Featuregesteuerte Architekturgestaltung zwecks Wartbarkeit und Evolution von Produktlinien

Periklis Sochos, Matthias Riebisch, Ilka Philippow

FG Softwaresysteme / Prozessinformatik
Technische Universität Ilmenau
98684 Ilmenau, Germany
{periklis.sochos|matthias.riebisch|ilka.philippow}@tu-ilmenau.de
http://www.theoinf.tu-ilmenau.de/pld

Abstract: Die Wartung und Weiterentwicklung von Software-Produktlinien stellen wegen des Umfangs und der angestrebten Nutzungsdauer herausfordernde Aufgaben dar. Die Zerlegung gemäß Separation of Concerns bildet ein grundlegendes Prinzip zur Verbesserung der Wartbarkeit eines Systems, weil es zu starker Kapselung und zu geringer Kopplung von Komponenten führt. Features stehen bei der Entwicklung von Produktlinien wichtige Abstraktionen zur Verfügung. Ihre direkte Verbindung mit der Architektur trägt zu vereinfachter Wartung und Evolution bei. In der hier vorgestellten Methode Feature-Architecture Mapping (FArM) werden Features zur Steuerung der Architekturentwicklung verwendet. Dieser Beitrag stellt die Umsetzung von (nicht-funktionalen) Qualitäts-Features durch die Methode FArM vor. Solche Qualitäts-bezogenen Features lassen sich üblicherweise nicht direkt in Architekturmerkmale abbilden; Defizite der Wartbarkeit sind eine häufige Folge. Das Vorgehen bei der Umsetzung wird anhand einer Produktlinie für mobile Kommunikationsgeräte erläutert, die Komponenten mit Plug-In-Schnittstellen aufweist.

1 Einleitung

Softwaresysteme sind nur so lange nutzbar, wie sie an geänderte Anforderungen angepasst werden können. Da bei der Entwicklung zukünftige Veränderungen nicht vorhersehbar sind, muss die Komplexität der Software so gering wie möglich gehalten werden, um evolutionäre Weiterentwicklung zu vereinfachen, was durch das Qualitätsmerkmal Wartbarkeit ausgedrückt wird. Typischerweise führt eine Vielzahl von Abhängigkeiten innerhalb von Software dazu, dass eine kleine Änderung der Anforderungen umfangreiche Änderungen der Implementierung nach sich ziehen. Die Einheiten der Software müssen logisch gegliedert, stark gekapselt und voneinander gut entkoppelt sein, um Wartbarkeit und Evolution zu unterstützen. Von den zahlreichen diesbezüglichen Ansätzen des Software Engineering hat das Prinzip *Separation of Concerns* – deutsch etwa Zerlegung nach Zuständigkeiten oder Aspekten – eine große Bedeutung. Dabei geht es um die Schaffung solcher Einheiten – im Folgenden als Architekturelemente bezeichnet – die jeweils nur eine Anforderung erfüllen. Bei einer Veränderung dieser Anforderung ist dann nur dieses Architekturelement betroffen.

Dieses Prinzip der Separation of Concerns lässt sich mit den meisten Implementierungstechniken nicht vollständig umsetzen, weil diese nur eine Zerlegung nach *einem* Aspekt

erlauben. Die Aspektorientierte Programmierung sowie weitere generative Programmiertechniken [CE00] erlauben zwar eine Umsetzung dieses Prinzips, wirken sich jedoch durch eine Zersplitterung des Quellcodes nachteilig auf dessen Verständlichkeit aus, wodurch die Wartbarkeit beeinträchtigt wird. Außerdem steht eine breite Akzeptanz dieser Techniken in der industriellen Praxis bisher aus.

Die hier vorgestellte Methode Feature-Architecture Mapping (FArM) verwirklicht das Prinzip der Separation of Concerns für einen Großteil der Anforderungen, ohne generative Techniken zu erfordern. Sie setzt das Prinzip damit zwar nicht vollständig um, leistet jedoch einen wichtigen Beitrag zu Wartbarkeit und Evolution. Die Methode wurde ursprünglich für Produktlinien auf Basis der Komponententechnologie entwickelt, weil hierbei üblicherweise Featuremodelle zur Verfügung stehen. Sie lässt sich jedoch auch unabhängig von Produktlinien zur Architekturentwicklung von Softwaresystemen einsetzen.

Features als für den Kunden sichtbare Merkmale stellen wichtige Abstraktionen von Anforderungen dar, die eine Reihe von Aktivitäten im Lebenszyklus von Software-Produktlinien steuern. Sie unterstützen die Kapselung von Anforderungen, widerspiegeln die Domäne und ihre Variabilität und erlauben eine Steuerung der Weiterentwicklung. Darüber hinaus stellen sie ein wesentliches Kommunikationsmittel zwischen den an der Entwicklung und Nutzung Beteiligten dar [RSP03]. Ein Feature ist dabei definiert als "a logical unit of behaviour that is specified by a set of functional and quality requirements representing an aspect valuable to the customer and system architect" entsprechend der Definitionen in [Bo00] und [Ri03]. Leider werden die Verbindungen zwischen Features und den Elementen der Architektur bisher methodisch nur wenig unterstützt, was zu Mängeln in der Wartung und Weiterentwicklung von Produktlinien führt.

Feature-Architecture Mapping (FArM) wurde als Methode entwickelt, um eine direkte Abbildung zwischen Features und Architekturelementen zu erreichen. Dazu führt die Methode FArM eine Reihe von Transformationen des Featuremodells durch, um im Sinne von Vorarbeiten der Architekturentwicklung die Features zu zerlegen oder zu gliedern, bis jedes Feature durch genau ein Architekturelement direkt implementierbar ist. Die Methode geht dabei vom ursprünglichen Featuremodell einer Produktlinie aus. Die Methode verzichtet auf generative Techniken zur Separation of Concerns wie etwa bei [OT01] zugunsten von besserer Wartbarkeit und geringerer Komplexität der Implementierung. Diese Vorteile werden allerdings mit Einschränkungen der Methode erkauft, da sich nicht alle Features a priori innerhalb eines einzigen Architekturelements implementieren lassen. Bei den bisherigen Anwendungen der Methode in den Domänen der Informationsverarbeitung und der mobilen Systeme sind solche Fälle allerdings noch nicht aufgetreten.

