf die Anwendbarkeit auf Softwarekomponenten überprüft. Hardware-Komponenten wurden nicht mit in das Systemmodell integriert. Es spricht jedoch nichts gegen eine Erfassung im Komponentenmodell. Zur automatisierten Erstellung der Hardwaregrundlage könnte man sich als Ergebnis der Komponentenwahl Bestückungslisten für die Fertigung vorstellen.

Eine Software ist ohne Dokumentation nicht vollständig. Die Ableitung verwendete zur Dokumentation der entstandenen Applikation lediglich die den Komponenten beigelegten Anleitungen. So besteht die Gebrauchsanweisung aus vielen Dokumenten, die auch Merkmale erläutern, die in den Komponenten abgeschaltet wurden. Ein einzelnes Dokument, in dem nur die implementierten Merkmale erläutert sind, fehlt. Auch die Dokumentation muss hier merkmalbasiert erstellt werden.

Die Behandlung der Beschaffenheitsmerkmale sollte weiter verbessert werden. Der Einfluss von Lizenz- oder Preismodellen auf die automatisierte Ableitung muss künftig genauer untersucht werden.

Im Rahmen der Arbeit wurden die Ansätze zur Modifikation einer bestehenden Anwendung nur angeschnitten. Eine weitergehende Untersuchung könnte hier mehr Klarheit bzgl. der Durchführbarkeit bringen.

Die Ableitung von Applikationen aus einem Merkmalmodell ist nicht immer von Erfolg gekrönt. Fehlende Komponenten, die bestimmte Merkmale implementieren sollen, oder Komponentenkombinationen, die nicht zusammen interagieren können, führen zum Fehlschlag einer Ableitung. Um dies zu vermeiden, könnte man eine dynamische Merkmalauswahl entwickeln, welche jeweils die speziellen Umstände der verfügbaren Komponentenbasis berücksichtigt. So könnten dort auch die Beschaffenheitsmerkmale direkt in die Auswahl einfließen.

Ein an die Merkmalbasis angepasstes Komponentendesign erleichtert die automatisierten Ableitung. Durch die frühzeitige Berücksichtigung der Beziehungen zwischen den Merkmalen kann die Anzahl der zu beachtenden Sonderfälle verringert werden.

Eine Merkmalauswahl kann man verkleinern, indem nur die Merkmale zur Wahl angeboten werden, die in den oberen Schichten einer Anwendungsarchitektur lokalisiert sind. Beispielsweise erübrigt sich die Auswahl der DVB-Karte bzw. des dazugehörigen

Treibers für ein vdr-System, wenn bereits das Merkmal „Digitaler Fernsehempfang“ gewählt ist.

Wenn Entscheidungen über die Anforderungen des Systems getroffen werden, wählt man nie alle Merkmale aus, sondern setzt bereits ein grundsätzliches Merkmalset voraus. Wenn beispielsweise die Merkmale eines Hauses beschrieben werden sollen, dann werden nie alle Merkmale genannt. Man verbindet bereits mit dem Begriff Haus eine Reihe von Merkmalen, die nicht ausdrücklich genannt werden müssen, aber als notwendig erachtet werden. Die Merkmale Dach oder Hauswände werden automatisch mit dem Begriff Haus verbunden, nicht jedoch ein Balkon oder ein Gästezimmer. Was ausgewählt wird, sind die Sonderwünsche, die von der allgemeinen Vorstellung abweichen. Hier wäre ein Ansatzpunkt für zukünftige Forschungen zur Optimierung der Merkmalauswahl.

7. Zusammenfassung

Diese Arbeit befasste sich mit der automatisierten Ableitung einer komponentenbasierten Applikation auf der Grundlage einer vom Kunden erstellten Merkmalauswahl. Ausgangspunkt bildete das im FORE-Ansatz entwickelte Datenmodell. Das Ziel bestand darin, aus vorgefertigten Komponenten eine Applikation erstellen zu können, die den Ansprüchen eines Kunden genügt. Durch die Verwendung von Merkmalen zur Anforderungsspezifikation ist es möglich, diese Ableitung mit geringen technischen Hintergrundwissen durchzuführen.

