

Integrierte und hybride Konstruktion von Software-Produktlinien

Kurzfassung

zur Erlangung des akademischen Grades Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von

Dipl. Inf. Ulrich Dinger
geboren am 28.04.1981 in Halle/Saale

Betreuender Hochschullehrer: Prof.Dr.rer.nat.habil.Dr.h.c. Alexander Schill

Dresden im April 2009

Einleitung, Motivation und Zielstellung

Durch die Industrialisierung im Automobilbau gelang es vor 100 Jahren, das Automobil zu einem einer breiten Masse zugänglichen Produkt zu machen. Durch zunehmende Konkurrenz in der Mitte des 20. Jahrhunderts wurden die Automobilhersteller gezwungen, verstärkt auf die Wünsche der Kunden nach personalisierten Fahrzeugen einzugehen. Dies gelang durch die Einführung von so genannten Produktlinien, wobei eine systematische Wiederverwendung von Fahrzeugkomponenten auch konzernübergreifend zum Einsatz kommt. Resultierend daraus ist es heute für einen Hersteller möglich, kostengünstig und schnell qualitativ hochwertige Fahrzeuge auf Basis von erprobten Komponenten zu realisieren.

Im Jahre 1968 wurde erstmals der Wunsch formuliert, diese Konzepte auch für die Software-Entwicklung anzuwenden. Daraus entstanden Ansätze, so genannte Software-Produktlinien zu erstellen, die mit Hilfe von Software-Kompositionssystemen wiederverwendbare Software-Komponenten zu angepassten Produkten assemblieren. Das oberste Ziel dabei ist wie im Automobilbau eine ingenieurmäßige Wiederverwendung von Software-Artefakten, um kostengünstig und schnell qualitativ hochwertige, an Kundenwünsche angepasste Software entwickeln zu können.

Die Ziele der Arbeit bestehen in der Untersuchung und Entwicklung von Ansätzen, um Software-Produktlinien neu erstellen zu können sowie existierende Produkte zu Produktlinien umzuarbeiten. Da die beiden Teilziele unterschiedliche Anforderungen beinhalten, sollen sie getrennt voneinander untersucht und in einem nachgelagerten Schritt die Gemeinsamkeiten zwischen beiden analysiert werden, um daraus Empfehlungen abzuleiten, wie Software entwickelt werden sollte, die nicht von Anfang an als Produktlinie konzipiert ist, ohne dabei die spätere Umarbeitung unnötig zu erschweren. Um den daran arbeitenden Menschen die Arbeit so leicht wie möglich zu machen, ist ein weiteres Ziel die größtmögliche Automatisierung des Prozesses der Erstellung und Wartung von Produktlinien sowie die Vermeidung der Speicherung redundanter Daten. Um die Wirksamkeit der Konzepte aufzuzeigen soll eine Referenzarchitektur erstellt und die dazu notwendigen Software-Werkzeuge realisiert werden.

Als Grundlage für die Arbeit dienen drei Thesen, die besagen, dass bei der Neuerstellung einer Software-Produktlinie der Einsatz einer automatischen Planungskomponente Vorteile gegenüber dem Stand der Technik bedeutet. Durch Nutzung einer einfachen, erweiterbaren Komponentenbeschreibungssprache lässt sich existierende Software so umarbeiten, dass daraus konfigurierbare Produkt-Basen als Grundstein für eine weitere Umarbeitung zu Software-Produktlinien erstellen lassen. Eine Kombination der automatischen Planungskomponente mit der Komponentenbeschreibungssprache ist möglich und dadurch entsteht die Möglichkeit, die Vorteile der Beschreibungssprache bereits während der Erstellung eines neuen Software-Systems ausnutzen und später umarbeiten zu können.

Um die Richtigkeit der vorgestellten Thesen und Konzepte zu beweisen, wird die Referenzarchitektur sowie die dazu notwendigen Software-Werkzeuge auf Basis der Programmiersprache Java als Erweiterungen der Eclipse-Entwicklungsumgebung realisiert und anhand von zwei Software-Projekten aus dem Umfeld der Medizintechnik, der *elektronischen Fallakte* und *Soarian Integrated Care*, evaluiert

Stand der Technik

Zur Realisierung von Software-Produktlinien wurden verschiedene Ansätze entwickelt, welche nachfolgend kurz vorgestellt werden. Im Anschluss daran wird deren konzeptionelles Vorgehen zusammengefasst und die damit im Zusammenhang stehenden Probleme erläutert.

Die im Rahmen einer Analyse erkannten Gemeinsamkeiten und Unterschiede zwischen einzelnen Instanzen einer Produktlinie werden beim *featureorientierten Ansatz* als Merkmale (engl. *Feature*) formalisiert. Zu diesem Zweck wurde ein so genanntes *Feature Modell* samt graphischer Notationstechnik eingeführt. Das Ziel von *Feature-Modellen* ist die Dokumentation über die generellen Fähigkeiten und Merkmale der Anwendungen aus der Sicht des Endbenutzers innerhalb der von der Produktlinie abgedeckten Domäne. Die Erstellung einer konkreten Variante der Produktlinie erfolgt durch Instanziierung des Feature-Modells, was als Eingabe für nachfolgende Transformations- und Generationsschritte eines Kompositionsprogramms im Sinne der Software-Kompositionssysteme dient.