Dieser Beitrag befasst sich mit den Kernaktivitäten von FArM, den Transformationen auf Basis von Qualitäts-bezogenen Features. Kapitel 2 führt die Methode FArM und deren Prozesse ein. Kapitel 3 stellt die industrielle Fallstudie vor, die für die Beschreibung der Methode verwendet wird. Kapitel 4 geht kurz auf Transformationen von solchen Features ein, die die Architekturentwicklung nicht bestimmen, während Kapitel 5 den Prozess der Transformationen von Qualitäts-bezogenen Features vorstellt. Kapitel 6 zeigt die Implementierung dieser Transformationen im Umfeld der Produktlinie der mobilen Kommunikationsgeräte Blackberry. Die Kapitel 7 und 8 stellen verwandte Ar-

beiten sowie Zusammenfassung und Ausblick vor.

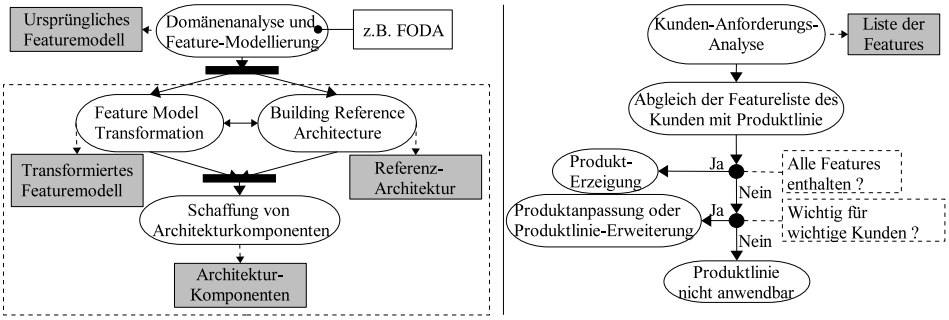

Abbildung 1: FArM - Phasen der Produktlinien- (links) und Produktentwicklung (rechts)

2 FArM - Feature-Architecture Mapping

Die Methode FArM ist in zwei Phasen der Produktlinien- und Produkt-Entwicklung aufgeteilt (Abbildung 1). Das ursprüngliche Featuremodell, das z.B. mit der Methode FODA [Ka90] erstellt wurde, wird in FArM in den zwei bidirektionalen Prozessen Feature Model Transformation (FMT) und Building Reference Architecture (BRA) verarbeitet und umgestaltet. Die Transformation FMT wird von einem Feature-Modellierungs-Team ausgeführt. Ausgehend von den Informationen im Featuremodell wie z.B. den Spezifikationen der einzelnen Features, wird das ursprüngliche Featuremodell transformiert und in engem Zusammenwirken mit dem Architekturteam ausgewertet. Das Architekturteam verarbeitet, prüft und bewertet im Rahmen des Architekturentwicklungs-Prozesses BRA die vorgeschlagenen Feature-Transformationen und setzt sie in Veränderungen der Systemarchitektur um. Der Transformationsprozess ist in vier Transformationsphasen eingeteilt, die in Abhängigkeit von architekturbezogenen und Qualitätskriterien, von Architekturanforderungen, von Implementierungserfordernissen sowie von Hierarchie- und Interaktionsbeziehungen zwischen Features durchlaufen werden (siehe Abbildung 2).

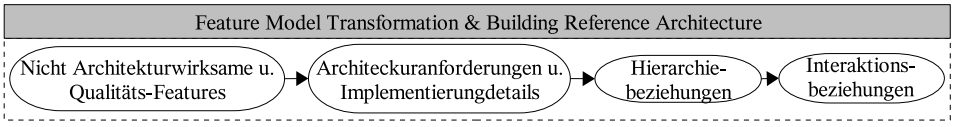

Abbildung 2: FArM Transformationsphasen

Die erste Transformationsphase bezieht sich auf die nicht für die Softwarearchitektur bestimmenden Features wie z.B. Gestaltung der Hardware, Sicherstellung von physikalischen Eigenschaften oder Preis, sowie auf Qualitäts-Features, die nichtfunktionale Merkmale beschreiben. Diese Features lassen meist keine direkte Zuordnung zu Architekturbestandteilen zu, sie werden deshalb vom Feature-Modellierungs-Team ermittelt und umgesetzt (siehe Kapitel 4 und 5). Danach erscheinen im Featuremodell nur noch funktionale Features. Das Architekturteam führt die nächste Transformationsphase aufgrund der Architektur-Anforderungen und Implementierungsdetails durch. Dabei werden

Features eingeführt, die nur von einem architekturbezogenen Standpunkt aus sichtbar sind. Idealerweise enthält danach das transformierte Featuremodell alle Features der Produktlinie, die variabel und damit Gegenstand der Produktgestaltung sind.

Die nächste Transformationsphase behandelt zusammengehörende Features. Das Feature-Modellierungs-Team prüft alle Verfeinerungs- und Enthaltenseins-Beziehungen zwischen Features, und parallel dazu untersucht das Architekturteam die Kommunikationsstruktur der dazugehörigen Systemkomponenten. Dabei werden die Komponentenschnittstellen festgelegt. Als Ergebnis dieser Transformationsphase sollte jedes Feature in genau einer Komponente implementierbar sein, und die Kommunikation zwischen den Komponenten wiederspiegelt die Struktur des Featuremodells. Als letzte Transformationsphase werden die Interaktionsbeziehungen zwischen den Features analysiert und umgesetzt. Dazu werden meist Kommunikationsstrukturen unabhängig vom Featuremodell etabliert. Muss in einer der Phasen ein neues Feature eingeführt werden oder tritt ein Konflikt auf, der nicht lösbar ist, müssen möglicherweise zu vorherigen Phasen zurückgekehrt werden.

