Hochschule für angewandte Wissenschaften Würzburg-Schweinfurt
Fakultät Informatik und Wirtschaftsinformatik

Bachelorarbeit

Über die Automatisierung der Entwicklung von Software Generatoren

vorgelegt an der Hochschule für angewandte Wissenschaften Würzburg-Schweinfurt in der Fakultät Informatik und Wirtschaftsinformatik zum Abschluss eines Studiums im Studiengang Informatik

René Ziegler

Eingereicht am: 24. Januar 2018

Erstprüfer: Prof. Dr. Peter Braun
Zweitprüfer: M.Sc. Tobias Fertig

Zusammenfassung

TODO

Abstract

TODO

Danksagung

Inhaltsverzeichnis

1. Einführung **1**
1.1. Motivation . . . 2
1.2. Zielsetzung . . . 3
1.3. Aufbau der Arbeit . . . 3

2. Grundlagen **4**
2.1. Modellgetriebene Softwareentwicklung (MDSD) . . . 5
2.1.1. Domäne . . . 5
2.1.1.1. Definition . . . 5
2.1.1.2. Domänenanalyse . . . 5
2.1.1.3. Feature Modelling . . . 5
2.1.2. Metamodell . . . 5
2.1.2.1. Definition . . . 5
2.1.2.2. Abstraktes Modell . . . 5
2.1.2.3. Konkretes Modell . . . 5
2.1.3. Domänenspezifische Sprache (DSL) . . . 5
2.1.3.1. Definition . . . 5
2.1.3.2. General Purpose Language (GPL) . . . 5
2.1.3.3. Interne DSLs . . . 5
2.1.3.4. Externe DSLs . . . 5
2.1.3.5. Parser . . . 5
2.1.4. Code Generator . . . 5
2.1.4.1. Definition . . . 5
2.1.4.2. Techniken zur Generierung von Code . . . 5
2.1.4.3. Zusammenhang zu Transformatoren . . . 5
2.1.4.4. Abgrenzung zum Compilerbau . . . 5
2.1.5. MDSD Entwicklungsprozess . . . 5
2.2. Software Engineering . . . 5
2.2.1. Architektur . . . 5
2.2.1.1. Prinzipien in der Softwaretechnik . . . 5
2.2.1.2. Trennung durch Modelle . . . 5
2.2.1.3. Modell-Transformations-Pipeline . . . 5
2.2.2. Design Pattern objektorientierter Programmierung . . . 5
2.2.2.1. Visitor Pattern . . . 5
2.2.2.2. Builder Pattern und Fluent Interfaces . . . 5

2.2.2.3. Factory Pattern 5

3. Problemstellung **6**
3.1. Wahl der zu unterstützenden Features 6
3.2. Definition der Variabilitäten auf Basis einer Referenzimplementation . . . 6
3.3. Verwendung von Zwischenmodellen 6
3.4. Entscheidung zwischen Meta-Generator und Transformator 6
3.5. Abstrakte Modellierung von Java-Code 6

4. Lösung: Spectrum (Proof of Concept) **7**
4.1. Verwendete Bibliotheken 8
4.1.1. JavaParser mit JavaSymbolSolver 8
4.1.2. JavaPoet 8
4.2. Architekturübersicht 8
4.2.1. Amber 8
4.2.1.1. Annotationen 8
4.2.1.2. Parser 8
4.2.1.3. Modell 8
4.2.2. Cherry 8
4.2.2.1. Generator 8
4.2.2.2. Modell 8
4.2.2.3. Generierte Builder 8
4.2.3. Jade 8
4.2.3.1. Transformator 8
4.2.4. Scarlet 8
4.2.4.1. Generator 8
4.2.4.2. Modell 8

5. Evaluierung **10**
5.1. Softwarequalität 10
5.1.1. Functionality 10
5.1.2. Maintainability 10
5.1.3. Performance 10
5.1.4. Usability 10
5.2. Grenzen des Lösungsansatzes 10

6. Zusammenfassung und Ausblick **12**
6.1. Zusammenfassung 12
6.2. Ausblick 12

A. Dokumentation **13**
A.1. Verwendung der Annotationen 13
A.2. Verwendung der generierten CodeUnit-Builder 13

A.3. Klassendokumentation 13
A.3.1. Amber 13
A.3.2. Cherry 13
A.3.3. Jade 13
A.3.4. Scarlet 13

