B A C H E L O R A R B E I T

in der Fachrichtung
Wirtschaftsinformatik

T H E M A

Konzeption einer DSL zur Beschreibung von Benutzeroberflächen für profil c/s auf der Grundlage des Multichannel-Frameworks der deg

Eingereicht von:	Niels Gundermann (Matrikelnr. 5023)
	Woldegker Straße 34
	17033 Neubrandenburg
	E-Mail: gundermann.niels.ng@googlemail.com
Erarbeitet im:	7. Semester
Abgabetermin:	13. Februar 2015
Gutachter:	Prof. Dr. Johannes Brauer
Co-Gutachter:	
Betrieblicher Gutachter:	Dipl.-Ing. Stefan Post
	Woldegker Straße 12
	17033 Neubrandenburg
	Tel.: 0395/5630553
	E-Mail: stefan.post@data-experts.de

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Kapitel 1

Motivation

In der heutigen Zeit werden Programme auf vielen unterschiedlichen Gräten[1] von ausgeführt. Die Benutzeroberfläche neben der internen Umsetzung immer ein wichtiger Faktor, der für den Erfolg einer Anwendung eine große Rolle spielt. [LW] Damit einher geht die Usability[2] einer Anwendung. Denn eine *[...] schlechte Useability führt zu Verwirrung und Miss- bzw. Unverständis beim Kunden* [Use12]. Dadurch geht letztendlich Umsatz verloren. Wenn ein Programm auf unterschiedlichen Gräten ausgeführt wird, muss der Entwickler bei der traditionllen Entwicklung[3] mehrere Graphical User Interfaces (GUI)[4] bereitstellen. Folglich werden mehrere GUIs mit unterschiedlichen Toolkits oder Frameworks entworfen. Diese Framworks haben einen starken imperativen Character, sind schwer zu erweitern und sie verhalten sich unterschiedlich abhängig von der speziellen Implementierung. [KB11] Der Entwickler muss das GUI bei diesem Ansatz für jedes Framework explizit beschreiben.
Ein anderer Ansatz zur Beschreibung von Benutzeroberflächen ist das Model-Driven Development. Damit sollen UIs anhand der implementierten Funktionen automatisch erzeugt werden können. [SKNH05] Allerdings wird die Darstellung dieser generierten UIs von der Darstellung von traditionell implementierter Benutzerschnittstellen übertroffen. [MHP99].
Eine Überlegung, die sich daraus ergibt, ist, ob man diese beiden Ansätze

[1] Desktop, Smartphone, Tablet
[2] Siehe Glossar: Usability
[3] Siehe Glossar: Traditionelle UI-Entwicklung
[4] Siehe Glossar: GUI

zur Implementierung von UIs (traditionell und Model-Driven) verbinden kann. Somit kann die genaue Beschreibung der Darstellung mit einer höheren Abstraktion verbunden werden.
In dieser Arbeit wird versucht diese Idee umzusetzen. Bei der Umsetzung wird sich auf die UIs der Anwendung *profil c/s*. Profil c/s ist INVEKOS[5] -Programm welches von der deg als Client-Server-Anwendung entwickelt wird. In dieser Arbeit wird versucht diese Idee an einem ausgewählten Beispiel umzusetzen.

[5]Siehe Glossar: InVeKoS

Kapitel 2

Problembeschreibung und Zielsetzung

2.1 Allgemeine Anforderungen an Benutzeroberflächen von profil c/s

Die wichtigste (primäre) Anforderung für diese Arbeit bezieht sich auf den Client von profil c/s. Dieser soll sowohl in Web-Browsern (Web-Client) als auch standalone auf einem PC (Standalone-Client) ausgeführt werden können.
Um eine effiziente Arbeitsweise zu ermöglichen, kamen (sekundäre) Anforderungen wie *Erweiterbarkeit der Frameworks*, *Abstraktion* und die *Ausdruckskraft*[1] der Sprachkonstrukte, die zur Entwicklung verwendet werden.
In Abbildung 2.1 und Abbildung 2.2 ist das GUI eines Zuwendungsblatt[2] eines Förderantrag[3] zu sehen. Für den Aufbau sind nur die Tabelle und die darunter stehenden Buttons, sowies das Bemerkungsfeld (im Web-Client auf der rechten Seite und im Standalone-Client in der Mitte) von Bedeutung. Dass der Aufbau der GUI in beiden Clients ähnlich ist, liegt an der Umsetzung der GUI.

[1] Siehe Glossar: Ausdruckskraft
[2] Siehe Glossar: Zuwendungsblatt
[3] Siehe Glossar: Förderantrag

