Desktop Searcher - eine Software-Produktlinie für die Suchengine Lucene

Sebastian Bress, Alexander Grebhahn, Sönke Holthusen und Reimar Schröter
Fakultät für Informatik
Otto-von-Guericke-Universität (OvGU) Magdeburg
{*sebastian.bress, alexander.grebhahn, soenke.holthusen, reimar.schroeter*}*@st.ovgu.de*

I. Einleitung

In der Softwareentwicklung ist es oft nötig ein bestehendes Programm auf die Wünsche von mehreren, verschiedenen Kunden maßzuschneidern. Die dafür notwendigen Anpassungen sind auf herkömmlichen Prinzipien (beispielsweise Objektorientierte Programmierung (OOP)) sehr kostenintensiv und schwer wartbar. Eine geeignete Technologie um diese Probleme zu entgehen, ist die Generierung mittels Softwareproduktlinien (SPL). Das Prinzip der SPL ermöglicht eine Erstellung von verschiedenen Programmausprägungen anhand einer Basisimplementierung und einer Auswahl von verschiedenen Funktionen (sogenannten Features). Im Rahmen eines Projektes der Lehrveranstaltung „Erweiterte Programmierkonzepte für maßgeschneiderte Datenhaltung“ (EPMD) der Otto-von-Guericke-Universität Magdeburg wurde eine bestehende Software zu einer SPL erweitert. In diesem Zusammenhang wurden Möglichkeiten zur Analyse und Entwicklung einer SPL untersucht. Neuere Ansätze, wie zum Beispiel Aspektorientierte Programmierung (AOP) und Featureorientierte Programmierung (FOP), versprechen dabei eine bessere Unterstützung zur Erstellung von SPL, als etablierte Technologien wie Frameworks und Präprozessoren.

In dieser Arbeit erfolgt eine Analyse der Ansätze AOP und FOP im Hinblick auf ihre Tauglichkeit zur Erweiterung und Modularisierung eines bestehenden Programms. Dabei handelt es sich um eine Applikation, mit der Dateien indexiert werden können, um dann in ihnen zu Suchen. Im Grundlagenkapitel wird dazu das bestehende Programm vorgestellt. Des Weiteren wird besprochen, welche Funktionalität in der SPL existieren soll. Nachdem dies geschehen ist, werden im Anforderungsanalyse- & Implementierungskapitel die Ansätze im Bezug auf ihre Tauglichkeit zur Umsetzung der beschriebenen SPL analysiert. Außerdem wird auf Probleme, die während der Implementierung auftraten eingegangen. Abschließend wird im Evaluierungskapitel auf die Anzahl der möglichen Varianten der SPL eingegangen.

II. Grundlagen

A. Basisimplementierung

Als Grundlage der SPL, wurde ein Programm verwendet, dass im Rahmen der Lehrveranstaltung „Information Retrieval“ implementiert wurde. Es dient zur Indexierung von Dateien und Dateiinhalten auf Grundlage der Information Retrieval Software Bibliothek „Lucene“[1]. Die Erweiterung stellte durch eine grafische Oberfläche eine Suchfunktionalität zur Verfügung. In Abbildung 1 ist der generelle Aufbau des ursprünglichen Programms abgebildet. Grundlegend besteht das Programm aus drei Komponenten, den Content-Handlern, dem Indexer und der GUI. Die Hauptfunktionalität des Indexer wird durch die Klassenbibliothek Lucene bereitgestellt. Die Dateiinhalte, nach denen gesucht werden kann, werden dem Indexer über verschiedene ContentHandler übergeben. Zur Kommunikation zwischen Lucene und dem Benutzer dient die Grafische Oberfläche.

Damit eventuelle Bedürfnisse von potentiellen Benutzern besser entsprochen werden kann, ist eine größtmögliche Variabilität wünschenswert.

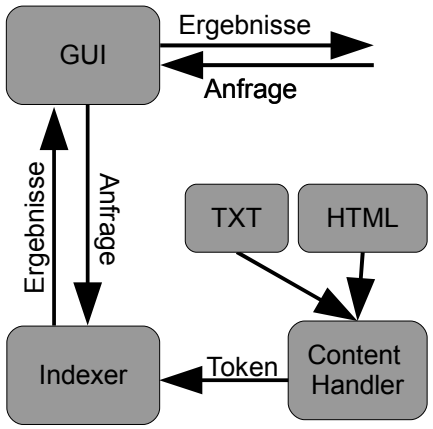

Abbildung 1. Architektur des ursprünglichen Programmes.

