02-2015 | Sommer | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Aus der Community – für die Community

Java aktuell

D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

JAVA IST SUPER STARK

Programmierung
JavaScript für Java-Entwickler

Cloud Computing
Software-Architekturen in wolkigen Zeiten

Applikationsserver
JBoss vs. WebLogic Server

JavaServer Faces
Interview mit Spec Lead Ed Burns

4 191978 304903 02

iJUG Verbund

Wolfgang Taschner
Chefredakteur Java aktuell

Ihr Feedback an die Java aktuell

Seit der vorletzten Ausgabe stehen bei jedem Artikel ein QR-Code und ein Link, über den Sie mit der Redaktion beziehungsweise dem Autor Kontakt aufnehmen können. Für ganz schnelle Leser besteht die Möglichkeit, einfach per „like" oder „unlike" zu signalisieren, ob der Artikel die Erwartungen erfüllt hat oder nicht. Wer zudem noch eine Anregung oder Frage mitteilen will, füllt einfach das entsprechende Feld aus.

Zahlreiche Leser habe dieses neue Feature bereits genutzt. Spitzenreiter bei den „Gefällt mir"-Klicks waren die unbekannten Kostbarkeiten des SDK. In dieser Reihe hatte Bernd Müller von der JUG Ostfalen in der vorletzten Ausgabe die Klasse „Objects" näher beleuchtet. Dies zeigt wieder einmal, dass es oft die kleinen Dinge sind, die von großem Interesse sind.

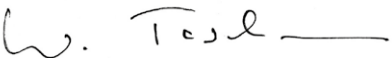
Sehr gut angekommen sind auch die Artikel „Apache DeviceMap", „Pimp my Jenkins", „Die Vorteile von Forms und Java vereint", „Scripting in Java 8" und „JDK 8 im Fokus der Entwickler". Die Bewertung ist für mich eine gute Hilfe bei der Auswahl künftiger Artikel.

Auf anderem Wege erreicht mich das Feedback des Lesers Dr. Klaus Gims. Er hat sein gesamtes Berufsleben mit Performance-Messung und -Tuning verbracht. Daher interessierte ihn der Artikel „Java Performance-Tuning" besonders. Er möchte gerne um das wichtige Thema „Performance und nichtfunktionale Anforderungen" eine Diskussion anstoßen. Seine Meinung: „Ich kann dem Artikel nur zustimmen. Allerdings fehlt ein wichtiger Aspekt. Um die Performance kümmert man sich in vielen Projekten frühestens zu Beginn, oft sogar erst am Ende der Funktionstests. Das hat sich seit der Großrechner-Ära kaum geändert. Das kommt mir vor, als prüfe man die Statik eines Gebäudes erst nach Fertigstellung des Rohbaus. First make it run – then make it fast ist noch immer die beliebteste Vorgehensweise. Was bei dem Entwickler einer Klasse, eines Packages oder einer Komponente noch gehen mag, ist für einen Software-Architekten ein schwerer Fehler. Damit wird ein produktiver Einsatz auf die Version 2.0 verschoben. Wohl dem, der diese Zeit noch hat. Im günstigsten Fall sprengt das jedes jedes Budget. Oft aber ist ein Mitbewerber schneller und die Nische für das Produkt ist schon besetzt."

Wie ist Ihre Meinung dazu? Über den nebenstehenden QR-Code und den Link können Sie direkt darauf antworten.

http://ja.ijug.eu/15/2/1

In diesem Sinne freue mich wieder auf viel Feedback.
Ihr



In Xtend geschriebener Code ist meist kompakter und eleganter als sein Java-Äquivalent

Der Autor ist ein sehr großer Fan von Java, aber auch von C#. Er stellt einige interessante Seiten von C# vor

Die Korrektheit erzeugter PDF-Dokumente überprüfen – nicht manuell, sondern automatisiert

Das Java-Tagebuch

Andreas Badelt, Leiter der DOAG SIG Java

Das Java-Tagebuch gibt einen Überblick über die wichtigsten Geschehnisse rund um Java – in komprimierter Form und chronologisch geordnet. Der vorliegende Teil widmet sich den Ereignissen im vierten Quartal 2014.

2. Oktober 2014

Intel tritt OpenJDK bei

Zum Abschluss der JavaOne hat Intel angekündigt, der OpenJDK-Community beizutreten und unter anderem Bibliotheks-Funktionen für Big Data und maschinelles Lernen beizusteuern.

http://jaxenter.de/news/intel-tritt-openjdk-bei-176615

8. Oktober 2014

MVC-Expertengruppe nimmt Form an

Die Expertengruppe für den neuen UI-Standard neben den Komponenten-orientierten JavaServer Faces nimmt bald Betrieb auf, die ersten Mitglieder sind bereits nominiert. Ganz so neu ist auch Action- orientiertes MVC nicht, um das es hier geht, aber die Standardisierung ist es. Es gab und gibt viele Bedenken, ob der JSR nicht viele Jahre zu spät kommt, in denen erst Struts und später Spring-MVC inmitten vieler anderer Frameworks zum De-facto-Standard wurden. Aber wie im letzten Tagebuch angemerkt, hat sich die Java-Community in der Umfrage zu Java EE 8 mehrheitlich die Aufnahme eines neuen MVC-Standards in die Enterprise Edition gewünscht. Die Expertengruppe scheint breit gestreut zu sein – einen offiziellen Spring-Repräsentanten sucht man vergeblich, dafür sind Vertreter von RedHat, IBM, LifeRay und mehrere deutsche Repräsentanten dabei.

http://bit.ly/1urfjnS

10. Oktober 2014

Ceylon 1.1 veröffentlicht

Die „andere Insel", das von RedHat initiierte Ceylon, ist soeben in der Version 1.1 erschienen – nach eigener Aussage das größte Release bislang mit mehr als 1.400 Fixes und vielen neuen Features, darunter die Verbesserung der Interoperabilität mit Java Generics und Overloading sowie mit dynamisch typisiertem JavaScript und einer Integration mit dem „vert.x"-Framework. Vollständige Abwärtskompatibilität ist nicht gegeben, aber das Sprachmodul wird nun als stabil angesehen. In Zukunft soll es keine Releases mehr geben, die die Lauffähigkeit existierender Programme durch API-Änderungen beeinträchtigen.

http://ceylon-lang.org/download

13. Oktober 2014

Virtual JBoss User Group

Für JBoss-Interessierte, die sie noch nicht kennen: Markus Eisele wirbt in seinem Blog für die Virtual jBoss User Group (VjBUG).

http://www.meetup.com/JBoss-User-Group-Worldwide

14. Oktober 2014

Silicon Valley Code Camp for Kids 2014

Eine Auflage der „Devoxx4Kids" in großem Stil und ein absolut vorbildliches Projekt, um in Kindern den Spaß an IT zu wecken, ist das „Silicon Valley Code Camp for Kids 2014".

Mehr als 300 Kinder durften an einem Wochenende in 24 einzelnen Sessions unter Anleitung mit Raspberyy PIs, Arduinos, LEGO Mindstorms etc. experimentieren und Programmier-Grundlagen lernen. Details und Bilder in Arun Guptas Blog *(siehe „http://blog.arungupta.me/silicon-valley-code-camp-kids-2014")*.

Wer so etwas lokal veranstalten möchte, kann auf der Startseite Infos holen.

http://www.devoxx4kids.org/join-us

15. Oktober 2014

Java SE 8 für WebLogic

Weblogic 12.1.3 ist jetzt für Java SE 8 zertifiziert (SE, nicht EE wohlgemerkt – Letzteres dauert ja leider noch ein bisschen). Damit sind nun einige hervorragende Features (Lambdas, Streams etc.), die mit EE 7 und EE 6 prinzipiell nutzbar sind, auch tatsächlich freigeschaltet.

https://blogs.oracle.com/WebLogicServer/entry/weblogic_server_12_1_3

17. Oktober 2014

Java EE 8: JMS 2.1, JAX-RS 2.1 und JSON-B starten

Nach MVC haben in den vergangenen Tagen mehrere weitere JSRs die Aufstellung ihrer Expertengruppe in Angriff genommen und um Bewerbungen gebeten: JMS 2.1, JAX-RS 2.1 und JSON-B. Falls sich jemand dafür interessiert: Nigel Deakin, designierter Spec Lead für JMS 2.1, hat einen Artikel über den Bewerbungsprozess für eine JSR Expert Group geschrieben. Natürlich besteht auch immer die Möglichkeit, sich über das „Adopt a JSR"-Programm an der Entstehung des JSRs zu beteiligen – das übrigens bei der nächsten JavaLand-Konferenz wieder zum Mitmachen einlädt, mit einer ganzen Reihe von Spec Leads und Expert Group-Mitgliedern vor Ort.

http://bit.ly/1lE2eKQ

21. Oktober 2014

MVC-Diskussion fortgesetzt

Und nochmal die Diskussion um MVC. Reza Rahman hat in einem interessanten Blog-Eintrag auf „java.net" erklärt, warum der neue Standard JSF nicht ablösen wird, und die Stärken und Schwächen beziehungsweise Ein-

satzgebiete der beiden zusammengefasst: Da Action-basierte MVC-Frameworks in der Regel weniger komplex und damit leichter zu erlernen sind, werden sie insbesondere dort angewendet, wo klassische Web-Entwickler auf ihre gewohnte Weise (typischerweise öffentlich zugängliche) Web-Anwendungen erstellen und die volle Kontrolle behalten wollen. Der Einsatz von JSF ist eher dort interessant, wo erfahrene Java-Entwickler mit wenig Aufwand, ohne viel „Boiler Plate Code", komplexe Web-Anwendungen entwickeln wollen, inklusive wiederverwendbarer UI-Komponenten.
https://blogs.oracle.com/theaquarium/entry/why_another_mvc_framework_in

21. Oktober 2014

Henrik Ståhl: Java SE 8 wird Default

Java SE 8 wird sehr schnell angenommen – nicht nur im Weblogic Server. Daher ist es nun auf „java.com" als Default-Download festgelegt worden, wie Henrik Ståhl von Oracle in seinem Blog verkündet. Das Auto-Update wird dann im ersten Quartal 2015 alle JREs automatisch auf die Version 8 aktualisieren, solange es nicht abgeschaltet wird. Ebenso wird dann auch bald wieder der kostenlose Support für die Vorgängerversion eingestellt. SE 7 soll nach April 2015 nur noch für zahlende Nutzer unter Support sein.
https://blogs.oracle.com/henrik/entry/moving_on_to_java_8

22. Oktober 2014

Arbeit der MVC-Expertengruppe jetzt öffentlich

Manfred Riem hat die öffentliche Seite für die MVC Spezifikation erstellt, auf der der jeweils aktuelle Stand der Arbeit der Expertengruppe zu sehen sein wird. Wer jetzt ein halbfertiges Spezifikations-Dokument erwartet, wird allerdings enttäuscht sein. Im Wesentlichen handelt es sich um das Mail-Archiv und vor allem Source Code – aber der ist ja auch viel spannender als Doku, oder?
https://java.net/projects/mvc-spec

23. Oktober 2014

Events in CDI 2.0: Neue Features

JSR 365 (CDI 2.0) ist schon seit einiger Zeit unterwegs und die Expert Group hat CDI-Events als einen der Schwerpunkte für das nächste Release ins Visier genommen. Unter anderem sollen folgende Features zum Event- Mechanismus hinzukommen: Asynchrone Events, „Conversation Begin/End"-Events, Prioritäten („Event Ordering"), Observer-Deaktivierung sowie die Möglichkeit, einen Observer für eine Gruppe von „qualified events" zu definieren. Der Blog-Eintrag zum JSR-Status verlinkt auch zu einem detaillierten Google-Doc, das die Überlegungen der Expert Group dokumentiert.
https://blogs.oracle.com/theaquarium/entry/events_in_cdi_2_0

11. November 2014

Wahlen zum Executive Committee

Die jährlichen Wahlen zum Executive Committee des Java Community Process haben stattgefunden – dem Gremium, das über die Standardisierung von Java wacht und über alle JSRs entscheidet. Es waren acht Sitze (alleiniges Vorschlagsrecht bei Oracle) und fünf frei wählbare Sitze zu vergeben. Bei ersteren wurden alle vorgeschlagenen Kandidaten akzeptiert: Freescale, Gemalto M2M GmbH, Goldman Sachs, SAP, Software AG, TOTVS und V2COM. MicroDoc ist als neues Mitglied dabei und der Name „Nokia" damit auch aus dem JCP verschwunden. Die frei wählbaren Sitze gingen an ARM Inc., Azul Systems, Hazelcast, Inc., Werner Keil und – auch neu dabei – Geir Magnusson Jr. Entsprechend ist die Marokko JUG wieder draußen. Sie war erst im letzten Jahr knapp ins Executive Committee gewählt worden und ist nun ebenso knapp gescheitert. Mit SouJava aus Brasilien und der London Java Community sind die JUGs aber weiterhin vertreten.
https://jcp.org/en/whatsnew/elections

14. November 2014

DZone: 2014 Guide to Enterprise Integration

Empfohlener Lesestoff für alle Entwickler und Architekten, die mit Enterprise-Integration zu tun haben: der frisch herausgegebene DZone „2014 Guide to Enterprise Integration". Ich sage das jetzt nicht, weil mit Markus Eisele und Daniel Bryant zwei der Autoren auch am JavaLand beteiligt sind, Letzterer für Adopt-OpenJDK in der „Early Adopters Area". Auch wenn ich natürlich immer wieder gerne die Werbetrommel für diese Veranstaltung rühre …
http://dzone.com/research/guide-to-enterprise-integration

15. November 2014

Cloud-Betriebssystem OSv im Betatest

Das minimalistische und quelloffene Cloud-Betriebssystem OSv startet den Betatest. OSv ist für den Betrieb einer einzigen Applikation (etwa einer JVM-basierten Anwendung) auf einem Hypervisor vorgesehen, und wirbt dafür mit deutlichen Vorteilen bei Performance und Server-Management.
http://osv.io

17. November 2014

JavaLand 2015: Das Programm ist online

Nach dem überwältigenden Erfolg mit der Erstauflage wird das JavaLand im März 2015 wieder für zwei Tage geöffnet. Das (absolut objektiv würde ich sagen: hervorragende) Programm ist jetzt online gegangen und Registrierungen sind bereits möglich.
http://www.javaland.eu

20. November 2014

Oracle kündigt weitere Features für Java 9 an

Auf der OpenJDK-Website sind weitere neue Features für Java 9 angekündigt worden, darunter „Unified JVM Logging", verbesserte „Compiler Control" für die JIT-Kompilierung und „Remove GC Combinations Deprecated in JDK 8" – also sozusagen die Meta-Garbage Collection für den Garbage-Collection-Code. Die aktuelle Gesamtliste ist verfügbar.
http://openjdk.java.net/projects/jdk9

27. November 2014

JSR 375: Java EE Security API

Ein neuer Java Specification Request (JSR 375) mit dem Titel „Java EE Security API" ist eingereicht worden. Ziel ist es, das gesamte Security-API von Java EE zu vereinfachen, zu standardisieren und zu modernisieren. Themen sind unter anderem User Management, Password Aliasing, Role Mapping, Authorization und Authentication.
https://blogs.oracle.com/theaquarium/entry/jsr_375_java_ee_security

Nachtrag: Der JSR ist vor Weihnachten einstimmig angenommen worden, die Expert Group allerdings noch etwas dünn besetzt.
https://jcp.org/en/home/index

3. Dezember 2014

Vier neue JSRs für Java EE 8 und SE 9

Drei weitere JSRs für Java EE 8 sind in den vergangenen Tagen auf den Weg gebracht worden: „Java EE Management API 2.0" (JSR 373) wird den Vorgänger aus JSR 77 um REST-Interfaces erweitern und Deployment-Unterstützung in den Standard einbauen. JSON-P wird einen kleineren Update auf die Version 1.1 erfahren (JSR 374). Nummer drei, die neue Java EE Security API, wurde bereits erwähnt. Darüber hinaus ist ein JSR für das „Java Platform Module System" eingereicht worden (JSR 376), der die beiden älteren Ansätze „Java Module System" (JSR 277) und „Improved Modularity Support in the Java Programming Language" (JSR 294, beide im JCP als „schlafend" gekennzeichnet), ersetzt.

Die Modularisierung ist für JAVA SE 9 gedacht, und soll dann auch für EE 9 genutzt werden. Der Anspruch des JSRs ist, ein Modul-System für Java zu standardisieren, und nicht die Java-Plattform selbst zu modularisieren. Letzteres soll dann im „Java SE 9 Platform Umbrella"-JSR definiert werden.

Die Arbeiten am Modul-System und die gleichzeitige Standardisierung sind eine Co-Produktion zwischen OpenJDK-Projekt und JCP, wobei Chief Platform Architect Mark Reinhold für beide Seiten verantwortlich ist. Für das OpenJDK gibt es dazu unter dem Projekt „Jigsaw" bereits die „JDK Enhancement Proposals" (JEP) 200, 201 und 220, die die Modulstruktur des JDK, eine Reorganisation des JDK-Source-Codes beziehungsweise die Restrukturierung der JDK- und JRE-Laufzeit-Images zum Thema haben. Ein weiterer JEP als Gegenstück zu JSR 376 ist in Planung.

http://mreinhold.org/blog

4. Dezember 2014

iJUG zur Java-Sicherheit

Der Einsatz von Java im Browser nimmt zwar immer mehr ab, aber in einigen Bereichen, insbesondere im Unternehmens-Umfeld, wird es immer noch intensiv genutzt. Der iJUG hat daher auf seiner Webseite auf Grundlage der jüngsten Java-Updates aktuelle Sicherheitsempfehlungen veröffentlicht, die auch für Laien verständlich sind. Zur Diskussion um Java-Browser-Plug-ins gab es Ende November auch einen interessanten Artikel auf Heise Online: „Browser-Plug-ins auf dem Rückzug". Dort werden die Konflikte beschrieben zwischen den Browser-Entwicklern, die sukzessive Plug-ins abschalten, aber auch nicht einheitlich vorgehen, und Projekten wie FireBreath, die auf Plug-ins setzen.

http://bit.ly/1Bds6tt

8. Dezember 2014

Schwere Sicherheitslücken in der Google App Engine

Nochmal das Thema Sicherheit: Externe Spezialisten haben eine Reihe von Sicherheitslücken in der Google App Engine gefunden, die es ihnen sogar ermöglichen, aus der JVM-Sandbox auszubrechen. Ganz lapidar heißt es da unter anderem: „We achieved native code execution (ability to issue arbitrary library / system calls)." Google wird sicherlich mit Hochdruck daran arbeiten, diese Lücken zu schließen; zunächst haben sie aber den Zugriff für das Spezialisten-Team gesperrt. Diese vermuten aber, dass die Sperre routinemäßig aufgrund der Aktionen des Teams erfolgte und schnell wieder aufgehoben wird, da sich Google normalerweise beim Thema „Sicherheit" recht offen (kleiner Wortwitz …) und kooperativ zeigt.

http://seclists.org/fulldisclosure/2014/Dec/26

9. Dezember 2014

Voxxed, ein neues Java-Portal

Ein neues Portal für Java-Themen, insbesondere Konferenz-Inhalte, ist online. Der Name ist nicht ganz zufällig ein Anagramm von Devoxx, das Portal wurde auf der Eröffnungs-Keynote der Devoxx offiziell gestartet.

http://www.voxxed.com

15. Dezember 2014

Adopt-a-JSR for Java EE

Wer mal aus erster Hand erfahren möchte, was bei Adopt-a-JSR abläuft: Heather vanCura blogged regelmäßig über die virtuellen und nicht-virtuellen Treffen und veröffentlicht die Zugangsdaten für erstere *(beispielsweise unter „https://blogs.oracle.com/jcp/entry/adopt_a_jsr_for_java1")*. Im Dezember gab es virtuelle Meetings mit den inzwischen dreizehn weltweit beteiligten JUGs und Java EE 8 Specification Lead Linda DeMichiel. Übrigens gibt es auch eine dedizierte und sehr informative Adopt-Seite für die Referenz-Implementierung GlassFish.

https://glassfish.java.net/adoptajsr

18. Dezember 2014

JavaScript-Frameworks und die JVM

Node.js-Applikationen auf der JVM ausführen und mit Mission Control die Performance analysieren? Nashorn macht es möglich. Erik Costlow hat in einem Blog-Eintrag einen Einblick gegeben, was mit der neuen JavaScript-Engine in der JVM und Projekt Avatar möglich ist.

https://blogs.oracle.com/java-platform-group/entry/node_js_and_io_js

6. Januar 2015

Abstimmung zum Controller for MVC

Model View Controller (MVC) wird auf JAX-RS aufbauen und seine Mechanismen nutzen – nicht auf dem Servlet-API. Das Thema wurde lange innerhalb der Expert Group diskutiert, eine Abstimmung, die die Entscheidung bringen sollte, endete unentschieden. Daraufhin haben die beiden Specification Leads gemeinsam entschieden. Wer sich für Details der Diskussion interessiert, wird in der Mailingliste fündig.

https://java.net/projects/mvc-spec/lists/users/archive/2014-10/message/0

Andreas Badelt
Leiter der DOAG SIG Java

Andreas Badelt ist Senior Technology Architect bei Infosys Limited. Daneben organisiert er seit 2001 ehrenamtlich die Special Interest Group (SIG) Development sowie die SIG Java der DOAG Deutsche ORACLE-Anwendergruppe e.V.

http://ja.ijug.eu/15/2/2

Software-Architekturen in wolkigen Zeiten

Agim Emruli, mimacom

Cloud Computing hat bereits das Tal der Illusion und Desillusion hinter sich gelassen und wird nun zunehmend in Unternehmen eingesetzt. Cloud-Systeme setzen Software-Architekturen voraus, die robust und ausfallsicher sind und vor allem mehr oder weniger Hardware-Ressourcen adaptieren können. Dieser Artikel geht auf die Herausforderungen von Cloud-orientierten Software-Architekturen im Java-Umfeld ein und zeigt mögliche Entwurfsprinzipien auf.

Eine der Hauptmotivationen für den Einsatz von Cloud Computing sind Kosteneinsparungen im Betrieb der Anwendungen. Diese lassen sich durch den Einsatz von einfacher, standardisierter und nicht ausfallsicherer Hardware erzielen. Diese Hardware wird allgemein auch als „Commodity class"-Hardware *(siehe „http://en.wikipedia.org/wiki/Commodity_computing")* bezeichnet. Sie bringt eine durchschnittliche Rechenleistung, steht aber aufgrund der sehr geringen Preise in einer hohen Anzahl zur Verfügung. Für Anwendungen, die auf Basis dieser Hardware betrieben werden, sind bereits beim Entwurf einige Vorkehrungen zu treffen, damit die Verfügbarkeit jederzeit sicher gestellt ist und diese Anwendungen auch auf neue Lastanforderungen dynamisch skalieren können. Einige Entwurfsprinzipien haben sich in den letzten Jahren bereits mehrfach in der Umsetzung von Cloud-orientieren Architekturen bewährt.

Zustandlosigkeit

Enterprise Anwendungen besitzen in der Regel einen persistenten Zustand der in einer relationalen oder NoSQL-Datenbank gehalten wird. Darüber hinaus haben viele Anwendungen einen transienten Zustand, der zumeist im Hauptspeicher der Server-Anwendung gehalten ist. Im Web-Umfeld werden dazu HTTP-Sessions eingesetzt, die unter anderem Anmelde-Informationen oder Informationen zur aktuellen Webseite des Benutzers innerhalb einer Web-Anwendung festhalten. Das Halten von flüchtigen Zustands-Informationen ist jedoch in einer Cloud-Anwendung problematisch, da einzelne Server im Rahmen eines Hardware-Ausfalls diesen Zustand verlieren oder aber neue Server im Rahmen einer Skalierung hinzukommen können.

Optimalerweise verzichtet man daher im Rahmen einer Cloud-orientieren Architektur komplett auf das Halten eines Zustands im Server-Hauptspeicher. Dies lässt sich durch den Einsatz von „Single Sign-on"-Verfahren mit einem zentralen Authentifzierungs-Service für die Anmelde-Informationen erreichen. Darüber hinaus bieten heutige Browser erweiterte Möglichkeiten, um Zustands-Informationen direkt im Browser abzulegen.

Falls sich die Verwendung von HTTP-Sessions nicht ganz vermeidet lässt, macht es Sinn, einen zentralen Cache zur Haltung der Sessions einzusetzen. Dieser Cache kann in vielen Cloud-Umgebungen als hochverfügbarer Service von der Anwendung genutzt werden. Amazon Web Services stellt beispielsweise einen hochverfügbaren Memcached- oder Redis-Cache-Service bereit *(siehe „http://aws.amazon.com/elasticache")*.

Andere Cloud-Plattformen haben ebenfalls ähnlich hochverfügbare Services im Angebot. *Abbildung 1* zeigt eine mögliche Topologie einer Tomcat-Server-Farm mit der Verwendung eines Memcached-Clusters auf. Dort legen alle Tomcat-Instanzen sämtliche Sitzungs-Informationen im Memcached-Cache ab, womit allen Servern sämtliche Session-Informationen zur Verfügung stehen. Diese Topologie kann durch die Rekonfigura-

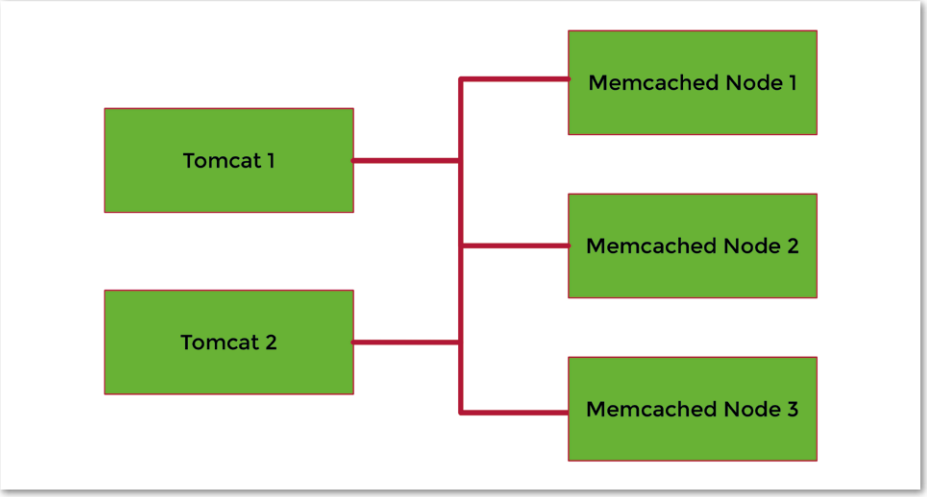


Abbildung 1: Tomcat-Cluster mit Memcached-Cache

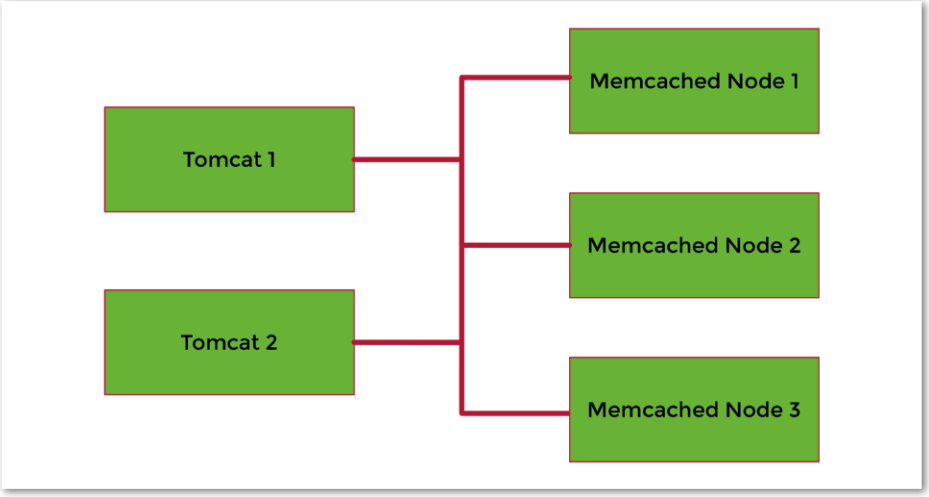


Abbildung 2: „Master & Read Replica"-Topologie

tion des Tomcat-Session-Manager *(siehe „https://code.google.com/p/memcached")* einfach erstellt und aufgebaut werden. Der durchgängige Einsatz von zustandlosen Komponenten sowie die Begrenzung des Zustands auf die Datenbanken und Client-Computer sind die Basis für eine skalierende Architektur.

Horizontale Skalierung

Im Cloud-Umfeld lassen sich Anwendung von einer kleinen Test-Installation bis auf eine weltweit funktionierende Plattform skalieren. Cloud-Umgebungen erlauben eine begrenzte vertikale Skalierung durch die Vergrößerung einzelner Rechenkapazitäten, etwa durch mehrere CPUs oder Arbeitsspeicher pro Server-Instanz. Neben den vertikalen Skalierungen können Anwendungen und auch Dienste horizontal durch die Duplikation von Server-Instanzen nahezu unbegrenzt skaliert werden.

Die Grenzen in der horizontalen Skalierung stellen meistens Datenbanken dar, da sich diese in der Regel nur vertikal aber selten horizontal skalieren lassen. Einzelne Datenbanken sowie auch Datenbank-Dienste der Cloud-Umgebungen erlauben es jedoch, einzelne Datenbanken horizontal für lesende Zugriffe zu skalieren. Dadurch können lesende Operationen, die häufig die Mehrzahl der Zugriffe darstellen, auf einer „read only"-Instanz durchgeführt werden, während eine zentrale Master-Instanz alle schreibenden Anfragen entgegennimmt. Die Last zwischen einer Master-Instanz und mehreren „read only"-Instanzen wird somit aufgeteilt. *Abbildung 2* zeigt den Aufbau einer Datenbank innerhalb der Amazon-Webservices-Cloud *(siehe „http://aws.amazon.com/de/rds")*, die eine Master- und mehrere lesende Instanzen besitzt.

Anwendungen können nun nach Zugriffsart auf eine „read only"-Instanz oder aber auf die Master- und „read only"-Instanz simultan zugreifen. Datenbanken wie MySQL bieten bereits Treiber, die den simultanen Zugriff auf Master- und „read only"-Knoten ermöglichen. Die Anwendungen müssen lediglich durch die Konfiguration der aktuellen JDBC-Connection dem Treiber signalisieren, ob es sich um einen lesenden oder schreibenden Zugriff handelt. Der Datenbank-Treiber sendet die Anfragen anschließend auf die jeweils korrekte Instanz. *Listing 1* zeigt die Verwendung einer Master- und „read only"-Datenbank innerhalb einer Java-Applikation.

```
public void demo() {
   ReplicationDriver driver = new ReplicationDriver();

   Connection conn = driver.connect("jdbc:mysql:replication://
master,slave1,slave2",new Properties());

   //schreibender Zugriff
   conn.setReadOnly(false);
   conn.setAutoCommit(false);
   conn.createStatement().executeUpdate("UPDATE....");
   conn.commit();

   //lesender Zugriff
   conn.setReadOnly(true);
   conn.createStatement().executeQuery("SELECT ....");
}
```

Listing 1

Das Listing dient lediglich zu Veranschaulichung. Das Connection-Management sollte in Enterprise-Applikation durch Frameworks (wie Spring) oder aber einem Application-Server erfolgen. Das Spring-Framework besitzt innerhalb des Spring-Cloud-AWS-Projekts bereits eine vollständige Unterstützung für die Konfiguration und Verwendung von Read-Replica-Instanzen *(siehe „http://cloud.spring.io/spring-cloud-aws")*.

Der Einsatz dieser Read-Replica-Instanzen hat jedoch auch Auswirkungen auf die Daten-Architektur einer Anwendung. Grundsätzlich garantieren Datenbanken atomare sowie konsistente Zugriffe. Beim Einsatz von Read-Replica-Instanzen kann es jedoch vorkommen, dass nach dem Schreiben der Daten diese beim nächsten Zugriff auf die „Read only"-Instanz noch nicht repliziert sind. Damit erhält die Anwendung unter Umständen veraltete Daten. Um dies zu verhindern, müssen Datenbank-Zugriffe so konzipiert sein, dass kritische Daten wie Bestands-Informationen und Buchungskonten immer von der Master-Instanz gelesen werden, während unkritische Stammdaten oder BI-Anwendungen auf die „Read only"-Instanzen zugreifen. Neben der Datenbank gilt es jedoch auch die Applikation selbst zu skalieren. Dies erfolgt in Cloud-Umgebungen optimalerweise automatisch.