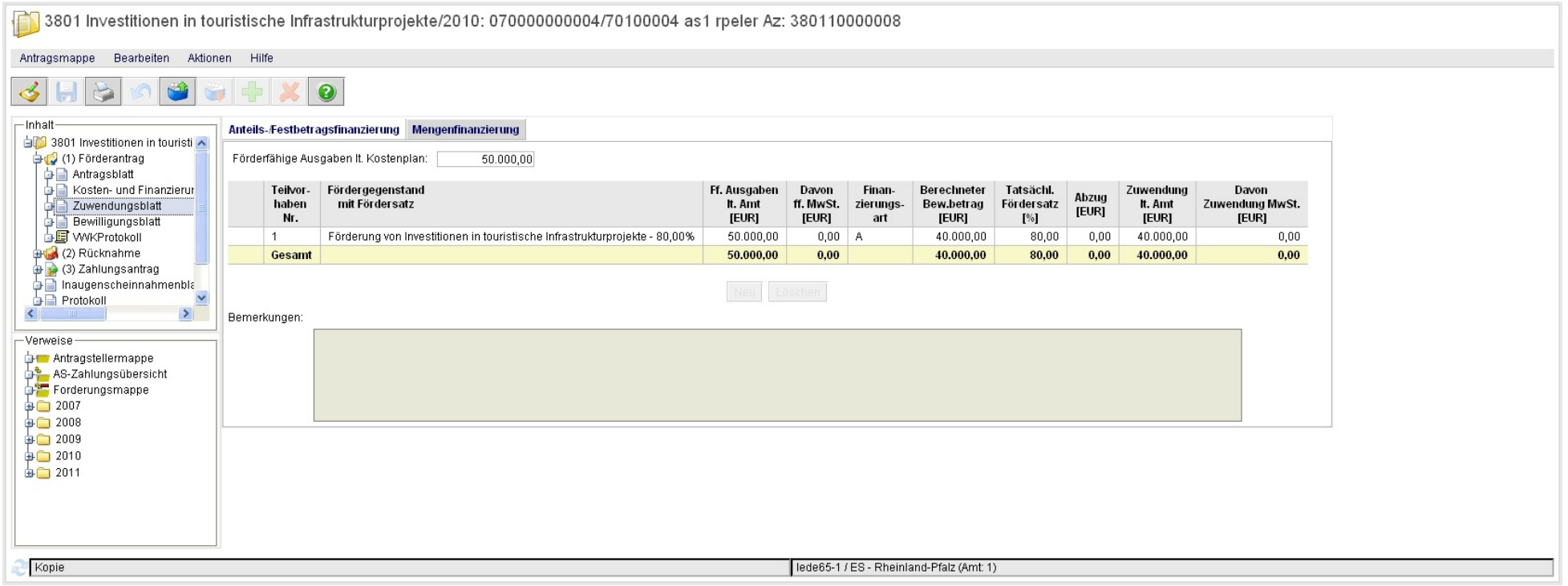

Abbildung 2.1: Web-Client: Zuewndungsblatt [deG07]

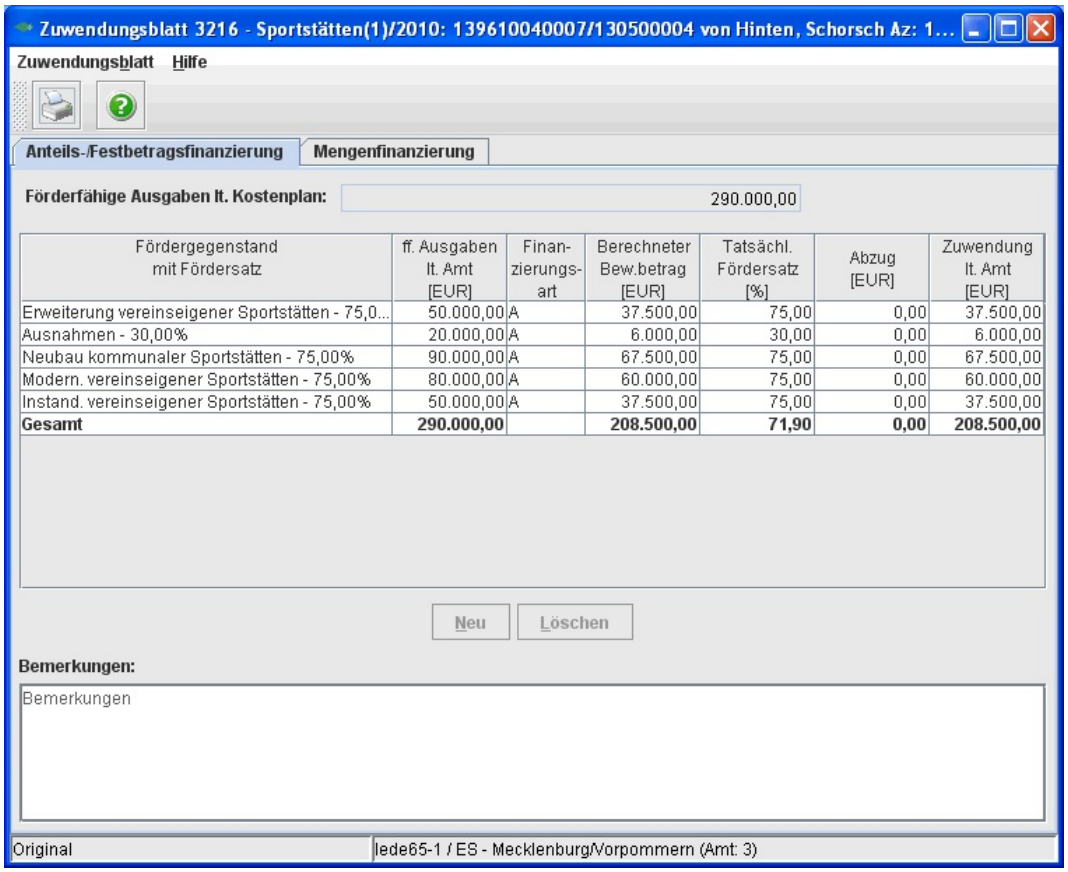

Abbildung 2.2: Standalone-Client: Zuewndungsblatt [deG07]

2.2 Umsetzung der Benutzerschnittstellen für mehreren Plattformen in der deg (Ist-Zustand)

Für die Umsetzung der primären Anforderung wäre es möglich gewesen für den Web-Client und dem Standalone-Client separate GUIs mit unterschiedlichen Frameworks zu entwickeln. Die deg hat jedoch eine Lösung erarbeitet mit der es möglich ist, ein einmal beschriebenes GUI auf mehrere Plattformen zu protieren. Das reduziert den Aufwand zur Entwicklung neuer GUIs, durch eine höhere Abstraktion. Zugleich fördert die einmalige Beschreibung auch eine ähnlichen Aufbau der GUI im Web- und Standalone-Client. Die Lösung der deg ist das *Multichannel-Framework* (MCF). Die Architektur des Multichannel-Frameworks ist Abbildung 2.3 zu entnehmen. Innerhalb dieses Frameworks werden die GUIs mittels so genannter *Präsentationsformen* beschreiben. Aus Präsentationsformen mithilfe der *Component-Factories* GUIs erzeugt werden, die auf unterschiedlichen Frame-

