Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

David Ratza

Scala vs. Java: Ein kriteriengestützter Vergleich auf Basis aktueller Webtechnologien

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

David Ratza

Scala vs. Java: Ein kriteriengestützter Vergleich auf Basis aktueller Webtechnologien

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Olaf Zukunft
Zweitgutachter: Prof. Dr. Friedrich Esser

Abgegeben am 24. Oktober 2011

David Ratza

Thema der Bachelorarbeit
Scala vs. Java: Ein kriteriengestützter Vergleich auf Basis aktueller Webtechnologien
Stichworte
Java, Scala, Webframework, Lift
Kurzzusammenfassung
In dieser Arbeit werden die aktuellen Java und Scala Webtechnologien untersucht und miteinander verglichen. Dazu wird ein passendes Testszenario mit den ausgewählten Webtechnologien realisiert. Die Realisierungen werden anhand der festgelegten Vergleichskriterien untersucht und bewertet. Mit der Bewertung wird die aktuelle Situation von Scala als Webtechnologie präsentiert.

David Ratza

Title of the paper
Scala vs. Java: - A criteria supported comparison based on current web technologies
Keywords
Java, Scala, Webframework, Lift
Abstract
In this paper current Java and Scala web technologies will be explored, analysed and compared to each other. For each selected technology a web application will be implemented on the basis of a predefined testscenario. The implemented web applications will be evaluated with previously defined criterias. The evaluation will provide indication for the current position of Scala as a technology for the web.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

Kapitel 1

Einleitung

1.1 Motivation

Mit der Einführung der Programmiersprache Scala ist ein möglicher Nachfolgekandidat für die schon in die Jahre gekommene Programmiersprache Java erschienen. Die Kompatibilität mit der Java Virtual Maschine, Interoperabilität mit der Sprache Java als auch innovative Konzepte sorgen dafür, dass Scala langsam aber stetig neue Entwickler für sich gewinnt. Laut der Organisation[1], die die Entwicklung von Scala vorantreibt, ist die Programmiersprache für den produktiven Einsatz reif. Dies ist auch die Voraussetzung, um mit Java überhaupt konkurrieren zu können.

Java als Plattform hat ihre Dominanz über den Bereich der serverseitigen Sprachen erlangt. Dieser Bereich ist deshalb so spannend, weil die heutige Gesellschaft immer häufiger auf die Angebote, die das Internet bietet, zugreift. Die computerscheuen Generationen werden irgendwann mit den Facebook-, Twitter-, Google+- o.ä. Generationen ersetzt. Die Nutzerzahlen und damit die Anforderungen an die Webanwendungen steigen kontinuierlich und setzen voraus, dass die darunterliegenden Technologien sich weiter entwickeln. Dieser Technologietrieb führt zu der Annahme, dass die Technologien, die den Anforderungen nicht genügen, durch neue, bessere, schnellere oder einfachere ersetzt werden.

Aus diesem Anlass bietet es sich an, die aktuelle Situation von Scala im Vergleich zu Java zu untersuchen. Der Bereich der Web-Technologien soll dabei als Bezug dienen.

1.2 Zielsetzung

Ziel dieser Arbeit ist es einen wissenschaftlichen Vergleich im Sinne des Software-Engineerings durchzuführen, um die wesentlichen Unterschiede zwischen Java und Scala im Bereich der Web-Technologien festzustellen. Zu diesem Anlass müssen geeig-

[1] Typesafe - `www.typesafe.com`

nete Vergleichskriterien und ein Testszenario festgelegt werden. Die zur Umsetzung des Testszenarios eingesetzten Technologien sollen anhand der festgelegten Vergleichskriterien evaluiert werden. Die Ergebnisse der Evaluierung werden gegenübergestellt, um die Erkenntnisse zu den Möglichkeiten der eingesetzten Technologien zu gewinnen. Aus den gewonnenen Erkenntnissen sollen je nach Möglichkeit Schlüsse für die Zukunft von Scala im Bereich der Webtechnologien gezogen werden.

1.3 Inhaltlicher Aufbau

Der Einstieg in die Materie wird mit dem Kapitel Grundlagen ermöglicht. Nach einem kurzen Einstieg enthält dieses Kapitel einen kompakten Exkurs zum Thema Programmierparadigmen und Typsysteme. Diesem folgt die Vorstellung von Java samt Geschichte, Eigenschaften und Sprachkonzepten. Als Erweiterung dazu dient ein kleiner Abschnitt, welcher die wesentlichen Informationen zu Java Virtual Maschine enthält. Diesem folgt die Vorstellung von Scala, die nach gleichem Schema, wie die Vorstellung von Java gestaltet ist. Nachdem die beiden Sprachen vorgestellt wurden, werden mit dem Fokus auf Scala die signifikanten Sprachfeatures und Syntaxunterschiede zwischen den beiden Sprachen aufgezeigt. Im weiteren Verlauf der Grundlagen fällt der Schwerpunkt auf die Web-Technologien. Er enthält die Philosophie der Web-Frameworks als auch die Konzeption und Ausführung des Auswahlverfahrens. Die Vorstellung der ausgewählten Web-Technologien schließt das Kapitel Grundlagen ab.

Nach den Grundlagen erfolgt die Vorstellung des Szenarios, welches zum Vergleich und als Benchmarkinggrundlage benutzt wird.

Dieser Vorstellung folgt die Konzeption des Vergleichsverfahrens. Dabei wird die Vorgehensmethodik, die möglichen Vergleichskriterien, sowie die Verfahrenskonkretisierung beschrieben.

Anschließend wird im Kapitel Design die Architektur unabhängig von den eingesetzten Technologien entworfen. Darauf folgt das Kapitel Realisierung, das aus zwei Teilen, für die Java- und Scala-Realisierung, besteht.

Die Realisierungen werden im darauf folgenden Kapitel unter Berücksichtigung der festgelegten Vergleichskriterien bewertet.

Nach der Bewertung wird ein Fazit gezogen, dem ein Ausblick folgt Diese Arbeit wird anschließend mit einer Zusammenfassung abgeschlossen.

Kapitel 2

Grundlagen

2.1 Einleitung

Der Inhalt dieses Kapitels widmet sich den im Fokus dieser Arbeit stehenden Programmiersprachen und den mit ihnen verbundenen Web-Technologien. Vor der Vorstellung der Grundlagen zu den Programmiersprachen werden zwei grundlegende Aspekte aus der Programmierwelt eingeführt. Es handelt sich hier um die Programmierparadigmen und Typsysteme. Anschließend wird die Sprache vorgestellt, die im Jahr 1995 erschienen ist und momentan als meistgenutzte Programmiersprache der Welt gilt: Java. Als Nächstes werden grundlegende Informationen zu der Laufzeitumgebung geschildert, dank der Java die Plattformunabhängigkeit erreicht hat. Diesem Abschnitt folgt die Vorstellung der neuen multiparadigmen Programmiersprache, die für viele bereits jetzt als Java Nachfolger gilt: Scala. Daraufhin werden die Syntaxunterschiede zwischen den beiden Sprachen vorgestellt und die neuen Sprachfeatures, die Scala zu bieten hat präsentiert. Dieser Vorstellung folgt eine Übersicht der aktuellen, auf den vorgestellten Sprachen basierenden Web-Technologien. Anhand symbolischer Kriterien wird anschließend für jede der Sprachen ein zur Realisierung zeitgemäßer Webanwendungen geeignetes Web-Framework ausgewählt. Im Anschluss werden die ausgewählten Web-Frameworks detaillierter vorgestellt.

2.2 Paradigmen und Typsysteme

Der folgende Abschnitt soll vor dem Einstieg in die Grundlagen der in dieser Arbeit im Vordergrund stehenden Programmiersprachen zwei wichtige Aspekte skizzieren. Diese beziehen sich auf das ganze *Programmiersprachenuniversum* und werden im Vorfeld vorgestellt. Es geht hier um Programmierparadigmen und Typsysteme der Programmiersprachen. Die wichtigsten Begrifflichkeiten zu den beiden Aspekten werden nachfolgend in getrennten Abschnitten erläutert. Auf diese wird im weiteren Verlauf dieses Kapitels verwiesen.

2.2.1 Programmierparadigmen

Mit einem Programmierparadigma ist ein fundamentaler Programmierstil gemeint bzw. eine Programmiermethodik, die angewandt wird um spezielle Probleme im Bereich des Software Engineerings zu lösen. Nachfolgend werden die Programmierstile vorgestellt, die mit den in dieser Arbeit eingesetzten Programmiersprachen assoziiert werden. Die Beschreibung lehnt sich an [Pie10] und [Wik11].

Imperativ Die Struktur von imperativen Programmen ist an die Computerarchitektur angelehnt (*Von-Neuman-Architektur*). Anweisungen werden in einer Reihenfolge abgearbeitet. Dabei können Werte von Variablen gelesen und verändert werden. Es gibt Anweisungen, die benutzt werden können um Wiederholungen und bedingte Sprünge zu realisieren.

Deklarativ In deklarativer Denkweise wird der Programmcode auf eine sehr verständliche Weise formuliert. Damit wird es effektiv und *gehirngerecht* organisiert. Ein gutes Beispiel dazu ist die *SQL*[2]

Objektorientiert Der Programmcode wird in Objekten organisiert. Diese kapseln Methoden und Daten. Die Objekte dienen meistens der Abstraktion der realen Gegenstände. Dieses Paradigma fällt unter das Konzept der Deklarierung, also einer verständlicheren Formulierung.

Generisch Die generisch geschriebenen Programme bzw. Programmabschnitte werden allgemeingültig definiert ohne die Parameter im Vorfeld zu spezifizieren. Das heißt, dass sie einmal geschrieben werden und immer wieder mit verschiedenen Typen von Parametern ausgeführt werden können.

Reflexiv Dieses Paradigma erlaubt Abbildung, Untersuchung und Modifikation der Programmkomponenten zur Laufzeit des Programms. Damit kann beispielsweise eine fehlende Funktionalität zum benötigten Zeitpunkt nachträglich realisiert werden.

Funktional Die Struktur des Programms wird mit Funktionen realisiert. Dies ist eine sehr mathematische Denkweise, die u.a. keine nachträglich veränderbaren Variablen erlaubt. Die Funktionen können andere Funktionen aufrufen und werden selbst als Daten behandelt. Damit können sie an andere Funktionen übergeben werden.

Concurrent Die Realisierung von nebenläufiger Ausführung von unterschiedlichen Prozessen eines Programms wird dank diesem Paradigma möglich. Das Vorhandensein von mehreren Recheneinheiten eines Prozessors soll damit effektiv ausgenutzt werden.

[2]Structured Query Language - Eine Abfragesprache für relationale Datenbanksysteme

2.2.2 Typsysteme der Programmiersprachen

Mit dem Begriff Typsystem assoziiert man in der Welt der Programmiersprachen eine Komponente, die die Rolle des Administrators für die Typen der Variablen übernimmt. Die Hauptaufgabe ist die Überwachung und ggf. automatische Feststellung des Wertebereichs der Variablen. Die Typsysteme werden folgendermaßen klassifiziert (vgl. [Pie10]):

- **Statisch** typisiert sind Sprachen, bei denen jede Variable lebenslänglich einen Typ beibehält.
- **Dynamisch** typisiert sind die Sprachen, deren Variablen keinen festgelegten Typ haben und ihn somit beliebig oft ändern können.
- Von einem **expliziten** Typsystem spricht man, wenn der Typ der Variable unmittelbar bei ihrer Anlage definiert werden muss.
- Wenn das Typsystem in der Lage ist, den Typ der Variable selbst zu ermitteln, spricht man von einem **impliziten** Typsystem.
- Ein **sicheres** Typsystem reagiert spätestens zur Laufzeit auf die Unstimmigkeiten bei den Zuweisungen.
- in **unsicheren** Typsystemen wiederum, kann es passieren, dass eine fehlerhafte Zuweisung nicht erkannt wird und zum Programmabsturz führt. Beispielprogrammiersprache: C

2.3 Java

Die Sprache, die ihr Leben unter dem Namen *Oak*[3] begonnen hat, erschien zum ersten Mal im Jahr 1995. Zehn Jahre nach ihrer Erscheinung wurde sie zur Programmiersprache des Jahres gekürt. Den ersten Platz in dem Index[4] der populärsten Programmiersprachen der Welt hatte sie aber schon 2001 erobert. Zweimal musste sie ihn kurzzeitig an C abgeben, ansonsten war sie und ist sie bis heute die populärste Programmiersprache. Die rasche Verbreitung verdankt Java der Tatsache, dass der Fortschritt der Informatik in den Bereichen Sprach- und Compiler-Entwicklung in ihrer Entwicklungszeit viel höher war als bei den älteren konkurrierenden Sprachen wie C oder C++. Man war damit in der Lage, die bewährten Konzepte zu kombinieren, was die Sprache einfach und vor allem plattformunabhängig gemacht hat. Signifikant war auch die Tatsache, dass man aus den Fehlern der Vergangenheit lernen konnte und diese bei dem Design der neuen Sprache vermieden hat. Diese Aspekte haben unter anderem dazu beigetragen, dass Java sehr schnell viele Programmierer begeistern konnte.

[3] Objekt Application Kernel - `http://www.bergt.de/lexikon/lex/ol.php`

[4] `http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html` Stand: 10.03.2011

2.3.1 Geschichte

James Gosling[5], der Urvater von Java und damaliger Mitarbeiter von Sun Microsystems[6], sollte eine Sprache entwickeln, die für ein interaktives Fernsehsystem zum Einsatz kommen sollte. Diese sollte die Komplexität und vor allem die mangelnde Sicherheit der Pointerarithmetik der damals gängigen Programmiersprachen C und C++ beheben. Daraufhin hat Gosling das Design von Java entwickelt, den originalen Compiler und die virtuelle Maschine (siehe Abschnitt 2.4) implementiert. Bei dem Sprachdesign hat sich Gosling der hochklassigen Konzepte der damaligen Programmiersprachen (siehe Abbildung 2.1) bedient und diese elegant in Java integriert. Besonders interessant war die virtuelle Maschine. Dank ihr sollte die Sprache in der

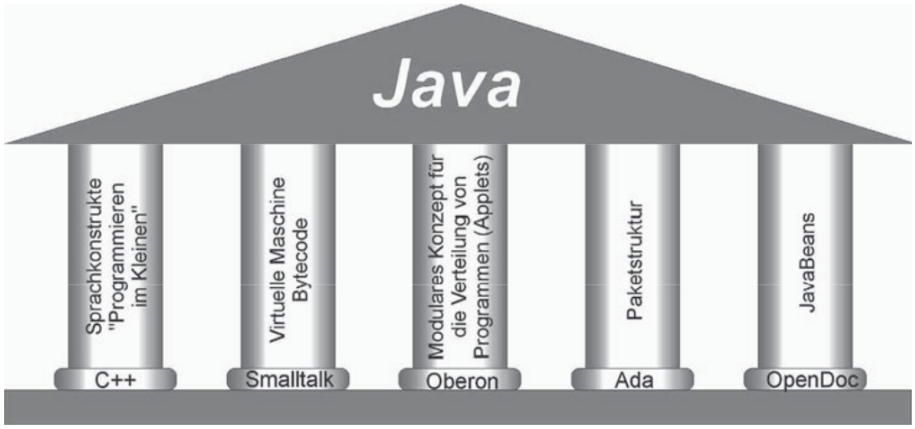


Abbildung 2.1: Die Väter von Java nach [HMHG11]

Lage sein auch andere Konsumergeräte zu bedienen, zu steuern und damit plattformunabhängig sein. Leider ist der Versuch in diesem Bereich fehlgeschlagen und nach mehreren Richtungswechseln entdeckte man die Fähigkeiten von *Oak* für das World Wide Web. Die Anforderung Programmcode über das Netzwerk empfangen zu können, ohne Schaden im Falle fehlerhafter Programme anzurichten, konnte *Oak* gerade dank der virtuellen Maschine gerecht werden. Somit war der Weg für die Ausstattung des Webs für zusätzliche Interaktionen frei. Ab diesem Zeitpunkt hat man übrigens beschlossen die Sprache Java zu nennen, da *Oak* aus Copyright-Gründen nicht verwendet werden konnte. Ihren ersten Erfolg in dem neuen Einsatzbereich hat Java mit den *Applets*[7] gefeiert. Auf diese kleine Java Applikationen konnte man in den HTML Seiten verweisen und der dazugehörige Programmcode wurde dann vom Web-Server geladen und auf dem Client (innerhalb des Web-Browsers) ausgeführt. Nach einer gewissen Zeit sind aber auch die Applets weniger interessant geworden. Dies hat aber Java in

[5] `http://en.wikipedia.org/w/index.php?title=James_Gosling&oldid=416521949`
[6] `http://en.wikipedia.org/w/index.php?title=Sun_Microsystems&oldid=416409763`
[7] Eine Verkleinerungsform von *Applikationen* [HMHG11]

weiterer Ausbreitung nicht negativ beeinflusst. Es sind neue Konzepte und Technologien entstanden, dank welchen Java im Bereich der serverseitigen Sprachen fungiert. Sie wird dazu benutzt, unternehmenskritische Anwendungen zu realisieren, die dank der Plattformunabhängigkeit möglichst nach dem Motto *write once, run anywhere* funktionieren sollen. Der Bereich der serverseitigen Ausführung mit dem Fokus auf die Webanwendungen soll auch in dieser Arbeit als Vergleichsarena dienen. Dadurch soll ermittelt werden, wie sich eine neue Programmiersprache im Vergleich zu der schon *erfahrenen* und *bewährten* schlägt und ob sie gerade in diesem Vergleich in der Lage ist diese in der Zukunft komplett zu ersetzen (vgl. [HMHG11],[Ull10],[Mü06]).

2.3.2 Eigenschaften

Java ist eine Arbeitssprache. Sie ist nicht das Material einer Doktorarbeit, sondern eine Sprache für den Job [Gos97]. Um diese Behauptung zu stützen, bietet es sich an die wesentlichen Charakteristika von Java zu skizzieren.

Einfachheit Die Reduktion des Sprachumfangs um Pointerarithmetik, Mehrfachvererbung, Operator-Overloading und Erweiterung um automatisches Speicherbereinigungsmanagement machen die Sprache für den Programmierer angenehm einfach.

Objektorientiertheit Die Kapselung von Daten und Methoden in Objekten folgt unter anderem dem Gedanken der Wiederverwendbarkeit von Softwaremodulen und der Optimierung des Verwaltungsaufwands bei großen Projekten.

Verteilbarkeit Zahlreiche Bibliotheken für Netzwerkkommunikation bieten optimale Unterstützung bei der Client-Server Programmierung.

Robustheit Ausnahmebehandlung, starke Typisierung, automatische Speicherverwaltung und Verzicht auf die Pointerarithmetik reduzieren erheblich die Wahrscheinlichkeit der Entstehung von ungewollten Systemfehlern.

Sicherheit Der Programmcode wird vor der Ausführung mehrstufig getestet. Ein Verifier überprüft den Code auf syntaktische Korrektheit und Typsicherheit. Dank dem Security Manager können wiederum Richtlinien zur Ausführung von sensitiven Operationen definiert werden.

Plattformunabhängigkeit Das kompilierte Java Bytecode kann auf jeder Rechner-Plattform ausgeführt werden, auf der eine Java Virtual Machine läuft.

Offenheit Java steht unter der GPL 2 Lizenz und ist somit Open Source. Die Implementierungen stehen jedem Interessenten offen zur Verfügung.

2.3.3 Sprachkonzepte

Java zeichnet sich durch die im Abschnitt 2.3.2 erwähnten Eigenschaften aus. Diese sind jedoch unmittelbar mit Konzepten verbunden, die man in Java realisiert bzw. integriert hat. Um Bezug auf die Abbildung 2.1 zu nehmen, hat Java ihre Syntax an C++ angelehnt, aber die lästige Pointerarithmetik entfernt. Die Idee der virtuellen Maschine hat sich im Smalltalk bewährt, der ersten puren objektorientierten Sprache. Java folgt auch dem Paradigma der Objektorientierung, aber nicht im puren Sinne. Dies ist auf Grund der Existenz von primitiven Datentypen, die dem Performancevorteil wegen nicht als Objekte designed worden sind. Von Smalltalk hat sich Java auch das Klassenbaumkonzept abgeschaut. So gibt es in der untersten Ebene der Hierarchie die Klasse `java.lang.Object`, von der alle anderen Klassen entweder direkt oder durch ihre Superklassen indirekt erben. Die Verteilung von Applets, die je nach Anfrage geladen werden, lehnt sich an Oberons Konzept der Modularisierung. Die Paketenstruktur hat man von der Programmiersprache *Ada*[8] übernommen, um Klassen zu bündeln, die inhaltlich zusammenhängen. Mit JavaBeans hat man ein Konzept der Wiederverwendbarkeit von vorgefertigten Komponenten realisiert. Diese haben anfangs den Einsatz in Form von GUI Elementen gefunden, haben sich aber auch in den anderen Bereichen bewährt. Die Idee von diesem Konzept stammt von einer offenen Dokumentenarchitektur namens OpenDoc. Diese hat es ermöglicht beliebige Dokumentteile in ihre Struktur aufzunehmen.

Im Juli 2011 erschien das lang erwartete Release von Java 7.0[9]. Seit Dezember 2006 (Erscheinung der Java SE 6) wurde die Sprache mehr gewartet als weiter entwickelt. Mit dem neuen Release wurden leider weniger spektakuläre Neuerungen vorgestellt. Dies soll sich aber mit dem, für Mitte 2013 geplanten, Release von Java 8 ändern. Java soll unter anderem das Konstrukt der Lambda Expressions bekommen, was ursprünglich für das Release 7 vorgesehen war.

Seit der release[10] Version hat man die Sprache mehr oder weniger kontinuierlich um weitere Bibliotheken und Sprachkonzepte erweitert. Es sind zu viele, um sie alle an dieser Stelle zu erwähnen, deshalb werden zum Abschluss dieses Abschnitts nur die signifikanten davon skizziert.

Generics Parametrisierte Typen, die zusätzliche Typsicherheit liefern und Eliminierung der lästigen Typcasts ermöglichen.

Concurrency Utilities Thread-safe Datenstrukturen für Programmierung von Anwendungen mit parallel laufenden Prozessen.

Annotations Möglichkeit der Einbindung von Metadaten in den Quellcode. Klassen,

[8] Ada Tools and Resources - `http://www.adapower.com/`
[9] http://www.oracle.com/us/corporate/press/444374
[10] JDK 1.0 Codename *Oak*

Methoden und Felder können für spezielle Behandlung markiert oder bestimmte Flags für den Compiler gesetzt werden.

Reflections Möglichkeit von Untersuchung, Instantiierung und Modifikation von den zur Laufzeit nicht bekannten Objektklassen.

2.4 Java Virtual Machine (JVM)

Durch die Kompilierung des Quellcodes der Programmiersprachen wird üblicherweise der Maschinencode erzeugt. Dieser wird für spezielle Rechner-Plattformen (Windows, Linux, Solaris, Mac OS) und eine bestimmte Prozessorarchitektur (z.B. x86 auch als IA-32[11] bekannt) generiert und gilt nur für diese. Eine Portierung des so kompilierten Programms ist ohne eine erneute Kompilierung für die neue Plattform und Prozessorarchitektur nicht möglich. Bei Java erfolgt keine Erzeugung des Maschinencodes, zumindest nicht direkt. Der Java Compiler optimiert und übersetzt den Java Quellcode in Bytecode (eine Art von Zwischencode). Dieser Zwischencode wird von dem Bytecode-Interpreter der abstrakten Maschine (bei Java die JVM) interpretiert und auf der Zielplattform ausgeführt (siehe Abbildung 2.2). Die virtuelle Maschine hat sich

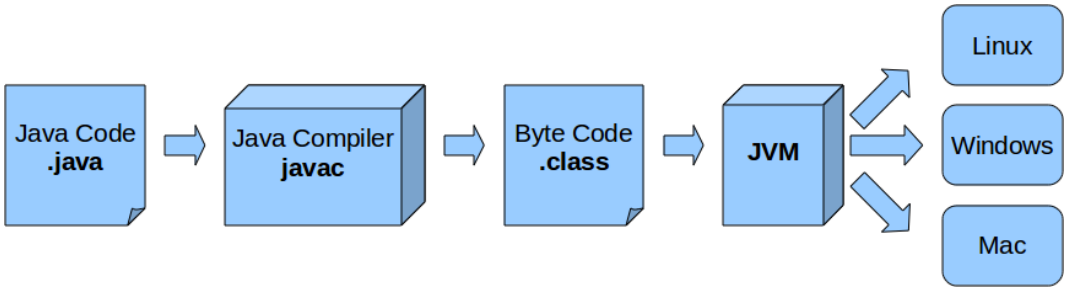


Abbildung 2.2: Ausführung des Java Programms

bekanntlich in Smalltalk sehr gut bewährt. Die Idee einer abstrakten Rechenmaschine existiert aber schon länger als Smalltalk. Die Definition der ersten abstrakten Maschine stammt von Peter J. Landin[12][Cra06]. Dieser hat Anfang der sechziger Jahre die SEDC[13] Maschine entworfen, welche die Ausdrücke des λ-Kalküls auswerten sollte. Die Idee wurde bis heute sehr oft adaptiert. Es gibt mehrere Beispiele, die vor als auch nach der JVM realisiert worden sind. Ein aktuelles Beispiel davon ist Dalvik[14], welches sich stark an der JVM orientiert. Um den historischen Werdegang der Idee in einem Ausdruck zusammenzufassen, kann man die virtuelle Maschine als elegantes abstraktes architecture-independence Pattern bezeichnen.

[11] **I**ntel **A**rchitecture with **32** bit.
[12] Peter J. Landin ($*1930; \dagger 3\,Juni\,2009$) - ein britischer Pionier der Informatik
[13] State **S**tack, **E**nvironment stack, **C**ontroll list, **D**ump stack [Cra06]
[14] Google's virtuelle Maschine für Mobile Geräte (Hauptbestandteil der Android Plattform)

Zurückkehrend zu der JVM muss man einen interessanten Aspekt erwähnen. Die JVM weiß eigentlich gar nichts von der Programmiersprache Java selbst. Der JVM ist lediglich nur das spezielle Binärformat der class-Dateien bekannt. Diese enthalten den bereits erwähnten Bytecode in dem sich unter anderem die JVM Instruktionen befinden. Dieser Aspekt ermöglicht somit die Wiederverwendbarkeit der JVM, da der Bytecode nicht explizit nur von Java Compiler generiert werden muss. Es gibt bereits zahlreiche Programmiersprachen, deren Compiler den Java-Bytecode generieren und somit von der JVM aktiv profitieren. Dazu gehören unter anderem JRuby, Groovy, Clojure, Jython und vor allem Scala, die in dieser Arbeit neben Java eine zentrale Position einnimmt.

2.5 Scala

Der Name der Programmiersprache, von der hier die Rede ist, leitet sich aus dem Begriff *scalable language* ab. Auf Deutsch spricht man hier von einer skalierbaren Sprache, also einer Sprache, die sich im übertragenen Sinne dynamisch anpassen kann. Es fängt damit an, dass Scala mit dem Gedanken des *Wachsens* mit den Anforderungen ihrer Benutzer designed worden ist. Der Sprachkern ist an sich kompakt gehalten, wurde aber mit dem Gedanken der einfachen Erweiterbarkeit konzipiert. Konstrukte, die wie fest in die Sprache eingebaut aussehen, können in Wirklichkeit zu einer Bibliothek gehören. Die Fusion des objektorientierten und funktionalen Programmierstils macht es möglich neue Arten der Programmierpattern und Abstraktionen der Komponenten zu formulieren. Unter anderem dadurch erstrecken sich die Einsatzmöglichkeiten von Scala von der Erstellung von kleinen Skripten bis hin zum Bau von großen komplexen Systemen. Ferner bieten die in die Sprache integrierten Konstrukte Möglichkeiten zur Realisierung skalierbarer Systeme, die in der *Many-Cores Ära* immer gefragter werden (vgl.[OSV10],[Ess11b]).

Wird dadurch auch Scala gefragter und vor allem, wird sie Java in der Zukunft ersetzen? Das wird die Zeit zeigen, aber es gibt bereits jetzt Meinungen von dem Java-Schöpfer und anderen Sprachdesignern, die auf die JVM gesetzt haben:

> *If I were to pick a language to use today other than Java, it would be Scala.*
>
> James Gosling (JavaOne 2008, Community Booth)

> *[. . .]Scala, it must be stated, is the current heir apparent to the Java throne. No other language on the JVM seems as capable of being a "replacement for Java" as Scala[. . .]* [Nut09]
>
> Charles Nutter - Schöpfer von JRuby

> *[...]Though my tip though for the long term replacement of javac is Scala. I'm very impressed with it! I can honestly say if someone had shown me the Programming in Scala book by Martin Odersky, Lex Spoon & Bill Venners back in 2003 I'd probably have never created Groovy[...]* [Str09]
>
> James Strachman - Schöpfer von Groovy

Als Gegenmeinung gibt es aber auch einen Software Design Forscher, der in seinem Blog schreibt, dass er mit Scala für sein Projekt überfordert war und wieder zu Java zurückgekehrt ist.

> *[...]The syntax has so many ways of doing things that it can be bewildering[...] The type system is just too complicated for me. The docs include "use cases" that approximate the real types to make them simpler and more understandable[...] The killer problem with Scala is that the tools are just not ready for prime time[...] I wish Scala the best of luck, but I have work to do.* [Edw11]
>
> Jonathan Edwards - Research Fellow of SDG[a] at MIT[b]

[a] Software Design Group - `http://sdg.csail.mit.edu/`
[b] Massachusetts Institute of Technology - `www.mit.edu`

Dieser Blogartikel ist auch unter anderem deshalb interessant, weil man dort auch Kommentare von Martin Odersky selbst findet. Sein Team arbeitet an einem Plugin für Eclipse IDE und verspricht somit eine bessere Toolunterstützung für Scala. Auf jeden Fall soll dieser Artikel nicht davon abhalten, die Geschichte, Eigenschaften und Sprachkonzepte der neuen und innovativen Sprache kennenzulernen. Die Geschichte von Scala ist unter anderem deshalb interessant, da ihr Ursprung gerade mit Java eng verbunden ist.

2.5.1 Geschichte

Die Geschichte von Scala fängt im Jahr 2001 an der École Polytechnique Fédérale de Lausanne - kurz EPFL an. Die Hintergründe der Idee für die Schöpfung dieser Sprache werden klarer, wenn man zuerst den Fokus auf das Portrait ihres Erfinder richtet. Es soll die Erfahrung in dem Bereich Sprachentwicklung hervorheben. Martin Odersky hat in den Jahren 1986-1989 an der ETHZ[15] als Doktorand zusammen mit Niklaus Wirth[16] an den Programmiersprachen Modula-2 und Oberon gearbeitet. Danach hat er sich der Erforschung der Grundlagen der Programmiersprachen und dem funktionalen Programmieren gewidmet. Im Jahr 1995 hat ihn das Release von Java dazu bewegt, mit der JVM zu experimentieren. Ein Jahr nach dem Release von Java hat er eine Sprache namens Pizza veröffentlicht. Mit Hilfe von dieser Sprache hat er gezeigt, welche

[15] Eidgenössische Technische Hochschule Zürich
[16] schweizer Informatiker. Turing-Preisträger. Erfinder der Programmiersprache Pascal

funktionalen Konzepte auf der JVM realisierbar sind. Es geht hier um Generics, High-Order Functions und Pattern-Matching. Das hat schnell das Interesse von Sun geweckt. Sun war nämlich sehr an den Generics interessiert und durch Kooperation ist das Project Generic Java (GJ) entstanden. Noch bevor das Konstrukt der Generics offiziell in Java eingeführt wurde, war Sun von Oderskys Compiler für GJ so begeistert, dass dieser seit Version J2SE 1.3 als der Standard-Compiler in Java zum Einsatz kommt. Nach den Erfahrungen mit Pizza und GJ hat Odersky angefangen eine neue minimale Forschungssprache namens Funnel zu entwickeln. Sie sollte funktionale Aspekte mit dem Einsatz von JVM und deren Plattformbibliotheken kombinieren. Die Sprache hat sich aber als zu akademisch und damit als wenig praktisch erwiesen. Ab diesem Zeitpunkt hat Odersky mit einem Team von Sprachdesignern angefangen an einer neuen Sprache zu arbeiten. Scala sollte nicht so akademisch wie Funnel sein, aber deren Ideen adaptieren. Außerdem sollte sie pragmatischer sein und den Fokus auf die Kompatibilität mit den vorhandenen Plattformen legen. Die erste Version wurde im Jahr 2003 veröffentlicht. Die zunehmende Aufmerksamkeit hat die Sprache aber erst ab der Version 2.0 nach dem Redesign gewonnen. Momentan befindet sich Scala in der Version 2.9.1 und wird mit Erfolg produktiv eingesetzt und ständig um neue Bibliotheken erweitert (vgl. [Ode08a],[VSO09]).

2.5.2 Eigenschaften

> *Scala at the moment doesn't need to be a language for the average Java programmer. The programmers we want to appeal to are the expert programmers—the good programmers. Our aim is to make them much more productive than they are with Java. I believe over time there will be enough teaching materials and enough good tools to also make Scala appeal to more average programmers.* [Ode10a]
>
> Martin Odersky - Schöpfer von Scala

Die Aussage von Martin Odersky ist klar, Scala ist in erster Linie nicht für Programmieranfänger bestimmt. Bis auf die Einfachheit und eine andere Open Source Lizenz weist Scala dennoch alle im Abschnitt 2.3.2 erwähnte Java Eigenschaften auf. Für den Programmieranfänger kann Scala z.B. wegen Operator-Overloading, Implicits und High-Order-Functions ziemlich komplex erscheinen. Verfügt man über gute Programmierkenntnisse und Hintergrundwissen in Scala, so profitiert man von ihren Charakteristika, was unter anderem in die Produktivität steigern kann.

Hybride Scala vereint mehrere Programmierparadigmen, um von ihren Konzepten zu profitieren. Im Vordergrund stehen das objektorientierte und funktionale Paradigma. Zusätzlich ermöglicht Scala den imperativen und dank der Möglichkeit von Kreierung eigener DSLs[17] den deklarativen Programmierstil.

[17] Siehe: `http://martinfowler.com/bliki/DomainSpecificLanguage.html`

Pure Objektorientiertheit In Scala gibt es keine Ausnahmen, wie das im Fall von primitiven Datentypen in Java der Sonderfall ist. Um die Einheitlichkeit zu gewährleisten und unschöne Seiteneffekte zu vermeiden gibt es in Scala nur Objekte.

Skalierbarkeit Einsatzmöglichkeiten von Scala reichen von der Erstellung von kleinen Skripten bis zur Realisierung von komplexen Systemen. Zum anderen bietet Scala neues Konzept für Nebenläufigkeit der Prozesse an, um die Skalierung von Performance mit dem Einsatz von Multicore-Prozessoren zu ermöglichen.

Multiplattformfähigkeit Ebenso wie Java ist Scala dank der JVM plattformunabhängig. Neben der JVM kann Scala auch auf der .Net Plattform ausgeführt werden. Es ist zwar noch mit einigen Einschränkungen verbunden, aber selbst Microsoft bemüht sich darum, Scala für die .Net Entwickler verfügbar zu machen[18]. Damit bedient Scala JVM und CLR[19] und das macht die Sprache multiplattformfähig.

Vollständige Interoperabilität Alle Java Klassen und Bibliotheken können in Scala Programme importiert und dort genutzt werden.

