# Devolution

Marcel Jaeschke, Stephan Kauschka, Hinrich Harms
Fakultät für Informatik
Otto-von-Guericke Universität Magdeburg
marcel.jaeschke@st.ovgu.de, {kauschka, hharms}@cs.uni-magdeburg.de

10. März 2008

## 1 Einführung

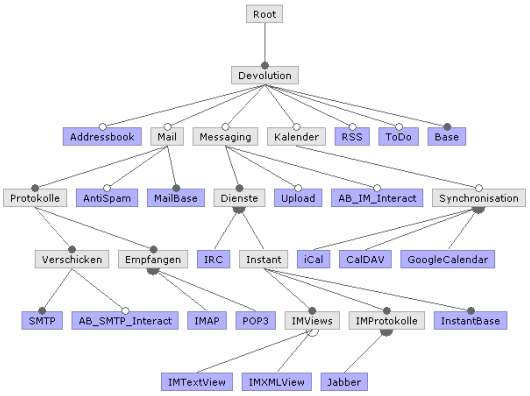


Abbildung 1: Das Featurediagramm von Devolution

Im Rahmen der Vorlesung Erweiterte Programmierkonzepte für maßgeschneiderte Datenhaltung im Wintersemester 2007/2008 sollte ein Softwareprojekt zum Thema der Vorlesung erstellt werden. Wir haben uns für die featureorientierte Entwicklung einer Softwareproduktlinie für ein Kommunikationszentrum entschieden. Dazu wurde ein Basisprogramm namens Devolution geschrieben, welches durch zusätzliche Features erweitert werden kann. Nach einer Domainanalyse wurde das obige Featurediagramm entwickelt (siehe Abbildung 1). Innerhalb dieser Projektarbeit wurden die zusätzlichen Features Adressbuch, Instant Messaging und E-Mail umgesetzt. Der Code wurde in Java mithilfe der Entwicklungsumgebung FeatureIDE geschrieben.

## 2 Devolution

Im folgenden Abschnitt soll die Struktur und das Konzept der Basis erläutert werden.
Devolution kann als Kommunikationszentrum angesehen werden. Dabei wurden verschiedene Kommunikationswege und deren Interaktion beziehungsweise die zentrale Verwaltung der Wissensbasis berücksichtigt, welche alle als Module bezeichnet und realisiert wurden.

### 2.1 Aufbau

Jedes Modul ist eine Kindklasse von der abstrakten Klasse 'Modul'. Diese Klasse definiert die benötigten Methoden zur Kommunikation mit der Basis. Modulabhängige Methoden werden nicht von der Basis verwendet, sondern werden nur für die modulinterne bzw. intermodulare Interaktion benötigt.
Dadurch ist es möglich in der Basis von Devolution eine Liste von Modulen zu verwalten. Da Listen wiederum iterierbar sind, ist es nicht erforderlich eine Vielzahl an Methoden zu verfeinern, sondern nur eine Methodenfolge auf jedes Element der genannten Liste anzuwenden um neue Module (die die Hauptfeatures repräsentieren) anzumelden.
Um die GUI von der eigentlichen Funktionsweise der Module zu trennen, verwenden wir ein ähnliches Prinzip. Jedes Modul hat eine Klasse welches die GUI des Moduls organisiert. Diese GUI-Klasse ist eine Kindklasse von 'ModulView'. Die GUI von Devolution interagiert nur mit diesen Kindklassen. Durch diese Organisation werden die Belange der Module gekapselt.

### 2.2 Interfaces vs. abstrakte Klassen

Bei der Umsetzung der Module haben wir uns gegen Interfaces entscheiden da abstrakte Klassen den Vorteil haben für triviale Methoden (Getters und Setters) den Methodenkörper zu definieren. Dadurch wird der modulspezifische Quellcode kleiner und übersichtlicher. Weiterhin ist sichergestellt das diese Funktionen bei jedem Modul denselben Algorithmen folgen.
Sollte ein Feature alle Module betreffen so ist es nur erforderlich die abstrakte Klasse zu Verfeinern. Ist die Methode stark modulabhängig, also auch abstrakt, so gibt es für den Aufwand kein Vorteil, denn dieser liegt allein darin, dass der Code sauber gehalten wird.
Ein weiteres Szenario wäre die Implementierung eines neuen Moduls. Hierbei ist zwar der Erstaufwand sehr hoch, da alle Modulmethoden geschrieben werden müssen, andererseits ist klar definiert wo was passieren muss.