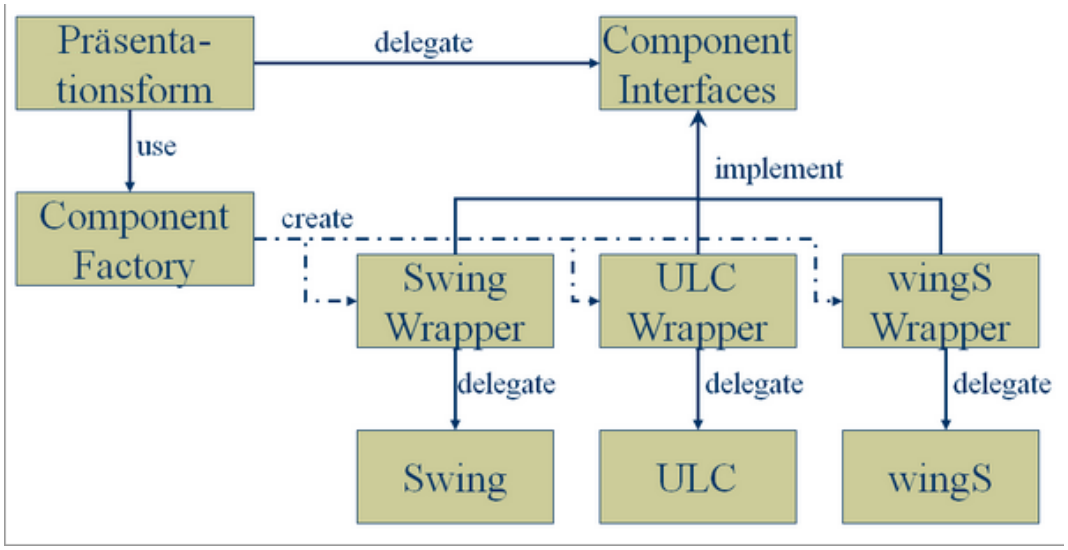

Abbildung 2.3: Architektur des Multichannel-Frameworks [Maa07]

works basieren[4] und das *Component-Interface* implementieren. Das *Component-Interface* wird für die Interaktion mit den Komponenten der unterschiedlichen Frameworks benötigt. Mit dem MCF ist die deg in der Lage ihre GUIs für das *Swing*[5] -Framework und für das *wingS*[6] -Framework mit nur einer GUI-Beschreibung zu erzeugen.

2.3 Probleme des Multichannel-Frameworks

Beim Einsatz des MCF treten jedoch Probleme auf. Das erste Problem bezieht sich auf die integrierten Framworks (Swing und wingS). Beide Frameworks sind verwaltet und werden nicht mehr gewartet. Um auch in der Zukunft den Anforderungen der Kunden nachkommen zu können müssten beide Frameworks von den Entwicklern der deg selbst weiterentwickelt werden. Eine andere Möglichkeit wäre es, wenn die deg andere und modernere Frameworks einsetzt um den nötigen Support der Framework-Entwickler nutzen zu können.

Das MCF ist in der Theorie so konzipiert, dass es leicht sein sollte neue Frameworks zu integrieren (siehe Abbildung 2.3. In der Praxis wurde die Einfachheit einer solchen Integration jedoch widerlegt. Ein Problem, wel-

[4]Hier: Swing, ULC und WingS. Wobei ULC bei der deg nicht mehr im Einsatz ist.
[5]Siehe Glossar: Swing
[6]Siehe Glossar: wingS

ches bei der Integration neuer Frameworks aufkommt, ist, dass sich das MCF sehr stark an Swing orientiert und die GUIs vor allem vom GridBagLayout[7] stark beeinflusst sind. Ein solches Layout steht nicht in allen Frameworks zur Verfügung. Da die Beschreibung der GUI über ein solches Layout vollzogen wird, ist es der Umgang mit dem GridBagLayout[8] innerhalb des Frameworks eine Voraussetzung für die Intergration in das MCF. Zusammenfassen sind folgende Probleme des MCF zu nennen:

- verwendeten Frameworks sind inaktuell
- Starke Orientierung an Swing

2.4 Zielsetzung

Das langfristige Ziel der deg bzgl. des MCF ist es, eine Lösung zu entwickeln, welche das MCF ablösen kann. Anzustreben ist eine Lösung, die neben den primären Anforderungen die sekundären Anforderung besser umsetzt als das MCF.
In dieser Arbeit wird ein Ansatz unterschucht, bei dem es möglich ist, die oben genannten sekundären Anforderungen besser umzusetzen. Kern des Ansatzen ist eine DSL, mit deren Hilfe die UIs beschrieben werden sollen. Eine DSLs kann so konzipiert werden, dass sie ausreichend abstrakt, erweiterbar und ausdruckstärker ist als das MCF. Die primären Anforderungen dürfen dabei nicht außer Acht gelassen werden.
Die genaue Lösungsidee mittels DSL, welche in dieser Arbeit verfolgt wird, ist in Kapitel 4 beschrieben.

[7] Siehe Glossar: GridBagLayout
[8] Siehe Glossar: GridBagLayout

Kapitel 3

Domänenspezifische Sprachen

3.1 Begriffsbestimmungen

Sprache/Programmiersprache

Rein formal betrachtet ist eine Sprache ist eine beliebige Teilmenge aller Wörter über einem Alphabet. *Ein Alphabet ist eine endliche, nichtleere Menge von Zeichen (auch Symbole oder Buchstaben genannt)* [Hed12, S.6]. Zur Verdeutlichung der Definition einer Sprache sein V ein Alphabet und $k \in N$ [1]. *Eine endliche Folge* $(x_1, \ldots, x_k)$ *mit* $x_i \in V (i = 1, \ldots, k)$ *heißt Wort über* V *der Länge* k [Hed12, S.6].

Bei Programmiersprachen grenzt die Bestandteile einer Sprache wie folgt ab:

abstrakte Syntax

Die abstrakte Syntax ist eine Datenstruktur, welche die Kerninformationen eines Programms beschreibt. Sie enthält keinerlei Informationen über Details bezüglich der Notation. Zur Darstellungs dieser Datenstruktur werden abstrakte Syntaxbäume genutzt. [VBK+13, S.179].