Ausdrucksstärke *In Scala schreibt man weniger Code und weniger Code bedeutet weniger Fehler!* [Ess11b]

2.5.3 Sprachkonzepte

Um die Fülle so interessanter und wertvoller Eigenschaften zu realisieren bedarf es der entsprechend gut durchdachten Konzepte, Erfahrung und Zeit. Die Konzeptionierung von Scala hat zwar im Jahr 2001 angefangen, aber genauer genommen wurde sie schon mit der Sprache Pizza initiiert. Bis zu dem Zeitpunkt des ersten Release von Scala hat man prototypisch mit den Sprachen Pizza und Funnel dazu gelernt und hilfreiche Erfahrungen gesammelt, vor allem, wie bewährte und auch neue Konzepte in eine Sprache integriert werden können. Man hat sich genauso wie im Fall von Java an bewährten Konzepten aus der Welt der existierenden Programmiersprachen orientiert. Was man an Scala auch gut erkennen kann, ist der Fortschritt in der Entwicklung von Programmiersprachen. Scala verfügt außer Reflections über alle Sprachkonzepte von Java (siehe Abschnitt 2.3.3). Diese sind aber unterschiedlich realisiert worden. In erster Linie ist alles in Scala Objekt. Somit ist sie im puren Sinne objektorientiert. Sie charakterisiert sich durch eine von Java verschiedene Klassenhierarchie (siehe Abbildung 2.3). Die Wurzel repräsentiert die abstrakte Klasse `scala.Any`. Von dieser Klasse erben `scala.AnyRef` für alle Referenztypen (mitunter alle Java Klassen werden von dieser Klasse abgeleitet) und `scala.AnyVal` für alle Werttypen und `scala.Unit`, was

[18] `http://it-republik.de/dotnet/news/Scala-trifft-.NET-059789.html`
[19] Common Language Runtime - .Net#'s JVM Rivale

einem besseren `void` von Java entspricht. Die Syntax von Scala unterscheidet sich

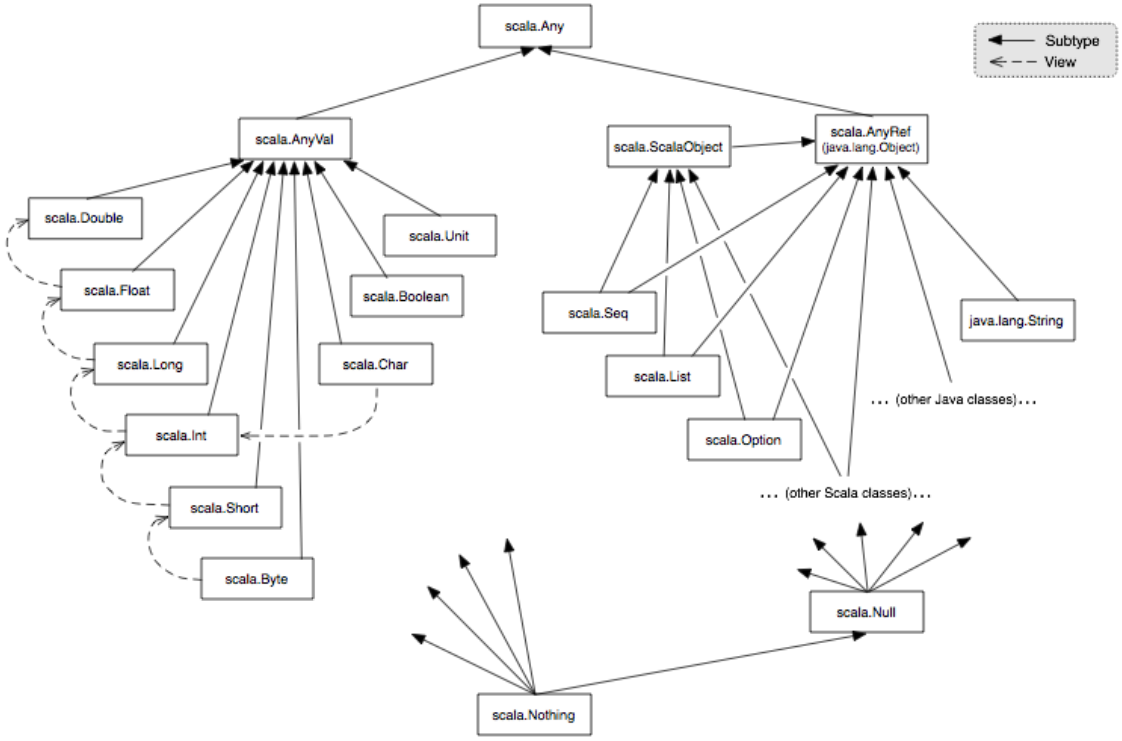


Abbildung 2.3: Scala Class Hierarchy nach [Ode08b]

von der Java-Syntax dadurch, dass sie sich an die Sprache ML[20] anlehnt und nicht an C++, wie das der Fall von Java ist (einige Syntaxunterschiede werden im Abschnitt 2.6 vorgestellt). ML war auch der Pionier, wenn es um das Typsystem geht, welches man in Scala findet. Es handelt sich hier um ein statisches, implizites und sicheres System für die Typisierung [Pie10]. Außerdem hat man zukunftsweisend neue und weitaus komplexere Konzepte in die Sprache integriert als das der Fall bei Java war. Nachfolgend werden die wichtigsten davon angelehnt an die Informationen aus [Ode10b] und [Ess11b] skizziert.

First-class Functions können wie First-class Objekte als Parameter übergeben werden oder als Ergebnis einer Routine zurückgeben werden.

High-Order Functions sind First-class Funktionen mit der speziellen Fähigkeit, andere Funktionen als Parameter zu empfangen und zurückzugeben. Dieses Konzept stammt von Haskell[21]

[20] Meta Language - siehe `http://en.wikipedia.org/wiki/ML_(programming_language)`
[21] Siehe: `http://haskell.org/`

Pattern-Matching ist ein mächtiges Übereinstimmungsprüfungs-Konzept, welches auch von Haskell stammt.

Traits sind eine Mischform aus Javas Interfaces und abstrakten Klassen, sind aber viel mächtiger als die beiden genannten zusammen. Traits kommen ursprünglich von der Programmiersprache Self[22]

Views sind *intelligente* Cast-Operationen, die selbst programmiert werden können um Typen in andere Typen automatisch zu transformieren.

Actor Model ist ein concurrent programming Konzept, welches die Möglichkeit von Erschaffung eigenständiger funktionaler Recheneinheiten (den Aktoren) bietet, die sich asynchron über Nachrichtenaustausch unterhalten können. Dieses Konzept wurde von der Sprache Erlang[23] übernommen.

XML Expressions stellen eine Möglichkeit dar, XML[24] Code direkt im Programmcode zu definieren, in Variablen abzulegen und nativ zu verarbeiten. Scala ist bei der Realisierung dieses Konzepts der Pionier.

2.6 Sprachfeatures und Syntaxunterschiede

Dieser Abschnitt richtet sein Hauptaugenmerk auf die Sprachfeatures und Syntaxunterschiede, die Scala mit sich bringt. Natürlich können wegen dem begrenzten Rahmen dieser Arbeit nicht alle Aspekte detailliert vorgestellt werden. Dazu eignet sich besser ein Standardwerk wie [OSV10]. Dennoch sollte eine Skizze der wichtigsten Sprachfeatures und Unterschiede zwischen den beiden Sprachen in dieser Arbeit nicht fehlen. Zuerst wird eine Gegenüberstellung bezüglich der Sprachkonstrukte gezeigt. Auf der einen Seite finden sich die von Java bekannten Konstrukte, die in Scala fehlen. Dem gegenüber stehen die neuen Konstrukte, die den Einzug in Scala gehalten haben:

2.6.1 Klasse, Konstruktor, singuläres Objekt

Bei der Sprache, die Objektorientierung im puren Sinne unterstützt, darf das Konstrukt der Klassen und Objekte nicht fehlen. In dem unten aufgeführten einfachen Beispiel macht sich der Unterschied zu Java durch den Konstruktor, das Schlüsselwort für die Methodendefinition und das `object` Konstrukt bemerkbar. Der primäre Konstruktor wurde in die Klassendefinition integriert und folgt als Parameterliste direkt dem Klassennamen. Alle Initialisierungen werden im Body der Klasse realisiert. Die

[22] Siehe: `http://selflanguage.org/`
[23] Siehe: `http://www.erlang.org/`
[24] eXtensible Markup Language - siehe: `http://www.w3.org/XML/`

Nicht in Scala	**Neu** in Scala
primitive Typen	Singleton Objekte
static Member	Operatoren als/in Methodennamen
Arrays als Sprachkonstrukt	Traits als Mixins
Interfaces	abstrakte Datentypen
enums als Sprachkonstrukt	(high-order) Functions
Wildcards und raw Types	high-order/existential Types
switch, break, continue	pattern matching

Tabelle 2.1: Existenz der Sprachkonstrukte in Scala gegenüber von Java[Ess11a]

sekundären Konstruktoren müssen mit `def this(...)` deklariert werden und als erstes einen anderen Konstruktor aufrufen. Die Instanzen werden wie in Java mit `new` erzeugt.

Listing 2.1: Klasse mit sekundärem Konstruktor und einem Companion Object

```
1  import java.util.Calendar
2
3  class Person(name: String, age:Int){
4    private val creationTimestamp = Calendar.getInstance().getTime()
5
6    def this(name: String){
7      this(name, 18)
8    }
9  }
10
11 object Person {
12   def apply(name: String, age: Int) = new Person(name, age)
13   def sayHallo = println("Hallo!")
14 }
15
16 // direkte Instantiierung
17 new Person("John", 21)
18 // Companion Object als Factory
19 Person("Max", 22)
20 // Aufruf der Methode des Objekts
21 Person sayHallo
```

Des Weiteren gibt es in Scala keine statischen Methoden und Felder, was eher dem Gedanken der OO entspricht. Die statischen Methoden und Member können in singulären Objekten definiert werden. Diese werden auch als Module bezeichnet und folgen der Umsetzung des Singleton Design-Patterns. `Predef` gehört zu den bekanntesten Modulen in Scala und enthält die wichtigsten Definitionen und Hilfsmethoden wie `println`. Es wird automatisch importiert und steht an jeder Stelle zur Verfügung. Besonderen Status bekommen Module, die in der gleichen Datei mit einem Namen der dort bereits definierten Klasse definiert werden. Man spricht dann von einem *companion object*. Aus Sicht des Objekts bezeichnet man die dazugehörige Klasse als *companion class*. Der Vorteil der Symbiose zwischen den beiden Konstrukten ist die Möglich-

keit des gegenseitigen Zugriffs auf die privaten Member. [Ess11b] schildert konkrete Beispiele für diesen Vorteil.

Per Konvention bestimmt die letzte Anweisung einer Methode in Scala ihren Rückgabetyp und erspart damit die explizite Angabe von `return`. Dank dem Inferenzmechanismus mmüssen der Rückgabetyp als auch das Semikolon zum Abschluss der Anweisung nicht angegeben werden.

Die Sichtbarkeit von Membern und Methoden ist anders als im Fall von Java standardmäßig auf `public` gesetzt und muss ebenfalls nicht explizit angegeben werden. Ansonsten steht `protected` und `private` zur Verfügung. Die Methoden werden mit dem Schlüsselwort `def` und die Felder mit `val` oder `var` definiert. Mit `val` gekennzeichnete Felder werden bei der Zuweisung gesetzt und können nicht mehr verändert werden. Dies entspricht dem `final` aus Java und soll in der funktionalen Welt im Sinn der Thread-Sicherheit möglichst statt dem veränderbaren `var` verwendet werden.

2.6.2 Traits

Das Konstrukt der Interfaces von Java, mit dem nur die Definition von Signaturen der Methoden und Konstanten möglich war, wird in Scala durch ein neues Sprachmittel ersetzt. Mit Traits ist es optional möglich, die definierten Methoden mit einer Implementierung zu versehen. Die Traits werden mit dem Schlüsselwort `trait` definiert und haben keinen Konstruktor. Listing 2.2 veranschaulicht eine einfache Verwendung von Traits.

Listing 2.2: Einfache Trait-Demonstration

```
1  trait Nicknamed {
2    val nickname = "no nickname :("
3  }
4
5  trait Jumping {
6    def jump = println("jumping high !")
7  }
8
9  class Pet extends Nicknamed with Jumping {
10   override val nickname = "Lucky"
11 }
```

Bei dem Einsatz von Traits handelt es sich nicht um eine klassische Vererbung. Die durch Verwendung von Traits entstehenden Hierarchien werden Mixins genannt, da die Traits in andere Module förmlich eingemischt werden. In dem Beispiel kann man sehen, wie die Komposition aus Klasse und zwei weiteren Traits realisiert werden kann. Die Erweiterung der Klasse um ein Trait beginnt mit `extends`, gefolgt von dem Traitnamen. Jedes weitere Trait wird mit dem Schlüsselwort `with` eingemischt. Beim Überschreiben der bereits implementierten Methoden und Felder muss das Schlüsselwort `override` zwingend verwendet werden. Das im Beispiel verwendete statische Hinein-Mixen kann auch je nach Bedarf dynamisch, erst bei der Instantiierung erfolgen. Der Gebrauch von Traits wird dadurch flexibilisiert.

Die Traits können von Traits und Klassen erweitert werden, wobei die dadurch entstehende Reihenfolge *well formed* sein muss. D.h. *die Klassen müssen der mittels extends festgelegten Sub/Supertyp-Beziehung folgen* [Ess11b]. Die Problematik der Auflösung der überladenen Methoden löst der Linearisierungs-Algorithmus[25] auf. Der Hierarchiebaum von Mixins, welcher die Form eines azyklischen Graphen annehmen kann, wird damit in einer Liste *flachgemacht*. Dadurch entfallen die problembereitenden Verzweigungen und die Aufrufe der überladenen Methoden können ohne Probleme aufgelöst werden.

2.6.3 Self-Types

Um den Zugriff auf *this* in verschachtelten Hierarchien zu vereinfachen, bietet Scala die Möglichkeit ein Alias dazu zu definieren. Im Listing 2.3 wird es mit `aliasForThis` realisiert. Würde das Trait `Machine` intern tiefere Hierarchien aufweisen, so könnte darin dieser Bezeichner zum Zugriff auf `this` aus der `Machine`-Sicht verwendet werden.

Listing 2.3: Self-Type in Scala

```
1  trait Cleaning {
2    def clean = println("cleaning a bit around!")
3  }
4
5  trait Talking {
6    def talk = println("hi there!")
7  }
8
9  trait Machine {
10   aliasForThis: Talking with Cleaning =>
11
12   def action = {
13     clean; talk; clean
14   }
15 }
16
17 class Robot extends Machine with Talking with Cleaning
```

Eine wichtigere Funktion die von Self-Types angeboten wird, ist die Möglichkeit der Definition von *Depends-On* Beziehungen. Das macht den Typ, der vom Self-Type repräsentiert wird, von anderen Typen abhängig. Eine *Is-A* Beziehung wird erst bei der konkreten Implementierung realisiert. Bei dieser kann das passende Mixin selbst gewählt und eingemischt werden. Allerdings sind die Typ-Abhängigkeiten in dem Beispiel von `Machine` nicht nach außen sichtbar, da nicht mehr in der Trait-Deklaration vorhanden. In Java mussten in `Machine` intern Interfaces und jeweils ein Member dazu definiert werden.

[25] siehe [Ess11b, S.204]

2.6.4 Funktionen

In Scala zählt das Konstrukt der Funktionen zu den wichtigsten dieser Sprache. Diese werden als First-class-Objekte behandelt, weil sie direkt als Werte übergeben werden können und nicht indirekt in Objekten, wie das in der OO Welt üblich ist.

First Order Functions

Es gibt zwei Möglichkeiten, um Funktions-Werte bzw. Literale anzulegen. Mit `val fnc1: T => R = arg => functionBody` werden Einargumentfunktionen und mit `val fnc2:(`T_1`, `T_2`,...,`T_n`) => R = (`arg_1`,...,`arg_n`) => functionBody` Mehrargumentfunktionen angelegt [Ess11b]. Listing 2.4 beinhaltet entsprechende Beispiele.

Listing 2.4: Beispiele der Funktionstypen in Scala (vgl. [Ess11b])

```
1 val f1: Int => Int = i => i * 2 // f1: (Int) => Int = <function1>
2 val f2: (Double, Double) => Double = (x,y) = x*y // f2: (Double, Double) => Double = <function2>
3 val f3: () => String = () => "Some String" // f3: () => () => java.lang.String = <function0>
4 val f4: Unit => Unit = Unit => println("Hello World !") // f4: (Unit) => Unit = <function1>
```

Die Funktionen werden mit dem Namen und den zu dem Funktionstyp passenden Parametern aufgerufen. Mit dem Funktionstyp ist hier `T=>R` (Ausnahme für Funktionen mit einem Parameter) und `(`T_1`,`T_2`,...,`T_n`) => R` gemeint. Die Aufrufe `f5(21)`, `f6(.5,.5)` liefern jeweils mit `42` und `0.25` nachvollziehbare Ergebnisse. Die Funktion `f7` erwartet in diesem speziellen Fall mit `()` eine leere Parameterliste und kann deshalb nur mit `f7()` aufgerufen werden. Sie liefert den definierten String als Rückgabewert zurück. Noch spezieller ist die Funktion `f8`. Sie erwartet zwar ein Argument vom Typ Unit, kann aber einen beliebigen Parameter entgegenehmen. Erklärung dazu: If e has some value type and the expected type is Unit, e is converted to the expected type by embedding it in the term $\{e; ()\}$ [Ode11]. Ausführung von `f8()` oder `f8(2)` liefert `Unit` zurück, was mit dem Rückgabetyp der Funktion `println` übereinstimmt. Als Nebeneffekt erscheint `Hello World !` in der Console.

Die Anlage einer Funktion kann auch in kürzerer Form erfolgen. Es geht hier um anonyme Funktionen, die nach dem Schema `(`arg_1`:`T_1`,...,`arg_n`:`T_n`) => functionBody` angelegt werden können [Ess11b]. Dabei muss der Name nicht definiert werden und der Rückgabetyp der Funktion wird implizit durch den Rückgabetyp von `functionBody` festgelegt. Nachfolgend wird eine äquivalente Anlage der im Listing 2.5 definierten Funktionen gezeigt. Im Kommentar stehen die REPL Ausgaben nach der jeweiligen Ausführung.

Listing 2.5: Beispiele der anonymen Funktionen in Scala (vgl. [Ess11b])

```
1 (i:Int) => i * 2 // res0: (Int) => Int = <function1>
2 (x:Double, x:Double) => x*y // res1: (Double, Double) => Double = <function2>
3 () => "Some String" // res2: () => java.lang.String = <function0>
4 (u:Unit) => println("Hello World !") // res3: (Unit) => Unit = <function1>
```

Die Funktionsargumente müssen nun geklammert werden und werden durch die Angabe des Typs begleitet. Die einzige Ausnahme stellt die leere Parameterliste `()` dar.

High Order Functions

Im Vorfeld dieses Abschnitts müssen die Begriffe der Methode und Funktion geklärt werden. Nach [Ess11b] unterscheiden sich die Begriffe unter anderem durch:

Methode wird immer mit `def method(params)` eingeleitet, hat immer einen Namen, ist an die Instanz gebunden, ist kein Wert und kann damit nicht direkt durch Variablen referenziert werden oder als Argument an andere Funktionen übergeben werden.

Funktion kann anonym (ohne Namen) definiert werden, als Parameter/Ergebnis in anderen Methoden auftreten und keine Typ-Parameter in der Signatur enthalten.

[Ess11b] macht darauf aufmerksam, dass die Scala Einführungen in die Funktionen höherer Ordnung oft mit dem Einsatz von Methoden erfolgen. Das Listing 2.6 zeigt eine Methode als Funktion höherer Ordnung, um den Unterschied zu präsentieren. Diese erwartet eine Funktion mit Integer als Parameter und Boolean als Rückgabewert sowie mit `Int*` eine variable Anzahl von Argumenten vom Typ Integer.

Listing 2.6: Methode als High-order-Function

```
1 def printFiltered( predFunc: Int => Boolean, values: Int* ) = {
2   for( value <- values; if predFunc( value ) )
3     print(value+" ")
4 }
```

Mit `(x:Int) => x%2 == 0` und `(x:Int) => x%2 == 1` gibt es zwei mögliche anonyme Funktionen als Prädikate für die geraden und ungeraden Zahlen, die als Parameter eingesetzt werden können. In der Funktion selbst wird über die variable Liste von Argumenten iteriert und die übergebene Funktion mit jedem Element in einer konditionalen Bedingung ausgeführt. Das Ergebnis der Evaluierung entscheidet, ob das Element in der Console ausgegeben wird. Bevor ein möglicher Aufruf dieser Methode vorgestellt wird, folgt eine äquivalente Umsetzung der `printFiltered` Methode als pure Funktion höherer Ordnung (siehe Listing 2.7).

Listing 2.7: Pure Funktion höherer Ordnung

```
1 val printFilteredFnc: ( Int => Boolean, Int* ) => Unit =
2                 (predFunc, values) => {
3                   for(  value <- values; if predFunc( value ) )
4                         print(value+" ")
5                 }
```

Ein möglicher Aufruf der beiden Ausführungen präsentiert das Listing 2.8.

Listing 2.8: Ausführung der Funktionen höherer Ordnung

```
printFiltered((x:Int) => x%2 == 0, 1,2,3,4)
// 2 4
printFilteredFnc( _ %2 == 0 , 1,2,3,4)
// 2 4
```

Die Aufrufe führen zum gleichen Ergebnis, weisen aber unterschiedliche Schreibweise des ersten Parameters auf. Der Platzhalter `_` kann für das Konstrukt `(x:Type) => x` eingesetzt werden.

2.6.5 Pattern-Matching

Das von Java bekannte `switch-case` Konstrukt, was mit der Version 7 neben den ganzen Zahlen auch Zeichenketten auf einen bestimmte Wert testen kann, gibt es in Scala in dieser Form nicht. Das in Scala zu `switch-case` äquivalent Konstrukt folgt dem Erkennen von Mustern, wie das der Fall in der Programmiersprache Haskell ist. Es erlaubt Wert- und Typ-Prüfungen sowie Erkennung von inneren Strukturen von Objekten. Das im Listing 2.9 vorgestellte Beispiel zeigt nur die grundlegende Funktionalität von Pattern-Matching. Das ganze Einsatzspektrum dieses Konzepts kann hier wegen dem begrenzten Umfang nicht vorgestellt werden.

Listing 2.9: Pattern-Matching Beispiel nach [Ess11b]

```
def simpleMatch(x: Any) = x match {
  case 0 => println ("Null")
  case 1|2|3|4|5 => println("zwischen 1 und 5")
  case i: Int if i < 101 => println("Int unter Hundert")
  case j: Int => println(j \+ " ist ueber Hundert")
  case d: Double => println("Absolutwert: "+scala.math.abs(d))
  case s: String => println(s.toUpperCase)
  case _ => println("UMO<Unknown Matching Object>")
}
// --- Test ---
simpleMatch(0)            // Null
simpleMatch(3)            // zwischen 1 und 5
simpleMatch(99)           // Int unter Hundert
simpleMatch(101)          // 101 ist ueber Hundert
simpleMatch(-1.0)         // Absolutwert: 1.0
simpleMatch("hallo welt") // HALLO WELT
simpleMatch(true)         // UMO<Unknown Matching Object>
```

Die Methode `simpleMatch` erwartet einen Parameter von einem beliebigen Typ. Dieser Parameter wird mittels `match` auf bestimmte Muster geprüft. Jedes Muster wird mit `case` definiert. Dabei können feste Werte, Wertgruppen, bestimmte Typen und weitere Kriterien als Übereinstimmungskriterium definiert werden. Wird ein `case` getroffen, so führt das zur Evaluierung des Code-Blocks hinter dem Doppelpfeil (`=>`). Das `case` mit der Wildcard `_` fängt alle möglichen Muster ab.

Beim Pattern-Matching ist es wichtig, dass die Reihenfolge der cases so definiert ist, dass alle definierten Fälle erreicht werden können. So kann der Wildcard-case aus dem Listing 2.9 nur am Ende stehen, da in anderem Fall alle drunterliegende cases

nicht erreichbar wären (Compiler markiert diese Unstimmigkeit). Anders ist es im Fall, wenn keine Übereinstimmung getroffen wird. Dies resultiert mit einem `MatchError`.

2.6.6 Views

In Scala ist es möglich Methoden und Funktionen zu definieren, die ein Typ in ein anderes überführen. Es handelt sich dabei um so genannte implizite Konvertierungen, die mit dem Schlüsselwort `implicit` eingeleitet werden. Damit ist es möglich zusätzliche Funktionalität in eine Klasse hinzuzufügen, ohnen diese explizit zu erweitern. Das Listing 2.10 zeigt einen einfachen Einsatz dieses Features.

Listing 2.10: Implizite Konvertierung

```
1  implicit val str2pimped: String => PimpedString = str => new PimpedString(str)
2  implicit def str2int(str: String) = Integer.parseInt(str)
3
4  def addTen(value: Int) = value + 10
5
6  class PimpedString(str: String) {
7    def printPimped() = println("Pimped " + str)
8  }
9
10 // --- Test ---
11 "Library".printPimped() // Pimped Library
12 println( addTen("32") ) // 42
```

Die Klasse String verfügt über eine Methode namens `printPimped()` nicht. Der Compiler sucht in den betroffenen Scopes nach einer Funktion oder Methode, die den String in einen Typ überführen kann, der diese Mehode besitzt. Gibt es sie, so wird die Konvertierung vorgenommen und die Methode ausgeführt. Im anderen Fall wird eine Exception geworfen. Der Aufruf von `addTen` akzeptiert einen Parameter vom Typ `Int`. In dem Test wurde aber eine Zeichenkette übergeben. Das geht deshalb, weil hier eine entsprechende Konvertierungsfunktion gefunden werden kann.

In den Beispiel wurden jeweils eine Funktion und eine Methode für implizite Konvertierung definiert. Beide Ausführungen können auch Umgekehrt realisiert werden. Existieren im selben Scope eine Funktion und eine Methode, die gleiche Konvertierung vornehmen, so wird die Funktion vom Compiler bevorzugt [Ess11b].

2.6.7 Native XML

Neben den Literalen für die Zahlen, Zeichenketten und Funktionen unterstützt Scala die XML-Literale. Es ist damit möglich XML-Code innerhalb des Scala-Codes zu schreiben. Die XML-Literale können nur verarbeitet werden, wenn sie wohlgeformtes XML repräsentieren. Die XML-Literale sind in die Sprache fest eingebaut. Die Möglichkeit der Verarbeitung bietet jedoch eine Bibliothek, die sich im Package `scala.xml` (vgl. [SR11]).

Eine einfache Demonstration des Einsatzes von XML in Scala zeigt das Listing 2.11. Die Kommentare enthalten die Ausgaben, die REPL produzieren würde (ggf. passend umgebrochen).

Listing 2.11: Scala XML Beispiel

```
1  case class Person(val name:String, val age:Int ){
2    def toXml = <person name={name}><age>{age}</age></person>
3  }
4
5  object Person {
6    import scala.xml.NodeSeq
7    def fromXml(xml: scala.xml.NodeSeq) =
8      new Person((xml \ "@name").text.toString(), xml.head.child.head.text.toInt)
9  }
10
11 // --- Test ---
12
13 val pers = Person.formXml(<person name="Max"><age>18</age></person>)
14 // pers: Person = Person(John,18)
15 val xml = pers.toXml
16 // xml: scala.xml.Elem = <person name="John"><age>18</age></person>
17 val personCopy = Person.fromXml(pers.toXml)
18 //personCopy: Person = Person(John,18)
19
20 val xmlNode = pers.toXml ++ personCopy.toXml
21 //xmlNode: scala.xml.NodeSeq =
22 //     NodeSeq(<person name="John"><age>18</age></person>, <person name="John"><age>18</age></person>)
```

Das Listing zeigt wie die Repräsentation einer Personeninstanz als XML Representation ausgegeben werden, als auch vom XML erzeugt werden kann. In der Methode `toXml` der Klasse Person können XML Literale direkt verwendet werden. In geschweiften Klammern kann Scala-Code eingebetet werden, um die XML Inhalte dynamisch zu generieren. Das Ergebnis des Aufrufs von `toXml` ist die XML Repräsentation der Personeninstanz in Form von `scala.xml.Elem`.

In der Methode `fromXml` wird ein Parameter von Typ `NodeXml` erwartet. Es Repräsentiert eine Sequenz von XML Elementen, aber auch ein einzelnes XML Element. Mit dem \ Operator kann nach bestimmten Kindelementen gesucht werden. Mit Angabe von `@` werden entsprechende Attribute der Kindelemente extrahiert. Bei `NodeSeq` handelt es sich um eine `Seq[Node]`, also stehen den XML Literalen die üblichen Methoden, die man auch in Collections findet zur Verfügung. Den Einsatz von einer davon kann man in der Extraktion des `age` Attributs sehen. Mit der Collection-Methode `head` kann auf das erste Element zugegriffen werden. Die Methode `child` liefert alle Kindelemente der Node zurück. Um auf den Inhalt des Elements zuzugreifen nutzt man die `text` Methode. Diese liefert eine Zeichenkette, die in dem Beispiel zu Integer umgewandelt werden musste.

Ganz unten in dem Listing, werden `scala.xml.Elem` Elemente konkateniert. Als Ergebnis wird eine `scala.xml.NodeSeq` geliefert.

2.7 Web Application Frameworks

2.7.1 Hintergrund

Das World Wide Web entstand in 1990[26] und war ursprünglich sehr statisch. Die Interaktion der User mit den Inhalten war sehr eingeschränkt. Um die damals publizierten Inhalte zu modifizieren musste der Autor den Inhalt der Publikation lokal editieren und anschließend auf den Server hochladen. Für das Modifizieren von Inhalten und vor allem ihrer Darstellung war die Kenntnis der Formatierungskonventionen nötig, die mit HTML[27] eingeführt wurden. Um dieses manuelle Vorgehen loszuwerden wurde das CGI[28] Standard eingeführt, um den Web Server mit externen Anwendungen interagieren zu lassen. Mit den steigenden Requestzahlen stieg die Server-Last in CGI basierten Webseiten, da für jeden Request ein neuer Thread erzeugt wurde. Effizientere Lösungen mussten gefunden werden, da die Anzahl der Webseiten, unter anderem wegen des E-Commerce Einzugs, ab 1995 dramatisch gestiegen ist[29]. Man hat darauf die serverseitigen Skripting Technologien entwickelt, um die Inhalte dynamisch und automatisch erzeugen zu können. Zu den bekannten serverseitigen Skripting Technologien gehören: PHP[30], JSP[31], ASP .NET[32] und ColdFusion[33]. Im Laufe der Zeit wurden die Webanwendungen immer komplexer und damit stieg auch die Komplexität ihrer Entwicklung. Man war gezwungen zu reagieren und neue Entwicklungsmethoden und Hilfen zu entwickeln um die Komplexität in den Griff zu bekommen und dennoch schnelle Entwicklung und Wartung zu gewährleisten. Die Lösung dafür findet man heute in Web Application Frameworks.

2.7.2 Begriffsbestimmung und Funktionen

Die Webanwendungen wie Google+, Facebook, Amazon, Ebay und Twitter basieren zwar auf den grundlegenden Webtechnologien (HTTP, HTML, CSS, AJAX), von den traditionellen Webseiten unterscheiden sie sich jedoch durch folgende, recht diffizile Eigenschaften:

- anspruchsvolle Anwendungslogik
- viele komplexe und dynamisch generierte Seiten
- häufiger Einsatz von AJAX

[26] http://www.time.com/time/magazine/article/0,9171,990627,00.html
[27] Hyper Text Markup Language
[28] Common Gateway Interface - siehe `http://www.w3.org/CGI/`
[29] `http://www.isoc.org/internet/history/2002_0918_Internet_History_and_Growth.ppt`
[30] PHP: Hypertext Processor siehe: `http://www.php.net/`
[31] Java Server Pages - `http://www.oracle.com/technetwork/java/javaee/jsp/index.html`
[32] Active Server Pages .NET siehe: `http://msdn.microsoft.com/en-us/library/aa286483.aspx`
[33] siehe: `http://www.adobe.com/de/products/coldfusion/?promoid=BQSUS`

- häufiger Bedarf an Änderungen und Erweiterungen
- Internationalisierung
- hohe Browser-Kompatibilität

Unterschiede gibt es auch in der Strategie der Entwicklung von Webanwendungen solcher *Komplexitätsklasse*. Um diese mit einem Entwicklerteam zu realisieren und vor allem später zu handhaben benötigt es an speziell durchdachter Strukturierung und Einhaltung vordefinierter Konventionen. Das alles und noch mehr wird von den Web Application Frameworks (dt. Web-Rahmenwerke) geliefert.

A Web Application Framework (WAF) is a reusable, skeletal, semi-complete modular platform that can be specialized to produce custom web applications, which commonly serve the web browsers via the Http(s) protocol. It includes building blocks of services and components that are essential for constructing sophisticated feature-rich business service and collaboration systems [. . .][SH06].

Mit dem Einsatz des WAFs sollen signifikant die Entwicklungszeit als auch der Entwicklungs- und Wartungsaufwand vermindert werden. Die Realisierung erfolgt mit einer Art von Plattform, die vorgefertigte und wiederverwendbare Konstrukte liefert. Diese können gut für die Konstruktion von anspruchsvollen, aber auch einfachen Webanwendungen gebraucht werden. Typischerweise gehören zu den zwei wichtigsten Aspekten, durch die sich die heutigen Webanwendungen auszeichnen, das Speichern der Daten in der Datenbank und die Interaktion mit dem Benutzer mittels webbasierter Benutzeroberfläche. Für den Aspekt der Persistenz wird häufig objektrelationales Mapping (ORM) verwendet. Für den zweiten Aspekt, also die Präsentationsschicht, wird in den meisten WAFs das MVC Strukturierungsmuster (siehe Abbildung 2.4) verwendet. Das MVC und einige spezielle ORM-Muster werden von [Fow02] beschrieben.

Weitere Entlastungen für den Entwickler können die WAFs in Form von grundlegenden Webanwendungsdiensten anbieten. Automatische Verwaltung der Session, Autorisierung des Benutzes, Benutzerverwaltung (z.B. sicherer Benutzer-Login, Passwortwiederherstellung), Gruppenverwaltung und diverse andere müssen dann nicht selbst entwickelt werden. Damit liegt der Schwerpunkt der Entwicklung auf den anwendungsspezifischen Modulen.

Ein weiterer Aspekt, der für den Einsatz von WAFs spricht, ist die Tatsache, dass es sich bei vielen WAFs um offene Architekturen handelt. Diese basieren auf etablierten und akzeptierten Standards (z.B. Java, .Net, XML, XSLT, Servlet, JSP, JDBC) und Technologien (z.B. JUnit, XUnit, Ant, Log4j, JDom, Xalan, Xerces, Lucene). Das hilft erfahrenen Entwicklern bei schneller Entwicklung und Support des Systems ohne dabei großen Lernaufwand zu treiben [SH06].

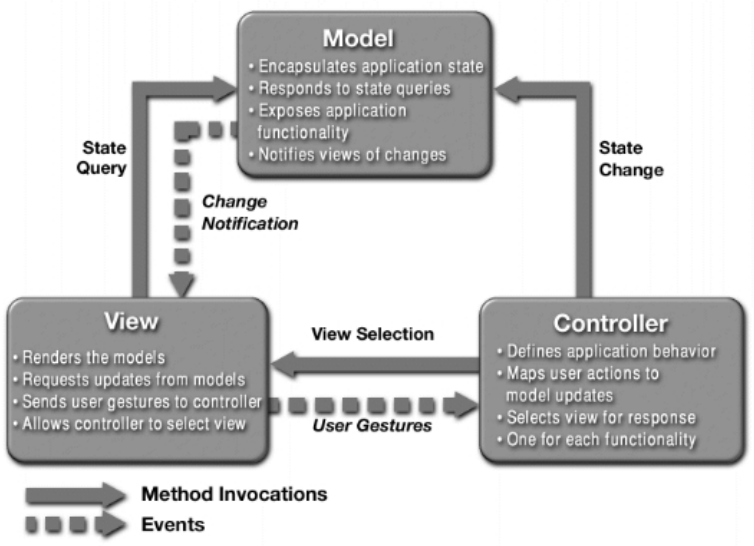


Abbildung 2.4: Allgemeine Implementierung von Modell View Controller [Eck07]

Der Einsatz von WAFs bringt aber auch Nachteile mit sich, die man nicht außer Acht lassen soll. Im Wesentlichen muss man die Einarbeitungszeit, komplizierte Austauschbarkeit, Betroffenheit im Falle Softwarebugs und Sicherheitslücken, evtl. Uneignung des WAFs für spezielle Aufgaben und ein recht unüberschaubares Angebot der Auswahlmöglichkeiten einkalkulieren.

2.7.3 Designphilosophie und Klassifikation

Um den Entwicklungsprozess einer Webanwendung zu vereinfachen versuchen die WAFs bestimmte Designphilosophien umzusetzen. Von WAF zu WAF ist es unterschiedlich, welche der folgenden von [SH06] identifizierten Design-Prinzipien auf welche Art und Weise umgesetzt werden.