Im Verlauf der Arbeit wurde das FORE-Datenmodell mit einem Komponenten- und einem Systemmodell erweitert, um den Ansprüchen der automatisierten Ableitung zu genügen. Für das Komponentenmodell wurden die Zusammenhänge zwischen Komponenten und Merkmalen eingehend untersucht. Dabei wurden von FORE abweichende Merkmaldefinitionen eingeführt. Hier ist insbesondere die Abänderung der Definition der Schnittstellenmerkmale und die Einführung von Beschaffenheitsmerkmalen zu nennen. Die Komponenten werden im Modell anhand ihrer Merkmale beschrieben. Das Systemmodell stellt die Konfiguration der Komponenten in der zukünftigen Anwendung dar.

Der hier vorgestellte Prozess zur Ableitung gliedert sich grob in zwei Teilprozesse. Im ersten Schritt werden mit Hilfe des Komponentenmodells die Komponenten ermittelt, die in der späteren Anwendung eingesetzt werden sollen. Dazu werden die Komponenten den Merkmalen der Merkmalauswahl zugeordnet und Konflikte mittels einer für den Kunden leicht verständlichen Prioritätenliste gelöst. Im Verlauf dieses Schrittes wird das Systemmodell erzeugt, in dem die Konfiguration der Komponenten für die Ableitung enthalten ist. Der zweite Schritt befasst sich mit der Erstellung der eigentlichen Anwendung. Anhand des Systemmodells werden die Komponenten konfiguriert und der Applikation zugeführt.

Der entwickelte Prototyp demonstriert die Funktionsweise des Ableitungsprozesses. Im Rahmen des *vdr*-Projektes erstellt dieses Java-Programm Anwendungen, die Merkmale besitzen, welche aus den Anforderungen des Kunden ermittelt wurden.

Diese Arbeit liefert eine Methode, mit der komponentenbasierte Anwendungen automatisiert aus einer Merkmalauswahl erstellt werden können.

Literaturverzeichnis

[Ahr04] Ahrensberg, Rolf (2004): *Femon Plug-In* für *vdr*, http://www.saunalahti.fi/~rahrenbe/vdr/femon/

[And03] Andresen, Andreas (2003): *Komponentenbasierte Softwarenentwicklung mit MDA, UML und XML*, Carl Hanser Verlag, München, Wien

[Ant04] Apache.org: *Ant, a Java-based build tool,* http://ant.apache.org/

[Atk02] Atkinson, Colin; Bayer, Joachim; Bunse, Christian; u.a (2002): *Component-based Product Line Engineering with UML,* Addison-Wesley, London, New York, San Francisco, u.a.

[Bar02] Bartelmus, Christoph (2002): *Linux Infrared Project*, http://www.lirc.org/

[Ber98] Bernstein, Larry (1998): *Importance of Software Prototyping* in *Software Tech News*, Vol.2 No. 1, S. 3 & 9, http://www.dacs.dtic.mil/awareness/newsletters/listing.shtml

[Boe88] Boehm, Barry (1988): *A Spiral Model of Software Development and Enhancement*, IEEE Computer, Vol.21, Ausg. 5, Mai 1988, S. 61-72

[Boe98] Boehm, Barry (1998): *Using the WinWin Spiral Model: A Case Study*, IEEE Computer, Vol.31, Ausg. 7, Juli 1998, S. 33-44

[Böl02] Böllert, Kai (2002): *Objektorientierte Entwicklung von Software-Produktlinien zur Serienfertigung von Software-Systemen*, Dissertation, TU-Ilmenau

[Bos00] Jan Bosch (2000): *Design and Use of Software Architectures, Adopting and evolving a product-line approach*, Addison-Wesley, London, New York, San Francisco, u.a

[Com03] Berwolf, Gerald; Kornexl, Ronny (2003): *Commander Plug-In für vdr*, http://www.vdrportal.de/board/portal_downloads.php

[Cza98] Czarnecki, Krzysztof (1998): *Generative Programming*, Dissertation, TU-Ilmenau

[Cza00] Czarnecki, Krzysztof; Eisenecker, Ulrich (2000): *Generative Programming – Methods, Tools and Applications*, Addison-Wesley, Boston, San Francisco, New York, u.a.