[1] N ist die Menge der natürlichen Zahlen einschließig der Null. [Hed12, S. 6]

konkrete Syntax

Die konkrete Syntax beschreibt die Notation der Sprache. Demnach bestimmt sie, welche Sprachkonstrukte der Nutzer einsetzen kann, um ein Programm in dieser Sprache zu schreiben. Die konkrete Syntax wird in so genannten Parse-Bäumen (konkrete Syntaxbäume) dargestellt. [Aho08, S.87]

statische Semantik

Die statische Semantik beschreibt die Menge an Regeln bezüglich des Typ-Systems, die ein Programm befolgen muss. [VBK+13, S.26]

ausführbare Semantik

Die ausführbare Semantik ist abhängig vom Compiler. Sie beschreibt wie ein Programm zu seiner Ausführung funktioniert. [VBK+13, S.26]

Programmiersprachen werden dazu verwendet, um mit einem Computer Instruktionen zukommen zu lassen. [FP11, S.27] [VBK+13, S.27]

General Purpose Language (GPL)

Bei GPLs handelt es sich um Programmiersprachen, die Turing-vollständig sind. Das bedeutet, dass mit einer GPL alles berechnet werden kann, was auch mit einer Turing-Maschine[2] berechenbar ist. Völter et. Al. behaupten, dass alle GPLs aufgrund dessen untereinander austauschbar. Dennoch sind Abstufungen bei der Ausführung dieser Programmiersprachen zu machen. Unterschiedliche GPL sind für spezielle Aufgaben optimiert. [VBK+13, S.27f]

[2] Siehe Glossar: Turing-Maschine

Domain Specific Language (DSL)

Eine DSL ist eine Programmiersprache, welche für eine bestimmte Domain[3] optimiert ist. [VBK+13, S.28] Das Entwickeln einer DSL ermöglicht es, die Abstarktion der Sprache der Domäne anzupassen. [gho11, S.10] Das bedeutet, dass Aspekte, welche für die Domäne unwichtig sind, auch von der Sprache außer Acht gelassen werden können (Abstraktion). Die Semantik und Syntax sollten dieser Abstraktionebene angepassen sein. Darüber hinaus sollte ein Programm, welches in einer DSL geschrieben wurde, alle Qualitätsanforderungen erfüllen, die auch bei einer Umsetzung des Programms mit anderen Programmiersprachen realisiert werden. [gho11, S.10f] Eine DSL ist demnach in ihren Ausdrucksmöglichkeiten eingeschränkt. Je stärker diese Einschränkung ist, desto besser ist die Unterstützung der Domäne sowie die Ausdruckskraft der DSL. [FP11, S.27f] In machen Fällen findet eine Unterscheidung zwischen technischen DSLs und fachlichen DSL statt. Markus Völter unterscheidet diese beiden Kategorien im Allgemeinen dahingehend, dass technische DSLs von Programmierern genutzt werden und fachliche DSL von Personen, die keine Programmierer sind (bspw. Kunden bzw. Personen, die sich in der Domäne auskennen). [VBK+13, S.26]

Grammatik

Grammatiken und insbesondere Grammatikregeln können dazu verwendet um Sprachen zu beschreiben und somit auch den Aufbau eines Computerprogramms. [Hed12, S.23f] Für die Definition einer Grammatik verweise ich auf den Praxisbericht [Gun14, S.5ff]. Grammatiken können in einer Hierachrie dargestellt werden (*Chomsky-Hierarchie*). [, S.32f] Bei Programmiersprachen handelt es sich dabei um *kontextfreie Sprachen*, da diese Sprachen entscheidar sind und somit von einem Compiler[4] verarbeitet werden. [Hed12, S. 16f]

[3] Siehe Glossar: Domäne

[4] Siehe Glossar: Compiler

Parser

Ein Parser ist ein Teil der Infrastruktur der DSL. [gho11, S.211] Er ist dafür verantworlich aus dem DSL-Script ein Output zu generieren, mit dem weitere Aktionen durchgeführt werden können. [gho11, S.212] Der Output wird in Form eines Syntax-Baums (Parse-Baum) (AST[5]) generiert. [FP11, S.47] Ein solcher Baum ist laut Martin Fowler eine weitaus nutzbarere Darstellung dessen, was mit dem DSL-Script dargestellt werden soll. Daraus lässt sich auch das semantische Model generieren. [FP11, S.48]

Semantisches Model

Das semantische Model ist eine Repräsentation dessen, was mit der DSL beschrieben wurde. Es wird laut Martin fowler auch als das Framework oder die Bibliothek betrachtet, welche von der DSL nach außen hin sichtbar ist. [FP11, S.159] In Anlehnung an Ghosh ist das semantische Model mit dem AST gleichzusetzen, der durch eine laxikalische Analyse des DSL-Scripts mithilfe eines Parsers erzeugt wird. Somit wird es als Datenstruktur betrachtet, dessen Struktur von der Syntax der DSl unabhängig ist [gho11, S.214]. Das Gleichsetzen des semantischen Models mit dem AST ist laut Martin Folwer in den meisten Fällen nicht effektiv. Grund dafür ist, dass der AST sehr stark an die Syntax der DSL gebunden, wohingegen das semantische Model von der Syntax unabhängig ist. [FP11, S.48]

[5]Siehe Glossar: Abstrakter Syntax Baum

Generator

In Anlehnung an Martin Fowler ist ein Generator ist ein Teil der einer DSL Umgebung[6] , der für das Erzeugen von Quellcode für die Zielumgebung[7] zuständig ist. [FP11, S.121] Bei der Generierung von Code wird zwischen zwei Arten unterschieden.

Transformer Generation

Bei der Transformer Generation wird das semantische Model als Input verwendet, woraus Quellcode für die Zielumgebung generiert wird. [FP11, S.533f] Eine solte Generation wird oft verwendet, wenn ein Großteil des Output generiert wird und die Inhalte des semantischen Models einfach in den Quellcode der Zielumgebung überführt werden können. [FP11, S.535]

Templated Generation

Bei der Templated Generation wird eine Vorlage benötigt. In dieser Vorlage befinden sich Platzhalter, an deren Stelle der Generator speziellen Code generiert. [FP11, S.539f] Diese Art der Codegenerierung wird oft verwendet, wenn sich in der generierte Quellcode für die Zielumgebung viele statische Inhalte befinden und der dynamisch generierte Anteil sehr einfach gehalten ist. [FP11, S.541]

[6] Siehe Glossar: DSL Umgebung
[7] Siehe Glossar: Zielumgebung

3.2 Anwendungsbeispiele

Die Anwendungsbereiche für DSLs sind sehr unterschiedlich. Die bekanntesten DSL sind Sprachen wie *SQL* (zur Abfrage und Manipulation von Daten in einer realtionalen Datenbank), *HTML* (als Markup-Sprache für das Web) oder *CSS* (als Layoutbeschreibung). [gho11, S.12] Alle Sprachen besitzen eine eingeschärkte Ausdrucksmöglichkeiten und sind von der Abstraktion her direkt auf eine Domäne (jeweils dahinter in Klammern genannt) zugeschnitten. [gho11, S.12f]
Weitere Beispiele für DSL befinden sich im Bereich der Sprachen für Parser-Generatoren (*YACC*, *ANTLR*) oder im Bereich der Sprachen für das Zusammenbauen von Softwaresystemen (*Ant*, *Make*). [gho11, S.12]
Für den Bereich der UI-Entwicklung gibt es ebenfalls Anwendungsbeispiele. Diese werden in Kapitel 5 genauer beleuchtet.

