45th International Scientific Colloquium
Ilmenau Technical University
October 04 - 06, 2000

Detlef Streitferdt / Kai Böllert / Matthias Riebisch

Feature-Modell und Architektur einer Systemfamilie

Einleitung

Ein Großteil der Aufwände in der Softwareentwicklung wird in die Anpassung, Weiterentwicklung und Änderung bestehender Systeme investiert. Durch Wiederverwendung der Entwicklungsleistungen sollen Kosten und Zeit eingespart werden. In diesem Beitrag betrachten wir Systeme, die aus der gleichen Domäne stammen. Nicht nur die Systeme selbst sind ähnlich, sondern auch viele der Kundenwünsche. Spezielle Kundenwünsche wurden bisher als Änderungen in existierende Systeme eingearbeitet. Dieses Vorgehen ist problematisch, da bereits nach wenigen Iterationszyklen viele Versionen vorhanden sind. Ihr Zusammenhang ist schwer nachvollziehbar, die Struktur der Systeme wird unübersichtlich und die Wartung erfordert einen hohen Aufwand.

In diesem Umfeld sind Systemfamilien ein aufkommender Ansatz zur schnelleren und kostengünstigeren Softwareentwicklung. Eine Systemfamilie beschreibt eine Gruppe ähnlicher Software-Systeme, denen eine gemeinsame Architektur sowie gemeinsam genutzte Komponenten zugrundeliegen. Darüber hinaus implementiert jedes Mitglied der Systemfamilie nur noch seine spezifischen Besonderheiten. Durch den hohen Grad an Wiederverwendung sowohl auf Modell- als auch auf Implementierungsebene können neue Familienmitglieder in kurzer Zeit mit hoher Qualität von kleinen Teams entwickelt werden.

Softwareentwicklung unter Berücksichtigung des Systemfamilienkonzeptes betrachtet in der Analysephase die Gemeinsamkeiten und Unterschiede der jetzt und in der Zukunft zu erstellenden Mitglieder der Systemfamilie. Ein Verfahren hierfür ist „Feature-Oriented Domain Analysis“ [4], welches gemeinsame und variable Eigenschaften von Familienmitgliedern in einem Feature-Modell dokumentiert. Diese Eigenschaften fließen in den Architekturentwurf der Systemfamilie ein.

Das Entwurfsziel ist eine generische Architektur [1][2], welche erst durch einen Generierungsschritt [3] in die Systemarchitektur einer konkreten Anwendung überführt wird. In Abbildung 1 sind die relevanten Teile des Softwareentwicklungsprozesses zu sehen. Bestehende Applikationen dienen als Eingabeinformation für die Domänenanalyse. Gemeinsamkeiten und Unterschiede werden herausgearbeitet und als Ergebnis in einem Feature-Modell abgelegt. Während der Entwurfsphase wird, unter Berücksichtigung der Variabilitäten, eine generische Architektur entwickelt, welche die Basis für die Generierung neuer Familienmitglieder darstellt.

Anforderungen sind entweder schon vorhanden oder müssen neu erarbeitet werden und sind den Features zugeordnet, die wiederum mit der generischen Architektur verbunden sind. Grau unterlegt sind die Modellteile, welche in diesem Beitrag näher betrachtet werden und zwischen denen eine neue Verbindung vorgestellt wird.