Verzeichnisse **14**

Literatur **16**

Eidesstattliche Erklärung **17**

Zustimmung zur Plagiatsüberprüfung **18**

1. Einführung

Als Henry Ford 1913 die Produktion des Modell T, umgangssprachlich auch Tin Lizzie genannt, auf Fließbandfertigung umstellte, revolutionierte er die Automobilindustrie. Ford war nicht der erste, der diese Form der Automatisierung verwendete. Bereits 1830 kam in den Schlachthöfen von Chicago eine Maschine zum Einsatz, die an Fleischerhaken aufgehängte Tierkörper durch die Schlachterei transportierte. Bei der Produktion des Oldsmobile Curved Dash lies Ranson Eli Olds 1910 erstmals die verschiedenen Arbeitsschritte an unterschiedlichen Arbeitsstationen durchführen. Fords Revolution war die Kombination beider Ideen. Er entwickelte eine Produktionsstraße, auf welcher die Karossen auf einem Fließband von Arbeitsstation zu Arbeitsstation befördert wurden. An jeder Haltestelle wurden nur wenige Handgriffe von spezialisierten Arbeitern durchgeführt [5].

Fords Vision war es, ein Auto herzustellen, welches sich Menschen aller Gesellschaftsschichten leisten konnten. Durch die Reduktion der Produktionszeit der Tin Lizzie von 12,5 Stunden auf etwa 6 Stunden konnte Ford den Preis senken. Kostete ein Auto des Model T vor der Einführung der Produktionsstraße 825$, erreichte der Preis in den Jahren danach einen Tiefststand von 259$ [4]. Setzt man diesen Preis in ein Verhältnis mit dem durchschnittlichen Einkommen in den USA, das 1910 bei jährlich 438$ lag, kann man sagen, dass Fords Traum durch die eingesetzten Techniken Realität wurde [3].

Im Zuge der weiteren Entwicklung der Robotik wurden immer mehr Aufgaben, die bisher von Menschen am Fließband durchgeführt wurden, von Automaten übernommen. In der Automobil-Industrie war General Motors der erste Hersteller, bei welchem die Produktionsstraßen im Jahr 1961 mit 66 Robotern des Typs Unimation ausgestattet wurden. Bis zur Erfindung des integrierten Schaltkreises in den 1970ern waren die Roboter ineffizient. Der Markt für industrielle Roboter explodierte jedoch in den Folgejahren. Im Jahr 1984 waren weltweit ungefähr 100.000 Roboter im Einsatz [1, 7].

Die industrielle Revolution prägte die Autoindustrie: von der Erfindung auswechselbarer Teile 1910 bei Ransom Olds, über die Weiterentwicklung des Konzepts unter der Verwendung von Fließbändern bei Ford im Jahr 1913, bis hin zur abschließenden Automatisierung mit Industriellen Robotern in den frühen 1980ern [1].

1.1. Motivation

> „If you can compose components manually, you can also automate this process.“

Das hervorgehobene Zitat nennen Czarnecki und Eisenecker die Automation Assumption. Diese allgemein gehaltene Aussage lässt die Parallelen, die die beiden Autoren zwischen der Automatisierung der Automobilindustrie und der automatischen Code Generierung sehen, erkennen. Dafür müssten die einzelnen Komponenten einer Softwarefamilie derart gestaltet werden, dass diese austauschbar in eine gemeinsame Struktur integriert werden können. Des weiteren müsste klar definiert sein, welche Teile eines Programms konfigurierbar seien und welche der einzelnen Komponenten in welcher Konfiguration benötigt werden. Setzt man dieses definierte Wissen in Programmcode um, könnte ein solches Programm eine Software in einer entsprechenden Konfiguration generieren [1].

Konkret bedeutet dies, dass entweder eine vorhandene Implementierung in Komponenten zerlegt werden muss oder eine für die Zwecke der Codegenerierung vorgesehene Referenzimplementierung geschrieben wird. Codeabschnitte, die in Ihrer Struktur gleich sind, sich jedoch inhaltlich unterscheiden, müssen formal beschrieben werden [6]. Ein solches abstraktes Modell wird dann mit Daten befüllt. Schlussendlich wird ein Generator implementiert, der den Quellcode für unterschiedliche Ausprägungen eines Programms einer Software-Familie, auf Basis des konkreten Modells, generieren kann [2].