B. FOP vs AOP

Die nachfolgenden Ausführungen wurden, wenn nicht anders angegeben, den Folien der Vorlesung EPMD[2] bzw. aus [1], [2], [3] und [4] entnommen. Die beiden Programmierkonzepte FOP und AOP zur Erzeugung einer SPL besitzen Vor- und Nachteile bezüglich verschiedenartiger Code-Erweiterungen. Zum besseren Verständnis, werden zunächst homogene und heterogene, sowie statische und dynamische

[1] http://lucene.apache.org/java/docs/index.html
[2] http://wwwiti.cs.uni-magdeburg.de/iti_db/lehre/epmd/2010/

Erweiterungen definiert. Diese werden dabei auf ihre Unterstützung durch FOP und AOP untersucht.

Heterogene Erweiterungen fügen unterschiedlichen Code an unterschiedlichen Stellen hinzu. Sie werden von FOP gut unterstützt, da sie explizit durch Verfeinerungen und Kollaborationen umgesetzt werden können. Im Gegensatz dazu können mittels AOP keine expliziten Kollaborationen durchgeführt werden. Eine heterogene Erweiterung mittels AOP ist dennoch möglch.

Homogene Erweiterungen verändern ein Programm an verschiedenen Stellen um den selben Code. Dies führt bei FOP zu Codereplikationen, die sich negativ auf die Softwarequalität auswirken. Durch die Möglichkeit der Quantifizierung von Join-Points durch Pointcuts in AOP werden homogene Programmerweiterungen unterstützt.

Statische Erweiterungen verändern die statische (syntaktische) Struktur eines Programms. Beispiele hierfür sind das Hinzufügen von Methoden und Feldern zu einer Klasse. Bei der FOP können nicht nur bestehende Klassen um Methoden und Felder erweitert werden, es können auch neue Klassen hinzugefügt werden. FOP unterstützt damit statische Erweiterungen gut. Mit AOP ist es nur möglich bestehende Klassen um Felder und Methoden zu erweitern. Damit ist die Unterstützung der statischen Erweiterungen eingeschränkt.

	FOP	AOP
heterogen	gute Unterstützung, Verfeinerungen und Kollaborationen	eingeschränkte Unterstützung keine expliziten Kollaborationen
homogen	Keine Unterstützung, Eine Verfeinerung pro Join-Point (Code-Replikation)	gute Unterstützung-Wildcards und logische Verknüpfungen von Pointcuts
statisch	gute Unterstützung-Attribute, Methoden, Klassen	eingeschränkte Unterstützung-Attribute, Methoden
dynamisch	schlechte Unterstützung-einfach dynamisch (Erweiterung von Methoden)	gute Unterstützung- komplex dynamisch

Tabelle I
VERGLEICH FOP AOP.

Dynamische Erweiterungen verändern das Verhalten existierender Bestandteile eines Programms. Dabei wird auf bestimmte Ereignisse im Programmablauf gesondert reagiert. Generell wird zwischen einfachen und komplexen dynamischen Erweiterungen unterschieden. Bei einfachen dynamischen Erweiterungen können Methodenausführungen verändert werden. Es können jedoch keine Laufzeitbedingungen angegeben werden. Des Weiteren kann auf den Kontext von Ereignissen nur eingeschränkt zugegriffen werden. Wie beispielsweise auf die Argumente, den Rückgabewert und das Objekt selbst. Im Gegensatz dazu können komplexe dynamische Erweiterungen auf alle Arten von Programmereignissen reagieren. Zusätzlich können Laufzeitbedingungen angegeben und somit auf den kompletten dynamischen Kontext zugegriffen werden. Die einfachen dynamischen Erweiterungen werden von FOP und AOP unterstützt. Die komplexen dynamischen Erweiterungen sind in FOP nicht möglich, bei AOP werden sie hervorragend unterstützt.

In Tabelle I wird ein Überblick über die unterstützten Erweiterungen gegeben.

C. Angestrebte Funktionalität

Damit die SPL erstellt werden kann, wurde die bestehende Funktionalität des Programms analysiert und dabei in verschiedene Teilfunktionalitäten gegliedert. Außerdem wurde überlegt, welche zusätzlichen Funktionen für eventuelle Benutzer von Interesse sein könnten. Dies führt zu dem Feature-Modell aus Abbildung 2.

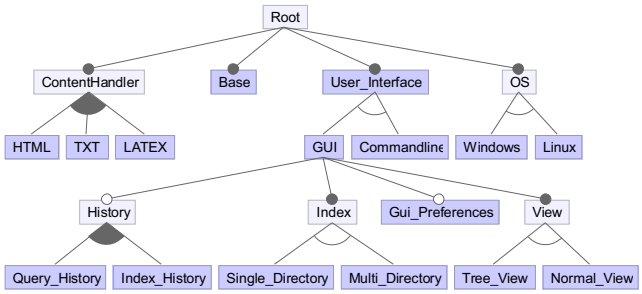

Abbildung 2. Feature-Modell des Desktop Searchers.