3.3 Model-Driven Software Development (MDSD)

In der Einleitung wurde schon der Model-Driven Ansatz in Verbindung mit UI-Entwicklung erwähnt. Dieser Ansatz versucht den technischen Lösungen der IT-Industrie einen gewissen Grad an Agilität zu verleihen. [SKNH05] Das ist damit verbunden, dass die Produktion von Softwareprodukten schneller und besser von statten geht und mit weniger Kosten verbunden ist. [DM14, S.71] Erreicht wird dies indem die Modelle formaler, strenger, vollständiger und konsistenter beschrieben werden. [VBK+13, S.31] Die Kernidee ist, dass die Modelle Quellcode oder Funktionalitäten beschreiben und diese in der Evolution der Software immer wiederverwendet werden können. [DM14, S.72] Somit wird wiederkehrender oder schematischer Quellcode vermieden und es ist möglich diese Modelle auch in anderen Anwendungen zu verwenden. [DM14, S.71] Daraus lassen sich folgende Ziele des MDSD ableiten:

- schnelleres Entwicklen durch Automatisierungen
- bessere Softwarequalität durch automatisierte Transformationen (Generation) und formalen Model-Definitionen

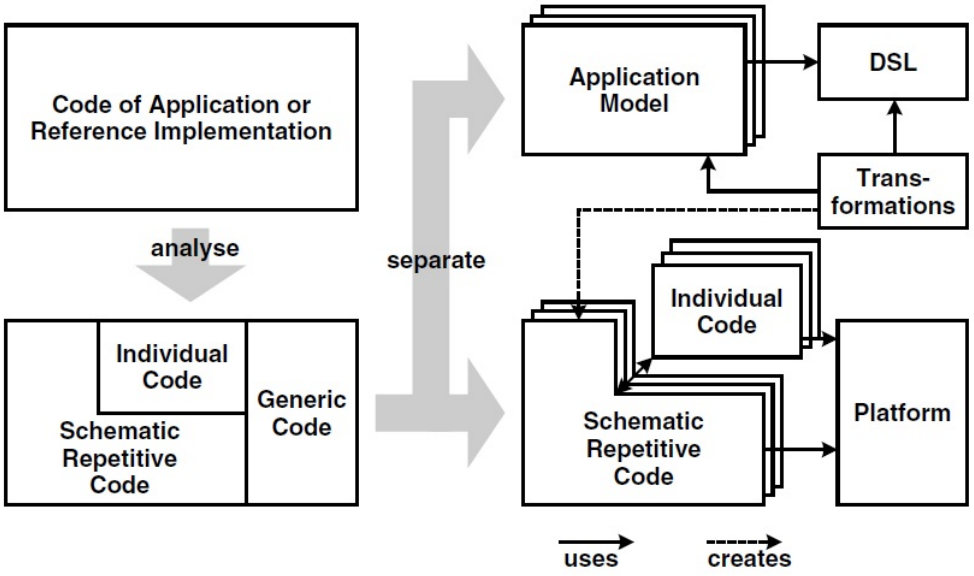

Abbildung 3.1: Die grundlegenden Ideen hinter dem MDSD [mds06, S.15]

- Verhinderung von Wiederholungen und besseres Management von veränderbare Technologien durch die separierung der Gegenstandsbereiche.....
- Bessere Wiederverwendung von Architekturen, Modelierungssprachen[8] und Generatoren/Transformatoren
- Verringerte Komplexität durch höhere Abstraktion

[mds06, S.13f] Die Modelle sind somit nicht länger nur zur Dokumentation geeignet. Sie sind Teil der Software. [mds06, S.14f] Die Modelle sind dabei auf ein bestimmtes Domänenproblem angepasst. Um diese Modelle zu beschreiben wird eine DSL benötigt. [mds06, S.15] In Abbildung 3.3 ist die Idee des MDSD schematisch dargestellt.

3.4 Abgrenzung zu GPL

Wie zu Beginn dieses Kapitels schon erwähnt sind GPLs. Sprachen mit denen alles Berechnet werden kann, was auch mit einer Turing-Maschine berechenbar ist. [VBK+13, S.27] Folglich kann mit einer GPL jedes berechenbare

[8] bspw. eine DSL

Problem gelöst werden. Eine DSL hat diese Eigenschaft nicht. Da sie auf eine bestimmte Domäne zugeschnitten ist, können auch nur Probleme innerhalb dieser Domäne mit ihr gelöst werden. [VBK+13, S.28] Martin Fowler bezeichnet diese Eigenschaft des Domänen Fokus als ein Schlüsselelement der Definition einer DSL. [FP11, S.27f] Allein aus dieser Abgrenzung lassen sich bereits Vor- und Nachteile von DSLs oder GPLs ableiten. Diese werden im folgenden Kapitel genauer betrachtet.

3.5 Vor- und Nachteile von DSL gegenüber GPL

Gründe weshalb man eine DSL nutzen sollte, gibt es viele. Allerdings dürfen die Herausforderungen, die aufkommen noch bevor man eine DSL einsetzen kann, nicht außer Acht lassen. Bei der Erklärung der Vorteile wird davon ausgegangen, dass eine DSL zum Einsatz bereit steht. Bei den Nachteilen hingegen wird i.d.R davon ausgegangen, dass eine solche DSL erst noch entwickelt werden muss.

Vorteile

- **Verbesserte Produktivität bei der Entwicklung der Software** [FP11] [VBK+13] [gho11]
- **Bessere Kommunikation mit Domänen-Experten** [FP11] [VBK+13] [gho11]
- **Veränderung des Ausführungskontextes** [FP11]
- **Alternativ berechenbares Model** [FP11]
- **Höhere Qualität** [VBK+13]
- **Einfachere Validierung und Verifizierung** [VBK+13]
- **Einfache Migration** [VBK+13]
- **Plattform unabhängig** [VBK+13]
- **Ausdruckskraft** [gho11] [FP11]

- **Skalierbarkeit** [gho11] Wobei Völter das eher als Gefahr sieht.