Beim *entscheidungsorientierten Ansatz* werden die Eigenschaften der Produktlinie in einem so genannten Domänen-Modell hinterlegt. Zusätzlich erfolgt die Beschreibung von möglichen Entscheidungen, die während einer Variantenwahl getroffen werden können. Die Beschreibungen werden unter Berücksichtigung von technischen und nichttechnischen Beschränkungen zusammen mit der Abbildung auf konkrete Artefakte im so genannten Entscheidungsmodell gespeichert. Eine Variantenauswahl erfolgt durch das Treffen von Entscheidungen basierend auf diesem Entscheidungsmodell. Dies führt zu einer Zahl von Artefakten, die analog zum featureorientierten Ansatz mit Hilfe eines Kompositionsprogramms zu dem gewünschten Produkt assembliert werden.

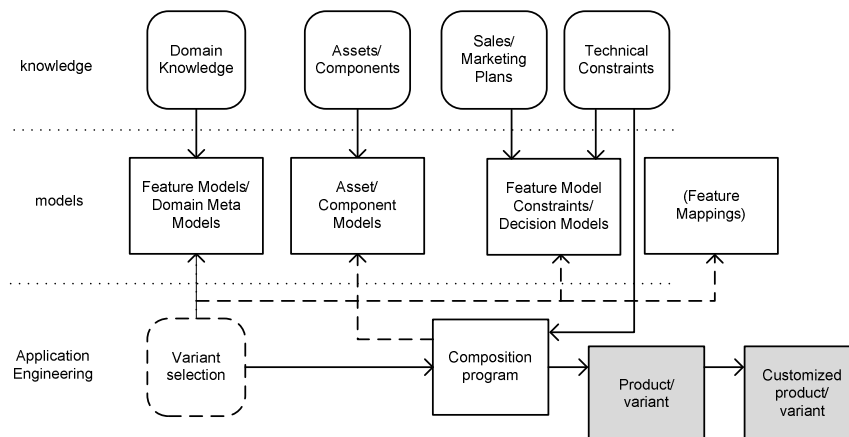


Abbildung 1: Konzeptionelles Vorgehen beim Software-Product-Line-Engineering

Abbildung 1 stellt das konzeptionelle Vorgehen beider Ansätze im Überblick dar, welches dem eines Software-Kompositionssystems entspricht. Das Wissen über die Domäne der Software-Produktlinie (*domain knowledge*) führt zur Erstellung der Feature-Modelle bzw. des Domänenmodells. Die zur Verfügung stehenden Artefakte (*assets*) bestehen unter anderem aus den Software-Komponenten und Dokumentationen und werden mit Hilfe entsprechender Modelle beschrieben. Technische sowie nicht-technische Einschränkungen führen zu den Beschränkungen (*constraints*) des Feature- bzw. des Entscheidungsmodells. Optional werden Abbildungen von Features bzw. Entscheidungen auf Artefakte des Systems (*mappings*) definiert. Soll eine konkrete Variante der Produktlinie erstellt werden, erfolgt die

Variantauswahl auf Grundlage der Feature- bzw. Entscheidungsmodelle. Diese Auswahl wird auf die Artefakte abgebildet, welche mit Hilfe eines Kompositionsprogramms unter Verwendung geeigneter Kompositionstechniken das gewünschte Produkt erstellen. In weiteren nachgelagerten Schritten kann die Variante erweitert und angepasst werden, um Sonderwünsche zu erfüllen, die die Produktlinie selbst nicht vorsieht.