Sowohl bei der Umsetzung von einzigartigen Anwendungen, als auch bei der Verwirklichung von Software mit mehreren Varianten, kann die Verwendung von bereits verfügbaren Code Generatoren oder die Entwicklung eigener Code Generatoren vorteilhaft sein. Die Entwicklungsgeschwindigkeit könnte erhöht, die Softwarequalität gesteigert und Komplexität durch Abstraktion reduziert werden [6]. Allgemein wird weniger Zeit benötigt, um eine größere Vielfalt an ähnlichen Programmen zu entwickeln [1].

Bisher müssen fast alle Teilaufgaben bei der Umsetzung eines Code Generators manuell durchgeführt werden. Werkzeuge wie Language Workbenches können Code bis zu einem gewissen Grad automatisiert generieren oder interpretieren. Sie haben aber in erster Linie die Aufgabe, den Entwickler beim Design von externen domänenspezifischen Sprachen zu unterstützen und dienen als Entwicklungsumgebung für die Arbeit mit der Sprache [2].

1.2. Zielsetzung

In dieser Arbeit soll untersucht werden, wie weitere Teile der Prozesse in der modellgetriebenen Softwareentwicklung automatisiert werden können. Potentiell könnte hier eine zusätzliche Ebene der Indirektion dieses komplexe Thema weiter vereinfachen und somit Code Generierung für mehr Entwickler zugänglich machen.

Im Speziellen wird analysiert, nach welcher Struktur ein Generator aufgebaut sein muss, um diesen von einer Referenzimplementierung des erwünschten Generats abzuleiten. Zu diesem Zweck wird eine beispielhafte Java Anwendung erarbeitet, welche als Ausgabeprodukt den Code der ursprünglichen Implementierung repliziert. Für die Transformation der Referenzimplementierung wird vorerst nur auf einen Teil des Java 8 Sprachschatzes Rücksicht genommen.

Anschließend wird ermittelt, welche Schritte unternommen werden müssen, um die bestehende Referenzimplementierung in ein abstraktes Modell abzuleiten. Dabei geht es vor allem darum, wie der vorhandene Quelltext bearbeitet werden muss und in welcher Form das abstrakte Modell generiert werden sollte, um es als Eingabemodell für den erarbeiteten Generator zu verwenden. Hierfür wird die erste beispielhafte Anwendung weiterentwickelt, um die Generierung von Varianten der transformierten Referenzimplementierung zu ermöglichen.

Es wird in dieser Arbeit nicht auf die wirtschaftlichen Aspekte bei der Verwendung von Code Generatoren eingegangen.