Während aller Transformationsphasen sind eine Reihe von Veränderungen am Featuremodell vorgesehen, wie Hinzufügen neuer Features, Integration von Features, Aufteilung oder Veränderung der Anordnung eines Features im Featuremodell. Nach jeder Veränderung werden Traceability-Links zwischen ursprünglichem und transformiertem Featuremodell eingefügt bzw. aktualisiert [Ri04], die zur Beschreibung von Ziel und Entscheidungen benötigt werden, um die Verständlichkeit und Verfolgbarkeit und damit die Wartbarkeit zu sichern. Schließlich werden die Komponenten innerhalb einer Plug-In-Architektur implementiert, die die benötigte Kapselung und Entkopplung sicherstellt, die eine einfache Erstellung von Endprodukten aus der Produktlinie ermöglicht.

Dieser Beitrag konzentriert sich auf die erste Transformationsphase. Eine detailliertere Diskussion der Prozesse von FArM ist in [So04] und [SPR04] enthalten.

3 Die Blackberry-Produktlinie

Die Eignung der Methode FArM wurde bereits auf einer frühen Entwicklungsstufe durch Anwendung bei einer Fallstudie auf dem Gebiet der integrierten Entwicklungsumgebungen IDE überprüft [SPR04]. Für die Validierung der Methode in einem größeren Rahmen wird sie bei einer Fallstudie auf dem Gebiet der mobilen Kommunikationsgeräte eingesetzt, die anhand der Blackberry-Produktlinie durchgeführt wird [RIM04]. Die Blackberry-Plattform stellt Lösungen zur drahtlosen Kommunikation für einen breiten Anwendungsbereich zur Verfügung. Es gibt eine Vielfalt an Modellen für unterschiedliche Einsatzzwecke; durch Lizenzierung wird die Plattform auch durch Geräte weitere Hersteller wie Siemens und Nokia unterstützt. Abbildung 3 zeigt das Modell Blackberry 7230.

Das Gerät weist ein großformatiges Farbdisplay und eine menügesteuerte Navigation auf. Die Menüsteuerung erfolgt durch ein Rad, das den Aufruf und die Auswahl von Menüs erlaubt, und das gemeinsam mit einer Escape-Taste am rechten Rand angebracht ist. Eine QWERTZ-Tastatur ermöglicht die einfache Eingabe von Text und Daten. Eine USB-Schnittstelle ermöglicht die Kopplung zu PCs und weiteren Geräten; weitere

Schnittstellen wie IrDA und Bluetooth sind vorgesehen.

Dieser Beitrag geht von einem angenommenen ursprünglichen Featuremodell auf Basis der Gerätespezifikation und der Marktsituation aus, weil vertrauliche Firmeninformationen gewahrt werden müssen. Die FArM-Transformationen führen dann zu Varianten der verfügbaren Systeme. Die Fallstudie umfasst außerdem die Entwicklung eigener, zusätzlicher Komponenten, die auf Basis der Plug-In-Standards und der API der Blackberry-Plattform erstellt und mit Hilfe der verfügbaren IDE [RIM03] implementiert wurden. Abbildung 4 zeigt einen Ausschnitt des ursprünglichen Featuremodells. Die Bezeichnungen der Features werden in Englischer Sprache wiedergegeben, wegen der ebenfalls Englischen Bezeichnungen der Qualitätsmerkmale der ISO 9126 [ISO01].

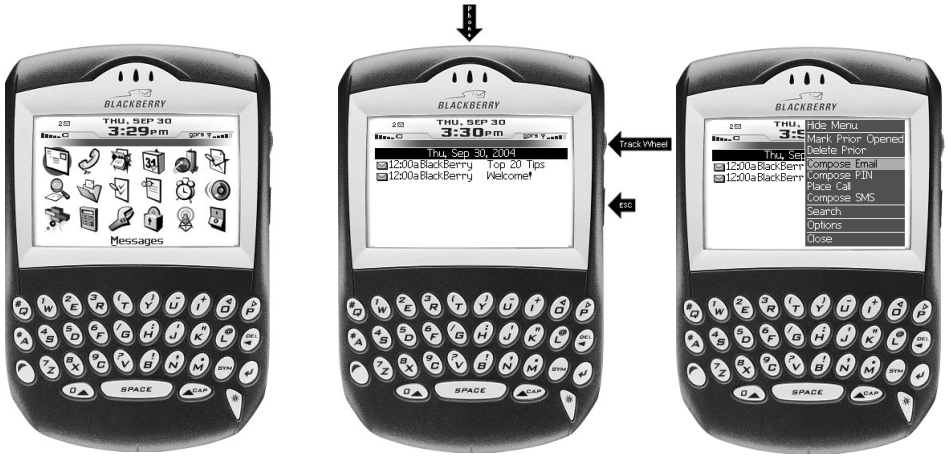

Abbildung 3: Blackberry 7230

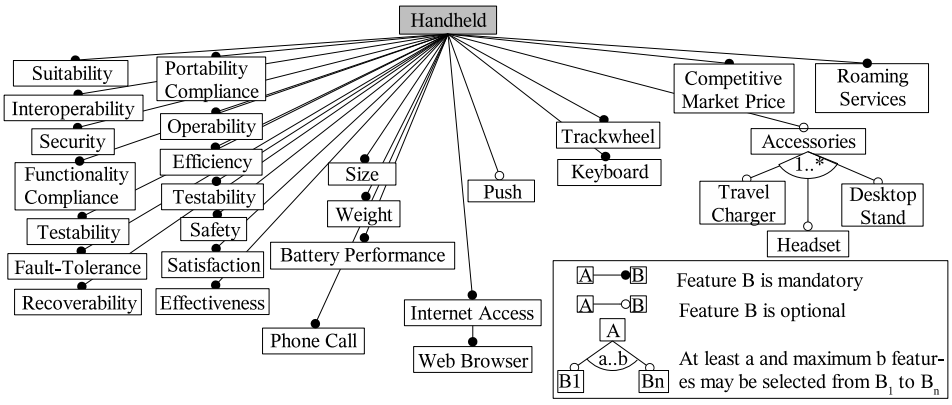

Abbildung 4: Ausschnitt des ursprünglichen Blackberry-Featuremodells

Die Formatierungsvorgaben für diesen Beitrag ermöglichen in den folgenden Abschnitten die Unterscheidung von Features ohne direkte Implementierung mittels Hervorhebung durch Fettdruck wie z.B. **Suitability**, während Features mit Umsetzung in eine

Komponente fett kursiv dargestellt werden, wie z.B. ***SMS***. Architekturkomponenten werden kursiv hervorgehoben, wie z.B. *Exception Handler*.