Zu den Problemen, die im Zusammenhang mit diesem Vorgehen auftreten, zählt zum einen das manuell erstellte Kompositionsprogramm. Dabei handelt es sich um einen komplexen Arbeitsablauf, der parametrisiert durch eine Variantauswahl ausgeführt wird, um das gewünschte Produkt zu assemblieren. Die Erstellung und Wartung dieses Programms erfolgt von Hand. Eine zunehmende Komplexität der Produktlinie macht die Wartung für einen Menschen schwer und es muss beim Hinzufügen von neuen Kompositionsschritten eine steigende Zahl von Seiteneffekten beachtet werden. Das Kompositionsprogramm besteht aus einzelnen Kompositionsschritten, bei denen Artefakte bzw. Komponenten zu einem Produkt zusammengefügt werden. Ist die Integration bzw. Adaption einer Komponente über mehrere Kompositionsschritte verteilt, hat dies eine Fragmentierung des Kompositionsprogramms zur Folge, was es einem Entwickler wiederum dessen Wartung und Erweiterung erschwert. In den Feature- bzw. Entscheidungsmodellen findet eine Vermischung von technischen und nicht-technischen Beschränkungen statt. Zudem sind im Kompositionsprogramm sowie in den Komponentenmodellen technische Beschränkungen beispielsweise in Form von Reihenfolgeabhängigkeiten und Inkompatibilitäten zwischen einzelnen Komponenten implizit vorhanden, auf die während der Variantauswahl kein Zugriff besteht. Die Feature- und die Komponenten-Modellierung erfüllen im Rahmen des dargestellten Prozesses ähnliche Aufgaben. Während die Feature-Modellierung die nach außen sichtbaren Variabilitäten modelliert, werden durch die Komponenten-Modellierung die intern sichtbaren Variabilitäten des Software-Systems abgebildet. Dabei besteht das Problem der Grenzziehung zwischen intern- und extern sichtbaren Variabilitäten. Als weiteres Problem besteht die Tatsache, dass der Prozess der Produkthanpassung dem eigentlichen Vorgehen nachgelagert ist. Dies hat zu Folge, dass einmal angepasste Produkte nicht automatisiert umgearbeitet werden können. Nutzt man im Rahmen einer Umarbeitung vorhandener Altsoftware existierende Komponentenbeschreibungssprachen, so treten zusätzliche Probleme wie mangelnde Anpassbarkeit der Sprachen, Probleme bei der Referenzierung zwischen verschiedenen Modellen und unterschiedliche zu Grunde liegende Realisierungstechnologien auf.

Konzepte

Da bei der Entwicklung von Software-Produktlinien ohne existierende Altsysteme sowie bei der Umarbeitung existierender Systeme unterschiedliche Anforderungen existieren, werden zwei voneinander getrennte Konzepte vorgestellt, das *Forward-Engineering* und das *Reverse-Engineering*. Anschließend ermittelt der *hybride Ansatz* die Gemeinsamkeiten der beiden und leitet daraus Empfehlungen ab, wie Software zu entwickeln ist, die nicht von Anfang an als Software-Produktlinie konzipiert ist.

Forward-Engineering

Das *Forward-Engineering* als Prozess der Neuerstellung einer Software-Produktlinie setzt eine erweiterte Feature-Beschreibungssprache ein, die eine Modellierung der gesamten Variabilitäten der Software-Produktlinie ermöglicht. Dazu zählen die für einen Kunden nach außen sichtbaren Variabilitäten genau wie die intern sichtbaren Variabilitäten der technischen Komponenten. Durch eine semantische Beschreibung der Kompositionsschritte, die Komponenten in ein System integrieren, ist der Einsatz einer automatischen

Planungskomponente möglich, die auf Basis einer Variantenauswahl ein dazu passendes Kompositionsprogramm synthetisiert. Damit ist es möglich, alle technischen Beschränkungen zwischen Komponenten in den Kompositionsschrittbeschreibungen zu kapseln, was zu einer Trennung von technischen und nicht-technischen Beschränkungen führt. Eine Variantenauswahl besteht aus einer Instanzierung des Feature-Modells, wobei bereits während dieses Prozesses die entsprechenden Beschränkungen des Feature-Modells überprüft werden. Um die in den Kompositionsschritten enthaltenen technischen Beschränkungen zu respektieren, wird bereits während der Variantenauswahl versucht, mittels der Planungskomponente ein gültiges Kompositionsprogramm zu synthetisieren. Der Prozess der Produkthanpassung, d. h. eine Anpassung einer Variante an Anforderungen, die die Produktlinie bisher nicht unterstützt, erfolgt durch Erweiterung der Feature-Modelle, Bereitstellung entsprechender Kompositionsschritte und Einsatz der automatischen Planungskomponente zur Synthetisierung des Kompositionsprogramms. Abbildung 2 stellt das Konzept des *Forward-Engineering* graphisch dar.

