

# Aufbau einer Software Factory zur Erstellung spezialisierter Bildverarbeitungslösungen

Torsten Heup, Thomas Berlage (Betreuer)

FIT LIFE  
Fraunhofer Institut für angewandte Informationstechnik  
Schloß Birlinghoven  
DE-53757 Sankt Augustin  
torsten.heup@fit.fraunhofer.de  
thomas.berlage@fit.fraunhofer.de

**Abstract:** Für die Erstellung experimentspezifischer Bildverarbeitungslösungen wird im Rahmen einer dreimonatigen Diplomarbeit eine Software Factory entwickelt, die zu erzeugende Applikationen anhand von drei Sichten definiert. Die Implementationssicht wird durch eine mit Hilfe von Annotationen um semantische Informationen erweiterte Java-Syntax beschrieben. Die Komposition der Anwendungskomponenten erfolgt mittels einer durch introspection gewonnenen XML-basierenden Konfigurationssprache. Für die Bestimmung passender Komponentenparameter wird ein trainierbares System verwendet, das durch die Vorgabe von Lernbeispielen konfiguriert wird.

## 1 Einleitung

Im Rahmen eines Projektes des Fraunhofer Instituts für angewandte Informationstechnik wird die Bildanalyseplattform Zeta für Anwendungen im Bereich der biologischen Bildgebung entwickelt. Auf der Grundlage dieses Systems lassen sich experimentspezifische Applikationen erstellen, mit denen Bilddaten hinsichtlich verschiedenster Kriterien ausgewertet werden können. Bei den dabei erzeugten Anwendungen handelt es sich um Projekte, die sich aufgrund der Heterogenität der auszuwertenden Daten in Teilen stark voneinander unterscheiden. Eine derartige Anwendung soll hier zunächst beispielhaft skizziert werden.

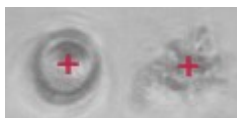


Abb. 1: Links lebende,  
rechts tote Zelle

Abbildung 1 zeigt Aufnahmen einer lebenden und einer abgestorbenen Zelle im Vergleich. Eine typische Anwendung für eine Bildverarbeitungsapplikation im Bereich der biologischen Bildgebung läge etwa darin, periodisch gemachte Aufnahmen einer Zellkultur auszuwerten und statistische Daten über das Verhältnis lebendiger zu abgestorbener Zellen zu erfassen. Zwar ist es durchaus möglich, ein Verfahren zur automatischen

Auswertung derartiger Bildaufnahmen mit vertretbarem Aufwand zu implementieren, leichte Veränderungen der Daten stellen für ein solches Verfahren allerdings eine schwer zu überwindende Hürde dar. In Praxis variieren die betrachteten Zelltypen sowie die Art der Aufnahme von Anwendung zu Anwendung sehr stark.

Um eine bestehende Applikation für einen neuen Bildtyp nutzbar zu machen ist es notwendig, tiefgreifende Veränderungen an der Software vorzunehmen. So müssen für einen neuen Problemfall eventuell neue Bildfilter oder Auswertungsalgorithmen implementiert werden, bestehende Ketten von Filtern sowie der Ablauf der Auswertungsschritte müssen gegebenenfalls modifiziert und ergänzt werden. Ferner ist es notwendig, die Konfiguration der einzelnen Komponenten anzupassen. Dieser Schritt ist in der Regel sehr zeitaufwändig, da jeder Bildfilter und jedes Analysemodul über eine Vielzahl von zu justierenden Parametern verfügt.

Ungeachtet der weitreichenden Anpassungen, die für jeden neuen Anwendungsfall vorgenommen werden müssen, lässt sich für die zu erzeugenden Produkte eine breite Basis gemeinsamer Eigenschaften identifizieren. Die auf dieser Basis erzeugten Applikationen können somit als Softwareproduktlinie [We06] verstanden werden.

## **2 Existierende Lösungsansätze**

Es existiert eine Vielzahl möglicher Ansätze, die das Ziel verfolgen, die Entwicklung von Software innerhalb einer Produktlinie zeit- und kostengünstiger zu gestalten. Der Schwerpunkt liegt dabei einheitlich auf der Idee, die gleich bleibenden Bestandteile der Softwareprodukte hinreichend zu abstrahieren, um eine einfache Wiederverwendung zu ermöglichen.