## 3 Adressbuch

Das Adressbuch-Modul ist eine sehr einfach gehaltene Komponente und stellt alleine keine Besonderheit da. Sie sichert nur bestimmte Daten welche von anderen Modulen benutzt werden können. Um Anfragen zu reduzieren liefert die Komponente selber keine Informationen sondern Objekte der Klasse Buddy, welche wiederum im

Lazy-Verfahren den Datenbestand des Adressbuches abfragt.
Buddies können nicht direkt angefordert werden, d.h. man kann nur eine Liste aller bekannten Kontakte anfordern, welche eine bestimmte Eigenschaft haben (z.B. der Vorname lautet Jochen). Egal wie das Ergebnis ausfällt folgt auf der Anfrage das Ergebnis in Form einer Liste. Wie die Liste verarbeitet wird, liegt nicht mehr im Aufgabenbereich des Adressbuches. Das Ansteuern der Eigenschaften der Kontakte geschieht über Konstanten der Klasse Buddy, dadurch ist eine methodenarme Implementierung möglich gewesen.

### 3.1 Probleme

Leider ist es mit der gegenwärtigen Version von FeatureIDE noch nicht möglich mit ENUM zu arbeiten. Dadurch können undefinierte Konstanten verwendet werden, was zu einem Fehlverhalten der Software mit eventuellem Datenverlust führen kann.

## 4 Instant Messaging

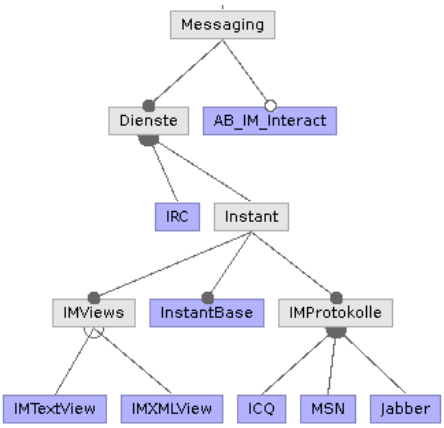


Abbildung 2: Ausschnitt aus dem Featurebaum mit dem Messaging-Teil

In diesem Teil von Devolution geht es um den Austausch von 'Sofortnachrichten' (Instant Messages). Dazu wurde ein weiteres Modul in der Devolution-Basis registriert. Das Modul an sich wurde wiederum in mehrere Unterfeatures unterteilt (siehe Abbildung 2), um eine einfache Erweiterung um zusätzliche Funktionen in Form von weiteren Features zu ermöglichen.

### 4.1 Aufteilung

Die Grundfunktionalität wurde dabei in dem Feature 'InstantBase' realisiert. Die Komponente zur Anzeige von Chatsitzungen ist dabei in ein eigenes Feature ausgelagert, sodass neben einer einfachen Textanzeige der Nachrichten auch andere Möglichkeiten denkbar sind, wie z.B. eine grafische Anzeige, die die Nachrichten mittels XML und XSLT darstellt. Weiterhin ist das Konzept so ausgelegt, dass jedes beliebige Protokoll in einem extra Feature implementiert werden kann. In diesem Projekt wurde als Beispielprotokoll XMPP (das Jabber-Protokoll) implementiert.

### 4.2 Instant Messaging Basis

Wie bei allen Modulen wurde zunächst die Hauptklasse 'Devolution' verfeinert, um ihr das Instant Messaging bekannt zu machen, sodass Menüeinträge und ein Einstellungsdialog erzeugt werden können. Die meisten der Einträge werden allerdings erst durch die Protokollfeatures erstellt, da sie zum größten Teil spezifisch sind, wie z.B. der Benutzername und das Passwort.
Zur Anzeige der Kontaktliste wurde im Sidepanel ein JTree erstellt, dessen Knoten von den jeweiligen Protokollen erstellt werden. Dazu wird den Protokollen die Funktion 'createRosterTree' angeboten, die verfeinert werden kann. Zur Repräsentation der Einträge gibt es die Klasse 'IMBuddy', welche wiederum von den Protokollen bei Bedarf um zusätzliche Funktionen erweitert werden kann. Die aktuell geöffneten Sitzungen werden über 'IMTabs' verwaltet.