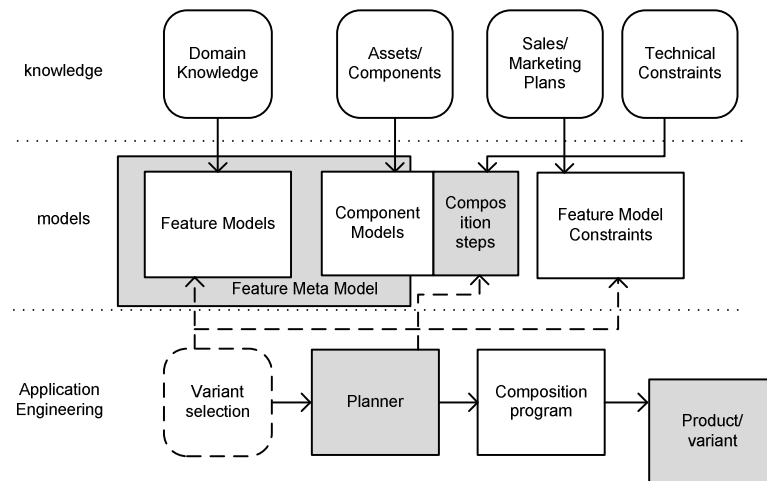


Abbildung 2: Geändertes Konzept zur Realisierung von Software-Produktlinien

Zur Umsetzung der in Abbildung 3 dargestellten Referenzarchitektur werden verschiedene Meta-Modelle bzw. Beschreibungssprachen benötigt. Dazu zählt zum einen das bereits erwähnte erweiterte Feature-Meta-Modell, mit dem sich die Variabilitäten der Produktlinie sowie der daran beteiligten Komponenten abbilden lassen. Zur Realisierung der einzelnen Kompositionsschritte sowie der Kompositionsprogramme wird eine Web-Service-Programmiersprache eingesetzt, zu deren Ausführung wird die entsprechende Ausführungsumgebung (Workflow-Engine) verwendet. Die Programmiersprache wird um ein Meta-Modell zur semantischen Beschreibung der Kompositionsschritte erweitert, welches aus den Meta-Modellen zur Beschreibung von Vorbedingungen sowie Effekten besteht und auf dem Feature-Meta-Modell basiert. Die syntaktischen Schnittstellen und semantischen Beschreibungen der Kompositionsschritte werden in einer Service Registry hinterlegt, auf die die Planungskomponente Zugriff hat. Eine Anfrage an die Planungskomponente wird als semantische Beschreibung des gewünschten (komplexen) Kompositionsschrittes formuliert. Da diese technische Sicht für Mitarbeiter einer Vertriebsabteilung von Ort nicht das geeignete Abstraktionsniveau darstellt, wird eine entsprechende Benutzerschnittstelle bereitgestellt, die eine Variantenauswahl über eine baumbasierte Sicht erlaubt und die Konvertierung in das Anfrageformat vornimmt. Die Referenzarchitektur ist als eine serviceorientierte Architektur unter Verwendung der Web-Service-Technologien konzipiert. Aus diesem Grund sollen alle beteiligten Komponenten als

Web-Services realisiert werden und die Kommunikation zwischen ihnen über das Austauschen entsprechender Nachrichten erfolgen.

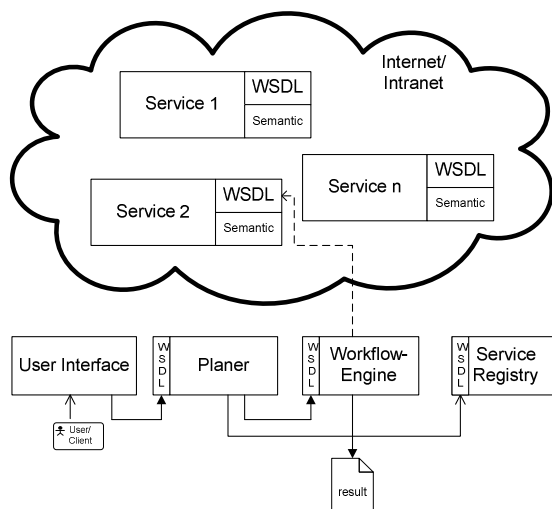


Abbildung 3: Referenzarchitektur für das *Forward-Engineering*

Reverse-Engineering

Um im Rahmen der Umarbeitung existierender Software zu einer Software-Produktfamilie, dem *Reverse-Engineering*, so wenig wie möglich Änderungen an dem existierenden System vornehmen zu müssen und zu so vielen Zeitpunkten wie möglich ein lauffähiges System vorhalten zu können, werden logische, technische Komponenten zusätzlich zu dem existierenden System modelliert und ihre Konfigurationsmöglichkeiten beschrieben. Die ursprüngliche Applikationslogik bleibt erhalten und muss durch Erstellung von so genannten *Wrappern* in der Form angepasst werden, dass die beschriebenen Konfigurationsmöglichkeiten berücksichtigt werden.

In einem ersten Schritt sind das gesamte existierende Altsystem sowie alle zusätzlich modellierten Komponenten Teil aller ausgelieferten Produkte. Es findet keine Anpassung des Build-Prozesses bzw. des Kompositionsprogramms abhängig von der Variantenauswahl statt. Die Unterscheidung findet lediglich durch Anpassung der Komponentenmodelle statt. Dieser Prozess wird als *Konfiguration* bezeichnet. Um die Konfigurationen beim Starten eines Produktes zuweisen zu können, wird ein entsprechender Mechanismus benötigt, welcher in Form eines Komponenten-Containers vorgesehen ist. Abbildung 4 stellt das Konzept des *Reverse-Engineering* als Abänderung des in Abbildung 1 vorgestellten Ansatzes graphisch dar.

Das Kernkonzept des *Reverse-Engineering* besteht aus einem einfachen, erweiterbaren Komponenten-Meta-Modell. Die Einfachheit drückt sich dadurch aus, dass der Kern dieses Meta-Modells lediglich einen Mechanismus zur eindeutigen Identifizierung von Komponenten sowie zur Speicherung von Teilen des Komponenten-Modells vorsieht. Durch die Verwendung von Ecore-Meta-Modellen lässt sich das Komponenten-Meta-Modell beliebig erweitern, so dass an die jeweiligen Anforderungen des Projektes angepasste Komponenten-Beschreibungssprachen verwendet werden können.