4 Transformationen aufgrund von Features, die die Architekturentwicklung nicht bestimmen

Die Transformationen aufgrund von Features, die für die Software-Architektur keine direkten Auswirkungen haben, werden hier nur kurz dargestellt, damit die logische Einheitlichkeit des Beitrags verbessert wird. Solche Features führen zu keiner direkten Umsetzung in Software-Architekturelemente.

Als Ausgangspunkt nutzen die Experten des Feature-Modellierungs-Teams die in FArM festgelegten Kategorien, um entsprechende Features zu finden. Nach Abstimmung mit dem Architektur-Team werden die Features in das transformierte Featuremodell umgesetzt, wobei Traceability-Links zwischen ursprünglichem und transformierten Featuremodell gebildet werden. Bei den FArM-Kategorien für Features, die für die Software-Architektur keine direkten Auswirkungen haben, handelt es sich um die folgenden:

- **Physikalische Features** – wiederspiegeln die physikalischen Eigenschaften des Systems. Bei der Blackberry-Produktlinie handelt es sich um **Size**, **Weight**, **Battery Performance**.
- **Geschäftliche Features** – hängen mit rein geschäftlichen Aspekten des Systems zusammen, wie z.B. dem Marketing. In dieser Fallstudie handelt es sich um **Competitive Market Price**, **Roaming Services** und **Accessories** mit den dazugehörigen Unter-Features.

Die Entscheidungen bei der Umsetzung dieser Features werden hier kurz wiedergegeben:

- **Size**, **Weight**, **Battery Performance.** Die Eigenschaften bezüglich **Size** und **Weight** haben keinen direkten Zusammenhang mit der Software-Architektur. **Battery Performance** wird definiert als die mit der Batterie mögliche Betriebsdauer, sie wird ebenfalls nicht direkt von der Software-Architektur bestimmt. Alle diese Features können durch Hardware-bezogene Entwurfsentscheidungen umgesetzt werden.
- **Competitive Market Price.** Dieses Feature ist durch unternehmerische Entscheidungen umzusetzen. Dazu gehören regelmäßige Risikoanalysen, ein qualifiziertes Entwicklerteam oder die Entscheidung für stabile und wirksame Technologien.
- **Accessories – Travel Charger, Headset, Desktop Stand.** Diese Features können durch direkt entsprechende Hardware-Elemente umgesetzt werden, ohne dass die Software-Architektur wesentlich beeinflusst wird.
- **Roaming Services** geben an, in welchen Ländern Weiterleitung ermöglicht wird. Dieses Feature wir durch Verträge örtlichen Unternehmen und durch deren serverbasierte Dienste umgesetzt, es beeinflusst die Software-Architektur

des Mobilgerätes nicht direkt.

5 Transformationen auf Basis von Qualitäts-bezogenen Features

Diese Transformationsphase wird auf den Ebenen der Featuremodelle (Phase FMT) und der Software-Architektur (Phase BRA) durchgeführt. Das Feature-Modellierungs-Team identifiziert alle Qualitäts-Features ausgehend von den FArM-Kategorien. Falls Qualitäts-Features im bisherigen Verlauf noch nicht quantitativ definiert worden sind, muss dies jetzt erfolgen, z.B. mittels der Profiles-Methode aus [Bo00]. Des weiteren werden eine oder mehrere der unten genauer beschriebenen FArM-Umsetzungsmethoden angewandt: Feature-konsistente Umsetzung, Integration in ein existierendes funktionales Feature oder Erzeugung neuer Features. Falls keine dieser Methoden anwendbar ist, wird das betreffende Feature mit Architekturmitteln umgesetzt. In diesem Fall nutzt das Architekturteam Architektur- oder Entwurfsmuster, um bisher nicht behandelte Features umzusetzen. Sie prüfen die vom Feature-Modellierungs-Teamvorgeschlagenen Transformationen und untersuchen deren Implementierbarkeit.

Die Methode FArM wendet das Qualitätsmodell der ISO 9126 [ISO01] an. Die Qualitäts-Features wiederspiegeln Qualitätsanforderungen; die durch diesen Standard definierte Kategorisierung ist deshalb für die Klassifikation von Qualitäts-Features anwendbar.
In der Blackberry-Fallstudie wurden folgende Qualitäts-Features identifiziert, die auf die ISO- Kategorisierung zurückgeführt werden können (wieder als englische Bezeichnungen): Functionality: **Suitability**, **Interoperability**, **Security**, **Functionality Compliance**, Reliability: **Fault-Tolerance**, **Recoverability**, Usability: **Operability**, Efficiency: **Efficiency**, Maintainability: **Testability**, Portability: **Portability Compliance**, Quality in Use: **Satisfaction**, **Safety**, **Effectiveness**.
Die vier bereits genannten Umsetzungsmethoden von FArM werden nun vorgestellt und im folgenden Kapitel im Rahmen der Fallstudie angewendet.

Direkte Umsetzung. Ein mit dieser Umsetzungsmethode behandeltes Feature des ursprünglichen Featuremodells wird nicht in das transformierte Featuremodell übernommen, weil dieses Feature für die Software-Architektur keine direkten Auswirkungen hat (siehe Kapitel 4). Dies trifft auch für einige der Qualitäts-Features zu. Stattdessen wird ein Traceability-Link eingeführt, der von diesem Feature zur Wurzel des transformierten Featuremodells zeigt und Informationen über die Entwicklungsaktivitäten für dieses Feature aufnimmt, wie z.B. der Einsatz einer Hardware-Komponente.