Einfachheit Minimaler, einfacher Code mit Einsatz von POJOs statt exzessiver Nutzung von XML-Konfigurationsdateien.

Konsistenz Einheitliche Komponenten und Konventionen.

Effizienz Gute Performance und Skalierbarkeit (auch durch Unterstützung von Clustering).

Integration Einfache Integration der existierenden, bewährten Lösungen.

Wiederverwendbarkeit Einfache Wiederverwendbarkeit der Frameworkkonstrukte

Non-intrusive Grafikdesigner freundliche, strikte Trennung von Markup (z.B. HTML) und Programmlogik.

Diagnose Unterstützung der Fehlersuche durch Bereitstellung der Diagnose- und Debugginginformationen.

Entwicklungswerkzeuge Höchstmögliche Toolunterstützung ohne Abhängigkeiten für spezielle Entwicklungswerkzeuge.

Neben der Identifizierung von Design-Prinzipien wird von [SH06] auch eine Klassifizierung von WAFs formuliert. Die verschiedenen WAFs werden anhand der internen Struktur und grundlegenden Arbeitsweise folgendermaßen klassifiziert:

Requestbasiert Diese Art von WAFs benutzt Controller und Aktionen um die eingehenden Requests zu verarbeiten. Da die Requests an sich zustandslos sind, greift man auf die serverseitige Sessions zurück, um die Zustände abzuspeichern. Die requestbasierten WAFs unterscheiden sich untereinander in der Art und Weise, wie die Logik auf die URLs abgebildet wird als auch in der Strukturierung und der Art der Datenübergabe an die Geschäftslogik.

Komponentenbasiert Die Interna der Requestverarbeitung werden in dieser Art von WAFs abstrahiert. Die Logik wird unabhängig vom Webmedium in wiederverwendbaren Komponenten gekapselt. Das WAF verwaltet den Zustand automatisch anhand der in den Komponenteninstanzen gekapselten Daten. Das Entwicklungsmodell, welches in dieser Art von WAFs praktiziert wird, ähnelt der Art und Weise wie die Desktop-GUIs entwickelt werden. Verschiedene Komponenten-APIs und Art der Interaktion unter den Komponenten unterscheiden die WAFs dieser Klasse untereinander.

Hybrid Eine Kombination aus beiden zuvor erwähnten Klassen sieht vor, dass der Datenfluss und Logikablauf requestbasiert abläuft. Damit besteht volle Kontrolle über URLs, Forms, Parameter, Cookies und Pfadinformationen. Statt der Controller und Aktionen kann das Komponentenmodell verwendet werden. Die Komponenten können zusammengefasst, gruppiert und in anderen Projekten wiederverwendet werden. Mit diesem Ansatz wird die Wiederverwendbarkeit der Komponenten mit der vollen Kontrolle über die Steuerungsmöglichkeiten kombiniert.

Meta Framework Bietet einen Satz von Schnittstellen für grundlegende Dienste und ein Grundgerüst, welcher sich um weitere Komponenten und Dienste erweitern lässt. Typischerweise implementiert die Struktur das Inversion of Control Pattern um die Abhängigkeiten zwischen den Komponenten zu lösen und damit ihren Einsatz zu flexibilisieren. Andere Frameworks und Komponenten lassen sich dadurch einfacher integrieren. Das Framework der Frameworks lautet oft die Bezeichnung für die Vertreter dieser Klasse.

RIA-basiert Die Minimierung der Kommunikation mit dem Server durch Verlagerung der Anwendung auf den Client ist die Kernidee dieses Frameworks. Die

Realisierung erfolgt durch die Möglichkeit der lokalen Verarbeitung der Benutzerinteraktionen. Nur wenn nötig werden Daten mit dem Server ausgetauscht und Aktualisierungen der Seite vorgenommen. Diese Art von Framework erlaubt Webanwendungen zu realisieren, die den Desktopanwendungen am nächsten stehen. Sie verfügen über ein eigenes Zustands- und Interaktionsmodell.

2.8 Auswahlverfahren

Der Unterabschnitt 2.7.2 gibt einen Überblick, wie wichtig die Web Application Frameworks für die Entwicklung von Webanwendungen sind. Da in Rahmen dieser Arbeit eine Webanwendung entwickelt wird, steht nun die Auswahl eines geeigneten Web-Rahmenwerks an. Das Web Application Framework soll das Erreichen der Anforderungen einer klassischen und zeitgemäßen Webanwendung erfüllen, welche im Kapitel 3 unter Berücksichtigung der in diesem Abschnitt erwähnten Kriterien definiert wird.

Auf Grund der großen Vielfalt an Web Application Frameworks, besonders im Java-Bereich, wird ein Auswahlverfahren benötigt, um ein geeignetes Web Application Framework auszuwählen. Eine Evaluierung von AJAX-fähigen Application Webframeworks wurde bereits von [LN08] erfolgreich umgesetzt. Diese Umsetzung basiert jedoch auf durchgeführten Evaluierungen der umgesetzten Implementierungen. In diesem Abschnitt kann dies auf Grund des beschränkten Rahmen der Arbeit nicht der Fall sein kann. Die dort angewandten Verfahren werden somit nur angepasst adaptiert. Sie werden aber gleichermaßen für die Auswahl eines Java- als auch eines Scala-Web Application Frameworks angewandt.

Vorerst gibt es eine Vorauswahl an Web Application Frameworks für Java und Scala basierend auf eigenen Recherchen. Für die Auswahl des Web Application Frameworks

Tabelle 2.2: Web Application Frameworks - die Vorauswahl

Java	**Scala**
Java Server Faces 2	Circumflex
Google Web Toolkit (GWT)	Bowler
Spring MVC	Lift
Stripes	Pinky
Struts 2	Scalatra
Tapestry	sweetscala
Vaadin	
Wicket	

müssen nun Auswahlkriterien definiert werden, anhand von denen das Auswahlverfahren ausgeführt wird. Viele von ihnen können von [LN08] übernommen werden. Die Auswahlkriterien leiten sich aus den Fragen ab, die an die Anforderungen an das Framework gestellt werden. Die Fragen sehen folgendermaßen aus:

1. Ist die AJAX-Unterstützung eingebaut oder muss sie über externe Komponenten integriert werden (extra Aufwand)?
2. Bietet das Framework Konzepte um das View von der Logik sauber zu trennen?
3. Kann das Framework ohne Einsatz von XML konfiguriert werden?
4. Bietet das Framework eine qualitätive Dokumentation mit der man schnell in die Materie einsteigen kann?
5. Unterstützt das Framework die Internationalisierung und Lokalisierung?
6. Bietet das Framework Validierungsmechanismen für die Benutzereingaben?
7. Wird das Framework aktiv entwickelt ? Gibt es Foren, Mailinglisten?
8. Gibt es Anhaltspunkte um einzuschätzen, ob man mit dem Framework für die Zukunft planen kann?
9. Wie unterstützt das Framework das Testen?
10. Gibt es für das Framework Entwicklungstools?

Die aus den zuvor aufgelisteten Fragen sich ergebenden Entscheidungskriterien werden mit zusätzlicher Gewichtung versehen, um die Wichtigkeit für die einzelnen Aspekte hervorzuheben (siehe Tabelle 2.8). Die von [LN08] definierte Legende für die Bewer-

Bez.	**Entscheidungskriterium**	**Gewicht.**
K1	AJAX Unterstützung	0,15
K2	Trennung View und Logik	0,15
K3	XML-freie Konfiguration	0,1
K2	Qualitätive Dokumentation	0,1
K5	Internationalisierung & Lokalisierung	0,1
K6	Validierung	0,1
K7	Community-Aktivität	0,1
K8	Zukunftsicherheit	0,1
K9	Testbarkeit	0,05
K10	Toolunterstützung	0,05

Tabelle 2.3: Entscheidungskriterien angelehnt an [LN08]

tung wird im Original übernommen (siehe Tabelle 2.8). Die Auswertungen für Java WAFs (siehe Tabelle 2.8) und für Scala WAFs (siehe Tabelle 2.6) basieren auf den Informationen, die aus der Projektseite des jeweiligen Web Application Frameworks und der dort bereitgestellten Dokumentation entnommen werden konnten. Die gewonnenen Erkenntnisse, die die Grundlage für die einzelnen Werte darstellen, können für jedes evaluierte Web Framework aus dem Anhang A entnommen werden.

Bewertungswert	Definition
1,0	Alternative fully satisfies business requirement or decision criterion.
0,5	Alternative partially satisfies business requirement or decision criterion.
0	Unknown or null/balanced (The alternative neither satisfies nor dissatisfies business requirement or decision criterion.)
- 0,5	Alternative partially dissatisfies business requirement or decision criterion.
- 1	Alternative fully dissatisfies business requirement or decision criterion.

Tabelle 2.4: Legende für die Bewertung nach [LN08]

Der Auswertung zufolge entsprechen Apache Wicket für Java und Lift Webframework für Scala den aufgestellten Kriterien am besten.

	JSF		**Spring MVC**		**Stripes**		**Struts 2**		**Tapestry**		**Wicket**		**GWT**		**Vaadin**	
	Raw	Gew.	Raw	Gew.	Raw	Gew.	Raw	Gew.	Raw	Gew.	Raw	Gew.	Raw	Gew.	Raw	Gew.
K1	0,5	0,075	1	0,15	0	0	0,5	0,075	0,5	0,075	1	0,15	1	0,15	1	0,15
K2	0,5	0,075	1	0,15	1	0,15	0,5	0,075	1	0,15	1	0,15	1	0,15	1	0,15
K3	0,5	0,075	-0,5	-0,05	1	0,1	0	0	1	0,1	1	0,1	0,5	0,05	1	0,1
K4	1	0,1	1	0,1	1	0,1	1	0,1	1	0,1	1	0,1	1	0,1	1	0,1
K5	0,5	0,05	0,5	0,05	1	0,1	1	0,1	1	0,1	1	0,1	1	0,1	1	0,1
K6	1	0,1	1	0,1	0,5	0,05	0,5	0,05	1	0,1	1	0,1	1	0,1	1	0,1
K7	1	0,1	1	0,1	0,5	0,05	1	0,1	1	0,1	1	0,1	1	0,1	1	0,1
K8	1	0,1	0,5	0,05	0,5	0,05	0,5	0,05	0,5	0,05	1	0,1	1	0,1	0,5	0,05
K9	0,5	0,05	1	0,1	1	0,1	1	0,05	1	0,1	0,5	0,025	0	0	0,5	0,025
K10	1	0,05	1	0,05	0,5	0,0025	1	0,05	1	0,05	1	0,05	1	0,05	0,5	0,025
Summe	7,5	0,775	7,5	0,8	0,7	0,7025	0,7	0,65	0,9	0,925	9,5	9,75	0,85	0,9	0,85	0,9

Tabelle 2.6: Evaluierung der Java Web Application Frameworks

	Bowler		**Circumflex**		**Lift**		**Pinky**		**Scalatra**		**sweetscala**	
	Raw	Gew.	Raw	Gew.	Raw	Gew.	Raw	Gew.	Raw	Gew.	Raw	Gew.
K1	1	0,15	0,5	0,075	1	0,15	0,5	0,075	0,5	0,075	0,5	0,075
K2	0,5	0,075	0	0	1	0,15	0	0	0	0	0	0
K3	1	0,1	1	0,1	1	0,1	1	0,1	1	0,1	1	0,1
K4	0,5	0,05	0	0	1	0,1	0	0	0	0	0	0,1
K5	1	0,1	0	0	0,5	0,05	0,5	0,05	0	0	1	0,1
K6	0,5	0,025	0,5	0,05	1	0,1	0	0	0,5	0,05	0	0
K7	1	0,1	1	0,1	1	0,1	1	0,1	1	0,1	1	0,1
K8	0	0	0	0	1	0,1	0	0	0,5	0,05	0	0
K9	0	0	0,5	0,0025	1	0,05	0	0	1	0,05	0	0
K10	0,5	0,0025	1	0,05	1	0,05	0,5	0,0025	1	0,05	0,5	0,0025
Summe	6	0,6025	4,5	3,775	9,5	0,95	3,5	0,3275	5,5	5,25	4,0	0,4775

Tabelle 2.8: Evaluierung der Scala Web Application Frameworks

2.9 Apache Wicket

Das Ergebnis der Evaluierung zeigt, dass Apache Wicket den zuvor definierten Auswahlkriterien am besten genügt. Dieses Web Framework wird in dieser Arbeit Java als Web-Technologie repräsentieren. Es handelt sich dabei um ein komponentenbasiertes Präsentationsframework, welches sich an das MVC-Pattern anlehnt.

Zum Zeitpunkt des Erscheinens der ersten finalen Version[34] gab es schon viele bekannte Java Web Frameworks. Auf die Frage *Why "Reinvent the Wheel"?* antworten die Autoren von Wicket: *"because this time we could make it rounder!"*[35]. Zum Zeitpunkt der Erstellung dieser Arbeit war Version 1.4.17 die letzte stabile Veröffentlichung und Version 1.5 erreichte den Status Release Candidate 5.1.

In den nachfolgenden Unterabschnitten werden die Interna und die Funktionsweise der Frameworks präsentiert. Zuerst folgt aber eine kurze Vorstellung der wichtigsten Konzepte.

2.9.1 Framework Struktur

Die wichtigsten Bausteine des Apache Wicket Frameworks mit ihren Beziehungen stellt die Abbildung 2.5 dar. Das Framework setzt auf die Servlet API 2.3 (Version 1.4.17) und 2.5 (ab Version 1.5). Der unmittelbar darauf agierende `WicketFilter` (bzw. `WicketServlet`) untersucht, verarbeitet und/oder leitet alle eingehende Requests an die Wicket-Anwendung weiter. Er stellt die Umsetzung des Front-Controller-Patterns[36] dar. Dieser Front-Controller wird in der `web.xml` deklariert. In dieser Deklaration wird auch die `WicketApplication`-Klasse als `applicationClassName`-Parameter registriert. Sie ist die zentrale Anlaufstelle jeder Wicket-Anwendung. Mit der Methode `getHomePage` wird die Startseite der Anwendung festgelegt. Mit der `init` Methode, die nur einmal ausgeführt wird, können die Anwendungskonfiguration und die Konfiguration aller gemeinsamen Ressourcen, die seitenübergreifend genutzt werden können, vorgenommen werden. Mit der Methode `newSession` wird für jeden User separat das Sessionobjekt erzeugt. Die `WebSession` enthält die für die Nutzerinteraktionen notwendigen Informationen. So enthält die `WebSession` mindestens eine `PageMap`, in der alle Seiten abgelegt sind, die von dem Benutzer aufgerufen worden sind. Es werden nicht nur die zuletzt aufgerufenen Seiten gespeichert, sondern auch evtl. ältere Versionen der aktuellen Seite. So kann der Zustand der Anwendung zu jedem Zeitpunkt wiederhergestellt werden. Öffnet der Benutzer eine weitere Browser-Registerkarte oder ein Fenster-Popup so wird eine neue `PageMap` erzeugt. Die Daten der `WebSession` werden im `SessionStore` gespeichert, was standardmäßig in `HttpSession` des Packets `javax.servlet` geschieht, aber die Offenheit für eigene Implementierung lässt. Ebenso in der Abbildung 2.5 zu sehen sind die weiteren Komponenten, deren wesentliche

[34] Version 1.0 - 7 Juni 2005 - `http://wicket.sourceforge.net/wicket-1.0/`

[35] `http://wicket.apache.org/meet/introduction.html`

[36] `http://java.sun.com/blueprints/corej2eepatterns/Patterns/FrontController.html`

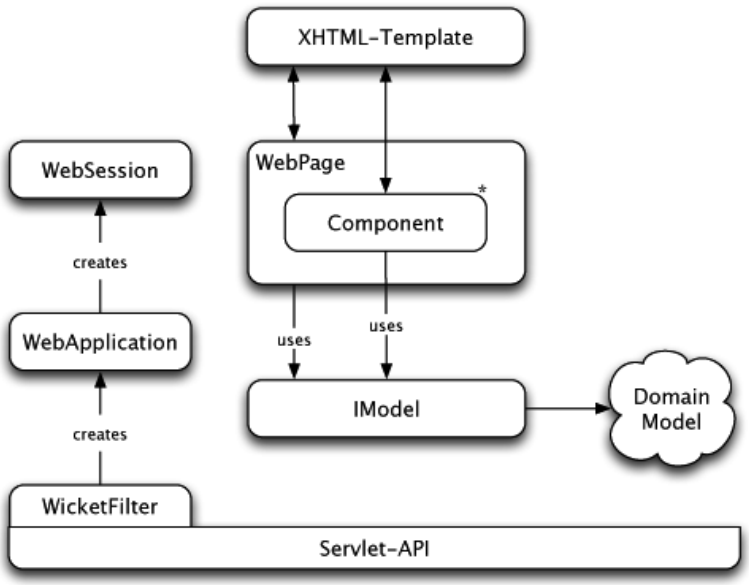


Abbildung 2.5: Bausteine einer Wicket-Anwendung nach [FMS09]

Rolle in der Interaktion mit dem Benutzer besteht: `WebPages`, `Components`, XHTML-Templates und das `IModel` Interface für die Wicket-Models. Ein Beispiel solcher Interaktion präsentiert die Abbildung 2.6. Eine Benutzeranfrage wird in Form von Request an den Server gesendet. Wicket leitet es an die Komponente in Form eines Methodenaufrufs weiter. An dieser Stelle wird der Unterschied zu dem typischen MVC-Pattern sichtbar, da die Komponente sowohl den Controller als auch Teile des Views repräsentiert. Die Event-Handler-Methoden in der Komponente wie z. B. `onClick()` oder `onSubmit()` bilden die Controller-Funktionen ab. Damit können Benutzereingaben verarbeitet, das Modell aktualisiert und ggf. ein neuer View aufgerufen werden. Bei Modellen handelt es sich um die Repräsentation der Domänenklassen der Anwendung wie z.B. Produkt oder Kunde. Auf die Domänenobjekte kann dank dem `IModel-Delegate-Interface` von vielen Komponenten aus zugegriffen werden. Ist die Verarbeitung in den Event-Methoden abgeschlossen, wird der View gerendert. Per default ist es die Quelle des Requests oder die im Controller per `setResponsePage()` gesetzte Seite. Der Markup der Seite wird aus dem zugehörigen Template und der Ausgaben der in ihr enthaltenen Komponenten generiert. Die Komponenten liefern ihr Markup durch den Aufruf der Methode `renderComponent()`. Weitere Details zu der View Technik werden im Unterabschnitt 2.9.3 vorgestellt. Als Nächstes werden die Phasen des Requests vorgestellt.

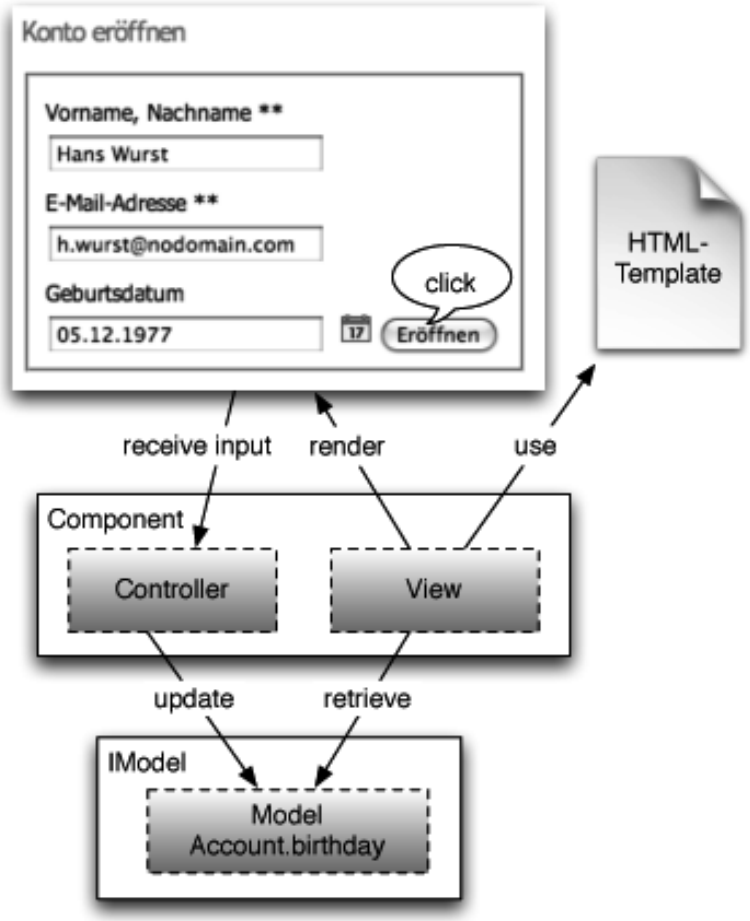


Abbildung 2.6: Beispielinteraktion in Wicket nach [FMS09]

2.9.2 Request Handling

Zur Veranschaulichung der operativen Funktionsweise des Frameworks bietet es sich an, den Prozess der Verarbeitung der Requests vorzustellen. Damit wird der Weg von dem Stellen der Anfrage bis zum Senden der Antwort schrittweise verfolgt. Eine vereinfachte grafische Darstellung dieses Vorgangs liefert die Abbildung 2.7. Die interessante Phase beginnt nach der Erzeugung von dem `RequestCycle` Objekt. Dieses ist zuständig für die Abarbeitung der von der Wicket Anwendung gekapselten `HttpRequest` und `HttpResponse`. `RequestCycle` führt die Bearbeitung aus, indem es verschiedene Aufgaben an `RequestCycleProcessor` delegiert. So wird mit dessen Hilfe das Ziel der Abfrage (`RequestTarget` - alte Seite wird deserialisiert oder eine neue erstellt) ermittelt, die Event-Behandlung gestartet und das Response erstellt. Anschließend kümmert sich `RequestCycle` um die Bereinigung, indem alle temporären Dateien gelöscht werden (Aufruf der `detach()` Methode auf jede Komponente), sowie um die

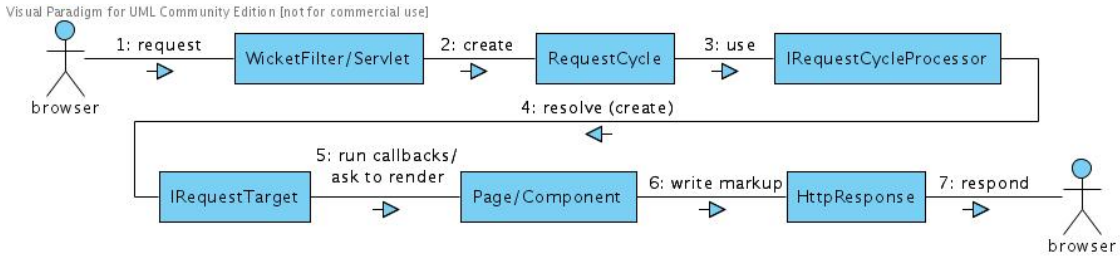


Abbildung 2.7: Zyklus der Verarbeitung von einem Request

Ablage der Seite in der `PageMap`. Die Seite wird auch serialisiert und im `PageStore` abgelegt. Die Anwendung kann danach weitere Requests empfangen.

2.9.3 View Technik

Die grundlegende Abstraktion in einer Apache Wicket Anwendung stellen die Seiten mit ihren Komponenten dar. Realisiert wird eine Seite durch eine Ableitung der Klasse `WebPage`. Bei einer Seite handelt es sich um die höchste Ebene einer Containerkomponente. In die Hierarchie einer Seite können natürlich beliebige weitere Containerkomponenten, wie z.B. Panels, Tabellen oder die einfachsten Komponenten, wie Buttons oder andere, mit der Methode `add(...)` hinzugefügt werden. Manche Komponenten wie Buttons, DropDowns oder Links können Benutzerereignisse verarbeiten. In den entspechenden Event-Callback-Methoden kann der eigene Code implementiert werden (siehe Listing 2.12).

Listing 2.12: onSubmit Callback in der Button Klasse

```
1 Button next = new Button("nextButton") {
2       public void onSubmit(){
3               // Eingabewerte verarbeiten,
4               // eventueller Aufruf der Businesslogik
5       }
6 };
```

Die Anwendungslogik kann damit nur in Seiten (`WebPage`) und in Komponenten aufgerufen werden und nicht im Markup, wie das der Fall von JSP ist. Ein weiterer Grund wieso das gar nicht möglich ist, ist die Tatsache, dass Wicket ein wohlgeformtes (X)HTML-Markup für das Seiten- und Komponenten-Design verwendet. Ein Beispiel einer einfachen Wicket-Seite mit dem zugehörigen Template zeigt die Abbildung 2.8. Die Verbindung zwischen den Komponenten, die in der Seitenklasse instantiiert werden, und dem Template geschieht mittels einer Wicket-ID. Diese wird im Markup durch ein `wicket:id` Attribut realisiert. Dieses markiert die Stelle in dem Markup, wo der gerenderte Inhalt der Komponente eingebettet wird.

Im Wicket übernehmen also die Seiten die wichtigsten Aufgaben: Darstellung, Benutzerereignisverarbeitung und die Auswahl der Folgeseite. Sie können die gesamte

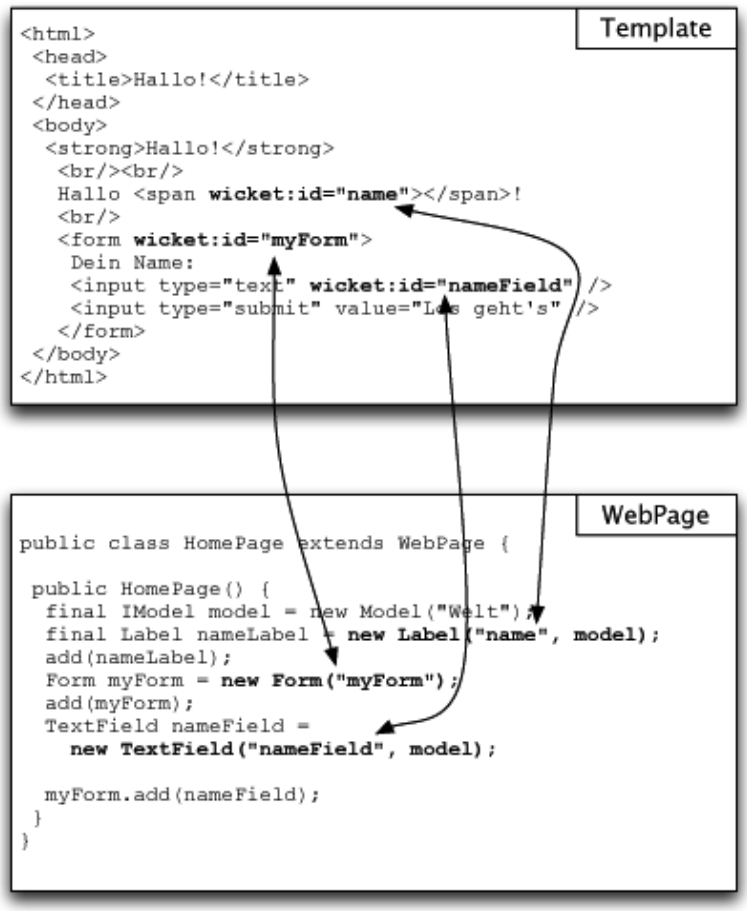


Abbildung 2.8: Seitenvorlage mit ihrer Implementierung [FMS09]

Logik in der Seitenklasse vereinen oder stoßen selbst die entsprechenden Aktionen mit der enthaltenen Logik an. Wegen dieser Strategie bezeichnet man Wicket als ein seitenzentriertes Framework. Vorteile ergeben sich in guter Übersichtlichkeit und dem Aspekt, dass die Logik des Controllers unmittelbaren Zugriff auf die Seiteneigenschaften hat.

Man muss aber hinzufügen, dass die Seiten nicht nur aus Java und (X)HTML bestehen. Ihnen können auch solche Artefakte wie CSS, JavaScript, Images oder Internationalisierungs-Ressource-Bundles hinzugefügt werden. All diese Artefakte werden mittels Klassenpfad referenziert und können problemlos von vielen Seiten wiederverwendet werden. Zu guter Letzt gibt es noch einen erwähnenswerten Ansatz, der das Verhalten des Controllers, aber auch des Views der Komponenten beeinflussen kann. Es handelt sich hier um die sogenannten *Behaviors*, also gekapseltes und wiederverwendbares Verhalten, mittels dem das Komponentenverhalten erweitert werden kann. Solche Behaviors reagieren auf entsprechende Events wie z.B. `onClick()` und können beispielsweise JavaScript ausführen oder CSS-Attribute setzen.

2.10 Lift Web Framework

Das bekanteste, am weitesten entwickelte und auf Scala basierende Web-Framework heißt Lift. Es wird seit Anfang 2007 aktiv entwickelt und bereits in namhaften Plattformen eingesetzt. In erster Linie profitiert Lift von den Möglichkeiten, die durch die Programmiersprache Scala ermöglicht wurden. Dank dem jungen Alter profitiert Lift zusätzlich von innovativen Konzepten, die sich mit ihrem Einsatz in anderen bekannten Web-Frameworks bewährt haben.

2.10.1 Konzepte

Ähnlich wie es bei der Programmiersprache Scala der Fall war, versuchte man die bewährten Konzepte in einem Web-Framework zu vereinen. Dabei hat man sich an dem aktuellen Fortschritt der Web-Technologie orientiert und Merkmale folgender Web-Frameworks als Vorbilder genommen (vgl. [FK11]):

Ruby on Rails[37] Die Konfiguration durch Einhaltung von Konventionen [38] und Verfolgung des DRY[39] Prinzips steht bei Ruby on Rails an oberster Stelle. Lift setzt diese Prinzipien unter anderem um in der Verwendung der Maven-Projektstruktur und dem Einsatz von XHTML-Tags, anhand derer die Klassen gefunden werden können.

Seaside[40] Dank Scala kann Lift Nutzen aus Closures und Callback-Funktionen ziehen. Damit wurde eine geeignete Basis für AJAX- und Comet-Funktionen als auch für die Erstellung und Verarbeitung von Formularen geschaffen. Ähnlich wird in dem in Smalltalk programmierten Seaside Web-Framework vorgegangen, um zustandsbehaftete Web-Applikationen zu realisieren.

Apache Wicket Mit herkömmlichen HTML-Werkzeugen sollte es möglich sein, die Templates bearbeiten zu können, ohne dabei unbekannte Elemente benutzen zu müssen. Diese Designer-freundliche Idee wurde in Apache Wicket umgesetzt, indem man ausschließlich valide XHTML-Elemente in den Templates erlaubt hat. Lift Templates werden angelehnt an diese Idee erstellt.

Django[41] Das in Python geschriebene Web-Framework bietet einen Automatismus zur Erzeugung des Administrationsbereichs, mit dessen Hilfe alle Aufgaben der Benutzerverwaltung realisiert werden können. Ferner bietet es die Möglichkeit der Erstellung von CRUD Seiten für andere Domain-Objekte. Der Entwickler kann davon auch in Lift profitieren.

[38] Convention over Configuration - `http://softwareengineering.vazexqi.com/files/pattern.html`

[39] *Don't repeat yourself*

Des Weiteren profitiert Lift von Scala Konzepten. Die Kompatibilität mit Java ermöglicht nicht nur den Zugriff auf alle möglichen Java Bibliotheken, sondern auch die Ausführung der Lift Anwendungen (exportiert als .war - Web-Archive) auf beliebigen Servlet-Containern. Die native Unterstützung von XML-Literalen und eine integrierte Bibliothek ermöglichen eine einfache Manipulation von XHTML Elementen. Zu guter Letzt basiert die Unterstützung der Comet Technik auf dem Actor-Konzept von Scala.

2.10.2 Struktur

Im Vergleich zu Apache Wicket handelt es sich bei Lift um ein Full-Stack Web-Framework. Es wird mit allen wichtigen Bibliotheken ausgeliefert, die zur Entwicklung von vollwertigen Web-Applikationen benötigt werden. Diese sind aufeinander abgestimmt und bilden ein Framework-Stack. Die mit Lift entwickelten Applikationen werden in einem J(2)EE Servlet Container ausgeführt, welcher wiederum die JVM als Laufzeitumgebung benutzt. Das ganze Ecosystem von Lift Applikationen präsentiert die Abbildung 2.9. Zuerst werden die Kernkomponenten des Frameworks beschrieben

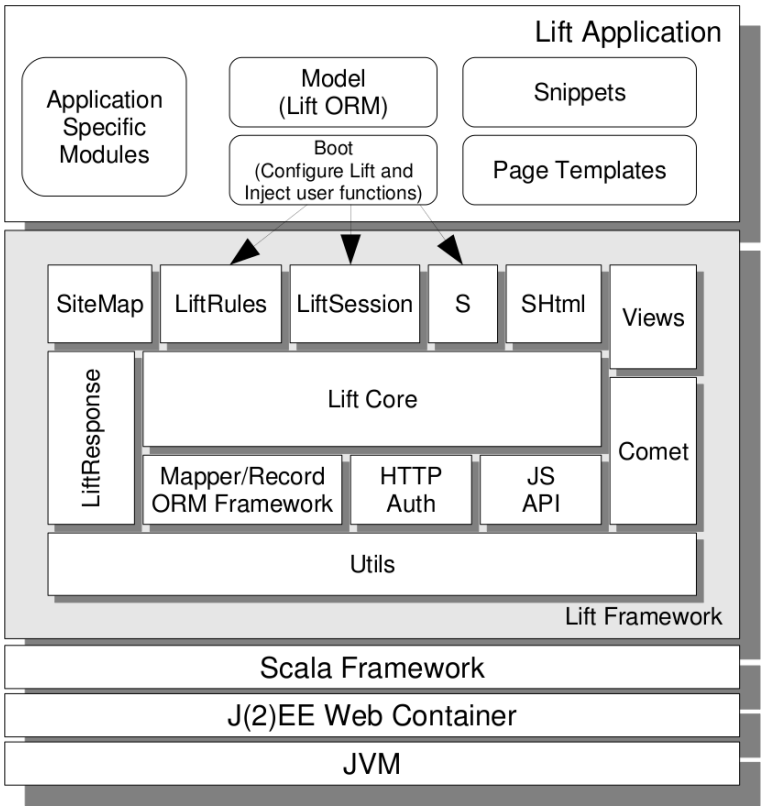


Abbildung 2.9: Architektur von Lift [CBDW11]

und anschließend die Komponenten, mit dessen Hilfe von den die Lift-Applikationen erstellt werden.

LiftCore Die Engine des Frameworks, die für den Request/Response Zyklus, die Rendering Pipeline und die Aufrufe von Benutzerfunktionen verantwortlich ist.

SiteMap Registriert und regelt den Zugriff auf alle Seiten der Lift-Anwendung. Ermöglicht Erstellung und Gruppierung von hierarchisch komplex aufgebauten Menüs.

LiftRules Globaler Konfigurationsmechanismus von Lift.

LiftSession Repräsentation des Session-Zustands.

S Das S steht für *statefull* und es repräsentiert das zustandsbehaftete Objekt, welches den Kontext eines Request/Response enhält. Es ermöglicht den Zugriff unter anderem auf folgende Komponenten:

- Cookies
- Zeitzonen, Internationalisierung und Lokalisierung
- HTTP Headers, Document Types

Außerdem ermöglicht es Request- und Session-Rewrites, Weiterleitungen und Definition von Dispatch Funktionen, die Manipulation der Session erlauben.

SHtml Enthält diverse Helper-Funktionen für die Erstellung von (X)HTML Artefakten.

Views Die `LiftView` Objekte ermöglichen die direkte Generierung von XML Inhalten und erlauben damit die Erstellung von Seiten ohne dabei auf die (X)HTML Templates zuzugreifen.

LiftResponse Die Basis der kompletten Hierarchie von Lift-Response Klassen. Als Abstraktion des Response representiert es das, was an den Client als Antwort übertragen wird.

Comet Repräsentiert die Comet-Actor Schicht, die es ermöglicht asynchron Inhalte an den Client zu schicken.

ORM Eine leichtgewichtige Bibliothek, die für das objektrelationale Mapping zuständig ist. In Lift gibt es zwei verschiedene Ausführungen davon. Der *Mapper* wurde mit Lift 1.0 eingeführt und ist für die relationalen Datenbanksysteme ausgelegt. *Record* dagegen wurde für alle Datenbanksysteme gedacht und unterstützt momentan auch NoSQL Systeme wie z.B. MongoDB[42] oder CouchDB[43].