### 4.3 Anzeigefeature für Chatsitzungen

Die eigentliche Anzeige der Nachrichten übernimmt die ChatPanel-Klasse im Feature IMTextView. Sie bietet der IM Basis eine Funktion 'addMessage' an, sodass die Anzeige einfach ausgetauscht werden kann (durch ein anderes Feature ersetzt werden kann), ohne dass Änderungen im restlichen Quellcode nötig sind.

### 4.4 Jabber

Als Beispielprotokoll wurde XMPP implementiert. Es wird die aktuelle Kontaktliste vom Server abgerufen und in Devolution angezeigt und man kann natürlich sowohl Nachrichten verschicken als auch empfangen.
Durch Verfeinern der Funktion 'createRosterTree' werden ausgehend von der Wurzel der Kontaktliste unter dem Eintrag 'Jabber' die Kontakte hinzugefügt. Wenn sich die Kontaktliste ändert, z.B. wenn ein Benutzer sich an- oder abmeldet, wird die Kontaktliste neu aufgebaut und die Basis benachrichtigt, den Baum neu zu zeichnen.
Weiterhin wurde die Klasse 'IMBuddy' verfeinert, um Jabber typische Felder, wie die Jabber-ID und eine eventuell offene Chatsitzung, zu speichern. Durch das Konzept der featureorientierten Programmierung ist es ohne Weiteres möglich, dass verschiedene Protokolle die Klasse um veschiedene Merkmale erweitern und man immer noch alle Features unabhängig voneinander an- und abwählen kann.

### 4.5 Observer Pattern

Mit dem Hintergrund der ereignisorientierten Programmierung wurden viele Probleme durch das Observer Pattern gelöst. So sind die Klassen 'JabberListener' und 'JabberRoster' Listener, die benachrichtigt werden, wenn eine neue Nachricht eintrifft bzw. sich an der Kontaktliste etwas ändert. Weiterhin bietet die Klasse 'JabberRoster' die Möglichkeit, von weiteren Listenern beobachtet zu werden, wenn sich etwas an der Kontaktliste ändert. So bekommt die Anzeige des JTrees für die Kontaktliste mit, das sich etwas geändert hat und zeichnet den Baum neu.

## 5 E-Mails

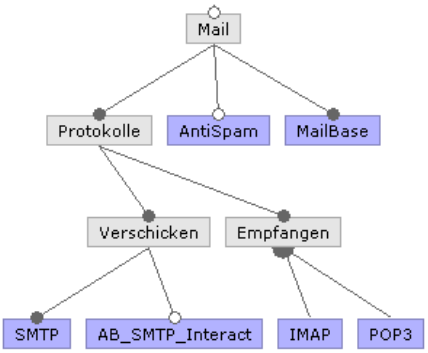


Abbildung 3: E-Mail relevanter Ausschnitt aus dem Featurebaum

Hier wird das Senden und Empfangen von E-Mails mittels Devolution behandelt. Wiederum wurde dazu ein weiteres Modul in der Devolution-Basis registriert. Auch bei diesem Feature hat es sich angeboten mehrere Unterfeatures zu erstellen (siehe Abbildung 3), von denen einige optional, andere obligatoriach sind.

### 5.1 Aufteilung

Das Mail Feature besteht aus den Unterfeatures MailBase, AntiSpam, SMTP, POP3, IMAP und einem Interaktionsfeature, auf das später eingegangen wird. Die Grundfunktionalität wird in MailBase implementiert, weswegen das Feature obligatorisch ist. Auch das Feature SMTP zum Verschicken von E-Mails ist obligatorisch, weil es zu den grundlegenden Funktionen eines Mailprogramms gehört. Trotzdem wurde dafür ein eigenes Feature eingerichtet, da es ein eigener Belang ist und somit modularisiert werden sollte. Die Protokolle zum Empfangen von E-Mails sind dagegen optional, allerdings muss mindestens eins ausgewählt werden damit die Möglichkeit zum Empfangen garantiert ist. Auf das Implementieren des AntiSpam Features wurde aus Zeitgründen verzichtet.