Automatische Skalierung

Cloud-Umgebungen bieten die Möglichkeit, manuell einzelne Server-Instanzen hochzufahren oder diesen Vorgang komplett zu automatisieren. Diese Automatisierung kann dabei

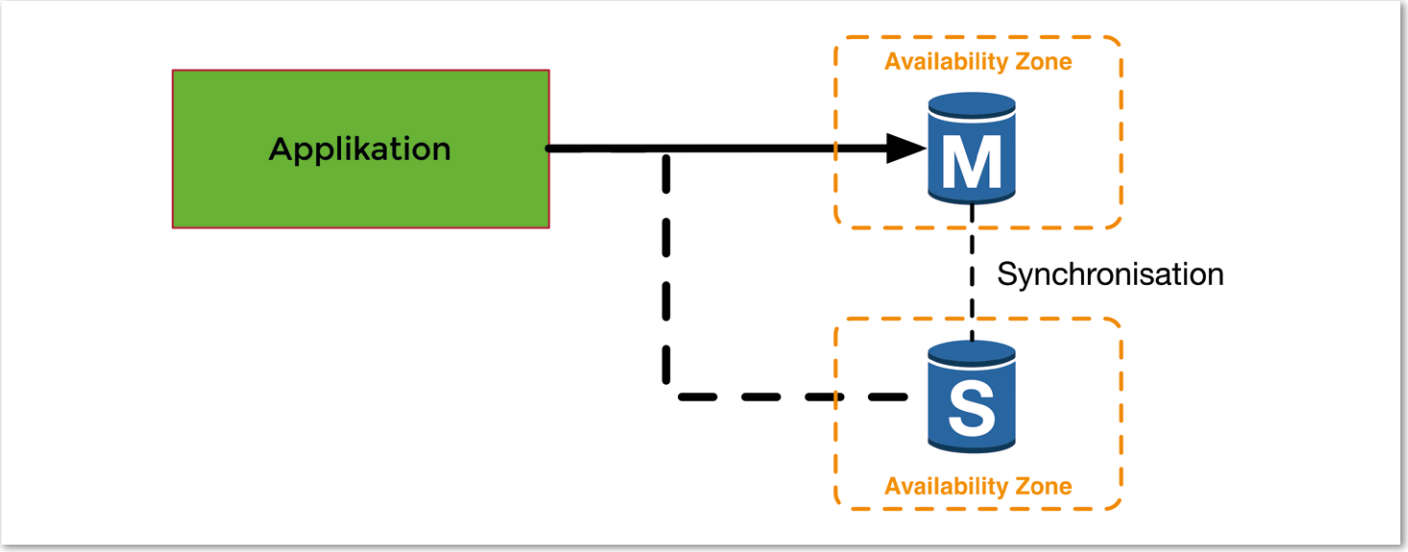


Abbildung 3: „Master & Slave"-Replikation

retrospektivisch auf Basis verschiedener Parameter wie CPU- oder Netzwerk-Auslastung der bestehenden Instanzen erfolgen. Bestenfalls kann es aber auch proaktiv auf Basis von externem Wissen über die zukünftige Auslastung geschehen. Dieses Wissen ist meistens bereits im Vorfeld während der Konzeptionsphase bekannt. So führt beispielsweise Netflix aufgrund des Konsumverhaltens der TV-Zuschauer bereits eine proaktive Skalierung ihrer Server- Kapazitäten durch *(siehe „http://techblog.netflix.com/2013/11/scryer-netflixs-predictive-auto-scaling.html")*.

Damit die retrospektivische oder proaktive automatische Skalierung funktioniert, ist es jedoch erforderlich, dass die Anwendung zustandlos arbeiten kann. Neben der bereits besprochenen Thematik von HTTP-Sessions gilt dies auch für Daten, die auf der Instanz abgelegt sind. Da Server in einer Cloud-Umgebung dynamisch gestartet und gestoppt werden, können keine Daten darauf abgelegt sein. Es dürfen daher keine Daten auf dem Server erzeugt, sondern auf einem zentralen Speicherdienst hochgeladen werden.

Praktisch jede Cloud-Plattform liefert einen hochverfügbaren Speicherdienst zur Ablage von Dateien. Dies ist zum Beispiel der Simple-Storage-Service auf der Amazon-Web-Service-Plattform. Neben den reinen Anwendungsdaten sollten auch Log-Daten zentral über eine Log-Aggregierungsplattform gesammelt und archiviert werden. Dieses Verfahren stellt sicher, dass für spätere Analysen keine wichtigen Diagnose-Informationen fehlen.

Bei abfallender Last wird eine Server-Instanz zuerst in einen passiven Modus versetzt, damit keine weiteren Anfragen an diese gesendet werden können. Anschließend terminiert man die Instanz mit ihren kompletten Daten. Neben der Skalierung sollten auch Möglichkeiten zur Last-Reduzierung auf den Servern in Betracht gezogen werden. Dies wird unter anderem durch den Einsatz von Content Delivery Networks erreicht.

Content Delivery Network

Enterprise-Anwendungen besitzen einen großen Anteil an statischen Ressourcen. Bei Web-Anwendungen sind dies zumeist Bilder, JavaScript-Code sowie auch HTML-Fragmente. Diese statischen Ressourcen kann ein Application-Server bereitstellen, was jedoch unnötige sowie verhältnismäßig teure Rechenzeit beansprucht. Die Verlagerung aller statischen Inhalte in ein Content Delivery Network kann alle Anfragen von weltweit verteilten Edge-Servern bedienen. Die Anwendungsserver selbst beantworten lediglich dynamische Anfragen, die zu einer Geschäftsfunktion führen.

Content Delivery Networks sind kostengünstige Möglichkeiten, Daten performant auf Basis der verbrauchten Bandbreite, und nicht der Rechenzeit, zur Verfügung zu stellen. Daher sollte ein Architektur-Entwurf gerade bei Web-Applikationen auch die Evaluation einer Client-lastigen Architektur vorsehen. Dabei erfolgt das Rendering der Inhalte komplett auf dem Browser-Client. Der Server bedient lediglich REST/JSON-Webservice-Anfragen. Alle anderen Inhalte stellt ein Content Delivery Network dem Client zur Verfügung. Bei dessen Einsatz sollten die Inhalte möglichst auf einen geringen Bandbreiten-Verbrauch hin optimiert abgelegt werden. Verschiedene Maßnahmen, wie Minifizierung von JavaScript, Konvertierung von HTML in JavaScript-Templates sowie die Verwendung von CSS-Sprites, können den Bedarf an Bandbreite für die Bereitstellung der Inhalte, und die damit verbundenen Kosten, drastisch reduzieren.

Der Einsatz von redundanten sowie globalen Edge-Servern macht die Content Delivery Networks auch hochredundant. Dies gilt jedoch nur dann, wenn auch sichergestellt ist, dass die Applikationsserver sowie Datenbanken über mehrere Rechenzentren verfügbar sind.

„n+1"-Design

Ein „n+1"-Design stellt sicher, dass sämtliche Komponenten redundant ausgelegt sind. Dies ist vor allem beim Einsatz von „Commodity class"-Hardware notwendig, da die Ausfallwahrscheinlichkeit verglichen mit hochverfügbarer Hardware deutlich höher ist. Bei dem erwähnten Beispiel zur horizontalen Skalierung von Datenbanken sind zwar im laufenden Betrieb die Lesezugriffe optimiert, ein Ausfall der Master-Datenbank würde jedoch einen Totalausfall für alle schreibenden Operationen in der Datenbank bedeuten. Daher ist es ratsam, die komplette Datenbank für den Fall, dass die Master-Datenbank ausfällt, ein weiteres Mal zu spiegeln.

Hochverfügbare Datenbank-Dienste der Cloud-Hersteller erlauben es, sogenannte „Slave-Datenbanken" zu erstellen. Diese können von der Anwendung nicht direkt benutzt werden. Zudem befinden sich die Slave-Datenbanken meist in einem anderen, isolier-

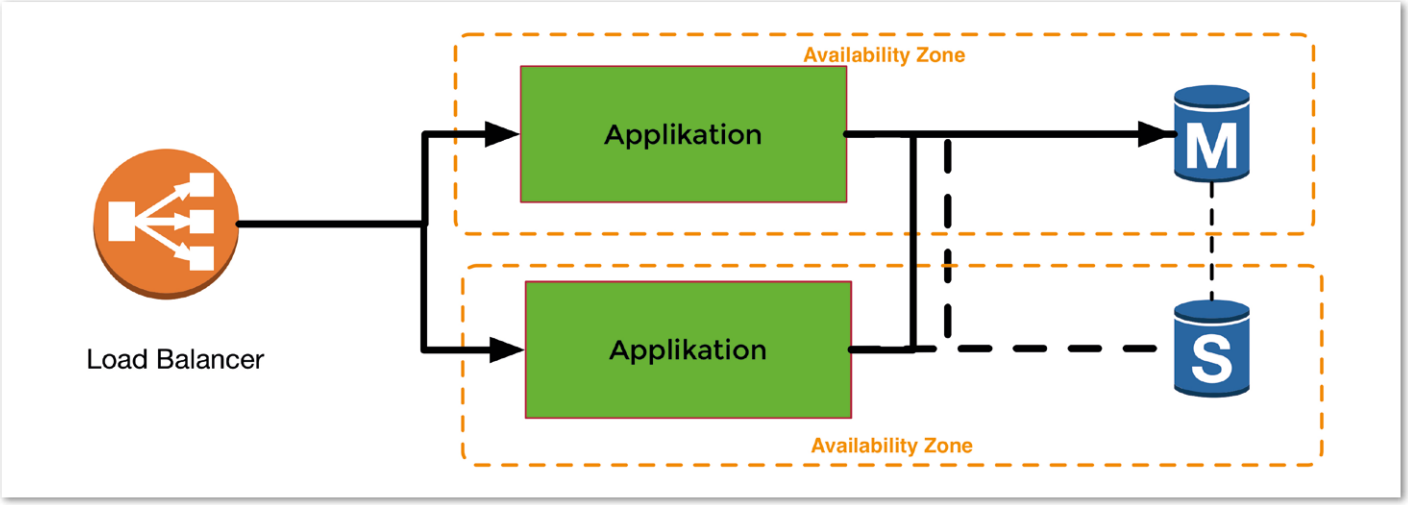


Abbildung 4: Parallelbetrieb einer Anwendung über mehrere Rechenzentren

ten Rechenzentrum. Bei einem Ausfall des Master-Knoten wird auf den Slave-Knoten umgestellt. Die Anwendung kann dann auf dem neuen Master-Knoten ihre Verarbeitung fortsetzen.

Abbildung 3 zeigt den Aufbau einer „Master & Slave"-Datenbank-Topologie auf Basis des Amazon-Relational-Database-Service. Mit dieser lassen sich Datenbank-Ausfälle in einem Rechenzentrum kompensieren. Der Ausfall eines Rechenzentrums kann jedoch auch die Anwendungsserver betreffen. Daher muss im Rahmen des „n+1"-Designs die Anwendung selbst redundant über mehrere Rechenzentren ausgelegt sein, um deren Komplettausfall im Vorfeld auszuschließen.

In *Abbildung 4* ist die komplette Anwendung auf mehrere Rechenzentren („Availability Zones") dupliziert verteilt. Im Regelbetrieb greifen beide Anwendungen auf eine zentrale Master-Datenbank zu. Bei deren Ausfall greifen beide Anwendungen auf die Slave-Datenbank zu. Beide Anwendungen werden im Betrieb von einem hochverfügbaren Load Balancer *(siehe „http://aws.amazon.com/de/elasticloadbalancing")* angesprochen. Beim Ausfall einer Anwendung wird der gesamte Traffic auf die verbleibende Instanz weitergeleitet. Neben der Ausfallsicherheit können Cloud-Anwendungen auch auf mehrere geografisch unabhängige Regionen verteilt sein.

Globale Skalierung

Alle großen Cloud-Dienste erlauben es, Anwendungen weltweit über mehrere Regionen zu verteilen. Das kann vor allem bei international eingesetzten Anwendungen wie Suppy-Chain-Management-Software oder Kunden-Applikationen Sinn machen, die weltweit aufgerufen werden. Die Daten sind in diesem Fall meistens sowieso geografisch getrennt gelagert, sodass eine Replikation zwischen den einzelnen Regionen nicht notwendig ist. Damit lassen sich auch optimale Zugriffszeiten bei Verwendung eines Content Delivery Network und einem möglichst Client-nahen Rechenzentrum erzielen.

Failure Injection Testing

Auch bei einer optimal verteilten Anwendung über mehrere Rechenzentren und Regionen können Ausfälle jederzeit vorkommen. Optimalerweise werden die Ausfälle bereits im laufenden Betrieb permanent simuliert. Unter „Failure Injection Testing" wird die laufende Simulation von Fehlern im Produktivsystem verstanden. Somit sind Fehlersituationen Teil des täglichen Betriebs.

Die Verfahren im Failure Injection Testing beginnen bei der Simulation von langsamen Service-Aufrufen, indem zwischen dem Client und dem Server eine künstliche Latenz injiziert wird, um das korrekte Load-Balancing sowie Timeout-Verhalten der Anwendung sicherzustellen. Darüber hinaus können auch im laufenden Betrieb Instanzen terminiert werden.

Eine optimal skalierende sowie ausfallsichere Anwendung sollte den Ausfall einer unkritischen Masse von Server-Instanzen ohne Auswirkungen auf das Anwendungsverhalten verkraften. Internet-Konzerne wie Netflix stellen bereits verschiedene Tools bereit, um ein Failure Injection Testing umzusetzen *(siehe „http://techblog.netflix.com/2011/07/netflix-simian-army.html")*.

Fazit

Dieser Artikel beschrieb einen begrenzten Ausschnitt aus möglichen Entwurfsprinzipien für Cloud-Architekturen. Weitere Prinzipien wie asynchrones Design, sowie die Umsetzung einer Microservice-Architektur folgen eventuell in einer der nächsten Ausgaben. Der Umstieg auf eine der Public- oder Private-Cloud-Plattformen bietet enorme Chancen, hält aber auch den einen oder anderen Fallstrick in der Umsetzung von Applikationen in der Cloud bereit.

Agim Emruli
agim.emruli@mimacom.com

Agim Emruli ist Geschäftsführer bei mimacom und beschäftigt sich mit Software-Architekturen auf Basis von Amazon-Web-Services und Cloud Foundry. Er ist zudem Spring-Cloud-Committer.

http://ja.ijug.eu/15/2/3

„Ich glaube, dass die Expression Language der heimliche Held der gesamten Web-Ebene von Java EE ist …“


Ed Burns

JavaServer Faces (JSF) spielt nach wie vor eine wichtige Rolle als Framework zur Entwicklung grafischer Benutzeroberflächen für Web-Applikationen. Bernd Müller, Professor für Software-Technik und Mitglied der JUG Ostfalen, sprach für Java aktuell darüber mit Ed Burns, Consulting Member of the Technical Staff Oracle America Inc. und Spec Lead für JavaServer Faces.

Vor knapp zwei Jahren ist das neue Release von JavaServer Faces herausgekommen. Wird JSF 2.2 bereits in neuen Projekten eingesetzt?
Ed Burns: Es trifft zwar zu, dass Java EE 7 im Frühsommer 2013 auf den Markt kam, doch kommerziell einsetzbare Implementierungen sind noch in der Entwicklungsphase. Normalerweise müssen die Spec Leads nach dem Regeln des Java Community Process eine Referenz-Implementierung bereitstellen, was auch für Oracle gilt. Wir haben GlassFish 4.0 als Referenz-Implementierung für Java EE 7 freigegeben. Obwohl ich persönlich davon überzeugt bin, dass GlassFish ein adäquater Applikations-Server für den produktiven Einsatz ist, sehe ich, dass viele Kunden sehr gespannt auf WebLogic 12.2.1 sind, die erste Version unseres Flaggschiff-Applikations-Servers, die Java EE 7 unterstützt.

Expression Language 3 unterstützt bestimmte Lambda-Ausdrücke. Bedeutet dies, dass JSF und Java EE jetzt Java 8 vollständig unterstützen oder dass zumindest die gleiche Funktionalität verfügbar ist?
Ed Burns: Das ist eine gute Frage. Ich freue mich sehr über die Aufnahme der Lambda-Ausdrücke in Java EE 7 über EL 3.0. Ich glaube, dass EL der heimliche Held der gesamten Web-Ebene von Java EE ist, und Java-Apps auf der Server-Seite sich deutlich von reinen clientseitigen Technologien wie AngularJS unterscheiden. Wenn man die Stärke der Lambda-Ausdrücke kennt, aber nicht nach Java 8 migrieren kann, bietet sich die Verwendung von EL 3.0 auf Java SE 7 ab, entweder innerhalb einer EE-App oder auch standalone auf Java SE. Wir planen, EL in Java EE 8 zu überarbeiten, um die Verwendung von Lambda-Ausdrücken anzugleichen, da sie einen großen Teil der Stärke von Java SE 8 ausmachen.

Es gab einige Unzufriedenheit in der Entwickler-Gemeinde, weil neue Releases von Avatar 2.0 und Angular 2.0 nicht kompatibel mit Vorgängerversionen sind. Welche Regeln bestehen für die Kompatibilität in Java EE und kann JSF nicht-kompatible Änderungen abfangen, Stichwort: „Old School-Web-Framework“?
Ed Burns: Ich sprach im Dezember 2014 über die Kompatibilitäts-Geschichte für Java EE in einem Tweet unter *„https://twitter.com/edburns/status/540137166072860672“*. Dieser Tweet ist mit einem Dokument verlinkt, in dem Java-EE-Architekt Bill Shannon detailliert unsere Kompatibilitäts-Regeln beschreibt. Ich bin nicht in der Position, auf Avatar einzugehen, glaube aber, dass es sicherlich gute Gründe gibt, die UI-Logik aus der Server-Ebene in die Browser-Ebene zu verschieben. Die Code-Komplexität ist jedoch wahrscheinlich keiner davon.

JSF 2.2 unterstützt HTML5 out of the Box. Gibt es Pläne, wie es mit HTML5 weitergeht?
Ed Burns: HTML5 kam gerade erst Ende Oktober 2014 heraus. Eine wichtige Entscheidung, die sehr spät in der Entwicklung von HTML5 fiel, nenne ich das „große Streichen von Anforderungen“. Danach reduzierte sich die Größe der HTML5-Spezifikation durch die Auslagerung früherer Kern-Technologien von HTML5 in separate unabhängige Spezifikationen. Mehrere davon waren prädestiniert für JSF-Entwicklung, etwa Web-Speicher und WebSocket. Da es sich jetzt um separate Spezifikationen handelt, kann JSF nicht mehr sicher darauf bauen. Ein weiterer zu berücksichtigender Aspekt ist, dass die Komponenten-Bibliotheken, die auf JSF aufbauen, sehr gut geeignet sind, um HTML5 und verwandte Technologien zu nutzen.

Es gibt einen neuen JSR für den Model View Controller (MVC). Was ist der Grund dafür und was bedeutet dieser für JSF?
Ed Burns: Wie ich in meinem Oracle-Tech-Net-Artikel „Warum ein weiterer MVC?“ auf *„http://www.oracle.com/technetwork/articles/java/mvc-2280472.html“* bereits sagte, war der entscheidende Faktor für MVC die Anforderung der Anwender. Kurz gesagt, wenn man den Spring-Stack mit dem Java-EE-Stack vergleicht, umfasst Spring das Action-orientierte MVC und Java EE nicht. Die MVC-Spezifikation soll diese Lücke füllen.

In Java EE 7 ist die HTML5-Verfügbarkeit ein treibendes Thema. Was bedeutet das genau und welche Auswirkungen hat das auf Java EE 8?
Ed Burns: Vor dem großen Abspecken des HTML5-Plans im Jahr 2014 *(siehe „http://dev.w3.org/html5/decision-policy/html5-2014-plan.html“)* konnte man sagen, dass HTML5 gleichbedeutend war mit einer „offenen Web-Plattform“. In diesem Sinn haben wir den Begriff HTML5 in Java EE 7 verwendet. Konkret bedeutet es eine Sammlung von Technologien, die es erleichtern, Apps zu erstellen, die die Komponenten der Open-Web-Plattform im Rahmen des Server-seitigen

Java verwenden. Dazu gehören insbesondere JSON, WebSocket, JAX-RS sowie JSFs „HTML5 Friendly Markup"-Feature.

Servlet 4.0 soll HTTP/2 für Java EE bringen. Warum ist HTTP/2 so wichtig und welche Funktionen bietet Servlet 4.0 für Entwickler?
Ed Burns: HTTP/2 ist für Entwickler wichtig, weil es ist die erste Revision des grundlegendsten und populärsten Aplications-Protokolls seit mehr als einem Jahrzehnt ist. Das sichtbarste Feature von HTTP/2 für Servlet-Entwickler wird wahrscheinlich Server-Push sein.

Ich habe gehört, dass HTTP/2 über Vollduplex-Kommunikation und Server-Push verfügt. Ist das kein Ersatz für WebSocket?
Ed Burns: Es ist zwar richtig, dass HTTP/2 eine volle Duplex-Frame-basierte Kommunikation zwischen Client und Server bietet, aber es gibt keine entsprechende JavaScript-API für den Zugriff im Browser, wie bei Server-Sent-Ereignisse („EventSource") und WebSocket. Dies ist der wichtigste Grund dafür, warum diese HTTP/2-Funktionen kein Ersatz für WebSocket sind.

Was bedeutet HTTP/2 für Java EE und speziell für JSF? In welcher Form wird HTTP/2 Java EE beeinflussen?
Ed Burns: HTTP/2 bedeutet für Java EE dasselbe wie für das Web im Allgemeinen: die Ladezeiten der Seiten werden schneller. Traditionell ist dies für überwiegend statische Sites mit wenig oder gar keiner Interaktion wichtig. Beispielsweise profitieren Nachrichtenseiten enorm von HTTP/2, wie in dem Artikel unter *„http://www.phillipadsmith.com/2014/07/deploying-your-static-news-app-like-a-samurai.html"* beschrieben ist.

Die Freiheiten in der Front-End Entwicklung sind meist unberechenbar. Wie reagierte JSF auf diese Volatilität und Bedeutung über die Jahre?
Ed Burns: Eine Sache, die JSF von anderen Web-Frameworks unterscheidet, ist ihr Charakter als eine Abstraktion und nicht nur als ein Web-Framework. Man kann JSF als ein Architektur-Muster betrachten, das beschreibt, wie alle Teile einer Web-Anwendung arbeiten und wie sie interagieren. Mit dieser Sicht können sich die individuellen Teile, die eine Web-App ausmachen, mit Trends und Best Practices ändern, aber der Gesamtrahmen ist immer noch gleich. Kurz gesagt, Abstraktionen überdauern die Zeit.

Hinweis: Ed Burns hält im Rahmen der JavaLand 2015 einen Schulungstag zum Thema "Java EE 7 aus einer HTML5-Perspektive".

Bernd Müller
bernd.mueller@ostfalia.de

Bernd Müller ist Professor für Software-Technik an der Ostfalia (Hochschule Braunschweig/Wolfenbüttel). Er ist Autor mehrere Java-EE-Bücher, Sprecher auf nationalen und internationalen Konferenzen und engagiert sich in der JUG Ostfalen sowie im iJUG.

http://ja.ijug.eu/15/2/4

Einstieg in die Liferay-Portal-Entwicklung unter Verwendung von JSF

Frank Schlinkheider und Wilhelm Dück, ITSD Consulting GmbH

Liferay ist der am meisten verbreitete Portal-Server und JSF ist der offizielle Java-Standard für die Web-Entwicklung. Dieser Artikel zeigt, wie beide Technologien zusammenspielen und auf welche Art und Weise mit JSF einfache und effektive Liferay-Portlets entwickelt werden. Dabei lassen sich vorhandene Funktionalitäten wie der Team-Kalender, das Berechtigungssytem und der WYSIWG-Editor einfach nutzen. Zudem werden Gründe für den Einsatz geliefert, aber auch die Nachteile nicht verschwiegen.

Für Liferay sprechen viele Gründe; hier die wichtigsten:

- Unterstützung aller gängigen Industrie-Standards
- Erfüllung aller aktuellen Anforderungen eines Portal-Servers (Navigation, Themes, umfangreiches Rechte/Rollen-Konzept, SSO, Dokumenten-Management etc.)
- Es ist das einzige Produkt aus dem Open-Source-Umfeld, das laut Gartner mit Produkten von Microsoft, IBM und SAP mithalten kann
- Großer Funktionsumfang (Wiki, Foren, Benachrichtigungen, Sozialnetzwerk
- Komponenten etc.)
- Große aktive Community
- Workflow-Unterstützung (Geschäftsprozesse)
- Offene, flexible SOA-Strategie
- Enterprise-Support

Noch ein Schwergewicht

Man hört hin und wieder, dass Java Server Faces (JSF) nicht mehr zeitgemäß sei. Mit neueren Web-Technologien wie GWT, Wicket, Vaadin oder auch jQuery seien schneller und einfacher komfortable Web-Oberflächen zu entwickeln. Nichtsdestotrotz spricht vieles für den Einsatz des Java-Standard-Frameworks. Es gibt viele Entwickler, die über entsprechendes JSF-Know-how verfügen, und es existieren entsprechende Migrationswege für die Portierung von vorhandenen JSF-Web-Anwendungen zu neuen Portal-Anwendungen. Zudem wird standardmäßig JSF in den unterschiedlichen Portal-Servern mittels der später noch vorgestellten Faces Bridge unterstützt. Dank Komponenten-Bibliotheken wie PrimeFaces etc. lassen sich auch mit dem Standard-Web-Framework (JSF) anspruchsvolle Web-Anwendungen für den Unternehmenseinsatz entwickeln.

Allerdings gibt es auch Nachteile für den Einsatz von JSF in Liferay. Die mit Liferay ausgelieferten Standard-Portlets sind meistens nicht mit JSF entwickelt. Hier findet man Standard-JSP, Scriptlets, Struts, etc. Das bedeutet, für die Anpassung dieser Portlets finden, nach unserer Meinung, teilweise veraltete oder zumindest uneinheitliche Web-Technologien Verwendung. Ein weiterer Nachteil ist, dass JSF Komponentenbibliotheken wie z.B. PrimeFaces über grafische Komponenten verfügen, die sich bzgl. „Look and Feel" etwas anders verhalten als die Standard-Elemente von Liferay. Um ein einheitliches Erscheinungsbild im Portal zu erreichen, sollten die verwendeten Komponenten entsprechend ausgewählt werden. Liferay bietet inzwischen eine Reihe JSF Komponenten, die hier entgegenwirken und dem Layout und der Funktionalität der sonst in Liferay eingesetzten grafischen Bibliothek entsprechen.

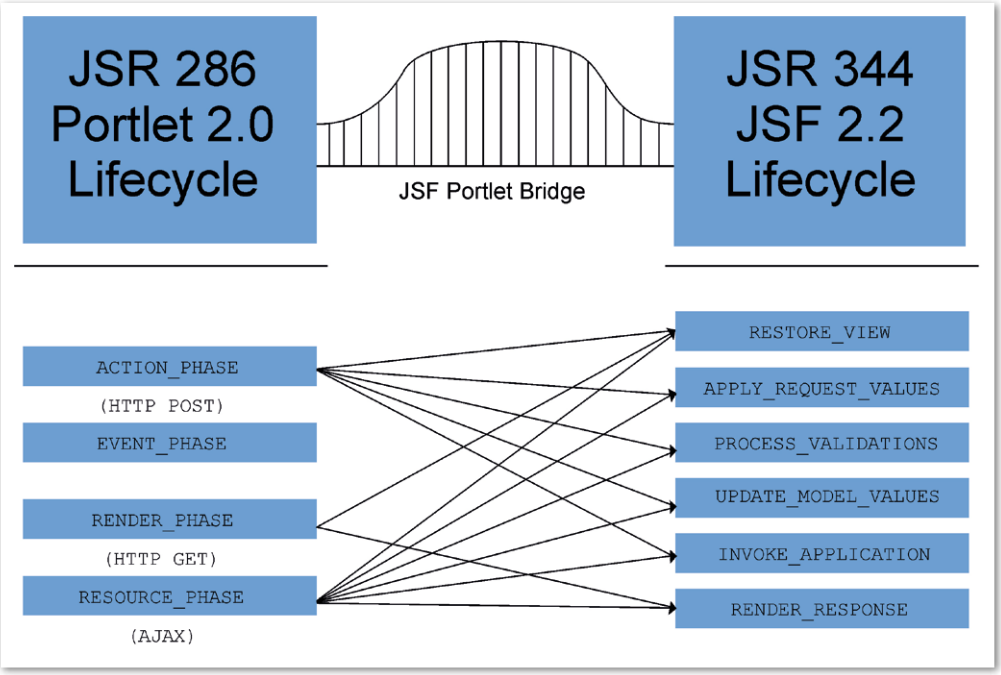


Abbildung 1: Portlet LifeCycle auf Faces LifeCycle mittels Bridge

Wie JSF ins Portal kommt

Nachdem die beiden JSRs für JSF und Portlets bei deren Spezifikation wenig Rücksicht

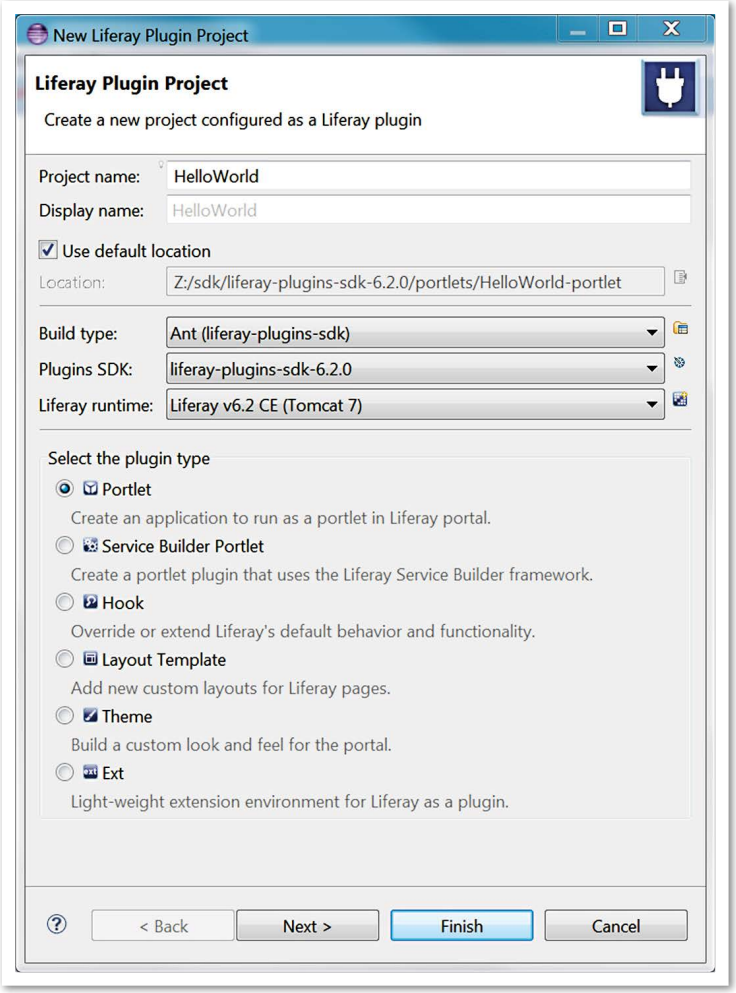


Abbildung 2: Wizard zur Erstellung eines Liferay-Plug-in-Projekts

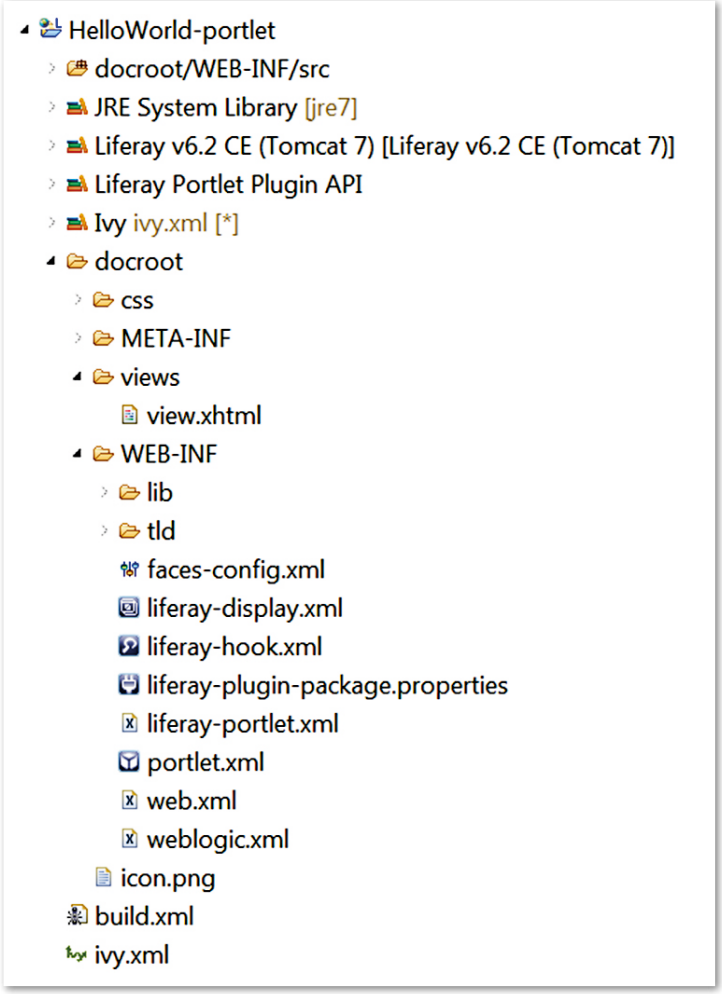


Abbildung 3: Projekt-Struktur eines Liferay-Plug-in-Projekts

aufeinander genommen haben, muss zwischen beiden eine Brücke geschlagen werden. Mit der JSR 301 wurde für Portlet 1.0 und JSF 1.2 und später mit der JSR 329 für die Portlet 2.0 Spezifikation die Faces Bridge ins Leben gerufen. Sie soll folgende Aufgaben erfüllen:

- JSF dem Portlet-Container zur Verfügung stellen
- JSF LifeCycle und Portlet LifeCycle aufeinander abstimmen
- Portal-Funktionalität der JSF-Anwendung verfügbar machen (etwa die Inter-Portlet-Kommunikation)

Abbildung 1 gibt einen einfachen Überblick, wie die Phasen des Portlet LifeCycle auf die des Faces LifeCycle mittels der Bridge abgebildet werden.