[42] `http://www.mongodb.org/`
[43] `http://couchdb.apache.org/`

HTTP Auth Erlaubt die Benutzung von Basic als auch Diggest Access Authentification und bietet damit mehr Kontrolle über das HTTP-Authentification Model[44].

JS API Die Abstraktionsschicht für JavaScript. Mit Hilfe von Scala Klassen/Objekten werden die JavaScript Artefakte abstrahiert und erlauben auch durch Kombination einfache Generierung von JavaScript-Code.

Utils Ein Satz an diversen Helper-Funktionen, die von Lift intern benutzt werden, aber auch dem Entwickler zur Verfügung stehen.

Das waren die Kernkomponenten, die das Lift-Framework ausmachen. Nachfolgend werden die Interna einer Lift Web-Applikation vorgestellt.

In der Abbildung 2.9 beinhaltet der Abschnitt *Lift Application* die Komponenten *Boot*, *Snippets*, *Page Templates*, *Model (Lift ORM)* und *Application Specific Modules*. Die *Boot*-Klasse ist dabei der Einstiegspunkt jeder Lift Web-Applikation und der in ihr enthaltene Code wird einmalig während des Ladens der Applikation in den Servlet-Container ausgeführt. Nach der Ausführung werden alle in ihr festgelegten Konfigurationsparameter eingefroren und können nicht mehr verändert werden. Ähnlich wie Apache Wicket verzichtet Lift auf die XML-Konfiguration und erlaubt die Web-Anwendung mit Hilfe vom nativen Code, also in Scala (in der Boot-Klasse) zu konfigurieren. Hier kommen die in der oberen Auflistung genannte Komponenten zum Einsatz. In der Boot-Klasse kann eine SiteMap definiert werden, in der alle Seiten definiert werden müssen, auf die ein möglicher Zugriff erfolgen kann. Wird eine Seite erstellt, aber nicht in die SiteMap eingetragen so wird sie auch nicht per direkte Angabe ihrer Lokation ausgeliefert. An alle Seiten können selbstdefinierte Funktionen angehängt werden, mit Hilfe von den festgelegt werden kann, wann eine Seite betreten werden kann (z.B. isLoggedIn, isAdmin, isDay, ...). Des Weiteren können mit `LiftRules` solche Aspekte wie Encoding-Type des Requests, HTML5 Renderingmodus, der Pfad zu den Snippets (als Konfigurations-Konvention), AJAX-Image für den Start und das Ende der asynchronen Anfrage, URL-Rewriting Regeln und viele andere Konfigurationen vorgenommen werden. Letzten Endes kann in der Boot-Klasse durch die Ableitung des Traits `ConnectionManager` die Datenbankverbindung definiert werden.

Eine weitere Komponente, die in einer Lift-Webapplikation zum Einsatz kommt, ist das ORM. Damit können die Domänenobjekte in Hinsicht auf deren persistente Haltung erstellt werden. Um ein bespielhaftes Objekt mit dem Lift-Mapper zu kreieren, wird der Klassentyp in einer rekursiven Beziehung zu `LongKeyedMapper[Klassentyp]` definiert. Will man auch einen Primärschlüssel in Form einer numerischen LongId haben, so kann man den Trait `IdPK` dazu hineinmixen. Die Felder des Domainobjekts werden als Singeltonobjekte deklariert und können die bereits definierten Feldtypen wie `MappedString`, `MappedDecimal`, `MappedDateTime` oder weitere ableiten. Mit

[44]RFC 2617 - `http://www.ietf.org/rfc/rfc2617.txt`

`MappedLongForeignKey` können Fremdschlüsselbeziehungen zu anderen Domainobjekten festgelegt werden. Wurden alle benötigten Domainobjekte auf diese Weise definiert, so braucht man sich um die Erstellung des dazugehörigen Datenbankschemas nicht zu kümmern. Mit Hilfe von dem `Schemifier` kann das Datenbankschema automatisch generiert werden. Dazu ruft man die `schemify` Methode auf und übergibt ihr alle Domainobjekte, die natürlich von den entsprechenden Mapper-Traits abgeleitet wurden. Der Aufruf dieser Methode erfolgt in der Boot-Klasse.

Die Komponenten Snippets und Page-Templates bilden die View-Grundlage und werden gesondert im Unterabschnitt 2.10.4 vorgestellt. Bei den *Application Specific Modules* handelt es sich um unabhängige Module, die zusätzlich angeboten werden. Mit deren Hilfe können Services wie Facebook, Twitter, PayPal, aber auch Scalate Template-Engine, OAuth, OSGi und weitere in die Entwicklung der Web-Anwendung integriert werden.

Nun sind bis auf die View-Komponente alle relevanten Teile einer Lift Web-Anwendung skizziert worden. Im nachfolgenden Abschnitt wird der Zyklus eines Requests vorgestellt.

2.10.3 Request Handling

Dieser Abschnitt widmet sich dem Prozess der Transformation eines Requests in ein Response. Den ganauen Ablauf dieses Prozesses stellt die Abbildung 2.10 dar. Der Prozess startet mit der Ankunft eines neuen Requests. Die an `LiftRules.early` angehängten Funktionen werden als erstes auf das `HttpServletRequest` Objekt angewandt. Dies geschieht noch, bevor es in die standardmäßige Verarbeitung kommt. Hier kann z.B. die (X)HTML Ausgabe auf UTF-8 Encoding gesetzt werden. Als Nächstes werden die in `LiftRules.rewrite` definierten URL-Transformationen vorgenommen, um z.B. benutzerfreundliche URL zu generieren. Dem folgt die Ausführung der in `LiftRules.onBeginServicing` definierten Hook-Funktionen, die beispielsweise das Logging aktivieren können. Nach diesem Schritt fängt das eigentliche Request-Processing an. Zuerst wird in der `LiftRules.statelessDispatchTable` nach den partiellen Funktionen nachgeschaut. Gibt es ein übereinstimmendes Matching für das aktuelle Request, so werden kein `S` Objekt und keine `LiftSession` erzeugt und die Erstellung des Responses übernehmen die definierten Funktionen. Diese Strategie ist für die Implementierung von REST APIs nützlich. Gibt es keine Übereinstimmungen, so sucht der SessionMaster-Actor nach einer Übereinstimmung der HTTP session ID (in Cookie oder in der URI). Sollte keine gefunden werden, so wird ein neues `LiftSession` Objekt erzeugt. Dieses hält die Informationen über den Zustand des Requests. Nach der Erzeugung dieses Objekts findet der Aufruf von den in `LiftSession.onSetupSession` definierten Hook-Funktionen statt. Nach der Ausführung dieser Funktionen wird das `S` Objekt initialisiert, welches den Status des aktuellen Requests und Responses repräsentiert. Mit `LoanWrappers` erlaub Lift dem Entwickler das Hinzufügen von eigenen Request Verarbeitungsfunktionen. Diese werden dem

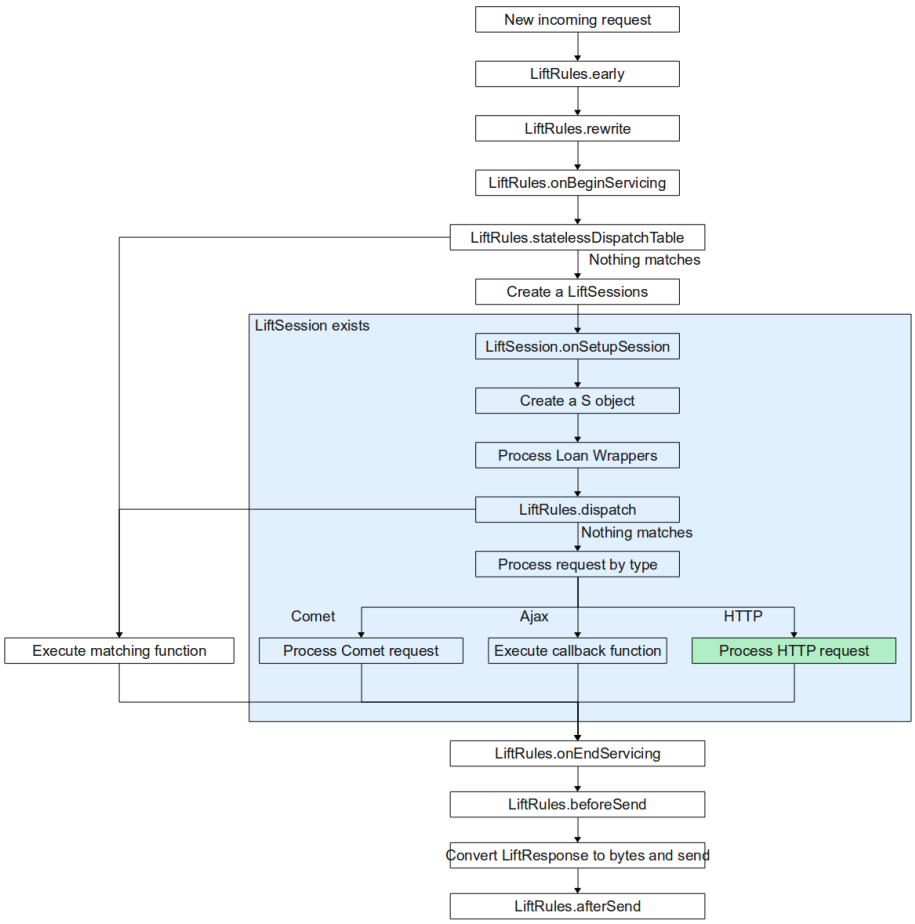


Abbildung 2.10: Globale Request Verarbeitung [CBDW11]

Objekt in der Boot Klasse mit `S.addAround` hinzugefügt und *umschließen* die eigentliche Request-Verarbeitungsfunktion. Die `apply` Methode eines `LoanWrappers` erhält als Parameter diese Funktion und gibt ihr Ergebnis am Ende zurück. Dabei können Pre- und Postroutinen ausgeführt werden. So kann beispielsweise vorweg sichergestellt werden, dass Ressourcen erreichbar sind. Nacher können diese ordnungsgemäß geschlossen werden. Wurden alle `LoanWrapper` abgearbeitet, wird in `LiftRules.dispatch` nach übereinstimmenden Dispatch-Funktionen gesucht. Im Falle des ersten Matchs übernimmt eine solche partielle Funktion, im Kontext der `LiftSession` und des `S` Objekts, die Generierung des `LiftResponse`s. Gibt es keine Dispatch-Funktionen oder keine stimmt mit dem Request überein, so wird der Prozess fortgesetzt. Es wird nun ermittelt, um was für ein Request es sich handelt und die entsprechende Aktion wird ausgeführt:

Comet Der Comet-Request wird verarbeitet und führt asynchron die Aktualisierung der Seite durch ohne sie neu zu laden.

Ajax Die Callback-Funktion des Users wird ausgeführt und ihr Ergebnis wird als Response in Form von JavaScript, XML oder eines anderen `LiftResponses` zurückgeliefert.

Regulärer HTTP Request Die genaue Verarbeitung dieses Typs von Request stellt die Abbildung 2.11 dar.

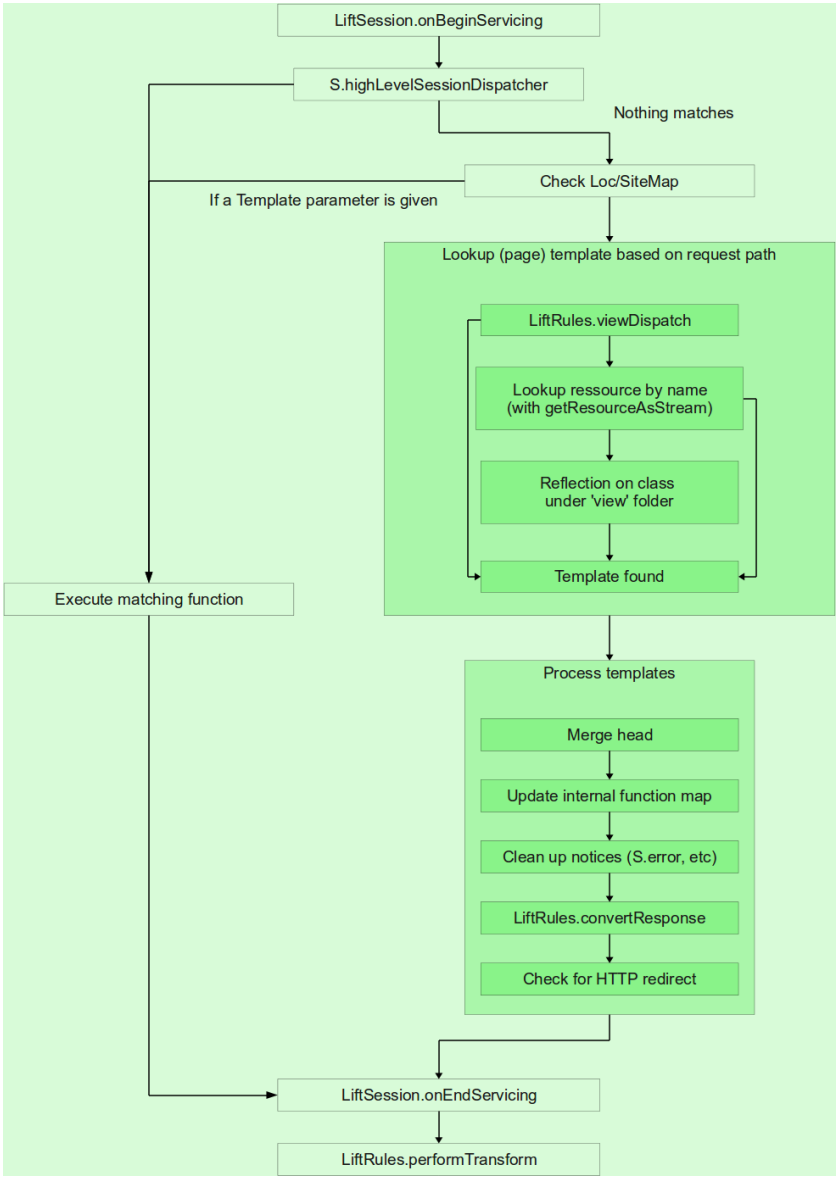


Abbildung 2.11: Verarbeitung des HTTP Requests [CBDW11]

Der Ausführung der in `LiftSession.onBeginServicing` definierten Hook-Methoden folgt die Ausführung der benutzerdefinierten Dispatch-Funktionen. Gibt es eine

Übereinstimmung unter der mit `S.addHighLevelSessionDispatcher` hinzugefügten Funktionen, so übernimmt diese die Erstellung des Responses. Andernfalls wird nach Template Parametern in der `SiteMap` gesucht. Diese können Templates in Form von XML enthalten. Werden keine solchen gefunden, so startet die Suche nach dem Template anhand des Request-Pfads. Das Template wird entweder durch eine `LiftRules.viewDispatch` Funktion, per ServletContext-Funktion `getResourceAsStream` oder durch die Ermittlung der View-Klasse anhand der ersten Komponente des Request-Pfads im per `LiftRules.addToPackages` definiertem Ordner gesucht. Für das gefundene Template startet die Verarbeitung. Es werden Head-Elemente zusammengefügt, benutzerdefinierte interne Elementfunktionen ausgeführt, die bereits gerenderten Fehlermeldungen aufgeräumt und anschließend aus Markup, Headers und Cookies ein `LiftResponse` instantiiert. Zu Schluss wird noch geprüft, ob ein HTTP redirect benötigt wird. Die Endphase dieser Verarbeitung schließt `LiftSession.onBeginServicing`, wo die anschließenden Hook-Methoden ausgeführt werden. Danach wird eine Sequenz von `LiftRules.responseTransformers`-Funktionen ausgeführt, die das Response noch vor dem Abschicken modifizieren können. Damit ist die HTTP-Request Verarbeitung abgeschlossen und der Prozess nimmt weiter den für alle Requests geltenden Pfad. Anschließende `LiftRules.onEndServicing` Hook-Methoden werden ausgeführt, nachdem das zustandshaltende `S` bereits zerstört wurde. Die letzte Möglichkeit, das Response zu verändern, bietet `LiftRules.beforeSend`, danach wird das Response in einen Byte-Stream umgewandelt und weggeschickt. Die in `LiftRules.afterSend` übernehmen die restlichen Aufräumarbeiten, was den Request-Lifecycle endgültig beendet.

2.10.4 View Technik

In Lift basiert die View-Technik auf Templates, Snippets, Views und Comet-Actoren. Die letzten drei davon sind die Komponenten, die einzig und alleine Programmcode enthalten können. Lift nimmt sich hier Apache Wicket als Vorbild und nutzt für seine Templates (X)HTML. Dies soll Kompatibilität mit den herkömmlichen HTML Werkzeugen garantieren und dem Designer eine unbekannte Syntax ersparen. Die Art und Weise, wie die Inhalte der Seiten in Lift generiert werden, unterliegt dem *View-First-Pattern*. Die Idee besteht darin, beim Seitenaufruf zuerst den Inhalt des Templates zu laden und auf Elemente und Methoden zu untersuchen. Danach findet erst die Ausführung der Methoden statt. Mit dieser Vorgehensweise ist es möglich, verschiedene voneinander unabhängige Komponenten in einer Seite zu platzieren. Diese Art von Modularisierung kommt der Wiederverwendbarkeit der Komponenten zugute. Ein einfaches Beispiel stellt die Abbildung 2.12 dar. Es stellt zwei Templates und zwei Snippets dar, wobei sich in einem von den Snippets eine Klasse und ein Objekt befinden. Das *default.html* Template liegt im *templates-hidden* Ordner. Ein direkter Zugriff ist nicht im Fall von diesem Ordner nicht möglich. Das *custompage.html* Tem-

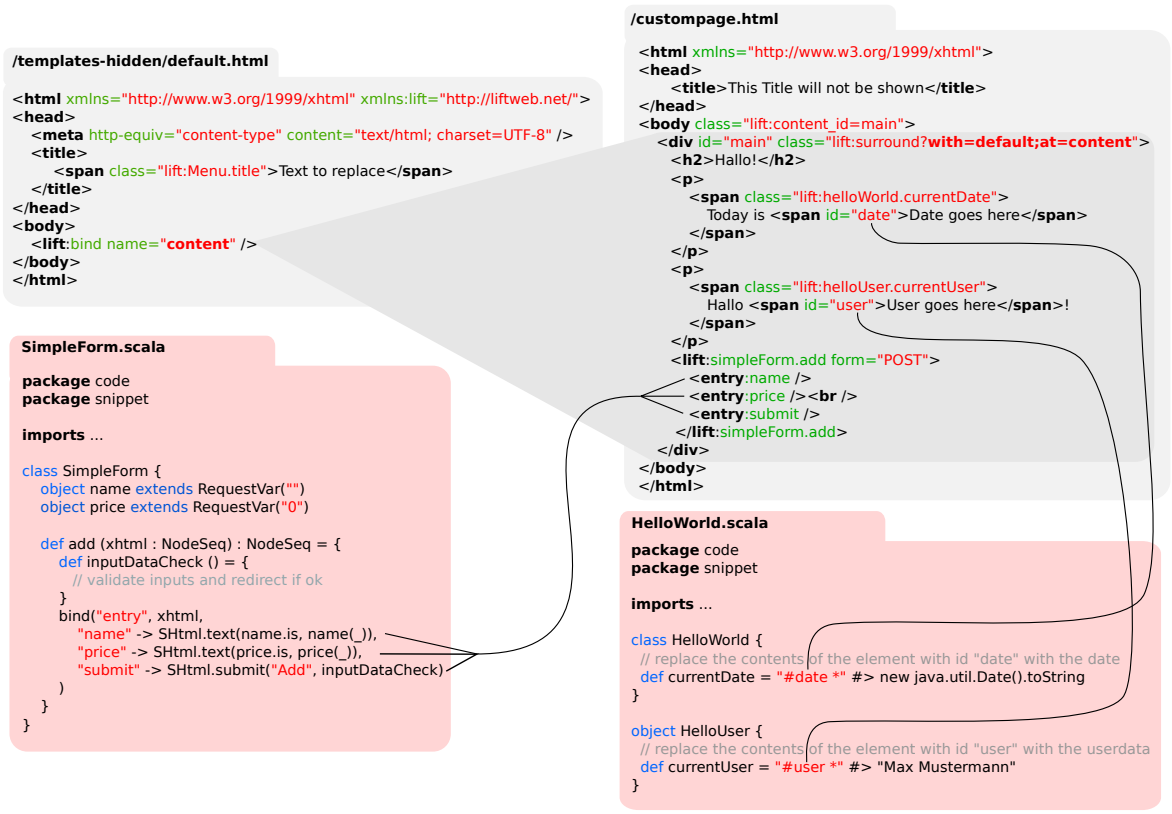


Abbildung 2.12: Beispiel: Seite mit 2 Snippets

plate soll dagegen eine beliebige öffentliche Seite repräsentieren. Das in der *custompage.html* mit `lift:surround?with=default;at=content` markiertes Markup wird in das *default.html* Template an die mit *content* markierte Stelle eingebettet. In der *custompage.html* werden drei Snippet-Methoden aufgerufen. Mit dem Aufruf der Methoden werden die mit den übereinstimmenden IDs markierten Elemente mit generiertem Inhalt ersetzt. In Snippet kann die Ersetzung mittels direkter Eingabe einer ID in Form von `"#id *"` gefolgt von dem Aufruf der Methode `#>` oder mit Hilfe der `bind` Methode erfolgen.

Das war eine Art, wie man Seiten in Lift generieren kann. Eine andere Möglichkeit erlaubt wiederum eine Vermischung von Präsentations-Code und Programmlogik. Dank der Möglichkeit von Scala XML Literale im Programmcode benutzen zu können, werden Markup Komponenten in Scala Klassen direkt erzeugt. Dies geschieht beispielsweise bei den Traits `MetaMegaProtoUser` und `CRUDify`, die für die automatische Generierung von Formularen eingesetzt werden. Ein Beispiel eines im Programmcode generierten Templates zeigt das Listing 2.13. Das Template `custom.html` aus der Abbildung 2.12 kann vollständig im Scala-Code umgesetzt werden. Zugriff auf so realisiertes Template erfolgt in diesem speziellen Fall per `/foo/bar`.

Listing 2.13: LiftView Beispielklasse

```
1  package code
2  package view
3
4  imports ...
5
6  class Foo extends LiftView {
7
8     override def dispatch: PartialFunction[String, () => Box[NodeSeq]] = {
9        case "bar" => () => Full(bar)
10       case _ => () => Empty
11    }
12
13    def bar(): NodeSeq = {
14       <div id="main" class="lift:surround?with=default;at=content">
15          <h2>Hallo!</h2>
16          <p>
17             <span class="lift:helloWorld.currentDate">
18                Today is <span id="date">Date goes here</span>
19             </span>
20          </p>
21          <p>
22             <span class="lift:helloUser.currentUser">
23                Hallo <span id="user">User goes here</span>!
24             </span>
25          </p>
26          <lift:simpleForm.add form="POST">
27             <entry:name />
28             <entry:price /><br />
29             <entry:submit />
30          </lift:simpleForm.add>
31       </div>
32    }
33 }
```

Nach [FK11] bestätigt dieser Aspekt die Tatsache, dass es sich bei Lift ”ausdrücklich nicht um ein Model-View-Controller-Framework” handelt, da durch die Generierung des Markups im Programmcode in Lift an vielen Stellen mit diesem Pattern gebrochen wird. Der durch View-First modifizierte MVC Ansatz (siehe Abbildung 2.13) eignet sich nach [Get11] besonders für View-Driven-Development. Die im Produkt eingebauten Möglichkeiten sind damit für das Rapid-Prototyping geeignet.

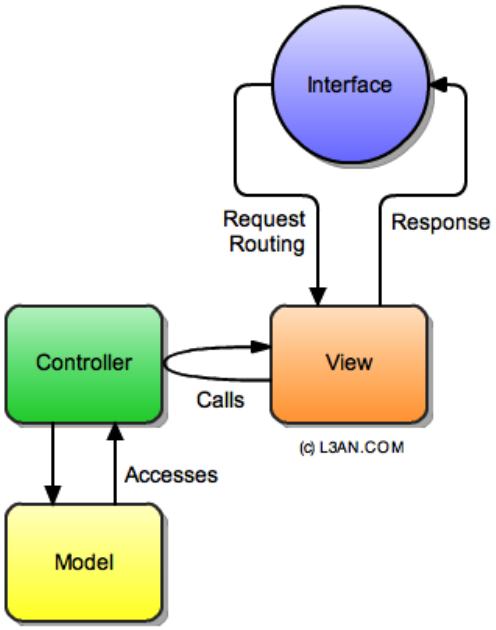


Abbildung 2.13: View First MVC (vgl. [Get11])

Kapitel 3

Szenario

3.1 Grundlegende Anforderungen

Ein Szenario, welches als Grundlage für einen wissenschaftlichen Vergleich zwischen unterschiedlichen Technologien dienen soll, muss bestimmten Anforderungen genügen. Damit soll gewährleistet werden, dass eine zu den ausgewählten Technologien passende Testvorlage eingesetzt wird. Mit Hilfe von dieser Vorlage sollen die Unterschiede zwischen den Technologien ermittelt werden. Nachstehend werden die für den Vergleich zwischen den ausgewählten Web-Frameworks grundlegenden Anforderungen festgelegt.

Gewährleistung der Allgemeingültigkeit Damit soll das Szenario bestenfalls beliebige Interpretationen erfüllen und nicht nur für einen speziellen Zweck ausgelegt sein.

Abstraktion bzw. Übertragbarkeit Es soll möglich sein, das Szenario oder dessen Aspekte in ein anderes Szenario zu übertragen.

Konkretheit Trotz Abstraktion soll das Szenario für eine Implementierung von konkreten Funktionen ausgelegt sein.

Mehrbenutzerbetrieb Auf Grund des WWW Gedankens muss das Szenario für simultanen Mehrbenutzerbetrieb ausgelegt sein.

Autorisierter Zugriff Das Szenario soll Aspekte enthalten, die nur über eine Autorisierung erreichbar sind.

Sichere und kontrollierte Datenmanipulation Veränderungen der im System enthaltenen Daten sollen unter Berücksichtigung des Sicherheitsaspekts erfolgen.

3.2 Auswahl des Szenarios

Für den Vergleich zwischen den jeweiligen Java und Scala Web Application Frameworks wird eine von Sun Microsystems entwickelte Referenzanwendung namens *Java Pet Store*[45] adaptiert. Es handelt sich dabei um eine Referenzanwendung, die entworfen wurde um die Fähigkeiten der Java Enterprise Edition 5 Plattform zu illustrieren und um zu demonstrieren, wie darauf basierende, AJAX-fähige Web 2.0 Anwendungen entwickelt werden können. Diese Beispielanwendung ist eine fiktive Spezialisierung eines Webshop-Szenarios, mit welchem viele Internetnutzer oft konfrontiert werden. Es ist somit auch ein allgemeines Beispiel, dessen konkrete Funktionen in andere Szenarien übertragen werden können. Damit wird das Reinstellen sowie das Abrufen von digitalen Informationen gemeint, was in vielen anderen Web-Szenarien stattfindet. Des Weiteren ist das Konzept des Webshops gerade für einen Mehrbenutzerbetrieb ausgelegt. Dabei spielt des Aspekt der Sicherheit oft eine wichtige Rolle. Effizienz und Skalierbarkeit sind hier weitere wichtige Aspekte, da man beachten muss, mit was für einem Aufkommen von Benutzern die großen Webshops heutzutage klarkommen müssen. Die im Abschnitt 3.1 erwähnten Anforderungen würden damit erfüllt werden.

3.3 Beschreibung des Szenarios

Auf Grund des beschränkten Rahmens dieser Arbeit werden nicht alle Komponenten der Pet-Store Referenzanwendung realisiert. In dieser Arbeit soll die Realisierung zum Benchmarkzweck und weniger als Beispiel zu Referenzieren dienen. Es werden deshalb nur die für den eigentlichen Vergleich wichtigen Komponenten mit evtl. Anpassungen prototypisch realisiert.

So gibt es einen Produktkatalog, in dem die Produkte aufgelistet werden und von jedem Besucher angeschaut werden können. Die Produkte werden in Kategorien unterteilt. Um die Produkte zu kaufen oder zu verkaufen, muss sich der Benutzer registrieren. Dazu steht ein entsprechendes Formular zur Verfügung. Ist der Benutzer registriert und in das System eingeloggt, stehen ihm die Möglichkeiten zum Kaufen und Verkaufen zur Verfügung. Üblicherweise werden in realen Ausführungen dieses Szenarios mehrere Rollen vorgesehen. Für das Szenario reichen zwei Basisrollen aus, da weitere Spezialisierungen keinen entscheidenden Mehrwert einbringen würden. Im nächsten Abschnitt werden diese kurz erläutert.

3.3.1 Szenarioaktoren

In dem Webshop Szenario werden einfachheitshalber nur die zwei folgenden Rollen definiert. Diese sind ausreichend, um die wichtigsten Funktionen eines Webshops zu präsentieren.

[45] Pet Store Article - `http://java.sun.com/developer/technicalArticles/J2EE/petstore/`

Gast ist die Standardrolle jedes Besuchers des Webshops. Ein Gast kann ohne Registrierung lediglich durch den Produktkatalog navigieren und nach den Produkten suchen.

Kunde ist einfachheitshalber die Rolle des registrierten Benutzers. Ein Kunde kann die Produkte in den Katalog einstellen sowie andere Produkte bestellen.

3.3.2 Anwendungsfalldiagramm

Die Abbildung 3.1 bietet einen Überblick über die grundlegenden Funktionen in dem Webshop-Szenario in Form eines Anwendungsfalldiagramms. Eine detaillierte Beschreibung der wesentlichen Teile dieses Diagramms befindet sich im Anhang B.

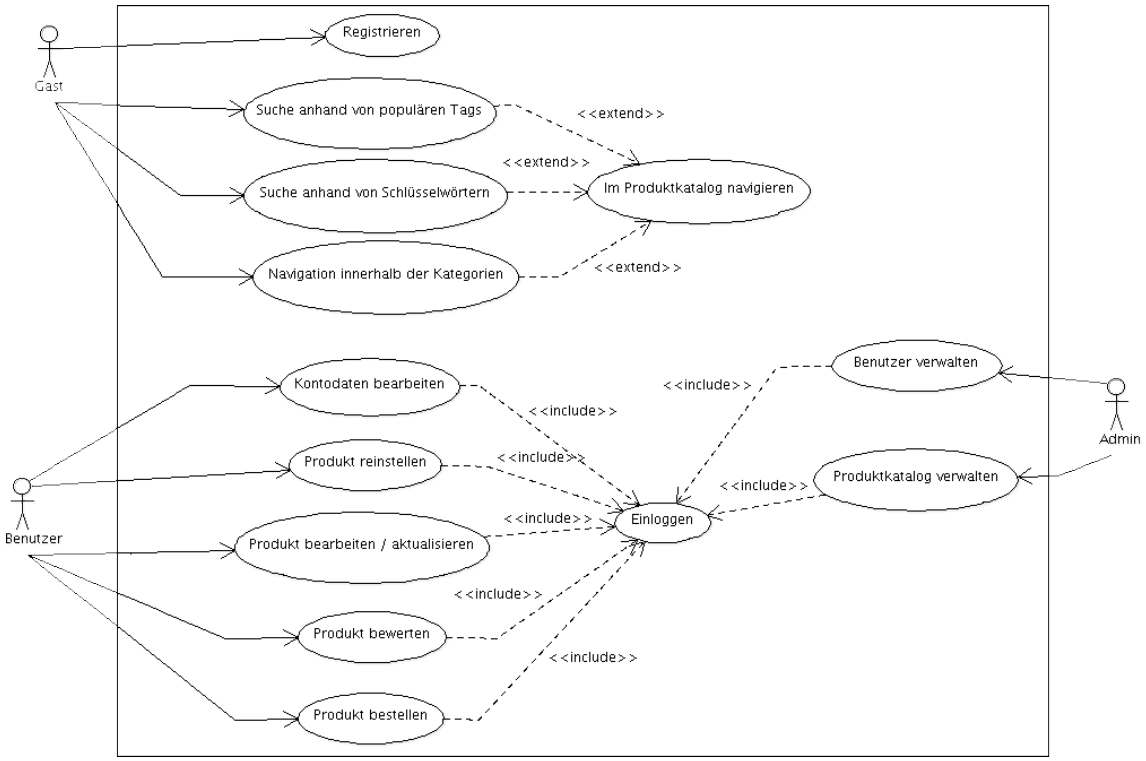


Abbildung 3.1: Rollenbasiertes Anwendungsfalldiagramm für das Webshop-Szenario

3.3.3 Aktivitätsdiagramme

Um sich ein besseres Bild von den Aktivitäten, die in dem Szenario realisiert werden können, zu machen, werden nachfolgend die wichtigsten Aktivitäten in den dafür entsprechenden Diagrammen vorgestellt. Die Abbildung 3.2 enthält ein Aktivitätsdiagramm mit den Aktionen, die ein User tätigen muss, um ein Pet in den Katalog zum

Verkauf einzutragen. Die darauf folgende Abbildung 3.3 zeigt, von zwei möglichen Einstiegspunkten aus gesehen, die Aktivitäten, die zu einem möglichen Pet-Kauf führen.