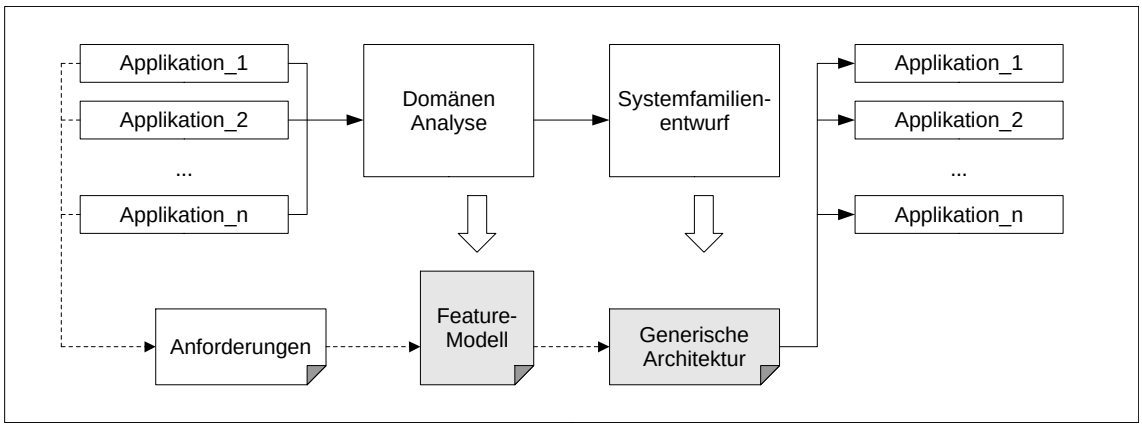

Abbildung 1: Entwicklungsprozeß

Im folgenden wird am Beispiel eines Bibliothekssystems erläutert, wie ein Feature-Modell und die zugehörige Architektur aussehen. Die Verbindung der beiden Teile ist Thema des letzten Abschnitts.

Feature-Modellierung

Das Feature-Modell ist ein Ergebnis der Domänenanalyse und enthält Gemeinsamkeiten und Unterschiede existierender sowie in Zukunft zu erstellender Anwendungen. Als Ausgangspunkt für die Entwicklung eines Feature-Modells dienen die Anforderungen von Bibliothekssystemen. Sechs dieser Anforderungen werden exemplarisch vorgestellt:

- Die Bibliothek muß Personen verwalten können.
- Die Bibliothek muß Bücher verwalten können.
- Bücher können von Bibliotheksnutzern ausgeliehen werden.
- Nach Ablauf der Leihzeit eines Buches wird der Bibliothekar informiert.
- Nutzer können per Post oder Email gemahnt werden.
- Nutzer können beliebige Bücher rezensieren.

Je nach Granularität wird eine Anforderung als Feature betrachtet oder mehrere Anforderungen werden zu einem Feature zusammengefaßt. Durch Auswahl von Features kann der Benutzer ein individuelles System zusammenstellen.

Ein Feature kann andere Features beeinflussen, was durch Bedingungen im Feature-Modell hinterlegt ist. Beispielsweise können Mahnungen in einer Bibliothek nur dann per Email versendet werden, wenn ein Internetanschluß vorhanden ist. Wird der Email-Versand als Feature gewählt, ist der Internetanschluß des Bibliothekssystems zwingend notwendig. Desweiteren existieren Features die zwar angegeben werden, aber nicht wählbar sind. Sie müssen auf jeden Fall im entstehenden Produkt enthalten sein und sind somit obligatorisch. Im Bibliothekssystem ist die Personenverwaltung obligatorisch.

Der Kunde kann durch Auswahl mehrerer Features entscheiden, was das gewünschte System leisten soll. Für ihn haben Features einen informellen Charakter. Für den Systementwickler sind die Features ein Einstiegspunkt in die interne Systemstruktur.

Features		System 1	System 2	System 3	System 4	System 5	System 6
Basisfunktionen	☑	x	x	x	x	x	x
Mahnwesen	☑	x	x	x	x	x	x
Benachrichtigung per Post	☐	x		x	x		x
Benachrichtigung per Email	☐		x	x		x	x
Rezensionen	☐				x	x	x
☑ - obligatorisch ☐ - optional							

Tabelle 1: Verschiedene Bibliothekssysteme

In Tabelle 1 sind die Features des Bibliothekssystems zu sehen. Das Feature *Basisfunktion* faßt die Verwaltung der Benutzer, der Bücher und den Leihverkehr zusammen, wobei diese Eigenschaften in jedem Bibliothekssystem vorhanden sein müssen. Berücksichtigt man alle obligatorischen und optionalen Features, ergeben sich in diesem Fall sechs unterschiedliche Systeme.