Aufmerksamen Lesern wird auffallen, dass hier kein JSR für den Einsatz von Portlet 2.x/3.0 in Kombination mit JSF 2.x genannt wird. Tatsache ist, dass es hier bisher noch keinen Standard gibt. Aktuelle Bridge-Implementierungen unterstützen aber zumindest Portlet 2.0 in Kombination mit JSF 2.x.

JBoss hat eine der ersten Bridge-Implementierungen geliefert. Der Einsatz im Liferay-Portal war aus unterschiedlichen Gründen oft mühsam und schwierig. Bei Problemen saß der Entwickler oft zwischen den Stühlen. Das eine Open-Source-Projekt verwies auf die nicht korrekte Umsetzung der fremden Komponente und umgekehrt. Man kann sagen, dass die beiden sich nicht immer grün waren; trotzdem wurde die Kombination „Liferay und JBoss Faces Bridge" produktiv in Projekten eingesetzt.

Im Jahr 2011 entschied sich Liferay, eine eigene Faces Bridge einschließlich Liferay-Erweiterungen zu entwickeln beziehungsweise weiterzuentwickeln. Folgende Vorteile ergaben sich dadurch:

- Alles aus einer Hand: Keine Schnittstellen-Probleme durch unterschiedliche Hersteller-Implementierungen
- Gute Unterstützung in der Liferay-Entwicklungsumgebung
- Keine Interoperabilitätsprobleme: Jede Liferay-Version besitzt immer eine passende Faces-Bridge-Version
- Einfacher Zugriff auf den Liferay Context mittels EL-Expressions (aktuell angemeldeter User, Permissions etc.)
- Eigene JSF-Tag-Lib, die die Verwendung von Liferay-Controls innerhalb von JSF ermöglicht, etwa für die Verwendung des Liferay-HTML-Editors, der auch für das Wiki, Forum etc. Verwendung findet

„Hello World"-Beispiel mit Wizard

Nach der Theorie ein kleines Praxis-Beispiel: Die eingesetzte Liferay-IDE 2.2.x basiert auf Eclipse Luna und wurde um wichtige Liferay-Plug-ins und Wizards erweitert. Als Erstes entsteht mit dem Wizard ein neues Liferay-Projekt *(siehe Abbildung 2)*.

Für das Beispiel dient der Plug-in-Type „Portlet" und dessen Name soll „HelloWorld" lauten. An den Projekt-Namen wird damit au-

Abbildung 4: Ausgabe des HelloWorld-Portlets (englisch)

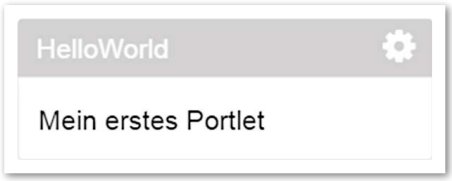


Abbildung 5: Ausgabe des HelloWorld-Portlets (deutsch)

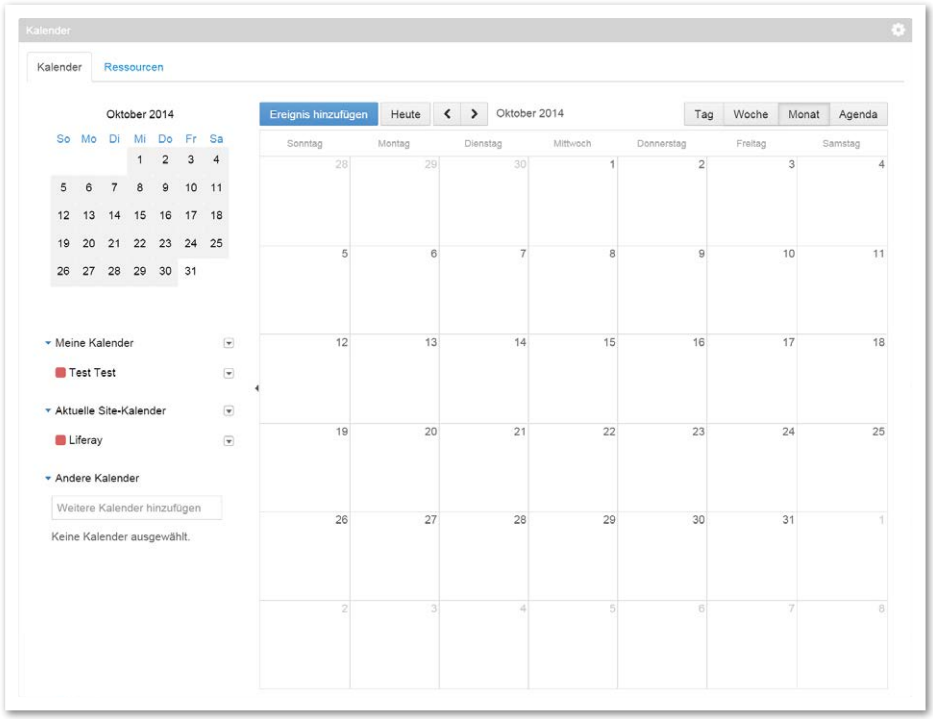


Abbildung 6: Liferay-Kalender-Portlet

tomatisch „-portlet" angehängt. Im nächsten Schritt legt man fest, dass es sich um ein JSF 2.x-Portlet mit der PrimeFaces-Bibliothek handeln soll. Auf diese Art entsteht auf einfache Weise eine JSF-2.x-Portlet-Projekt-Struktur. Die benötigten Komponenten und Bibliotheken (JSF, PrimeFaces etc.) werden dabei automatisch in das Projekt geladen. *Abbildung 3* zeigt die Projekt-Struktur.

Im Verzeichnis „docroot\WEB-INF" sind die Konfigurationsdateien zu finden, insbesondere „portlet.xml", in der die meisten Portlet-spezifischen Konfigurationen vorgenommen werden. Die „xhtml"-Seiten sind im Verzeichnis „docroot\views" und der Classpath für die Java-Dateien lautet „docroot\WEB-INF\src". Das Projekt kann nun direkt auf den zuvor eingerichteten Liferay-Server eingerichtet und das Portlet auf einer Seite hinzugefügt werden. Innerhalb des Portlets wird der Text „Hello HelloWorld!" ausgegeben *(siehe Abbildung 4)*.

Beim genauen Betrachten der Datei „view.xhtml" unter „docroot\view" stellt man fest, dass der Ausgabetext nicht vorkommt. Stattdessen kommt wie üblich Expression Language (EL) zum Einsatz: „<h:outputText value="#{i18n['HelloWorld-hello-world']}" />".

„i18n" steht bekanntlich für „internationalization" und es wird in entsprechenden Sprachdateien für die aktuell eingestellte Sprache nach dem in Hochkommata gesetzten Schlüssel (in unserem Fall „HelloWorld-hello-world") gesucht und der zugeordnete Wert ausgegeben. Der Wizard hatte die „Language_en_US.properties" generiert.

Um sicherzustellen, dass für alle Sprachen wenigstens immer eine englische Begrüßung erscheint, muss die Sprachdatei

```
<hook>
     <language-properties>
        Language.properties
     </language-properties>
  <language-properties>
        Language_de_DE.properties
     </language-properties>
</hook>
```

Listing 1

```
<h:form>
  <h:panelGrid columns="2">
  <!-- Titel -->
  <h:outputText value="Titel:" />
  <h:inputText value="#{calendarBean.title}" />
  </h:panelGrid>

  <h:outputText value="Beschreibung:" />
  <br />
  <!-- Beschreibung über ein Liferay-JSF-Tag -->
  <liferay-ui:input-editor value="#{calendarBean.description}" />

  <h:panelGrid columns="2">
  <h:outputText value="Kalender:" />
  <!-- Auswahl des Kalenders -->
  <h:selectOneMenu value="#{calendarBean.calendarName}">
     <f:selectItems value="#{calendarBean.calendars}" />
  </h:selectOneMenu>

  <h:outputText value="Datum:" />
  <!-- Datum (mit dem Datepicker von PrimeFaces) -->
  <p:calendar value="#{calendarBean.startDate}" pattern="dd/MM/yyyy" />

  <h:outputText />
  <!-- Button zum Speichern (Methode „save" aus CalendarBean) -->
  <h:commandButton action="#{calendarBean.save}" value="Speichern" />
  </h:panelGrid>
</h:form>
```

Listing 2

„Language_en_US.properties" in „Language.properties" im Portlet-Projekt umbenannt werden.

Für die deutsche Übersetzung legt man eine zweite Sprachdatei mit dem Namen „Language_de_DE.properties" an, kopiert den Inhalt aus der Standard-Sprachdatei „Language.properties", fügt ihn in die deutsche Sprachdatei ein und trägt beim entsprechenden Key den Text „Mein erstes Portlet" ein: „HelloWorld-hello-world=Mein erstes Portlet". Damit beide Dateien aktiviert werden, müssen sie als Sprachdateien registriert sein. Dies geschieht in der Datei „liferay-hook.xml" *(siehe Listing 1)*. *Abbildung 5* zeigt dann das Ergebnis.

```
public void save() throws SystemException, PortalException {

  ExternalContext ec = FacesContext.getCurrentInstance()
           .getExternalContext();

  PortletRequest request = (PortletRequest) ec.getRequest();

  // liefert die ID des in der Liste ausgewählten Kalenders
  long calendarId = getCalendarId();

  // aus Titel und Beschreibung werden zwei Maps inkl.
  // Übersetzungen erstellt
  Map<Locale, String> titleMap = new HashMap<>();
  titleMap.put(LocaleUtil.getDefault(), title);
  Map<Locale, String> descriptionMap = new HashMap<>();
  descriptionMap.put(LocaleUtil.getDefault(), description);

  // holen der ServiceContext-Informationen
  ServiceContext serviceContext = ServiceContextFactory
           .getInstance(CalendarBooking.class.getName(), request);

  // Erzeugen eines Kalendereintrages in Liferay
  CalendarBookingLocalServiceUtil.addCalendarBooking(
     Long.parseLong(ec.getUserPrincipal().getName()), calendarId,
     new long[0], 0, titleMap, descriptionMap, "", startDate(),
     endDate(), false, "", 900000, "email", 300000, "email",
     serviceContext);
 }
```

Listing 3

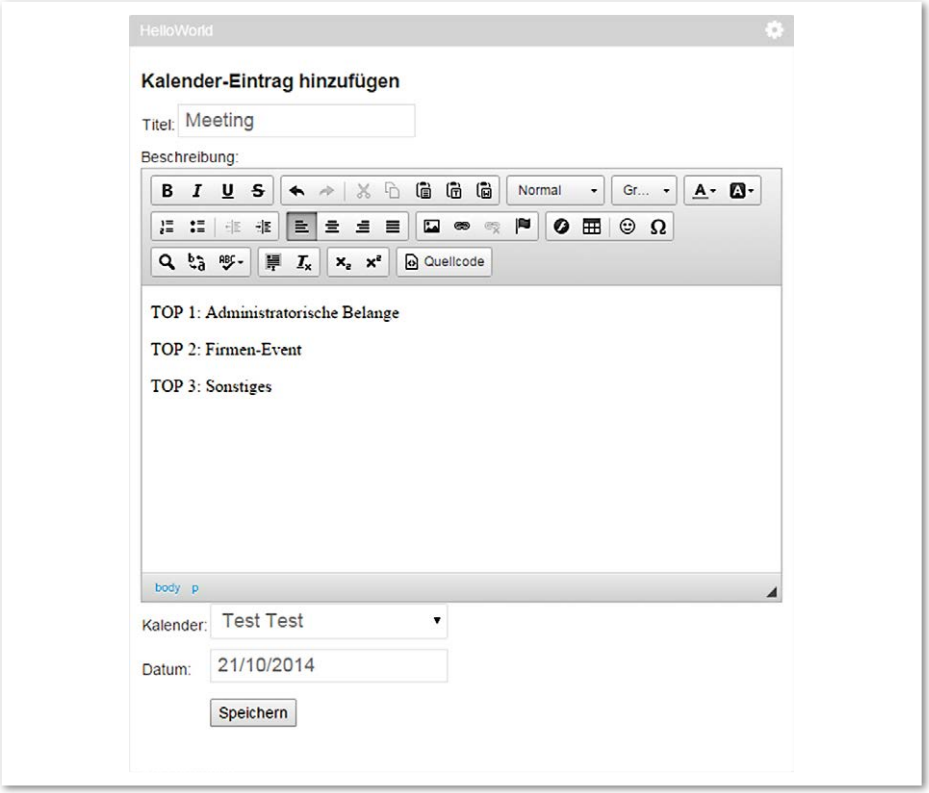


Abbildung 7: „Hello World"-Portlet für die Anlage eines Termins

Anlegen eines Kalender-Events

Eines der von Liferay mitgelieferten Portlets ist „calendar portlet". Dieses wurde für die Version 6.2 von Liferay komplett überarbeitet und verfügt nun über echte Team- und Einladungsfunktionalitäten. Ziel ist es nun, mittels des Portlets einen Team-Kalender-Eintrag zu erzeugen. *Abbildung 6* zeigt den zu Beginn leeren Kalender.

Um das Ziel zu erreichen, werden in der Datei „view.xhtml" die entsprechenden Felder für die Eingabe angelegt *(siehe Listing 2)*. Zusätzlich entsteht die bereits erwähnte Klasse „CalendarBean", die per Annotation als „ManagedBean" gekennzeichnet ist. Sie verwaltet die Attribute „title", „description", „calendarName" sowie „startDate" und bietet eine save-Methode zum Speichern der Eingaben *(siehe Listing 3)*.

Um einen Kalender-Eintrag erstellen zu können, kommt die Service-Klasse „CalendarBookingLocalServiceUtil" des Kalender-Portlets von Liferay zur Anwendung. Sie bietet die Funktion „addCalendarBooking()", die mit den übergebenen Parametern einen neuen Kalender-Eintrag erstellt. Nach dem Deployment des Portlets lassen sich anschließend die Kalender-Informationen eintragen und ein Termin in Liferay erzeugen *(siehe Abbildung 7)*.

Berechtigungen anlegen

Diese Erweiterung zeigt, wie einfach JSF auf das Liferay-Berechtigungssystem zugreifen kann. Um das Anlegen eines Kalender-Eintrages mit einer entsprechenden Berechtigung versehen zu können, ist in der Datei „resource-actions/default.xml" ein Eintrag einzufügen *(siehe Listing 4)*.

Innerhalb des Tags „supports" werden die Berechtigungen definiert, die das Portlet unterstützt. Im Tag „guest-defaults"

```
<permissions>
  <supports>
     <action-key>ADD_ENTRY</action-key>
  </supports>
  <guest-defaults>
  </guest-defaults>
  <guest-unsupported>
     <action-key>ADD_ENTRY</action-key>
  </guest-unsupported>
</permissions>
```

Listing 4

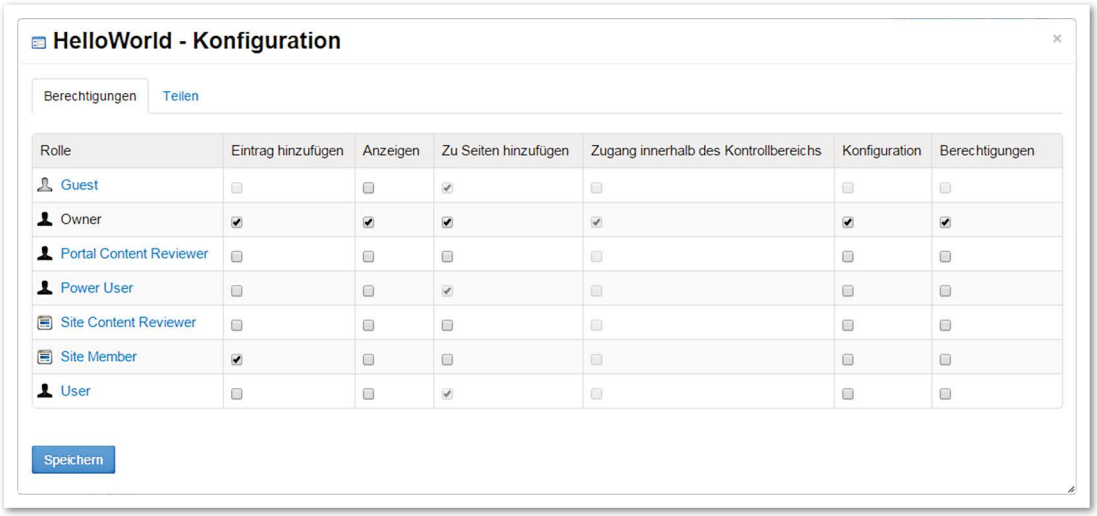


Abbildung 8: Berechtigungen setzen

werden die Standard-Berechtigungen eines nicht angemeldeten Benutzers festgelegt und unter „guest-unsupported" die Berechtigungen festgelegt, die einem unangemeldeten Nutzer niemals zugewiesen werden sollten.

Um die Berechtigungsdefintionen zu registrieren, wird in „portlet.properties" (Datei muss im Classpath liegen) „resource.actions.configs=resource-actions/default.xml" eingetragen. Die Berechtigungen lassen sich nach dem Bereitstellen des Portlets von einem angemeldeten Benutzer mit entsprechender Berechtigung einfach ändern. Dazu muss er im Portal in der oberen rechten Ecke des angezeigten Portlets auf das kleine Zahnrad klicken. Unter „Konfiguration" lassen sich dann die Berechtigungen anpassen *(siehe Abbildung 8)*.

Damit die Berechtigungen auch in der Anwendung Berücksichtigung finden, ist noch die „view.xhtml" anzupassen. Alle Eingabefelder in der „xhtml"-Datei sind innerhalb eines Form-Tag. Dessen „rendered"-Attributs bestimmt, ob die darzustellende HTML-Form inklusive Inhalt angezeigt werden soll oder nicht *(siehe Listing 5)*.

Als „Guest", also als unangemeldeter User, wird nichts angezeigt. Sobald jemand in einer Rolle mit entsprechenden Berechtigungen angemeldet ist, kann er Kalender-Einträge anlegen. Nach der Anlage über das Portlet ist der Eintrag im Liferay-Kalender zu sehen *(siehe Abbildung 9)*.

```
<h:form
rendered="#{liferay.userHasPortletPermission['ADD_ENTRY']}">
  …
</h:form>
```

Listing 5

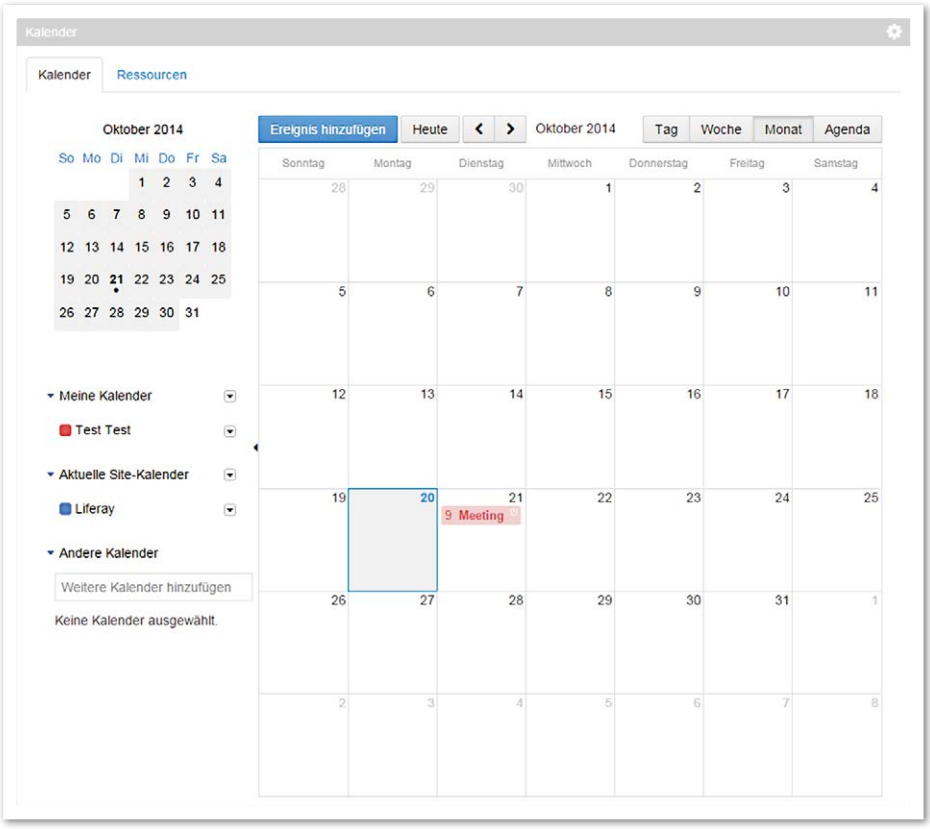


Abbildung 9: Kalender-Portlet mit dem neuen Termin

Fazit

JSF kann auf einfache Weise Portlets für Liferay entwickeln. Die Liferay Faces Bridge liefert alle wichtigen Funktionen, um das Beste aus beiden Welten – Portal-Technologie und Web-Entwicklung – mit JSF zu kombinieren. Seitdem Liferay eine eigene Faces Bridge liefert, stehen dem JSF-Entwickler alle Möglichkeiten der Portlet-Entwicklung einschließlich einfachen Zugriffs auf Liferay-

Core-Funktionalitäten zur Verfügung. Wer allerdings eine Plattform-unabhängige Entwicklung von Portlets anstrebt, sollte bei der Portlet-Entwicklung die zusätzlichen LR-Bridge-Funktionen, die über den Bridge-Standard hinausgehen, mit Bedacht einsetzen. Man kann nur hoffen, dass auch Liferay selbst in den neuen Versionen ihre eigene JSF-Bridge für die Core-Portlets-Entwicklung verwenden wird, auch wenn dieses Vorgehen Veränderungen an den bisherigen Anpassungskonzepten wie Hook oder Ext-Plug-in bedeuten würde.

Weitere Informationen

1. Liferay: http://www.liferay.com/de
2. JavaServer Faces: http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

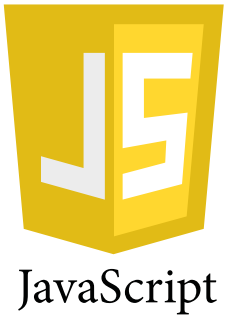
Frank Schlinkheider
fs@itsd-consulting.de

Frank Schlinkheider ist Senior Consultant bei der ITSD Consulting GmbH und dort unter anderem für die Entwicklung von Portal-Lösungen verantwortlich. Neben normalen Liferay-Entwicklungsthemen beschäftigt er sich mit Migrationsmöglichkeiten von vorhandenen IT-Systemen hin zu Enterprise-Portal-Systemen. Darüber hinaus gehört die Umsetzung von BPM-Lösungen (jbpm, Activiti oder Camunda BPM) und die Einführung von Enterprise-2.0-Systemen in Unternehmen und Verwaltungen (etwa mit Social Office) zu seinen Schwerpunkten.

Wilhelm Dück
wilhelm.dueck@itsd-consulting.de

Wilhelm Dück ist bei der ITSD Consulting GmbH als Java-Entwickler tätig und arbeitet seit einem Jahr intensiv mit Liferay in Kombination mit JSF. Neben seiner Tätigkeit als Dozent in Liferay-Schulungen beschäftigt er sich schwerpunktmäßig mit Anpassungen von Liferay-Enterprise-2.0-Systemen.

http://ja.ijug.eu/15/2/5



JavaScript für Java-Entwickler

Niko Köbler, www.javascript-training.net

Ein Workshop im JavaLand 2015 sollte sich kein Java-Entwickler entgehen lassen, wenn er zukunftsfähige (Unternehmens-)Web-Anwendungen entwickeln möchte.

In den letzten zwei Jahren hat sich JavaScript wieder mehr und mehr als ernst zu nehmende Programmiersprache auch in Unternehmensanwendungen durchgesetzt. Sei es im Frontend (Browser), wo sich dank der JavaScript-Frameworks AngularJS, Backbone, Ember etc. die Single-Page-Web-Applications mit HTML5, CSS3 und JavaScript mehr und mehr durchsetzen, oder im Backend auf dem Server, wo sich auch hierzulande „Node.js" mit seinem Ansatz der asynchronen und nicht-blockierenden Request-Verarbeitung („Evented I/O") langsam immer breiter macht. Zusätzlich wird die serverseitige JavaScript-Verwendung dank Nashorn – der JavaScript Engine in der JVM seit Java 8 – immer populärer. Es wird also höchste Zeit für den versierten Java-Entwickler, sich auch mit JavaScript als Programmiersprache auseinanderzusetzen.

Auf den ersten Blick meinen viele Entwickler immer noch, dass JavaScript schon allein aufgrund der Ähnlichkeit im Namen irgendwas mit Java zu tun haben muss. Auch die Sprach-Syntax ist ja sehr ähnlich. Also versuchen sich diese Entwickler mit ersten JavaScript-Programmen und -Anwendungen, ohne aber die Sprache und ihre Eigenschaften genauer kennengelernt und sich mit ihnen beschäftigt zu haben. Dabei ist JavaScript nicht nur aufgrund der Tatsache, dass es eine Script-Sprache ist, der kleine Bruder von Java und nur ein Subset des Sprachumfangs. Die Namens- und Syntax-Ähnlichkeit sind neben dem Grundsatz „Write once, run everywhere" auch die einzigen Gemeinsamkeiten, die beide Sprachen miteinander haben. Doch man kann sie sehr gut zusammen anwenden.

In seinem Workshop im JavaLand 2015 beschäftigt sich der Autor nicht mit der unübersichtlichen Fülle von JavaScript-Frameworks, sondern führt Java-Entwickler in die Sprachgrundlagen von JavaScript ein. Dies kann sehr effizient geschehen, da die meisten Java-Entwickler über ausreichend Vor-

wissen verfügen, das auch für JavaScript in leicht abgeänderter Form wiederverwendet werden kann. So ist beispielsweise das Prinzip der Vererbung jedem bekannt, es muss nur um die Eigenschaften der in JavaScript verwendeten prototypischen (Pseudo-)Vererbung erweitert werden, um die Sprache besser zu verstehen.

Auch auf die Bedeutung von Funktionen wird eingegangen, schließlich sind Funktionen in JavaScript der Dreh- und Angelpunkt der Sprache. Eine Funktion an eine andere Funktion als Parameter übergeben? Kein Problem, schnell hat man den bekannten Callback-Mechanismus implementiert und zur Verfügung. Genauso schnell landet man dadurch aber auch in der ebenso bekannten „Callback-Hölle". Wodurch sich der nächste Themenblock auftut: Sichtbarkeiten, Scopes und Closures. Welchen Wert nimmt das Schlüsselwort „this" wann ein und wie geht man damit um? Welche Gültigkeit haben Variablen? Gibt es sowas wie „public" und „private"? Für Programmierer, die damit noch nicht viel gearbeitet haben, vielleicht etwas verwirrend, aber wenn man sich genauer damit beschäftigt, doch recht einfach zu verstehen.

In der Kommunikation mit REST-Services hat sich mittlerweile das JSON-Format als quasi-Standard durchgesetzt. JSON steht für „JavaScript Object Notation". Wie Objekte damit einfach beschrieben werden können, ist schnell und einfach zu erlernen. Dass es jedoch auch – ähnlich wie bei XML – ein JSON-Schema gibt, mit dessen Hilfe die Objekt-Strukturen nicht nur definiert (und somit kommunizierbar gemacht), sondern Objekte dagegen auch validiert werden können, ist vielen Entwicklern nicht bekannt. Die Teilnehmer erfahren, wie das einfach und effektiv möglich ist.

Oft haben JavaScript-Anwendungen den (unberechtigten) Ruf, sehr schnell in eine unübersichtliche Code-Struktur zu fallen, die nicht mehr refaktorisier- und wartbar sei. Nur, weil eine Sprache nicht beziehungsweise dynamisch typisiert ist, heißt das nicht, dass sie nicht strukturiert ist. Mittels gängiger Pattern und einer Modul-Struktur lässt sich der Programmcode sehr übersichtlich und wartbar gestalten.

Die Weiterentwicklung des Ökosystems (IDEs und andere Werkzeuge) gibt dem JavaScript-Entwickler auch hier mächtige Werkzeuge an die Hand, wie man sie in ähnlicher Form auch aus der Java-Welt gewohnt ist. Der Workshop gibt einen Überblick, welche Werkzeuge es für welche Aufgaben gibt und wie man sie anwendet – von Modul-Systemen über Code-Analysen (logischer Syntax-Check, Erfüllung von Qualitäts-Vorgaben) bis hin zum Build und Deployment (Minimizing, Uglifying, Packaging).

Wem am Ende JavaScript immer noch zu unsicher, weil untypisiert, ist, dem zeigt der Autor, wie man mit Typescript statisch typisiertes JavaScript einschließlich Klassen schreiben und zu nativem JavaScript transpilieren kann. Typescript ist ein Superset von JavaScript und stellt bereits jetzt Sprach-Features aus dem kommenden Standard ECMAScript 6 (momentan ist ES 5.1 aktuell) wie auch eigene Features zur Verfügung. So ist die (statische) Typisierung von Variablen zwar Typescript-spezifisch, jedoch wird es die Möglichkeit der JavaScript-Klassen auch direkt in ES 6 geben.

Sollte am Ende des Workshops noch ein wenig Zeit bleiben, gibt der Autor gerne eine kurze Einführung in Nashorn, um JavaScript auch auf der JVM ausführen zu können und nicht nur auf Browser und Node.js/V8 beschränkt zu sein.

Das alles hört sich nach sehr viel Stoff für nur einen Tag an. Jedoch sind viele Sprach-Eigenschaften sehr schnell erklärt und werden anhand von Beispielen verdeutlicht. Zum Üben des neu erlernten Stoffes führen alle Teilnehmer ebenfalls immer wieder kleine Beispiele selbst durch, um das theoretische Wissen auch praktisch zu verankern.