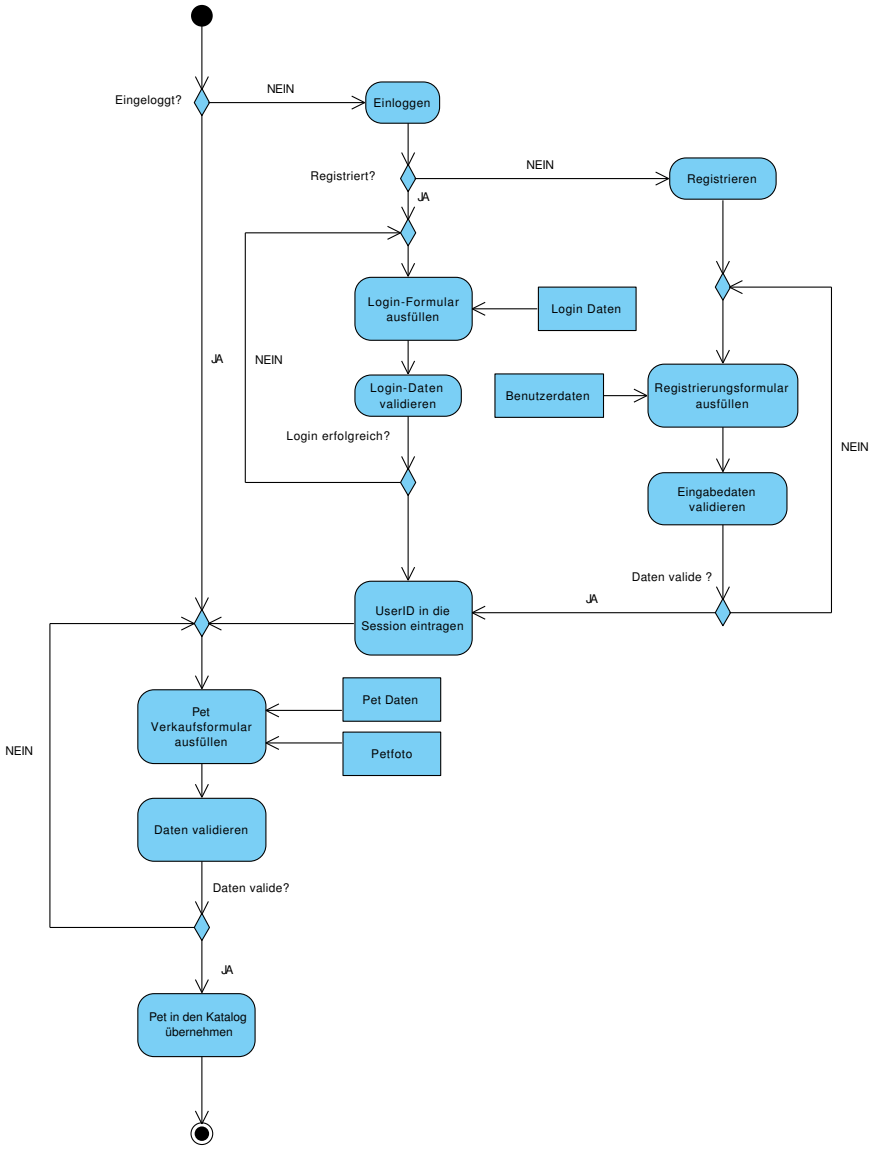


Abbildung 3.2: Aktivitätsdiagramm Verkaufsaktion

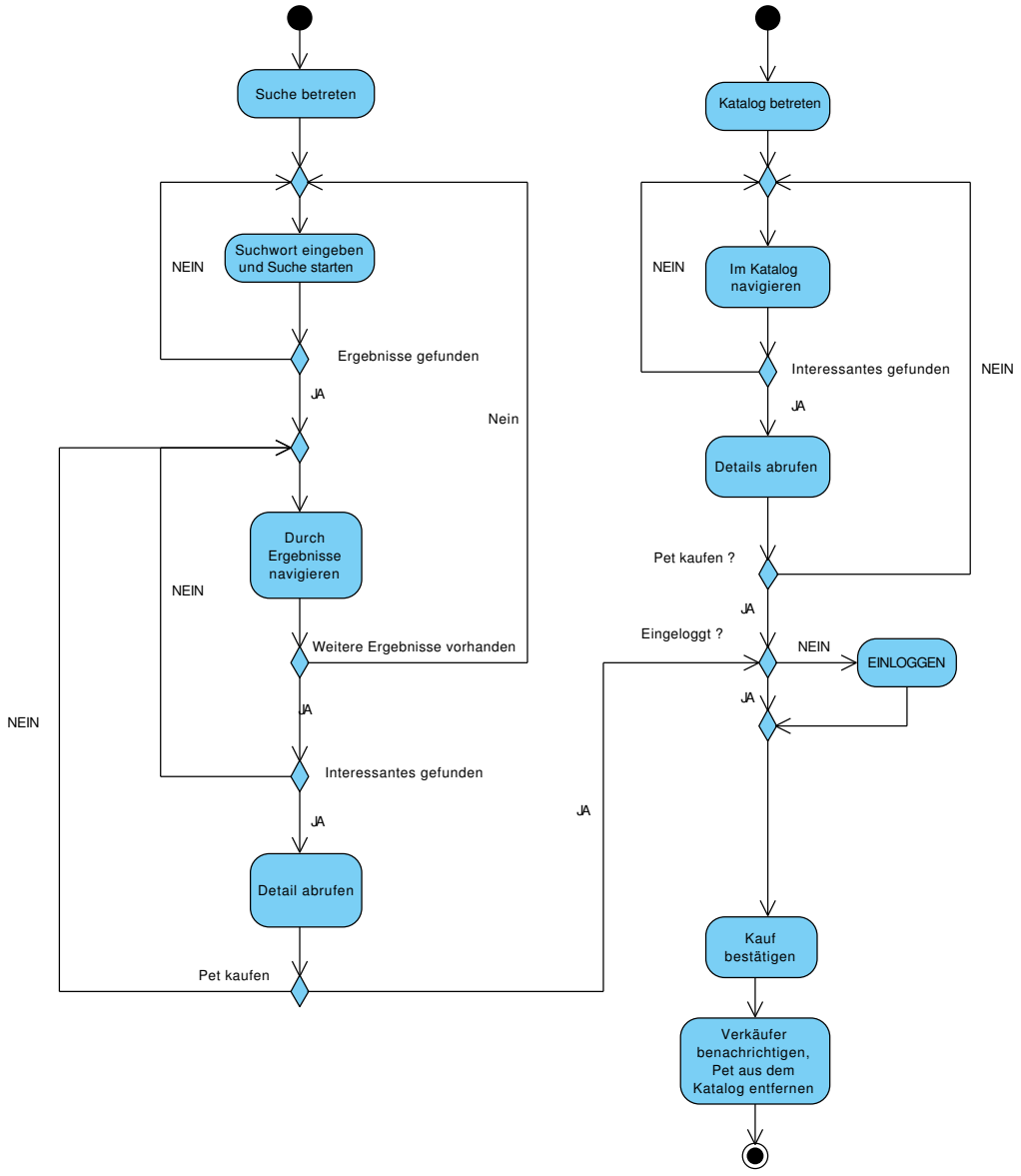


Abbildung 3.3: Aktivitätsdiagramm Kaufaktion

Kapitel 4

Vergleichsverfahren

4.1 Goal Question Metric (GQM)

Zur Ausführung einer systematischen Software-Messung, die zum späteren Softwarevergleich herangezogen wird, benötigt es entsprechender Software-Maße. Dabei können Größen wie z.B. Speicherbedarf, Responsezeiten und nach [Sim01] sogar Checklisten bzw. andere Überprüfungstechniken als Alternativen eingesetzt werden. Die Software-Maße unterliegen wiederum bestimmten Kriterien, z.B. entspricht das Zeit- und Verbrauchsverhalten der Effizienz. Die relevanten Kriterien, die zur Software-Messung herangezogen werden sollen, können aus dem Kontext der Ziele, die man mit der Messung erreichen will, extrahiert werden. Ein Verfahren, welches zu diesem Zweck angewandt werden kann, wurde 1984 von Victor R. Basili und David Weiss entwickelt. Das Konzept von GQM legt eine Grundlage für ein Messverfahren von Software fest. Die Idee von GQM basiert auf der Herleitung der Software-Messungen aus den gesetzten Zielen, die man mittels Durchführung des Messprozesses erreichen will. Ein Vergleich von unterschiedlicher Software bietet in diesem Fall ein relativ großes Spektrum an Vergleichsmöglichkeiten. Der Fokus kann sowohl auf Quantität (Code-Umfang, Anzahl-Module, -Klassen etc.) als auch auf Qualität (Architektur-Stil, Coding-Style-Guidelines, Dokumentationsrichtlinien etc.) gesetzt werden. Der GQM-Ansatz wird unter anderem auch dazu verwendet, den Prozess der Software Entwicklung zu verbessern, um die gesetzten Geschäftsziele erreichen zu können. Das GQM-Verfahren besteht aus drei Schritten (vgl. [EDBS05]):

Ziele setzen z.B. bessere Performance, Kundenzufriedenheit erhöhen, Produktivität steigern.

Fragen stellen wie können die Ziele erreicht werden, z.B. welche Komponenten verlangsamen das System? Ist die Anwendung benutzerfreundlich genug?

Metriken bestimmen Die gestellten Fragen sollen damit beantworten werden, z.B. Performance-Test, Usability-Test.

Die Abbildung 4.1 zeigt eine Topologie der Komponenten der GQM, welche die Intentionen der Norm ISO 15939 - "Systems and software engineering – Measurement process" berücksichtigen soll. Mit GQM gibt es also ein Vorgehenstemplate, welches

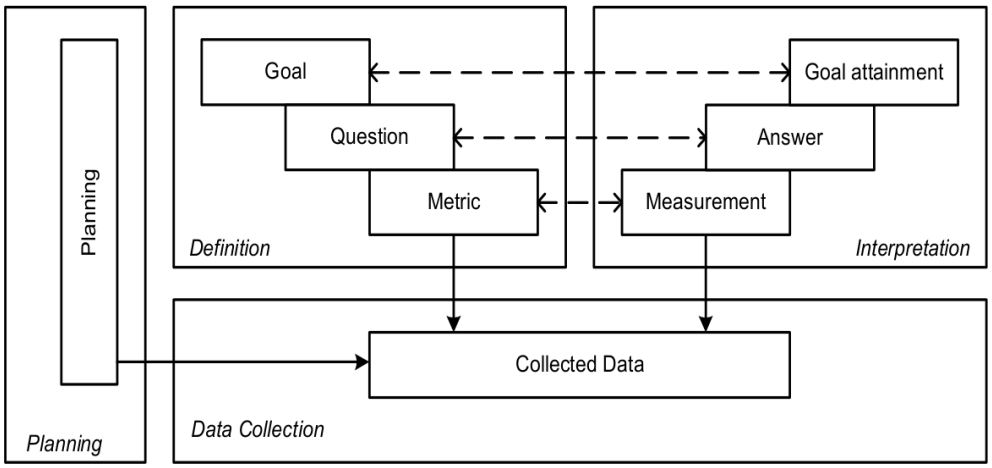


Abbildung 4.1: Topologie der Goal Question Metric nach ISO 15939 (vgl. [EDBS05])

mit Zielen, Fragen und vor allem Metriken gefüllt werden kann. Welche Metriken möglich sind, erläutert der nachfolgende Abschnitt.

4.2 Mögliche Vergleichskriterien

Um Software anhand der Kriterien zu vergleichen bietet es sich an, die Norm `ISO/IEC 9126` zu berücksichtigen. Als Leitfaden für die Bewertung von Softwareprodukten legt diese Norm die sechs Hauptqualitätsmerkmale für Software fest: Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz, Wartbarkeit[46] und Portabilität[47]. Die sechs Hauptmerkmale unterteilen sich in weitere Teilmerkmale (siehe Abbildung 4.2). In den nachfolgenden Unterabschnitten werden diese Merkmale und Teilmerkmale angelehnt an [Mal07] beschrieben und auf Eignung als Vergleichskriterium geprüft. Dabei wird festgelegt ob die Auswertung des Merkmals sich auf die zu untersuchende Umsetzung oder auf die dazu verwendete Technologie bezieht.

4.2.1 Funktionalität

Das Vorhandensein von Funktionen, die festgelegte und vorausgesetzte Erfordernisse erfüllen [Fra07]. Die Funktionalität der Software unterteilt sich in:

[46] Synonym zu Änderbarkeit
[47] Synonym zu Übertragbarkeit

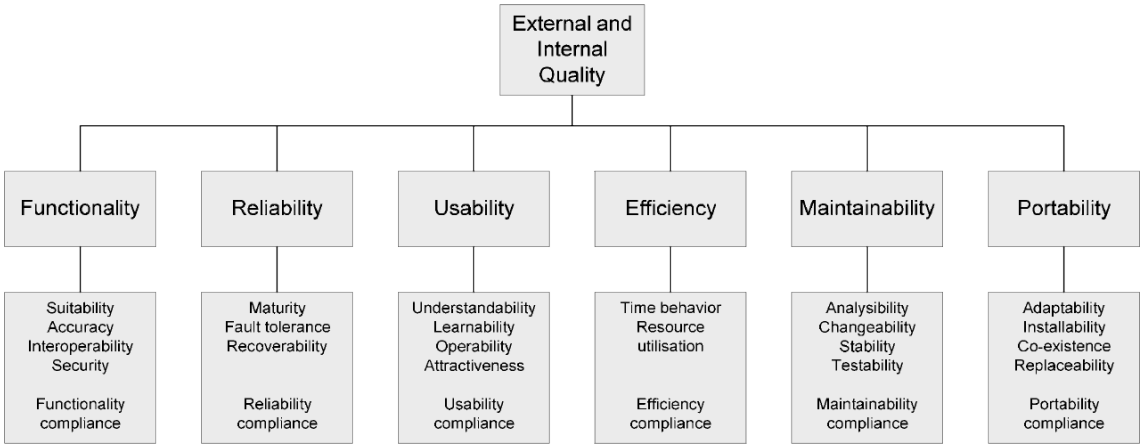


Abbildung 4.2: Software-Qualitätsmerkmale [Mal07]

Angemessenheit Präzision und Korrektheit der Menge der implementierten Funktionen.

Genauigkeit Genauigkeit und Präzision der implementierten Funktionen.

Interoperabilität Korrektheit der Implementierung der Funktionen zum Austauch von Daten und der Austauschformate.

Sicherheit Vollständigkeit der Benutzerzugriffe auf das System, Kontrollierbarkeit und Protokollierbarkeit der Daten und möglichst geringe Häufigkeit der korrupten Daten.

Um die Funktionalität und ihre einzelnen Teilmerkmale zu messen kann, auf die Web-Anwendungen bezogen, eine Reihe von Tests durchgeführt werden. [Fra07] listet Tests der Klassen, Komponenten, Integration bis hin zu umfangreichen funktionalen Systemtests, welche Links, Cookies, Plugins und Sicherheitsvorkehrungen überprüfen auf.

Auf die eingesetzten Technologien bezogen, bietet es sich an, eben das Vorhandensein von Funktionen, die festgelegte und vorausgesetzte Erfordernisse erfüllen, zu überprüfen. Einige davon, die unter dieses Kriterium fallen, wurden im Abschnitt 2.8 als Auswahlkriterien definiert und können zur Bewertung herangezogen werden. Es handelt sich dabei um die View-Technik, AJAX-Unterstützung, Internationalisierung und Validierungsmechanismen. Zusätzlich wird der Aspekt der Persistenz hier als letztes Bewertungskriterium aufgenommen, welcher als fester Bestandteil beinahe zu jeder Webanwendung gehört.

4.2.2 Zuverlässigkeit

The capability of the software product to maintain a specified level of performance when used under specified conditions. (NORM ISO/IEC 9126 Teil 1, S.8) Die Zuverlässigkeit unterteilt sich in folgende Teilmerkmale:

Reife Mit der fallenden Anzahl der Fehler in der Anwendung steigt ihre Reife.

Fehlertoleranz Je weniger Ausfälle aufgrund von Fehlern in einer Anwendung auftreten, desto besser ist die Fehlertoleranz.

Wiederherstellbarkeit Lange Verfügbarkeit, kurze Wiederanlaufzeiten bei einem Ausfall.

Sinngemäß bezieht sich das Testen der Zuverlässigkeit auf das fertige Produkt. [Fra07] definiert Tests für Ausfallsicherheit und Verfügbarkeit. Dabei werden solche Aspekte wie Ausfall der Server oder des Load Ballancers im Cluster in Betracht gezogen sowie Formeln für die Ermittlung der Verfügbarkeit und der Ausfallrate.

$$Verfuegbarkeit = MTBF^{48} \div (MTBF + MTTR^{49}) \tag{4.1}$$

$$Ausfallrate = 1 - Verfuegbarkeit \tag{4.2}$$

Diese Aspekte sind auf Grund des kleinen Umfangs der Umsetzung für diese Arbeit nicht relevant.

4.2.3 Benutzbarkeit

The capability of the software product to be understood, learned, used and attractive to the use, when used under specified conditions. (NORM ISO/IEC 9126 Teil 1, S.8)

Verständlichkeit Vollständigkeit der Produktdokumentation und selbsterklärende bzw. gut nachvollziehbare Funktionen.

Erlernbarkeit Vollständigkeit der Dokumentation bzw. der Hilfe Funktionen.

Bedienbarkeit Komfort und gut nachvollziehbare Interaktion mit dem Produkt.

Attraktivität Ansprechende und anpassbare Benutzeroberfläche des Produkts.

Die Tests für die Benutzbarkeit bzw. Gebrauchstauglichkeit beziehen sich üblicherweise auf die Anwendungen, mit denen die User konfrontiert werden. Im Fall dieser Arbeit und für dieses Kriterium bietet es sich an, den Fokus auf die eingesetzten Technologien zu richten. Schließlich muss die Umsetzung des Szenarios gleichen Inhalt, möglichst

[48] Mean Time Between Failure
[49] Mean Time To Repair

gleiche Oberfläche (z.B. konform mit der Norm ISO 9241 Teil 10 - *Grundsätze der Dialoggestaltung*) und gleiche Funktionen aufweisen.

Der Grad der Relevanz dieses Qualitätmerkmals für diese Arbeit steigt, wenn man die eingesetzten Technologien in Betracht zieht. Die vier Qualitätsteilmerkmale der Benutzbarkeit müssen aber in diesem Fall entsprechend interpretiert werden. Dies kann für die einzelnen Teilmerkmale folgend umgesetzt werden:

Verständlichkeit und Erlernbarkeit Checkliste über die verfügbaren Bücher, Dokumentationen, Beispiele und eine auf Erfahrung basierte Definition der Lernkurve.

Bedienbarkeit Auswertung der vorhandenen Entwicklungswerkzeuge und ihrer Qualität.

Attraktivität Checkliste über die speziellen Ansätze, Funktionen des Frameworks bzw. der dahinter stehenden Technologie.

4.2.4 Effizienz

The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions. (NORM ISO/IEC 9126 Teil 1, S.10)

Zeitverhalten (Performanz) Kurze Reaktionszeiten, bei schneller Verarbeitung von vielen Anfragen innerhalb einer kleinen Zeiteinheit.

Verbrauchsverhalten Grad der Ausnutzung von Ressourcen.

Um den Grad der Effizienz zu bestimmen, kann eine Reihe von Test durchgeführt werden. Von [Fra07] werden folgende Tests definiert, um dieses Software-Qualitätsmerkmal zu testen.

Performance- / Lasttest Überprüfung des Zeitverhaltens, der Mengenverarbeitung und des Ressourcenverbrauchs eines Systems unter normalen Systembedingungen als auch bei steigender Systemlast. Um quantitative Aussagen zu dieser Art des Tests zu treffen gibt es drei Metriken:

- RPS (Requests Per Second) - Gibt Anzahl der bearbeiteten Anfragen pro Sekunde an. Zeigt das Interaktionsausmaß zwischen dem Browser und dem Webserver. Der Webserver ist am Limit, wenn diese Größe bei steigender Last konstant bleibt.
- CPU - Prozessorauslastung des Servers in Prozent.
- QR (Queued Requests) - Gibt die Anzahl der Abfragen in der Warteschlange an. Sollte die Anzahl über eine längere Zeit größer Null sein, bedeutet es, dass der Server unter ”Stress” steht.

Skalierbarkeitstest Überprüfung der Fähigkeit, ein System auf Rechnern von unterschiedlicher Größe einsetzen zu können. Dabei kann untersucht werden, um welchen Faktor das System schneller arbeitet, wenn man z.B. die Anzahl von Prozessoren verdoppelt. Je größer der Faktor, desto besser skaliert das System. Andererseits kann man überprüfen, wie sich das System bei einer Verdopplung der Last (Anzahl User/ Transaktionen) verhält. Kommt es mit doppelten Hardware-Ressourcen nicht aus, so skaliert es nicht optimal und es muss evtl. explizite Eingriffe in die Software geben.

Speicherlecktest Überprüfung eines auf Dauer angelegten Performanztests auf das Entstehen von Speicherlecks. Die sog. "memory leaks " können auf Grund von Software-Fehlern entstehen, die Teile des Arbeitsspeichers nach der Benutzung nicht freigeben. Nach bestimmter Zeit steht dem Web-Server kaum Arbeitsspeicher zur Verfügung, sodass ein weiterer Betrieb nicht möglich ist und es im schlimmsten Fall zum Absturz kommen kann. Speicherlecks können daran erkannt werden, dass Antwortzeiten oder Übertragungsraten mit der Zeit nachlassen. Ein solcher Test sollte über einen längeren Zeitraum durchgeführt werden.

Von den drei Teilmerkmalen ist auf Grund des Umfangs nur der Performance- / Lasttest sinnvoll. Dieser bezieht sich aber speziell auf die im Laufe dieser Arbeit implementierte Umsetzung des Szenarios.

4.2.5 Änderbarkeit

The capability of the software product to be modified. (NORM ISO/IEC 9126 Teil 1, S.10)

Analysierbarkeit Identifikation der Fehler und die Analysierbarkeit der Auswirkungen von Änderungen bzgl. der Anforderungen.

Modifizierbarkeit Behebbarkeit der Fehler oder Umsetzbarkeit der Änderungen bzgl. der Anforderungen.

Stabilität Seiteneffektfreiheit im Kontext der Behebung von Fehlern und Umsetzung der Änderungen bzgl. der Anforderungen.

Testbarkeit Ausführbarkeit von Tests.

Die Änderbarkeit bzw. Wartbarkeit der Software wird unter anderem mit Hifle sog. statischen Code-Analysen überprüft. Diese werden mit dem Ziel der Ermittlung der Komplexität des Source-Codes durchgeführt. Das Ziel kann unter anderem sein, Rückschlüsse ziehen zu können, wie schnell sich Änderungen an der Software umsetzen lassen. Zu diesem Zweck wurden verschiedene Metriken entwickelt, die sich auf die Source-Code Analyse spezialisieren:

LOC- und NCSS-Metrik *Lines of Code* Metrik, auch Zeilenmetrik genannt, ist als einfachste Metrik für die Bestimmung der Programmkomplexität gedacht. Eine Weiterentwicklung der LOC-Metrik ist die Non Commented Source Statements Metrik. NCSS ignoriert im Vergleich zu LOC-Metrik sämtliche Kommentare im Source-Code. Die Aussagekraft dieser Metriken kann in bestimmten Fällen gering sein. Faktoren wie Programmierstil und Ausdruckstärke der Programmiersprache werden durch diese Metriken nicht berücksichtigt. So sind diese Metriken eher als Volumenmetriken zu gebrauchen als um die Programmkomplexität zu bestimmen.

Halstead-Metriken Ein Satz von Metriken wurde von Maurice Howard Halstead vorgestellt, mit welchen der Zusammenhang zwischen der Programmkomplexität und der Auftrittscharakteristik verschiedener lexikalischer Elemente hergestellt werden sollte. Halstead trennt die Elemente des Source-Codes in Operatoren und Operanden (siehe Abbildung 4.3). Daraus werden vier Basis-Größen für Halsteads Metrik extrahiert:

$$\eta_1 = Anzahl\, unterschiedlicher\, Operatoren \tag{4.3}$$

$$\eta_2 = Anzahl\, unterschiedlicher\, Operanden \tag{4.4}$$

$$\mathrm{N}_1 = Gesamtzahl\, der\, Vorkommen\, aller\, Operatoren \tag{4.5}$$

$$\mathrm{N}_2 = Gesamtzahl\, der\, Vorkommen\, aller\, Operanden \tag{4.6}$$

Letztendlich ermöglichen Halsteads Metriken folgende Aussagen zu treffen:

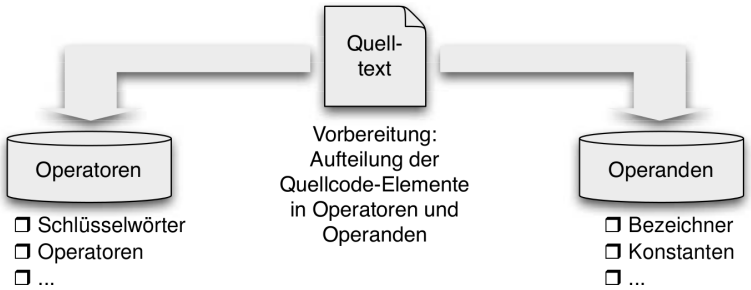


Abbildung 4.3: Halsteads Aufteilung des Source-Codes (vgl. [Hof08])

$$Laenge\, des\, Programms : N = \mathrm{N}_1 + \mathrm{N}_2 \tag{4.7}$$

$$Volumen\, eines\, Programms : V = N \times \log_2 \eta \tag{4.8}$$

$$Minimalvolumen\, eines\, Programms : V^* = \eta^* \times \log_2 \eta^* = (2+\eta_2) \times \log_2 (2+\eta_2) \tag{4.9}$$

$$Level\, der\, Implementierung : L = \frac{V^*}{V} \tag{4.10}$$

$$Schwierigkeitsgrad(difficulty) : D = \frac{1}{L} \tag{4.11}$$

$$Programmieraufwand(effort) : E = V \times D \tag{4.12}$$

Im Bezug auf die eingesetzten Technologien ist es ebenso relevant zu bewerten, wie diese zu den in diesem Abschnitt vorgestellten Merkmalen beitragen. So bietet es sich an, z.B. in Form einer Checkliste zu bewerten welche Testmöglichkeiten in Hinsicht auf die Testbarkeit zur Verfügung stehen.

4.2.6 Übertragbarkeit

The capability of the software product to be transferred from one environment to another (NORM ISO/IEC 9126 Teil 1, S.11)

Anpassbarkeit Grad der Anpassung an andere Einsatzbedingungen (z.B. veränderte Hardware, Betriebssystem, organisatorische Einheiten, Benutzerzahlen).

Installierbarkeit Schwierigkeit bzw. Einfachheit der Installation innerhalb einer Systemumgebung.

Koexistenz Grad der Einschränkungen bei dem Einsatz mit anderen Softwareprodukten innerhalb einer Systemumgebung.

Austauschbarkeit Möglichkeit eines Austausches gegen eine andere Anwendung, bzw. der Teile einer Anwendung durch andere Teile.

Durch den eingeschränkten Rahmen dieser Arbeit und die geringe Relevanz fällt der Aspekt der Anpassbarkeit als Vergleichsmerkmal weg. Dies ist darauf zurückzuführen, dass es nur ein Testsystem geben wird, auf dem die Anwendung und die dazu benötigten Komponenten installiert werden ohne die Möglichkeit der späteren Änderungen an der Hardware.

Im Bezug auf die Übertragbarkeit definiert [Fra07] Tests für die Installierbarkeit und Deinstallierbarkeit. Anhand von Checklisten können die beiden Operationen genauer getestet werden. Eine quantitative Aussage kann im Bezug auf den Teilmerkmal Installierbarkeit getroffen werden. Es ist möglich für die beiden Web Application Frameworks den Installationsaufwand in Zeiteinheiten zu messen, um die Installationskomplexität vergleichen zu können. Je mehr Zeit gebraucht wird, in der das System bis zur Ausführung einer "Hello World!"-Beispielanwendung benötigt wird, desto höher ist die Installationskomplexität. Zwar analog dazu, aber nicht im Kontext dieses Kriteriums kann die Implementierungskomplexität ermittelt werden. Diese kann in Zeiteinheiten ab der erfolgreichen Instandsetzung des Systems bis zur fertigen Umsetzung gemessen werden.

Auf die eingesetzten Technologien bezogen ist der Aspekt der Austauschbarkeit interessant. Dem kann der Begriff Lose Kopplung zugeordnet werden, denn je loser

die Kopplung zwischen den Komponenten, desto einfacher ist die Möglichkeit diese auszutauschen.

4.3 Verfahrenskonkretisierung

Im ersten Abschnitt dieses Kapitels wurde die Basisidee des Verfahrens vorgestellt, mit dessen Hilfe das Ziel dieser Arbeit erreicht werden soll. Kriteriengestützt soll quantitativ ermittelt werden, wie sich Scala im Vergleich zu Java in dem gerade von Java dominierten Bereich der serverseitigen Programmiersprachen, mit dem Fokus auf die Web-Technologien, schlägt. Die Umsetzung des im Abschnitt 3 beschriebenen Referenzszenarios soll dabei als Vergleichsgegenstand fungiern. Die daraus resultierende Software muss dementsprechend gemessen werden, um einen Vergleich realisieren zu können. Dazu wurden im Abschnitt 4.2 die von der Norm `ISO 9126` definierten Softwarequalitätsmerkmale als mögliche Vergleichskriterien erläutert und in der Hinsicht auf Herleitung von quantitativen Aussagen analysiert. In diesem Abschnitt wird das Vergleichsverfahren in Bezug auf den speziellen Vergleich, der in dieser Arbeit durchgeführt wird, konkretisiert. Die Relevanz der Vergleichskriterien für diesen Fall ergibt sich durch die Berücksichtigung folgender Annahmen:

- Es werden unterschiedliche Web-Technologien verglichen, die auf zwei verschiedenen Sprachen basieren, welche wiederum auf derselben Plattform laufen.
- Für jede Umsetzung gelten identische Anforderungen an Funktionalität.
- Für jede Umsetzung gelten identische Anforderungen an Benutzbarkeit.
- Auf Grund des Rahmens, dieser Arbeit können keine zeitintensiven Tests, die über einen längeren Zeitraum laufen, ausgeführt werden.

Die Vergleichskriterien, die sich auf Grund der o.g. Annahmen ergeben, sind somit Funktionalität, Effizienz, Änderbarkeit, Übertragbarkeit und zu guter Letzt der Entwicklungsaufwand. Sinngemäß dazu wird in den nachfolgenden Abschnitten eine entsprechende Goal Question Metrik entwickelt.

4.3.1 Festlegung der Ziele

Die Festlegung der Ziele erfolgt aus der Perspektive der Instanz, die unmittelbaren Einfluss auf den Erfolg der Durchsetzung von Technologien in Projekten hat. Es handelt sich dabei um die Perspektive des Projekt-Managers. Im ersten Schritt dieses Verfahrens wird das Hauptziel bzw. die Hauptziele definiert. Logischerweise ist es für einen Projekt-Manager unter anderem wichtig, ein performantes und effizientes Produkt schnell auf den Markt zu bringen. Es könnte auch wichtig sein mit dem Produkt als Pionier auf dem Markt zu erscheinen und es nach und nach später zu verbessern

und zu erweitern. Zusammengefasst könnte die Definition von den Hauptzielen aus Sicht eines Projekt-Managers im Bezug auf die Entwicklung eines neuen Produkts folgendermaßen aus:

Ziel 1 kurze Time-to-Market

Ziel 2 gute Performance

Ziel 3 gute Effizienz

Mit Time-to-Market ist hier die Dauer von der Produktentwicklung bis zur Plazierung des Produkts auf dem Markt gemeint. Mit der Performance, in Hinsicht auf die Web-Technologien, sind die Antwort, bzw. die Reaktionszeiten gemeint. Mit Effizienz wird das Speicherverhalten festgelegt. Zur besseren Verständlichkeit muss der Time-to-Market Aspekt hinsichtlich der damit verbundenen Teilaspekte verfeinert werden. Folgende Aspekte spielen dabei eine wichtige Rolle: Produktionsbereitschaft (Installierbarkeit, Portierbarkeit), Einarbeitungszeit, die Änderbarkeit. Sie werden nun als folgende Unterziele aufgenommen:

Ziel 1.1 schnelle Produktionsbereitschaft

Ziel 1.2 geringe Szenarioumsetzungsgszeit

Ziel 1.3 einfache Änderbarkeit

Zu den festgelegten Zielen werden im nachfolgenden Abschnitt die Fragen definiert.

4.3.2 Herleitung der Fragen

Um die ausgewählten Technologien anhand der gesetzten Ziele vergleichen zu können müssen nun Fragen definiert werden, was man im Bezug auf die eigentlichen Ziele wissen will. Die Fragen werden nun in tabellarischer Form definiert (siehe Tabelle 4.3.2). Im nächsten Abschnitt werden die für die Beantwortung der Fragen relevanten

Frage	Ziel	Abgeleitete Frage
1	1.1	Wie viel Aufwand muss für die Produktionsbereitschaft des Frameworks getrieben werden ?
2	1.2	Wie viel Aufwand muss getrieben werden, um das Szenario umzusetzen ?
3	1.3	Wie schnell lassen sich die Änderungen am bestehenden Code vornehmen ?
4	1.3	Welchen (physikalischen) Umfang hat die Umsetzung ?
5	2	Wie schnell werden die Seiten geladen ?
6	3	Wie hoch ist die Speicherauslastung ?

Tabelle 4.1: Goal-Question-Metrik - die konkreten Fragen

Metriken festgelegt.

4.3.3 Festlegung der Metriken

In dem letzten Schritt werden die Messverfahren festgelegt, anhand von denen quantitative Aussagen zu den gestellten Fragen hergeleitet werden können. Diese sind für den Vergleich zwischen den Technologien grundlegend.

Frage	**Anzuwendende Metrik**
1	Zeitmessung der Installation und Konfiguration von allen benötigten Komponenten bis zur Ausführung von einem ”Hello World!” Beispiel.
2	Die Summe der Zeitmessung für die Einarbeitung in die Technologie und die Entwicklung des Szenarios.
3	Die Zeit der Umsetzung, einer festgelegten Änderung.
4	NCLOC Metrik zur Ermittlung des physikalischen Umfangs.
4	Halsteads Metriken für die Länge und das Volumen der Umsetzung.
5	Messung des Zeitverhaltens.
6	Messung der durchschnittlichen Speicherauslastung.

Tabelle 4.2: Goal-Question-Metrik - die anzuwendenden Metriken

In der Abbildung 4.4 wurden alle Schritte dieser konkreten Umsetzung des Goal Question Metric Verfahrens zusammengefasst.

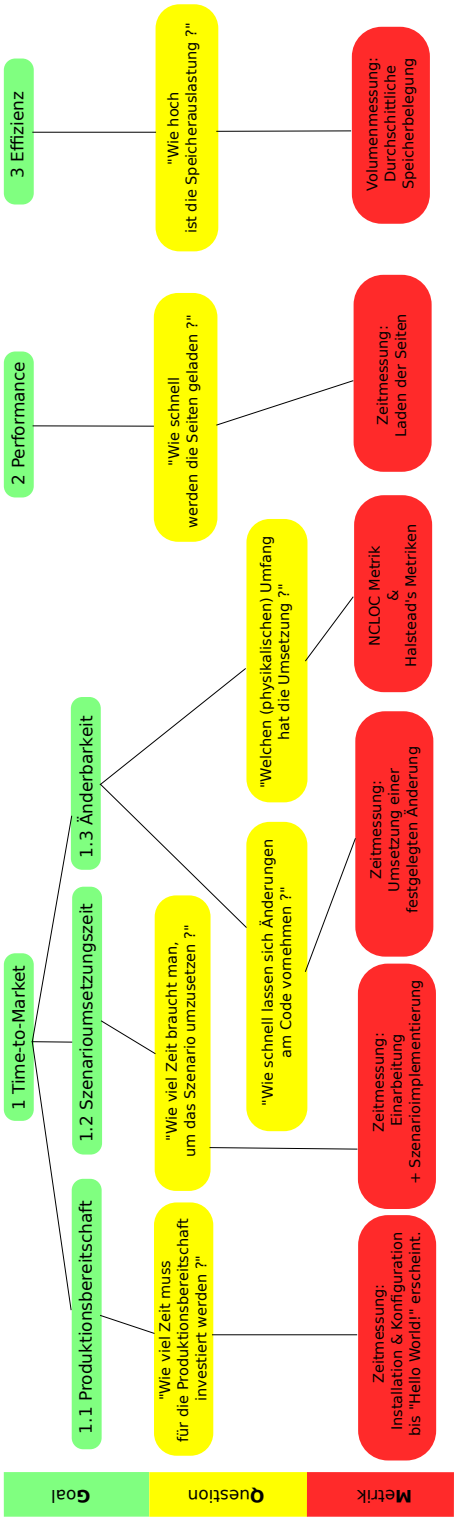


Abbildung 4.4: Konkrete Umsetzung der Goal Question Metrik

Kapitel 5

Design

5.1 Entwurfsziele

Im Abschnitt 4.3.1 wurden im Sinne der Durchführung des GQM-Ansatzes bereits einige Ziele festgelegt, die für den Entwurf eine wichtige Rolle spielen. Nun muss eine geeignete Architektur festgelegt werden, um vor allem das Ziel der einfachen Änderbarkeit zu erreichen. Das Erreichen dieses Ziels muss im ausgewogenen Verhältnis zu der Entwicklungszeit stehen. Denn wie bereits erwähnt, kann eine schnelle Markteinführung entscheidend für das Produkt sein. Das fertige System sollte also möglichst schnell mit geringem Aufwand implementiert werden und einfach zu ändern sein. Die Architektur muss also klar strukturiert werden. Ein passender Begriff hierfür ist *Separation of Concerns*, kurz SoC (dt. Trennug der Belange). Damit sollten Schichten und Komponenten so entwickelt werden, dass sie nur die ihnen bestimmten Aufgaben ausführen. Dieses Vorgehen steigert die Modularität der Software, was zu einem besser wartbaren Code führen sollte. Übertreibt man mit diesem Ansatz, so kann sich das negativ auf die Komplexität und dem entsprechend auf die Wartbarkeit des Systems auswirken. Ein Ziel, welches auf dem SoC aufbauen kann, ist die Austauschbarkeit. Die Module und vor allem die Schichten, sollten möglichst lose gekoppelt sein. Damit vereinfacht man eine spätere Auflösung der Abhängigkeiten. Letztendlich ist das ein Ziel, welches in der Software Entwicklung immer eine zentrale Rolle spielt, die Verfolgung des DRY Prinzips und damit die Erstellung von wiederverwendbarem Code.

5.2 Architekturstil (Schichtenarchitektur)

Ein Architekturstil legt Arten von Architekturelementen fest und gibt Architekturregeln an, die die Schnittstellen und Beziehungen der Architekturelemente einschränken [Lil08].