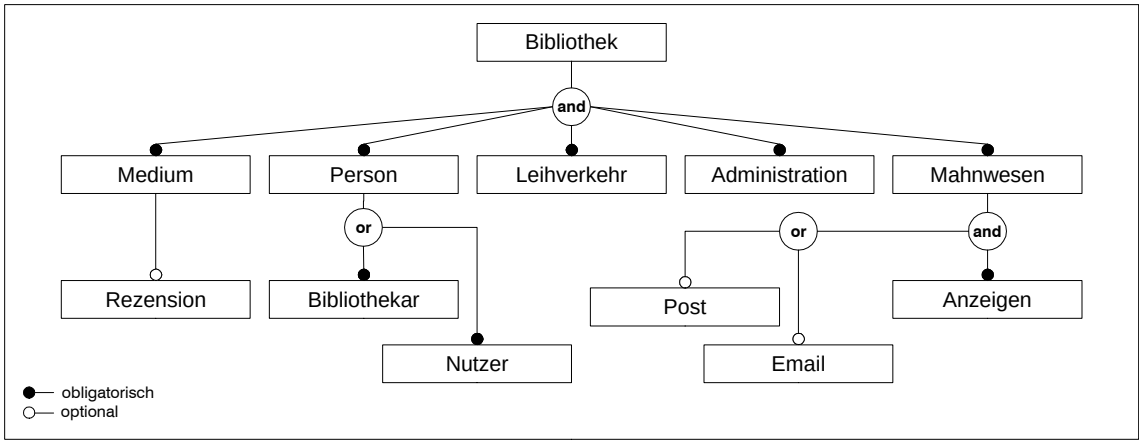

Abbildung 2: Feature-Diagramm der Bibliothek

Das Feature-Diagramm in Abbildung 2 ist die grafische Darstellung des Feature-Modells. Die Wurzel dieses Baumes, auch als Konzept bezeichnet, referenziert das System selbst. Alle weiteren Features werden mit Hilfe von logischen Operatoren im Baum angeordnet. Die Features sind über AND, OR oder XOR verknüpft.

Softwarearchitekturen

Softwarearchitekturen bieten eine implementierungsnahe Sicht auf eine Applikation und erleichtern dem Entwickler das Verständnis der Struktur und des Verhaltens eines Software-Systems. Durch die standardisierten Diagramme der Unified Modeling Language (UML) wird die Architektur beschrieben.

Ein Feature ist nicht auf einen bestimmten Bereich der Architektur begrenzt, sondern beeinflußt sie an mehreren Stellen, wie auch Änderungen der Anforderungen oder die Einarbeitung neuer Anforderungen eine verzweigte Beeinflussung der Architektur nach sich ziehen. Dadurch weicht die Architektur schnell auf und ist bereits nach wenigen Änderungszyklen schlecht verständlich und schwer wartbar. Die flexible Architektur einer Systemfamilie muß alle unterschiedlichen Features eines Systems enthalten. Än-

derungen und Erweiterungen sollen jederzeit eingearbeitet werden können, ohne die Übersichtlichkeit und Verständlichkeit der Architektur negativ zu beeinflussen.