[Die03] Dietzel, Ralph (2003): *Konzeption und Entwicklung einer Systemfamilie für eine Universal-Fernbedienung auf Basis eines Palm-Handhelds*, Diplomarbeit, TU-Ilmenau

[eClass00] eClass (2000): *Standard für Materialklassen und Warengruppen - Merkmalleisten-Aufbau*, Institut der deutschen Wirtschaft Köln Consult GmbH, Köln

[Grä00] Grässle, Patrick; Baumann, Henriette; Baumann, Philippe (2000): *UML projektorientiert – Geschäftsprozessmodellierung, IT-System-Spezifikation und Systemintegration mit der UML*, Galileo Press, Bonn

[Gri98] Griss, Martin L.; Favaro, John; d'Alessandro, Massimo (1998): Integrating Feature Modeling with the RSEB, in Proceedings of 5th International Conference on Software Reuse, Victoria, Canada

[Hei01] Heineman, George T.; Councill, William T. (2001): *Component-Based Software Engineering – Putting the Pieces Together*, Addison-Wesley, Boston, San Francisco, New York, u.a.

[ID04] Information & Design (2004): *Paper Prototyping*, http://www.infodesign.com.au/usabilityresources/design/paperprototypinggraphics.asp

[Lat02] Latapie, Simon, et al. (2002): *Video Streaming Project*, http://www.videolan.org/

[Kan90] Kang, Kyo C., et al. (1990): *Feature-Oriented Domain Analysis (FODA) Feasibility Study*, (CMU/SEI-90-TR-21, ADA 235785), Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA

[Kan98] Kang, Kyo C., et al. (1998): *FORM: A Feature-Oriented Reuse Method,* In: *Annals of software engineering : ASE*.Volume 5 - 1998, Baltzer Science Publ., Amsterdam

[Mac04] Macromedia (2004): *Shockwave Player*, http://www.macromedia.com/software/shockwaveplayer/

[Mef03] Meffert, Florian (2003): *Konzeption einer Videodatenverteilung im Rahmen des Digitalen Video Projektes (DVP)*, Diplomarbeit, TU-Ilmenau

[Met02] Metzler, Marcus; Metzler, Ralph (2002): *linuxTV.org-Projekt*, http://linuxtv.org

[Neu01] Neumann, Gustaf (2001): *Wirtschaftsinformatik I*, Lucius & Lucius, Stuttgart

[Orn04] Orndorff, Robert (2004): *File Manager Commanders*, http://www.rmonet.com/commander/index.html

[oV94] o.V. (1994): *Das Große EDV & PC Lexikon*, Isis, Chur (Schweiz)

[oV02] o.V. (2002): *Interoperabilität*, http://www.quality.de/lexikon/interoperabilitaet.htm

[PM04] Projektmagazin (2004): *Prototyping*, http://www.projektmagazin.de/glossar/gl-0099.html

[Pou97] Poulin, Jeffrey S. (1997): *Measuring Software Reuse, Principles, Practices and Economic models*, Addison-Wesley, Reading (Massachusetts), Harlow (England), Menlo Park (California)

[Roy70] Royce, W. W., (1970): *Managing the Development of Large Software Systems,* Proceedings of IEEE WESCON

[Schm02] Schmidinger, Klaus (2002): *vdr Projekt Homepage*, http://www.cadsoft.de/people/kls/vdr

[Sch01] Schryen, Guido (2001): *Komponentenorientierte Softwareentwicklung in Softwareunternehmen*, Deutscher Universitäts-Verlag, Wiesbaden

[Sei00] Software Engineering Institute (2000): *Domain Engineering: A Model-Based Approach*, http://www.sei.cmu.edu/domain-engineering/domain_engineering.html

[Str04] Streitferdt, Detlef (2004): *Family-Oriented Requirements Engineering*, Dissertation, TU-Ilmenau

[Szy02] Szyperski, Clemens; Gruntz, Dominik; Murer, Stephan (2002): *Components Software – Beyond Object Oriented Programming*, 2. Auflage, ACM Press, New York

[Wal02] Wallnau, Kurt C.; Hissam, Scott A.; Seacord, Robert C. (2002): *Building Systems from Commercial Components*, Addison-Wesley, Boston, San Francisco, New York, u.a.

[Wei99] Weiss, David M.; Lai, Chi Tau Robert (1999): *Software Product-Line Engineering: A Family-Based Software Development Process*, Addison-Wesley, Boston, San Francisco, New York, u.a.