Wie bereits erwähnt folgt Apache Wicket dem MVC Ansatz. In Lift wiederum wird ein von dem MVC abgewandelter View-First Ansatz verfolgt. Beide Umsetzungen er-

lauben Freiheiten, die sich negativ auf die spätere Wartbarkeit auswirken können. So können in den Event-Callback-Methoden der Wicket-Komponenten beliebige Operationen ausgeführt werden. Ähnlich ist die Situation in den Snippet Methoden von Lift. Um die Zuständigkeiten besser zu trennen, bietet es sich an, die Schichten eindeutiger zu trennen. Mit dem Schichtenmodell gibt es einen Architekturstil, welcher zu diesem Zweck adaptiert werden kann. Der Aufgabenbereich des zu realisierenden Szenarios besteht aus der Speicherung, Verarbeitung und Präsentation der Daten. In der Client-Server Architektur, die unter anderem durch Webanwendungen repräsentiert wird, entspricht es in diesem Fall einer typischen Drei-Schichten-Architektur.

5.2.1 Persistenz

Diese Schicht ist für die Interaktion mit der Datenbank zuständig. Sie macht die Domänenobjekte persistent und holt sie auf Anfrage der höheren Schichten wieder heraus. Sie ist ebenfalls für Transaktionsmanagement zuständig und möglichst lose gekoppelt, um ihre Austauschbarkeit zu vereinfachen.

Wicket

Wicket verfügt über keine integrierte Persistenzlösung. Die Auswahl einer geeigneten Persistenzlösung muss des Entwickler selbst treffen. Für die Umsetzung des Szenarios, würde es reichen ein objektrelationales Mapping selbst zu implementieren. Eine Musterlösung, die mit Annotations und Reflections realisiert wurde und auf JDBC aufsetzt, wird von [Ess08] angeboten. Mit Java Persistence API steht aber ein mächtiges JEE Standard zur Verfügung, welches wiederverwendet werden kann. Dieses kann zusätzlich durch Domain Access Objects gekapselt werden. Ein Transaktionsmanagement muss ebenso entweder intern oder über externe Bibliotheken realisiert werden.

Lift

Mapper und Record sind die zwei Persistenzlösungen, die Lift in seinem Stack anbietet. Für die Realisierung des Szenarios wird eine relationale Datenbank verwendet, was für den Einsatz von Mapper spricht. Record ist momentan für die NoSQL Datenbanken ausgelegt. Ein Transaktionsmanagement wird von Lift ebenso geboten.

5.2.2 Anwendungslogik

Alle Verarbeitungsmachanismen, die die gesamte Geschäftslogik darstellen, werden in dieser Schicht gekapselt. Dazu können entsprechende Service- bzw. Managerklassen implementiert, die für die höhere Schicht die benötigte Logik mittels Interfaces zur Verfügung stellen. Sollen bestimmte Interna der Domainobjekte verborgen sein, können Transfer-Objekte verwendet werden, die durch eine Service-Facade transformiert

und zwischen den Schichten gereicht werden können. Letztendlich werden diese Funktionalitäten in den fachlichen Controllern des jeweiligen Frameworks benutzt.

5.2.3 Präsentation

Die Interaktion mit dem Benutzer wird mit Hilfe dieser Schicht realisiert. Hier kommen die unterschiedlichen View-Techniken zum Einsatz, auf die sich die ausgewählten Frameworks spezialisiert haben. Beide Frameworks zeichnen sich in diesem Aspekt von anderen Frameworks durch strikte Einhaltung von SoC aus. Auf innovative Weise verbieten sie den Einsatz von Anwendungslogik in dem Präsentationsmarkup.

5.3 Datenmodell

Für das Datenmodell des Szenarios werden folgende Entitäten vorgesehen:

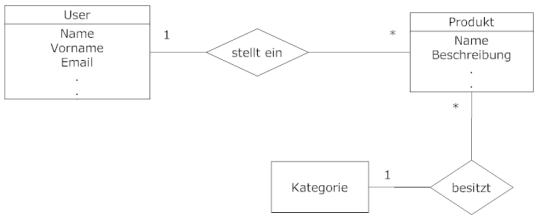


Abbildung 5.1: Entity Relationship Modell des Pet Store Szenarios

User Repräsentiert die Benutzer des Systems.

Produkt Repräsentiert die eingestellten Produkte.

Kategorie Repräsentiert eine Mengenbezeichnung, mit der die eingestellten Produkte kategorisiert werden können.

Zwischen den Entitäten ergeben sich Beziehungen, die durch das Entity-Relationship-Modell veranschaulicht werden (siehe Abbildung 5.1). Ein Benutzer, der sich im System registriert hat, kann mehrere Produkte ins Pet Store reinstellen. Ein Produkt wird einer Kategorie zugeordnet.

5.4 Systemkomponenten

Für das zu implementierende System ergeben sich aus dem im Abschnitt 3 vorgestellten Szenario folgenden Anwendungsbereiche:

Startseite Einstieg in die Anwendung.

Registrierung Bietet die Möglichkeit sich im System zu registrieren, um zusätzliche Systemfunktionen nutzen zu können.

Login Loginbereich für die bereits angemeldeten Benutzer.

Katalog Bietet Zugriff auf alle eingestellten Produkte.

Suche Bietet Möglichkeit nach Produkten zu suchen.

Produktdetails Bietet erweiterte Informationsansicht für ein ausgewähltes Produkt.

Produkteingabe Bietet die Möglichkeit neue Produkte einzustellen.

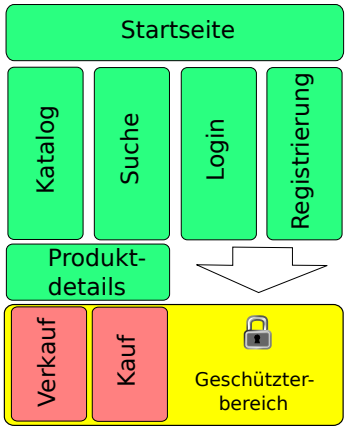


Abbildung 5.2: Übersicht der Anwendungsbereiche

Kapitel 6

Realisierung

6.1 Wicket Petstore

6.1.1 Persistenz

Bereits im Abschnitt 2.9 wurde erwähnt, dass es sich bei Apache Wicket lediglich um ein Präsentationsframework handelt. Für die Verwaltung der Persistenzschicht muss der Entwickler selbst sorgen. Für die Realisierung der Persistenzschicht in dem Pet-Store Szenario standen zwei Optionen zur Verfügung: entweder ein direkter Einsatz von der JDBC-Bibliothek oder ein Object Relational Mapper. Auf Grund der kleinen Größe der Anwedung würde man mit der ersten Option auf einfache Weise das Ziel erreichen. Die zweite Option erfordert zusätzliche Konfigurationen und Einarbeitung, bietet aber dafür mehr Möglichkeiten und eignet sich besser für die späteren Erweiterungen, die im Fall von Software fast immer der Fall sind.

Die Entscheidung fiel auf die Java Persistence API 2.0 und deren Referenzimplementierung EclipseLink[50]. Die Integration dieser Bibliotheken erfordert einerseits eine sentsprechende Erweiterung der `pom.xml` um die benötigten Abbhängigkeiten (siehe Listing 6.1) und Repositories (siehe Listing 6.2) und andererseits eine Konfiguration der Persistenzmodule in Form einer `persistence.xml` Datei (siehe Listing 6.3).

Listing 6.1: Maven Abhängigkeiten für die Realisierung der Persistenzschicht

```
1  <!-- MYSQL Connector -->
2  <dependency>
3     <groupId>mysql</groupId>
4     <artifactId>mysql-connector-java</artifactId>
5     <version>5.1.17</version>
6  </dependency>
7
8  <!-- JPA 2.0 Bibliothek -->
9  <dependency>
10    <groupId>org.eclipse.persistence</groupId>
11    <artifactId>javax.persistence</artifactId>
```

[50] EclipseLink - `http://www.eclipse.org/eclipselink`

```
12    <version>2.0.0</version>
13 </dependency>
14
15 <!-- EclipseLink Bibliothek -->
16 <dependency>
17    <groupId>org.eclipse.persistence</groupId>
18    <artifactId>eclipselink</artifactId>
19    <version>2.2.1</version>
20    <scope>runtime</scope>
21 </dependency>
```

Listing 6.2: Repositories mit den benötigten Abbhängigkeiten

```
1  <!-- EclipseLink Repository -->
2  <repository>
3     <id>EclipseLink Repo</id>
4     <url>http://mirror.selfnet.de/eclipse/rt/eclipselink/maven.repo/</url>
5  </repository>
6
7  <!-- Repository containing JPA 2.0 jars-->
8  <repository>
9     <id>maven2-repository.dev.java.net</id>
10    <name>Java.net Repository for Maven</name>
11    <url>http://download.java.net/maven/2/</url>
12    <layout>default</layout>
13 </repository>
```

Listing 6.3: Grundlegende Konfiguration der Persistenzmodule

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <persistence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://java.sun.com/xml/ns/persistence \
4     http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
5     version="2.0" xmlns="http://java.sun.com/xml/ns/persistence">
6     <persistence-unit name="petstore" transaction-type="RESOURCE_LOCAL">
7        <class>de.wicket.petstore.model.Product</class>
8        <class>de.wicket.petstore.model.Category</class>
9        <class>de.wicket.petstore.model.User</class>
10       <properties>
11          <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
12          <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost/wicket_petstore" />
13          <property name="javax.persistence.jdbc.user" value="root" />
14          <property name="javax.persistence.jdbc.password" value="admin" />
15
16          <!-- EclipseLink should create the database schema automatically -->
17          <property name="eclipselink.ddl-generation" value="create-tables" />
18          <property name="eclipselink.ddl-generation.output-mode" value="database" />
19       </properties>
20    </persistence-unit>
21 </persistence>
```

Mit diesen Konfigurationen kann nun im Projekt auf den MySQL-Connector und auf die JPA 2.0 API zugegriffen werden. In der `persistence.xml` wurden auch die Domainobjekte innerhalb einer `persistence-unit` deklariert. Im Projekt werden die Domainobjekte als POJO-Klassen definiert und mit den JPA Annotationen erweitert. Die Listungs 6.4 und 6.5 zeigen die Realisierung der Produkt- und der Category-Klasse.

@Entity markiert eine POJO-Klasse als persistente Entität (auf der Klassenebene)

@Id zeichnet eine Membervariable als eindeutigen Identifizierer.

@GeneratedValue definiert die Strategie der Erzeugung der mit `@Id` markierten Membervariable. Die Erzeugungsstrategie wurde auf `GenerationType.AUTO` gesetzt und diese entspricht bei MySQL dem bekannten `AutoIncremet`.

@OneToMany markiert eine `1:n` Beziehung. Das Attribut `targetEntity` definiert die Verbundentität und das Attribut `mappedBy` definiert den Namen der Membervariable, die die Beziehungsreferenz in der Verbundentität hält.

@ManyToOne markiert die Membervariable, die zu der `@OneToMany` inverse `n:1` Beziehung.

@NamedQueries Container, welcher für die Organisation der vordefinierten Abfragen dient.

@NamedQuery Eine Art vordefinierter Abfrage, die einmalig beim Start der Anwendung erzeugt wird. Die einmalige Konkatenierung der Queryparameter sorgt für einen Performancegewinn beim öfteren Benutzen der Query, die dann nur mit Parametern gefüllt werden muss.

Listing 6.4: Product Domainentität

```
1  package de.wicket.petstore.model;
2
3  imports ...
4
5  @Entity
6  @NamedQueries({
7          @NamedQuery(name="Product.findAllProducts",query="SELECT p FROM Product p"),
8          @NamedQuery(name="Product.findProduct",query="SELECT p FROM Product p WHERE p.id = :id"),
9  })
10 public class Product implements Serializable{
11         @Transient
12         private static final long serialVersionUID = -5680271669626998233L;
13
14         @Id
15         @GeneratedValue(strategy = GenerationType.AUTO)
16         private Long id;
17
18         private Category category;
19
20         private String name;
21         private String description;
22         private String imageUrl;
23         private Double price;
24
25         @ManyToOne
26         public Category getCategory() {
27                 return category;
28         }
29
30         public void setCategory(Category category) {
31                 this.category = category;
```

```
32                if (!category.getProducts().contains(this)) {
33                      category.getProducts().add(this);
34                }
35        }
36
37        // getters and setters
38        // ...
39  }
```

Listing 6.5: Category Domainentität

```
1   package de.wicket.petstore.model;
2
3   imports ...
4
5   @Entity public class Category implements Serializable{
6
7   @Transient private static final long serialVersionUID = 6648559336137698504L;
8
9   @Id @GeneratedValue(strategy = GenerationType.AUTO) private Integer id;
10
11  @Basic private String title;
12
13  @OneToMany(targetEntity=Product.class, mappedBy="category") private
14  Collection<Product> products = new LinkedList<Product>();
15
16  // getters, setters // ... }
```

Mittels Annotationen werden also Metadaten an die Objekte angehängt, die zur Laufzeit mittels Java Reflection ausgewertet werden. Diese Informationen beeinflussen die endgültige Transformation der Domänenobjekte. Die Domänenobjekte verfügen selbst über keine Funktionen, die eine Interaktion mit der Datenbank verursachen. Diese Funktionalitäten liegen im Verantwortungsbereich der Komponente, die das `javax.persistence.EntityManager` Interface implementiert. Diese wird in diesem Fall von dem Persistence-Provider `EclipseLink` bereitgestellt. Die Instantiierung des `EntityManagers` übernimmt die Komponente, die das `javax.persistence.EntityManagerFactory` implementiert. Der `EntityManager` wird unter Angabe des Namens der *PersistenceUnit*, welche in der `persistence.xml` definiert worden ist, erstellt. Die Entitäten können dem `EntityManager` unter anderem zur Anlage, Modifikation und dem Löschen in dazu vordefinierten Methoden übergeben werden (`persist`, `merge`, `remove`). Für das Auslesen aus der Datenbank werden die bereits erwähnten `NamedQueries` verwendet. Die Funktionalitäten des `EntityManagers` werden in der Umsetzung des Szenarios in domainbezogene Services gekapselt. Deren Benutzung wird im Abschnitt 6.1.2 veranschaulicht.

6.1.2 Anwendungslogik

Die Anwendungslogik in der Wicket-Umsetzung wird in den Eventmethoden der Seiten gekapselt. Diese enthalten meistens ein `Form`-Element welches die grundlegenden Methoden `onSubmit` und `onError` enthält. Bei den Komponenten die AJAX-

Aktionen ausführen ist es die `onClick` Methode. Diese Methoden nutzen greifen auf die Facade zu, die die Businessschicht von der Darstellungsschicht trennt. Sie interagreirt unmittelbar mit den DAOs. Die DAOs nutzen den `EntityManager` und die an den Domainobjekten definierte `NamedQueries`, um alle mit der Persistenz verbundene Aufgaben zu realisieren. Für die von den Benutzern hochgeladenen Bilder wurde ebenso eine `ImageService` Klasse definiert. Diese Verwaltet den Zugriff auf die Bilddateien, die nicht in der Datenbank, sondern auf der Festplatte abgelegt werden. Des weiteren wurden sicherheitsrelevante Anwendungsbereiche mit Hilfe der `SimplePageAutorizationStrategy` realisiert. Um die Bereiche, die nur für die authorisierten Benutzer zugänglich sind zu schützen bietet Wicket eine Möglichkeit der Definition einer `AuthorizationStrategy`. Diese wird in der `WicketApplication` Klasse, in Form von Erweiterung der abstrakten Klasse `SimplePageAutorizationStrategy`, gesetzt. Um die Seiten der Anwendung zu schützen reicht eine Markierung mit einem beliebig definiertem Marker-Interface. Dieser muss dann im Konstruktor der `SimplePageAutorizationStrategy` übergeben werden. Um die Identifikation des eingeloggten Benutzers zu behalten wurde die `Session` Klasse entsprechend erweitert.

6.1.3 User-Interface

In Apache Wicket spielen die Seiten eine zentrale Rolle. Wie bereits in den Grundlagen erwähnt, realisieren sie die Schnittstelle zum Benutzer und übernehmen die Funktionen des Controllers. In der Abbildung 6.1 sieht man unter anderem alle Seiten mit ihren Beziehungen, die zur Umsetzung des Szenarios erstellt wurden. Die von der `WebPage` abgeleitete `AbstractFormPage` kapselt lediglich den grundlegenden Markup für die abgeleiteten Seiten. `LogoutPage` enthält wiederum keinen Markup, dafür aber eine Controller-Aktion, die das Ausloggen des Benutzers realisiert (sinngemäß nur für eingeloggte Benutzer sichtbar). Eine Ebene tiefer gibt es die `MenuPage`, die nur das Seitenmenü in den Standard-Markup einbettet. Alle anderen Seiten, die das Menü anzeigen, leiten von dieser ab. Bis auf die `BuyerPage` und `SellerPage` können die Seiten ohne Einschränkungen betreten werden. Die zwei geschützten Seiten sind mit dem `SecurePageInterface` markiert und unterliegen der `SimplePageAutorizationStrategy`. Diese erlaubt nur eingeloggten Benutzern den Zugriff und sorgt für die Erstellung einer Session, die Informationen über den autorisierten User enthält. Die Abbildung präsentiert auch die Struktur der Realisierung mit allen Komponenten. Sie zeigt auch, welche Services von welchen Seiten in Anspruch genommen werden.

6.1.4 AJAX

In der Wicket-Realisierung konnte AJAX zur Validierung der Eingaben während der Registrierung eingesetzt werden. Es wurde ein AJAX-Verhalten für alle Eingabefelder

definiert (sehe Listing 6.6). Damit konnte, im Fall eines `onkeyup`-Events, die Validierung der Eingabekomponente angetriggert werden.

Listing 6.6: Aufruf des AjaxFormValidatingBehaviors

```
1 AjaxFormValidatingBehavior.addToAllFormComponents(form, "onkeyup",Duration.ONE_SECOND);
```

6.1.5 Testen

Apache Wicket bietet eine praktische Möglichkeit die Seiten der Anwendung zu testen. Die Tests können ausserhalb von dem Servlet-Container ausgeführt werden. Dies erfolgt in der JUnit bzw. TestNG Manier. Dazu muss in dem Ordner `test` eine Klasse angelegt werden, in der die Testmethoden definiert werden. Ein entsprechendes Beispiel liefert das Listing 6.7. Es stellt einen Test der Login-Seite. Es wird zunächst geprüft, ob die zu testende Seite korrekt gerendert wird. Anschließend wird mit einem `FormTester` die Simulierung der Eingaben realisiert und das Formular abgeschickt. Zu guter Letzt wird geprüft, ob die Weiterleitung auf die Hauptseite funktioniert hat.

Listing 6.7: Category Domainentität

```
1  public class TestLoginPage
2  {
3          private WicketTester tester;
4
5          @Before
6          public void setUp()
7          {
8                  tester = new WicketTester(new WicketApplication());
9          }
10
11         @Test
12         public void loginRendersSuccessfully()
13         {
14             tester.startPage(LoginPage.class);
15             tester.assertRenderedPage(LoginPage.class);
16             FormTester ft = tester.newFormTester("form");
17             ft.setValue("EMail", "test@test.com");
18             ft.setValue("Password", "test");
19             ft.submit();
20             tester.assertRenderedPage(HomePage.class);
21         }
22 }
```

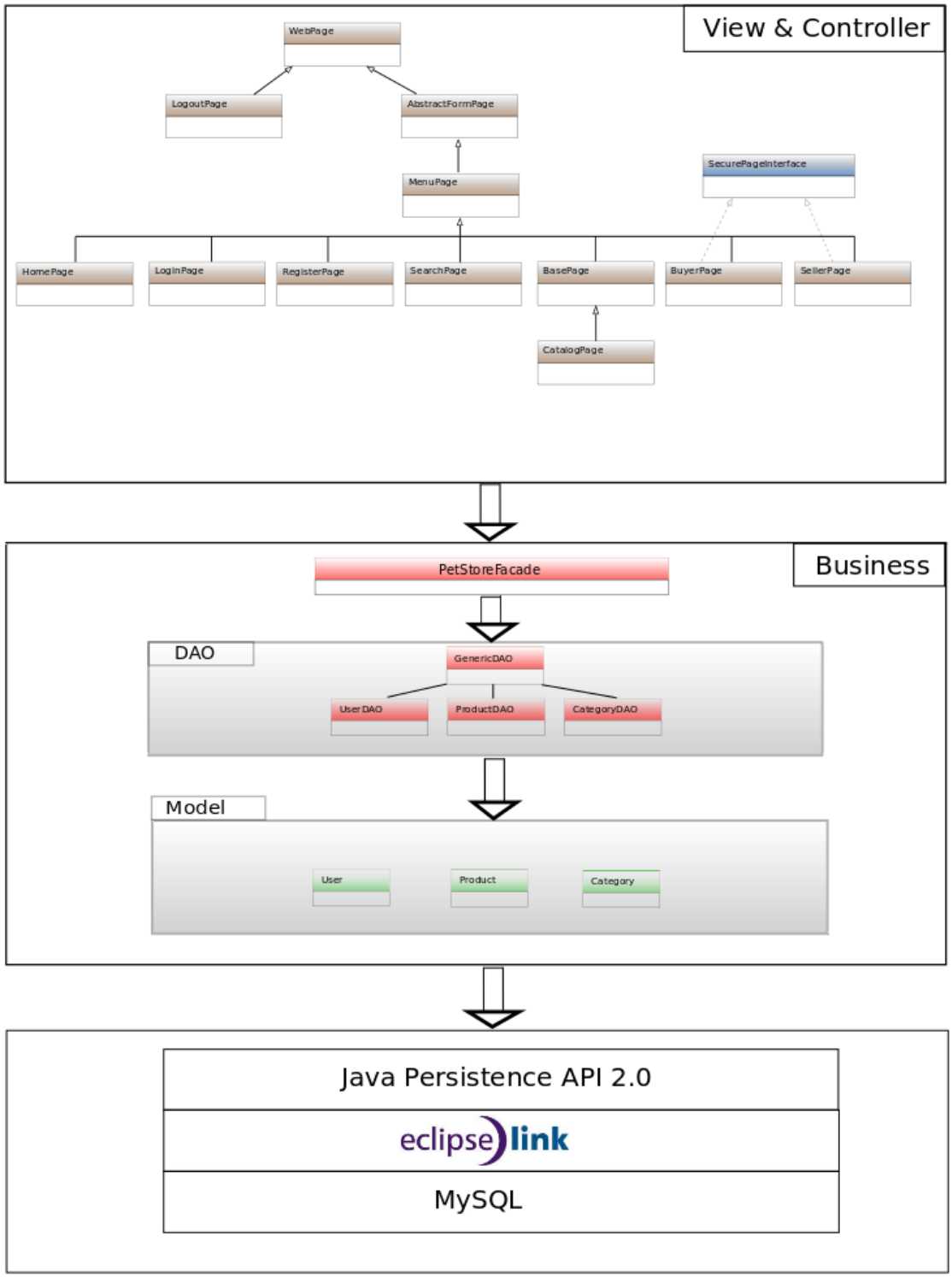


Abbildung 6.1: Struktur der Umsetzung des Wicket-Szenarios

6.2 Lift Petstore

6.2.1 Persistenz

In der Realisierung der Domainobjekte des Szenarios wurde die Mapper-Variante ausgewählt. Die Abhängigkeiten waren standardmäßig in der im Anhang C.2 vorgestellten Konfigurationsklasse eingetragen. So konnte man direkt mit der Definition von Domain-Objekten starten. Dies erfolgte durch Erweiterung des `LongKeyedMapper[T]` im Fall der Klasse und `LongKeyedMetaMapper[T]` im Fall des Companion-Objekts. Durch das Einmixen von dem `IdPK` Helper-Traits wird die `primaryKeyField` Methode, die vom `LongKeyedMapper` als abstrakt definiert ist, standardmäßig in Form von `MappedLongIndex(this)` implementiert. Alle Modelle werden auf Grund der Konvention in der Package `code.model` definiert. Für die Realisierung der Relationen zwischen den Objekten werden die Traits `LongMappedMapper`, `OneToOne`, `OneToMany` und `ManyToMany` verwendet. Die Listings 6.8 und 6.9 stellen die Realisierung der Domain-Objekte Produkt und Category dar, zwischen denen eine `n:1` Relation existiert.

Listing 6.8: Umsetzung des Domainobjekts Produkt

```
1  package code.model
2
3  imports ...
4
5  class Product extends LongKeyedMapper[Product] with IdPK with OneToMany[Long, Product]{
6    def getSingleton = Pet
7
8    object name extends MappedString(this, 64){
9      override def displayName = "Pet's name"
10   }
11
12   object description extends MappedTextarea(this, 100) {
13     override def textareaRows = 2
14     override def textareaCols = 50
15     override def displayName = "Description"
16   }
17
18   object price extends MappedDouble(this){
19     override def displayName = "Price"
20   }
21
22   object image_url extends MappedString(this, 64){
23      override def displayName = "Pet's picture"
24      override def asHtml = {
25       val url = Helpers.appendParams("/image/"+Full(get).openOr("default.jpg"),Nil)
26       Elem(null,"img",new UnprefixedAttribute("src",url,Null),xml.TopScope)
27     }
28   }
29
30   object category extends LongMappedMapper(this, Category)
31
32 }
33
34 object Pet extends Pet with LongKeyedMetaMapper[Pet] {
35        override def fieldOrder = List(name, description, price, image_url)
36 }
```

Listing 6.9: Umsetzung des Domainobjekts Category

```
1  package code.model
2
3  imports ...
4
5  class Category extends LongKeyedMapper[Category] with IdPK with OneToMany[Long, Category] {
6    def getSingleton = Category
7
8    object title extends MappedString(this, 64){
9      override def displayName = "Title"
10   }
11
12   object pets extends MappedOneToMany(Pet, Pet.category, OrderBy(Pet.name, Ascending))
13
14 }
15
16 object Category extends Category with LongKeyedMetaMapper[Category]
```

Um die Datenbankschemata für die Mapper-Objekte automatisch generieren zu lassen wurde in der Boot Klasse der `Schemifier` benutzt. Der Methode `schemify` wurden alle `MetaMapper` Objekte übergeben, für die das Schema generiert werden soll.

6.2.2 Anwendungslogik

Die Anwendungslogik in Lift-Anwendungen wird in den Snippet-Methoden gekapselt. Alle Snippet Klassen werden wie im Fall von den Modellen nach Konvention in dem Ordner `/code/snippet` definiert. Das Listing 6.10 zeigt das Prinzip, nach welchem die Snippets in der Lift-Realisierung des Szenarios erstellt wurden. Das Snippet Objekt enthält zunächst die Variablen, in denen die Eingabedaten für den Zeitraum der Abarbeitung des Requests festgehalten werden. Die Verarbeitung der Logik geschieht in der Methode, die im Markup aufgerufen wird. In der Methode kann eine Validierungsfunktion definiert werden, in der die Validierung der Eingaben stattfindet. In dem Fall des `ProductNew`-Snippets wurde kein Ziel für das Abschicken des Request definiert, so ist die Seite, von der die Snippet-Methode aufgerufen worden ist, das Ziel selbst. Das Statefull-Objekt `S` liefert die Informationen, ob das Formular abgeschickt worden ist. Bei dem ersten Aufruf werden die im Markup definierten Felder mit Defaulteingaben vorinitialisiert und mit Hilfe der `bind`-Methode als Markup an das Template zurückgegeben. Die `bind`-Methode ruft eine `chooseTemplate`-Methode um einen speziell markierten Teil des Markups zu verarbeiten. Was damit gemeint ist, wird im Abschnitt 6.2.3 erklärt. Wurde das Formular mit validen Daten abgeschickt, so wird in diesem Fall das Domainobjekt `Product` instanziiert, mit den eingegebenen Daten gefüllt und mit dem Aufruf der `save` Methode in der Datenbank persistent gemacht. Anschließend übernimmt die `bind` Methode die Generierung des Markups, welches für das Response benutzt wird.

Listing 6.10: Snippet-Klasse für das Hinzufügen von Produkten

```
1  package code.snippet
2
```

```
3  imports ...
4
5  object ProductSnippet {
6
7    // the request-local variable that hold the file parameter
8    private object theUpload extends RequestVar[Box[FileParamHolder]](Empty)
9    // the request-local variable that hold the name parameter
10   private object name extends RequestVar("")
11   // the request-local variable that hold the description parameter
12   private object description extends RequestVar("")
13   // the request-local variable that hold the price parameter
14   private object price extends RequestVar("0")
15
16   def add(xhtml: NodeSeq): NodeSeq = {
17     val errorBuffer = new ListBuffer[String]
18
19     def processEntryAdd() = {
20       errorBuffer.clear()
21       // check each field if not empty and add errorBuffer += "ErrorMessage to show"
22       errorBuffer.foreach(S.error(_))
23       !errorBuffer.isEmpty
24     }
25
26     if (S.get_? || processEntryAdd) {
27       bind("entry", chooseTemplate("choose", "get", xhtml),
28         "name" -> SHtml.ajaxText(name.is, input => {
29          if( !input.matches("^[A-Za-z]+$") )
30             DisplayMessage("messages",Text("Name can only contains letters"), 10 seconds, 1 second)
31          else
32             name(_)
33             DisplayMessage("messages",Text("Name ok"), 10 seconds, 1 second)
34         }),
35         "category" -> SHtml.select( allCategories.map(categorySelectionBox) ,
36                Empty, (x:String) => category(x), "class" -> "myselect"),
37         "description" -> SHtml.textarea(description.is, description(_)),
38         "price" -> SHtml.text(price.is, price(_)),
39         "file_upload" -> fileUpload(ul => theUpload(Full(ul)))
40       )
41
42     } else {
43
44       def getPetImageNode(imageName: Box[String]): NodeSeq = {
45         val url = appendParams("/image/" + imageName.openOr("default.jpg"), Nil)
46         <img src={ url } width="100"/>
47       }
48
49       theUpload.is.map(v => storeFile(v))
50       Pet.create.name(name.is).
51         description(description.is).
52         image_url(theUpload.is.map(v => Text(v.fileName)).open_!.toString()).
53         price(price.is.toDouble).save
54
55       def storeFile(fileParam: FileParamHolder) = {
56         val dir = new File("/tmp/", fileParam.fileName)
57         val output = new FileOutputStream(dir)
58         output.write(fileParam.file)
59         output.close()
60       }
61
62       bind("entry", chooseTemplate("choose", "post", xhtml),
63         "name" -> name.is,
64         "description" -> description.is,
65         "price" -> price.is,
```

```
66          "image" -> theUpload.is.map(v => getPetImageNode(Full(v.fileName))))
67      }
68    }
69  }
```

6.2.3 User-Interface

Im vorherigen Abschnitt wurde gezeigt, wie ein Modul zur Verarbeitung der Anwendungslogik in der Umsetzung des Szenarios mit Lift aufgebaut wurde. Die Snippet-Methode, die im Listing 6.10 enthalten ist, wird beim Aufruf der Seite ausgeführt, auf der ein Benutzer ein Produkt in den Pet-Store reinstellen will. Das Template ist im Listing 6.11 abgebildet. Mit Hilfe des `lift:surround`-Tags wird es beim Aufruf in ein Standard Template mit grundlegender HTML-Struktur eingebettet. Das Template ist in zwei Bereiche aufgeteilt: get und post. Diese werden in Bezug auf die Request-Phasen und Validität der Eingaben entsprechend zur Transformation durch die Snippet-Methode ausgewählt.

Listing 6.11: Seite für das Einstellen der Produkte

```
1  <lift:surround with="default" at="content">
2  <lift:ProductSnippet.add form="POST" multipart="true">
3         <choose:get>
4         <table>
5                 <tr>
6                         <td>Name:</td>
7                         <td><entry:name></entry:name></td>
8                 </tr>
9                 <tr>
10                        <td>Description:</td>
11                        <td><entry:description></entry:description></td>
12                </tr>
13                <tr>
14                        <td>Price (in EUR):</td>
15                        <td><entry:price></entry:price></td>
16                </tr>
17                <tr>
18                        <td>Picture (Select a file to upload):</td>
19                        <td><entry:file_upload></entry:file_upload></td>
20                </tr>
21                <tr>
22                        <td><input type="submit" value="Add"/></td>
23                        <td></td>
24                </tr>
25        </table>
26        </choose:get>
27
28        <choose:post>
29        <p>
30        Name: <entry:name></entry:name><br />
31        Desc: <entry:description></entry:description><br />
32        Price: <entry:price></entry:price><br />
33        <entry:image></entry:image><br >
34        </p>
35        </choose:post>
36 </lift:Simple.add>
37 </lift:surround>
```

6.2.4 AJAX

In der Lift-Realisierung konnte die Funktionsweise von AJAX anhand der Validierung eines Eingabefeldes getestet werden. Das `SHtml`-Objekt bietet unter anderem die Möglichkeit ein AJAX-Eingabefeld zu generieren. Das Listing 6.12 demonstriert eine mögliche Verwednung von AJAX zu validierungs Zwecken. In dem AJAX-Feld wird die Eingabe mit einem regulären Ausdruck getestet, ob der Inhalt nur aus Buchstaben besteht. In beiden Fällen wird eine Meldung in das Element mit der id `message` eingebettet.

Listing 6.12: Ausschnitt aus der Snippet-Klasse für die Produktdateneingabe

```
1   bind("entry", chooseTemplate("choose", "get", xhtml),
2     "name" -> SHtml.ajaxText(name.is, input => {
3       if( !input.matches("^[A-Za-z]+$") )
4         DisplayMessage("messages",Text("Name can only contains letters"), 10 seconds, 1 second)
5       else
6         name(_)
7         DisplayMessage("messages",Text("Name ok"), 10 seconds, 1 second)
8        }),
9     "category" -> SHtml.select( allCategories.map(categorySelectionBox) ,
10                    Empty, (x:String) => category(x), "class" -> "myselect"),
11    "description" -> SHtml.textarea(description.is, description(_)),
12    "price" -> SHtml.text(price.is, price(_)),
13    "file_upload" -> fileUpload(ul => theUpload(Full(ul)))
14    )
```

Statt `DisplayMessage` können auch selbst erzeugte Markupelemente an die definierte Stelle eingefügt oder andere Funktionen vom Typ `JsCmd` ausgeführt werden.

6.2.5 Testen

In Lift übernehmen die Snippets, als gewöhnliche Klassen oder Objekte, die Transformierung des Markups. Die Methoden, die dazu benutzt werden, können mit Testbibliotheken getestet werden. In Lift konnte beispielsweise die Generierung des Image-Elements mit der Testbibliothek `scala-specs` getestet werden (siehe Listing 6.13).

Listing 6.13: Ausschnitt aus der Snippet-Klasse für die Produktangabe

```
1   class SearchTestSpecsAsTest extends JUnit4(SearchTestSpecs)
2   object SearchTestSpecsRunner extends ConsoleRunner(SearchTestSpecs)
3
4   object SearchTestSpecs extends Specification {
5
6     "Search Snippet" should {
7       "get correct image element for the default image" in {
8         val search = new Search
9
10        val imageElem = search.getImageElem(Full("default.jpg"))
11        imageElem must be equalTo(<img src="/image/default.jpg" width="100"/>)
```