### 5.2 MailBase

Wie bereits erwähnt wird die Basisfunktionalität des Mail Features hier implementiert. Zu dieser Funktionalität gehört einerseits das Anlegen von Accounts, auf die dann die einzelnen Sende- und Empfangsprotokolle zurückgreifen. Andererseits gehören auch die ganzen GUI-Komponenten die zum Navigieren innerhalb des Mail-Moduls nötig sind dazu. Weiterhin wurde hier die GUI zum Empfangen von Mails implementiert, da diese für alle Protokolle gleich ist. Dieses GUI greift dabei auf ein Receiver-Interface zurück, das sowohl POP3 als auch IMAP implementieren müssen.

### 5.3 SMTP

Das Verschicken von Mails wurde im SMTP Feature umgesetzt. Das Sendeprotokoll übernimmt der SMTPTransmitter und die GUI die MailWritePane. Diese wurde zur besseren Modularisierung erst hier und nicht schon im MailBase Feature implementiert, was jedoch kein Problem darstellt, da dieses Feature obligatorisch ist.

### 5.4 POP3 und IMAP

Das Empfangen von E-Mails kann entweder durch POP3 oder IMAP geschehen. Innerhalb der entsprechenden Features werden nur die Protokolle umgesetzt und die Basis-GUI verfeinert, sodass beim Erstellen von Accounts das jeweilige Protokoll auch ausgewählt werden kann.

## 6 Feature-Interaktion

Innerhalb von Devolution gibt es auch die Möglichkeit, dass mehrere Features untereinander interagieren. So können sowohl der Instant Messager als auch das E-Mail Programm auf das Adressbuch zurückgreifen. Da diese Interaktion aber nur durch die gewählte Featureaufteilung nötig und somit implementierungsabhängig ist, konnte sie herausgelöst und in eigenen Modulen implementiert werden.

### 6.1 Adressbuch und Instant Messaging

Wie bereits erwähnt können die beiden Features Adressbuch und Instant Messaging interagieren - der Instant Messenger befragt dabei das Adressbuch, ob schon ein Eintrag mit der Jabber-ID vorhanden ist. Wenn das der Fall ist, wird für die Anzeige der Kontaktliste der Nickname aus dem Adressbuch benutzt.
Die Interaktion wurde in dem eigenen Feature 'AB_IM_Interact' modelliert. Das ermöglicht eine hohe Flexibilität, da nicht nur Adressbuch und Instant Messenger unabhängig voneinander ausgewählt werden können, sondern man kann auch noch entscheiden, ob, wenn beide Features ausgewählt sind, noch die Interaktion zwischen den beiden Features ins Projekt

integriert werden soll oder nicht.

### 6.2 Adressbuch und E-Mails

Auch das E-Mail Feature ist fähig mit dem Adressbuch zu interagieren. So kann beim Verschicken von Mails der Empfänger aus dem Adressbuch bestimmt werden. Dazu wird, sobald man in das entsprechende Feld den Empfänger eintippt, der Text mit den Adressen und Namen im Adressbuch verglichen. Alle Treffer werden dann dem Nutzer angezeigt und die E-Mail Adressen als mögliche Empfänger zur Verfügung gestellt.
Die Interaktion wurde in dem eigenen Modul 'AB_Mail_Interact' implementiert und kann dann ausgewählt werden, wenn sowohl das Feature E-Mail als auch das Feature Adressbuch gewählt wurden. E-Mail und Adressbuch interagieren dabei nur statisch, d.h. der Code vom Adressbuch wird nur referenziert und die zeitliche Abfolge spielt bei der Interaktion keine Rolle.
Anzumerken ist, dass zur Umsetzung dieses Features die MailView und die MailWritePane aus dem Basisfeature verfeinert wurden. Da dieses Feature vorher jedoch nicht vorausgesehen wurde, konnte die actionPerformed() Methode nicht einfach verfeinert werden, sondern musste komplett überschrieben werden, da mittels FeatureIDE keiner Verfeinerung innerhalb einer Methode möglich ist. Die dadurch entstehende Codereplikation hätte mittels AspectJ vermieden werden können.