[XP99] Wells, Donvan (1999): *Extreme Programming*, http://www.extremeprogramming.org/

Anhang A - FORE-Datenmodell

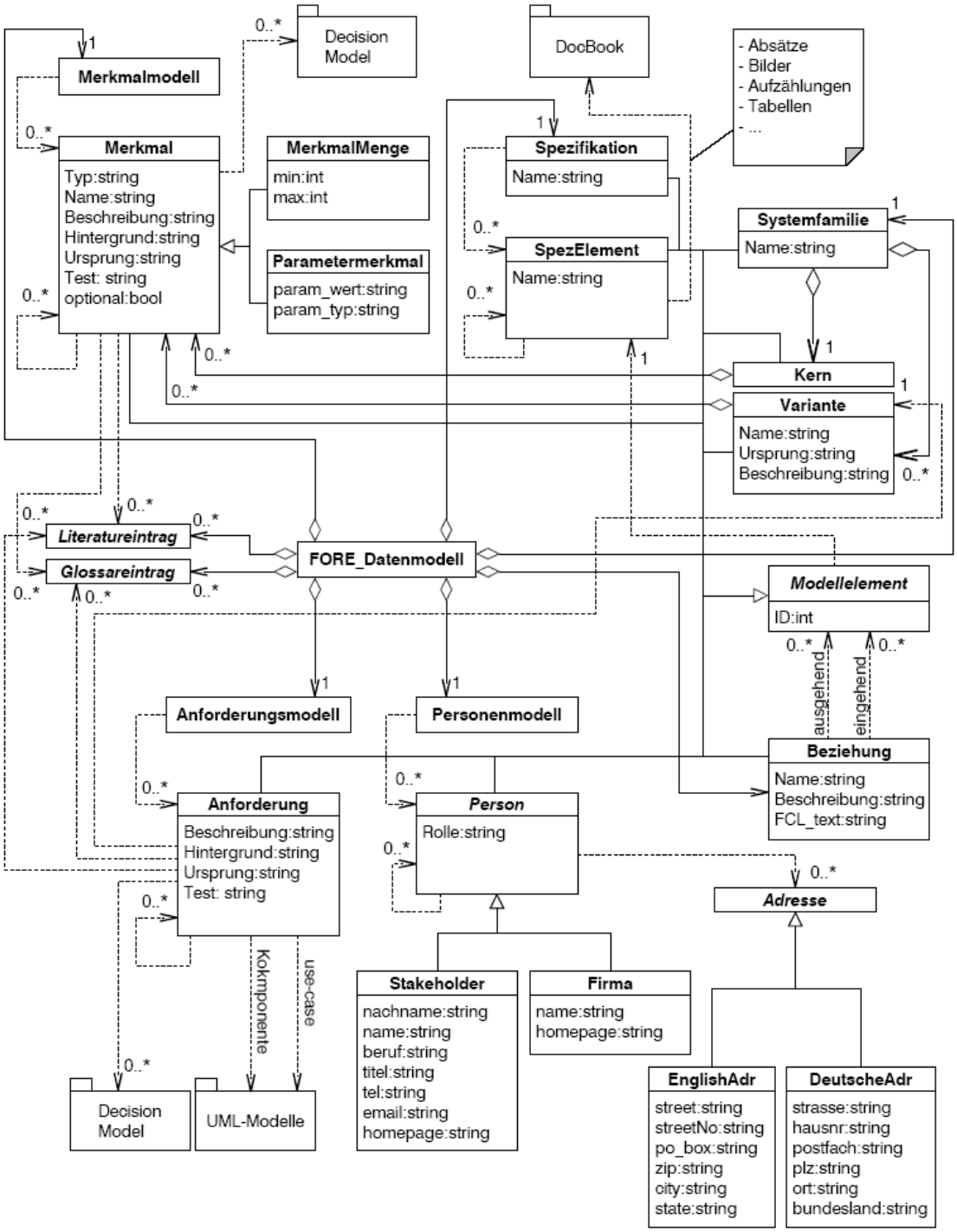

Abb. 7-1: Gesamtes FORE-Datenmodell ([Str04], S. 156)