Integration in ein existierendes funktionales Feature. Falls Qualitäts-Features nicht direkt umgesetzt werden können, führt das Feature-Modellierungs-Team – als Teil linguistischer Textanalysen – eine lexikalische Analyse der Feature-Spezifikation, der Beschreibungen der zugeordneten Anforderungen sowie dazugehöriger Entwurfsentscheidungen durch. Bei dieser Analyse werden identifizierte lexikalische Strukturen der existierenden funktionalen Features verglichen. Ähnlichkeiten weisen auf Möglichkeiten hin, Qualitäts-Features mit funktionalen Features zu vereinigen, z.B. indem deren Beschreibung erweitert wird. Dieser Prozess kann durch entsprechende Werkzeuge der Textanalyse unterstützt werden.

Erzeugung neuer Features. Eine lexikalische Analyse wird ebenso für die Identifikati-

on von funktionalen Features durchgeführt, die eine Realisierung von Qualitäts-Features ermöglichen. Wesentlicher Schritt dabei ist die Identifikation von Objekten oder Subjekten in der Beschreibung der betreffenden Qualitäts-Features, die Hinweise auf neue funktionale Features enthalten. Die außerdem enthaltenen Verben und Adjektive beschreiben den Effekt sowie die Rolle der neuen Features in der Produktlinie.

Umsetzung durch Architektur- oder Entwurfsmuster. Lässt sich keine der genannten Umsetzungsmethoden anwenden, wird dieses Feature vom Architekturteam umgesetzt, wobei Architektur- oder Entwurfsmuster eingesetzt werden. Ein Ausschnitt einer Zusammenstellung von Architektur- oder Entwurfsmustern und ihrer Zuordnung zu Qualitäts-Features ist in Tabelle 1 wiedergegeben [Bo00], weitere solche Zuordnungen sind Gegenstand laufender Forschungsarbeiten auf dem Gebiet der Architekturentwicklung. Allerdings tragen Architektur- oder Entwurfsmuster in den meisten Fällen dazu bei, künstliche Abstraktionen einzuführen und die logische Strukturierung des Systems zu durchbrechen. Die Wartbarkeit wird dadurch nicht verbessert, und die Grenzen der Separation of Concerns ohne generative Techniken werden damit erreicht. Deshalb sollte diese Umsetzungsmethode von FArM nur als letzte Möglichkeit angewendet werden.

Tabelle 1: Architektur- oder Entwurfsmuster für Qualitäts-Features

Qualitäts-Feature	Architekturmuster	Entwurfsmuster
Reliability	Betriebssystemprozesse, DBMS, Model-View-Controller, Presentation-Abstraction-Control	Selbstbeobachtung, Redundanz, Fassade, Observer
Efficiency	Betriebssystemprozesse, Betriebssystem-Threads, Scheduler auf Applikationsebene	Observer
Maintainability	Betriebssystemprozesse, Remote Method Invocation, Presentation-Abstraction-Control	Fassade, Observer, Abstrakte Fabrik

6 Die Auflösung und Umsetzung der Blackberry-Qualitätsfeatures

Dieses Kapitel beschäftigt sich mit der Umsetzung der Qualitäts-Features der Blackberry-Produktlinie durch eine oder mehrere der Umsetzungsmethoden von FArM. Sie dient dabei zu deren Illustration und Erläuterung. Zwecks Vermeidung von Mehrdeutigkeiten werden die oben eingeführten englischen Bezeichnungen für Features und Komponenten, bei Bedarf mit deutscher Übersetzung verwendet.

Suitability. Dieses Qualitäts-Feature bezieht sich in der Blackberry-Produktlinie auf die Möglichkeiten eines Systems, bestimmte Aufgaben wie Durchführung eines Telefonanrufs, Zugriff auf eine Webseite oder Unterstützung der Push-Technologie. Dieses Feature wird durch seine Integration in existierende funktionale Features umgesetzt, wie in die Features **Phone Call**, **Internet Access** oder **Push**.

Interoperability. Für die Blackberry-Plattform bezieht sich dieses Feature auf Internet-, Unternehmens- oder Server-seitige Software wie Gateway Server, Unternehmens-Firewalls und Server-Software wie Microsoft Exchange Server oder Lotus Domino. Die Umsetzung dieses Features kann durch das Hinzufügen der entsprechenden Features erfolgen, wie ***Gateway, Firewall, Microsoft Exchange*** und ***Lotus Domino.***

Security - Schutz. Dieses Feature wird umgesetzt in die Verschlüsselungs- und Authentifikationseigenschaften des Blackberry. Zum einen werden Datenpakete verschlüsselt. Außerdem werden durch Einführung der Features ***Encryption*** und ***Code Sharing*** verhindert, dass Applikationen auf die Daten anderer Applikationen zugreifen oder unautorisierten Zugriff auf die API der Blackberry-Plattform vornehmen. Die unautorisierte Nutzung von Programmen oder Daten wird außerdem durch das **Login**-Feature beeinflusst, sowie durch die kommerzielle Berechtigungen und übliche Schutzmechanismen wie Beschränkung von Test- oder Demoversionen.

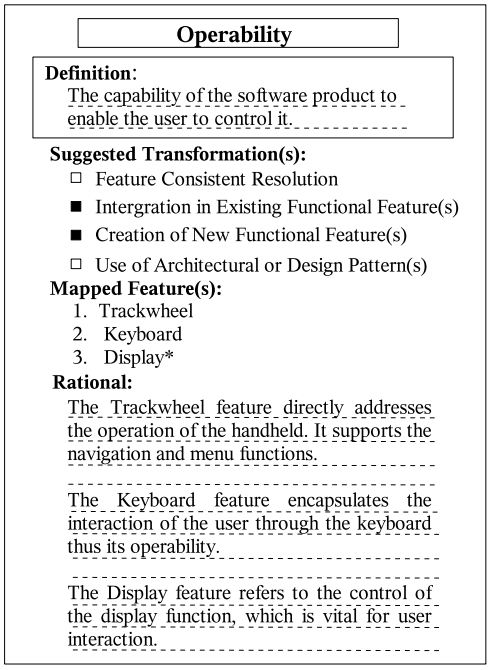

Abbildung 5: Beschreibung des Traceability-Links für Operability

Functionality Compliance. Zur Erfüllung der Anforderungen an die Funktionalität muss das System den existierenden Standards der Mobilfunk-Industrie entsprechen. Aus diesem Grund werden verschiedene Standards als neue Features zum transformierten Featuremodell hinzugefügt, wie das Feature ***Network Protocols*** mit den Unter-Features: ***HTTP, GPRS***, und ***WAP*** sowie ***Communication Standards*** mit den Unter-Features: ***BlueTooth*** und ***USB.*** Die entsprechenden Komponenten werden dann zur Architektur hinzugefügt.