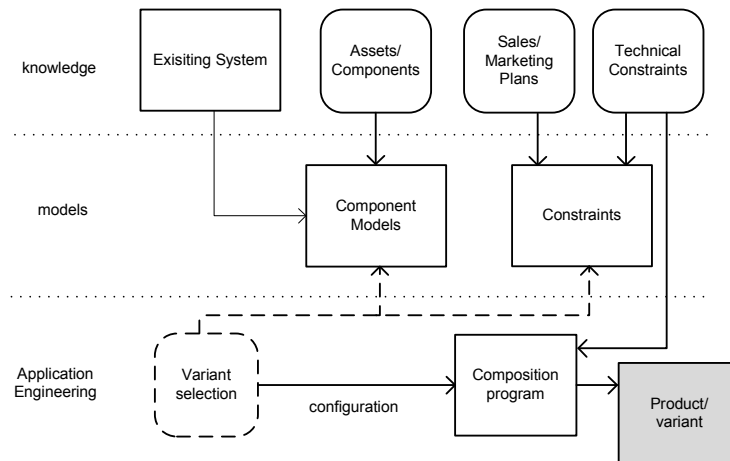


Abbildung 4: Umarbeitung eines existierenden Software-Systems

Hybrider Ansatz

Auf Grund von Budget-Planungen sowie unklaren Anforderungen bezüglich zu erwartendem Kundenkreis wird es in der Praxis vorkommen, dass neu zu erstellende Software-Systeme nicht von Beginn an als Produktlinien konzipiert werden, sondern erst später entsprechend den *Reverse-Engineering*-Ansätzen umgearbeitet werden müssen. Aus diesem Grund sieht der *hybride Ansatz* eine Integration beider Ansätze in dem Sinne vor, dass bei der Erstellung von Systemen stets die *Reverse-Engineering*-Konzepte von Anfang an zu verwenden sind. Das ermöglicht zum einen die Erstellung von komponentenbasierten, konfigurierbaren Software-Systemen, die Bereitstellung bzw. Generierung entsprechender Programmierschnittstellen und in die Eclipse-Entwicklungsumgebung integrierter Editoren. Zum anderen ermöglicht es, später auch die Ansätze des *Forward-Engineering* anwenden zu können, ohne die Struktur des Systems erneut unnötig zu ändern.

Betrachtet man die zu integrierenden Ansätze, so kann man feststellen, dass bei beiden für die Aufgabe der Beschreibung von Variabilitäten eines Systems verschiedene Modellierungssprachen eingesetzt werden, die über ähnliche Ausdrucksstärke verfügen und aufeinander abgebildet werden können. Zum einen dient das erweiterte Feature-Meta-Modell des *Forward-Engineering* als Grundlage für die Komponentenbeschreibungssprache. Zum anderen wird das Ecore-Meta-Meta-Modell im Rahmen des *Reverse-Engineering* zur Erweiterung des Komponenten-Meta-Modells verwendet.

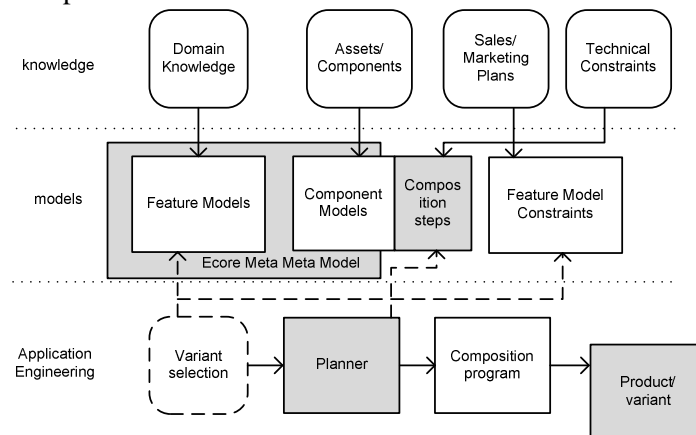


Abbildung 5: Angepasstes Konzept für den hybriden Ansatz

Um den Software-Entwicklern eine Einarbeitung in die Konzepte zu vereinfachen und unnötige Transformationen zwischen Modellen zu vermeiden, ist ein ganzheitlicher Ansatz unter Verwendung von so wenig wie möglich Technologien sinnvoll. Da es sich bei Ecore um ein in der Eclipse-Gemeinschaft weit verbreitetes Meta-Meta-Modell handelt und das im Rahmen der vorliegenden Arbeit vorgeschlagene Feature-Meta-Modell proprietär ist, wird Ecore als Grundlage für den *hybriden Ansatz* gewählt. Dementsprechend wird das Feature-Meta-Modell durch das Ecore-Meta-Meta-Modell ersetzt und es müssen alle Meta-Modelle des *Forward-Engineering* angepasst werden. Abbildung 5 stellt das Konzept des hybriden Ansatz als Anpassung der Abbildung 2 noch einmal dar.