Nachteile

- **Sprach-Cacophony** [FP11] [gho11]
- **Kosten für die Entwicklung der DSL** [FP11] [VBK+13] [gho11]
- **Ghetto Slang** [FP11]
- **Blinkered Abstraktion** [FP11]
- **Großes Know-How gefordert** [VBK+13] [gho11]
- **Investitions-Gefängnis** [VBK+13]
- **Unvollständige DSLs** [VBK+13]
- **Kulturelle Herausforderungen** [VBK+13] [gho11]

Zusammenfassend ist der Aufwand für die Vorbereitung des Einsatzes einer DSL sehr hoch. Wurde eine DSL jedoch eingeführt, wird sich der Arbeitsaufwand um ein Vielfaches verringern.

3.6 Interne DSL

Bei einer internen DSL handelt es sich um eine DSL, die in eine GPL integriert sind. Sie übernehmen dabei das Typ-System der GPL in die sie integriert sind. [VBK+13, S.50] In Betrachtung der Ziele auf Kapitel 3.3 können einige dieser mit Applikation Interfaces (API) erreichen werden. Diese APIs werden i.d.R. für GPLs zur Verfügung gestellt. Martin Fowler sieht den großen Unterschied zwischen API und DSL darin, dass das eine DSL neben einem abstrahierten Vokabular auch eine spezifische Grammatik nutzt. [FP11, S.29] Zwischen einer internen DSL und einem API steht das *Fluent Interface* ([VBK+13, S.50]), welches als Implementierungstechnik von internen DSL im Folgenden genauer beleuchtet wird.

3.6.1 Implementierungstechniken

Fluent Interfaces

Parse-Tree Manipulation

Annotationen

3.7 Externe DSL

3.7.1 Implementierungstechniken

Bei den Implementierungstechniken von externen DSL geht es um die Art und Weise, wie der DSL-Code vom Parser in ein semantisches Model oder einem AST überführt wird. [FP11, S.89]

Parser Generator

Parser Kombinator

Vermischung mit anderen Sprachen

3.8 Nicht-Textuelle DSL

Bei den in den letzten Kapiteln vorgestellten internen und externen DSLs handelt es sich um textuelle DSLs. Auch wenn eine DSL eine bestimmte domäne repräsentiert, bedeutet dies nicht, dass diese Repräsentation immer textuell erfolgen muss. [gho11, S.19] Es gibt einige Gründe, mit einer nicht-textuellen DSL zu arbeiten:

- Viele Domänenprobleme können durch die Domänen-Nutzer besser durch Tabellen oder grafischen Darstellungen erklärt werden
- Domänenlogik ist in textueller Form oft zu komplex und enthält zu viele syntaktische strukturen
- visuelle Modelle sind einfacher zu durchdringen und zu verändern durch Domänenexperten

[gho11, S.19] Für diesen Ansatz muss der Domänen-Nutzer die Repräsentation des Wissens über eine Domäne in einem Editor (Projection Editor)

visualisieren. Mit diesem Editor kann der Domänen-Nutzer die Sicht auf die Domäne verändern, ohne auch nur eine Zeile code schreiben zu müssen. Im Hintergrund generiert dieser Editor den Code, welcher Sicht auf die Domäne modelliert. [gho11, S.19f]

Kapitel 4

Entwicklung einer Lösungsidee

4.1 Allgemeine Beschreibung der Lösungsidee

Eine Lösungsidee für die in Kapitel 2.3 beschriebenen Probleme wurde im Kapitel 2.4 bereits angedeutet. Es geht um die Nutzung einer DSL zur Beschreibung von GUIs. Diese GUIs sollen durch die DSL so beschrieben werden, dass sie in der Domäne von profil c/s für unterschiedliche UI-frameworks genutzt werden kann. Damit kann das MCF langfristig betrachtet abgelöst werden.

4.2 Architektur

In diesem Lösungsansatz ist die DSL der Ausgangspunkt. Die abstrakte Beschreibung der GUI wird über die DSL vorgenommen. Ein Generator kann nach dem parsen für diese Beschreibung der GUI frameworkspezifischen Code generieren. Somit ist die Integration neuer Frameworks an die Implementierung eines spezifischen Generators gekoppelt. Abbildung 4.1 zeigt die Architektur für diesen Ansatz auf. Dabei wurden exemplarisch drei unterschiedliche Generatoren für unterschiedliche Frameworks mit aufgenommen.
Somit wird erreicht, dass die GUI weiterhin nur einmal beschrieben werden muss.

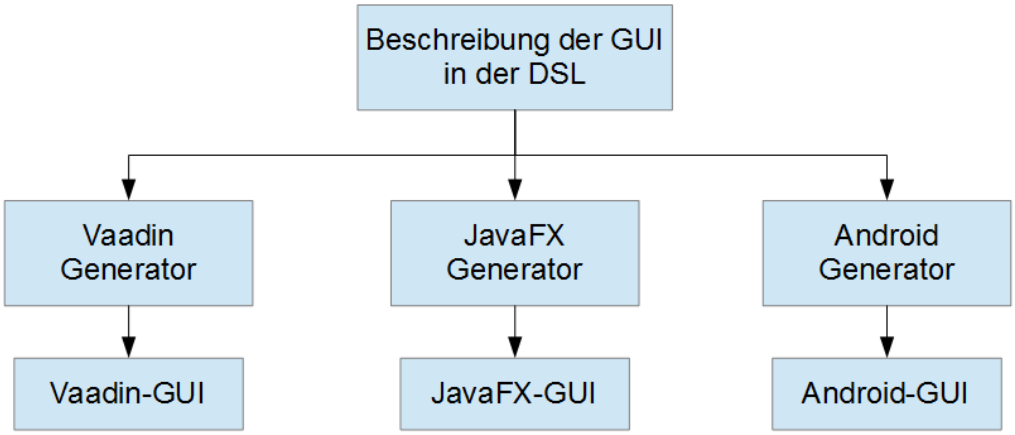

Abbildung 4.1: DSL-Ansatz für gleich GUIs auf unterschiedlichen Plattformen

4.3 Vorteile gegenüber dem Multichannel-Framework

Wie in Kapitel 2.3 erläutert, weißt das MCF einige Probleme auf. Mit dem neuen Ansatz kann das Problem der inaktuellen Frameworks und das Problem der starken Orientiertung an Swing (oder an ein anderes Framework) beseitigt werden. Eine DSL sollte sich nicht an Besonderheiten bestehender Frameworks orientieren, sondern an dem Domänenproblem. [mds06, S.15] Von daher ist sichergestellt, dass die Integration von unterschiedlichen Frameworks gleicher Maßen gut funktioniert.