Fault Tolerance. Dieses Feature wird auf Architekturebene durch die Einführung des

Features ***Exception Handler*** umgesetzt. Diese Komponente wird von den anderen Komponenten bei Auftreten einer Ausnahme benachrichtigt und veranlasst eine entsprechende Reaktion. Programmierrichtlinien werden steuern ebenfalls die Anwendung der Komponente *Exception Handler.*

Recoverability. Das Feature ***Exception Handler*** steuert, dass rechtzeitig die richtigen Operationen nach Ausnahmen und Fehlern vorgenommen werden. Die Daten-Wiederherstellung selbst wird von einem Feature ***Data Recovery*** bewirkt. Es stellt sicher, dass Sicherheitskopien der wesentlichen Systemdaten vorgenommen werden, die eine Wiederherstellung im Fehlerfall ermöglichen. Das Zusammenwirken der entsprechenden Komponenten bewirkt die Realisierung dieses Features.

Operability - Benutzbarkeit. Die lexikalische Analyse der Beschreibung dieses Features verdeutlicht seine Integration in die existierenden funktionalen Features ***Trackwheel*** und ***Keyboard*** sowie das Hinzufügen des Features ***Display*** zwecks Kapselung des Treibers für die Anzeige des Mobilgerätes. Zu letzterem gehört z.B. die Steuerung von Kontrast, Helligkeit und Hintergrundbeleuchtung. Die dem Traceability-Link hinzugefügte Beschreibung ist in Abbildung 5 dargestellt, wegen der lexikalischen Analyse in englischer Sprache.

Efficiency. Dieses Feature wird vom Feature-Modellierungs-Team anhand der ISO-9126-Klassifikation verfeinert zu den Qualitäts-Features **Efficient Time Behavior** und **Resource Utilization,** wobei letzteres wieter verfeinert wird zu den Qualitäts-Features **Efficient Memory Management** und **Efficient Storage Utilization.**

Efficient Time Behavior. Dieses Feature wird zunächst als die Zeitdauer einzelner Aktivitäten wie z.B. des Programmstarts, des Sicherns oder Öffnens einer Datei, des Ladens einer Webseite und der Bearbeitung interner Anfragen definiert. Für jeden Teil der Definition muss das Feature-Modellierungs-Team die entsprechenden funktionalen Features identifizieren und deren Beschreibung erweitern. Beispielsweise wird die Zeitdauer für die Ausführung eines Programms als Bedingung in die Beschreibungen aller funktionalen Feature der Blackberry-Plattform aufgenommen, die ein Programm implementieren, wie z.B. dem Feature **Login**. Außerdem liefert eine lexikalische Analyse der Beschreibungen der existierenden Features Informationen darüber, welche Features Dateien laden oder speichern. Die Beschreibungen dieser Features erhalten entsprechende Bedingungen. Die mit dem Internet zusammenhängenden Teile werden in die Features **Internet Access** und **Web Browser** integriert. Die weiteren Teile werden auf ähnliche Weise transformiert.

Efficient Memory Management. Diese Qualitäts-Feature wird durch eine Kombination der Methoden umgesetzt. Das Feature ***Memory*** wird hinzugefügt, um die Speichernutzung der Systemprozesse anzuzeigen und um die Ursache verlorener Elemente, sog. Leaks zu finden. Außerdem wird eine Reihe von Projektmanagement-Maßnahmen wie die Schaffung von Programmierrichtlinien [RIM03] eingeführt und eine API für die Abfrage der Speichernutzung zur Laufzeit definiert, ähnlich den Klassen Memory oder MemoryStats im Paket net.rim.device.api.system.

Efficient Storage Utilization. Hierfür wird das Feature Storage hinzugefügt, um den Speicher des Mobilgeräts zu verwalten und um eine einheitliche API zur Speicherverwaltung bereitzustellen.

Testability. Das Feature-Modellierungs-Team schlägt für dieses Feature nur seine Integration in das Feature ***Exception Handler*** vor, weil dieses zur Verifikation des Systemverhaltens während der Test beiträgt. Andererseits führt das Architekturteam während des BRA-Prozesses zur Verbesserung der Wartbarkeit das Feature ***Logger*** ein, das auf dem Observer-Pattern aus Tabelle 1 basiert.

Portability Compliance – Erfüllung von Portabilitätsstandards. Eine allgemeine Forderung der Portabilität besteht darin, dass sowohl Blackberry-Applikationen als auch -Komponenten auf verschiedenen – insbesondere früheren – Versionen der Plattform lauffähig sind. Dieses Feature wird durch Feature-konsistente Umsetzung behandelt, indem die folgende Menge von Lösungen eingeführt wird:

1. eine neue Blackberry-API muss eine Obermenge der früheren API umfassen,
2. nicht unterstützte Methoden-Implementierungen werden in der API-Dokumentation dargestellt,
3. der Compiler warnt vor künftig wegfallenden – sog. depreceated – Methoden, und
4. die Compilierung von depreceated oder neuen Methoden wird von Compiler verweigert.

Effectiveness, Safety, Satisfaction – Effektivität, Sicherheit, Benutzerzufriedenheit. Diese Qualitäts-Features werden vom Standpunkt des Nutzers aus untersucht. Wegen ihrer Allgemeingültigkeit haben sie einen Einfluss auf viele Aspekte des Systems. Sie werden umgesetzt durch eine Spezifikation der verschiedenen Umstände, unter denen das System getestet wird. Die Entwickler können dann mögliche Mängel des Systems identifizieren, die eins oder mehrere dieser Features beeinflussen könnten. Die dabei gefundenen Features werden erweitert, so dass sie die Qualitäts-Features erfüllen. Beispiel: Bei einem Test der Sicherheit des Systems zeigt sich unter bestimmten Benutzungsumständen ein Sicherheitsproblem. Die Entwickler überarbeiten das Qualitäts-Feature **Security** und verbessern parallel dazu die entsprechenden Komponenten ***Encryption***, ***Code Sharing*** und ***Login***.