1.3. Aufbau der Arbeit

2. Grundlagen

2.1. Modellgetriebene Softwareentwicklung (MDSD)

2.1.1. Domäne

2.1.1.1. Definition

2.1.1.2. Domänenanalyse

2.1.1.3. Feature Modelling

2.1.2. Metamodell

2.1.2.1. Definition

2.1.2.2. Abstraktes Modell

2.1.2.3. Konkretes Modell

2.1.3. Domänenspezifische Sprache (DSL)

2.1.3.1. Definition

2.1.3.2. General Purpose Language (GPL)

2.1.3.3. Interne DSLs

2.1.3.4. Externe DSLs

2.1.3.5. Parser

2.1.4. Code Generator

2.1.4.1. Definition

2.1.4.2. Techniken zur Generierung von Code

2.1.4.3. Zusammenhang zu Transformatoren

3. Problemstellung

3.1. Wahl der zu unterstützenden Features

3.2. Definition der Variabilitäten auf Basis einer Referenzimplementation

3.3. Verwendung von Zwischenmodellen

3.4. Entscheidung zwischen Meta-Generator und Transformator

3.5. Abstrakte Modellierung von Java-Code

4. Lösung: Spectrum (Proof of Concept)

4.1. Verwendete Bibliotheken

4.1.1. JavaParser mit JavaSymbolSolver

4.1.2. JavaPoet

4.2. Architekturübersicht

4.2.1. Amber

4.2.1.1. Annotationen

4.2.1.2. Parser

4.2.1.3. Modell

4.2.2. Cherry

4.2.2.1. Generator

4.2.2.2. Modell

4.2.2.3. Generierte Builder

4.2.3. Jade

4.2.3.1. Transformator

4.2.4. Scarlet

4.2.4.1. Generator

4.2.4.2. Modell

Listing 4.1: Beispiel für einen Quelltext

```
1
2 public void foo() {
3     // Kommentar
4 }
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut vehicula felis lectus, nec aliquet arcu aliquam vitae. Quisque laoreet consequat ante, eget pretium quam hendrerit at. Pellentesque nec purus eget erat mattis varius. Nullam ut vulputate velit. Suspendisse in dui in eros iaculis tempus. Phasellus vel est arcu. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Integer elementum, nulla eu faucibus dignissim, orci justo imperdiet lorem, luctus consectetur orci orci a nunc.

Praesent at nunc nec tortor viverra viverra. Morbi in feugiat lectus. Vestibulum iaculis ipsum at eros viverra volutpat in id ipsum. Donec condimentum, ligula viverra pharetra tincidunt, nunc dui malesuada nisi, vitae mollis lacus massa quis velit. Integer feugiat ipsum a volutpat scelerisque. Nulla facilisis augue nunc. Curabitur eget consectetur nulla. Integer accumsan sem non nisi tristique dictum.

Sed lacinia eu dolor sed congue. Ut dui orci, venenatis id interdum rhoncus, mattis elementum massa. Proin venenatis elementum purus ut rutrum. Phasellus sit amet enim porta, commodo mauris a, bibendum tortor. Nulla ut lobortis justo. Aenean auctor mi nec velit fermentum, quis ultricies odio viverra. Maecenas ultrices urna vel erat ornare, quis suscipit odio molestie. Donec vel dapibus orci, vel tincidunt orci.

Etiam vitae eros erat. Praesent nec accumsan turpis, et mollis eros. Praesent lacinia nulla at neque porta aliquam. Quisque elementum neque ac porta suscipit. Nulla volutpat luctus venenatis. Aliquam imperdiet suscipit pretium. Nunc feugiat lacinia aliquet. Mauris ut sapien nec risus porttitor bibendum. Aenean feugiat bibendum lectus, id mattis elit adipiscing at. Pellentesque interdum felis non risus iaculis euismod fermentum nec urna. Nullam lacinia suscipit erat ac ullamcorper. Sed vitae nulla posuere, posuere sem id, ultricies urna. Maecenas eros lorem, tempus non nulla vitae, ullamcorper egestas nibh. Vestibulum facilisis ante vel purus accumsan mattis. Donec molestie tempor eros, a gravida odio congue posuere.

Sed in tempus elit, sit amet suscipit quam. Ut suscipit dictum molestie. Etiam quis porta mauris. Cras dapibus sapien eget sem porta, ut congue sapien accumsan. Maecenas hendrerit lobortis mauris ut hendrerit. Suspendisse at aliquet est. Quisque eros est, scelerisque ac orci quis, placerat suscipit lorem. Phasellus rutrum enim non odio ullamcorper, sit amet auctor nulla fringilla. Nunc eleifend vulputate dui, a sollicitudin tellus venenatis non. Cras condimentum lorem at ultricies vestibulum. Vestibulum interdum lobortis commodo. Nullam rhoncus interdum massa, ut varius nisi scelerisque id. Nunc interdum quam in enim bibendum vulputate.

5. Evaluierung

5.1. Softwarequalität

Nach Balzert S.111

5.1.1. Functionality

5.1.2. Maintainability

5.1.3. Performance

5.1.4. Usability

5.2. Grenzen des Lösungsansatzes

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut vehicula felis lectus, nec aliquet arcu aliquam vitae. Quisque laoreet consequat ante, eget pretium quam hendrerit at. Pellentesque nec purus eget erat mattis varius. Nullam ut vulputate velit. Suspendisse in dui in eros iaculis tempus. Phasellus vel est arcu. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Integer elementum, nulla eu faucibus dignissim, orci justo imperdiet lorem, luctus consectetur orci orci a nunc.