Ein weiterer Vorteil ist, dass durch die wegfallende Orientiertung an Swing auch die Beschreibungsform ausdrucksstärker wird. Grund dafür ist, dass die syntaktischen Strukten, die in Swing vorhanden sind, nicht mehr benötigt werden.

Dazu kommt, dass fachliche Konzepte zur Beschreibung der UIs benutzt werden können und die Umsetzung auf technischer Ebene von den Generatoren übernommen wird.

Kapitel 5

GUI-DSL

5.1 Beschreibung der Anforderung an die GUI-Beschreibung

Die allgemeinen Anfroderung an die GUI wurden in Kapitel 2.1 erläutert. Die folgenden Anforderungen beziehen sich auf die Aspekte der GUI die beschrieben werden müssen, da sie von den Entwicklern im Verlauf der Zeit geändert werden müssen, oder es nicht sinnvoll ist diese zu abstrahieren, da keine Wiederverwendung stattfindet. Bei der Beschreibung der Anforderungen muss darauf eingegangen werden, welche Elemente der GUI mit anderen Elementen kommunizieren können und wie die GUI beschrieben wird.

Traditionell werden GUIs mit Hilfe von Layout-Containern strukturiert. In der Vergangenheit hat sich gezeigt, dass die strukturierung über ein spezifisches Layout zu einer Orientierung an ein bestimmtes Framework führt. Von daher ist das Layout in der GUI-Beschreibung soweit zu abstrahieren, dass die Grundsturktur[1] von Generator bestimmt wird.

Triviale Elemente einer GUI[2] , die für profil c/s relevant sind, müssen beschrieben werden können. Welche Attribute dieser Elemente beschrieben werden, wird in Kaptiel 8 genauer analysiert. Im Gegensatz dazu steht, dass die GUI durch die Beschreibung fachlicher Modelle automatisch generiert wird und somit keine Elemente einer GUI beschrieben werden müssen (Mo-

[1] Das zu Grunde liegende Layout

[2] Siehe Glossar:

tivation für MDSD [SKNH05]). Jedoch soll den Entwicklern weiterhin die Möglichkeit gegeben werden, die UIs selbst zu entwerfen. Grund dafür ist, dass es ein zu großer Aufwand wäre alle Module von profil c/s auf MDSD umzustellen und die GUIs generieren zu lassen. Nur die Beschreibung der GUI umzustellen scheint überschaubar zu sein.
Weiterhin müssen auch komplexe Elemente einer GUI beschrieben werden können. Unter komplexen GUI-Elementen in profil c/s versteht man vordefinierte Zusammensetzungen von trivialen GUI-Elementen, die für einen bestimmten Input einen entsprechenden Output auf der Benutzerschnittstelle erzeugen. Die Verarbeitung des Input ist dabei bereits an die Domäne angepasst. An diesem Punkt kann die vorherige Anforderung wiederum angefochten werden, da komplexe GUI-Elemente auf einer höheren Abstraktionsebene liegen als triviale GUI-Elemente. Die Anzahl der komplexen GUI-Elemente für profil c/s reicht jedoch nicht, um alle Benutzerschnittstellen für profil c/s zu beschreiben. Außerdem ist die Wiederverwendbarkeit einiger komplex GUI-Elemente (gemäß dem Fall, dass eine Vielzahl solcher Elemente bestünde und sich die Benutzerschnittstellen von profil c/s damit abbilden ließen) meiner Meinung nach fragwürdig.
Da bereits Ansätze von DSLs zur Beschreibung von GUIs existieren muss geprüft werden, ob sie die oben beschreibenen Anforderungen umsetzen können. Dazu werden im folgenden Kapitel drei ausgewählte DSLs vorgestellt und im nachfolgenden Kapitel hinsichtlich der Anforderungen bewertet.

5.2 Vorstellung ausgewählter DSLs zur Beschreibung von GUIs

5.2.1 The Snow

5.2.2 glc-dsl

5.2.3 Sculptor

5.3 Bewertung

Anforderung	The Snow	glc-dsl	Sculptor
keine Layout-Spezifikation			
Beschreibung trivialer Elemente			
Beschreibung komplexer Elemente			

Tabelle 5.1: Bewertung ausgewählter DSLs zur Beschreibung von GUIs

Kapitel 6

Evaluation des Frameworks zur Entwicklung der DSL

6.1 Vorstellung ausgewählter Frameworks

Zur Umsetzung der DSL und der Generatoren wird ein Framework benötigt, welches dafür notwendige Funktionalitäten bereit stellt. Hierzu werden die Frameworks *PetitParser*, *Xtext* und *MPS* kurz vorgestellt und im Anschluss verglichen.

6.1.1 PetitParser

6.1.2 Xtext

6.1.3 MPS

6.2 Vergleich und Bewertung der vorgestellten Frameworks

Kapitel 7

Aufteilung der Anforderungen auf Sprache und Generator

7.1 Anforderung an die neue DSL

7.2 Anforderung an den Generators

Kapitel 8

Entwicklung einer DSL zur Beschreibung der GUI in profil c/s

8.1 Analyse der Metadaten der GUI

8.2 Syntax

8.3 Semantik

Kapitel 9

Entwicklung des Generators für das Generieren von Klassen für das Multichannel-Framework

9.1 WAM-GUI Architektur

9.2 Syntax und Semantik für die Beschreibung der GUIs

9.3 Umsetzung des frameworkspezifischen Generators

Kapitel 10

Zusammenfassung und Ausblick

Titel anhang a

Glossar

Förderantrag *[...] ist ein Antrag, den der Begünstigte einreicht, wenn er sich eine Maßnahme fördern lassen möchte* [dat14]. 5