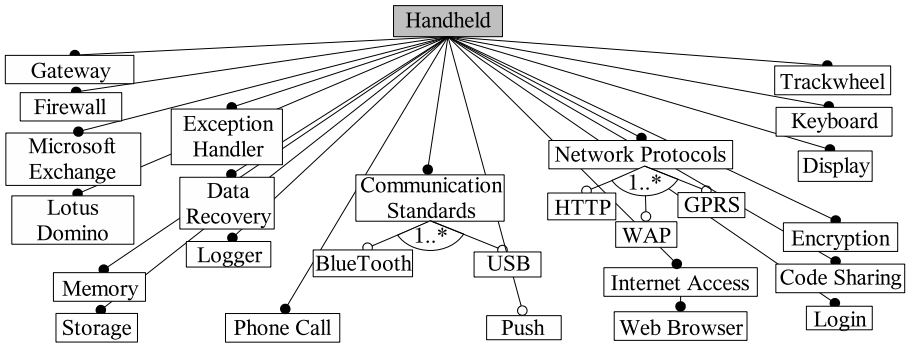

Abbildung 6: Transformiertes Featuremodell (Ausschnitt)

Das transformierte Feature-Modell als Ergebnis ist in Abbildung 6 dargestellt. Beim

Vergleich mit dem ursprünglichen Featuremodell von Abbildung 4 wird deutlich, dass nun ausschließlich funktionale Features enthalten sind. Beispielsweise ist das Qualitäts-Feature **Efficiency** nun in die vorher bereits vorhandenen Features ***Login***, ***Internet Access*** und ***Web Browser*** sowie in die beiden neuen Features ***Memory*** und ***Storage*** integriert. Dadurch wird die Zuordnung zwischen Features und der Architektur verstärkt. Die Verständlichkeit wird verbessert, und die Variabilität der Lösungen steigt.

7 Verwandte Arbeiten

Eine Reihe von Methoden für Produktlinien setzt Features in den Mittelpunkt der Entwicklungsanstrengungen. Die wichtigsten Methoden sind FODA [Ka90], sein Nachfolger FORM [Ka98] und FeatuRSEB [GJM98]. FODA konzentriert sich hauptsächlich auf die Modellierung der Domäne einer Produktlinie und trägt wenig zu Entwurf und Implementierung bei. FORM konkretisiert FODA in dieser Hinsicht, löst jedoch die Frage des Zusammenhangs zwischen Features und der Architektur nicht. FeatuRSEB entwickelte sich aus einer Wiederverwendungsmethode mit Use-Case-basiertem Vorgehen, in das Featuremodelle integriert wurden. Dabei werden Traceability-Links genutzt, um eine Abbildung von Features auf Use-Case-basierte Klassen zu erreichen. Als Ergebnis ergibt sich jedoch eine hohe Komplexität dieser Links selbst bei kleinen Produktlinien, wodurch der Nutzen dieser Abbildung verloren geht.

Einige generative Programmiertechniken wurden allein oder in Verbindung mit den obengenannten Methoden angewendet, um die Abbildung von Features zur Architektur im Sinne des Prinzips Separation of Concerns zu verbessern, wie der Hyperspace-Ansatz [OT01], der in Kombination mit FeatuRSEB zur Methode HyperFeatuRSEB weiterentwickelt wurde [Bo02], die Methode GenVoca [BG97] sowie die Aspektorientierte Programmierung [Ki97].

Der Hyperspace-Ansatz kapselt die zu einem Gesichtspunkt gehörenden Teile einer Software in Komponenten, die als Hyperslices bezeichnet werden. Solche Komponenten enthalten dann alle Methoden und Datenzugriffe, die spezifisch für diesen Gesichtspunkt sind. Durch einen Generator wird aus einer Menge solcher Komponenten ein lauffähiges System erzeugt, indem darauf Kompositionsmethoden angewendet werden. Die Methode HyperFeatuRSEB integriert den Hyperspace-Ansatz mit der FeatuRSEB-Methode, indem Hyperslices entsprechend der Features gebildet werden. Diese Features werden durch ein Use-Case-basiertes Vorgehen entwickelt. Die Implementierung der Features in Form von Hyperslices kann dann variabel zu Produkten einer Produktlinie zusammengefügt werden. Die Nachteile dieses Ansatzes zeigen sich in eingeschränkter Wartbarkeit der Produktlinie, die durch Abhängigkeiten zwischen den Hyperslices verursacht wird, die nicht im Modell dargestellt werden.

GenVoca bildet Systeme aus Schichten, die jeweils Programme enthalten, von denen jedes ein Feature implementiert. Seine Zerlegung basiert auf den gleichen Prinzipien wie die des Hyperspace-Ansatzes, woraus ähnliche Probleme entstehen.

Die Aspektorientierte Programmierung führt Aspekte als Quellcode-Beschreibungen von Gesichtspunkten ein, wodurch ebenfalls das Prinzip der Separation of Concerns umgesetzt werden soll. Aspekte stellen Features auf niedrigem Abstraktionsniveau dar. As-

pekt-Module sind schwer zu warten, weil ihr Zusammenwirken schwer verständlich ist. Außerdem konzentriert sich diese Methode mehr auf die Quellcode-Ebene als auf die Architektur. Eine detailliertere Diskussion ist in [Bo00] enthalten.

8 Zusammenfassung und Ausblick

Dieser Beitrag stellt einen Überblick über die Methode Feature-Architecture Mapping FArM vor. Diese Methode strebt eine stärkere Zuordnung zwischen den Features einer Produktlinie und seiner Architektur an. Eine Anzahl von Transformationen führt Veränderungen an Featuremodellen aus, um dieses Ziel zu erreichen. Dieser Beitrag konzentriert sich auf die erste Transformationsphase von FArM, die Transformation von Features, die keine direkte Auswirkungen auf einzelne Architekturelemente haben, sowie von Qualitäts-Features. FArM setzt solche Features in ein transformiertes Featuremodell um, damit eine direkte Implementierung eines Features in jeweils einer Komponenten möglich wird.