Realisierung

Zur effizienten Realisierung der vorgestellten Konzepte wurde ein modellgetriebener, auf Generatoren basierender Ansatz gewählt. Dabei erfolgt die Implementierung auf Basis der in Abbildung 6 dargestellten Implementierungstechnologien. Die Grundlage bilden die Programmiersprache Java sowie das Eclipse RCP zur Erstellung beliebiger, plattformunabhängiger Anwendungen. Das Ecore-Meta-Meta-Modell wird zur Beschreibung der in den vorherigen Kapiteln entwickelten Meta-Modelle eingesetzt. Das FXL Framework bietet ein auf XML-Technologien basierendes Generator-Framwork. Darauf aufsetzend bietet der FXL Generator sowie der FXL Editor Generator die Möglichkeit, für Ecore-Meta-Modellen Programmierschnittstellen sowie baumbasierte Editoren als Erweiterung der Eclipse RCP zu generieren. Für die Realisierung der Konzepte des *Forward-Engineering* werden zudem die Web-Service-Programmiersprache SLL sowie dessen Ausführungsumgebung OpenXL eingesetzt. Die Konzepte des *Reverse-Engineering* können komplett durch Anpassung der mit Hilfe des FXL Editor Generators erzeugten Editoren realisiert werden.

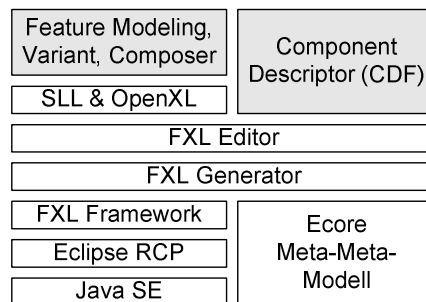


Abbildung 6: Implementierungstechnologien

Die Realisierung erfolgt für alle zu erstellenden Software-Werkzeuge analog zu dem in Abbildung 7 dargestellten Vorgehen. Mit Hilfe des FXL Generators wird für das jeweilige (Ecore-) Meta-Modell eine Java-Programmierschnittstelle generiert, mit dessen Hilfe Modelle erstellt, bearbeitet, geladen und gespeichert werden können. Der FXL Editor Generator ermöglicht die Generierung anpassbarer, baumbasierter Editoren, mit deren Hilfe die Modelle bearbeitet werden können. Zur Verwaltung mehrerer Modelle müssen verschiedene Artefakte von Hand implementiert werden. Dazu zählt beispielsweise ein Container zur Verwaltung mehrerer Feature-Modelle. Um die Editoren nutzbar zu machen, müssen sie durch Bereitstellung internationalisierter Texte sowie von Bildern angepasst werden.

Auf Grundlage dieses Konzeptes werden die Editoren für die Feature-Modellierung, die semantische Beschreibung der Kompositionsschritte, der Editor für die Variantenauswahl, der Komponenten-Editor sowie der Produktkonfigurations-Editor erstellt. Zusätzlich dazu

wird eine prototypische Implementierung der Planungskomponente basierend auf einem Planer-Meta-Modell erstellt.

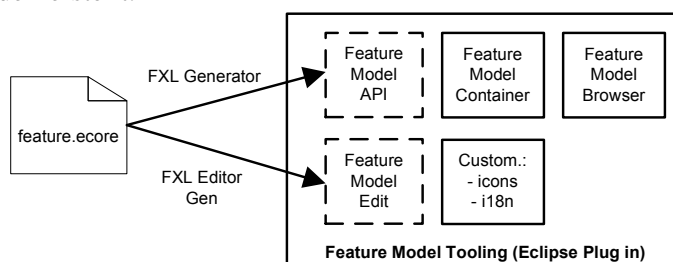


Abbildung 7: Realisierungskonzept der Editoren für die Feature-Modellierung

Evaluierung

Um die Konzepte zu evaluieren, wurden die erstellten Werkzeuge auf die Projekte aus dem Umfeld der Medizintechnik, angewandt.

Forward-Engineering

Die Konzepte sowie die dafür erstellten Software-Werkzeuge des *Forward-Engineering* wurden für das Beispiel der *Elektronischen Fallakte* prototypisch umgesetzt. Abbildung 8 zeigt eine vereinfachte Darstellung der vier dabei entstandenen Feature-Modelle. Der *ProblemSpace* beinhaltet die Variabilität aus der Sicht eines Mitarbeiters der Vertriebsabteilung bzw. eines Endkunden. Dazu gehören einfache Beschreibungen der Kernfunktionalitäten des Systems wie die Suche nach Patienten sowie die Möglichkeit, die Funktionalität der Informationssicherheit mit einem Schlüsselwort zu beschreiben. Die technische Sicht beinhaltet die Beschreibung der potentiell am System beteiligten technischen Komponenten sowie deren Parametrisierung. Dazu gehört zusätzlich zu den abgebildeten Services auch die Möglichkeit der Wahl der Persistierungsart (Datenbank, Dateisystem oder Mockup) für die Komponenten. Die eFA-Infrastruktur besteht aus der Wahl einer Datenbank sowie des zu verwendeten Applikationsservers inklusive deren im Rahmen von eFA notwendigen Parametrisierungen. Das EFA-Feature-Modell bestehend aus dem Feature *EFA* fasst die drei Untermodelle zusammen.