Niko Köbler
info@javascript-training.net

Niko Köbler ist freiberuflicher Software-Architekt, Developer und Coach für (Java-)Enterprise-Lösungen, Integrationen und Web-Development. Er ist Mitbegründer der Qualitects Group, berät und unterstützt Kunden verschiedenster Branchen, hält Workshops und Trainings und führt Architektur-Reviews durch. Neben seiner Arbeit bei der Java User Group Darmstadt (JUG DA) schreibt er Artikel für Fachzeitschriften und ist regelmäßig als Sprecher auf Fachkonferenzen anzutreffen.

http://ja.ijug.eu/15/2/6

Java 9 mit neuen JEPs

Im Sommer waren die ersten JDK Enhancement Proposals (JEPs) bekannt geworden. Inzwischen hat Oracle auf der OpenJDK-Website weitere Neuerungen angekündigt. Nach wie vor bleibt das große Thema die Modularisierung, die in das Projekt Jigsaw einfließt. Die neuen JEPs betreffen andere Bereiche.

Ein einheitliches Logging-System für alle JVM-Komponenten (Java Virtual Machine) steht mit dem JEP 158 an. Die Steuerung Optionen der JVM-Compiler soll zur Laufzeit ermöglicht werden. Diese Verbesserung bringt JEP 165 mit sich.

Mit Java 7 waren im Rahmen des JSR 334 „Project Coins" kleine Sprach-Features eingeführt worden. Für Java 9 ist es an der Zeit, Anpassungen von vier Aspekten im JEP 213 vorzunehmen. Eine weitere Neuerung gibt es mit dem JEP 214: Die Garbage-Collection-Kombinationen, die mit dem JDK 8 als „deprecated" galten, sollen nun entfernt werden. Ein API für „Datagram Transport Layer Security (DTLS)" in der Version 1.0 ist mit dem JEP 219 angedacht. Darüber hinaus soll in das Javadoc-Tool, eine Option implementiert werden. Derzeit generiert das Tool HTML-Seiten in der Version 4.01. In Zukunft soll mit dem JEP 224 per Kommando-Zeile auch die Möglichkeit gegeben werden, HTML5 zu nutzen.

Besser Java programmieren mit Xtend

Moritz Eysholdt, itemis AG

Gelegentlich kann Java etwas umständlich sein. Wer diesen Gedanken schon einmal hatte, findet in diesem Artikel mit Xtend eine Ergänzung oder sogar Alternative zu Java. In Xtend geschriebener Code schafft es meist, kompakter und eleganter als sein Java-Äquivalent zu sein.

Geschickte Entwickler können mit Xtend nicht nur schneller, sondern auch lesbareren Code schreiben. Da der Xtend-Compiler Java-Code erzeugt, integriert sich dieser problemlos in jedes Java-Projekt. Xtend selbst ist statisch typisiert, Open Source und es ist vorzügliche Werkzeugunterstützung vorhanden.

Wer mit Java programmiert, muss sehr korrekt arbeiten. Zeilen müssen mit ";" beendet werden, Deklarationen von Variablen und Methoden benötigen Typangaben und, da Java keine Properties kennt, finden wir die "get"- und "set"-Prefixe bei unzähligen Methodenaufrufen. Auch die Deklaration der Getter und Setter kann man schwer dem Java-Compiler überlassen, da hierzu fundierte Kennnisse über Annotation Processing und Javas Bytecode notwendig wäre.

Man hilft sich mit IDEs, die Teile des lästigen Boilerplate-Codes per Wizard-Dialog generieren. Aber ist das eine gute Lösung? Mehr Lines Of Code machen eine Anwendung mit Sicherheit nicht wertvoller. Wer die gleichen Use Cases mit weniger Codezeilen implementieren kann, hat im Nachhinein weniger Aufwand, diese zu warten und besitzt mehr Agilität bei Refactorings.

Auf die Lesbarkeit kann sich eine verminderte Anzahl von Codezeilen positiv auswirken, da kompakter Code eher dazu tendiert die Absicht des Autors zu kommunizieren. Nebensächliche Implementierungsdetails werden in den Compiler, Bibliotheken und Frameworks ausgelagert.

Andererseits bietet das Vorhandensein eines statischen Typsystems in Java ein nicht zu unterschätzender Vorteil: Entwicklungsumgebungen wie Eclipse oder IntelliJ sind in der Lage, verlässliche Code-Vervollständigung, Validierung, Rename-Refacorings und vieles mehr zu bieten. Für dynamisch typisierte Sprachen wie Groovy ist dies konzeptbedingt nicht möglich, da zur Entwicklungszeit nicht vollständig bekannt ist, welche Felder und Methoden tatsächlich auf einem Objekt aufrufbar sind. Die am Anfang dieses Artikels beklagte Geschwätzigkeit von Java als Anlass zu nehmen, zu einer dynamisch typisierten Sprache zu wechseln, würde also die mächtige Entwicklungsumgebung zu einem bloßen Texteditor degradieren.

Besser programmieren

Xtend bietet die syntaktische Kompaktheit einer dynamisch typisierten Sprache, ist jedoch statisch typisiert. Darüber hinaus bringt es eine Vielzahl an Konzepten mit, die sich im täglichen Programmieralltag als überaus nützlich erweisen. Im Gegensatz zu Scala erfindet Xtend kein eigenes Typsystem, sondern verwendet das von Java bekannte. Damit bleibt die Einstiegshürde niedrig und eine harmonische Koexistenz von Java- und Xtend-Code im selben Projekt ist gewährleistet *(siehe Abbildung 1)*.

Dieser Artikel kann aus Platzgründen nicht alle Konzepte vorstellen, sondern beschränkt sich auf die im Beispiel erforderlichen. Für alle weiteren Konzepte sei auf die Webseite und Dokumentation verwiesen *(siehe „http://xtend-lang.org")*. Das Beispiel liegt in zweifacher Implementierung vor: Einmal in Xtend und einmal in Java. Es werden vier Klassen definiert: „SmartHome", „Room", „SmartLight" und „SmartHomeTest". In „SmartHome#getSmartLights()" werden drei SmartLight-Objekte instantiiert, Räumen zugeordnet und als unsortierte Menge zurückgegeben.

In „SmartHomeTest" werden die „SmartLights" abgefragt, in String-Repräsentationen umgewandelt und sortiert zeilenweise in einem String abgelegt. Diesen String vergleicht der Test mittels Assert mit einer Erwartungshaltung. Strings anstelle von Listen, Sets oder gar Objektgraphen miteinander zu vergleichen, hat den großen Vorteil, dass im Falle eines fehlschlagenden Tests der Stringvergleich hervorragend die Abweichungen zwischen Erwartung und tatsächlichem Ergebnis illustriert. Eclipse stellt zu diesem Zweck einen Diff-Dialog bereit.

Syntaktischer Vergleich mit Java

Xtend versucht, die syntaktischen Differenzen zu Java gering zu halten, um Überraschungen zu vermeiden. Die wichtigsten Unterschiede sind:

- Methoden werden mit „def" deklariert
- lokale Variablen werden mit „val" oder „var" deklariert. „val" entspricht einer „final"-Variable in Java.
- Semikolons am Anweisungsende sind optional
- Runde Klammern für parameterlose Methodenaufrufe sind optional
- „return"-Schlüsselwörter sind optional. Bei weggelassenem „return" wird der Wert des letzten Ausdrucks im Block zurückgegeben.
- „Getter" und „Setter" wie können wie Felder verwendet werden. Anstelle von „setID("couch")" und „int myID = getID()" ist also „id = "couch"" und „val myID = id" erlaubt.

- Typ-Definitionen sind optional, sofern sie vom Compiler inferiert werden können. Dies funktioniert beispielsweise bei lokalen Variablen und den Rückgabe-Typen von Methoden. Inferieren bedeutet, dass der Compiler die Typen während des Compilierens aus dem Context, beispielsweise der Initialisierung einer Variablen, ableitet. Es sind also immer Typen vorhanden. Dies ist nicht zu verwechseln mit optionaler Typisierung wie sie Google Dart unterstützt. Dort forcieren weggelassene Typen eine dynamische Typisierung.
- Für den Fall, dass die syntaktischen Vereinfachungen durch Xtend Mehrdeutigkeit verursachen, setzt sich die nicht vereinfachte Variante durch.

Lambda-Ausdrücke und funktionale Programmierung

Lambda-Ausdrücke haben in Xtend eine leicht andere Syntax als in Java 8:

- *Xtend*
  „map[light | light.getID()]"
- *Java8*
  „map(light -> light.getID())"

SmartHome.xtend

```
1  package smarthome.xtend
2
3  import org.eclipse.xtend.lib.annotations.Accessors
4  import org.eclipse.xtend.lib.annotations.FinalFieldsConstructor
5  import org.junit.Test
6
7  import static org.junit.Assert.*
8
9  class SmartHome {
10   def getSmartLights() {
11     val livingRoom = new Room("LivingRoom")
12     val kitchen = new Room("Kitchen")
13     return #{
14       new SmartLight => [room = livingRoom id = "couch"],
15       new SmartLight => [room = livingRoom id = "dining"],
16       new SmartLight => [room = kitchen    id = "stove"]
17     }
18   }
19 }
20
21 @Accessors class SmartLight {
22   String id
23   Room room
24 }
25
26 @Accessors @FinalFieldsConstructor class Room {
27   val String id
28 }
29
30 class SmartHomeTest {
31   @Test def void testLights() {
32     val home = new SmartHome
33     val lights = home.smartLights
34     val actual = lights.map[id + "-" + room.id + "\n"].sort.join
35     val expected = '''
36       couch-LivingRoom
37       dining-LivingRoom
38       stove-Kitchen
39     '''
40     assertEquals(expected, actual)
41   }
42 }
```

SmartHome.java

```
1  package smarthome.java;
2
3  import static org.junit.Assert.assertEquals;
4
5  import java.util.HashSet;
6  import java.util.Set;
7  import java.util.stream.Collectors;
8
9  import org.junit.Test;
10
11 public class SmartHome {
12   public Set<SmartLight> getSmartLights() {
13     Room livingRoom = new Room("LivingRoom");
14     Room kitchen = new Room("Kitchen");
15     SmartLight couch = new SmartLight();
16     couch.setId("couch");
17     couch.setRoom(livingRoom);
18     SmartLight dining = new SmartLight();
19     dining.setId("dining");
20     dining.setRoom(livingRoom);
21     SmartLight stove = new SmartLight();
22     stove.setId("stove");
23     stove.setRoom(kitchen);
24     Set<SmartLight> result = new HashSet<>();
25     result.add(couch);
26     result.add(dining);
27     result.add(stove);
28     return result;
29   }
30 }
31
32 /* public */ class SmartLight {
33   private String id;
34   private Room room;
35   public String getId()          { return this.id; }
36   public void setId(String id)   { this.id = id; }
37   public Room getRoom()          { return this.room; }
38   public void setRoom(Room room) { this.room = room; }
39 }
40
41 /* public */ class Room {
42   private final String id;
43   public Room(final String id)   { this.id = id; }
44   public String getId()          { return this.id; }
45 }
46
47 /* public */ class SmartHomeTest {
48   @Test public void testLights() {
49     SmartHome home = new SmartHome();
50     Set<SmartLight> lights = home.getSmartLights();
51     String actual = lights.stream()
52         .map(l ->
53             l.getId() + "-" +
54             l.getRoom().getId() + "\n"
55         )
56         .sorted()
57         .collect(Collectors.joining());
58     StringBuilder expected = new StringBuilder();
59     expected.append("couch-LivingRoom\n");
60     expected.append("dining-LivingRoom\n");
61     expected.append("stove-Kitchen\n");
62     assertEquals(expected.toString(), actual);
63   }
64 }
```

Abbildung 1: Das Beispiel

Im Gegensatz zu Java kann bei Xtend zusätzlich die Parameter-Deklaration weggelassen werden. Dann kann über die implizite Variable „it" auf den Parameter zugegriffen werden. Member auf „it" sind auch unqualifiziert aufrufbar, so wie es bei Java von „this" bekannt ist. Der Ausdruck kann also weiter vereinfacht werden:

- „map[it.getID()]"
- „map[getID()]"
- „map[id]"

Wichtig zum Verständnis ist, dass die eckigen Klammern ein Lambda-Literal darstellen und kein Ersatz für die runden Klammern sind, die üblicherweise Methoden-Parameter umschließen. Die Abwesenheit der runden Klammern ist eine syntaktische Vereinfachung: Die volle Schreibweise ist „map([id])". Wenn der letzte Methodenparameter eine Lambda ist, darf diese jedoch auch nach den runden Klammern als „map()[id]" deklariert sein. Da die runden Klammern jetzt keine Parameter mehr umschließen, dürfen sie weggelassen werden.

Da die älteste von Xtend unterstützte Zielplattform Java 5 ist, lassen sich mit Xtend Lambda-Ausdrücke auch in Umgebungen verwenden, in denen kein Java 8 eingesetzt werden kann. Ein solches Szenario ist die App-Entwicklung für Android-Systeme.

Ein weiteres nützliches Feature zur funktionalen Programmierung, das Java 8 in diesem Falle nicht beherrscht, ist die Tatsache, dass es in Xtend keine Statements, sondern ausschließliche Expressions gibt. Expressions können im Gegensatz zu Statements Rückgabewerte haben. Eine elegante Alternative zu „var x; if (true) x =1 else x = 2;2 ist beispielsweise „val x = if(true) 1 else 2". Diese Schreibweise ist förderlich zur funktionalen Programmierung, da „x" plötzlich immutable sein kann.

In Java nicht vorhandene Konzepte

Das Beispiel in *Abbildung 1* verwendet einige in Java nicht vorhandene Konzepte:

- *Der „with"-Operator, „=>"*
  Auf der linken Seiten erwartet dieser ein Objekt und auf der rechten Seite eine Lambda, um das Objekt von der linken Seite als Parameter hereingereicht zu bekommen. Was banal klingt, ist in der Praxis überaus nützlich, um Objekte zu initialisieren ohne sie erst lokalen Variablen zuweisen zu müssen.
- *List und Set-Literale*
  „#[item1, itemi2]" instantiiert eine „java.util.List" und „#{item1, item2}" erstellt ein „java.util.Set".
- *Multi-Line-String-Literale*
  Diese sind vollwertige Template Expressions. Der String wird durch dreifache Einfach-Anführungszeichen eingeschlossen. Die Template Expressions können „FOR"- und „IF"-Statements enthalten und beherrschen „smart whitespace handling". Template-Sprachen haben traditionell das Problem, dass ein verständlich indentiertes Template unverständlich indentierten Ausgabetext erzeugt. Wer im Template so indentiert, dass die Ausgabe verständlich ist, macht meist das Template unleserlich. Der traditionelle Workaround ist es, einen Formatter über die erzeugte Ausgabe laufen zu lassen. „Smart Whitespace Hand-

ling" löst das Problem, indem im Template zwischen Ausgabe-Whitespace (Im Editor grau hinterlegt) und Formatting-Whitespace (im Editor weiß hinterlegt) unterschieden wird. Der Template-Autor erlangt also wieder Kontrolle über die Formatierung der Template-Ausgabe.

Extension Methods

Traditionell lassen sich in Java nur Methoden auf einem Objekt aufrufen, die dessen Klasse oder eine ihrer Superklassen deklariert. Für Programmierer kann dies sehr einschränkend sein, wenn die Klassen aus einer Bibliothek oder einem Framework kommen. Wäre es nicht praktisch, wenn es beispielsweise „java.io.File.listFilesRecursively()" gäbe?

Natürlich kann man sich sein eigenes „FileUtil.listFilesRecursively(File file)" bauen, aber dann müsste man „listFilesRecursively(file)" anstelle von „file.listFilesRecursively()" aufrufen. Mit nur einer Methode mag diese Unterscheidung pedantisch wirken, aber bei verketteten Methodenaufrufen wird der Unterschied signifikant.

In *Abbildung 1* wird „lights.map[id].sort.join" aufgerufen. Die Methoden „map", „sort" und „join" sind Extension-Methoden. Daher lässt sich „map" auf „lights" aufrufen, das vom Typ „java.util.Set" ist. Ohne Extension-Methoden sähe der Ausdruck so aus: „join(sort(map(lights, [id])))". Abgesehen von dem entstandenen Wald an Klammern fällt auf, dass man, um den Datenfluss durch die Methoden zu verfolgen, von rechts nach links lesen muss, also in der Reihenfolge „map", „sort", „join". Für Personen, die mit einer westlichen Muttersprache aufgewachsen sind, kann dies als überaus unintuitiv und umständlich bezeichnet werden. In Xtend kann jede Java- oder Xtend-Methode als Extension Methode verwendet werden, vorausgesetzt sie akzeptiert das Objekt, auf dem sie aufgerufen wird, als ersten Parameter. Zusätzlich müssen ihre Klassen (für statische Methoden) oder eine Instanz (für nicht-statische Methoden) als Extension-Provider importiert werden. Dies geschieht via „import static extension mypackage.MyClass.*" oder durch Platzierung des extension Schlüsselworts vor Feld-, Parameter-, oder Variablen-Deklarationen.

Active Annotations

Active Annotations sind in Xtend oder Java deklarierte Annotations, die mit einem Annotation-Prozessor versehen wurden. Der Prozessor wird vom Xtend-Compiler ausgeführt und kann den vom Compiler erzeugten Java-Quellcode beeinflussen. Im Beispiel werden die „Active Annotations @Accessors" und „@FinalFieldsConstructor" verwendet. Erstere erzeugt „Getter"- und „Setter"-Methoden für alle Felder in der annotierten Klasse und letztere erzeugt einen Konstruktor, welcher für jedes finale Feld der Klassen einen Parameter entgegen nimmt.

Active Annotations können beliebig Methoden und Felder hinzufügen oder löschen sowie Klassen, Interfaces oder Dateien erzeugen. Ebenso kann ein Prozessor Fehlermeldungen und Warnungen für Quellcode-Elemente erzeugen. Den Prozessor für eine „Active Annotation" zu implementieren ist in etwa so aufwändig wie einen kleinen Code-Generator zu schreiben. Hier zahlt es sich als großer Vorteil aus, dass der Xtend-Compiler Java-Quellcode erzeugt. Als Implementierer einer „Active Annotation" muss man lediglich mit Java-Quellcode vertraut sein. Beim Java-eigenen Mechanismus zum „AnnotationProcessing" ist ein tiefes Verständnis von Java- Bytecode unabdingbar. Typische Anwendungsfälle für Active Annotations sind:

- *Automatisierung von Java Design Patterns*
  Observer, Delegate, Singleton etc.
- *Automatisierung von durch Frameworks vorgesehene Patterns*
  Logging, JavaFX-Properties etc.
- *Automatisches Ableiten von XML- oder Properties-Dateien aus Java-Quellcode*
  Beispielsweise zwecks Internationalisierung oder zur Konfiguration eines verwendeten Frameworks
- *Automatisches Erzeugen einer statischen API zum Zugriff auf nicht in Java deklarierte Datenstrukturen*
  Die Beispiele reichen von der Ressourcen-XML auf Android-Systemen über

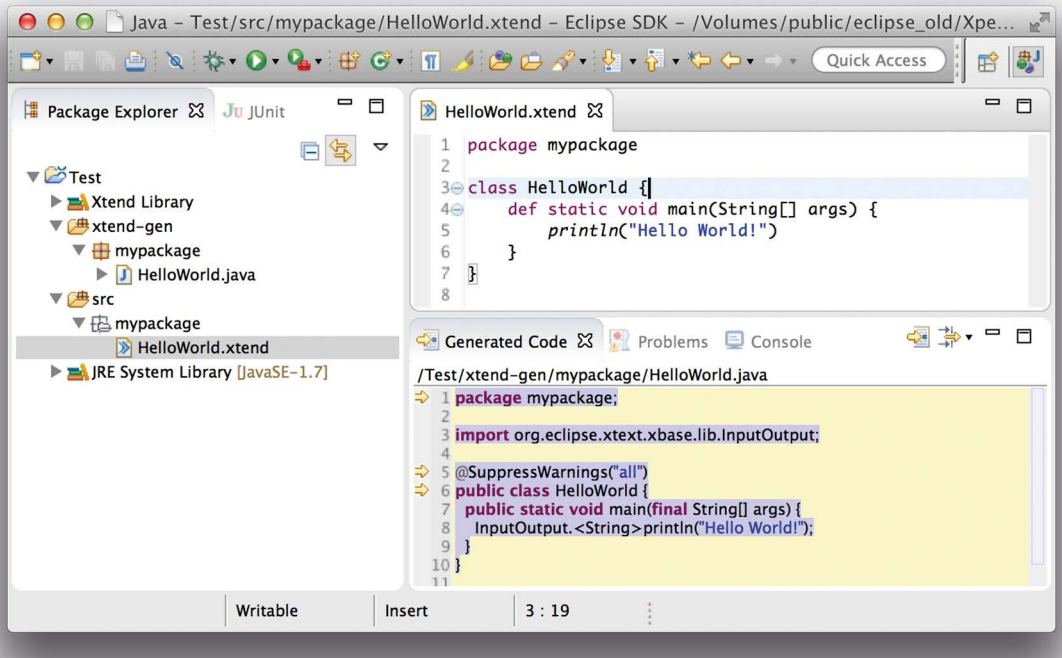


Abbildung 2: „Hello World!"-Beispiel und „Generated Code"-View

Entitäten von SQL oder Graph-Datenbanken bis hin zu APIs für Zugriff auf XML-Dateien, JSON-Daten oder REST-Schnittstellen. Ein statisches API steht im Gegensatz zu einem generischen API: Beispielsweise könnte eine Methode, um den Namen eines Elements abzufragen, in einem statischen API so aussehen: „String getName()". In einem generischen API muss jedoch die Methode „Object getValue("name")" aufgerufen werden. Generische APIs haben zwar den Vorteil, dass dem API-Designer die tatsächliche Struktur der Daten nicht bekannt sein muss, aber auch den Nachteil, dass durch nicht-vorhandene Attribute oder inkompatible Datenstrukturen verursachte Fehler erst zu Laufzeit bemerkt werden können. Mit einem statischen API kann diese Art von Fehler zuverlässig als Compilefehler erkannt werden.

Da der Xtend-Compiler in der IDE integriert ist, sind von Active Annotations erzeugte Typen, Felder und Methoden für alle Werkzeuge (Content Assist, Outline, Find References, Type Hierarchy, etc.) genau so sichtbar, als wären sie handgeschrieben.

Integration und Debugging

Seit Dezember 2014 gibt es ein Xtend-Plug-in für Eclipse, ein Plug-in für IntelliJ ist in Arbeit. Der Compiler kann über ein Maven-, ein Groovy-Plug-in oder direkt über Java aufgerufen werden. In Xtend beschriebene JUnit-Tests können von den üblichen, für Java vorgesehenen Werkzeugen, ausgeführt werden.

Xtend-Code ist als Xtend-Code überwachbar. Im Xtend-Code gesetzte Breakpoints werden wie erwartet vom Debugger beachtet. Ebenso funktioniert die Anzeige der Variablen und Felder inklusive Wertebelegung. Auch das Stepping arbeitet wie gewohnt. Darüber hinaus lässt sich während des Debuggens frei zwischen dem Xtend- und dem generierten Java-Code hin und her wechseln.

Getting Started

In Eclipse installiert man sich zunächst das Xtend-Plug-in oder lädt die „Eclipse IDE for Java and DSL Developers" herunter, die das Xtend-Plug-in bereits enthält. Anschließend lässt sich in der Java-Perspektive in einem neuen oder bestehenden Java-Projekt über „File" -> „New" -> „Xtend Class" eine neue Xtend-Class erzeugen. Falls die kleine Bibliothek „org.eclipse.xbase" noch nicht auf dem Klassenpfad ist, enthält die neu erzeugte Klasse einen Fehler und ein Quickfix bietet an, die Bibliothek dem Klassenpfad hinzuzufügen. Die im Beispiel verwendeten Methoden „map", „sort" und „join" stammen aus dieser Bibliothek. Wer jetzt eine statische main-Methode deklariert, kann wie gewohnt sein „hello world" via „Run As" -> „Java Application" ausführen *(siehe Abbildung 2)*.

Zum weiteren Verständnis von Xtend bietet es sich an, gelegentlich einen Blick in den „Generated Code"-View zu werfen. Dieser lässt sich über „Window" -> „Show View" -> „Other..." -> „Xtend" -> „Generated Code" öffnen. Er zeigt für im Editor selektierten Xtend-Code den erzeugten Java-Code an. Neben der Dokumentation ist der „Generated Code"-View eine pragmatische Möglichkeit, Verständnisfragen zu Xtend-Code schnell beantwortet zu bekommen.

Community

Zur Diskussion von Fragen gibt es bei Google Groups die Gruppe „Xtend Programming Language". Darüber hinaus gibt es auf der Xtend-Webseite *(siehe „http://xtend-lang.org")* neben Dokumentation eine stetig aktualisierte Liste von Artikeln, die Xtend zum Inhalt haben. Neuigkeiten werden üblicherweise über Twitter („@xtendlang") und „Google+ (+Xtend-langOrg)" verkündet.

Fazit

Xtend ist eine statisch typisierte Programmiersprache, die durch syntaktische Freiheiten, Typinferenz, pragmatische Konzepte und mächtige Automatisierungsmöglichkeiten kompakten Quellcode ermöglicht. Kompakter Code erlaubt es geübten Softwareentwicklern schneller Ergebnisse zu erzielen, besser ihre Intention zu transportieren sowie zukünftigen Wartungsaufwand gering zu halten. Da das Xtend-Typ-System dem von Java entspricht, integriert sich Xtend nahtlos mit bestehenden Java-Projekten.

Moritz Eysholdt
moritz.eysholdt@itemis.de

Moritz Eysholdt ist leidenschaftlicher Software-Entwickler und arbeitet für die itemis AG am Standort Kiel. Dort ist er an der Entwicklung von Xtext und Xtend beteiligt. Er ist Initiator des Xpect-Projekts, um das Testen von Xtext-DSLs zu vereinfachen. Man trifft ihn gelegentlich als Speaker auf Konferenzen oder beim Joggen.

http://ja.ijug.eu/15/2/7

Stuttgarter Test-Tage am 16. und 17. April 2015

"If you want more effective programmers, you will discover that they should not waste their time debugging, they should not introduce the bugs to start with." (Edsger Wybe Dijkstra)

Welche Alternativen zum „Programming by Debugging" gibt es denn? Diesen und anderen Fragen gehen die Telnehmer der Stuttgarter Test-Tage nach. Dabei liegt der Fokus nicht nur auf Tests, sondern auf all den Dingen, die zum Testen dazugehören: Tools, Bibliotheken, Prozesse oder moralische Unterstützung. Dabei soll neben dem theoretischen Grundwissen die Praxis nicht zu kurz kommen.

Weitere Informationen unter *http://www.jugs.de*

JUGS
java user group stuttgart


Java
C#

Ich liebe Java und ich liebe C#

Rolf Borst, Datenzentrale Baden-Württemberg

Ein Artikel über C# in einem Java-Magazin? Das geht doch gar nicht – oder vielleicht doch? Der Autor ist ein sehr großer Fan von Java, aber auch von C#. Dieser Artikel stellt einige interessante Seiten von C# vor.

Das besonders Tolle an Java und C# ist, dass sich beide Sprachen in ihrer Grund-Syntax sehr ähnlich sind. Das Beispiel in *Listing 1* lässt sich mit einer kleinen Änderung bei „Length" sowohl in Java als auch in C# compilieren. Trotzdem steckt bereits in diesem kleinen Beispiel sehr viel drin: Klassen-, Methoden- und Variablen-Definitionen, Schleifen, Abfragen, Kommentare etc. – alles ist sehr ähnlich oder sogar gleich.

Aber für Entwickler wäre es natürlich nicht interessant, wenn die Sprachen genau identisch wären. Die kleinen Unterschiede und Ideen machen eine Sprache aus. Hier gibt es für einen Java-Entwickler auch in C# viel zu entdecken. Dieser Artikel stellt einige dieser Ideen und Sprach-Features in C# vor.

Partial Classes

In Java ist man es gewohnt, eine Klasse immer genau in einer Datei zu speichern und die Package-Struktur über die Verzeichnisstruktur abzubilden. In C# gibt es keinen direkten Zusammenhang von Klassen und Dateinamen beziehungsweise Verzeichnis-Strukturen. Dateien können hier einen beliebigen Namen besitzen und auch mehrere öffentliche Klassen enthalten.

Eine Besonderheit sind die „Partial Classes". Damit ist es möglich, eine einzelne Klasse auf mehrere Dateien aufzuteilen. Dafür wird die reine Klassen-Definition mit dem zusätzlichen Modifier „partial" in jede Datei aufgenommen. Die Member-Variablen und Methoden können dann beliebig auf die einzelnen Dateien verteilt sein *(siehe Listing 2)*. Der Compiler mixt später die Definitionen aus allen Dateien zu einer einzigen compilierten Klasse zusammen. Eine Methode in Datei 1 kann dabei immer auch die Methoden und Definitionen aus Datei 2 verwenden und umgekehrt.

Dieses Feature ist vor allem dann interessant, wenn man einzelne Teile der Klasse über ein Generierungstool erzeugen möchte und andere Teile der Klasse von Hand implementiert werden sollen (MDA, GUI-Generatoren). In diesem Fall würde der Generator eine Datei erstellen und zusätzliche Implementierungen erfolgen in einer separaten zweiten Datei. Der Generator kann dadurch seine Datei immer wieder vollständig überschreiben, ohne auf zusätzliche Implementierungen Rücksicht nehmen zu müssen.

Die Generierung von Teilen der Klasse ist eine der wichtigsten Anwendungsgebiete für die „Partial Classes", dennoch lassen sich diese auch für andere Zwecke nutzen. So können beispielsweise Depracted-Methoden bis zur abschließenden Lösung in eine zweite Datei ausgelagert werden. Der Hauptsource kann sich damit vollständig auf die neue Implementierung konzentrieren und bleibt übersichtlich.

Eine weitere Anwendungsmöglichkeit ist der Änderungsschutz bei sensiblen Klassen. Die kritischen Methoden einer Klasse können über „Partial Classes" in eine separate Datei ausgelagert und über die Quelltextverwaltung besonders geschützt werden. Änderungen an dieser Datei dürfen dann beispielsweise nur ganz bestimmte Entwickler durchführen. Unkritische Erweiterungen der Klasse können in der zweiten, öffentlichen Datei von allen Entwicklern vorgenommen werden. Die Möglichkeit zur Aufteilung von einzelnen Klassen auf mehrere Dateien bietet darüber hinaus Potenzial für viele weitere Ideen.

Extension Methods

Java und C# sind im Kern objektorientierte Programmiersprachen. Trotzdem werden Operationen oftmals nicht direkt auf den Objekten ausgeführt, sondern die Objekte als Parameter an statische Methoden oder an Methoden in anderen Klassen übergeben. Soll beispielsweise ein Text in einem String-Objekt verschlüsselt werden, so ist dies nicht direkt mit einer Methode des String-Objekts möglich. Sowohl in Java als auch in C# kennt der Typ String keine Verschlüsselungsmethode und eine Ableitung der Klasse ist durch den final-Charakter der Klasse ebenfalls in beiden Sprachen nicht möglich. Eine mögliche Lösung könnte eine statische Helper-Methode *(siehe Listing 3)* sein. Dies funktioniert sowohl in Java als auch in C#.

Der C#-Source enthält aber eine Besonderheit: das Schlüsselwort „this" am Anfang

```
public class Einfuehrung {

   // Zähler für die Anzahl der Anfragen
   private int anzahlAnfragen = 0;

   /* Anzahl der geraden und durch drei teilbaren Zahlen */
   public String GetAnzahl(String praefix, int[] zahlen) {
       int anzahl = 0;
       for (int i = 0; i < zahlen.Length; i++) {
           if (zahlen[i] % 2 == 0 && zahlen[i] % 3 == 0) {
               anzahl++;
           }
       }
       anzahlAnfragen++;
       return praefix + ": " + anzahl;
   }

}
```

Listing 1: Ähnlichkeit von Java und C#

der Parameter-Liste. Es macht diese statische Methode zu einer „Extension Method" für den Typ, der hinter dem Schlüsselwort „this" folgt (in diesem Fall String). Die Methode wird damit automatisch als Erweiterung an den entsprechenden Typ angehängt und kann direkt auf den Objekten des Typs in der normalen Punkt-Syntax aufgerufen werden. Der „this"-Parameter selbst muss dabei nicht mehr angegeben werden, da das Objekt selbst sich in diesem Parameter an die Methode übergibt *(siehe Listing 4)*. Im Editor werden „Extension Methods", wie normale Methoden auch, direkt nach der Eingabe des Punkts in der Code-Vervollständigung angezeigt.