Anhang B – Beispielimplementierung

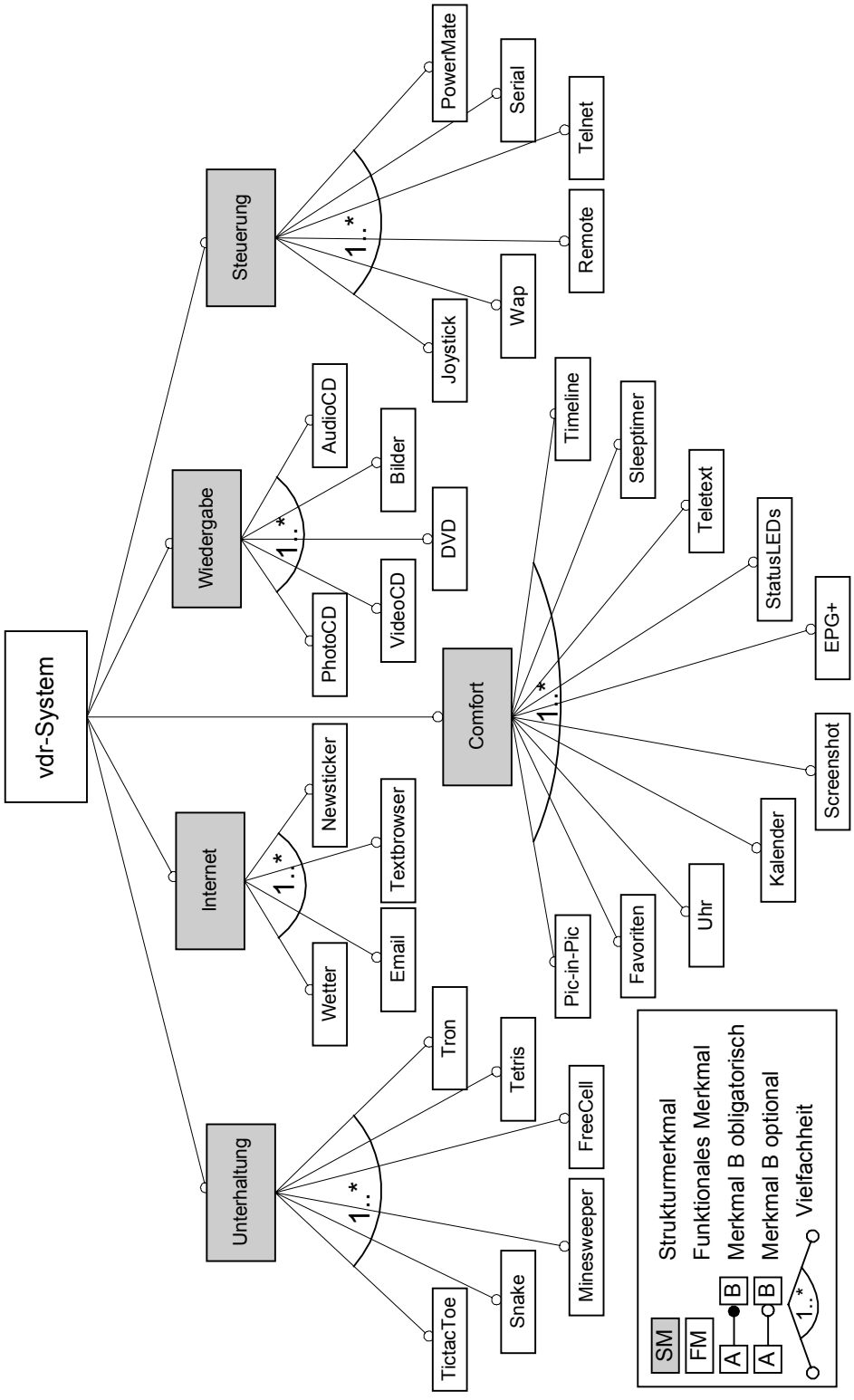

Abb. 7-2: Merkmalbaum der Beispielimplementierung

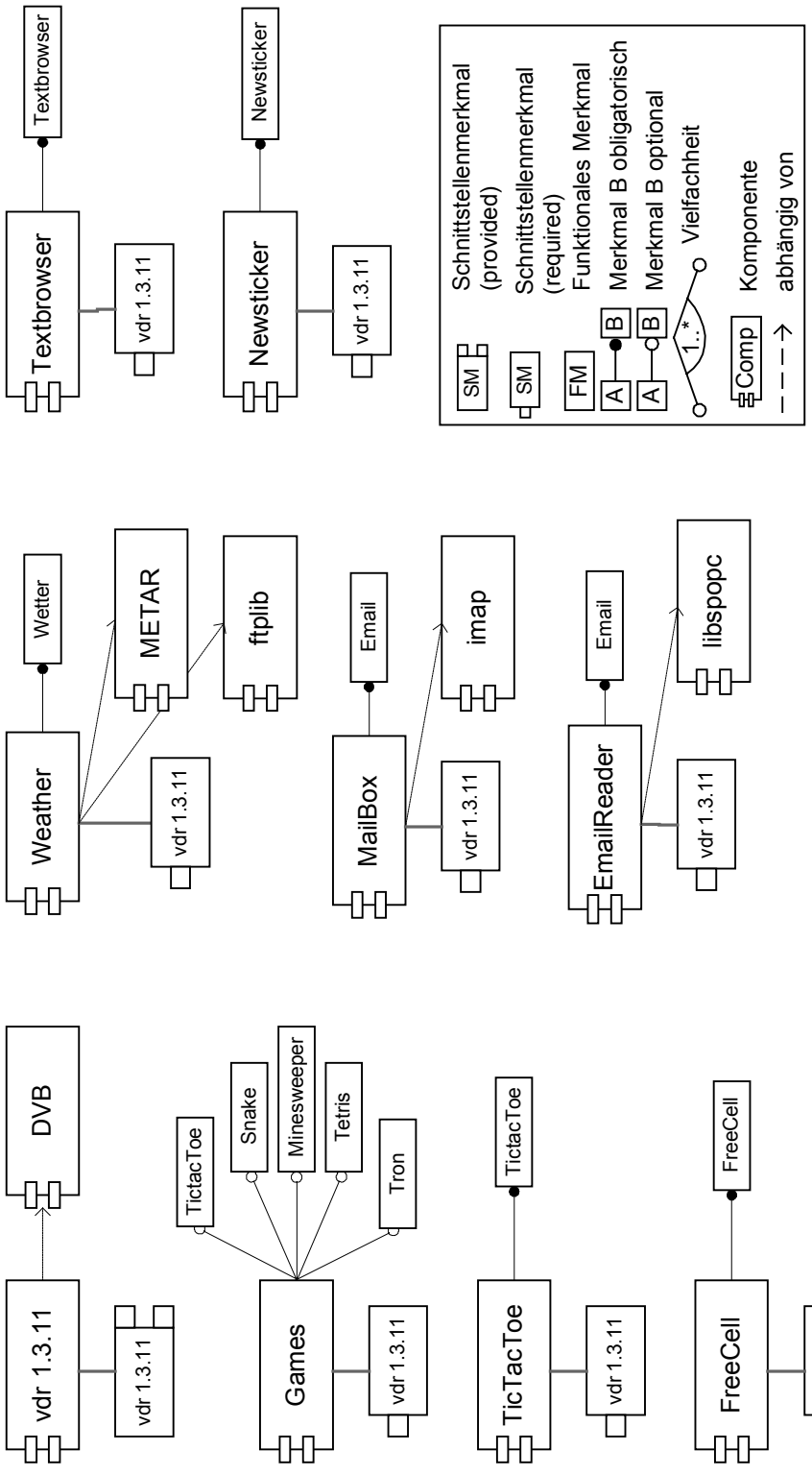

Abb. 7-3: Auswahl eingesetzter Komponenten

Glossar

Ableitung	Im vorliegenden Fall handelt es sich bei der Ableitung um die Erstellung einer installierbaren Fassung eines Programms bestehend aus vorgefertigten Komponenten..
Deployment	Siehe Installation.
Domäne (engl. domain)	Als Domäne werden bestimmte Arbeits- und Wissensgebiete bezeichnet. Wird Software für eine Domäne entwickelt, sind immer wiederkehrende und für diese Domäne typische Probleme zu lösen. Diese Probleme werden als *domänenspezifisch* bezeichnet. ([Str04], S. 10)
eClass	Dabei handelt es sich um einen Standard für Materialklassen und Warengruppen.
Installation	Entspricht der Installation der Komponenten auf einem spezifischen System und die Anpassung an dessen Gegebenheiten. Diesen Prozess nennt man auch Deployment. Es setzt eine Ableitung voraus.
Interoperabilität	„Zusammenarbeit in einem offenen System (gemäß dem Client/Server-Modell). Unabhängig von der verwendeten Hardware, den eingesetzten Betriebssystemen, der verwendeten Netzwerktechnologie und der Realisierung einer Anwendung kann eine Zusammenarbeit zwischen diesen Anwendungen erfolgen.“ [oV02]