Ein Ansatz besteht in der Erstellung einer Bibliothek von anwendungsspezifischen Komponenten, die die Zusammenstellung einer Applikation durch komponentenbasiertes Design ermöglicht. Eine solche Bibliothek bringt in der Bildverarbeitung oftmals nicht den gewünschten Nutzen, da die Konfiguration der exponentiellen Anzahl von Parameterkombinationen nicht zielgerichtet unterstützt wird.

Feature models stellen ein weitergehendes Konzept zur Erstellung von Anwendungen im Rahmen einer Softwareproduktlinie dar. Dabei stellt ein Feature eine einzelne Eigenschaft dar, die ein entsprechendes Produkt aufweisen kann. Anhand eines Feature Models werden dann die Beziehungen zwischen diesen Eigenschaften festgelegt. Eine Applikation wird erstellt, indem die für den konkreten Anwendungsfall relevanten Features ausgewählt werden und der entsprechende Programmcode anhand dieser Auswahl automatisch generiert wird. Dieser Ansatz geht dahingehend weiter als die reine Bereitstellung von Komponenten, als dass Feature Models semantische Informationen über das Zusammenwirken der Programmbestandteile beinhalten und somit bereits weite Teile der Applikation automatisch erzeugen lassen können [SZ05].

Die Nutzung von Feature Models zur Erzeugung von Bildanalysetools im zuvor geschilderten Kontext gestaltet sich jedoch dahingehend schwierig, als dass das Konzept auf einer statischen Menge von Komponenten aufbaut. Die beschriebenen Projekte erfordern jedoch stete Neuentwicklungen und Erweiterungen. Ferner wird der Schwerpunkt der Anwendungsentwicklung, die Auswahl und Konfiguration der verwendeten Parameter, durch Feature Models nicht begünstigt.

### 3 Software Factories

Software Factories erweitern die bestehenden Ansätze um zusätzliche Dimensionen. Im Gegensatz zu den vorangegangenen Konzepten wird die Applikation nicht allein durch einen Formalismus, sondern durch mehrere Sprachen spezifiziert. Dazu werden zunächst diejenigen Sichten auf das Projekt bestimmt, die für die Anwendungsentwicklung relevante Aspekte beinhalten. Für jeden dieser so genannten Viewpoints wird dann eine domänenspezifische Sprache definiert, die den jeweiligen Aspekt so präzise wie möglich beschreibt [GS04]. Von dem Konzept des Model Driven Development unterscheidet sich eine Software Factory dann dahingehend, dass nicht der Anspruch erhoben wird, eine umfassende und generell gültige Sprache für verschiedene Aspekte der Programmierung zu finden. Vielmehr wird jedem als relevant erachteten Aspekt eine eigene, spezialisierte Form der Darstellung zugewiesen. Naturgemäß werden sich die definierten Viewpoints in Einzelfällen tangieren, wenn gleiche oder abhängige Anwendungsaspekte beschrieben werden. Für einen solchen Fall muss ein Weg gefunden werden, in einer Sicht gemacht Veränderungen automatisch mit den anderen Sichten zu synchronisieren.

Im Kontext der beschriebenen Bildanalysewerkzeuge wurden in der Entwicklung drei Problemsichten als relevant identifiziert. Die erste Sicht beinhaltet die Implementation der von der Software benötigten Algorithmen als Softwarekomponenten, etwa in der Gestalt von Bildfiltern. Die Umsetzung solcher Algorithmen erfolgt im Rahmen des Projektes in der Programmiersprache Java. Java-Komponenten sind in der Regel nur in einem sehr geringen Maße selbstbeschreibend und beinhalten keine Informationen über ihren Verwendungskontext. Zwar lassen sich mit Hilfe externer Werkzeuge über verschiedene Introspection-Mechanismen eine Anzahl von Details auslesen, semantische Informationen jedoch nicht. Der Java Sprachstandard beinhaltet dazu seit JSE 5 die Möglichkeit, zur Beigabe von Metadaten eigene Schlüsselwörter zu definieren. Mithilfe dieser sogenannten *Annotationen* lassen sich sowohl einfache constraints als auch komplexe semantische Konstrukte direkt in den Quellcode einfügen. Die Metadaten werden so direkt an den Quellcode gebunden, ohne dass eine eigene Repräsentation, also etwa eine weitere Datei, notwendig wird. Ferner besteht die Möglichkeit, Annotationen zur Laufzeit innerhalb der Applikation zu verarbeiten. So können beispielsweise Codefragmente dynamisch zur Laufzeit erzeugt werden. Damit entfällt während der Entwicklung ein weiterer Zwischenschritt, der für einen klassischen Codegenerator notwendig wäre. Es bietet sich daher an, an dieser Stelle Annotationen als Basis für die domänenspezifische Sprache zu verwenden.