```
12      }
13    }
14  }
```

Mit dieser Bibliothek ist es möglich die Tests in beschreibender Form zu definieren. Damit können die Tests besser verstanden und nachvollzogen werden.

Kapitel 7

Bewertung

7.1 Systematik der Bewertung

Die Bewertung der Technologien und der mit ihnen realisierten Ausführungen des Szenarios erfolgt nach den im Abschnitt 4.2 aufgelisteten Kriterien. Je nach vorhandenem Messverfahren wird versucht für jedes Kriterium eine quantitative oder qualitative Aussage zum untersuchten Aspekt zu treffen. Dabei wird der Wert darauf gelegt, eine möglichst subjektive Bewertung durchzuführen. Man muss sich aber der Grenzen dieser Bewertung bewusst sein. Abgesehen von den ungleichen Erfahrungen mit Java und Scala aus dem Studium basieren die initialen Realisierungen ohne zuvor gesammelte Erfahrung mit den eingesetzten Web- und Persistenztechnologien. Mit den während der Realisierung gesammelten Erfahrungen würde eine wiederholte Realisierung für bestimmte Kriterien zu anderen Ergebnissen und weiteren Erkenntnissen führen.

7.2 Funktionalität

In diesem Abschnitt liegt die Konzentration auf der Bewertung der Funktionalität, die von den eingesetzten Technologien angeboten wird. Die einzelnen Merkmale, die unter dieses Kriterium fallen, wurden im Abschnitt 4.2.1 festgelegt. Die Auswertung zielt auf die Qualität, die bei der Realisierung festgestellt werden konnte. Als Darstellungsform wurde eine tabellarische Visualisierung der Ergebnisse ausgewählt. Die einzelnen Aspekte, die unter das Kriterium Funktionalität fallen, können besser gegenüber gestellt werden. Der Bewertungsschlüssel erstreckt sich von der vollen Zufriedenstellung bis zur vollen Unzufriedenheit. Die genaue Unterteilung präsentiert die Tabelle 7.1.

Bei der Realisierung des Szenarios mit den ausgewählten Technologien könnten, im Bezug auf deren Funktionalität, signifikante Erkenntnisse gewonnen werden (siehe Tabelle 7.2). So verfügen beide Web-Frameworks über ein gut durchdachtes Templatingsystem, welches mit herkömmlichen Webwerkzeugen kompatibel ist. Die Bindung der Daten an die Templates konnte einfach realisiert werden. Als komponentenba-

++	Volle Zufriedenstellung
+	Zufriedenstellung nur zum Teil
0	neutrale Bewertung
-	Teilweise unzufriedenstellend
--	Volle Unzufriedenheit
N.a.	Unbekannt, nicht zutreffend oder nicht vorhanden

Tabelle 7.1: Bewertungsschlüssel für die Funktionalität

siertes Framework ermöglicht Apache Wicket Erzeugung von Komponenten, die man exportieren und in anderen Wicket-Anwendungen einsetzen kann. Alle Artefakte einer Komponente (Sourcecode, Markup, CSS und JavaScript) sind an einer Stelle definiert und können leicht in ein Package gebündelt werden. Für Wicket stand auch eine größere Anzahl an Widgets und fertigen Komponenten zur Verfügung, die gut benutzt werden konnten. Bei Lift konnte der Einsatz von Scaffolding die Entwicklung signifikant beschleunigen, was bei Wicket nicht der Fall war. Beide Frameworks unterstützen HTML5. Wicket als komponentenbasiertes Framework unterstützt soweit nur wenige HTML5 Elemente.

Bei beiden Frameworks musste kein JavaScript geschrieben werden und die Erzeugung von Ajax-Elementen fiel nicht schwer. In der Realisierung konnte der volle Umfang der Interaktivitätsmöglichkeiten von Lift nicht eingesetzt werden. Dennoch unterscheidet es sich von Wicket durch Comet-Unterstützung und bessere Unterstützung von JSON.

Bei dem Einsatz von Lokalisierung konnte bei beiden Frameworks kein signifikanter Unterschied festgestellt werden. Beide Kontrahenten unterstützen die aktuellen Standards, die zur Lokalisierung und Internationalisierung der Webanwendungen benötigt werden.

Die Validierung ist in beiden Fällen gut. Bei Lift sind hauptsächlich die Mapper-Objekte für die Validierung der Daten zuständig und bei Wicket die Komponenten. In der Umsetzung des Szenarios konnten die benötigten Überprüfungen ohne großen Aufwand vorgenommen werden.

Der Aspekt der Persistenz liefert einige bedeutende Unterschiede. Wicket liefert hier keine eigene Lösung und erfordert selbst die Wahl einer Persistenztechnologie. Mit dem Einsatz von JPA konnten die Vorteile des JEE Standards genutzt werden, ohne den JEE-Container benutzen zu müssen. Integration und Benutzung erfolgten ohne Probleme, nahmen aber zusätzliche Zeit in Anspruch. Bei Lift konnte auf zwei integrierte Lösungen zugegriffen werden. Auf Grund der unzureichender Dokumentation und Beispiele für Record (welches sich für NoSQL DBMS eignet) fiel die Wahl auf den Mapper. Dieser ist bei der Benutzung einfach und integriert solche Konstrukte wie DAO und ermöglicht Scaffolding. Leider ist Mapper so konzipiert, dass die Domain-, Persistenz- und Darstellungslogik vermischt sind. Eine Ersetzung dieses Konstrukts

Kriterium	**Wicket**	**Lift**
View-Technik		
Templates	++	++
Datenbindung	++	++
Komponenten	++	N.a.
Widgets	++	+
Scaffolding	-	++
HTML5	+	++
AJAX		
Erzeugung Ajax Elemente	++	++
JavaScript erforderlich	nein	nein
Verwendung JSON	-	++
Comet Unterstützung	0	++
Internationalisierung		
Automatische Locale Ermittlung	++	++
Manuelles Setzen des Locales	++	++
Ressource-Bundles (Text)	++	++
Ressource-Bundles (XML)	++	++
Lokalisierte Templates	++	++
Validierung		
Pflichtfelder	++	++
Standard Validatoren vorhanden	++	++
Definition eigener Validatoren	++	++
Ajax-Validierung	++	++
Lokalisierte Validierung	++	++
Persistenz		
Persistenzlösung eingebaut	N.a.	++
Eigene Persistenz unterstützt NoSQL	N.a.	++
Austauschbarkeit der eigenen Persistenzlösung	N.a.	-
JPA Unterstützung	++	+

Tabelle 7.2: Bewertung der Funktionalität der eingesetzten Web-Frameworks

würde sich schwer gestalten, da es z.B. auf die Funktionalitäten im Mapper aufbaut. Diese müssten im Fall der Ersetzung des Mappers selbst nachimplementiert werden.

7.3 Benutzbarkeit

7.3.1 Dokumentation und Entwicklungswerkzeuge

Für die Einarbeitung in die Materie war der Bedarf hinsichtlich guter Dokumentation und guter Entwicklungswerkzeuge zur Unterstützung bei der Implementierung groß. Die Erkenntnisse, die unter diese Kriterien fallen, wurden qualitativ in tabellarischer Form bewertet (siehe Tabelle 7.3). Dabei wurde die Darstellung auf die eingesetzten Webframeworks und auf die Technologien, auf der diese basieren partitioniert. Die Bewertung basiert auf dem bereits bekannten Schema (siehe Tabelle 7.1)

Kriterium	**Java**	**Scala**
Dokumentation		
Bücher	++	++
Dokumentation	++	++
Community Größe	++	+
Community Aktivität	++	++
Entwicklungswerkzeuge		
IDE Support	++	+
3rd Party Bibliotheken	++	+
Kriterium	**Wicket**	**Lift**
Dokumentation		
Bücher	++	+
Dokumentation	++	++
Community Größe	+	++
Community Aktivität	+	++
Entwicklungswerkzeuge		
Build Management Tools	++	++
Build Management Tools IDE Integration	++	0
IDE Plugins	+	0
3rd Party Bibliotheken	+	+

Tabelle 7.3: Dokumentation und Entwicklungswerkzeuge der eingesetzten Technologien

Beginnend bei den Basistechnologien und deren Dokumentationen kann zwischen Java und Scala ein Unterschied in der Community-Größe festgestellt werden. Natür-

lich gibt es viel mehr Literatur zu Java, aber hier ist die Anzahl nicht entscheidend, sondern die Qualität. Für Scala gibt es mittlerweile sehr gute Bücher auf Deutsch und Englisch. Die rasche Entwicklung von Scala sorgt auch dafür, dass die Bücher zum Erscheinungszeitpunkt nicht auf der aktuellen Version aufbauen. Dies wir aber durch die Onlinemedien kompensiert, in denen die Neuerungen vorgestellt und dokumentiert werden.

Wenn es um die Entwicklungswerkzeuge geht, ist Java ganz klar vorn. Java wird in den IDEs nativ unterstützt, wobei die Unterstützung für Scala nur mit mehr oder weniger guten Plugins realisiert wird. Die REPL von Scala wirkt hier positiv, dennoch gibt es seitens von Scala einen Nachholbedarf für dieses in der Welt der Programmiersprachen als sensibel betrachtete Kriterium. In der Anzahl und Vielfalt der verfügbaren Bibliotheken ist Java vorne. Scala kann damit zwar nicht punkten, aber bietet dafür die Integration für Java Bibliotheken an.

Auf der Seite der Webframeworks gibt es für Wicket die bessere Dokumentation und momentan auch mehr Bücher. Bei Lift muss man feststellen, dass die Literatur relativ veraltet ist, was auf die schnelle Evolution des Frameworks zurückzuführen ist. Bei der Community kann der Unterschied zugunsten von Lift festgestellt werden. Hier merkt man die Größe und die Aktivitäten deutlicher als bei Apache Wicket.

Bei den Entwicklungswerkzeugen ist die Situation ähnlich wie im Fall von Java und Scala. Für die beiden Frameworks gibt es mit Maven und SBT gute Build Management Tools. Den Unterschied stellt man in der IDE-Integration von diesen fest. Maven kann gut integriert werden, SBT konnte wiederum nur für die Erzeugung von Eclipse-Projekten genutzt werden. Für beide Frameworks existiert eine ähnliche Situation, wenn es um die Verfügbarkeit von externen Bibliotheken und Komponenten geht.

7.3.2 Entwicklungsaufwand

Dieses Kriterium wird an dieser Stelle eingereiht, da die Qualität und Vollständigkeit, der zuvor bewerteten Kriterien darauf einen bedeutenden Einfluss haben. Als Maß für den Entwicklungsaufwand wurde intuitiv die Einarbeitungs- und Entwicklungszeit ausgewählt. Damit legt man eine quantitative Größe fest, von der man weitere Vergleichsaussagen ableiten kann. Sinngemäß musste eine entsprechende Einheit bestimmt werden, um die Unterschiede markieren zu können. Auf Grund der Projektgröße wurden die Einheiten *Man-Hour* und *Man-Day* ausgewählt. Der Entwicklungsaufwand wurde in drei Aspekte aufgeteilt: Einarbeitung in die Framework-, Persistenz-Technologie und die programmiertechnische Umsetzung des Szenarios. Die Einarbeitung wurde in zwei Aspekte aufgeteilt, da es sich bei JPA nicht um einen Bestandteil von Apache Wicket handelt. Man muss auch wiederholen, dass es seitens des Autors dieser Arbeit für die eingesetzten Technologien keine zuvor gesammelten Erfahrungen gab. Die Tabelle 7.3.2 stellt die für diese Messung relevanten Werte dar. Durch viele Beispiele und zur Verfügung stehenden Bücher, konnte man schnell ins Apache Wicket einsteigen.

	Apache Wicket	Lift Web
	Man-Hours / Man-Days	Man-Hours / Man-Days
Einarbeitung Framework	24 / 3	40 / 5
Einarbeitung Persistence	20 / 2,5	8 / 1
Implementierung	56 / 7	32 / 4
Total:	80 / 10	72 / 9

Tabelle 7.4: Entwicklungsaufwand in Zeiteinheiten

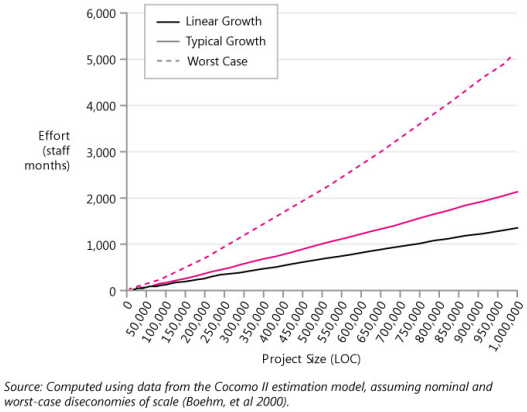
Problematisch war nur die Abwesenheit einer Persistenzkomponente, was die weitere Entwicklung vor die Wahl stellte: Selbst entwickeln oder eine bestehende Lösung integrieren? Mehrere Faktoren haben dazu beigetragen, dass JPA 2.0 und EclipseLink zur Realisierung der Persistenz benutzt wurden. Bei JPA handelt es sich um einen etablierten JEE-Standard und eine Benutzung ohne den JEE-Container möglich ist. Dies hat zwar zusätzliche Entwicklung eingespart, aber auch in zusätzlicher Einarbeitungszeit resultiert.

Im Fall von Lift musste mehr Zeit investiert werden, um in die anspruchsvolle Technologie einzusteigen. Des Weiteren gibt es für Lift weniger Beispiele und weniger Bücher. Es musste sogar festgestellt werden, dass in der Lift-Wiki Beispiele nicht funktionierten oder nicht mehr aktuell waren. Das ist unter anderem darauf zurückzuführen, dass die Entwicklung des Frameworks sehr aktiv vorangetrieben wird, was man an den relativ kurzen Release-Zyklen merkt. Diese werden auch von der Weiterentwicklung von Scala beeinflußt. Die Entwicklung ging dafür aber vergleichweise schnell. Der Aspekt, dass die Benutzerverwaltung bereits im Projekt implementiert war, hat dazu auch beigetragen. Dank dem Mapper konnten die Domainobjekte zügig realisiert werden.

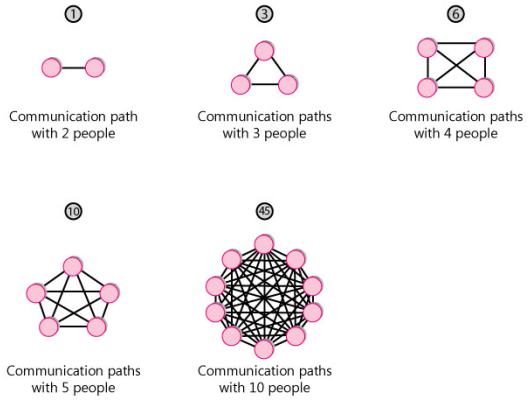
7.4 Änderbarkeit

7.4.1 Analisierbarkeit

Für die Ermittlung der Änderbarkeit wird unter anderem der Aspekt der Projektgröße herangezogen. Diese Entscheidung basiert auf den in [McC06] aufgezeigten Erkenntnissen. Dabei spielt der Projektumfang eine wichtige Rolle, denn mit der Projektgröße steigt seine Komplexität. Erfahrungsmessungen, die in dem Buch veranschaulicht wurden (siehe Abbildung 7.4.1), stellen klar, dass der Aufwand mit der Größe des Projekts nicht linear ansteigt, sondern exponentiell (Abbildung 7.1(a)). Als Begründung verwendet [McC06] unter anderem die Ursache, dass mit der Größe des Projekts der Bedarf an Kommunikation in den größer werdenden Koordinationsgruppen wächst. Damit wächst auch die Anzahl der Kommunikationspfade proportional zu der Anzahl der involvierten Kommunikationspartner (siehe Abbildung 7.1(b)).



(a) Korrelation des Entwicklungsaufwands zu Projektgröße aus [McC06]



(b) Wachstum der Kommunikationspfade aus [McC06]

Abbildung 7.1: Aspekte der Komplexität von Projekten aus [McC06]

Die Projektgröße wurde mit Hilfe der NCLOC und LOC Metriken bestimmt. Im Falle von Java konnten zahlreiche Tools gefunden werden, mit denen diese und diverse andere Metriken ausgeführt werden konnten. Für Scala wurde nur ein Tool gefunden, welches Scala-Code unter dem Aspekt NCLOC/LOC messen konnte: CLOC[51]. Die Ergebnisse der Messung stellt die Tabelle 7.4.1 dar.

Java Realisierung	**Scala Realisierung**
979	327

Tabelle 7.5: Anzahl der Lines of Code

Auf Grund des Einsatzes von Komponenten, DAOs, POJOs und einer Facade sieht man den Unterschied zwischen den Umsetzungen deutlich. Hier muss in Java viel Boilerplate-Code produziert werden. Einerseits spielt die Komplexität der Sprache eine entscheidende Rolle, aber andererseits ist auch die Größe des zu untersuchenden Software, für die Analisierbarkeit von der Bedeutung.

Die im Abschnitt 4.2.5 vorgestellte Halstead Metrik, konnte für Scala nicht angewandt werden. Der Begriff des Operators kann für Scala und Java nicht gleichgestellt werden. In Scala können Methodennamen auch die für die Operatoren vorgesehenen Symbole enthalten. Mit diesem fundamentalen Unterschied kann die Metrik von Halstead für Scala nicht eingesetzt werden. Dazu musste eine separate wissenschaftliche Analyse durchgeführt werden, um die Eignung dieser Metrik für Scala zu untersuchen.

[51] Count Lines of Code - `http://cloc.sourceforge.net/`

7.4.2 Testbarkeit

Für das Testen der realisierten Web-Anwendungen standen unterschiedlche Testansätze zur Verfügung. Das Testen von Wicket-Anwendungen kann mittels `WicketTester` realisiert werden. Damit konnte eine Seite ausserhalb von dem Servlet-Container, mit der Möglichkeit der Simulation der Eingabe von Daten, getestet werden. Dadurch kann viel Zeit gespart werden, da das ständige deployen der Anwendung entfällt.

In Lift konnten die Snippets dank einer Testbibliothek getestet werden. Diese erlaubte verständliche Ausformulierung der Testfälle. Dabei konnte das Testen, wie im Fall von Wicket, ausserhalb von dem Servlet-Container durchgeführt werden. Wenn man die beiden Testmöglichkeiten vergleicht, hat Wicket leicht die Nase vorn.

7.5 Effizienz

Um den Vergleich der Effizienz durchzuführen, wurde ein Tool namens JMeter[52] ausgewählt. Es ermöglicht definition von Tests, anhand von derer das Zeitverhalten der Webapplikationen gemessen werden kann. Dabei ist es möglich auch parallele Zugriffe zu simulieren und zu messen, wie sich die Anwendung unter steigender Last verhält. Für die Ausführung dieses Tests wurden gleiche Maßnahmen für beide Realisierungen vorgenommen. Während der Entwicklung des Szenarios wurden die Projekte in einem Test-Modus gebaut und ausgeführt. Für den Test wurde mit entsprechenden Einstellungen dafür gesorgt, dass die Anwendungen im produktiven Modus laufen. Dadurch konnten die in den Technologien eingesetzten Performancetechniken wie Caching von Templates aktiviert werden. Des Weiteren wurde der Level der Logausgaben entsprechend angepasst, um nicht jedes ankommende Request zu protokollieren.

Die Abbildung 7.2 zeigt den Versuchsaufbau samt der relevanten Informationen zu der eingesetzten Hardware und Software.

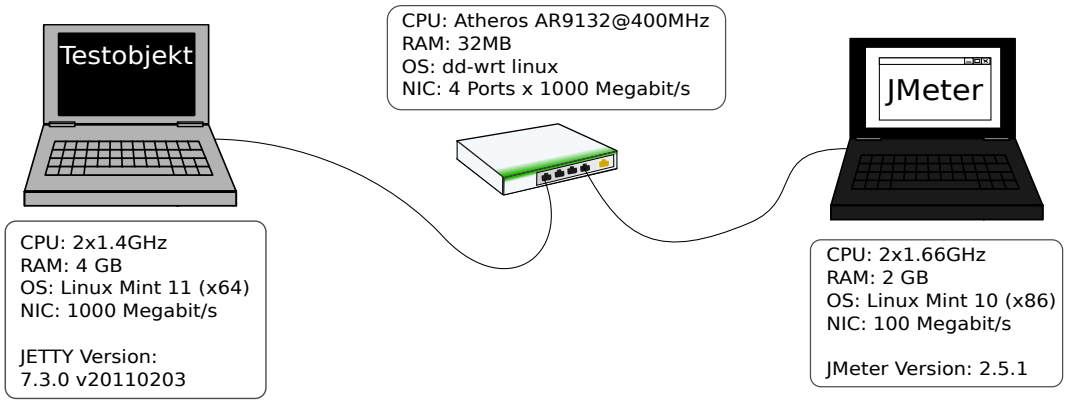


Abbildung 7.2: Aufbau der Benchmarking-Umgebung

[52] Apache JMeter - `http://jakarta.apache.org/jmeter/`

Die entwickelten Anwendungen sowie auch die Instanz von dem Testtool wurden auf separaten Rechnern ausgeführt. Beide Realisierungen wurden auf demselben System installiert und getestet. Das System wurde vor den Testreihen jeweils neu gestartet. Auch das testende System hat sich nicht verändert. Für die Testreihen der beiden Technologien wurde striktes Vorgehen eingehalten. Mit diesen Vorkehrungen, wurde dafür gesorgt, dass die Messergebnisse möglichst der Realität entsprechen und nicht verfälscht werden.

Als Grundlage für den Test wurde der Zugriff auf den Katalog ausgewählt. In diesem Abschnitt der Anwendung gibt es die Möglichkeit Anpassungen an der Anzahl der dargestellten Einträge zu verändern. Damit kann die Effizienz geprüft werden, wie schnell das Web-Framework die größer werdenden Antworten generiert. Die Tabelle 7.5 zeigt die protokollierten Ergebnisse der Testreihen.

Apache Wicket								
Einträge pro Seite	10		25		50		100	
Durchläufe	100	1000	100	1000	100	1000	100	1000
min (ms)	60	42	88	62	137	99	221	175
avg (ms)	116	61	164	86	230	133	339	209
90% Marke	199	79	306	113	407	173	565	249
max (ms)	578	562	521	799	803	1339	1441	2078
Lift Web								
Einträge pro Seite	10		25		50		100	
Durchläufe	100	1000	100	1000	100	1000	100	1000
min (ms)	50	32	53	34	57	36	64	43
avg (ms)	71	46	78	48	84	51	92	58
90% Marke	108	58	119	63	136	67	139	75
max (ms)	175	576	161	576	234	538	261	495

Tabelle 7.6: Einbenutzereffizienztest

Um das Verhalten der Web-Frameworks zu testen, wurde ein Mehrbenutzer mit der Erhöhung der Anzahl der Threads simuliert. Die Tabelle 7.5 zeigt die Ergebnisse für 2, 4, 8 und 16 Benutzer, die auf den Katalog (100 Einträge pro Seite) jeweils einhundert mal zugreifen.

Aus den Testergebnissen kann man klar die Dominanz des requestbasierten Web-Frameworks entnehmen. Lift kann die Antworten mit der steigenden Anzahl der zu generierenden Einträge effizienter und schneller als Apache Wicket liefern. Im Mehrbenutzertest liefert Lift nicht nur kürzere Antwortzeiten, sondern weist auch besseres Zeitverhalten mit der Erhöhung der parallelen Zugriffe. Beide Web-Frameworks produzierten während aller durchgeführten Testreihen keine Fehler.

Apache Wicket				
Benutzer	2	4	8	16
min (ms)	209	330	444	535
avg (ms)	420	699	1374	2507
90% Marke (ms)	733	1219	2232	3841
max (ms)	2228	4024	7955	15131
Fehlerquote %	0	0	0	0
Lift Web				
Benutzer	2	4	8	16
min (ms)	53	59	94	128
avg (ms)	116	189	359	745
90% Marke (ms)	165	304	518	1317
max (ms)	416	798	1853	2516
Fehlerquote %	0	0	0	0

Tabelle 7.7: Mehrbenutzerlasttest (100 Wiederholungen bei 100 Einträgen pro Seite)

7.6 Portierbarkeit

Im Fall von Apache Wicket lief die Installation reibungslos. Mit Hilfe des Packetmanagers wurden die Pakete von Maven 2 installiert und das `mvn`-Kommando stand dann sofort zur Verfügung. Als Nächstes wurde das im Anhang C.1 definierte Kommando ausgeführt und das Projekt angelegt. In der `pom.xml` wurden die benötigten Abhängigkeiten und Repository-Adressen eingetragen. Um die aktuellen Versionen zu erhalten war eine kurze Recherche nötig. Für die Entwicklungsbereitschaft von Apache Wicket selbst brauchte man keine Persistenzeinstellungen. Diese wurden aber einkalkuliert, auf Grund der Wahl der Persistenzstrategie. Als Nächstes wurden Eclipse Plugins `m2eclipse` und `qwickie` installiert, das Projekt ins Workspace importiert und eine *Run*-Konfiguration zum Kompilieren und Starten von dem integrierten Servlet-Container Jetty erstellt. Das ganze Prozedere hat ungefähr sechzig Minuten gedauert (siehe Tabelle 7.6). Das Vorgehen für die Installation von Lift Web Framework ist im Vergleich zu der von Apache Wicket einfacher. Lift Projektvorlagen funktionieren samt Persistenz *Out-of-the-box*. In dem Ordner musste lediglich ein Kommando ausgeführt werden und der Installationsautomatismus holte alle benötigten Komponenten und Abbhängigkeiten samt Scala ins Projekt. Das Ergebnis konnte sofort danach im Browser betrachtet werden. Auf mehr Aufwand stößt man, wenn man die in den Projektvorlagen mitgelieferte Version von Simple Build Tool 0.7.5 auf die neueste aktualisieren will (zu dem Zeitpunkt war dies die Version 0.10.1). Das Simple Build Tool wurde mit der Erscheinung der Version 0.10.x umgebaut und bietet nun mehr Performance und einfachere Konfiguration. Eine weitere Änderung endeckt man, wenn mann das Kommando `sbt jetty-run` ausführen will. Jetty Unterstützung wurde in ein Plugin ausgelagert und nicht standardmäßig in SBT integriert. Das Kommando wurde somit

Instalation von Apache Wicket		
Lfd.Nr.	**Aktion**	**Gebrauchte Zeit**
1	Installation Maven (über Packetmanager)	5 min
2	Anlage und Konfiguration des Projekts (pom, persistence)	30 min
3	Installation und Konfiguration Eclipse Plugins	25 min
	Total:	60 min
Instalation von Lift Webframework		
1	Runterladen, entpacken und SBT ausführen	5 min
2	Installation des Scala-Plugins für Eclipse	5 min
3	SBT Migration (Projekt und Jetty-PlugIn Konfiguration)	30 min
	Total:	40 min

Tabelle 7.8: Installationszeiten

nicht erkannt und man musste sich mit den Konfigurationsinterna auseinandersetzen. Das hat natürlich zusätzliche Zeit in Anspruch genommen.

Kapitel 8

Fazit und Ausblick

8.1 Fazit

Der im Rahmen dieser Arbeit durchgeführte Vergleich zwischen den aktuellen Java- und Scala-basierten Webtechnologien hat gezeigt, dass die neue Programmiersprache viel Potenzial für die Entwicklung von Webanwendungen mit sich bringt. Die funktionalen Konzepte sowie die native Unterstützung von XML stellen eine Grundlage für einen neuen, View-zentrierten Ansatz der Gestaltung von Webanwendungen dar. Die Umsetzung von grundlegenden Konstrukten kann mit Scala schnell realisiert werden und zeichnet sich außerdem durch eine vergleichbar gute Effizienz im Einsatz aus.

Die Schwierigkeit der Programmierung in Scala ist im Vergleich zu Java viel höher und die Lernkurve entsprechend flacher. Der Entwickler von Lift (die in dieser Arbeit eingesetzte Scala Webtechnologie) schildert in seinem Blog (siehe [Pol11]), warum Scala nicht einfach ist. In dieser Arbeit konnten die in dem Blog erwähnten Punkte gut nachvollzogen werden. Scalas Konstrukte sind anspruchsvoll und die Entwicklungswerkzeuge noch nicht hundertprozentig für den produktiven Einsatz geeignet.

Seitens Java sieht die Situation teilweise umgekehrt aus. Sie spiegelt die von [All11] beschriebene Situation. Man kann sich auf die Entwicklungswerkzeuge gut verlassen und die Sprache ist angenehm einfach. Man beschäftigt sich aber viel mehr mit der Integration von neuen Tools und Bibliotheken als mit der eigentlichen Programmierung.

Für Scala, als Technologie für das Web, sprechen innovative Konzepte und die Steigerung der Produktivität der Entwickler. Die Voraussetzung dafür ist aber eine gute Beherrschung dieser Sprache. Auch durchschnittliche Entwickler können mit Scala produktiver werden. Deren Fehlerwahrscheinlichkeit liegt aber weitaus höher, was zu hohen Kosten führen kann.

Zu guter Letzt ein Zitat eines Programmiersprachen-Pioniers:

> *A language that doesn't affect the way you think about programming, is not worth knowing.* [Per]
>
> Alan Perlis (einer der Erfinder von ALGOL[a])

[a] Programmiersprache ALGOL - `http://groups.engin.umd.umich.edu/CIS/course.des/cis400/algol/algol.html`

Zugegebenermaßen gehört Scala mit ihren innovativen Konzepten momentan zu den interessantesten Programmiersprachen. Sie ist zwar anspruchsvoll, aber es ist wert diese Sprache kennenzulernen und zu erlernen.

8.2 Ausblick

Mit Novell Vibe[53], foursquare[54], openstudy[55] gibt es bereits Webanwendungen, die von der AJAX-Unterstützung des Lift Webframeworks profitieren. In dieser Arbeit konnte dieser Aspekt nicht tiefgehend genug untersucht werden. Lift wird gerade für die Unterstützung von Comet gelobt. Es würde sich deshalb anbieten eine umfangreiche Untersuchung ausschließlich in diesem Bereich durchzuführen.

Des Weiteren wurde in den Realisierungen keine *Dependency Injection* verwendet. Dieser Aspekt spielt bei den Enterprise-Anwendungen eine Rolle und sollte näher untersucht werden.

Für die Zukunft von Lift ist es entscheidend wie stark Scala in der Programmierwelt an Bedeutung gewinnen wird. Scala ist anspruchsvoll und leidet unter einer weniger guten Unterstützung für Entwicklungswerkzeuge. Diese Tatsachen beeinflussen auch die Chancen von Lift in der Welt der Web-Frameworks. Hier kann noch viel in den nächsten Jahren passieren, da für Mitte 2013 der Release von Java 8 mit der Unterstützung von den lang erwünschten Closures kommt. Wie das die Weiterentwicklung der auf Java basierenden Frameworks beeinflussen wird, das wird die Zeit zeigen. Die Entwickler von Lift müssen aber zusehen, dass sie mit umfangreichen, aktuellen und Beispielen reichen Dokumentationen das Framework den Entwicklern von Webanwendungen schmackhaft machen.

[53] `http://www.novell.com/products/vibe-onprem/`
[54] `https://foursquare.com/`
[55] `http://openstudy.com/`

Kapitel 9

Zusammenfassung

Diese Arbeit vergleicht Java und Scala anhand einer exemplarischen Anwendung, die in den beiden Programmiersprachen realisiert wurde und die Basis für den Vergleich bildet. In den Grundlagen wird deutlich gemacht, dass Scala von dem Fortschritt der Technik in der Entwicklung der Programmiersprachen profitiert. Ähnlich wie damals in Java wurde in Scala auf neue Programmierkonzepte und Integration bewährter Techniken gesetzt. Die Fehler aus der Java-Welt hat man dabei vermieden. Diese Erkenntnisse führen zu der Aussage, dass Scala mit einer sehr ähnlichen Strategie wie Java erfolgreich werden will. Dabei hat Scala den Vorteil der JVM-Kompatibilität und Java-Interoperabilität.

Im Bereich der Web-Technologien wurde festgestellt, dass Java zum aktuellen Zeitpunkt mehr verschiedene Technologien zu bieten hat. Auf Java-Seite lag die Schwierigkeit aus der Menge zu wählen, wogegen bei Scala nur wenige Kandidaten zur Auswahl standen.

Anhand der festgelegten Kriterien wurden Web-Technologien, die jeweils auf Java und Scala basieren, untersucht und evaluiert. Diese Evaluation erfolgte mit dem Fokus auf die Entwicklung einer zeitgemäßen Webanwendung. Mit Apache Wicket und Lift Webframework wurden zwei Technologien ausgewählt, anhand derer der Einsatz von Java und Scala im Bereich der Web-Technologien untersucht werden sollte.

Zur Unterstützung der Untersuchung wurde ein bekanntes Testszenario ausgesucht, welches später mit den ausgewählten Technologien realisiert wurde. Die beiden Realisierungen wurden mit der Anlehnung an die Softwarequalitätsmerkmale der Norm ISO/IEC 9126 evaluiert. Dabei wurde festgestellt, dass die Realisierung mit der ausgewählten Java Web-Technologie mehr Zeit in Anspruch genommen hat und zu einem größeren Produkt geführt hat. Es ist darauf zurückzuführen, dass eine externe Persistenzkomponente benutzt wurde. In der Java Realisierung konnte dafür die strikte Trennung von Architektur-Schichten eingehalten werden. Die schnelle Realisierung mit der Scala Web-Technologie ist auf die Möglichkeit von Scaffolding zurückzuführen, womit viel Zeit gespart werden konnte. Die grundlegenden Konstrukte konnten schnell

realisiert werden. Schmerzlich musste man im Fall von Scala die noch nicht optimale Tool-Unterstützung feststellen. Hier hat Scala noch einen Nachholbedarf.

Bei der Performance-Untersuchung wurde festgestellt, dass die Scala-Realisierung die Antworten schneller generiert und effizienter auf die simulierten Mehrbenutzerzugriffe reagiert. Dies ist aber auf die Natur der ausgewählten Web-Frameworks zurückzuführen. In der Performance sind die requestbasierten Frameworks (u.a. Lift) den komponentenbasierten Frameworks (u.a. Wicket) überlegen.

Literaturverzeichnis

[All11] Eric Allman. Programming isn't fun any more. http://www.neophilic.com/b2evo/blogs/blog4.php/2011/09/02/programming-isnt-fun-any-more, September 2011. Zugriff: 2011.10.02.

[CBDW11] Derek Chen-Becker, Marius Danciu, and Tyler Weir. Exploring lift. http://exploring.liftweb.net/, July 2011. Zugriff: 2011.19.05.

[Cra06] Ian D. Craig. *Virtual Machines*. Springer Verlag London Limited, 2006.

[Eck07] Robert Eckstein. Java se application design with mvc. ORACLE Portal http://www.oracle.com/technetwork/articles/javase/mvc-136693.html, March 2007. Zugriff: 2011.04.02.

[EDBS05] Christof Ebert, Reiner Dumke, Manfred Bundschuh, and Andreas Schmietendorf. *Best Practices in Software Measurement - How to use metrics to improve project and process performance*. Springer-Verlag Berlin Heidelberg, 2005.

[Edw11] Jonathan Edwards. Switching to plan j. Jonathan Edwards' Blog http://alarmingdevelopment.org/?p=562, February 2011. Zugriff: 2011.04.15.

[Ess08] Prof. Dr. Friedrich Esser. *Java 6 Core Techniken*. Oldenbourg Wissenschaftsverlag, 2008.

[Ess11a] Friedrich Esser. Beyond oo. Scala Course papers, 2011. University of Applied Sciences Hamburg Germany.

[Ess11b] Prof. Dr. Friedrich Esser. *Scala für Umsteiger*. Oldenbourg Wissenschaftsverlag, 2011.

[FK11] Thomas Fiedler and Christoph Knabe. *Entwicklung von Web-Applikationen mit Lift und Scala*. Shaker Verlag Aachen, 2011.

[FMS09] Roland Förther, Carl-Eric Menzel, and Olaf Siefart. *Wicket: Komponentenbasierte Webanwendungen in Java*. dpunkt Verlag, 2009.

[Fow02] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman, Amsterdam, 2002.

[Fra07] Klaus Franz. *Handbuch zum Testen von Web-Applikationen - Testverfahren, Werkzeuge, Praxistipps*. Springer-Verlag Berlin Heidelberg, 2007.

[Get11] John Gethoefer. Lean agility: View-driven development (vdd). `http://l3an.com/post/654966108/view-driven-development-vdd`, July 2011. Zugriff: 2011.08.20.

[Gos97] James Gosling. The feel of java. *IEEE Computer*, 30:53–57, 1997.

[HMHG11] Cornelia Heinisch, Frank Müller-Hofmann, and Joachim Goll. *Java als erste Programmiersprache - Vom Einsteiger zum Profi*. Vieweg+Teubner Verlag, 6 edition, 2011.

[Hof08] Dirk W. Hoffmann. *Software Qualität*. Springer Verlag Berlin Heidelberg, 2008.

[Lil08] Carola Lilienthal. *Komplexität von Softwarearchitekturen*. PhD thesis, Universität Hamburg Fachbereich Informatik Arbeitsbereich Softwaretechnik, 2008.

[LN08] Tuukka Laakso and Joni Niemi, editors. *An Evaluation of AJAX-enabled Java-based Web Application Frameworks*. IEEE, 2008.

[Mal07] Stefan Malich. *Qualität von Softwaresystemen - Ein pattern-basiertes Wissensmodell zur Unterstützung des Entwurfs und der Bewertung von Softwarearchitekturen*. PhD thesis, Universität Duisburg-Essen, 2007.

[McC06] Steve McConnell. *Software Estimation: Demystifying the Black Art*. Microsoft Press, 2006.

[Mü06] Bernd Müller. *Java Server Faces. Ein Arbeitsbuch für die Praxis*. Carl Hanser Verlag München Wien, 2006.

[Nut09] Charles Nutter. Helping the jvm into the 21st century (the future: Part one). Charles Nutter's BLOG `http://blog.headius.com/2009/04/future-part-one.html`, April 2009. Zugriff: 2011.04.14.

[Ode08a] Martin Odersky. Scala's prehistory. `http://www.scala-lang.org/node/239`, August 2008. Zugriff: 2011.04.18.

[Ode08b] Martin Odersky. A tour of scala. `http://www.scala-lang.org/node/104`, August 2008. Zugriff: 2011.03.24.

[Ode10a] Martin Odersky. Scala is for good programmers. `http://www.artima.com/forums/flat.jsp?forum=270&thread=306728`, September 2010. Zugriff: 2011.05.17.

[Ode10b] Martin Odersky. Scala language specification: Version 2.8. `http://www.scala-lang.org/node/8590`, November 2010. Zugriff: 2011.03.24.

[Ode11] Martin Odersky. Scala language specification: Version 2.9. `http://www.scala-lang.org/docu/files/ScalaReference.pdf`, May 2011. Zugriff: 2011.03.24.

[OSV10] Martin Odersky, Lex Spoon, and Bill Venners. *Programming in Scala.* Artima - Mountain View, Calif., second edition, 2010.

[Per] Alan Perlis. Epigrams in programming. `http://www.cs.yale.edu/quotes.html`. Zugriff: 2011.10.20.

[Pie10] Lothar Piepmeyer. *Grundkurs funktionale Programmierung in Scala.* Carl Hanser Verlag München Wien, 2010.

[Pol11] David Pollak. Yes, virginia, scala is hard. `http://goodstuff.im/yes-virginia-scala-is-hard`, September 2011. Zugriff: 2011.10.19.

[SH06] Tony C Shan and Winnie W Hua. Taxonomy of java web application frameworks. IEEE International Conference on e-Business Engineering `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4031677`, 2006. DOI: 10.1109/ICEBE.2006.57, Zugriff: 2011.03.31.

[Sim01] Frank Simon. *Meßbasierte Qualitätssicherung.* PhD thesis, Brandenburgische Technische Universität Cottbus, 2001.

[SR11] Heiko Seeberger and Roman Roelofsen. *Durchstarten mit Scala.* entwickler.press, 2011.

[Str09] James Strachan. Scala as the long term replacement for java/javac? `http://macstrac.blogspot.com/2009/04/scala-as-long-term-replacement-for.html`, April 2009. Zugrif: 2011.04.14.

[Ull10] Christian Ullenboom. *Java ist auch eine Insel.* Galileo Computing, 9 edition, 2010.

[VSO09] Bill Venners, Frank Sommers, and Martin Odersky. The origins of scala. `http://www.artima.com/scalazine/articles/origins_of_scala.html`, September 2009. Zugriff: 2011.03.28.

[Wik11] Wikipedia. Programming paradigm. `http://en.wikipedia.org/wiki/Programming_paradigm`, 03 2011. Zugriff: 2011.04.23.

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 24. Oktober 2011 David Ratza

Anhang A

Evaluierung der Auswahl

A.1 Java Server Faces 2.0 - Zugriff: 2011-05-12

Verwendete Quellen:

- Offizieller Java EE 6 Tutorial[56]
- Aktuelles JSF 2.0 with Facelets and Ajax Tutorial[57]
- JSFUnit Project Page[58]

Tabelle A.1: JSF 2.0 - Untersuchungserkenntnisse

Kriterum	**Gewonnene Erkenntnisse**
AJAX Unterstützung	JSF ab Version 2.0 verfügt über eine Integrierte Ajax-Lösung.
Qualitative Dokumentation	Für den Erstanwender nicht optimal organisierte Dokumentation.
XML-freie Konfiguration	*Rather then hard-coding information into Java programs, many JSF values are represented in XML or property files.* Verbesserung gegenüber der Vorgängerversion duch Einsatz von Annotations. Dennoch eine `faces-config.xml` vorhanden.
Internationalisierung	Realisiert durch Java Platform Localization Classes.
Validierung	Vorhandene aber eingeschränkte Validierungsmöglichkeiten. Keine clientseitige Validierung möglich.
Community-Aktivität	Gegeben durch aktives Mojarra Project und dessen lebendige Mailinglist
Trennung View und Logik	A JavaServer Faces application can map HTTP requests to component-specific event handling and manage components as stateful objects on the server. Facelets statt JSP.
Zukunftssicherheit	Ein offizieler Bestandteil von Java EE 6. Kompatibel mit jedem standardisiertem Java EE 6 Server.
Testbarkeit	JSFUnit 2.0.0 mit Unterstützung von JUnit 4 und TextNG (noch in Beta)
Toolunterstützung	Unterstützung durch Netbeans und Eclipse 3.6+

[56] `http://www.oracle.com/technetwork/java/javaee/documentation/tutorials-137605.html`
[57] `http://www.coreservlets.com/JSF-Tutorial/jsf2/`
[58] `http://www.jboss.org/jsfunit/`

A.2 Spring MVC - Zugriff: 2011-05-12

`http://static.springsource.org/spring/docs/current/spring-framework-reference/html/`
Verwendete Quellen:

- Offizieler Spring Framework BLOG [59]
- Weiterer Spring Framework BLOG [60]

Tabelle A.2: Spring MVC - Untersuchungserkenntnisse

Kriterum	**Gewonnene Erkenntnisse**
AJAX Unterstützung	gegeben (Quelle 1)
Qualitative Dokumentation	Übersichtliche und verständliche Dokumentation mit Beispielen auf der Projektseite
XML-freie Konfiguration	Konfigurationen werden üblicherweise mit XML gemacht
Internationalisierung	Locales durch LocaleResolver und interceptors fürs Mapping in `org.springframework.web.servlet.i18n`
Validierung	Gegeben durch Spring's Validator Interface
Community-Aktivität	relativ aktives Team Blog
Trennung View und Logik	Gegeben mit vielen Möglichkeiten für Transformationen in andere Formate
Zukunftsicherheit	Durch die Etablierung gegeben, dennoch kann die lästige XML Konfiguration Entscheidung zu Günsten der neueren und einfacheren WAFs fallen.
Testbarkeit	Unterstützung von JUnit 4+, TestNG
Toolunterstützung	*SpringSource Tool Suite™ (STS) provides the best Eclipse-powered development environment for building Spring-powered enterprise applications.*

A.3 Stripes - Zugriff: 2011-05-12

Verwendete Quellen:

- Project Homepage [61]

Tabelle A.3: Stripes - Untersuchungserkenntnisse

Kriterum	**Gewonnene Erkenntnisse**
AJAX Unterstützung	Keine Integrierte AJAX Lösung. Realisierung durch externe AJAX Frameworks.
Qualitative Dokumentation	Gute und übersichtliche Dokumentation unterstützt duch viele verständliche Beispiele und User Additions
XML-freie Konfiguration	Konfiguration mittels Annotations
Internationalisierung & Lokalisierung	LocalePicker fürs Rausfinden des Locales für die Requests, Resource bundle(s) für Fehler und Feldnamen, und JSTL
Validierung	Diverse AnnotationDriven Validierungsmechanismen
Community-Aktivität	relativ aktive Mailingliste, IRC Channel & JIRA Bug Tracker
Trennung View und Logik	Models, Actions und Views mit JSP oder FreeMarker
Zukunftssicherheit	Eher keine positive Einschätzung, keine namenhaften Referenzen
Testbarkeit	JUnit und TestNG
Toolunterstützung	relativ alte Plugins für Netbeans und IntelliJ

[59] `http://blog.springsource.com/2010/01/25/ajax-simplifications-in-spring-3-0/`
[60] `http://viralpatel.net/blogs/2010/07/spring-3-mvc-internationalization-i18n-localization-tutor`
`html`
[61] `http://www.stripesframework.org/display/stripes/Home`

A.4 Struts 2 - Zugriff: 2011-05-12

Verwendete Quellen:

- Apache Struts 2 - Project Homepage [62]

Tabelle A.4: Apache Struts 2 - Untersuchungserkenntnisse

Kriterum	**Gewonnene Erkenntnisse**
AJAX Unterstützung	Integrierte AJAX-Unterstützung und Möglichkeit der Einbindung von externen AJAX Plugins. Result Actions müssen ggf. mit XML Kofiguriert werden.
Qualitative Dokumentation	Die Dokumentation ist relativ Übersichtlich und strukturiert. Man findet den Weg zu Tutorials, Guidelines und Beispielen.
XML-freie Konfiguration	Nur mit Einsatz eines Convention Plugins
Internationalisierung & Lokalisierung	I18n und Eisatz von Ressource Bundles gegeben. Locale kann in Requests mitgesendet werden und je nach Komponente umgesetzt werden.
Validierung	Zahlreiche Validierungsmöglichkeiten konfigurierbar je nach wahl mit XML oder Annotationes
Community-Aktivität	Relativ aktive Mailinglisten, kein Forum, unauffälige Projektlebendigkeit
Trennung View und Logik	Separate Model, Action und View Klassen und Alternative Templates Velocity und FreeMarker statt JSP
Zukunftssicherheit	Relativ positiv einzuschätzen, durch Zugehörigkeit zu Apache Foundation und mehrjähriger Existenz
Testbarkeit	Unterstützung von JUnit und TestNG
Toolunterstützung	Eclipse und Netbeans vorhanden

A.5 Tapestry - Zugriff: 2011-05-12

Verwendete Quellen:

- Project Homepage[63]

Tabelle A.5: Tapestry - Untersuchungserkenntnisse

Kriterum	**Gewonnene Erkenntnisse**
AJAX Unterstützung	Eingebaute Ajax Unterstützung basierend auf Prototype und Scriptaculous
Qualitative Dokumentation	Übersichtliche Doku mit Beispielen, und Videos
XML-freie Konfiguration	keine XML Konfigurationen, nur POJO's und Annotations
Internationalisierung	Einsatz von Property-Dateien, spezialisierter Templates, Erkennung von Locales und Möglichkeit ihrer Überschreibung mit Hilfe von PersistentLocale
Validierung	Validierung der Forms mit ValidationTracker, buildin Validators, Annotations
Community-Aktivität	Aktive Mailinglist, JIRA IssueTracker
Trennung View und Logik	Trennung mit Hilfe von Tapestry Templates (Tapestry Markup Language filed)
Zukunftssicherheit	Laut google Trends geht es seit paar Jahren langsam nach unten. Bei den älteren Versionen sollen an Rückwärtskompatibilität gemängelt haben. Ansonsten ein Apache Projekt
Testbarkeit	Integration Tests mit Selenium, TestNG und JUnit. 3rd Party Test Modules: Tapestry-Test und Tapestry-XPath
Toolunterstützung	Eclipse-plugin, IntelliJ out-of-the-box

[62] `http://struts.apache.org/2.2.3/index.html`
[63] `http://tapestry.apache.org/`

A.6 Wicket - Zugriff: 2011-05-13

Verwendete Quellen:

- Project Homepage[64]

Tabelle A.6: Apache Wicket - Untersuchungserkenntnisse

Kriterum	**Gewonnene Erkenntnisse**
AJAX Unterstützung	Eingebaute AJAX Lösung, zusätzlich kann Javascript manuel eingebaut werden und es gibt das wiQuery, ein jQuery Plugin
Qualitative Dokumentation	Gute Doku, mit Beispielen Code, Beispiel-Componente
XML-freie Konfiguration	Es gibt nur die web.xml ansonsten ALLES JAVA
Internationalisierung	Einsatz von Property-Dateien, einfache text Marker, Locale kann einfach mit getSession().setLocale(Locale.US) gesetzt werden
Validierung	eingebaute Validatoren, IValidator für eigene und FeedbackPanelfür Fehleranzeige
Community-Aktivität	Sehr aktive Mailingliste, IRC Channel, JIRA Issue Tracker
Trennung View und Logik	Templates sind HTML Dateien mit Tags die Wicked IDs enthalten. Templates können auch zu xhtml (Dateiendung) geändert werden.
Zukunftssicherheit	Bei Google Trends geht es langsam aber stetig nach oben. Ein Projekt der Apache Foundation.
Testbarkeit	Testing auf JUnit basierendem WicketTester
Toolunterstützung	Plugins für Eclipse, Netbeans und IntelliJ

A.7 Google Web Toolkit - Zugriff: 2011-05-13

Verwendete Quellen:

- Project Homepage[65]

Tabelle A.7: Google Web Toolkit - Untersuchungserkenntnisse

Kriterum	**Gewonnene Erkenntnisse**
AJAX Unterstützung	GWT setzt hauptsächlich auf AJAX
Qualitative Dokumentation	Sehr gute Doku mit vielen Beispielen
XML-freie Konfiguration	GWT Module werden mit XML definiert. Die XML Dateien können automatisch erzeugt werden. Um andere Module zu importieren bzw. zu Konfigurieren, muss die XML mit minimalen Aufwand angepasst werden.
Internationalisierung	Drei Techniken: statische mit Java-PropertyFiles, dynamische mit Benutzung von Module-Host-Pages (zu bestehenden Lokalisierungs-Systemen) und Einsatz von Localizable Interface.
Validierung	RequestFactory supports JSR 303 bean validation
Community-Aktivität	Google Group, GWT Issue Tracker
Trennung View und Logik	View wird automatisch aus Java Klassen generiert
Zukunftssicherheit	Durch zugehörigkeit und Unterstützung von Google gegeben
Testbarkeit	Relativ umständliches Testen des generierten JavaScripts. Unterstützung von JUnit
Toolunterstützung	Eclipse Plugin, Speed Tracer, GWT Designer und SDK Command-line Tools

[64] `http://wicket.apache.org/`
[65] `http://code.google.com/intl/de-DE/webtoolkit/`

A.8 Vaadin - Zugriff: 2011-05-13

Verwendete Quellen:

- Project Homepage[66]

Tabelle A.8: Vaadin - Untersuchungserkenntnisse

Kriterum	**Gewonnene Erkenntnisse**
AJAX Unterstützung	Auf GWT basierend
Qualitative Dokumentation	Gute Dokumentation und Tutorials, viele Beispiele, Videos und ein Buch
XML-freie Konfiguration	keine XML Konfiguration nötig
Internationalisierung	Locale kann gestetzt werden, was die internationalisierten Komponenten beeinflußt. Einfachere Umsetzung soll mit I18N4Vaadin Plugin möglich sein.
Validierung	Diverse eingebaute Validatoren und Validator interface für die eingene Prüfung der Formfelder
Community-Aktivität	Forum, Blog, Wiki, Chat, Twitter
Trennung View und Logik	Automatische Generierung der Komponenten aus Java Klassen
Zukunftssicherheit	Es basiert auf GWT und bietet zeitgemäßen Layout. Laut Google Trends geht es nach oben. Dennoch schwer einzuschätzen.
Testbarkeit	Mittels auf Vaadin abgestimmten Testing-Tool TestBench (Properitäre Lizenz für Komerzielle Zwecke erforderlich)
Toolunterstützung	Eclipse Add-on

A.9 Bowler - Zugriff: 2011-05-14

Verwendete Quellen:

- Project Homepage[67]

Tabelle A.9: Bowler - Untersuchungserkenntnisse

Kriterum	**Gewonnene Erkenntnisse**
AJAX Unterstützung	Lift-JSON für die Verwaltung von JSON Requests.
Qualitative Dokumentation	Minimale Dokumentation, mit einer Testcodezur einer Beispielapplikation
XML-freie Konfiguration	Doku listet keine Informationen über XML Konfiguration auf.
Internationalisierung	*simply copy your original template and suffix the filename before the file-ending with the locale.* Keine weitere Infos zur dynamischen Erzeugung von Button-Labels etc.
Validierung	Validatoren können dank dem Validations Trait realisiert werden. Es gibt auch das integrierte Recursivity Commons Modul, welches standard Validierungmachanismen anbietet.
Community-Aktivität	Kleine Google-Group, Author-Blog, Twitter und Code auf GitHub.
Trennung View und Logik	View wird in Ressource View und Layout zersetzt. Ressource View repräsentiert das Model und mit Layout wird die zugehörige Dekoration realisiert. Ausserdem benutzt Bowler Scala Template Engine - Scalate
Zukunftssicherheit	Das Projekt hat erst Version 0.3 erreicht und ist kaum bekannt.
Testbarkeit	Keine detailierten Angaben zu Thema Testing
Toolunterstützung	Simple Build Tool

[66] http://vaadin.com/home
[67] `http://bowlerframework.org/`

A.10 Circumflex - Zugriff: 2011-05-14

Verwendete Quellen:

- Project Homepage[68]

Tabelle A.10: Circumflex - Untersuchungserkenntnisse

Kriterum	**Gewonnene Erkenntnisse**
AJAX Unterstützung	Das HttpXMLRequests können laut Dokuërkannt werden. Ansonsten keine weitere Informationen zu Thema AJAX vorhanden.
Qualitative Dokumentation	Sehr minimalistisch gehalten
XML-freie Konfiguration	Konfiguration mittels Property-File: `cx.properties`
Internationalisierung	Locales können laut API Docs ermittelt werden.
Validierung	Keine Angaben
Community-Aktivität	Kleine Google-Group, GitHub - IssueTracker
Trennung View und Logik	Komponenten die ein Wrapper-Trait implementieren, sollen von den diversen View-Technologien unabhängig sein.
Zukunftssicherheit	Wenig bekannt und ohne nennenswerte Referenzen.
Testbarkeit	Dank eingenen Mock Objekten möglich.
Toolunterstützung	Maven und Simple Build Tool.

A.11 Lift Web Framework - Zugriff: 2011-05-14

Verwendete Quellen:

- Project Homepage[69]
- Project Wiki[70]

A.12 Pinky - Zugriff: 2011-05-14

Verwendete Quellen:

- Project Homepage[71]

A.13 Scalatra - Zugriff: 2011-05-14

Verwendete Quellen:

- Project Homepage[73]

[68] `http://circumflex.ru/`
[69] `http://liftweb.net/`
[70] `http://www.assembla.com/spaces/liftweb/wiki`
[71] `https://github.com/pk11/pinky/wiki`
[73]

Tabelle A.11: Lift Web Framework - Untersuchungserkenntnisse

Kriterum	**Gewonnene Erkenntnisse**
AJAX Unterstützung	Integrierte AJAX Unterstützung und Unterstützung von Comet Web Application Model.
Qualitative Dokumentation	Gute Doku, Wiki, Online Bücher und viele Beispiele.
XML-freie Konfiguration	Alles wird in Scala gemacht.
Internationalisierung	Ermittlung von Locale, Localisierte Templates, localisierte und formatierte Strings als auch die Ressource Bundles
Validierung	Etwas eingeschränkte Validierung durch selbstimplementierte Liste an Funktionen mit den ein Objekt validiert wird und ein Liste mit Errormeldunge zurückgibt.
Community-Aktivität	Große und aktive Google Gropup mit Mailinglist, Ticketing System und inoffizieles IRC Channel. Das Projekt wird kontinuierlich entwickelt.
Trennung View und Logik	XML Templates mit Lift-spezifischen Tags die keine Programmlogik enthalten können. View Klassen können auch XHTML und HTML5 konforme Dokumente generieren. MVCHelper als Option für die MVC-Enthusiasten.
Zukunftssicherheit	Momentan am meisten bekanntes Scala Web Framework. Gute Referenzen durch Twitter, LinkedIn und weitere bekannte Portale.
Testbarkeit	ScalaTest, Specs, mocking HTTP Requests mit MockWeb und Tests mit Selenium
Toolunterstützung	Maven, Simple Build Tool, Gradle

Tabelle A.12: Pinky - Untersuchungserkenntnisse

Kriterum	**Gewonnene Erkenntnisse**
AJAX Unterstützung	Laut Features-List: JSON (via xstream or json lib). Keine Beispiele
Qualitative Dokumentation	Sehr klein gehaltene Dokumentation mit ein Paar Beispielen.
XML-freie Konfiguration	Keine XML Konfiguration
Internationalisierung	Keine Informationen in der Doku
Validierung	Validierung mittels integriertem OVal[72]
Community-Aktivität	Sehr, sehr kleine Google Group
Trennung View und Logik	Laut Feature List FreeMarker und Velocity unterstützt.
Zukunftssicherheit	Das Projekt hat erst Version 1.5 erreicht und ist kaum bekannt.
Testbarkeit	Keine Angaben zu Thema Testing
Toolunterstützung	Simple Build Tool

Tabelle A.13: Scalatra - Untersuchungserkenntnisse

Kriterum	**Gewonnene Erkenntnisse**
AJAX Unterstützung	Laut der Infos auf der Project Homepage können AJAX Requests von HttpServletRequest ermittelt werden.
Qualitative Dokumentation	Lediglich ein Paar Infos und Beispiele auf der Project Homepage (GitHub)
XML-freie Konfiguration	Keine Informationen zur XML Konfiguration angegeben.
Internationalisierung	Keine Informationen vorhanden. Laut Google Group *We don't currently have any helper methods for doing this.*
Validierung	Keine Informationen Vorhanden.
Community-Aktivität	Eine 170+ Users Google Group und ein IRC Channel
Trennung View und Logik	Scalate als Template Engine.
Zukunftssicherheit	Für so ein kleines Projekt gibt hat Scalatra viele Enthusiasten. Ansonsten eher neutral einzuschätzen.
Testbarkeit	Integration von ScalaTest und Specs vorhanden.
Toolunterstützung	Simple Build Tool und Maven.

A.14 sweetscala - Zugriff: 2011-05-14

Verwendete Quellen:

- Project Homepage[74]

Tabelle A.14: sweetscala - Untersuchungserkenntnisse

Kriterum	**Gewonnene Erkenntnisse**
AJAX Unterstützung	Integration von jQuery vorgesehen ansonsten keine weitere Informationen vorhanden.
Qualitative Dokumentation	Eine sehr kleine Doku mit kleinen Beispielen.
XML-freie Konfiguration	Kein XML. Konfig Klassen. Einstellungen können sonst mittels `addProps("key"→"value")` oder `loadProps("/filepath/filename.config")` geladen werden.
Internationalisierung	Keine Informationen vorhanden.
Validierung	Simple Validator, Special Validator, Custom Validator und Möglichkeit von ihrer Verkettung.
Community-Aktivität	Sehr kleine Google Group. Letzte Aktivität ziemlich lange her.
Trennung View und Logik	Trennung durch SweetMVC
Zukunftssicherheit	Durch Projekt-Inaktivität gefährdet.
Testbarkeit	Keine Informationen vorhanden.
Toolunterstützung	Maven

[74] `http://code.google.com/p/sweetscala/`

Anhang B

Szenario Use-Cases

Use-Case	**UC01 - Registrierung**
Vorbedingung	Ein Benutzer (Gast) ist im System nicht registriert.
Nachbedingung	Der Benutzer ist ab nun dem System bekannt und erhält zusätzliche Privilegien. Er wird automatisch eingeloggt.
Akteure	Gast, der sich registrieren lassen will.
Beschreibung	Der Gast ruft die "Registrieren"-Aktion auf und bekommt das entsprechende Registrierungsformular dazu. Nach dem Validieren der Eingaben wird der Benutzer in das System eingetragen und kann Produkte bestellen.

Tabelle B.1: Use-Case UC01 - Registrierung

Use-Case	**UC02 - Login**
Vorbedingung	Ein Benutzer (Gast) ist im System registriert, aber nicht eingeloggt.
Nachbedingung	Der Benutzer hat sich mit der Email bzw. Benutzername ausgewiesen und ist im System eingeloggt.
Akteure	Benutzer, der sich einloggen will.
Beschreibung	Ein Benutzer besucht die Seite des Shops und will sich in das System einloggen. Der Benutzer ruft die "Login"-Aktion auf und bekommt das entsprechende Loginformular dazu. Nach dem Überprüfen der Email bzw. des Benutzernamens und des zugehörigen Passworts wird der Benutzer in das System eingeloggt.

Tabelle B.2: Use-Case UC02 - Einloggen in das System

Use-Case	**UC03 - Produktsuche anhand der Schlüsselwörter**
Vorbedingung	Ein Gast / Benutzer betritt die Seite.
Nachbedingung	Benutzer schaut sich die Produktdetails an.
Akteure	Ein registrierter Benutzer oder Gast.
Beschreibung	Ein Benutzer besucht die Seite des Shops und das ”Search”-Link. Nun bekommt er eine Maske, in der er Schlüßelwörter eintragen kann. Die Suche wird mit dem Anklicken des Buttons Search gestartet und der Benutzer bekommt einen Teil des Produktkatalogs zu sehen in dem Produkte, die das gesuchte Schlüsselwort enthalten, aufgelistet sind. Der Benutzer kann sich auch mittels Navigation die Produkte anschauen. Für Jedes Produkt können Produktdetails aufgerufen werden.

Tabelle B.3: Use-Case UC03 - Produktsuche (Schlüsselwörter)

Use-Case	**UC04 - Neues Produkts einstellen**
Vorbedingung	Ein Benutzer ist im System eingeloggt.
Nachbedingung	Neues Produkt ist in den Produktkatalog aufgenommen worden.
Akteure	Ein registrierter Benutzer.
Beschreibung	Benutzer navigiert in die Produktverwaltung und ruft die Aktion ”Neues Produkt einstellen” auf. Er bekommt ein Formular zu sehen, in das er Daten und ein Bild des neuen Produkts eintragen bzw. hochladen kann. Sind alle Daten eingetragen, so kann das neue Produkt in den Produktkatalog aufgenommen werden.

Tabelle B.4: Use-Case UC04 - Produkt einstellen

Use-Case	**UC05 - Ein Produkt bestellen**
Vorbedingung	Ein Benutzer ist im System eingeloggt.
Nachbedingung	Das Produkt wurde bestellt. Der Verkäufer bekommt eine Bestellungsbenachrichtigung.
Akteure	Ein registrierter Benutzer.
Beschreibung	Ein Benutzer ist im System eingeloggt und hat ein Produkt ausgewählt welches er kaufen will. Er ruft die Aktion ”jetzt kaufen” auf. Er bekommt eine Bestätigung, dass der Inhaber des Produkts per Mail benachrichtigt wird und sich wegen dem weiteren Ablauf der Transaktion melden wird.

Tabelle B.5: Use-Case UC05 - Produkt bestellen

Anhang C

Installation der Technologien

C.1 Wicket Setup

Apache Wicket unterstützt eine standardisierte Erstellung und Verwaltung der Projekte mittels dem Java Build-Management-Tool Maven. Die Anlage eines Projekts kann mit Hilfe eines für Maven generierten Kommandos erfolgen (Installation von Maven Version 2 vorausgesetzt). Für einen schnellen Einstieg mit Apache Wicket gibt es einen entsprechenden Maven-Archetyp: `wicket-archetype-quickstart`. Ein mit diesem Archetyp erstelltes Projekt ist bereits vorkonfiguriert und lauffähig. Listing C.1 enthält das entsprechende Maven Kommando, das zur Anlage der Projektstruktur für die Umsetzung des Szenarios verwendet wird. Das Kommando wurde auf der Apache Wicket Projektseite[75] mit den Parametern `de.wicket.petstore` für GroupID, `WicketPetstore` für ArtifactId und `1.5-RC5.1` für die zu verwendete Wicket Version generiert.

Listing C.1: Maven Kommando