Komponentenmonteur	Fachkraft, welche die Zusammenstellung der notwendigen Komponenten vornimmt, die Anforderungen des Kunden interpretiert und das Werkzeug zur Ableitung bedienen kann (in ([Szy03], S. 497f) als *Component Assembler* bezeichnet). Da im Deutschen ein zweideutiges Verständnis des Begriffs *Assembler* existiert, wird im Rahmen dieser Arbeit vom *Komponentenmonteur* gesprochen.

Komposition	Eine Komposition im Sinne der Komponentenlehre ist eine Zusammenstellung von Teilen, die ein Ganzes bilden. Die Teile einer Komposition sind die Komponenten. Bei der Zusammenstellung werden die Komponenten nicht verändert, ihr Verhalten kann aber über Ressourcen an die Komposition angepasst bzw. konfiguriert werden. ([Szy02], S. 550)
MAUT	*Multi-Attribute Utility Technique* Dabei handelt es sich um eine Entscheidungshilfe, mit der durch die Wichtung von Eigenschaften Gegenstände vergleichbar gemacht werden können. Verglichen werden können z. B. Komponenten anhand ihres Nutzwertes für den Kunden. (weitere Informationen: [Wal02], S. 115ff)
Monolithisches System	Ist ein Softwaresystem, das im Gegensatz zum komponentenorientierten System nicht durch Komponenten aufgebaut ist und somit seine Funktionalität nicht auf klar abgrenzbare Komponenten verteilet, die jeweils bestimmte Teilfunktionen zur Verfügung stellen. ([Neu01], S. 154)
Produktlinie	Begriff für eine Systemfamilie aus der Domäne der Betriebswirtschaft. Wird synonym für Systemfamilie verwendet. ([Str04], S. 210)
Ressourcen (i.V.m. dem Komponentenbegriff)	Ressourcen verändern das Erscheinungsbild der Komponente. Um diese an ein spezielles System anzupassen, kann auf die Ressourcen zurückgegriffen werden. „Resources [...] used by the component for presentation and user interface purposes“([Szy02], S. 552)