Der zweite relevante Aspekt bei der Erstellung eines Analysewerkzeuges beinhaltet die Auswahl und Zusammenstellung von Filtern zu einer auf das Problem angepassten Sammlung von Filterketten. Hierzu bietet es sich an, auf eine der bereits existierenden Lösungen zur Komposition von Komponenten zu Workflows zurückzugreifen. Das Spring Framework bietet dazu eine einfache XML-basierte Syntax, mit der sich eine Konfiguration mittels *Inversion of Control* (IOC) vornehmen lässt. Die grundlegenden Vorteile, die mit IOC einhergehen, gelten dabei uneingeschränkt im Rahmen einer Software Factory. Neben einer klaren Trennung zwischen Programmcode und Konfiguration wird so eine Austauschbarkeit der verwendeten Filter ermöglicht, ohne händisch in den Quelltext eingreifen und die Applikation für jeden Anwendungsfall neu

kompilieren zu müssen. Ferner bietet das Spring Framework bereits von sich aus eine große Anzahl von Werkzeugen zur automatisierten Erzeugung benötigter Programmbestandteile, etwa um Parameterwerte zwischen unterschiedlichen Darstellungsformen zu überführen. Das Vokabular der von Spring verwendeten Konfigurationssprache wird über die in Java zur Verfügung stehenden Introspection-Mechanismen direkt aus den verwendeten Komponenten abgeleitet. Für den Entwickler bedeutet dies, dass sich Änderungen in der Implementierungssicht direkt in der Konfigurationssicht niederschlagen, ohne dass die Sprache händisch angepasst werden muss. Damit wird die Anforderung eines Mappings zwischen den die Sichten beschreibenden Sprachen durch Spring automatisch erfüllt.

Die dritte Sicht beschreibt die Konfiguration der ausgewählten Komponenten für einen Bildtyp. Im klassischen Programmieransatz ist diese Konfiguration aufgrund der großen Anzahl möglicher Parameterkombinationen mit einem häufigen recompilieren des Quellcodes verbunden und daher extrem zeitaufwändig. Auch der Ansatz, die Parameter wie die Filterkomposition über externe Konfigurationsdateien zu verwalten, bleibt im Zuge eines langwierigen Trial and Error Verfahrens sehr arbeitsintensiv. In der Bildanalyse-Software Zeta wird daher ein Weg gewählt, der den Auswahlprozess zu weiten Teilen automatisiert. Dazu verfügt die Software über einen *trainierbaren Kern*, der die zu analysierenden Bildeigenschaften anhand von Beispielen über verschiedene Lernverfahren erschließt. Die domänenspezifische Sprache, die in dieser Sicht verwendet wird, unterscheidet sich damit grundlegend von den bereits beschriebenen DSLs und verfügt nicht einmal über eine textuelle Repräsentation. Statt dessen wird eine Sammlung von Lernbeispielen dazu verwendet, die Software so weit zu trainieren, dass für das Problem optimale Parameterkonfigurationen ausgewählt werden können. Für das zuvor beschriebene Beispiel genügt es dann, eine geringe Zahl von lebendigen bzw. abgestorbenen Zellen vorzugeben.

Mithilfe von Zeta wurden bereits eine Reihe von kundenspezifischen Applikationen generiert, die unterschiedliche Anwendungen der Durchlicht- und Fluoreszenzmikroskopie abdecken. Dabei wurde ein stabiler Kern von Komponenten etabliert, dessen grundlegende Operationen nur selten erweitert oder modifiziert werden müssen. Die im Rahmen der unterschiedlichen Anwendungsfälle der Durchlicht- bzw. Fluoreszenzaufnahmen dennoch notwendigen werdenden Veränderungen der Software lassen sich durch die erstellte Software Factory effizient behandeln. In einem weiteren Entwicklungsschritt soll nun eine Erweiterung des Sichtenmodells vorgenommen werden, um zusätzliche Domänen wie zum Beispiel die Datenvisualisierung zu erschließen.

## 4 Literaturverzeichnis

- [GS04] Greenfield, J; Short, K: Software factories – Assembling Applications with Patterns, Models, Frameworks and Tools. Wiley Publishing, 2004
- [SZ05] Sun, J; Zhang, H: Formal Semantics and Verification for Feature Modeling, Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems, 2005
- [We06] Weiss, D: Software Product-Line Engineering: A Family-Based Software Development Process, Addison-Wesley 1999