Als weiteres Beispiel enthält *Listing 5* die Methode „IsSet", mit der bei einem String überprüft werden kann, ob der String einen Wert enthält (mindestens ein Zeichen und nicht „null"). Diese Methode lässt sich sehr elegant mit „name.IsSet()" aufrufen. Hier stellt sich natürlich sofort die Frage: „Stopp, tritt hier nicht eine „NullpointerException" auf, wenn der Objekt-Name „null" ist?".

Hier zeigt sich eine Besonderheit der „Extension Methods". Diese können auch auf „null"-Objekte aufgerufen werden, weil der scheinbare Aufruf einer Instanz-Methode in Wahrheit dem Aufruf einer statischen Methode entspricht. Ob der erste Parameter der Methode dann „null" ist, kann sofort in der Methode abgefangen und entsprechend behandelt werden. Daher funktioniert der Aufruf von „name.IsSet()" auch dann, wenn name den Wert „null" enthält.

Unterschiedliche Parameter-Signaturen implementieren die „IsSet"-Methode in einer einheitlichen Form für die unterschiedlichsten Klassentypen und primitive Datentypen (Listen, Arrays, double etc.). Der Aufruf ist immer gleich und dennoch kann jeder Typ seine ganz spezielle Implementierung besitzen, die entscheidet, ob das Objekt „gesetzt" oder „leer" ist.

```
Datei1.cs:
public partial class Beispiel {
   private int variable1 = 0;
   public void Methode1() { }
}

Datei2.cs:
public partial class Beispiel {
   private int variable2 = 0;
   public void Methode2() { }
   public void Methode3() {
}
```

Listing 2: Aufteilung von Klassen auf mehrere Dateien mit Partial Classes

```
public static class SecurityUtils {

   public static String Verschluesseln(this String text, String pass-
wort) {
      // Verschlüsselung mit Framework
      return ...;
   }

}
```

Listing 3: Extension Method für den Typ String

```
String text = "Streng geheimer Text";
String geheim = text.Verschluesseln("Passwort");
```

Listing 4: Aufruf einer Extension Method

```
public static bool IsSet(this String value) {
   return !String.IsNullOrEmpty(value);
}
```

Listing 5: Extension Method IsSet für Strings

Language Integrated Query

Extension Methods sind zusammen mit den Lambda-Expressions einer der Grundbausteine von Language Integrated Query (LINQ). Damit können Abfragen auf die unterschiedlichsten Datenspeicher wie beispielsweise Datenbanken, Collections, XML-Strukturen oder Web-Quellen direkt in den Sourcecode integriert werden. Lambda Expressions und flexible Abfragen auf Collections sind auch in Java 8 möglich (Streams). C# geht hier aber noch einen Schritt weiter, indem die Abfragen nicht nur eine Verkettung von Methodenaufrufen darstellen, sondern die Abfragemöglichkeit direkt in die Syntax der Sprache integriert wurde. Die Sprache selbst kennt damit Begriffe wie „from", „where", „orderby" oder „select" *(siehe Listing 6)*.

Die Abfrage bekommt der entsprechende LINQ-Provider in Form eines Objektbaums (Expression Tree) übergeben und kann die Auswertungswünsche dann direkt in die SQL-Abfrage oder in den Aufruf eines Web Services einbauen.

Seit dem Erscheinen von LINQ sind eine Vielzahl von LINQ-Providern für die unterschiedlichsten Datenquellen und Web-Dienste entstanden. Durch die Implementierung der entsprechenden Interfaces können auch eigene LINQ-Provider erzeugt werden.

LINQ wurde 2007 mit C# 3.0 eingeführt und hat zu den vielleicht umfangreichsten Erweiterungen der Sprache seit der Einführung von C# geführt. Hier gibt es noch viele weitere Features wie „Implicitly Typed Local Variables", „Object Initializers", „Auto-Implemented Properties" oder „Anonymous Types" zu entdecken.

Asynchronous Methods

Die asynchrone Verarbeitung von Daten ist für viele Anwendungsgebiete wichtig. So wird ein Blockieren der Anwendung beispielsweise bei mobilen Geräten von den Anwendern heute einfach nicht mehr akzeptiert. Benutzeroberflächen müssen auch beim Laden großer Datenmengen oder umfangreichen Verarbeitungen weiterhin benutzbar bleiben. Die dafür notwendige Hintergrund-Verarbeitung in separaten Threads und die anschließende Synchronisierung von

Threads ist auch heute noch schwierig und fehleranfällig.

Um den Entwicklern das Leben hier etwas einfacher zu machen, wurden in C# 5.0 die „Asynchronous Methods" eingeführt. *Listing 7* zeigt ein kleines Beispiel dazu: Beim Klick auf einen Button in der Oberfläche soll eine bestimmte Web-Seite geladen und anschließend die Größe dieser Web-Seite angezeigt werden. Das Laden der Web-Seite soll dabei im Hintergrund erfolgen und die Anwendung nicht blockieren.

Mit den „Asynchronous Methods" ist es nun möglich, diese Aufgabe in einer einfachen Abfolge von Befehlen wie bei einer synchronen Programmierung zu codieren. Asynchronität kommt durch die Verwendung des „await"-Operators ins Spiel. Dieser verlagert bestimmte Methoden-Aufrufe in den Hintergrund. Die weitere Verarbeitung wartet an dieser Stelle auf das Ergebnis des „await"-Methoden-Aufrufs, ohne den Thread zu blockieren.

Im Beispiel werden Zeile A und Zeile B im Ausführungskontext des GUI-Threads durchgeführt, die Methode „GetResponseAsync" läuft im Hintergrund. Der GUI-Thread ist dabei nicht blockiert und nach dem Herunterladen der Datei kehrt die Ausführung auf magische Weise wieder zur weiteren Verarbeitung ab Zeile B zurück. Das Ergebnis der Hintergrund-Verarbeitung wird dabei als normales Methoden-Ergebnis geliefert, ohne das umständlich ein Wrapper-Objekt entpackt werden müsste.

Die Methoden unterstützen „await" explizit. Eigene Methoden müssen hierfür mit „async" markiert werden und spezielle Anforderungen erfüllen. Asynchronous Methods vereinfachen vieles. Die große Zahl an Fragen und Artikeln im Internet zeigt aber auch, dass es trotz der Verwendung von Techniken wie „Asynchronous Methods" auch weiterhin viele Fallstricke im Umgang mit mehreren Threads und Asynchronität gibt. Die Einführung der Schlüsselworte „async" und „await" wird bestimmt auch in C# nicht die letzte Weiterentwicklung auf diesem Gebiet bleiben.

Wohin die Reise geht

Wie bei Java auch, hat sich bei C# nach einer stürmischen Anfangsphase das Entwicklungstempo etwas verlangsamt und es dauert inzwischen immer ein paar Jahre, bis neue Sprach-Features oder Ideen in der Sprache umgesetzt werden. Mit dem Projekt „Roslyn" ist in den letzten Jahren aber ein neues, spannendes Thema hinzugekommen. In dessen Rahmen wurde der gesamte C#-Compiler in der Sprache C# selbst neu entwickelt.

Der Compiler wird zukünftig auch keine Blackbox mehr darstellen, sondern über eine Vielzahl von Schnittstellen Services für die Analyse und das Refactoring zur Verfügung stellen. So wie der Compiler selbst sind auch große Teile des „.NET"-Frameworks inzwischen Open Source. Man darf gespannt sein, welche neuen Ideen und Tools sich daraus entwickeln.

Fazit

Der Autor liebt Java, er liebt aber auch C#. Dieser Artikel hat einige interessante Sprach-Features von C# vorgestellt. Darüber hinaus gibt es noch vieles Weitere zu entdecken.

```
List<Person> personen = GetAllePersonen();

String wohnort = "Stuttgart";

var liste =
   from p in personen
   where p.Wohnort == wohnort && p.Beruf == "Entwickler"
   orderby p.Nachname, p.Vorname
   select p;
```

Listing 6: Einfache LINQ-Abfrage auf eine Liste im Speicher

```
private async void Laden_Click(object sender, RoutedEventArgs e) {

   // Zeile A:
   Inhalt.Text = "Wird geladen...";

   WebRequest request = WebRequest.Create("http://meinservice/
meinequelle");
   WebResponse response = await request.GetResponseAsync();

   // Zeile B:
   Inhalt.Text = "Größe der Seite: " + response.ContentLength;

   response.Close();

}
```

Listing 7: Asynchroner Aufruf mit await

Rolf Borst
webmaster@rborst.de

Rolf Borst entwickelt seit zehn Jahren Software für die Java-JEE-Plattform und liebt Java. Gleichzeitig war er von C# bereits bei den allersten Veröffentlichungen Ende der 1990er Jahre fasziniert. Seit dieser Zeit nutzt er C# immer wieder für seine privaten Projekte. Zurzeit verwendet er C# am liebsten für die Entwicklung von Apps.

http://ja.ijug.eu/15/2/8

Gestensteuerung und die nächste Welle der 3D-Kameras

Martin Förtsch und Thomas Endres, ParrotsOnJava.com

Innerhalb der letzten fünf Jahre gab es auf dem Gebiet der Natural User Interfaces (NUI) eine regelrechte Revolution, da eine Vielzahl neuer Geräte auf den Markt kam. Sie ermöglichen dem Nutzer zum Beispiel durch Gesten oder Sprache eine direkte Interaktion mit einer Benutzeroberfläche. Der Artikel zeigt, inwieweit diese Art der Steuerung sowohl intuitiv als auch komfortabel ist.

Tom Cruise im Jahr 2054. Vor einem mannshohen Display stehend bewegt er seine Hände und ordnet die dargestellten Informationen freihändig und nach seinen Vorstellungen auf dem Bildschirm an. Das war eine Schlüsselszene aus dem im Jahr 2002 erschienenen Kinofilm „Minority Report". Um dem Film etwas mehr wissenschaftlichen Charakter zu verleihen, bildete der Regisseur Steven Spielberg eine Expertenkommission und lud 15 Spezialisten auf diesem Gebiet in ein Hotel in Santa Monica ein. Dort wurden die Grundlagen für eine Computersteuerung entworfen. Diese noch zu entwickelnde Steuerung sollte nicht nur alle üblichen Interaktionen mit der Software vor einem Display erfüllen, sondern dabei auch noch futuristisch aussehen.

Geschichte

Wer sich heutzutage mit Gestensteuerung auseinandersetzen möchte, hat eine große Auswahl an verschiedenen Systemen, die sich unter anderem auch sehr gut mit Java ansteuern lassen. Zunächst ein sehr einfaches Java-Codebeispiel für die Leap Motion. Dieses kleine USB-Peripheriegerät ist dafür ausgelegt, Hände und Finger zu erkennen. Wie in *Abbildung 1* zu sehen, wird eine Hand über dem Gerät positioniert und deren dreidimensionale Position bestimmt. Die Leap Motion arbeitet mit einer Genauigkeit von bis zu 0,01mm und kann darüber hinaus auch einzelne Finger und Knöchel sowie deren Lage und Neigung erkennen.

Listing 1 zeigt den Zugriff auf Positions- und Lagedaten einer erkannten Hand. Im Beispiel wird das Listener-Konzept verfolgt, das eine Event-Methode „onFrame" anbietet. Diese erhält ein Controller-Objekt, über das per „controller.frame()" der Zugriff auf das aktuell erfasste Frame möglich ist. Der Aufruf „frame.hands().isEmpty()"überprüft, ob die Leap Motion in diesem Frame Hände erkennt.

Anschließend lassen sich durch einen einfachen Array-Zugriff die erkannten Handobjekte abfragen. In diesem Beispiel werden über „hand.palmNormal().roll()" die Daten zur seitlichen Neigung und mit „hand.direction().pitch()" die Vor- oder Rückwärtsneigung der Handfläche festgestellt. Daten zur Translation, also der Handposition oberhalb der Leap Motion, werden über „hand.palmPosition()" ermittelt. Damit können über diverse „get()"-Methoden die X-, Y- und Z-Koordinaten abgefragt werden.

Technisch funktioniert das Erkennen der Hände dadurch, dass drei Infrarot-emittierende Dioden von dem Gerät ausgehend ein Muster nach oben auf die Hände werfen. Zwei Infrarot-Kameras können dank eines durch die Parallaxe-Technik entstehenden 3D-Bilds Tiefen-Informationen gewinnen. Aus diesen wird über ein mathematisches Modell die Position der Hand errechnet.

Doch so einfach wie in diesem Beispiel gestaltet sich die Implementierung von Software zur Gestensteuerung nicht immer. Wirft man einen Blick zurück in die Geschichte der Natural User Interfaces, so trifft man

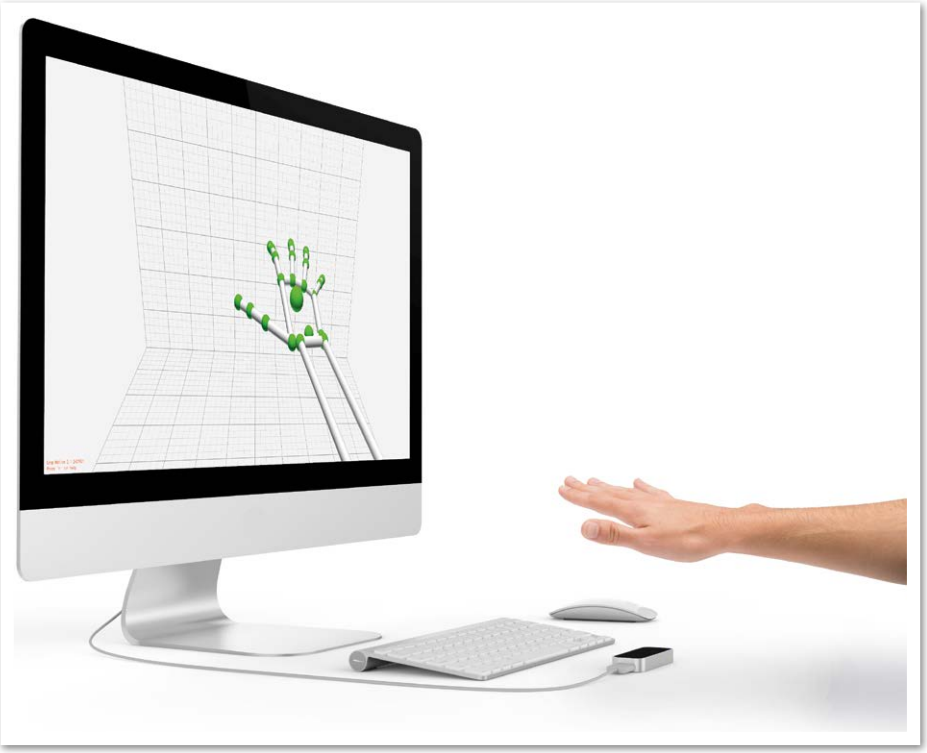


Abbildung 1: Leap Motion in Aktion mit einer über das Gerät positionierten Hand (Quelle: Leap Motion, „http://leapmotion.com")

```
public class LeapMotionListener extends Listener {
  public void onFrame(Controller controller) {
      Frame frame = controller.frame();
      if (!frame.hands().isEmpty()) {
          // Get the first hand
          Hand hand = frame.hands().get(0);

          // Get the hand's normal vector and direction
          Vector normal = hand.palmNormal();
          Vector direction = hand.direction();

          float handHeight = hand.palmPosition().getY();
          float handPitch = direction.pitch();
          float handRoll = normal.roll();
      }
  }
}
```

Listing 1

auf einen im Jahr 1990 in der WDR-Wissenschaftsshow vorgestellten Prototypen aus Darmstadt, der noch mit einer normalen Videokamera funktionierte. Dort ließ sich ein auf einem Bildschirm dargestellter Ball mit den Fingern frei bewegen und sogar greifen. „Man könnte meinen, dass das hier eine Spielerei ist, aber dahinter verbergen sich zumindest ganz ernste Absichten", sagte der luxemburgische Wissenschaftsjournalist Ranga Yogeshwar damals. Hintergrund dieses Versuchs war unter anderem, dass man dem Rechner viel mehr Eingabedaten übergeben musste, als es mit Tastatur oder Maus überhaupt möglich gewesen wäre.

Im Jahr 1993 kam eine für damalige Verhältnisse sehr fortschrittliche Webcam auf den Markt. Es handelt sich um die Silicon Graphics IndyCam, die im Verbund mit der Indy Workstation funktioniert. Noch heute ist sie im Deutschen Museum München als eines der wahrscheinlich ersten und wichtigsten Projekte im Bereich der Gestensteuerung zu begutachten. Zwei Jahre später benutze Siemens-Software die SGI IndyCam und entwickelte eine Software, die die Drehbewegungen des Kopfes vor der Kamera erkannte und diese auf einen dargestellten menschlichen 3D-Schädel übersetzte *(siehe Abbildung 2)*.

Dabei wurde jedoch nicht der Kopf erkannt, sondern offenbar nur die Bewegungen. Das lässt sich herausfinden, indem man seine Hand direkt vor der Kamera positioniert und nach links und rechts bewegt. Der auf dem Monitor dargestellte Schädel bewegt sich korrespondierend zu den Handbewegungen. Die IndyCam ist eine reine RGB-Kamera und erkennt nur das normale Farbspektrum. Durch diverse Algorithmen und eine einfache Bildsequenz-Technik werden Bewegungen von hautfarbenen Flächen erkannt und entsprechend interpretiert. Hintergrund dieser Herangehensweise ist, dass man aus einer einzelnen, einfachen Kamera keine echten Tiefen-Informationen gewinnen kann. Nichtsdestotrotz spricht man für damalige Verhältnisse von einem erstaunlichen Ergebnis.

Im Bereich der Gestensteuerung geschah in den darauffolgenden Jahren auf dem Verbrauchermarkt nicht viel. Erst mit der Entwicklung der Nintendo Wii Remote im Jahr 2006 bekam die Idee des quasi berührungslosen Steuerns neuen Auftrieb. Freilich kam bei der Wii ein Hardware-Controller zum Einsatz, den man in der Hand halten musste. Dennoch erschuf diese nichtstationäre Bedienung einer Spielekonsole gänzlich neue Freiheiten. Die kurz daraufhin erschienene Microsoft Kinect für die XBox ermöglichte diese Art der Bedienung sogar, ohne spezielle Hardware in der Hand halten zu müssen.

Nur wenig später (2008) implementierte eine Firma namens Oblong Industries ein Projekt namens „G-Speak". Dort gelang es, die noch im Jahr 2002 als Fiktion angesehene Szene aus „Minority Report" Realität werden zu lassen. Ein vor mehreren Displays stehender Mensch verschiebt Bilder von Display zu Display und kann dies relativ komfortabel freihändig erledigen. Dabei wird er von etlichen Kameras gefilmt und trägt spezielle, mit Sensoren ausgestattete Handschuhe *(siehe Abbildung 3)*.

Man kann sich vorstellen, dass die Arme bei dieser Art der Bedienung auf Dauer müde und sehr schwer werden. Dieser Effekt heißt „Gorilla Arm" und umfasst die Symptome schmerzender Muskeln, Muskelkater und angeschwollener Arme. Das liegt daran, dass diese Art Bedienung der grundlegenden menschlichen Ergonomie wiederspricht.

Stand der Technik

Erst mit der Veröffentlichung des Kinect-Sensors im Jahr 2010 und dem entsprechenden PC-Pendant im Jahr 2012 kam ein System auf den Markt, das eine echte berührungslose Steuerung ohne Verwendung von zusätzlichen Controllern zulässt. Das wird dadurch erreicht, dass neben einer RGB-Kamera eine zusätzliche Infrarotkamera integriert ist. Durch Aussenden eines durch Infrarot-Dioden generierten Sprenkelmusters (siehe „Speckle-Pattern") kann

Abbildung 2: Das Exponat zur Gestensteuerung von Siemens Software im Deutschen Museum München

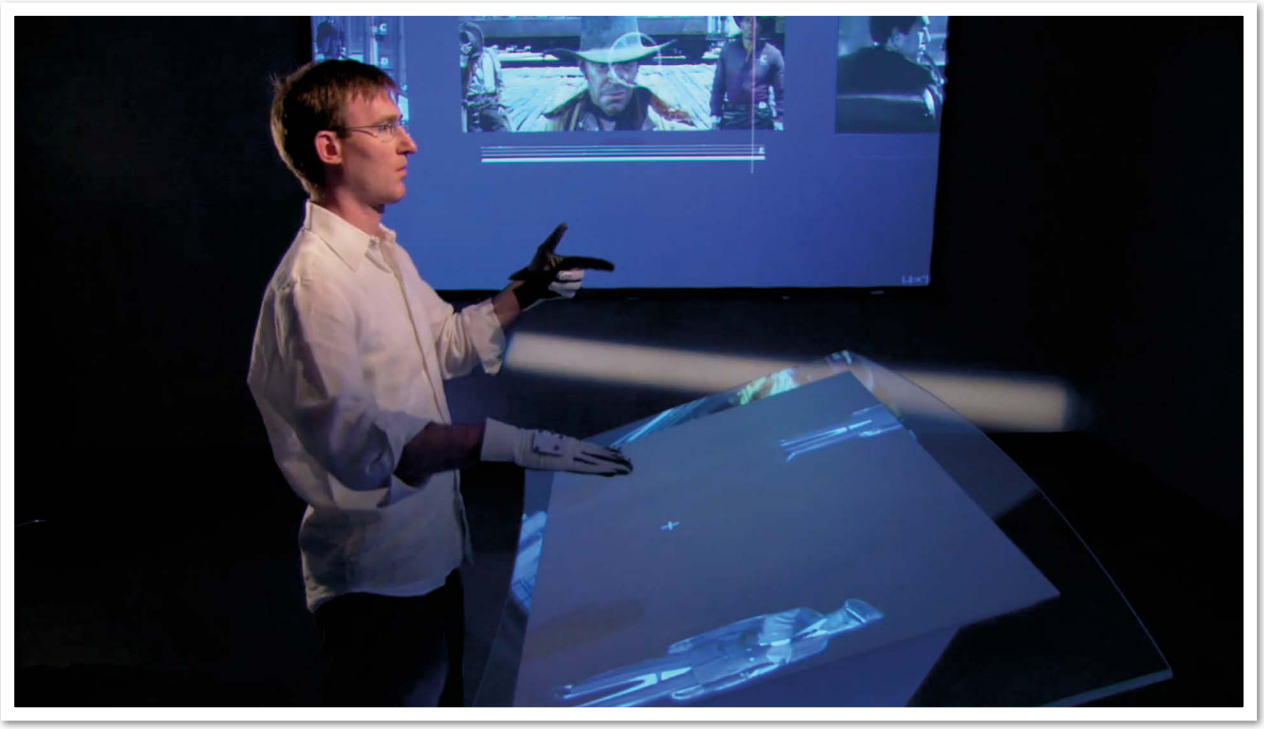


Abbildung 3: G-Speak ist eine real existierende Implementierung der aus dem Film "Minority Report" bekannten Vision (Quelle: Oblong Industries, „http://www.oblong.com")

mithilfe der IR-Kamera Tiefen-Information gewonnen werden.

Die neue Kinect 2 dagegen basiert auf der sogenannten „Time of Flight"-Technik. Dabei wird die Zeit berechnet, die die Lichtwellen von einem Emitter auf eine reflektierende Oberfläche und zurück benötigen (siehe „Continuous Wave"-Verfahren).

Doch nicht nur die renommierten Unternehmen spielen eine große Rolle auf dem Markt der Gestensteuerung. Es sind gerade auch die Underdogs, die sich – wie im Fall der Leap Motion – mittels Crowdfunding finanzieren und innovative Ideen auf den Markt bringen. An dieser Stelle seien beispielsweise das Thalmic Labs Armband „Myo" oder der „Fin"-Ring des indischen Start-ups RHLvision Technologies genannt. Bei „Myo" handelt es sich um ein Wearable Device, das nicht auf der Basis von Kameratechnik funktioniert. Das Armband misst durch Muskelkontraktionen hervorgerufene elektrische Aktivität. Die daraus gewonnenen Daten können zur Gestensteuerung benutzt werden.

Auch große Unternehmen beschäftigen sich mit dem Thema Gestensteuerung, die bis vor einiger Zeit noch nicht damit in Verbindung gebracht wurden. Mit dem Erscheinen der ersten Version des RealSense SDKs (vormals „Perceptual Computing SDK") betrat der Prozessorhersteller Intel die Bühne. Im Rahmen eines Joint Ventures mit dem Hardware-Hersteller Creative wurde eine erste Version der 3D-Kamera mit dem Codenamen „Senz3D" auf den Markt gebracht. Sie basiert auf einer ähnlichen Technologie wie bei der Kinect 2. Intel entwickelte das dafür benötigte Intel RealSense SDK. Erwähnenswert ist, dass neben der Erkennung von Händen, Fingern und vorimplementierten Gesten auch Gesichts- und Sprach-Erkennung möglich ist.

Intel arbeitet zurzeit an weiteren 3D-Kamera-Nachfolgern. Erste Prototypen geben eindeutige Hinweise darauf, wohin sich die Technologie entwickeln wird. Während die Kinect 2 bei einer Abmessung von 24,9 cm x 6,6 cm x 6,7 cm noch ein Kilogramm auf die Waage bringt, ist die neue Intel RealSense Kamera ein regelrechtes Fliegengewicht. Sie misst gerade einmal 10 cm x 0,8 cm x 0,38 cm. Durch die geringe Dicke ist eine Integration in Laptops und Tablets möglich *(siehe Abbildung 4)*.

Die Intel RealSense Kamera kann mit verschiedenen Sprachen programmiert

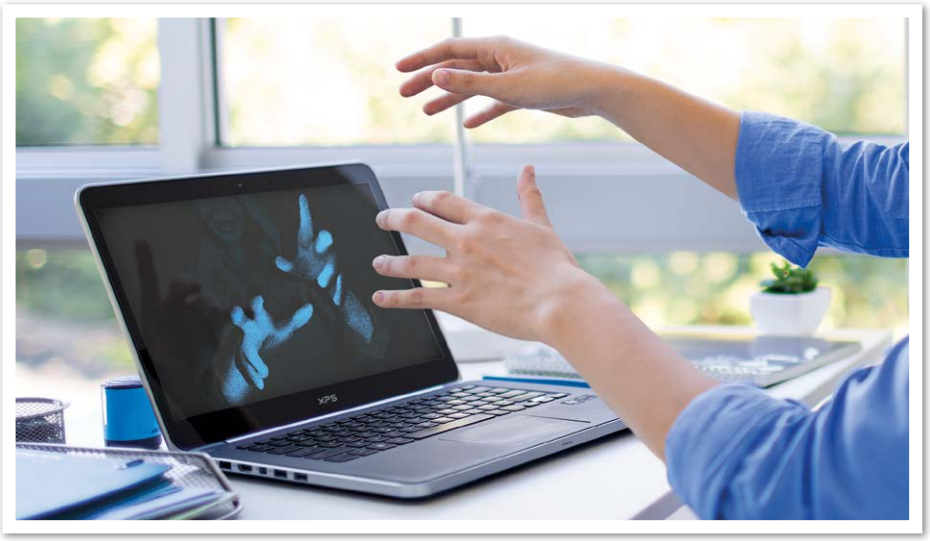


Abbildung 4: Intel-RealSense-Kameras werden so klein, dass eine Integration in Laptops und Tablets möglich wird (Quelle: Intel, „http://intel.com/realsense")

werden, in der Regel kommt aber C++ zum Einsatz. In der aktuellen Beta-Version des RealSense SDKs ist ein neues JavaScript-API hinzugekommen. Auch in Java gibt es die Möglichkeit, sich die Funktionalität dieser vielseitigen 3D-Kamera zu Nutze zu machen. In *Listing 2* ist der Zugriff auf Positionsdaten der Hand mit Intel RealSense demonstriert. Intel liefert einen Java-Wrapper, der sich sehr nah an der ursprünglichen C++-API bewegt. Dies erklärt auch, warum Methodennamen nicht den üblichen Java-Standards entsprechen.

Es wird zunächst ein Objekt der Klasse „PXCMSenseManager" erzeugt, der die Verwaltung des RealSense-API übernimmt. Mithilfe der Methode „QueryHand()" entsteht ein „PXCMHandModule", das benötigt wird, um Hände zu erkennen. Korrespondierend existieren auch Methoden zur Erkennung von Gesichtern und Emotionen.

Wie im Codebeispiel der Leap Motion *(siehe Listing 1)* muss auch hier zunächst eine Methode namens „AcquireFrame()" ausgeführt werden. Diese wartet nun, bis Daten zum Verarbeiten anliegen. Die weitere Ermittlung der Kameradaten wird nun unterbrochen, bis „ReleaseFrame()" aufgerufen wird. Es folgt das Abfragen der Handdaten durch „QueryHandData()". Dort wird ein Parameter übergeben, der angibt, in welcher Sortierung die Hände indexiert werden sollen. Der Parameter ist auf „ACCESS_ORDER_NEAR_TO_FAR" gesetzt. Die Hand, die sich also am nächsten zur Kamera befindet, bekommt den Index „0". Die Daten vom Typ „PXCMHandData" liegen daraufhin im Objekt „hand" vor. Nach einem zusätzlichen „QueryMassCenterImage()" lassen sich die erkannten Positionsdaten des Masseschwerpunkts der Hand ermitteln *(siehe Abbildung 5)*.

Doch wozu sollte man Gestensteuerung verwenden? Warum sollten sich Entwickler überhaupt mit diesem Thema beschäftigen? Zugegeben, in die Zukunft schauen kann niemand, noch ist das Thema recht jung. Die rasanten Entwicklungen in diesem Bereich in den letzten fünf Jahren lassen jedoch aufhorchen. Klassische Anwendungsgebiete waren seit jeher in der Unterhaltungsindustrie zu finden und dort insbesondere im Bereich der Spielekonsolen. Doch jüngste Entwicklungen werden auch dazu führen, dass diese Technologien auf eine breitere Anwenderschicht treffen.

Moderne SmartTVs besitzen heute die Möglichkeit, mittels spezieller Gesten Programme zu wechseln oder die Lautstärke zu ändern. Smartphones wie das Samsung Galaxy S4 lassen sich schon heute mit Air View durch Gesten bedienen.

Derzeit existieren erste Prototypen medizinischer Geräte, bei denen der Arzt berührungslos Patientendaten auf seinem Bildschirm nach seinen Vorstellungen darstellen lassen kann. Das kann für einen Chirurgen durchaus praktisch sein, da er keine Knöpfe anfassen muss und somit die Sterilität wahrt. Vor allem kann er sich Daten so anzeigen lassen, wie er es möchte, und muss diese Aufgabe nicht an andere delegieren.

Darüber hinaus gibt es erste universitäre Projekte, bei denen Chirurgie-Roboter per Gesten gesteuert werden. Mit der heutigen Genauigkeit der 3D-Kameras und der SDKs ist jedoch davon abzuraten, diese Techniken am lebenden Menschen auszuprobieren. Denn auf diesem Gebiet sind noch einige Jahre an Forschung nötig, um die Genauigkeit auf das nötige Maß zu erhöhen und Ausfallsicherheit zu gewährleisten. Ob diese Technologien zukünftig dann tatsächlich auch für sicherheitsrelevante Anwendungen verwendet werden, ist natürlich noch reine Spekulation.

Einsatzgebiete

Der Einsatz von Natural User Interfaces macht nur dann Sinn, wenn er sich positiv auf

```
// Create Sense manager and initialize it
PXCMSenseManager senseManager = PXCMSession.CreateInstance().CreateSenseManager();
senseManager.EnableHand(null);
senseManager.Init();

// Create hand configuration
PXCMHandModule handModule = senseManager.QueryHand();
PXCMHandData handData = handModule.CreateOutput();

while (true) {
  // Capture frame
  pxcmStatus status = senseManager.AcquireFrame(true);
  if (status.compareTo(pxcmStatus.PXCM_STATUS_NO_ERROR) < 0) break;

  handData.Update();

  // Get first hand data (index 0),
  PXCMHandData.IHand hand = new PXCMHandData.IHand();
  status = handData.QueryHandData(PXCMHandData.AccessOrderType.ACCESS_ORDER_NEAR_TO_FAR, 0, hand);

  if (status.compareTo(pxcmStatus.PXCM_STATUS_NO_ERROR) >= 0) {
  // Get some hand data
      PXCMPointF32 position = hand.QueryMassCenterImage();
      System.out.println(String.format("Center of mass for first hand at image position( % .2f, % .2f) ",
      position.x, position.y));
  }

      senseManager.ReleaseFrame();
}
```

Listing 2

Abbildung 5: Aktuelle Intel-RealSense-Kamera (F200) in einem Gehäuse zur Verwendung als USB-Peripheriegerät (Quelle: Intel, „http://intel.com/realsense")

die Effizienz auswirkt. Gestensteuerung kann in besonderen Fällen sogar einen Sicherheitsgewinn bedeuten. So hat der Autohersteller Hyundai mit seiner „HCD-14 Genesis-Concept Interior"-Demo im Jahr 2013 ein Gestensteuerungs-System vorgestellt, um das Entertainment-System im Auto zu bedienen.