Schnittstelle (engl.: interface)	Ist eine Übergangs- oder Verbindungsstelle, z. B. zwischen Programmen. Über sie erfolgt der Austausch von Daten. Durch spezielle Interfaces kann man Teile mit unterschiedlichen Normen miteinander verbinden. ([oV94], S. 309f)
Systemfamilie	Bezeichnung eines Softwaresystems, welches durch seine Referenzarchitektur die effiziente Ableitung von Familienmitglieder ermöglicht. ([Str04], S. 211)

Abkürzungsverzeichnis

AIFF	Audio Interchange File Format
COTS	Commercial off the shelf
FAST	Family-Oriented Abstraction, Specification and Translation
FCL	Feature Constraint Language
FeatuRSEB	Featured RSEB
FODA	Feature-Oriented Domain Analysis
FORE	Family Oriented Requirements Engineering
KobrA	Komponentenbasierte Anwendungsentwicklung
MAUT	Multi-Attribute Utility Technique
MP3	MPEG Audio Layer-3
OCL	Object Constraint Language
OSD	On Screen Display
RSEB	Reuse-driven Software Engineering Business
UML	Unified Modeling Language
vdr	Video Disk Recorder
WAV	Wavefrom Audio Format
WMA	Windows Media Audio

Erklärung über Hilfsmittel

Die vorliegende Arbeit habe ich selbständig und ohne Benutzung anderer als der angegebenen Quellen angefertigt. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Quellen entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Ilmenau, den _____ _____

Marco Kaiser