Zunächst soll es mit dem Programm möglich sein, nicht nur .txt und .html Dateien zu indexieren, sondern auch .tex Dateien. Diese neue zusätzliche Funktionalität ist im Feature `LATEX` zusammengefasst. Da die Funktionalität der anderen `ContentHandler` optional sein soll, ist für sie jeweils ein eigenes Feature vorgesehen. Des Weiteren sollte die Möglichkeit bestehen, anstelle einer `GUI` für die Benutzung der Suchengine ein Consoleninterface zu verwenden. Dieses zusätzliche Feature wird im Featurebaum als `Commandline` bezeichnet. Ferner soll es die Möglichkeit geben mehrere Lokale Ordner mit ihren Unterordnern zu indexieren. Da es unter Umständen zu restriktiv währe die Indexierung auf einen Ordner zu beschränken. Deswegen wird diese Funktioanlität durch ein zusätzliches optinales Feature bereitgestellt. Dieses wird im nachfolgenden `Multi_Directory` genannt. Außerdem soll dem Benutzer die Möglichkeit gegeben werden, nicht nur nach Wörtern in Dateien zu suchen, sondern auch nach den größten beziehungsweise den ältesten Dateien. Diese Funktionalität soll sowohl in der `GUI`, als auch in der `Commandline` vorhanden sein. Dabei soll diese Erweiterung in der `Commandline` immer vorhanden sein, während

sie in der GUI extra ausgewählt werden kann. Damit die Benutzbarkeit des Programms erhöht wird, soll die Möglichkeit bestehen, dass automatisch eine History über eingegebene Anfragen, als auch über die indexierten Ordner angelegt wird. Im Abbildung 2 wird diese Funktionalität durch den Teilbaum History dargestellt. Des Weiteren sollen zwei verschiedene Arten der Ergebnisdarstellung innerhalb der GUI existieren. Dabei soll es eine Ansicht geben, in der die gefundenen Ergebnisse, nach der Anzahl der gefundenen Terme in ihr sortiert sind und eine, in der die gefundenen Dateien nach ihrem Ort auf der Festplatte sortiert sind. Damit bei der Anzeige auch auf Besonderheiten von verschiedenen Betriebssystemen geachtet werden kann, werden durch die Features Windows und Linux Betriebssystemenbesonderheiten betrachtet.

III. Anforderungsanalyse & Implementierung

Die Entwicklung einer SPL erfordert eine genaue Analyse des zu erweiternden Programms. Dabei müssen die Vorteile einer Implementierung mittels FOP und AOP voneinander abgewägt werden. In diesem Zusammenhang erfolgte zur Auswahl der geeigneten Technik eine Analyse der einzelnen zu implementierenden Features. Die einzelnen Features wurden nach Art ihrer vermutlich verwendeten Erweiterungen eingeteilt. Im Rahmen dieser groben Schätzung ergibt sich Tabelle II. Nach dieser Kalkulation handelt es sich bei den zu erwartenden Anpassungen hauptsächlich um homogene, statische und einfach dynamische Erweiterungen. Wie im Grundlagenkapitel ersichtlich wurde, besitzt die AOP enorme Vorteile bei der Nutzung von homogenen und komplex dynamischen Erweiterungen. Diese Art von Erweiterungen werden in dieser SPL jedoch nicht benötigt. Im Bereich der heterogenen Erweiterungen, die in dieser Projekt vor allem zu erwarten sind, besitzt FOP einige Vorteile. Vor allem die Übersichtlichkeit des Quellcodes bleibt bei dieser Art der Realisierung in FOP erhalten. Im Bereich der statischen und einfach dynamischen Erweiterungen sind keine großen Vorteile von FOP und AOP bezüglich dieser SPL vorhanden. Demnach besitzt Featureorientierte Programmierung zumindest bei der Übersichtlichkeit einige Vorteile, die zur Entscheidung der Nutzung von FOP bei der Implementierung der SPL führte.

	heterogen	homogen	statisch	einfach dynamisch	komplex dynamisch
LATEX	✓	–	✓	✓	–
TXT	✓	–	✓	✓	–
HTML	✓	–	✓	✓	–
Tree_View	✓	–	✓	✓	–
Normal_View	✓	–	✓	✓	–
GUI	✓	–	✓	✓	–
Console	✓	–	✓	✓	–
Single_Directory	✓	–	✓	✓	–
Multi_Directory	✓	–	✓	✓	–
Query_History	✓	–	✓	✓	–
Index_History	✓	–	✓	✓	–
Windows	✓	–	✓	–	–
Linux	✓	–	✓	–	–

Legende:

✓ zutreffend
– nicht zutreffend

Tabelle II
Zu erwartende Erweiterungen.