Es kann von Vorteil sein, wenn man die Karte des Navigationssystems freihändig skalieren kann, ohne explizit einen Knopf drücken zu müssen. Natürlich existieren heutzutage auch Knöpfe am Multifunktions-Lenkrad. Jedoch ist dieser Platz begrenzt, während die Anzahl der elektronischen Assistenten im Fahrzeug immer weiter ansteigt.

Wenn man die Scheibenwischer seines Autos per Scheibenwischergeste ein- und ausschalten könnte, würde dies unter Umständen zu Verwirrungen mit anderen Verkehrsteilnehmern führen. Daher empfiehlt es sich, Gesten nur dann zu verwenden, wenn sie nicht anderweitig interpretierbar sind. Ebenso versteht es sich von selbst, dass auch die Zahl der möglichen Gesten begrenzt werden muss. Die Verwendung Hunderter verschiedener Gesten würde fast zwangsläufig zu Problemen beim Erlernen und bei der Erkennung der Gesten führen.

Eine weitere interessante Frage stellt sich in Bezug auf die genutzten Gesten-Metaphern. Für die heutige Generation ist es beispielsweise absolut intuitiv, die Lautstärke eines Geräts mit einer Drehbewegung des gebeugten Daumen und Zeigefingers durchzuführen. Der Grund dafür liegt in der Verwendung von Potentiometern seitdem es Lautstärke-Regler gibt, die seit Jahrzehnten genau auf diese Art bedient werden. Setzt sich Gestensteuerung durch, können sich nach jahrelanger praktischer Anwendung ganz andere Standard-Metaphern entwickeln.

Fazit

3D-Kameras werden immer kleiner und werden schon heute in Laptops und demnächst sogar in Tablets eingebaut. Hewlett Packard begann als erstes, die Leap Motion in Laptops zu integrieren. In diesem Jahr folgt Intel mit der RealSense-Kamera in Ultrabooks und Tablets. Das bedeutet, dass immer mehr Benutzer in Kontakt mit den Technologien zur Gestensteuerung kommen werden. Entwickler sind nun einerseits gefragt, Expertise auf diesem Gebiet aufzubauen und andererseits auch Anwendungsideen zu entwickeln. Von Vorteil sind fundierte Kenntnisse im Bereich der User Experience sowie ein Gefühl dafür, ob Steuerungsmethoden intuitiv sind oder nicht.

Ein großer Nachteil von Gestensteuerung ist das nicht vorhandene haptische Feedback. Bei dieser Art von Bedienung fehlt generell das Gefühl, eine Schaltfläche oder einen Knopf betätigt zu haben. Um dem entgegenzukommen, hat Disney Research ein Prototyp mit dem Namen „Aireal" entwickelt, der einem durch gezielte pulsierende Luftstöße das Gefühl vermittelt, einen Knopf gedrückt zu haben. Alternativ könnte man natürlich auch mit Handschuhen arbeiten, um haptisches Feedback zu vermitteln. Derzeit existiert jedoch noch kein Gerät für den Verbrauchermarkt.

Die Simplizität eines Knopfs bleibt daher zunächst unübertroffen. Wenn eine einfache Aktion durch einen simplen Knopfdruck regelbar ist, so scheint eine Geste nur in Ausnahmefällen sinnvoll zu sein. Beim Entwickeln neuer Ideen und Ansätze mit Gestensteuerung sollte daher die Devise „Keep it simple and stupid" gelten. Man sollte also nur natürliche Gesten in seinen Programmen und Steuerungen verwenden.

Martin Förtsch
martin.foertsch@gmail.com

Martin Förtsch ist ein Software Consultant aus München und studierte Informatik und angewandte Naturwissenschaften. Nebenberuflich beschäftigt er sich intensiv mit der Entwicklung von Gestensteuerungs-Software für verschiedene Kamerasysteme. Unter anderem steuert er die „Parrot AR"-Drone berührungslos und engagiert sich in Open-Source-Projekten für diesen Bereich. Hauptberuflich setzt er seine Schwerpunkte im Bereich der Software-Entwicklung (insbesondere mit Java), Datenbanken und Suchmaschinen-Technologien.

Thomas Endres
thomas-endres@gmx.de

Thomas Endres ist studierter Informatiker (TU München) und leidenschaftlicher Software-Entwickler aus Bayern. Er engagiert sich nebenberuflich im Bereich der Gestensteuerung und steuert dort unter anderem Drohnen und Spiele berührungslos. Er arbeitet aber auch an anderen Open-Source-Projekten in Java, C# und allen Spielarten von JavaScript. Beruflich betätigt er sich als Software Consultant und gibt Vorlesungen an der FH Landshut.

http://ja.ijug.eu/15/2/9

Microservices und die Jagd nach mehr Konversion – das Heilmittel für erkrankte IT-Architekturen?

Bernd Zuther, codecentric AG

Die Software-Entwicklung begibt sich derzeit auf eine neue Evolutionsstufe: Fachabteilungen haben gelernt, mithilfe agiler Methoden Änderungen in kurzen Zyklen produktiv umzusetzen. Damit haben sie die Möglichkeit, Änderungen an einem Produkt mit echtem Benutzer-Feedback vermessen zu können und durch empirische Untersuchungen des Benutzerverhaltens fundierte Entscheidungen über neue Software-Funktionalitäten zu treffen. Dazu müssen aber die meisten IT-Architekturen erst einmal so auf Vordermann gebracht werden, dass einzelne Teile einer Software flexibel ausgetauscht und unabhängig voneinander ausgeliefert werden können.

Im Marketing beschreibt die Konversionsrate eine Kennzahl, die den prozentualen Anteil zwischen den Besuchen einer Webseite und dem Erreichen eines bestimmten Ziels darstellt. *Abbildung 1* zeigt die Formel zur Berechnung der Konversionsrate. In einem Online-Shop kann das beispielsweise das Absenden einer Bestellung oder der Klick auf einen bestimmten Button sein. Mit dem Ziel, dass ein Benutzer eine Bestellung abschließt, beschreibt die Konversionsrate den prozentualen Anteil der Personen, die von einem Kauf-Interessenten zu einem zahlenden Kunden umgewandelt werden.

Die Konversionsrate stellt eine sehr gute Kennzahl zum Bewerten von neuen Software-Funktionalitäten dar, denn mit ihrer Hilfe kann man die potenzielle Umsatzsteigerung hochrechnen, die durch eine Änderung zu erwarten ist. Die Konversionsrate sollte man unter anderem heranziehen, wenn Software-Funktionalität „A" von Software-Funktionalität „B" abgelöst werden soll.

Nach der Entwicklung von Software-Funktionalität „B" werden die Funktionalitäten „A" und „B" an zwei unterschiedliche Gruppen ausgespielt und das Verhalten der Benutzer in diesen Gruppen gemessen. Durch dieses Vorgehen hat man die Möglichkeit, frühzeitig zu reagieren, wenn die Benutzer Software-Funktionalität „B" nicht annehmen und die Konversionsrate durch Software-Funktionalität „B" sinkt. In diesem Fall kann man immer noch bei Software-Funktionalität „A" bleiben.

Dieses Verfahren wird „A/B-Testing" genannt. Ein einfaches Beispiel dafür ist das Austauschen der Farbe eines „Call to action"-Buttons in einem Bestellprozess (etwa der „Jetzt kostenpflichtig kaufen"-Button zum Abschließen eines Bestellvorgangs).

Feature Toogle

Für einfache A/B-Tests, wie für die Farbänderung eines „Call to action"-Buttons, kann mit dem sogenannten „Feature Toggle Pattern" *[1, 2]* gearbeitet werden. Dies ist ein recht einfaches „if then else"-Statement *(siehe Listing 1)*.

Mit einem Feature Toggle lassen sich unter anderem Bedien- oder Darstellungselemente ein- beziehungsweise ausblenden, Styles ändern oder sogar Backend-Systeme zur Laufzeit umschalten. An seine Grenzen stößt dieses Verfahren allerdings, wenn man in einem Online-Shop beispielsweise einen kompletten Bestellvorgang für einen A/B Test austauschen möchte. Dies liegt sehr häufig daran, dass die Software-Architekturen nicht dafür ausgelegt sind, konsequent A/B-Tests zu unterstützen, da in einer monolithischen Software-Architektur funktionale Kontexte häufig nicht ausreichend voneinander abgegrenzt sind.

Monolithische Software-Architektur

Ein Ansatz, den man sehr häufig in der Praxis findet, ist die sogenannte „monolithische Software-Architektur". In einer monolithischen Software sind die Benutzer-Schnittstelle sowie die Datenzugriffs- und Logik-Schicht in einem einzelnen Artefakt vereint. Die monolithische Software-Architektur bringt allerdings eine Reihe von Nachteilen mit sich:

- Die Anwendung ist häufig nicht besonders gut wart- und erweiterbar
- Bei Modifikationen können nicht vorhersehbare Neben-Effekte auftreten
- Änderungen müssen meistens mehreren Teams bekanntgegeben werden, da es häufig Modul-Verantwortlichkeiten gibt

```
if (benutzer.is_in_testgroup_a)
  change_button_to_green
else
  change_button_to_pink
end
```

Listing 1

$$\text{Konversionsrate} = \frac{\text{Anzahl der Zielerreichungen}}{\text{Besuche}}$$

Abbildung 1: Formel zur Berechnung der Konversionsrate

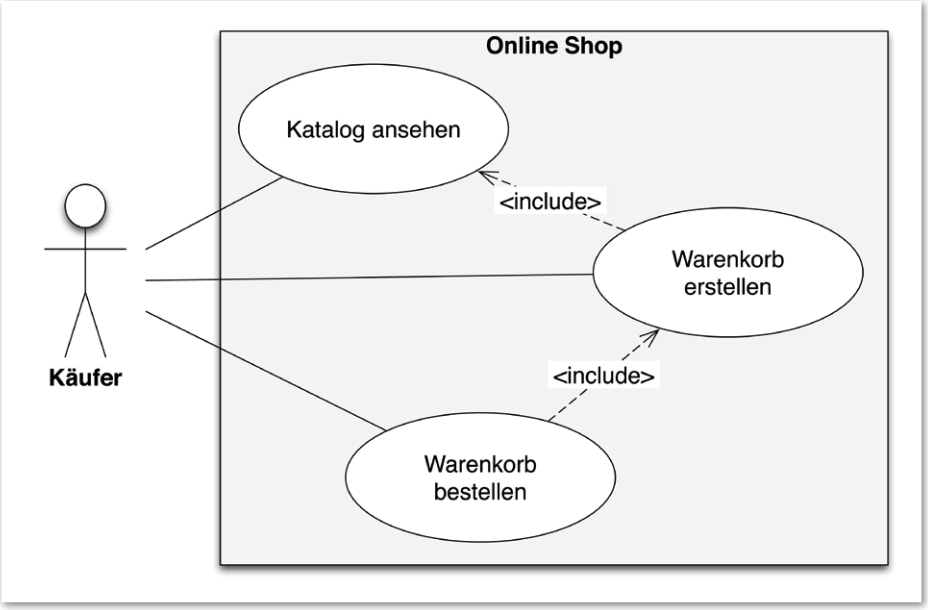


Abbildung 2: Anwendungsfalldiagramm eines Online Shop

und bei größeren Änderungen mehrere Module und damit Teams betroffen sind
- Teile des Software-Systems sind nur schlecht wiederverwendbar
- Eine Lastenverteilung von Teil-Komponenten ist nur schwer möglich
- Die Build-Laufzeiten können mehrere Stunden betragen

Natürlich besitzt diese Architektur-Form auch einige Vorteile, ansonsten würde man sie nicht so häufig vorfinden. Monolithische Anwendungen sind einfach zu übertragen und zu betreiben, da man sich um wenige Abhängigkeiten kümmern muss. Außerdem sind monolithische Anwendungen relativ einfach zu refaktorisieren, da sich der gesamte Quellcode meistens in einem Repository befindet. Auch in puncto Performance hat diese Architektur ihre Vorteile, da in der Regel wenige Remote Calls aufgerufen werden.

Man kann allerdings feststellen, dass monolithische Applikationen nur wenig wandelbar sind, was das konsequente Durchführen von A/B-Tests erschwert. Es ist aber wünschenswert, wenn man Anwendungen wie Lego-Bausteine ständig reorganisieren und mit neuen Bausteinen bestücken könnte. Eine Lösung für dieses Problem verspricht die sogenannte „Microservice-Architektur" *[3]*.

Microservice-Architektur

Es gibt verschiedene Versuche, den Begriff „Microservice" zu erklären. Einen ausführlichen Definitionsversuch findet man in Ralf Westphals Blog *[4]*. Für diesen Artikel wird eine einfache Erklärung verwendet, die auf den Gedanken von Timmo Freudl-Gierke *[5]* basiert. Für das weitere Verständnis ist ein Microservice ein autonomer Lieferant einer bestimmten Funktionalität, der alle Daten verwaltet, die der Service zum Liefern dieser Funktionalität benötigt. Daraus ergibt sich, dass jeder Microservice eine bestimmte Funktionalität der Geschäftslogik implementiert. Es bleibt aber offen, mit welchen Mitteln (Programmiersprache, Datenbank, Protokolle) diese Funktionalität implementiert wird. Es ist außerdem nicht ausgeschlossen, dass ein Microservice auch eine Benutzerschnittstelle haben kann.

Da jeder Microservice über eigene Daten verfügt, muss er ein eigenes, explizites Repository besitzen, das ihn mit den benötigen Daten beliefert. Ein solches Repository kann eine relationale Datenbank, aber auch jede beliebige andere Datenquelle sein.

Wenn man Funktionalität und Daten als eine zusammengehörende Einheit betrachtet, ergeben sich interessante Implikationen für die Zusammenarbeit von verschiedenen Teams. Bei Optimierungen an der Datenbank oder für Schema-Änderungen ist nur noch das Team verantwortlich, dem die Funktionalität gehört, und Änderungen müssen nicht mit anderen Teams abgesprochen werden, solange sich dabei die Schnittstelle nach außen nicht ändert. Eine solche Schnittstelle sollte möglichst plattformneutral sein, denn dann können einzelne Software-Komponenten beliebig für einen A/B Test ausgetauscht werden. *Abbildung 2* zeigt, welche minimale Funktionalität man von einem Online-Shop erwartet. In dem Diagramm sind die folgenden Anwendungsfälle zu sehen:

- Einem Benutzer muss ein Katalog von Produkten angezeigt werden
- Aus den angezeigten Produkten kann sich ein Benutzer einen Warenkorb zusammenstellen
- Diesen Warenkorb kann ein Benutzer bestellen

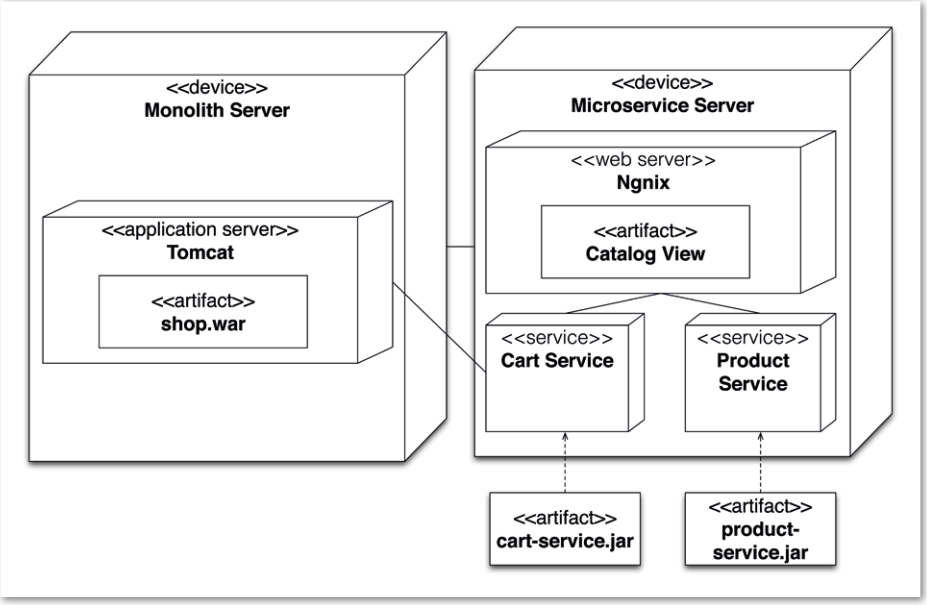


Abbildung 3: Verteilungsdiagramm einer Kombination aus monolithischer und Microservice-Architektur

Das Anwendungsfall-Diagramm zeigt also grundlegende Funktionalitäten. Im „Domain Driven Design" werden Funktionalitäten, die eine Fachlichkeit darstellen, als eigenständige zustandslose Services modelliert. Deshalb liegt es nahe, das Anwendungsfall-Diagramm als grobe Orientierungshilfe zu benutzen, um festzustellen, wie viele Microservices dieser Online Shop benötigen wird. Daraus ergeben sich die folgenden drei Microservices:

- Produkt-Service
- Warenkorb-Service
- Bestell-Service

Die Frage ist nun, ob diese Aufteilung schon ausreicht, wenn man das Ziel berücksichtigt, die Konversionsrate von verschiedenen Katalog-Ansichten oder Bestell-Prozessen vermessen und verbessern zu können. Die Antwort ist ein klares Nein. Um den Katalog oder den Bestellprozess optimieren zu können, sind zwei neue Microservices erforderlich. Diese stellen die Benutzeroberfläche bereit und man kann sie gegebenenfalls für einen A/B-Test durch andere Microservices austauschen. Daraus ergeben sich nun fünf Microservices, die man technisch mit Hilfe von drei REST-Services und zwei Single-Page-Web-Anwendung (SPA) zu einem Gesamtsystem zusammenfügen kann.

Bei diesen Kontextgrenzen kann man die fünf Dienste von fünf Teams parallel entwickeln lassen – vorausgesetzt, die Teams einigen sich auf gemeinsame Schnittstellen. Sie können dann autonom ihre Technologien festlegen. Das Produkt-Service-Team kann zum Beispiel MongoDB oder Elasticsearch als Datenbank für die Persistierung der Produkte benutzen und das Warenkorb-Service-Team alternativ eine Redis-Datenbank. Der Microservice, der die Katalog-Ansicht implementiert, steht in einer Kunden-Lieferanten-Beziehung zum Produkt-Service. Dafür kann dieser eine REST-Schnittstelle zur Verfügung stellen.

Migration eines Monolithen

Bisher kam der technische Schnitt eines Online-Shops zur Sprache, der sich in einem „Grüne Wiese"-Projekt anbietet. Wie soll man aber vorgehen, wenn man schon einen laufenden, monolithischen Shop betreibt, der im mittleren fünfstelligen Bereich Umsatz pro Stunde generiert? Ein Big-Bang-Umstieg wäre viel zu riskant, da die Gefahr groß ist, dass das neue System Fehler enthält oder vom Kunden nicht angenommen wird. Deshalb liegt die Frage nahe, ob man mithilfe von Microservices auch eine monolithische Anwendung im laufenden Betrieb migrieren kann.

Angenommen, man betreibt eine monolithische Anwendung, die die drei Anwendungsfälle aus dem vorangegangen Abschnitt implementiert. Technisch gesehen, handelt es sich dabei um eine Java-EE-Anwendung, die in Form einer WAR-Datei in den Servlet-Container Tomcat ausgeliefert wird.

Der Produktmanager möchte eine neue Katalog-Ansicht ausprobieren. Der neue Produktkatalog verfügt über sehr viele dynamische Bedienelement und die Frontend-Entwickler sind sich schnell einig, den neuen Katalog lieber mit einem neuen, hippen JavaScript-Framework zu programmieren, als mit ihrem heißgeliebten Web-Frontend-Framework Wicket, das in der monolitischen Anwendung zum Einsatz kommt. Natürlich möchte der Produktmanager den neuen Katalog erst mit 20 Prozent seiner Kunden testen, um zu validieren, ob der neue Katalog die Konversion wirklich erhöht, fachlich und technisch können also beide Systeme erst einmal nebeneinander existieren *(siehe Abbildung 3)*.

Aufgrund der vorangegangen Definition ist ein Microservice ein Lieferant einer bestimmten Funktionalität. Ein Monolith verfügt über verschiedene Funktionalitäten und für eine bestimmte Funktionalität kann dieser auch das führende System zur Bereitstellung der Funktionalität sein. Für unser Beispiel heißt das, dass die Java-EE-Anwendung weiter für den eigentlichen Bestellprozess verantwortlich bleibt.

JavaScript-Single-Page-Anwendung

Abbildung 3 zeigt ein Verteilungsdiagramm, das eine mögliche Kombination aus monolithischer und Microservice-Architektur enthält. In diesem Integrationsmodell wird eine JavaScript-Single-Page-Anwendung (SPA), die die Katalog-Funktionalität bereitstellt, vom Web-Server Ngnix ausgeliefert. Die SPA greift auf zwei REST-Schnittstellen zu, die in eigenen Prozessen auf dem Server laufen und über ein Fat Jar gestartet werden. Dies ist eine ausführbare Jar-Datei, in

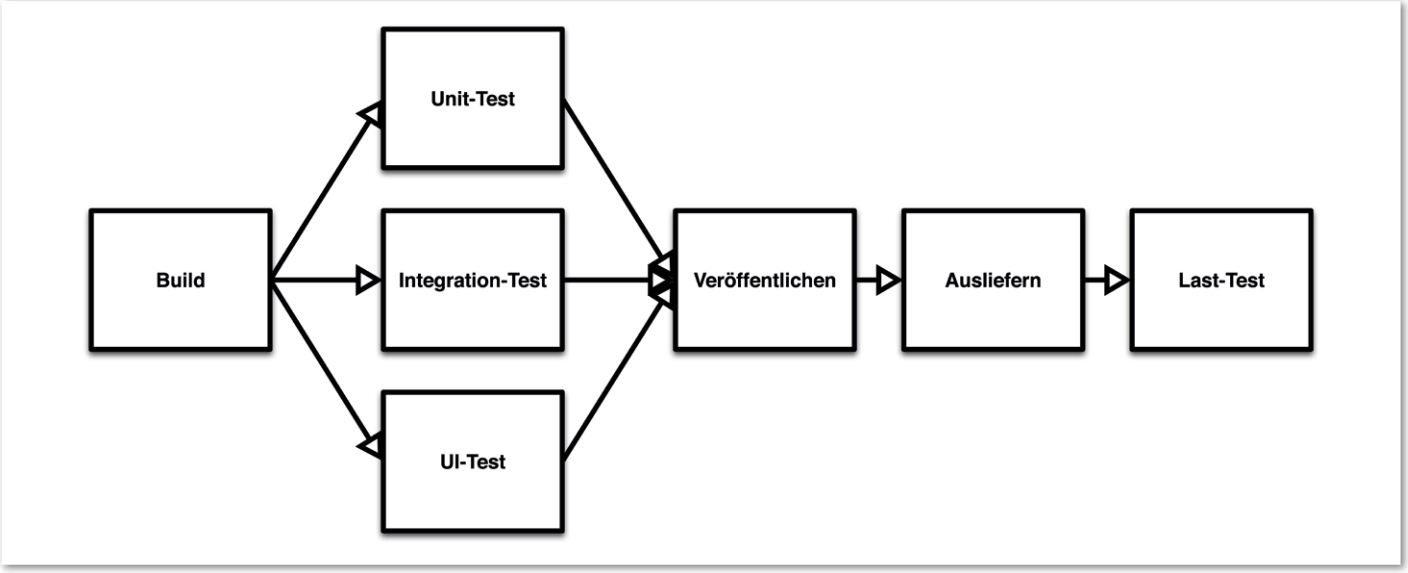


Abbildung 4: Deployment-Pipeline eines Monolithen

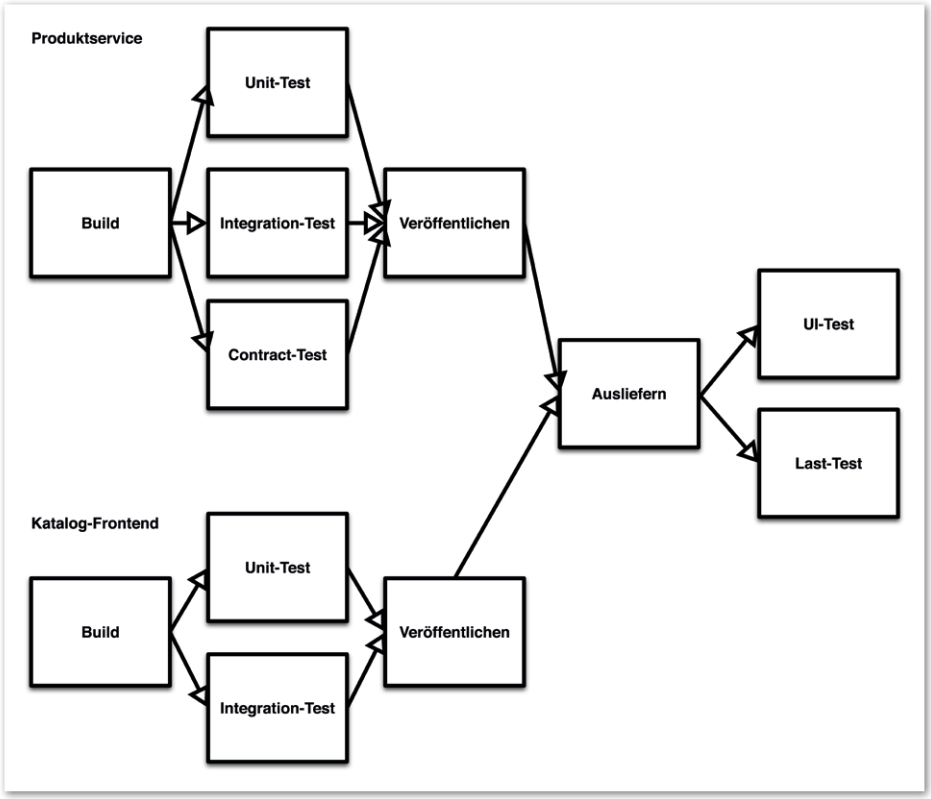


Abbildung 5: Multi-Deployment-Pipeline einer Microservice-Architektur

der alle Bibliotheken enthalten sind, die von einer Anwendung benötigt werden.

Ngnix sorgt neben der Auslieferung der SPA auch dafür, dass es beim Zugriff auf den Produkt- und Warenkorb-Service (Cart) nicht zu Problemen mit der „Same Origin"-Policy kommt, die in den meisten Browsern implementiert ist.

Um den alten, monolithischen Shop mit dem neuen Katalog zu verknüpfen, muss ein Benutzer die Möglichkeit haben, in den alten Shop einzuspringen und auf den Warenkorb-Service des neuen Katalogs zuzugreifen, um im Altsystem den Bestellprozess abzuschließen. Technisch kann das über einen Link *(etwa „https://checkout.microservice.io/shop/checkout?cart=b579b0b4-75c8-49c7-92e9-f8265c6e5d1c")* erreicht werden, man verlinkt also aus dem neuen Katalog in den alten Shop. Dieser reagiert auf den Link und sorgt dafür, dass über eine andere Strategie auf den Warenkorb-Service zugriffen wird.

So lässt sich Stück für Stück eine Anwendung in eine Microservice-Architektur migrieren. Außerdem hält man das Risiko bei der Migration sehr gering, denn wenn die neue Funktionalität nicht richtig funktioniert oder die Benutzer die neue Funktionalität nicht annehmen, kann man mit geringem Aufwand wieder die alte Implementierung verwenden. In *Abbildung 3* ist aber auch zu sehen, dass sich die Anzahl an Artefakten erhöht. Aus einem Artefakt (monolithische Architektur) sind nun vier Artefakte geworden. Continuous Delivery und Continuous Integration werden deshalb noch wichtiger. Denn die Flexibilität beim Zusammenstellen von einzelnen Fachlichkeiten zu einem Gesamtsystem erhält man nicht umsonst.