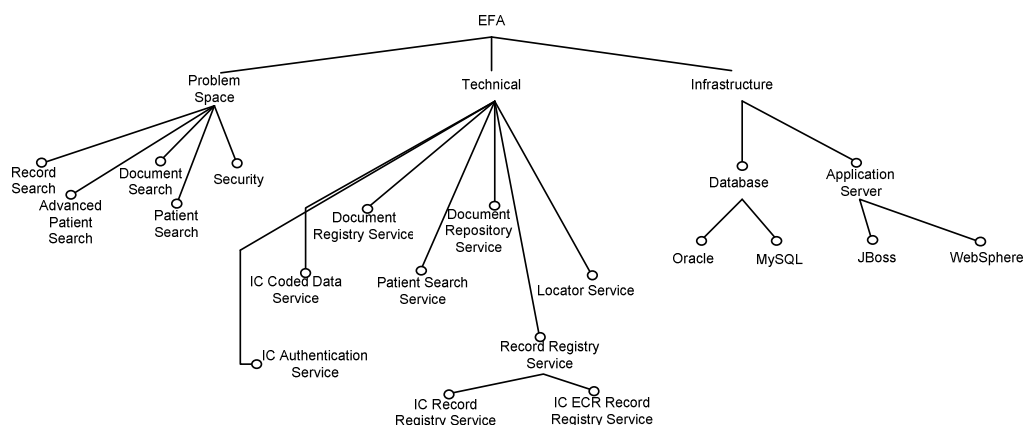


Abbildung 8: Erstellte eFA-Feature-Modelle (vereinfacht)

Für die Integration der technischen Komponenten in ein Produkt wurden entsprechende Kompositionsschritte samt semantischer Beschreibung bereitgestellt. Der Editor für die Variantenauswahl wurde entsprechend erzeugt. Damit ist es für einen Mitarbeiter der Vertriebsabteilung möglich, mit einem Endkunden auf einem sehr abstrakten Niveau das zu

erstellende System zusammenzustellen (vgl. Abbildung 9). Durch das integrierte Rollenkonzept können die Software-Entwickler das System später durch Nutzung der Feature-Modelle der technischen Sicht und der Infrastruktursicht beliebig verfeinern.

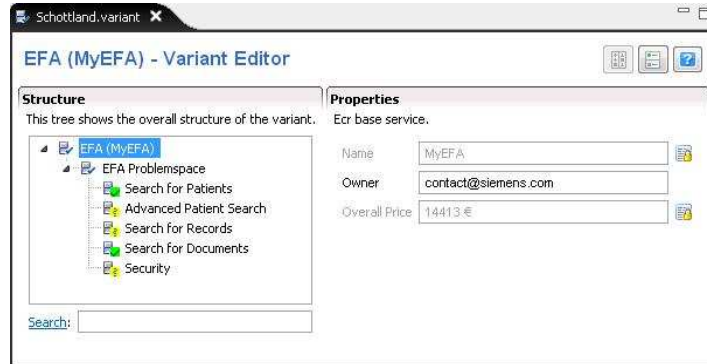


Abbildung 9: eFA-Variantauswahl für einen Mitarbeiter der Vertriebsabteilung

Ist eine Variante spezifiziert, kann mittels der automatischen Planungskomponente ein Kompositionsprogramm synthetisiert werden, was bei Ausführung durch die Workflow-Engine das gewünschte Produkt erstellt.

Reverse-Engineering

Für die Evaluierung der Ansätze des *Reverse-Engineering* wurden die Komponentenbeschreibungen für die (logischen) Soarian Integrated Care-Komponenten *Master Patient Index (MPI)* und *Electronic Health Record (EHR)* prototypisch implementiert und deren Variabilitäten mit Hilfe entsprechender Ecore-Meta-Modelle beschrieben. Um die Originalsoftware so wenig wie möglich zu modifizieren, wurden diese Komponenten in extra Verzeichnissen erstellt. Mit Hilfe des in den Komponenteneditor integrierten FXL-Generators wurden zu den Meta-Modellen gehörenden Java-Programmierschnittstellen und Editoren generiert.