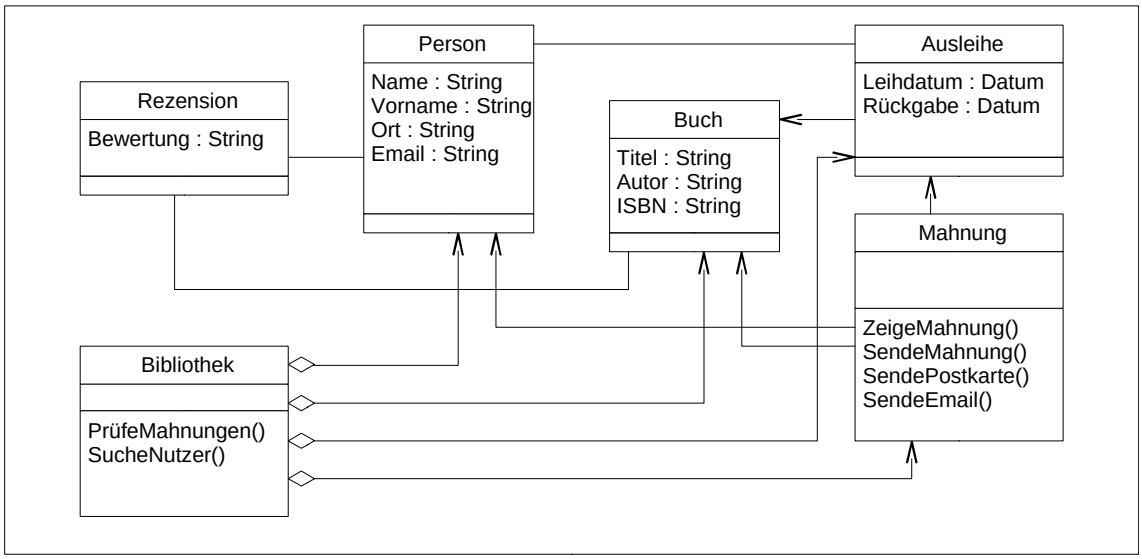

Abbildung 3: Statische Architektur der Bibliothek

In Abbildung 3 ist die statische Architektur durch ein Klassendiagramm dargestellt. Variable und gemeinsame Teile sind nicht direkt zu erkennen. Zudem sind die Features aus Tabelle 1 auf das gesamte Diagramm verteilt. Mahnungen werden durch eine eigene Klasse beschrieben, wirken sich aber zusätzlich auf die Attribute der Personen aus. Die Änderung des Mahnwesens einer Bibliothek erfordert die Bearbeitung der Klasse `Mahnung` und der Klasse `Person`. Besser wäre die Unterstützung durch ein methodisches Vorgehen.

Subject-Oriented-Design [5] bietet für dieses Problem eine Lösung an. Es ist möglich, eine flexible Architektur zu definieren, die jedes Feature getrennt betrachtet. Erst im Konfigurationsschritt werden die Teilmodelle zu einer Gesamtarchitektur zusammengefaßt, aus der eine lauffähige Applikation generiert wird.

Beispielhaft wurden für das Mahnwesen der Bibliothek drei Module erzeugt, die jeweils ein Feature realisieren. Abbildung 4 skizziert diese Module. Im einfachsten Fall werden alle Bücher der Bibliothek durchlaufen, um eventuelle Überschreitungen der Leihfrist festzustellen und gegebenenfalls Mahnungen auszusprechen. Das Modul *Anzeigen* stellt diesen Fall dar. Der Postversand von Mahnungen wird in einem eigenen Modul betrachtet. Jede Mahnung wird auf eine Postkarte gedruckt, wobei die Anschrift als neues Attribut einer Person hinzugefügt werden muß. Die obligatorische Anzeige einer Mahnung wird durch den Ausdruck einer Postkarte erweitert.

Die Konfiguration ermöglicht die Generierung der einzelnen Familienmitglieder, wobei die Zusammenstellung der Gesamtarchitektur aus den einzelnen Modulen über Regeln realisiert ist.

Wird ein Bibliothekssystem mit allen Features benötigt, so können im Fall des Mahnwesens die Klasse `Mahnen` und die Klasse `Person` einfach zusammenkopiert werden. Enthält die Bibliothek jedoch eine Methode, die je nach Modul ein anderes Verhalten aufweist, muß der Entwickler festlegen, wie die Zusammenführung der drei Module zu realisieren ist.