Nachdem FOP zur Implementierung der SPL ausgewählt wurde, erfolgte die Erstellung einer Codebasis. Diese Version wird in FOP zur Grundlage der Erstellung einer SPL benötigt. Darüber hinaus bestehen bei der Umsetzung einer SPL in FOP einige Probleme, die beachtet werden mussten. Zum einen existiert das sogenannte Preplanning Problem. Demnach können Features nicht immer unabhängig vom bestehenden Programm eingefügt werden. Im Fall von FOP muss beispielsweise die Struktur der Basisversion jegliches, diese Version erweiterndes, Feature unterstützen. Aus diesem Grund war eine gründliche Vorausplanung mit allen Teammitgliedern nötig. Jedem Teammitglied wurden anschließen die zu implementierenden Features zugewiesen. Da nicht alle Features unabhängig voneinander sind und miteinander interagieren, wurde versucht eine sinnvolle Aufteilung zu finden. Zur Implementierung miteinander interagierende Features, wie beispielsweise GUI_Preferences und Multi_Directory, war eine Kommunikation zwischen den Teammitgliedern unumgänglich. Im optimalen Fall wurden solche Features von einer Person implementiert.

Zum Teil war es bei der Implementierung notwendig, sogenannte Hook- Methoden einzufügen. Wenn beispielsweise mitten in einer Methode eine Erweiterung notwendig ist, wurden Methoden hinzugefügt, um einen Einstiegspunkt für die Funktionalitätserweiterung bereitzustellen. Diese Methoden können anschließend von den einzelnen Features verwendet und beispielsweise überschrieben werden. Vorallem aus diesem Grund ist das Preplanning für die Implementierung einer SPL von großer Wichtigkeit. Diese Hook-Methoden sollten möglichst vor der Entwicklung der einzelnen Features bekannt sein, um eventuelle Konflikte zu vermeiden.

IV. Evaluierung

Bei der Implementierung wurden die Arbeitspakete für die einzelnen Teammitglieder so eingeteilt, dass jeder für bestimmte Features zuständig war. Allerdings wurde schnell

klar, dass diese Strategie nur bedingt funktioniert. Dies war vor allem auf Abhängigkeiten zwischen einzelnen Features (benutzte Methoden und Felder) und auf die vorher schwer einzuschätzende Komplexität einzelner Features zurückzuführen.

Laut FeatureIDE sind mit der implementierten SPL 462 gültige Varianten möglich. Dies führt zu einem sehr hohen Testaufwand. Einige Features sind komplett unabhängig voneinander. So bestehen beispielsweise keine **Featureinteraktionen** zwischen den einzelnen ContentHandlern und den `User_Interface` Unterfeatures. Durch diese Unabhängigkeiten kann die Menge der zu testenden Varianten eingeschränkt werden. Durch die Auswahl aller ContentHandler reduzieren sich die möglichen Varianten auf 66.

Diese 66 Varianten zu testen ist immer noch sehr aufwendig, weshalb eine weitere Einschränkung notwendig ist. Da die Abhänigigkeiten von dem OS-Feature nur sehr gering sind (nur die `Tree_View`), wurde das Linux-Feature nur in Verbindung mit diesem Feature getestet. Dies führt zu einer weiteren, deutlichen Veringerung der zu testenden Varianten. Mit allen weiteren Varianten wurden erfolgreich dynamische und statische Tests durchgeführt.

V. Zusammenfassung

Im Rahmen dieser Arbeit wurde eine Software-Produktlinie aus einer bestehenden Desktop Suchengine entwickelt. Die dabei entstandenen konzeptuellen Probleme (Wahl des Paradigmas, Wahl der Architekturveränderung, Domänenanalyse) und praktischen Schwierigkeiten (Featurinteraktion, Testaufwand, Arbeitsteilung im Team) wurde aufgezeigt und erörtert, sowie Lösungsvorschläge erarbeitet. Dabei haben sich die Vorteile der Softwareproduktlinientechnologie bei der Erstellung verschiedener Varianten schnell bezahlt gemacht. Allerdings wurde auch der Preis deutlich, der in einem anspruchsvolleren Erstellungsprozess zu finden ist.

Literatur

[1] Sven Apel, Thomas Leich, and Gunter Saake. Aspectual feature modules. *IEEE Trans. Software Eng.*, 34(2):162–180, 2008.

[2] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. ACM/Addison-Wesley, 2000.

[3] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, volume 1241, pages 220–242. Springer, 1997.

[4] Christian Prehofer. Feature-Oriented Programming: A Fresh Look at Objects. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, volume 1241, pages 419–443. Springer, 1997.