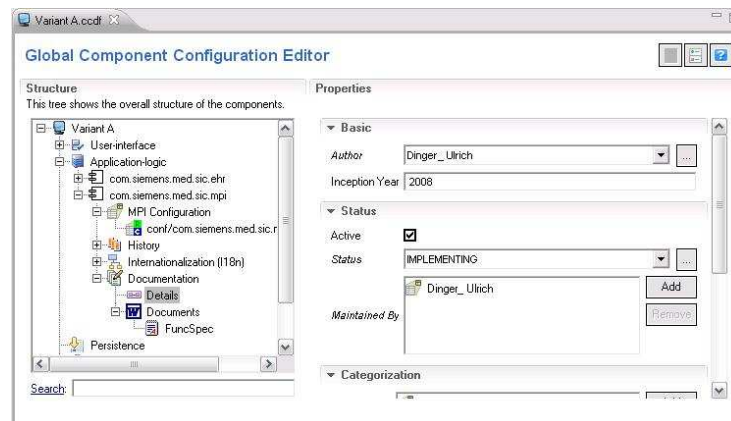


Abbildung 10: Produkt-Konfiguration von SIC (prototypisch)

Die neu erstellten Komponenten wurden in einem zweiten Schritt in den existierenden Build-Prozess eingebunden. Die Originalsoftware wurde unter Verwendung der generierten Programmierschnittstellen so angepasst, dass die Einstellungen aus den Komponentenmodellen geladen und in der Applikationslogik berücksichtigt werden. Diese Schritte führen zu der Möglichkeit, mit Hilfe des Produkt-Konfigurations-Editors (vgl. Abbildung 10) einen einheitlichen Überblick über alle auf diese Weise modellierten

Komponenten zu erhalten, deren Konfigurationen kundenspezifisch anzupassen und damit das Verhalten der Applikation zu steuern.

Zusammenfassung

Die vorliegende Arbeit hat sich mit der Erstellung von Software-Produktlinien nach dem Vorbild der Industrialisierung im Automobilbau beschäftigt. Im Rahmen der Untersuchungen zum Stand der Technik wurden die konzeptionelle Vorgehensweise verschiedener Ansätze evaluiert und damit einhergehende Probleme identifiziert. Da bei der Erstellung von Software-Produktlinien ohne existierende Altsysteme sowie bei der Umarbeitung existierender Systeme unterschiedliche Anforderungen existieren, wurden die vorgestellten Konzepte thematisch in das *Forward-* und *Reverse-Engineering* geteilt.

Die erste These der Arbeit besagte, dass der Einsatz einer automatischen Planungskomponente die im Rahmen des *Forward-Engineering* auftretenden Probleme lösen kann. Es wurde dafür ein erweitertes Feature-Meta-Modell beschrieben, das die Modellierung der gesamten Variabilitäten der Software-Produktlinie ermöglicht. Durch eine semantische Beschreibung der Arbeitsschritte, die bestimmte Komponenten in ein System integrieren, ist der Einsatz einer Planungskomponente möglich, die auf Basis einer Variantenauswahl ein dazu passendes Kompositionsprogramm synthetisieren kann. Die dafür benötigten Meta-Modelle zur Beschreibung der Kompositionsschritte sowie der Planungskomponente wurden vorgestellt. Das abgeänderte Konzept erlaubt durch Modellierung spezieller Änderungswünsche die Erstellung von angepassten Produkten, wobei die Vorteile der Synthetisierung der Kompositionsprogramme auch in diesem Fall genutzt werden können.

Die zweite These beinhaltete, dass der Einsatz einer einfachen, erweiterbaren Komponenten-Beschreibungssprache dabei hilft, existierende Systeme im Rahmen des *Reverse-Engineering* umzuarbeiten. Um dabei so wenig wie möglich Änderungen an einem existierenden System vornehmen zu müssen, wurde basierend auf dem einfachen, erweiterbaren Komponenten-Meta-Modell ein Ansatz vorgestellt, der das System um logische Komponenten erweitert und es erlaubt, dessen Konfigurationsmöglichkeiten mit Hilfe des Ecore-Meta-Meta-Modells als Erweiterungen des Komponenten-Meta-Modells zu beschreiben. Zusätzlich wurde eine Infrastruktur bereitgestellt, die die zentrale Speicherung von Produktkonfigurationen in Form von angepassten Komponentenmodellen erlaubt und die Zuweisung dieser Konfigurationen zum Zeitpunkt der Erstellung des Produkts oder zur Laufzeit ermöglicht.

Die dritte These besagte, dass eine Kombination beider Ansätze möglich ist und dazu führt, dass die Konzepte des *Reverse-Engineering* bereits bei der Erstellung von Software-Systemen angewandt werden können und diese später mit so wenig wie möglich Änderungen zu einer Software-Produktlinie erweitert werden können. Durch Untersuchung der Überschneidungen beim *Forward-* und *Reverse-Engineering* wurde im Kontext des hybriden Ansatzes die Entscheidung getroffen, auf Grund ähnlicher Mächtigkeit das Feature-Meta-Modell für die Modellierung von Variabilitäten durch das Ecore-Meta-Meta-Modell zu ersetzen und die erstellten Meta-Modelle des *Forward-Engineering* entsprechend anzupassen.

Die im Rahmen der Realisierung erstellten Software-Werkzeuge wurden auf Grundlage der vorgestellten Meta-Modelle und einem Framework zur Code- und Editor-Generierung als Erweiterungen der Eclipse-Entwicklungsumgebung realisiert. Während der Evaluierung wurden diese an zwei Beispielen aus dem Umfeld der Medizintechnik evaluiert und die Funktionstüchtigkeit der Konzepte und Werkzeuge nachgewiesen.