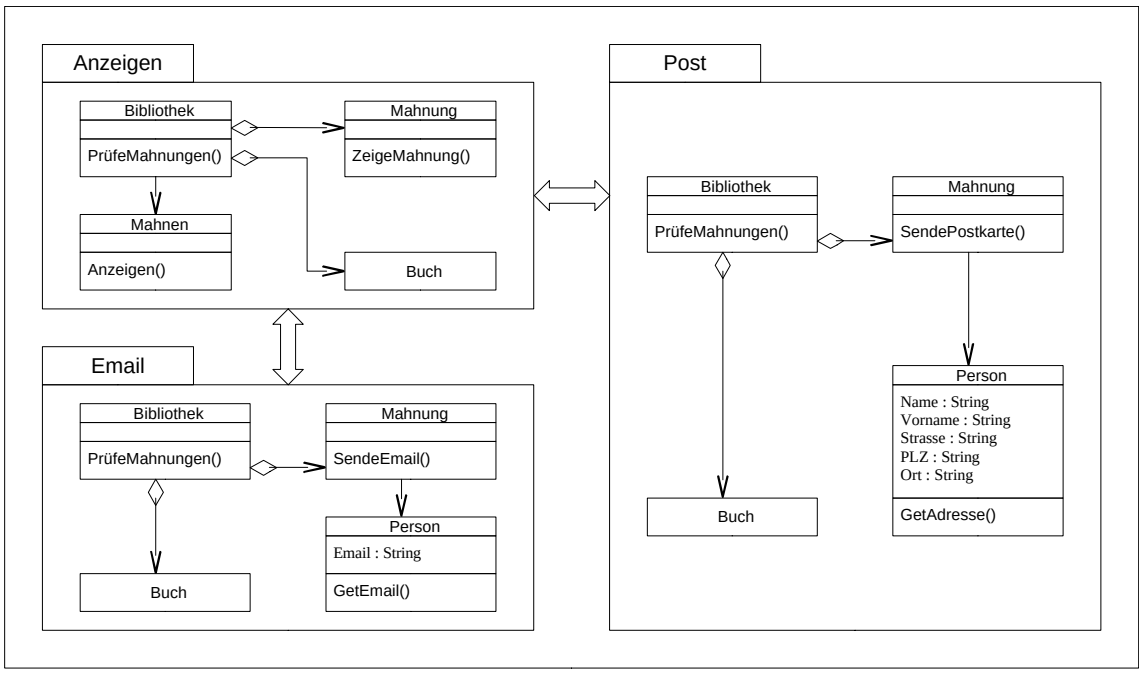

Abbildung 4: Flexible Architektur

Zuordnung von Features zu Modulen

Jedes Feature wird genau einem Modul zugeordnet, so daß es getrennt von der restlichen Architektur entwickelt werden kann. In Abbildung 5 sind die Verbindungen der verschiedenen Modellteile zu sehen. Die Feature-Modellierung und das Subject-Oriented-Design werden eingesetzt, um eine durchgängige Modellentwicklung zu gewährleisten und die Anforderungen über den gesamten Entwicklungsprozeß hinweg verfolgen zu können. Inkonsistenzen sind erkennbar und der Entwickler ist in der Lage Änderungen durch Abgleich der Modellverbindungen einzuarbeiten.