Praesent at nunc nec tortor viverra viverra. Morbi in feugiat lectus. Vestibulum iaculis ipsum at eros viverra volutpat in id ipsum. Donec condimentum, ligula viverra pharetra tincidunt, nunc dui malesuada nisi, vitae mollis lacus massa quis velit. Integer feugiat ipsum a volutpat scelerisque. Nulla facilisis augue nunc. Curabitur eget consectetur nulla. Integer accumsan sem non nisi tristique dictum.

Sed lacinia eu dolor sed congue. Ut dui orci, venenatis id interdum rhoncus, mattis elementum massa. Proin venenatis elementum purus ut rutrum. Phasellus sit amet enim porta, commodo mauris a, bibendum tortor. Nulla ut lobortis justo. Aenean auctor mi nec velit fermentum, quis ultricies odio viverra. Maecenas ultrices urna vel erat ornare, quis suscipit odio molestie. Donec vel dapibus orci, vel tincidunt orci.

Etiam vitae eros erat. Praesent nec accumsan turpis, et mollis eros. Praesent lacinia nulla at neque porta aliquam. Quisque elementum neque ac porta suscipit. Nulla volutpat luctus venenatis. Aliquam imperdiet suscipit pretium. Nunc feugiat lacinia aliquet. Mauris ut sapien nec risus porttitor bibendum. Aenean feugiat bibendum lectus, id mattis elit adipiscing at. Pellentesque interdum felis non risus iaculis euismod fermentum nec urna. Nullam lacinia suscipit erat ac ullamcorper. Sed vitae nulla posuere, posuere sem id, ultricies urna. Maecenas eros lorem, tempus non nulla vitae, ullamcorper egestas nibh. Vestibulum facilisis ante vel purus accumsan mattis. Donec molestie tempor eros, a gravida odio congue posuere.

Sed in tempus elit, sit amet suscipit quam. Ut suscipit dictum molestie. Etiam quis porta mauris. Cras dapibus sapien eget sem porta, ut congue sapien accumsan. Maecenas hendrerit lobortis mauris ut hendrerit. Suspendisse at aliquet est. Quisque eros est, scelerisque ac orci quis, placerat suscipit lorem. Phasellus rutrum enim non odio ullamcorper, sit amet auctor nulla fringilla. Nunc eleifend vulputate dui, a sollicitudin tellus venenatis non. Cras condimentum lorem at ultricies vestibulum. Vestibulum interdum lobortis commodo. Nullam rhoncus interdum massa, ut varius nisi scelerisque id. Nunc interdum quam in enim bibendum vulputate.

6. Zusammenfassung und Ausblick

6.1. Zusammenfassung

6.2. Ausblick

A. Dokumentation

A.1. Verwendung der Annotationen

A.2. Verwendung der generierten CodeUnit-Builder

A.3. Klassendokumentation

A.3.1. Amber

A.3.2. Cherry

A.3.3. Jade

A.3.4. Scarlet

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] K. Czarnecki und U. Eisenecker. *Generative Programming: Methods, Tools and Applications*. Addison-Wesley, 2000.

[2] M. Fowler. *Domain Specific Languages*. Addison-Wesley, 2011.

[3] o.V. *Zahlen & Fakten: Einkommen und Preise 1900 - 1999*. o.D. URL: https://usa.usembassy.de/etexts/his/e_g_prices1.htm.

[4] J. Reichle. *100 Jahre Ford T-Modell: Schwarze Magie*. 2010. URL: http://www.sueddeutsche.de/auto/jahre-ford-t-modell-schwarze-magie-1.702183.

[5] G. Sager. *Erfindung des Ford Modell T: Der kleine Schwarze*. 2008. URL: http://www.spiegel.de/einestages/100-jahre-ford-modell-t-a-947930.html.

[6] M. Stahl T. und Völter. *Modellgetriebene Softwarentwicklung: Techniken, Engineering, Management*. 2. Aufl. dpunkt.verlag, 2007.

[7] W. Wallén. *The history of the industrial robot*. Techn. Ber. Division of Automatic Control at Linköpings universitet, 2008. URL: http://liu.diva-portal.org/smash/get/diva2:316930/FULLTEXT01.pdf.

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorgelegte Bachelorarbeit selbstständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe. Alle benutzten Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate wurden als solche gekennzeichnet.

René Ziegler, am 24. Januar 2018

Zustimmung zur Plagiatsüberprüfung

Hiermit willige ich ein, dass zum Zwecke der Überprüfung auf Plagiate meine vorgelegte Arbeit in digitaler Form an PlagScan (www.plagscan.com) übermittelt und diese vorrübergehend (max. 5 Jahre) in der von PlagScan geführten Datenbank gespeichert wird sowie persönliche Daten, die Teil dieser Arbeit sind, dort hinterlegt werden.

Die Einwilligung ist freiwillig. Ohne diese Einwilligung kann unter Entfernung aller persönlichen Angaben und Wahrung der urheberrechtlichen Vorgaben die Plagiatsüberprüfung nicht verhindert werden. Die Einwilligung zur Speicherung und Verwendung der persönlichen Daten kann jederzeit durch Erklärung gegenüber der Fakultät widerrufen werden.

René Ziegler, am 24. Januar 2018