GridBagLayout ist ein Layout Manager innerhalb von Swing, welcher die Komponenten horizontal, vertical und entlang der Grundlinie anordnet. Dabei müssen die Komponenten nicht die gleiche Größe haben [Oraa]. 9

GUI ist die Schnittstelle zwischen dem Benutzer und dem Programm. 1

InVeKoS ist die Abkürzung für Integriertes Verwaltungs- und Kontrollsystem. Mit einem solchen Sysmten wird im allgemeinen sichergestellt, dass die durch den Europäischen Garantiefonds für die Landwirtschaft finanzierten Maßnahmen ordnungsgemäß umgesetzt wurden. Im speziellen bedeutet dies die Absicherung, von Zahlungen, die korrekte behandlung von Unregelmäßigkeiten und das wieder Einziehen von zu unrecht gezahlter Beiträge [Gen14]. 5

Swing ist ein UI-Framework für Java Applikationen [Orab]. xiii, 8, 9

Traditionelle UI-Entwicklung Bei der traditionellen UI-Entwicklung wird mit traditionellen UI-Toolkits gearbeitet. Bei diesen Toolkits wird Aufbau der GUI genau beschrieben. Für die Interaktion mit den UI-Widgets, werden Listener implementiert, die auf andere Events reagieren, die von anderen Widgets erzeugt generiert wurden. Events können zu unterschiedlichen Zeitpunkten generiert werden und es wird nicht festgelegt in welcher Reihenfolge sie bei anderen Widgest ankommen. [KB11]. 1

Usability beschreibt die Nutzerfreundlichkeit einer GUI, sowie auch die Nutzerfreundlichkeit einer Software. 1

. 8, 9

Zuwendungs-Berechner ist ein Werkzeug innerhalb von profil c/s. *Mit diesem Werkzeug kann der Sachbearbeiter die Zuwendung, die dem Antragsteller bewilligt werden soll, nach einem standardisierten Verfahren berechnen (siehe Abschnitt "Algorithmen"). Das Ergebnis wird im Zuwendungsblatt dokumentiert, das auch später mit demselben Werkzeug angesehen werden kann* [deG07]. xiv

Zuwendungsblatt ist die grafische Dokumentation der Ergebisse des Zuwendungs-Berechners innerhalb von profil c/s. xiv, 5

Literaturverzeichnis

[Aho08] AHO, ALFRED V: *Compiler: Prinzipien, Techniken und Werkzeuge*. Pearson Studium, 2008.

[dat14] DATA EXPERTS GMBH: *Förderantrag*. Profil Wiki der deg, März 2014. Zuletzt eingesehen am 02.12.2014.

[deG07] GMBH DATA EXPERTS: *Detailkonzept ELER/i-Antragsmappe*, Januar 2007. Letzte Änderung am 01.12.2014.

[DM14] DANIEL, FLORIAN und MARISTELLA MATERA: *Model-Driven Software Development*. In: *Mashups*, Data-Centric Systems and Applications, Seiten 71–93. Springer Berlin Heidelberg, 2014.

[FP11] FOWLER, MARTIN und REBECCA PARSON: *Domain-Specific Languages*. Addison-Wesley, 2011.

[Gen14] GENERALDIREKTION LANDWIRTSCHAFT UND LÄNDLICHE ENTWICKLUNG: *Das Integrierte Verwaltungs- und Kontrollsystem (InVeKoS)*. URL: http://ec.europa.eu/agriculture/direct-support/iacs/index_de.htm, November 2014. Zuletzt eingesehen am 02.12.2014.

[gho11] *DSLs in Action*. Manning Publications Co., 2011.

[Gun14] GUNDERMANN, NIELS: *Entwicklung einer Grammatik für eine DSL mit xText am Beispiel einer Sprache zur Definition von Pflichtprüfungen in profil c/s*, 2014. Praxisbericht.

[Hed12] HEDTSTUECK, ULRICH: *Einführung in die Theoretische Informatik*, Band 5. Auflage. Oldenbourg Verlag, 2012.

[KB11] KRISHNASWAMI, NEELAKANTAN R. und NICK BENTON: *A Semantic Model for Graphical User Interfaces*. Microsoft Research, September 2011. Verfügar unter URL:.

[LW] LU, XUDONG und JIANCHENG WAN: *Model Driven Development of Complex User Interface*. Technischer Bericht, Shandong University. Verfügar unter URL: http://ceur-ws.org/Vol-297/paper7.pdf.

[Maa07] MAASS, DIRK: *JWAMMC - Das Multichannel-Framework der data-experts gmbh*. Vortrag, Dezember 2007.

[mds06] *Model-Driven Software Development*. John Wiley Sons Ltd, Februar 2006.

[MHP99] MYERS, BRAD, SCOTT E. HUDSON und RANDY PAUSCH: *Past, Present and Future of User Interface Software Tools*. Technischer Bericht, Carnegie Mellon University, September 1999. Verfügar unter URL: http://www.cs.cmu.edu/ amulet/papers/futureofhci.pdf.

[Oraa] ORACLE: *Class GridBagLayout*. URL: https://docs.oracle.com/javase/7/docs/api/java/awt/GridBagLayout.html. Zuletzt eingesehen am 02.12.2014.

[Orab] ORACLE: *Swing*. URL: https://docs.oracle.com/javase/jp/8/technotes/guides/swing/index.html. Zuletzt eingesehen am 02.12.2014.

[SKNH05] SUKAVIRIYA, NOI, SANTHOSH KUMARAN, PRABIR NANDI und TERRY HEATH: *Integrate Model-driven UI with Business Transformations: Shifting Focus of Model-driven UI*. Technischer Bericht, IBM T.J. Watson Research Center, Oktober 2005. Verfügar unter URL: http://www.research.ibm.com/people/p/prabir/MDDAUI.pdf.

[Use12] USERLUTIONS GMBH: *3 Gründe, warum gute Usability wichtig ist*. URL: http://rapidusertests.com/blog/2012/04/3-gute-

grunde-fuer-usability-tests/, April 2012. Zuletzt eingesehen am 01.12.2014.

[VBK+13] VÖLTER, MARKUS, SEBASTIAN BENZ, LENNART KATS, MATS HELANDER, EELCO VISSER und GUIDO WACHSMUTH: *DSL Engineering*. CreateSpace Independent Publishing Platform, 2013.