Multi-Deployment Pipeline

Wie beschrieben, erhöht sich mit einer Microservice-Architektur die Anzahl der Artefakte,

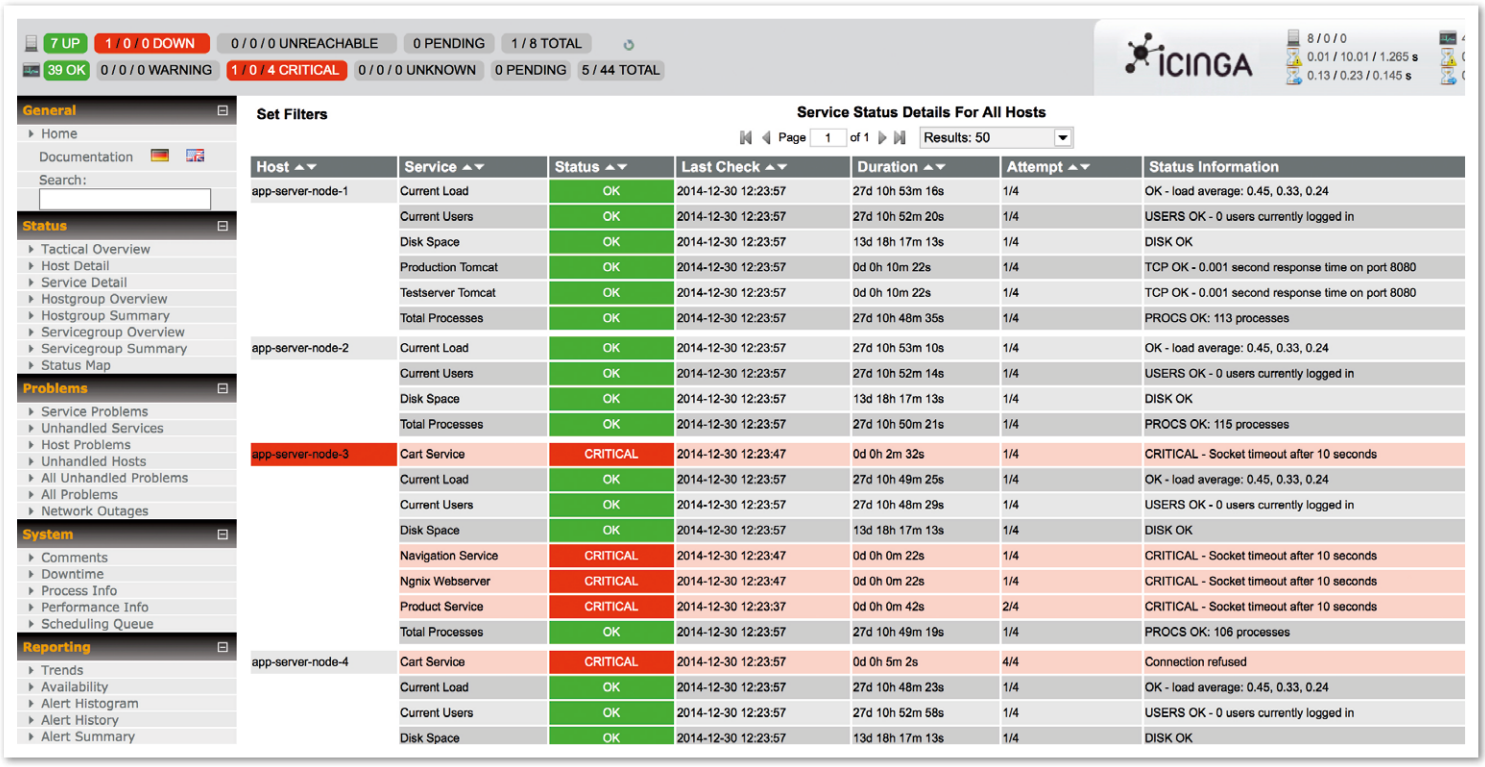


Abbildung 6: Icinga zur Überwachung einer Microservice-Architektur

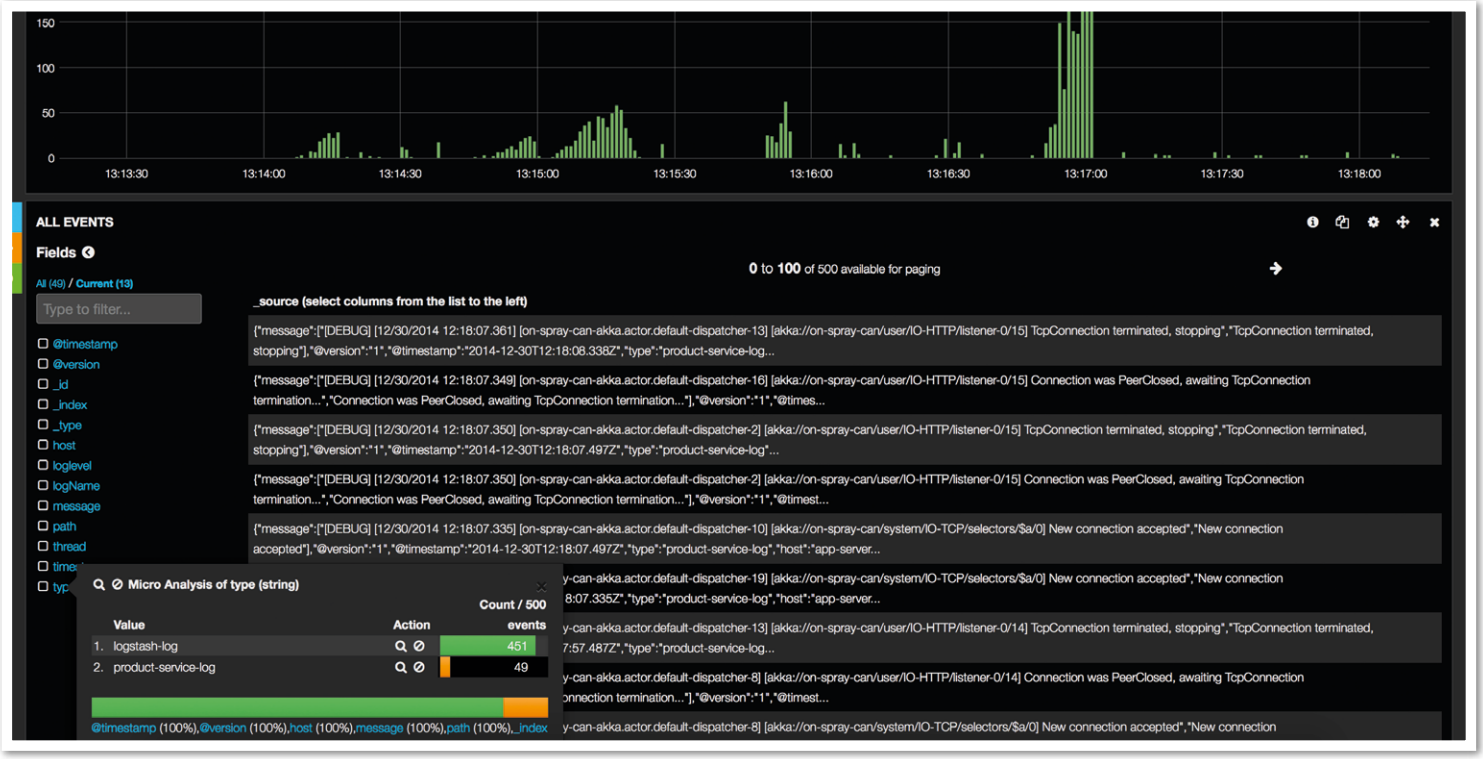


Abbildung 7: ELK-Stack für die Analyse verschiedener Log-Dateien

die für das Gesamtsystem erforderlich sind. *Abbildung 4* zeigt eine Deployment-Pipeline eines Monolithen, die mit dem Build der Software beginnt. Nachdem die Software gebaut ist, werden eine Reihe von Tests parallel ausgeführt und danach wird ein Artefakt veröffentlicht.

Dieses Artefakt kann, wie gesagt, eine WAR-Datei sein. Die Veröffentlichung erfolgt über einen Artefakt-Repository oder einen File-Server, von dem aus das Artefakt für die Auslieferung benutzt wird. Nach der Auslieferung können noch automatisierte Akzeptanz- und Lasttests laufen, um Integrationsprobleme festzustellen oder Performance-Regressionsfehler zu finden.

Abbildung 5 zeigt, wie sich die Deployment-Pipeline in einer Microservice-Architektur ändert. Exemplarisch sind zwei Komponenten dargestellt, die wir auch im Verteilungsdiagramm aus dem vorangegangen Abschnitt finden können: die Katalog-Darstellung („Catalog View") und der Produkt-Service. Sie sorgen dafür, dass ein Benutzer einen Katalog angezeigt bekommt. In *Abbildung 5* kann man sehen, dass diese Komponenten Zweige innerhalb der Deployment Pipeline darstellen, die nach dem Veröffentlichen der Artefakte wieder zusammenlaufen.

Nach dem Veröffentlichen werden die Artefakte zusammen ausgeliefert und das System kann integrativ getestet und dann durch anschließende UI- und Last-Tests überprüft werden. Auf Service-Ebene sollte man noch sogenannte „Contract-Tests" durchführen, die das Team schreiben sollte, das einen bestimmten Service benutzt. Diese Tests stellen sicher, dass die Schnittstellen eines Microservice nicht-abwärtskompatibel geändert werden.

Eine weitere Frage ist, welche Artefaktform man für eine möglichst einfache Auslieferung auswählt. Hier gibt es leider keine generelle Antwort. Container-Technologien wie Docker werden immer populärer und sind sicherlich ein weiterer Schritt nach vorn, um eine durchgängige Lieferkette vom Entwicklungssystem bis in Produktion gleich zu gestalten. Doch Container sind noch relativ jung und sicherlich nicht für alle Anwendungsfälle geeignet.

Um Software zu installieren, hat jedes Betriebssystem seine Eigenheiten: Windows hat Installationsroutinen, Linux eine Paketverwaltung und MacOS das sogenannte „Application Bundle". Auch diese Artefaktformen haben ihre Vorteile und fördern die Übertragbarkeit von Software *[6]*. Alle drei haben gemeinsam, dass auch Abhängigkeiten (Datenbanken, Webserver) ausgedrückt werden können, die benötigt werden, damit die Software fehlerfrei läuft.

Viele Wege führen damit zu einer durchgängigen Deployment-Pipeline und die einzige Bewertungsgrundlage sollte deshalb sein, ob die Software nach einem Commit automatisiert auf mindestens einer Umgebung ohne manuelle Schritte lauffähig ist. Falls ja, verwendet man bereits die richtige Technologie. Durch eine Multi-Deployment-Pipeline wird nun das gesamte System ausgeliefert. Doch wie überwacht man ein System, das aus mehreren Anwendungen besteht?

Monitoring

Zum Überwachen von komplexeren Microservice-Architekturen, kann man die freie Software Nagios benutzen *[8]*. Sie basiert auf einer objektorientierten Konfiguration. Bestandteil dieser Konfiguration sind Hosts, die über eine IP-Adresse definiert werden. Einem Host können die zu überwachenden Services zugeordnet werden. Die Überwachung erfolgt über sogenannte „Kommandos", die zur Alarmierung ausgeführt werden. Letztlich kann man noch definieren, wer bei bestimmten Unregelmäßigkeiten eine Nachricht erhalten soll. Mit Icinga *[9]* ist ein Nagios-Fork entstanden, der über eine modernere Web-Oberfläche verfügt und einen dynamischeren Entwicklungsprozess als Nagios besitzt *(siehe Abbildung 6)*.

Nagios oder Icinga bieten die Möglichkeit, die Verfügbarkeit der Services zu überwachen und SLA-Reports zu erstellen. Wenn man mit diesen Werkzeugen feststellt, dass

ein Service zu einer bestimmten Uhrzeit nicht verfügbar war, muss natürlich eine Analyse folgen, was die Ursache für das Problem ist. Dazu durchsucht man in der Regel die Logdateien einer bestimmten Applikation.

Erinnern wir uns an *Abbildung 3*, dann ergibt sich schon aus der Anzahl an Artefakten, dass das Gesamtsystem über mindestens vier Log-Dateien verfügt. Dazu kommen noch die Logdateien des Webservers und der Datenbanken. Damit müssen selbst in diesem einfachen Beispiel bis zu sieben Log-Dateien für die Ursachen-Analyse untersucht werden, die noch dazu auf mehreren Servern verteilt sind.

Um Korrelationen in unterschiedlichen Log-Dateien zu finden und den Zugriff auf die unterschiedlichen Logdateien zu zentralisieren, gibt es den sogenannten „ELK-Stack" *[10]*. Er besteht aus den folgenden Bestandteilen *(siehe Abbildung 7)*:

- *Elasticsearch*
  Indexiert und macht die Informationen in der Log-Datei durchsuchbar
- *Logstash*
  Sammelt die Log-Dateien ein, transformiert und normalisiert dieses und extrahiert bestimmte Daten daraus
- *Kibana*
  Bietet eine grafische Oberfläche für die Log-Datei-Analyse

Der ELK-Stack bietet somit die Möglichkeit, die Log-Dateien zu zentralisieren und verschiedene Log-Formate normalisiert durchsuchbar zu machen.

Fazit

Auch wenn Microservices derzeit sehr als Hype gelten, sind sie kein Allheilmittel. Ein genaues Verständnis der Domäne ist weiterhin erforderlich *[11]*. Nur dann kann man die Kontextgrenzen zwischen Services genau definieren und damit fachliche Konzepte technisch trennen. Außerdem ist es wichtig, dass man das Problem genau versteht, das der Fachbereich lösen möchte. Nur dann kann man eine technische Lösung definieren, die die Anforderungen genau erfüllt und technisch möglichst einfach ist.

Wenn man für die Kommunikation zwischen den unterschiedlichen Microservices einen technologieneutralen Vertrag wählt, ist die konkrete Technologie irrelevant, die zur Implementierung des Microservice verwendet wird. Wichtig ist, dass sich sowohl die Auslieferung als auch das Testen der produzierten Artefakte automatisieren lässt.

Die Berücksichtigung dieser Punkte bietet die Möglichkeit, mithilfe von A/B-Tests ein Software-Produkt ständig zu optimieren, die Konversionsrate zu verbessern und den Umsatz zu steigern. Mit dem richtigen Software-Schnitt besteht die Möglichkeit, das Software-System ständig zu verändern und langfristig den Erfolg eines Produkts durch Innovation zu sichern.

Mit dem Schnitt einer Software in verschiedene funktionelle Teile bekommen die Themen „Continuous Delivery" und „Continuous Integration" eine noch größere Bedeutung. Denn nur dann, wenn die Auslieferung des Software-Systems möglichst einfach ist, hat man die Zeit, in einem solchen verteilten System das Benutzerverhalten zu analysieren und die Software ständig zu optimieren. Bei einer solchen Analyse hilft unter anderem ein zentralisiertes Logging, wie es der ELK-Stack oder kommerzielle Lösungen bieten.

Eine Referenz-Implementierung des Online-Shops aus diesem Artikel ist im GitHub-Repository des Autors *[12]*. In dem Projekt „Appstash" befindet sich eine Vagrant-Datei, die ein komplettes Entwicklungscluster erstellt, das über eine Lecacy-Online-Shop-Anwendung, eine Microservice-Implementierung eines Katalogs, einen Jenkins, der eine komplette Multi-Deployment-Pipeline mithilfe der Debian-Paketverwaltung zeigt, und über das beschriebene Nagios-Monitoring sowie den ELK-Stack verfügt.

Quellen

[1] http://www.heise.de/developer/artikel/Continuous-Delivery-mit-dem-FeatureToggle-Pattern-1825477.html
[2] http://martinfowler.com/bliki/FeatureToggle.html
[3] http://martinfowler.com/articles/microservices.html
[4] http://blog.ralfw.de/2014/08/warnung-vor-dem-microservice-versuch.html
[5] jaxenter.de/artikel/nie-wieder-monolithen-176652
[6] http://bernd-zuther.de/deployment-ganz-einfach
[7] http://12factor.net
[8] http://www.nagios.org
[9] https://www.icinga.org
[10] https://bernd-zuther.de/splunk-marke-eigenbau-mit-elasticsearch-logstash-und-kibana
[11] http://www.effectivetrainings.de/blog/2014/12/18/be-good-was-ein-guter-entwickler-konnen-muss
[12] https://github.com/zutherb/AppStash

Bernd Zuther
bernd.zuther@codecentric.de

Bernd Zuther ist Diplom-Informatiker und seit dem Jahr 2008 als Software-Entwickler tätig. Derzeit arbeitet er bei der codecentric AG als Berater im Bereich „Agile Software Factory", wo er sich vor allem mit Big Data, Continuous Delivery und agilen Software-Architekturen beschäftigt. Bernd Zuther verfügt über viel praktische Erfahrung im Bereich „E-Commerce" und beschäftigt sich seit mehreren Jahren mit der Entwicklung von hochverfügbaren, individuellen Online-Shop-Systemen.

http://ja.ijug.eu/15/2/10

Maven-Repository bald verfügbar

Oracle hatte auf der OpenWorld in San Francisco die baldige Veröffentlichung eines Maven-Repository angekündigt. Dafür plädierte die Entwickler-Community seit geraumer Zeit. Inzwischen sei der Konzern in der Endphase des Projekts.

Das Repository soll in Zukunft unter *https://maven.oracle.com* nach einer SSO-Anmeldung zu finden sein. Auf der Plattform finden Entwickler nur Artefakte, die von Oracle bereitgestellt und für Kompilierung, Paketierung, Tests und Verwendung von Client-APIs verwendet werden, zum Beispiel JAR-Dateien von WLS, ADF, SOA, OSB, BPM und WCP.

Da Patches einen Support-Vertrag erfordern, sollen nur Release-Versionen (also 12.1.2, 12.1.3) zur Verfügung stehen.

Alles klar? Von wegen!

Von Glücksrädern, Bibliothekaren und schwierigen Fragen

Dr. Karl Kollischan, kobaXX Cosultants

Unser Leben besteht aus Entscheidungen – im Beruf wie auch im Privatleben. Häufig hängt unser Geld, unsere Karriere, unser Glück davon ab. Wem sollen wir unser Vertrauen schenken? Wen halten wir für kompetent? Welcher Mitarbeiter ist für eine bestimmte Aufgabe am besten geeignet? Sollen wir ein aus dem Ruder gelaufenes Projekt, in das schon viel Geld investiert wurde, retten oder abbrechen? Was macht uns in Zukunft glücklich? Für viele unserer wichtigen Entscheidungen kennen wir die Gründe. Wirklich?

Die moderne Kognitionspsychologie liefert verblüffende Antworten darauf, was in unserem Gehirn passiert, wenn wir Entscheidungen treffen. Wir haben zwei Denksysteme: ein langsames, rationales, das wir bewusst erfassen und ein schnelles, intuitives, das ständig präsent ist und uns automatisch in Bruchteilen von Sekunden mit Einschätzungen versorgt. Diese sind meistens richtig und sinnvoll, führen jedoch in bestimmten Situationen zu eklatanten Fehlentscheidungen. In der letzten Ausgabe wurden die spezifischen Charakteristiken, Stärken und Schwächen der beiden Systeme betrachtet und wie sie bei unseren Entscheidungen zusammenwirken. In diesem Artikel gehen wir tiefer auf einige spezielle Situationen ein und lernen mentale Muster kennen, die uns immer wieder in fatale Denkfallen tappen lassen.

WYSIATI – what you see is all there is

Steve wird wie folgt beschrieben: „Er ist sehr scheu und verschlossen, immer hilfsbereit aber ziemlich weltfremd und kaum an anderen Menschen interessiert. Als sanftmütiger und ordentlicher Mensch hat er ein Bedürfnis nach Ordnung und Struktur und eine Passion für Details."

Ist Steve nun Landwirt oder Bibliothekar? Wer auf Bibliothekar tippt, dem geht es so wie den meisten Menschen. Die Beschreibung ist eingängig und „eine gute Geschichte": man hat also sofort ein Bild von Steve, das dem Stereotyp eines Bibliothekars entspricht. Dabei hat man wichtige statistische Erwägungen außer Acht gelassen.

Wenn man davon ausgeht, dass die beschriebenen Persönlichkeitsmerkmale zu einem gleich hohen Anteil wie alle anderen in der Bevölkerung auftreten, gibt es mit hoher Wahrscheinlichkeit auch unter den Landwirten einige detailverliebte, scheue und sanftmütige Vertreter. Da es wesentlich mehr Landwirte als Bibliothekare gibt, ist es wahrscheinlicher, dass Steve Landwirt ist als Bibliothekar.

Erscheint das als kontraintuitiv? Das soll auch so sein, denn System 1 (Denkprozesse, die automatisch und mühelos ablaufen und zum größten Teil unterhalb der Bewusstseinsschwelle stattfinden, siehe letzte Ausgabe der Java aktuell) liefert aufgrund von Assoziationen eine schnelle Lösung. Wahrscheinlich weiß man sogar, dass es wesentlich mehr männliche Landwirte als Bibliothekare gibt – doch kam einem diese statistische Tatsache bei der Frage nach Steves Beruf in den Sinn? Vermutlich nein, es zählte nur, was man in diesem Augenblick wusste– WYSIATI!

Der WYSIATI-Effekt erklärt, warum wir in einer komplexen Welt mit partiellen Informationen so gut zurechtkommen. Er erklärt die Leichtigkeit und Schnelligkeit, mit der wir Entscheidungen treffen und Urteile bilden. Die von uns aus wenigen Fakten konstruierte Geschichte kommt der Wirklichkeit dabei meistens sehr nahe, so dass sie adäquate und zielführende Handlungen unterstützt.

Allgemein wird diese effiziente und mit wenig geistiger Anstrengung verbundene Art des Problemlösens als Heuristik bezeichnet. Doch wie wir am Beispiel von Steve sehen, hat diese Leichtigkeit und Schnelligkeit den Preis von systematischen Fehlern, die auch als kognitive Verzerrungen oder Bias bezeichnet werden. Im obigen Beispiel begehen viele Menschen den sogenannten „Basisraten-Fehler": Die Basisrate oder a priori Wahrscheinlichkeit wird nicht beachtet, die im Fall von Steve

die Verteilung von Landwirten, Bibliothekaren, Ärzten etc. in der Bevölkerung beschreibt, weil nur die Ähnlichkeit von Steve zum Stereotyp eines Bibliothekars ins Auge fällt.

Eine leichtere Frage beantworten

In Anbetracht der Komplexität der Welt und der Entscheidungen, die wir zu treffen haben, sind wir erstaunlich selten überfragt. Bisweilen müssen wir uns zwar geistig anstrengen, wenn wir mit Fragen wie nach dem Ergebnis von „17 x 24" konfrontiert werden. Doch bei den wirklich kniffligen Themen treffen wir unsere Entscheidungen oft mit großer Leichtigkeit.

Wir haben intuitive Meinungen und Urteile über fast alles was uns begegnet. Menschen sind uns auf Anhieb sympathisch oder unsympathisch, ohne dass wir viel über sie wissen. Wir sind von Geschäftsideen begeistert oder stehen ihnen skeptisch gegenüber, ohne dass wir sie gründlich analysiert hätten oder unsere Meinung begründen könnten. Wir berufen uns auf unser „Bauchgefühl".

Kahnemann schlägt für diesen Prozess, wenn wir intuitive Meinungen über komplexe Sachverhalte bilden, folgende Erklärung vor: Wenn eine befriedigende Antwort auf eine komplexe Fragestellung nicht schnell gefunden werden kann, ersetzt System 1 die ursprüngliche Frage durch eine einfachere Frage, die wir leicht beantworten können, ohne dass wir uns dessen – typisch für System 1 – bewusst wären. Eine Befragung deutscher Studenten nach ihrer Lebenszufriedenheit liefert dafür ein wunderbares Beispiel. Einer Gruppe von Studenten wurden zwei Fragen gestellt:

- Wie glücklich fühlen Sie sich zurzeit?
- Wie viele Verabredungen hatten Sie im letzten Monat?

Untersucht wurde die Korrelation zwischen den beiden Antworten. Fühlen sich die Studenten, die angaben, dass sie viele Verabredungen hatten, glücklicher als diejenigen mit wenigen Verabredungen? Zur Überraschung der Experimentatoren war dies nicht der Fall, die Korrelation lag nahe bei null. Offensichtlich war die Anzahl ihrer Verabredungen nicht das, was den Studenten als Erstes in den Sinn kam, wenn sie ihre Lebenszufriedenheit bewerten sollten. Eine andere Gruppe bekam die gleichen Fragen gestellt, jedoch in umgekehrter Reihenfolge:

- Wie viele Verabredungen hatten Sie im letzten Monat?
- Wie glücklich fühlen Sie sich zurzeit?

Die Ergebnisse waren hier völlig anders, die Korrelation der Antworten war so hoch, wie Korrelationen zwischen psychologischen Messgrößen nur sein können.

Was war geschehen? Die Erklärung ist einfach und ein gutes Beispiel für den Ersetzungsprozess. Die Frage nach der Lebenszufriedenheit ist schwierig und für eine Beantwortung müssen viele komplexe Faktoren berücksichtigt werden.

Offensichtlich spielte für die erste Gruppe die Anzahl ihrer Verabredungen keine große Rolle im Gesamtkonzept ihrer Lebenszufriedenheit. Doch anders als diese Gruppe mussten die Studenten der zweiten Gruppe gar nicht mehr angestrengt nachdenken – sie hatten schon die Antwort parat.

Nach ihrem Liebesleben gefragt, reagierten sie eindeutig emotional und fokussierten einen glücklichen beziehungsweise unglücklichen Aspekt ihres Lebens. Diese durch die Verabredungsfrage hervorgerufene Emotion war präsent bei der komplexen und schwierigen Frage nach dem Glück.

System 1 hat diese komplexe Frageerstellung ersetzt und stattdessen die leichtere Frage nach der Zufriedenheit mit dem Liebesleben – gemessen an der Zahl der Verabredungen – beantwortet.

Sind die Studenten der zweiten Gruppe nicht ganz klar bei Verstand? Glauben sie wirklich, dass die beiden Fragen synonym sind? Natürlich nicht. Würde man sie nach den beiden Konzepten fragen, würden sie sagen, dass man beides auseinander halten müsse. Aber sie wurden nicht danach gefragt; sie wurden gefragt, wie glücklich sie seien und System 1 hatte eine Antwort parat.

Die Frage funktioniert nicht nur mit „Verabredungen". Das gleiche Muster findet man bei Fragen nach der Beziehung zu den Eltern oder zur finanziellen Situation. Jede emotional bedeutsame Frage, die sich auf die Stimmung einer Person auswirkt, hat den gleichen Effekt: WYSIATI.

Selbstverständlich hat System 2 (bewusstes Denken, das Konzentration und Anstrengung erfordert, siehe letzte Ausgabe der Java aktuell) die Gelegenheit, diese intuitive Antwort kritisch zu hinterfragen. System 2 ist jedoch oft faul und folgt den Weg des geringsten Widerstands. Es unterstützt die heuristische Antwort, ohne genau zu überprüfen, ob dies tatsächlich angemessen ist. Das hat den Vorteil, dass man nicht überfragt ist und sich nicht anstrengen muss. Wahrscheinlich fällt einem nicht einmal auf, dass man nicht die ursprüngliche Frage beantwortet hat, ja nicht einmal, dass es eine schwierige Frage war, weil einem so mühelos und schnell eine Antwort einfiel. Man denke nur an die Schläger-und-Ball-Aufgabe aus der letzten Ausgabe.

Anker-Effekte

Sicherlich sind einem die beeindruckenden Küstenmammutbäume bekannt, die auch als „Redwoods" bezeichnet werden und an der Küste Nordkaliforniens und Oregons vorkommen. Es sind die größten Bäume der Welt – beträgt die Höhe des größten Küstenmammutbaums mehr oder weniger als 366 Meter? Wie hoch kann so ein Baum maximal werden? Man beantworte diese Frage schnell und intuitiv und ohne zu googeln.

Laut Wikipedia hat das höchste lebende Exemplar eine Höhe von etwas über 115 Meter. Lag das eigene Ergebnis wesentlich darüber? Dann hat System 1 wieder hervorragend gearbeitet, denn man wurde gerade mit der Zahl „366" geankert, die willkürlich in den Raum gestellt wurde. Besuchern des San Francisco Exploratorium wurde diese Frage gestellt, einmal wie oben mit der Zahl 366 Meter und einmal mit 55 Metern als Anker. Die mittleren Schätzwerte der beiden Gruppen betrugen 257 beziehungsweise 86 Meter.

Der Anker-Effekt wurde nicht nur in psychologischen Experimenten im Labor nachgewiesen, sondern zeigt sich auch im wirklichen Leben und auch bei Experten, die ihr Fachgebiet kennen. So sollten Immobilienmakler den Wert eines Hauses einschätzen, das tatsächlich zum Verkauf stand. Sie erhielten umfangreiches Informationsmaterial, das auch eine preisliche Forderung enthielt. Die eine Hälfte der Unterlagen enthielt eine höhere Preisvorstellung, die andere eine niedrigere.

Erwartungsgemäß wichen die beiden Gruppen in ihrer Einschätzung stark voneinander ab. Interessant ist, dass keiner der Makler, als sie nach ihren Entscheidungskriterien für ihre Einschätzung gefragt wurden, den Preis in der Informationsbroschüre genannt hat – im Gegenteil, sie beteuerten,

dass diese Preisvorstellung ihre Entscheidung in keinter Weise beeinflusst habe. Tatsächlich waren sie für den Anker-Effekt fast genauso anfällig, wie Studenten ohne Immobilienerfahrung, denen die gleiche Aufgabe gestellt wurde. Aber es kommt noch bizarrer.

Das Glücksrad-Experiment

Zusammen mit Tversky hat Kahnemann folgendes Experiment durchgeführt. Sie stellten ein manipuliertes Glücksrad auf, das entweder bei „10" oder „65" stehen blieb. Sie drehten das Glücksrad und forderten die Versuchsteilnehmer auf, die Zahl, bei der es stehen blieb, zu notieren. Anschließend fragten sie die Versuchsteilnehmer, wie hoch der Prozentsatz afrikanischer Staaten in den Vereinten Nationen sei. Personen, bei denen das Glücksrad bei „65" stehen blieb, gaben eine hohe Zahl an – im Durchschnitt 45 Prozent, die anderen eine niedrigere – hier 25 Prozent im Durchschnitt.

Kann man im Beispiel mit den Küstenmammutbäumen den Anker-Effekt noch halbwegs rational begründen, indem man etwa denkt, der Fragesteller wird wohl ein Experte sein und einen Grund für die in der Frage genannte Zahl (den Anker) gehabt haben, so liefert offensichtlich das Drehen eines Glücksrads keinerlei nützliche Informationen über irgendwas. Die Teilnehmer des Experiments hätten es schlichtweg ignorieren sollen. Aber sie taten es nicht.

Der Anker-Effekt wurde wieder und wieder durch Experimente bestätigt, oft mit absurden Anker-Schätzwert-Kombinationen wie Sozialversicherungsnummer, Preis einer Weinflasche, den letzten Stellen der Telefonnummer oder dem Geburtsjahr von Attila. Immer wurden die Schätzwerte von der Zahl beeinflusst, die den Personen im Vorfeld dargeboten wurde.

Der Anker-Effekt ist auf das Priming zurückzuführen, das schon in der letzten Ausgabe betrachtet wurde. Genauer handelt es sich dabei um eine Suggestion, die selektiv kompatible Informationen ins Gedächtnis ruft. Niemand hat wahrscheinlich geglaubt, dass ein Küstenmammutbaum tatsächlich 366 Meter hoch sein kann, aber seine Assoziationsmaschine erzeugt automatisch das Bild eines sehr hohen Baums. Und da System 1 immer eifrig bemüht ist, ein konsistentes Bild der Wirklichkeit zu erschaffen, tut es sein bestes, eine Welt zu erschaffen, in der der Anker die richtige Zahl ist.

Dies wird auch eindrucksvoll durch folgenden Versuch bestätigt: Versuchsteilnehmern wurden folgende zwei Anker-Fragen zur Temperatur gestellt: „Ist die Jahresdurchschnitts-Temperatur in Deutschland höher oder niedriger als zwanzig Grad Celsius?" und „Ist die Jahresdurchschnitts-Temperatur in Deutschland höher oder niedriger als fünf Grad Celsius?" Allen Teilnehmern wurden anschließend kurz Wörter dargeboten, die sie erkennen sollten. Das Ergebnis war, dass Teilnehmer, denen die Zwanzig-Grad-Frage gestellt wurde, Wörter mit Sommer-bezug, wie Sonne oder Strand, leichter und schneller erkannten als die andere Gruppe.

Bei der Fünf-Grad-Frage zeigte sich der gleiche Effekt bei Wörtern mit Winterbezug, wie Frost oder Ski. Der hohe beziehungsweise niedrige Temperaturwert aktiviert jeweils unterschiedliche Vorstellungskomplexe im Gedächtnis und erzeugt so verzerrte Schätzwerte für die Jahrestemperatur. Das gleiche Experiment wurde auch mit der Frage nach dem Durchschnittspreis deutscher Autos durchgeführt. Ein hoher Anker ließ Markennamen wie „Audi" oder „Mercedes" leichter erkennen, ein niedriger Anker Marken wie „Volkswagen".

Anker-Effekte im Alltag

Das Phänomen des Anker-Effekts ist im Alltag weit verbreitet und wird auch bewusst zur Manipulation unseres Kaufverhaltens eingesetzt, etwa indem im Fahrradladen ein edles Bike in einer sehr hohen Preisklasse ausgestellt ist. Anker-Effekte erklären auch, warum willkürliche Rationalisierung eine erfolgreiche Marketingstrategie ist.

In einem Supermarkt in den USA wurde Campbell's Soup um zehn Prozent billiger verkauft. An manchen Tagen stand bei den vergünstigten Dosen ein Schild mit der Aufschrift „Maximal 12 Dosen pro Person" an anderen Tagen ein Schild mit der Aufschrift „Keine Begrenzung pro Person". An den Tagen mit Begrenzung wurden pro Person durchschnittlich sieben Dosen gekauft, was in etwa doppelt so viele waren, als an den Tagen ohne Beschränkung. Natürlich spielen hier auch andere Effekte eine Rolle, so wird Begrenzung auch mit Verknappung assoziiert. Jedoch generiert die Zahl „12" auf dem Schild den gleichen Effekt, den auch ein Glücksrad generieren würde.

Interessant ist auch der Anker-Effekt bei der Obergrenze von Schadensersatzforderungen. Zum Beispiel haben sich Chemiefirmen oder Kliniken, die häufig mit derartigen Forderungen konfrontiert werden, für eine gesetzliche Obergrenze der Haftungssummen stark gemacht. Das klingt zunächst wie eine feine Sache für die Versicherungen. Zwar wird es keine höheren Forderungen mehr geben als die festgelegte Obergrenze – sagen wir eine Million Euro – geben. Jedoch bewirkt der Anker-Effekt, dass viele Entschädigungsbeträge, die ansonsten viel kleiner wären, in Richtung eine Million getrieben werden.

Das Zusammenspiel von System 1 und System 2

Welchen Anteil hat unser bewusstes Denken an der ganzen Sache? Urteils- und Entscheidungsaufgaben werden letztendlich von System 2 abgeschlossen. Auf dem orientalischen Basar glaubt man natürlich nicht den ersten Preis, den der Händler nennt, sondern weiß, dass höchstens die Hälfte realistisch ist. Aber dennoch arbeitet System 2 mit den Daten, die von System 1 automatisch und unwillkürlich geliefert werden. Daher unterliegt auch System 2 dem verzerrenden Einfluss von Ankern, weil dadurch bestimmte Gedächtnis-Inhalte leichter abrufbar sind oder manche Informationen leichter aufgenommen werden als andere.

Selbst wenn man jetzt einiges über den Ankereffekt weiß, in der tatsächlichen Entscheidungssituation hat unser System 2 keine Kontrolle über die unbewusst stattfindenden Operationen von System 1 und bekommt diese noch nicht einmal mit. Die Wirkung von Zufalls-Ankern, wie sie etwa von Glückrädern erzeugt werden, liefert dafür sicherlich das eindrücklichste Beispiel.