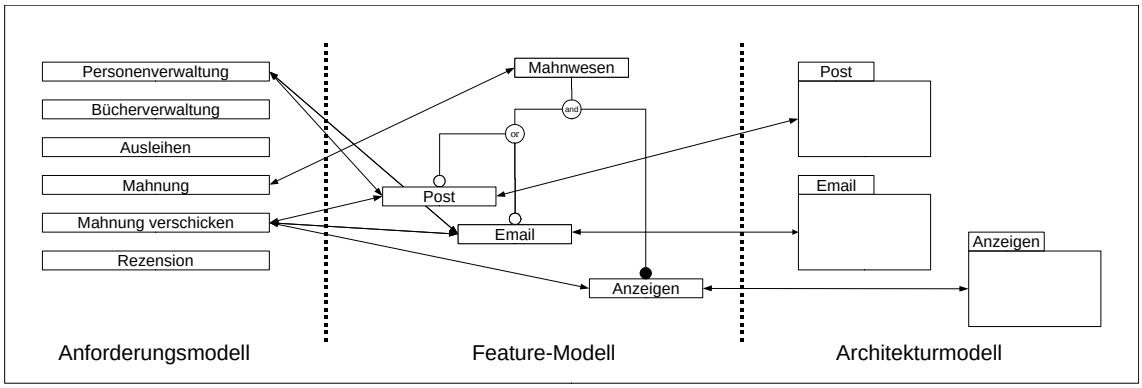

Abbildung 5: Interne Verbindungen des Systemmodells

Die gegenseitige Referenzierung (engl. *traceability*) aller Teilmodelle ist ein wichtiges Ziel moderner Softwareentwicklung. Eine Änderung der Anforderungen verweist auf die betroffenen Features und letztlich auf die betroffenen Module. Das so entstehende Modell erlaubt die Betrachtung der Systemfamilie durch das Feature-Modell und die übersichtliche Entwicklung einzelner Systemaspekte innerhalb von Modulen. Die Beziehungen von Features zu Modulen, ermöglichen einem Softwarehaus sofort nach Auswahl der Features durch einen Kunden ein entsprechendes System zu generieren.

Neue Kundenwünsche können in das Feature-Modell eingearbeitet werden und als Module die Architektur erweitern. Durch die Betrachtung eines Features als Modul, reduziert sich der Umfang auf wenige Architekturteile innerhalb eines Moduls, statt über die gesamte Architektur hinweg verstreut zu sein. Entwickler können sich auf ein Feature konzentrieren. Der Einfluß eines Features auf andere Module muß über Regeln explizit angegeben werden, wodurch Seiteneffekte von Anfang an berücksichtigt und vermieden werden können.

Es entsteht ein Modell, welches über diverse interne Referenzen die Konsistenz der enthaltenen Informationen wahren kann. Die Navigation im gesamten Modell ist dadurch möglich.

Zusammenfassung

Dieser Beitrag stellt den Zusammenhang zwischen der Benutzersicht auf die Eigenschaften eines Systems und der Entwicklersicht auf dessen interne Struktur her. Features werden aus den Anforderungen an die Systeme abgeleitet. Die bisher statische Architektur des Systems wird durch eine flexible Architektur ersetzt, die nach dem Systemfamilienansatz in einem Generierungsschritt die Erzeugung eines Zielsystems erlaubt. Diese flexible Architektur besteht aus einzelnen Modulen, die jeweils einem Feature zugeordnet sind. Die Auswahl von Features ergibt eine Konfiguration die ein bestimmtes Zielsystem beschreibt. Durch Zusammenführung der entsprechenden Module der Architektur, wird dieses Zielsystem generiert. Die Verbindung von Features und Modulen der Architektur erzeugt ein komplettes und durchgängiges Modell eines Software-Systems, dessen Konsistenz gewahrt bleibt.

Literaturverzeichnis

[1] D. Batory und B. Geraci, *Composition Validation and Subjectivity in GenVoca Generators*, IEEE Transactions on Software Engineering, Februar 1997, Seite 67—82.

[2] D.E. Perry, *Generic Architecture Descriptions for Product Lines, Development and Evolution of Software Architectures for Product Families*, 2nd Int. ESPRIT ARES Workshop, Februar 1998

[3] K. Czarnecki, *Generative Programming*, Dissertation, Technische Universität Ilmenau, 1999

[4] K. Kang et al., *Feature-Oriented Domain Analysis (FODA) Feasibility Study,* Technical Report CMU/SEI-90-TR-21, Carnegie Mellon Software Engineering Institute, 1990.

[5] S. Clarke et al., *Subject-Oriented Design: Towards Improved Alignment of Requirements, Design and Code*, OOPSLA '99, 1999

Autorenangabe(n):

Dipl.-Inf. Detlef Streitferdt

Dipl.-Wirtschaftsinformatiker (FH) Kai Böllert

Dr.-Ing. Matthias Riebisch

Technische Universität Ilmenau

Fakultät für Informatik und Automatisierung

Fachgebiet Prozeßinformatik

{detlef.streitferdt | kai.boellert | matthias.riebisch}@ theoinf.tu-ilmenau.de