Damit wird eine direkte Abbildung zwischen Features und der Architektur erreicht. Außerdem werden Informationen über Entwurfsentscheidungen im Featuremodell bzw. an Traceability-Links angefügt. Dadurch steigt die Verständlichkeit der Lösung, was sich auf die Wartbarkeit positiv auswirkt.

Engere Abhängigkeiten zwischen Features wie zum Beispiel sogenannte Feature Interactions werden durch die eine Reihe von Methoden behandelt, die gegenwärtig Gegenstand der Entwicklung sind [Ca02]. Bei den Featuremodell-Transformationen nach FArM werden solche Abhängigkeiten im Featuremodell beschrieben, um ihre Entkopplung durch diese Methoden zu unterstützen.

Veränderungen auf der Ebene von Features werden direkt unterstützt. Durch Komponenten, die Features direkt entsprechen, ist eine Konfiguration von Systemen entsprechend gewünschter Eigenschaften leicht möglich. Die von FArM favorisierte Plug-in-Architektur für Komponenten unterstützt solche Veränderungen ebenfalls. Veränderungen von Features selbst sind ebenfalls gut steuerbar, weil sie sich auf die Implementierung innerhalb einer Komponente konzentriert.

FArM kann die Wartbarkeit der Systeme unmittelbar als Ziel einsetzen, indem sie diese Eigenschaft als Qualitäts-Feature einer Produktlinie formuliert und dieses Feature durch Transformation in eine Architektur-Lösung umsetzt. Schließlich wird durch direkte Modellierung der Abhängigkeiten zwischen Features deren Auflösung unterstützt und eine Entkopplung erreicht. Durch die Verringerung der Anzahl der Abhängigkeiten wird die Komplexität der Lösung verringert. Damit werden gute Voraussetzungen für eine evolutionäre Weiterentwicklung über lange Zeiträume geschaffen. Die Blackberry-Produktlinie als industrielle Fallstudie wird zur Illustration der Methode und zum Nachweis ihrer Eignung verwendet.

9 Literatur

[BG97] Batory, D.; Geraci, B.J.; Composition Validation and Subjectivity in GenVoca Generators. In: IEEE Transactions on Software Engineering, vol. 23, no. 2, 1997, pp. 67–82.

[Bo00] Bosch, J.: Design & Use of Software Architectures - Adopting and Evolving a Product Line Approach. Addison-Wesley, 2000.

[Bo02] Boellert, K.: Objektorientierte Entwicklung von Software-Produktlinien zur Serienfertigung von Software- Systemen. Dissertation, TU Ilmenau, 2002.

[Ca02] Calder, M.; Kolberg, M.; Magill, M.H.; Reiff-Marganiec, S.: Feature Interaction – A Critical Review and Considered Forecast. Elsevier: Computer Networks, Volume 41/1, 2003. S. 115-141

[CE00] Czarnecki, C.; Eisenecker, U.: Generative Programming. Addison Wesley, 2000.

[GJM98] Griss, M.; Favaro, J.; d'Allesandro, M.: Integrating Feature Modeling with the RSEB, In: Proceedings of the 5th International Conference on Software Reuse (ICSR '98), pp. 76–85, 1998.

[ISO01] ISO: Software Quality Standard ISO/IEC 9126-1:2001. ISO, 2001.

[Ka90] Kang, K.; Cohen, S.; Hess, J.; Novak, W.; Peterson, A.: Feature- Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1990.

[Ka98] Kang, K.; Kim, S.; Lee, J.; Kim, K.; Shin, E.; Huh, M.: FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures, In: Annals of Software Engineering, vol. 5, 1998 , S. 143–168.

[Ki97] Kiczales, G.: Aspect-Oriented Programming, In: Proceedings of the 1997 European Conference on Object-Oriented Programming (ECOOP '97), Springer, 1997; S. 220–242.

[Ri04] Riebisch, M.: Supporting Evolutionary Development by Feature Models and Traceability Links. In: Proceedings 11th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS2004), Brno, Czech Republic, May 24-26, 2004, S. 370-377.

[Ri03] Riebisch, M.: Towards a More Precise Definition of Feature Models. In: M. Riebisch, J. O. Coplien, D, Streitferdt (Eds.): Modelling Variability for Object-Oriented Product Lines. BookOnDemand Publ. Co., Norderstedt, 2003. S. 64-76.

[RIM03] RIM: BlackBerry Java Development Environment Version 3.6 Developer Guide; Volume 1 – Fundamentals. Research In Motion Ltd. 2003, vol. 1.

[RIM04] RIM: Blackberry Handheld. Research In Motion Ltd. Online verfügbar unter http://www.blackberry.com, Stand 29.10.2004.

[RSP03] Riebisch, M.; Streitferdt, D.; Pashov, I.: Modeling Variability for Object-Oriented Product Lines. In: Buschmann, F.; Buchmann, A. P.; Cilia, M. (Eds.): Object-Oriented Technology. ECOOP 2003 Workshop Reader. Springer, Lecture Notes in Computer Science , Vol. 3013, 2004, pp. 165 - 178.

[OT01] Ossher, P.; Tarr, H.: Multi-Dimensional Separation of Concerns and the Hyperspace Approach. In: Software Architectures and Component Technology. Kluwer Academic Publishers, 2001. Kap. 10.

[So04] Sochos, P.: Mapping Feature Models to the Architecture, In: Proceedings of the First International Software Product Lines Young Researchers Workshop (SPLYR), 2004, S. 51–60.

[SPR04] Sochos, P.; Philippow, I.; Riebisch, M.: Feature-Oriented Development of Software Product Lines: Mapping Feature Models to the Architecture. In: Weske, M., Liggesmeyer, P. (Hrsg.): Object-Oriented and Internet-Based Technologies. Springer, LNCS 3263, 2004, p. 138-152