```
1 mvn archetype:generate
2       -DarchetypeGroupId=org.apache.wicket
3       -archetypeArtifactId=wicket-archetype-quickstart
4       -DarchetypeVersion=1.5-RC5.1
5       -DgroupId=de.wicket.petstore
6       -DartifactId=WicketPetstore
7       -DarchetypeRepository=https://repository.apache.org/
8       -DinteractiveMode=false
```

Die Ausführung dieses Kommandos resultiert mit in Anlage des Projekts, mit der für Maven charakteristischen Ordnerstruktur (siehe Abbildung C.1). In dieser Abbildung kann man die Artefakte erkennen, die Maven automatisch generiert bzw. aus dem Repository importiert hat:

pom.xml *Project Object Model* stellt die im XML Format enthaltene Repräsentation des Maven-Projekts dar. Diese Datei enthält von der Benamung bis zu Definitionen der Abhängigkeiten alle Konfigurationen, die mit dem Projekt verbunden sind. Für den Quickstart sind standardmäßig die Abhängigkeiten für den Jetty Servlet-Container, das Event-Logging und Unit-Testing vorkonfiguriert.

web.xml Filter und Filter-Mapping Konfiguration für den Servlet-Container. *WicketFilter* ist hier als `filter-class` definiert und nimmt die `WicketApplication` als Parameter entgegen. Filter-

[75] Apache Wicket Quickstart - `http://wicket.apache.org/start/quickstart.html`

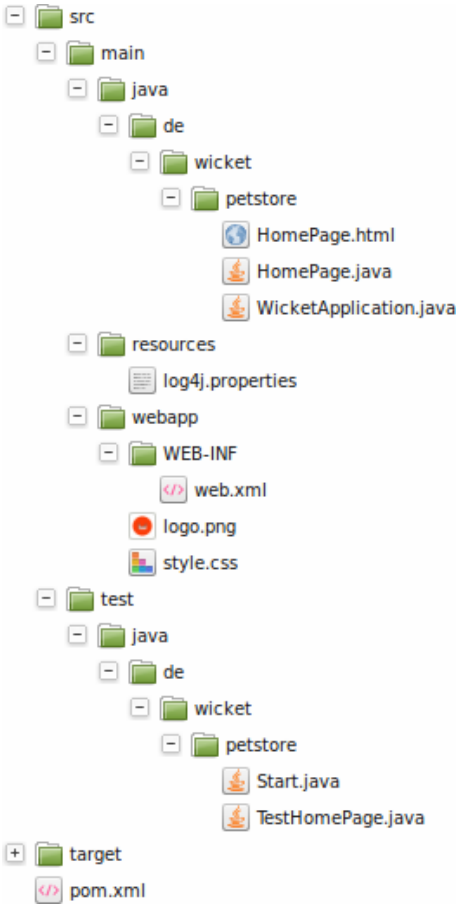


Abbildung C.1: Maven Projektstruktur der Wicket Quickstart Application

Mapping ist auf `/*` gesetzt. Damit wird die Anwendung standardmäßig geladen wenn auf den Server, ohne angabe weiterer Pfade, zugegriffen wird.

WicketApplication.java Die *Einstiegsklasse* der Wicket Application.

HomePage.java Die defaultmäßig generierte Klasse für die Startseite.

HomePage.html Markup der Startseite

Start.java Ausführbare Java Klasse, mit der die WicketApplication in einem embedded ServletContainer ohne explizites Deployment ausgeführt werden kann.

TestHomePage.java Die Testklasse für den Test der Startseite mit Hilfe des `WicketTesters`.

Mit dem Kommando `mvn jetty:run` wird die Anwendung gebaut und anschließend auf den automatisch gestarteten Jetty Servlet-Container exportiert. Zum Testen, ob alles geklappt hat, besucht man `http://localhost:8080` im beliebigen Web-Browser.

Zur Entwicklung des Szenarios wurde die aktuelle Version[76] der Eclipse Entwicklungsumgebung ausgewählt. Diese wurde mit dem Maven Plug-In `m2elipse`[77] und dem Wicket Plugin `qwickie`[78] erweitert.

C.2 Lift Setup

Das Aufsetzen eines Lift-Projekts erfordert keine zusätzlichen Installationen, solange eine Java Laufzeitumgebung installiert ist. Auf der Projektseite im Downloadbereich findet man für die aktuellsten Lift-Versionen entsprechende Archive, die *Ready-to-Run* Lift-Projekte enthalten. Die Version 2.4-M1 war die zu dem aktuellen Zeitpunkt neueste. Die in dem Archiv[79] enthaltenen Beispielprojekte sind zunächst nach der Scala Versionen 2.8 und 2.9 unterteilt. Für beide Versionen gibt es vier verschiedene Projektvorlagen:

lift_blank ist eine grundlegende Projektvorlage ohne vordefinierte Modelle und ohne CSS.

lift_basic ist eine auf HTML5 basierende Projektvorlage mit vordefiniertem User-Modell und Blueprint-CSS[80].

lift_xhtml wie die *basic* Variante, basierend auf XHTML.

lift_mvc bietet die Möglichkeit, statt nach dem View-First Pattern nach dem MVC-Prinzip zu entwickeln. Es beinhaltet keine vordefinierten Modelle, aber eine Blueprint-CSS.

Jede der Projektvorlagen enthält eine `sbt-launcher.jar`. Es handelt sich in diesem Fall um das Simple Build Tool in der Version 0.7.5. Zusätzlich gibt es jeweils für Unix und Windows ein entsprechendes Startskript. Wechselt man auf der Konsolenebene in diesen Ordner und führt das Kommando `sbt update ~jetty-run` aus, werden alle benötigten Komponenten samt Scala runtergeladen und das Projekt gebaut. Anschließend wird automatisch der integrierte Jetty Servlet-Container gestartet und die Anwendung darauf exportiert. Unter der Url `http://localhost:8080` sieht man das Ergebnis.

Die Struktur der Lift-Projekte orientiert sich an dem Maven-Standard, um die Kompatibilität zu diesem zu gewährleisten. Das wäre auch die andere Möglichkeit ein Lift-Projekt aufzusetzen. Allerdings konnte zu diesem Zeitpunkt in dem Scala-Repository[81] kein Archetyp für eine Projektvorlage gefunden werden, die auf der neuesten Scala Version basiert.

Um mit der Entwicklung des Szenarios, basierend auf der ausgewählten Vorlage, anfangen zu können, musste das Projekt konfiguriert werden. Dazu muss die `build.sbt` Datei, die sich im Hauptordner des Projekts befindet, modifiziert werden. Die *basic* Vorlage nutzt standardmäßig die H2[82] Datenbank. Diese musste gegen die MySQL ausgetauscht werden, um gleiche Voraussetzungen für den Vergleich zu gewährleisten. Dazu musste die entsprechende Abhängigkeit zu dem MySQL-Connector eingetragen werden.

Listing C.2: Lift-Konfigurationsklasse: LiftProject.scala

```
1 import sbt._
2
```

[76] Eclipse 3.7 - `http://www.eclipse.org/downloads/packages/eclipse-classic-37/indigor`
[77] m2eclipse - `http://www.eclipse.org/m2e/download/`
[78] qwickie - `http://code.google.com/p/qwickie/`
[79] `https://github.com/lift/lift_24_sbt/tarball/master`
[80] Blueprint-CSS - `http://www.blueprintcss.org/`
[81] http://scala-tools.org/repo-releases/net/liftweb/
[82] H2 Database Engine - `http://www.h2database.com/`

```
3  class LiftProject(info: ProjectInfo) extends DefaultWebProject(info) {
4    val liftVersion = property[Version]
5
6    lazy val JavaNet = "Java.net Maven2 Repository" at "http://download.java.net/maven/2/"
7
8    override def libraryDependencies = Set(
9      "net.liftweb" %% "lift-webkit" % liftVersion.value.toString % "compile",
10     "net.liftweb" %% "lift-mapper" % liftVersion.value.toString % "compile",
11     "org.mortbay.jetty" % "jetty" % "6.1.26" % "test",
12     "junit" % "junit" % "4.7" % "test",
13     "ch.qos.logback" % "logback-classic" % "0.9.26",
14     "org.scala-tools.testing" %% "specs" % "1.6.8" % "test",
15     //"com.h2database" % "h2" % "1.2.147"
16     "mysql" % "mysql-connector-java" % "5.1.17" % "compile->default"
17   ) ++ super.libraryDependencies
18 }
```

Um den Zugriff auf die lokal installierte MySQL Datenbank zu gewährleisten musste in der Boot-Klasse der `ConnectionManager` mit den auf die neue Datenbank abgestimmten Einstellungen versehen werden. Wegen der besseren Übersichtlichkeit wurde dieser in ein Scala-Objekt ausgelagert (siehe Listing C.2). Die Zugriffsdaten wurden in einer Properties-Datei hinterlegt.

Listing C.3: Modifizierter ConnectionManager in der Boot-Klasse

```
1  object DBVendor extends ConnectionManager with Logger {
2    def newConnection(name: ConnectionIdentifier): Box[Connection] = {
3      try {
4        Class.forName("com.mysql.jdbc.Driver")
5        val jdbcurl = (Props.get("db.url") openOr "jdbc:mysql://localhost/lift_petstore") +
6          "?user=" + (Props.get("db.user") openOr "") +
7          "&password=" + (Props.get("db.password") openOr "") +
8          "&" + Props.get("additionalurlparam").openOr("")
9        debug(jdbcurl)
10
11       val dm = DriverManager.getConnection(jdbcurl)
12       Full(dm)
13     } catch {
14       case e: Exception => e.printStackTrace; Empty
15     }
16   }
17   def releaseConnection(conn: Connection) { conn.close }
18 }
```