### 6.3 Schlussfolgerung zur Feature-Interaktion

Durch das Herauslösen der Interaktion wurde einerseits die Flexibilität bei der Featurekomposition erhöht, da die interagierenden Module nun nicht immer zusammen ausgewählt werden müssen. Andererseits wurde so das Featuretraceability Problem gelöst, weil die Interaktion nun genau lokalisierbar ist.
Da innerhalb dieses Projektes allerdings nur zwei Interaktionsfeatures herausgelöst wurden, konnten keine negativen Begleiterscheinungen beobachtet werden. So würde bei größeren Projekten die Anzahl der Interaktionen sicherlich höher sein und somit auch die Komplexität beim Herauslösen der Features und der anschließenden Featureselektion viel größer werden. Hinzukommt, dass hier nur Interaktionen zwischen zwei Features betrachtet wurden und deshalb immer nur ein neues Modul entstand, wodurch die Komplexität auch handhabbar blieb. Für kleine Projekte ist dieses Vorgehen daher noch praktikabel und zu empfehlen.

## 7 Fazit

Das Projekt 'Devolution' mit seinen Komponenten ist ein gutes Beispiel zur featureorientierten Programmierung. Da die meisten Erweiterungen antizipiert wurden, konnte ein Großteil des Codes ohne die durch FeatureIDE bereitgestellten Sprachkonstrukte umgesetzt werden. Bei 6697 lines of code (LOC) im gesamten Projekt waren nur 427 Refinements. Und von diesen hätte man nur 22 LOC eingespart, wenn Devolution aspektorientiert implementiert worden wäre. Diese Zahlen bestätigen die bereits in der Vorlesung vermutete Äußerung, dass der Hauptteil der Verfeinerungen mittels FOP umgesetzt werden kann und dass man, wenn bestimmte Punkte für Erweiterungen vorgesehen sind, diese relativ einfach umsetzen kann.
Außerdem hat sich durch dieses Vorausplanen ein weiterer Effekt gezeigt. Da zum Anmelden der einzelnen Module im Hauptprogramm eine Hashfunktion benutzt wurde, ist es völlig egal in welcher Reihenfolge die jeweiligen Features ausgewählt werden. Die entsthende GUI wird in jedem Fall gleich aussehen, d.h. alle Features befinden sich immer am selben Platz.
Zum Schluss ist noch zu erwähnen, dass durch die gute Modularisierung der Features diese, sobald der Basiscode vorlag, sehr gut getrennt voneinander entwickelt werden konnten.

## 8 Eine kurze Kritik

Bei diesem Projekt wurde die Funktionalität von FeatureIDE nur sporadisch eingesetzt. Durch das Vorausplanen der meisten Erweiterungen wurde es häufig nur benutzt um bestimmte Features zu selektieren, was aber auch ohne FeatureIDE realisierbar gewesen wäre. Der betriebene extra Aufwand durch dieses Tool lohnte sich daher kaum. Die Ansammlung von verschiedenen Dateien, zwei JAK- und eine JAVA-Datei, führte oft zu Verwirrungen. Das späte, fehlende oder fehlerhafte Anzeigen von Fehlern stellte eine Last dar. Auffällig war auch das Fehlen von Generics, ENUM und der speziellen for-Schleife. Letzteres könnte, wie es beim Compiler geschieht, automatisiert werden. Das Packages fehlen ist ein weiterer Minuspunkt. Alle Dateien in einem Ordner ist eine unschöne Lösung. Klassen in unterschiedlichen Belangen mit gleichen Namen werden so künstlich verboten.
Der Stand von JAVA 1.4 ist nicht ohne Grund veraltet. Entwickler sind unnötig wieder mit alten Problemen beschäftigt. FeatureIDE könnte Punkten wenn der Editor einen vergleichbaren Funtionsraum bieten würde wie der Java-Editor. Die Autovervollständigung und vorzeitige Fehlermeldungen fehlen bei der Entwicklung und führen zur Verunsicherung. Natürlich sind das größtenteils Eigenschaften die durch AHEAD bedingt sind, letztendlich bleibt aber ein eher unkomfortabler Eindruck.