Die Forschung zu Anker-Effekten macht deutlich, dass unser Denken und Handeln weit mehr vom augenblicklichen Umfeld beeinflusst wird, als man erkennt. Man kann allerdings nicht ständig bewusst und aufmerksam sein, das würde viel zu viel Kraft kosten und man würde die kognitive Leichtigkeit und Schnelligkeit verlieren, die in unserer komplexen Umwelt notwendig ist. Dennoch kann man in wichtigen Situationen besonders wachsam sein. Besonders anfällig ist man für Anker-Effekte, wenn man bei einem Urteil unsicher ist. Kürzlich konnte der Autor den Anker-Effekt in einer Planning-Poker-Runde an sich selbst beobachten: Zufällig hat er gesehen, wie ein Kollege – genauso wenig Experte wie er

bei der zu schätzenden Thematik – für sich und vielleicht unbewusst die Zahl Drei mit den Fingern gezeigt hat. Sofort erschien ihm das als sehr plausibler Wert für den Aufwand.

Es ist bekannt, wie wichtig es ist, bei Aufwandsschätzungen darauf zu achten, dass die Schätzer ihre Ergebnisse unabhängig voneinander abgeben. Dies wird normalerweise damit begründet, dass Meinungen von Experten oder Personen, die dafür gehalten werden, das Urteil beeinflussen. Dies ist richtig, doch die Anker-Effekte zeigen, dass die Beeinflussung noch viel subtiler ist. Nicht nur Experten-Meinungen haben Auswirkung auf unsere Entscheidungen, sondern einfach schon eine in den Raum gestellte Zahl!

Fazit

Bei wichtigen Entscheidungen sollte man besonders darauf achten, möglichst viele unabhängige Meinungen einzuholen. Dabei ist es manchmal wichtig, dass die Anonymität der Befragten gewahrt wird. Nicht zuletzt ist es eine gute Idee, bisweilen auch sein eigenes Urteil kritisch zu hinterfragen.

Weitere kognitive Verzerrungen werden in der nächsten Ausgabe vorgestellt. Unter anderem geht es um die Leichtigkeit, mit der bestimmte Informationen aus dem Gedächtnis abgerufen werden können, die die Einschätzung von Ereignissen beeinflussen und zum Beispiel zu einer verzerrten Wahrnehmung von Risiken führen. Oder dass die mit Verlusten verbundenen negativen Emotionen für uns wesentlich schwerer wiegen als die positiven Emotionen bei entsprechenden Gewinnen. Diese Verlust-Aversion ist oft dafür verantwortlich, dass allzu lange an Projekten festgehalten wird, die nicht mehr zu retten sind.

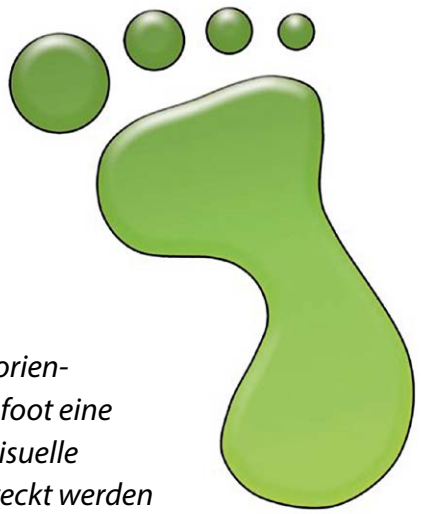
Dr. Karl Kollischan
karl.kollischan@kobaxx.com

Dr. Karl Kollischan ist selbstständiger Berater, Trainer und Coach. Seine Schwerpunkte sind agiles Denken und Arbeiten, Projektmanagement und Business-Analyse. Aktuell berät er eine große Organisation bei der Einführung agiler Software-Entwicklung.

http://ja.ijug.eu/15/2/11

Greenfoot: Einstieg in die objektorientierte Programmierung

Dennis Nolte



Greenfoot bietet jungen Menschen einen spielerischen Einstieg in die objektorientierte Programmierung in Java. Als ein „Meta-Framework" ermöglicht Greenfoot eine unkomplizierte Implementierung sogenannter „Mikrowelten", durch deren visuelle Repräsentationen bereits früh das Interesse für die Informationstechnik geweckt werden kann. Greenfoot ist zwar durch einige didaktische Einschränkungen limitiert – bietet prinzipiell aber den vollen Umfang der Programmiersprache Java.

Entwickelt wurde Greenfoot von Michael Kölling und Poul Henriksen, die Version 1.0 ist im Jahre 2006 veröffentlicht worden. Als ein Meta-Framework, in Java implementiert, bietet es die Möglichkeit, spielerisch interaktive „Mikrowelten" zu implementieren. Diese Programme werden selbst ebenfalls in Java implementiert. Dadurch ist Greenfoot plattformunabhängig. Außerdem ist Greenfoot unter „GPL 2 with Classpath Exception" lizenziert. Dadurch können die erstellten Szenarien auch unter anderen Lizenzierungen veröffentlicht werden.

Greenfoot bietet ein eigenes API, das unter der Maxime entworfen wurde, die im Rahmen der Erstellung von interaktiven Mikrowelten benötigten Methoden möglichst minimalistisch zu kapseln. Somit erhalten die Benutzer auf einfache Art und Weise Zugriff auf komplexere Funktionalitäten.

Wieso Greenfoot?

Greenfoot führt Programmieranfänger in die objektorientierte Programmierung mit Java

ein, ermöglicht den interaktiven Umgang mit Objekten und visualisiert die Klassenstruktur von Java-Projekten. Dieser integrative Ansatz begünstigt, dass Objekte und Klassen vom ersten Schritt an umgesetzt werden („Objects First"). Durch die Verinnerlichung dieser Konzepte entsteht ein klares Verständnis des objektorientierten Programmier-Paradigmas.

Eine Herausforderung dabei ist, dass an der Universität ein höherer Prozentsatz der Studenten über eine intrinsische Motivation für die Informationstechnik verfügt, während sich Schüler in erster Linie noch orientieren und nicht notwendigerweise eine solche Motivation mit sich bringen – auch da ihre Wahlmöglichkeiten im Vergleich zu der Vielzahl an Studiengängen begrenzt sind. Somit muss ein Werkzeug, das an der Schule eingesetzt werden soll, neben den Konzepten auch die Motivation beziehungsweise die Begeisterung für das Thema vermitteln. Greenfoot stellt ein solches Werkzeug dar und erzielt dies durch eine Visualisierung der gebotenen Szenarien sowie durch das Ermöglichen einer direkten Interaktion zwischen Nutzer und Mikrowelt.

Auch Lehrende sind gefordert, wenn sie bereits zu Beginn eines Kurses die objektorientierten Konzepte in korrekter und möglichst umfangreicher Form vermitteln möchten. Dies stellt sie unter Umständen vor hohen zeitlichen Aufwand, der in ihrer Tätigkeit als Lehrende nicht zu realisieren ist. Um diesen Aufwand zu verringern, bietet Greenfoot auch für Lehrende bereits eine Vielzahl von vorgefertigten Szenarien, die fachliche sowie didaktische Probleme ansprechen. Darüber hinaus gibt es mit dem sogenannten „Greenroom" *(siehe „http://greenroom.greenfoot.org")* die Möglichkeit zum Austausch mit anderen Lehrkräften.

Visualisierung und direkte Interaktion

Wie bereits erwähnt, verfügen Schüler in einem Informationstechnikkurs nicht zwingend über eine intrinsische Motivation für die Programmierung. Um die Motivation durch das Werkzeug zu kreieren, stellen die Entwickler Greenfoots folgenden Aspekt in den Mittelpunkt: Die theoretischen Konzepte sollen mit einer konkreten Erfahrung verknüpft werden. In ihren Augen wird dies zum einen durch die Visualisierung der Mikrowelt sowie der Objekte selbst und zum anderen durch die Möglichkeit der direkten Interaktion mit der Mikrowelt ermöglicht.

Während es vielerlei Werkzeuge gibt, die prinzipiell dieselbe Zielsetzung haben wie Greenfoot (etwa „Karel the robot" oder „BlueJ"), erfüllen diese in der Regel nur einen der beiden Aspekte. Greenfoot erzielt die reibungslose Kombination beider Aspekte und erweckt somit spielerisch Begeisterung für die Programmierung und Informationstechnik *(siehe Abbildung 1)*.

Durch die Visualisierung einzelner Objekte wird anhand von physischen Entitäten intuitiv ersichtlich, dass ein Objekt über Attribute sowie Methoden verfügt. Auch das Geheimhaltungsprinzip, Polymorphie und weitere objektorientierte Konzepte werden durch die Benutzeroberfläche dem Nutzer sehr diskret ersichtlich.

Flexible Gestaltung der Szenarien

Während der Großteil anderer didaktischer Entwicklungsumgebungen auf ein festes Szenario setzt, bietet Greenfoot die Möglichkeit, verschiedene Szenarien einzusetzen oder diese gar selbst zu entwickeln. Nun lässt sich argumentieren, dass ohne didaktischen Hintergrund erstellte Szenarien somit eine Fokussierung auf die Vermittlung wichtiger objektorientierter Konzepte fehlt. Somit wäre Greenfoot nur bedingt zum Selbststudium geeignet. Dabei gilt es jedoch zu beachten, dass Greenfoot außerordentlich viele Konzepte bereits durch seine Benutzeroberfläche vermittelt, wodurch selbst Szenarien ohne didaktischen Hintergrund einen Lerneffekt implizieren.

Zudem erreichen die Entwickler durch diese Eigenschaft ebenfalls ihr Ziel, die Zielgruppe für die Informationstechnik zu begeistern. Denn Menschen unterscheiden sich. Aufgrund unterschiedlicher Charakter-Eigenschaften, Interessen und sozialer Hintergründe ist es nur logisch, dass nicht jedes Szenario jeden Menschen ansprechen kann. Die flexible Gestaltung der Szenarien bietet die Chance, Gruppen oder Individuen basierend auf ihren Interessen anzusprechen. Somit ist Greenfoot ein universelles Werkzeug, welches sich bei einer Vielzahl von Interessengruppen anwenden lässt.

Kapselung von komplexen Funktionalitäten

Die Entwickler setzen auf eine API, die komplexe Funktionalitäten möglichst minimalistisch bündelt. Dadurch gelingt es den Greenfoot-Entwicklern, komplexe Funktionalitäten wie zum Beispiel die grafischen Darstellungen mitsamt von Animationen, so zu kapseln, dass Einsteiger problemlos auf

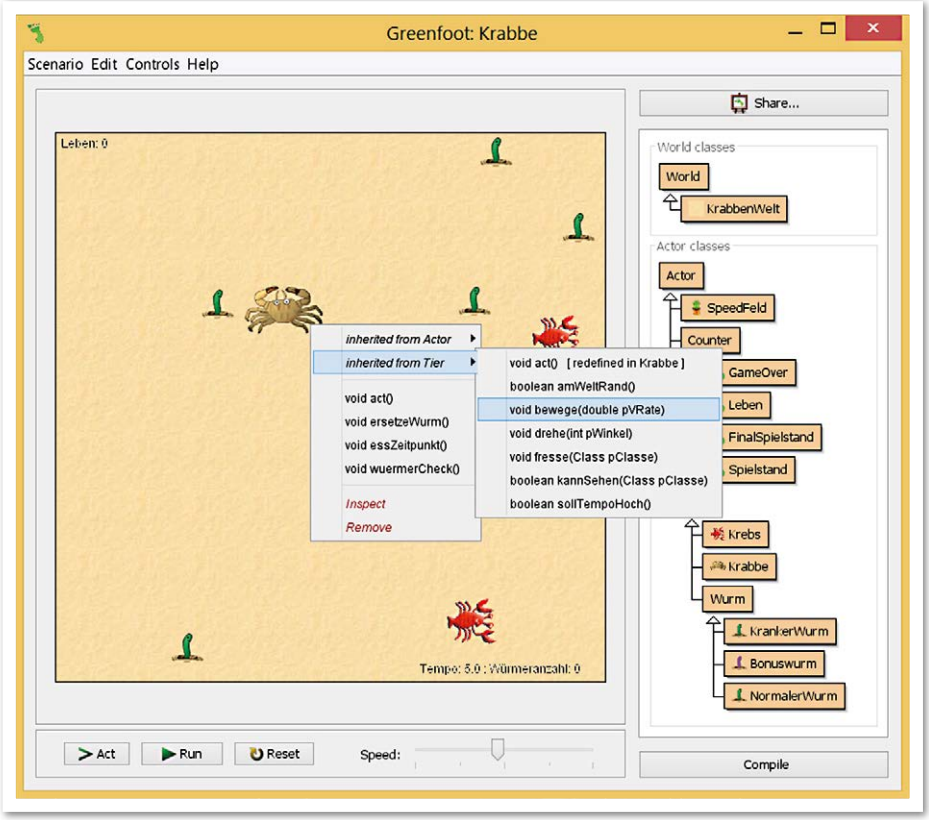


Abbildung 1: Visualisierung und direkte Interaktion

sie zugreifen können. Dadurch stehen komplexe Funktionalitäten auch Einsteigern zur Verfügung.

Bemerkenswert ist außerdem die ausführliche Dokumentation des API. Die Nutzer werden somit geleitet, dieses aktiv in erster Instanz zu nutzen, um ihre Probleme zu lösen. Zur Dokumentation eigener Programme kann eine Dokumentation in HTML-Form für das eigene Szenario direkt über die Benutzeroberfläche von Greenfoot generiert werden.

Motivation im Zeitalter von sozialen Netzwerken

Im Zeitalter der sozialen Netzwerke prägt der Vorgang des Teilens vermehrt die Gewohnheiten ihrer Nutzer. Durch das Teilen wird den Nutzern die Chance geboten, ihren Content oder den Content anderer publik zu machen. Diese Funktionalität bietet die Chance, dass der von ihnen bereitgestellte Content auch gesichtet, bewertet und diskutiert wird. Um also die für die Erlernung einer objektorientierten Programmiersprache benötigte Motivation zu stützen, bietet sich im Rahmen der Zielgruppe eine Funktionalität des Teilens an.

Greenfoot-Szenarien sind selbst allerdings nur in einem Greenfoot-Container lauffähig, da die Entwickler durch die Benutzeroberfläche bestimmte Konzepte der objektorientierten Entwicklung fördern – andere, komplexere zunächst aber auch verbergen. Die Entwickler haben in diesem Rahmen drei mögliche Varianten zum Teilen der eigenen Szenarien bereitgestellt.

Variante eins sieht vor, dass das Szenario als .jar Datei exportiert wird und somit beliebig verteilt wird. Diese Exportfunktion eignet sich besonders für Präsentationen des Szenarios, die nicht auf Quellcode-Ebene ausufern, sondern sich lediglich auf die Funktionalität des Szenarios fokussieren sollen. Ein Einsatz im Unterrichtskontext ist durchaus denkbar. Die zweite – nicht mehr zeitgemäße – Variante sieht vor, dass das Szenario als Java-Applet exportiert und somit auf Webseiten verfügbar gemacht werden könnte.

Die dritte Variante bietet jedoch den eigentlichen Kern der Implementierung der Funktionalität des Teilens innerhalb Greenfoots. Nach einer Registrierung besteht die Möglichkeit, seine Szenarien innerhalb der Greenfoot-Community zu veröffentlichen. Diese können nicht nur getestet und heruntergeladen werden, sie lassen sich auch bewerten und diskutieren. Durch das Vorhandensein einer aktiven Online-Community lässt sich ein aktiver Wissensaustausch feststellen.

Nutzbarkeit des erlernten Wissens

Greenfoot bietet die Chance, Erlerntes praktisch umzusetzen. Nachdem erste Fortschritte erzielt worden sind, können auch speziellere Schnittstellen wie Microsoft Kinect einbezogen werden. Auch eine Fokussierung auf die Programmierung für mobile Endgeräte mittels des Ablegers Droidfoot für das Android Betriebssystem ist möglich. Dadurch können die Nutzer in ihrer Begeisterung für die Informationstechnik weiter bestärkt werden.

Der ausschlaggebende Grund für diese Erweiterbarkeit ist die Verwendung von Java als Programmiersprache. Innerhalb Greenfoots lässt sich auf den vollen Funktionsempfang von Java zurückgreifen, wodurch bereits erstes Wissen über das Java-API aufgebaut werden kann. Daher gibt es eine Vielzahl an Entwicklungsumgebungen, die als Zwischenstufe zu professionelleren Varianten nach dem Erlernen der grundlegenden Konzepte mittels Greenfoot eingesetzt werden können. Diese vermitteln auch weiterführende Konzepte, erfüllen jedoch bewusst von den Entwicklern Greenfoots eingesetzte Prinzipien – wie die Kombination von direkter Interaktion und graphischer Visualisierung – nicht mehr.

Fazit

Greenfoot füllt eine Nische innerhalb der Menge der Entwicklungsumgebungen. Da junge Menschen immer früher mit der Programmierung in Kontakt kommen, empfiehlt es sich, bei ihnen von Beginn an ein grundlegendes Verständnis der Objektorientierung zu entwickeln. Um diesen Prozess möglichst effizient und dennoch diskret zu gestalten, wurde Greenfoot entwickelt. Von Greenfoot ist ein direkter Umstieg auf eine professionelle Entwicklungsumgebung möglich; als hilfreiche Zwischenstufe empfiehlt sich dabei die mit Greenfoot verwandte Entwicklungsumgebung „BlueJ".

Greenfoot setzt auf eine Reihe von Eigenschaften, die es in seiner Nischenfunktion bestärken. Durch die Gestaltung der Oberfläche werden viele objektorientierte Konzepte den Nutzern diskret beigebracht. Der Einsatz von Visualisierung sowie die Möglichkeit zur direkten Interaktion erzeugt bei den Nutzern die Begeisterung für die Programmierung und somit auch für die Informationstechnik. Diese wird noch bestärkt durch die Integration in eine aktive Online-Community, die gleichzeitig Lernenden sowie Lehrenden eine Vielzahl von Interaktionsmöglichkeiten und Wissensaustausch bietet. Greenfoot-Nutzer profitieren ebenfalls durch die Vielfalt an Szenarien, die es ermöglicht, andere Nutzer individuell anzusprechen.

Zusammenfassend lässt sich sagen, dass all diese Eigenschaften entweder die Vermittlung fachlichen Interesses oder die Vermittlung von Begeisterung fördern. Die Fokussierung auf diese beiden Aspekte entspricht der Intention der Entwickler und etabliert Greenfoot als einen validen Einstieg in die objektorientierte Programmierung.

Quellen

1. Poul Henriksen, Michael Kölling, greenfoot: combining object visualisation with interaction, Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications, October 24-28, 2004, Vancouver, BC, Canada
2. Michael Kölling , Poul Henriksen, Game programming in introductory courses with direct state manipulation, Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education, June 27-29, 2005, Caparica, Portugal
3. http://www.greenfoot.org
4. http://www.greenfoot-center.de

Dennis Nolte
dennis_nolte@gmx.de

Dennis Nolte belegt derzeit ein duales Studium der Wirtschaftsinformatik. Während seiner Schulzeit durfte er den Einsatz von Greenfoot im Unterricht selbst erleben und hat währenddessen eine Webseite über Greenfoot administriert.

http://ja.ijug.eu/15/2/14

The Best Way to Write SQL in Java

jOOQ – ein alternativer Weg, mit Java und SQL zu arbeiten

Lukas Eder, Data Geekery GmbH

In Java gibt es kein Standard-API, das die Ausdrucksstärke und Mächtigkeit von SQL direkt unterstützt. Alle Aufmerksamkeit ist auf objekt-relationales Mapping und andere höhere Abstraktionslevel gerichtet, beispielsweise OQL, HQL, JPQL, CriteriaQuery. jOOQ ist ein dual-lizenziertes Open-Source-Produkt, das diese Lücke füllt. Es implementiert SQL als typsichere domänen-spezifische Sprache direkt in Java und ist eine gute Wahl für Java-Applikationen, in denen SQL und herstellerspezifische Datenbankfunktionalität wichtig sind. Es zeigt, wie eine moderne domänen-spezifische Sprache die Entwicklerproduktivität stark erhöhen kann, indem SQL direkt in Java eingebettet ist.

Java feiert in diesem Jahr seinen 20. Geburtstag. Mit der JDK 1.1 wurde im Jahr 1997 das erste Mal Java Database Connectivity (JDBC) mitgeliefert und auch heute noch mit der JDK 1.8 stellt JDBC die Standardschnittstelle zwischen Java und SQL beziehungsweise NoSQL-Datenbanken dar. JDBC wurde sehr stark von Microsoft Open Database Connectivity (ODBC) inspiriert – ein sehr generisches und populäres Low-Level-API, das zu einer Zeit entwickelt wurde, als SQL sich selber erst gerade mit SQL-86, SQL-89 und SQL-92 als ISO/IEC Standard gegenüber alternativen Abfragesprachen durchgesetzt hatte.

SQL hat seitdem sehr viele Weiterentwicklungen gesehen; JDBC ist hingegen immer noch gleich geblieben. Auch nach all den Jahren schreibt man immer noch Code wie in *Listing 1*, in dem sich sechs typische Bugs eingeschlichen haben. Diese sind:

- *Zeile 04*
  Syntaxfehler, wenn „isAccount == false“ wegen eines fehlenden Leerzeichens vor dem „FROM“-Schlüsselwort
- *Zeile 07*
  Syntaxfehler und SQL-Injection sind möglich, falls die „type“-Variable unkontrollierte Benutzereingabewerte darstellt
- *Zeile 08*
  Es wird der falsche Bind-Index verwendet, falls „isAccount == false“, da die Variable aus Zeile 03 dann nicht existiert
- *Zeile 14*
  Der Spaltenname stimmt so nicht, da die Spalte in Zeile 02 auf „TXT“ umbenannt wurde
- *Zeile 15*
  Die „Clob“-Ressource wird nicht explizit mittels „Clob.free()“ freigegeben. Diese Änderung in JDBC 4.0 ist nur wenigen Programmierern bekannt und kann zu exzessivem Memory-Verbrauch führen.
- *Zeile 18*
  Die Ressourcen werden nicht sachgemäß geschlossen, was ebenfalls zu Ressourcenlecks führen kann. Dieses Problem wurde glücklicherweise mit Java 7 für die meisten Fälle adressiert.

Diese sechs Fehler entstehen sehr oft, wenn man mit JDBC dynamisches SQL implementiert. Jeder Entwickler, der Wartungsarbeiten an einem solchen Statement ausführen muss, riskiert mit jeder Änderung weitere Fehler einzuführen.

```
01: PreparedStatement stmt = connection.prepareStatement(
02:  "SELECT p.text txt" +
03:  (isAccount ? ", NVL(a.type, ?) " : "") +
04:  "FROM products p " +
05:  (isAccount ? " INNER JOIN accounts a USING (prod_id) " : "") +
06:  " WHERE p.cust_id = ? AND p.value < ?" +
07:  (isAccount ? " AND a.type LIKE '%" + type + "%'" : "");
08: stmt.setInt(1, defaultType);
09: stmt.setInt(2, custID);
10: stmt.setBigDecimal(3, BigDecimal.ZERO);
11: ResultSet rs = stmt.executeQuery();
12:
13: while (rs.next()) {
14:  Clob clob = rs.getClob("TEXT");
15:  System.out.println(clob.getSubString(1, (int) clob.length());
16: }
17:
18: rs.close();
19: stmt.close();
```

Listing 1

Die Geschichte von Java und SQL

Die logische Weiterentwicklung von Datenbank-Zugriffen in Java war Enterprise JavaBeans (EJB), später hat sich dann auch Java Persistence API (JPA) gegenüber der Konkurrenz (beispielsweise Java Data Objects) durchgesetzt. Der wichtigste Fortschritt von JPA gegenüber JDBC besteht darin, dass nicht mehr SQL geschrieben wird, sondern direkt ein Modell des Objektgraphen erstellt wird, das in etwa dem darunterliegenden relationalen Modell entspricht.

Mit diesem Ansatz kann der Objektgraph direkt in Java-Objekten materialisiert werden, ohne lange über die dafür benötigten SQL-Abfragen nachdenken zu müssen, was sich vor allem in schreibintensiven Applikationen bemerkbar macht, da weder die korrekte Reihenfolge, noch die Natur der Befehle („INSERT" oder „UPDATE"?) ausprogrammiert werden muss.

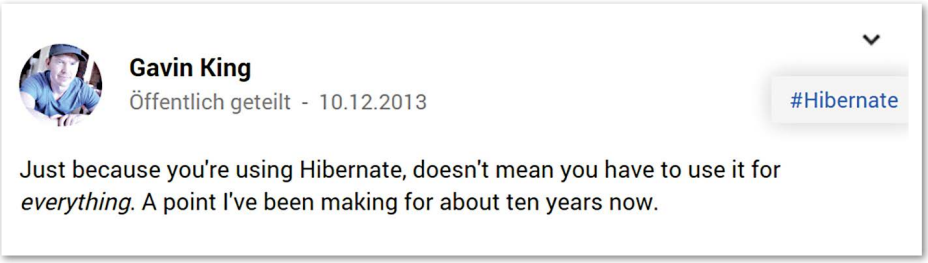


Abbildung 1: Gavin King auf Google +

```
| ID   | VALUE_DATE | AMOUNT |    BALANCE |
|------|------------|--------|------------|
| 9997 | 2014-03-18 |  99.17 |   19985.81 |
| 9981 | 2014-03-16 |  71.44 |   19886.64 |
| 9979 | 2014-03-16 | -94.60 |   19815.20 |
| 9977 | 2014-03-16 |  -6.96 |   19909.80 |
| 9971 | 2014-03-15 | -65.95 |   19916.76 |
```

Listing 2

```
| ID   | VALUE_DATE | AMOUNT |    BALANCE |
|------|------------|--------|------------|
| 9997 | 2014-03-18 |  99.17 |   19985.81 |
| 9981 | 2014-03-16 | +71.44 |  =19886.64 | n
| 9979 | 2014-03-16 | -94.60 |  +19815.20 | n + 1
| 9977 | 2014-03-16 |  -6.96 |   19909.80 |
| 9971 | 2014-03-15 | -65.95 |   19916.76 |
BALANCE(ROWn) = BALANCE(ROWn+1) + AMOUNT(ROWn)
BALANCE(ROWn+1) = BALANCE(ROWn) - AMOUNT(ROWn)
```

Listing 3

```
SUM(t.amount) OVER (
  PARTITION BY t.account_id
  ORDER BY     t.value_date DESC,
               t.id         DESC
  ROWS BETWEEN UNBOUNDED PRECEDING
         AND   1      PRECEDING
)
```

Listing 4

```
| ID   | VALUE_DATE |  AMOUNT |    BALANCE |
|------|------------|---------|------------|
| 9997 | 2014-03-18 | -(99.17)|  +19985.81 |
| 9981 | 2014-03-16 | -(71.44)|   19886.64 |
| 9979 | 2014-03-16 |-(-94.60)|   19815.20 |
| 9977 | 2014-03-16 |   -6.96 |  =19909.80 |
| 9971 | 2014-03-15 |  -65.95 |   19916.76 |
```

Listing 5

Der Nachteil von diesem Ansatz wird oft fälschlicherweise als objekt-relationaler „Impedanz Mismatch" bezeichnet, denn es handelt sich viel eher um einen Mismatch zwischen Objektorientierung und SQL selbst. Häufig will man sich nämlich als Entwickler nicht mit dem Objektgraphen auseinandersetzen, wie beispielsweise Gavin King, der ursprüngliche Entwickler von Hibernate, sagt *(siehe Abbildung 1)*.

SQL entwickelt sich in eine andere Richtung als JPA

SQL hat sich in vielen weiteren Versionen des Standards weiterentwickelt:

- *SQL-1999*
  Common Table Expressions, Grouping Sets, user-defined types, Boolean types
- *SQL-2003*
  Fensterfunktionen, XML-Unterstützung, Sequenzen und Identities, MERGE-Befehl
- *SQL-2008*
  Partitionierte JOINs, Blättern mit OFFSET
- *SQL-2011*
  Versionierte Tabellen, Zeitperioden

Neben diesen größeren Weiterentwicklungen gibt es noch viele kleinere Funktionalitäten, insbesondere aus dem OLAP-Bereich, die nur sehr wenig mit JPA und mit Objektgraph-Persistenz zu tun haben. Diese Funktionalitäten sind aber ungemein mächtig, und können zu sehr viel einfacheren und schnelleren Programmen führen.

Der RAM-Preis fällt und fällt. Es macht deshalb immer mehr Sinn, den verteilten Systemen und NoSQL-Trends aus den letzten fünf Jahren entgegenzuwirken und stattdessen auf Multi-Core-Prozessor-Architekturen komplette Datenbanken in den Speicher zu laden und Abfragen via SQL oder via gespeicherte Prozeduren sehr nahe an den Daten selbst auszuführen. Damit spart man sich sehr schnell einigen vom CAP-Theorem verursachten Ärger.

Ein Beispiel mit Fenster-Funktionen

Am besten ist die Mächtigkeit von modernem SQL am Beispiel von Fenster-Funktionen ersichtlich, ein SQL-Feature, das ins Vokabular eines jeden Java/SQL Entwicklers gehört. *Listing 2* zeigt das Beispiel einer Datenstruktur, wobei die Spalte BALANCE eine berechnete Spalte sein soll. Es handelt sich um einen klassischen laufenden Kontostand („running total"), der beispielswei-

se in Microsoft Excel ganz einfach gemäß einer Formel berechnet werden könnte *(siehe Listing 3)*.

Natürlich könnte man diesen laufenden Kontostand in Java berechnen, doch sehr viel einfacher geht es eben mit Fenster-Funktionen, zum Beispiel mit der Summe *(siehe Listing 4)*. Diese Summe berechnet für jede Reihe aus der „SELECT"-Klausel eine Summe aller Werte, die folgende Bedingung erfüllen:

- In derselben Partition (= Gruppe) liegen, wie die aktuelle Reihe. Sie müssen denselben Wert für die Kontonummer haben („PARITION BY ...")
- Absteigend sortiert nach Valuta und Buchungs-ID („ORDER BY ...")
- Über der aktuellen Reihe liegen („ROWS BETWEEN ... AND ...")

Viel einfacher als mit Text lässt sich dieses Beispiel visuell veranschaulichen. Jeder „BALANCE"-Wert kann berechnet werden als die Differenz zwischen dem neuesten „BALANCE"-Wert und der Summe aller Betragswerte, die über dem zu berechnenden „BALANCE"-Wert liegen *(siehe Listing 5)*.

Hinweis: Fenster-Funktionen sind in allen kommerziellen Datenbanken sowie in PostgreSQL und bald auch in Firebird 3.0 unterstützt.

Was hat das mit jOOQ zu tun?

jOOQ ist ein Java-API, das sich im stark SQL-orientierten Umfeld wachsender Beliebtheit erfreut. Die Idee hinter jOOQ ist ganz einfach. Vorausgesetzt, dass SQL als Sprache in einer Software strategisch wichtig ist, dann möchten Entwickler möglichst reibungslos SQL-Befehle von beliebiger Komplexität schreiben können. Mit jOOQ können sie dies direkt und typsicher in Java machen. Angenommen der Entwickler möchte die Umfrage-Ergebnisse einer fast repräsentativen Umfrage zum beliebtesten Datenbank-API auswerten, um zu dem Resultat zu kommen, das in *Listing 6* zu sehen ist. Die Spalten „RANK" und „PERCENT" sollen wiederum berechnet werden. Dafür wird sich der Entwickler eine SQL-Abfrage überlegen *(siehe Listing 7)*. Er kann diese in Java fast „1:1" identisch schreiben *(siehe Listing 8)* oder noch besser in Scala *(siehe Listing 9)*. Die Beispiele sind tatsächlich Java- oder Scala-Code und der jOOQ -Compiler kann auch etliche syntaktische Regeln überprüfen.

jOOQ ist eine interne Domänen-spezifische Sprache

SQL selbst kann ja bereits als eine Domain-specific Language (DSL) betrachtet werden. Generell unterscheidet man zwischen externen und internen DSLs. SQL ist für Java-Entwickler eine externe DSL, da die Sprache unabhängig von Java extern interpretiert und ausgeführt wird. Viele DSLs werden aber als intern implementiert, sie werden also direkt in Java (oder einer anderen Hauptsprache) umgesetzt. Ein Beispiel für eine solche interne DSL sind verschiedene Mocking-Frameworks (wie Hamcrest) für Unit-Testing. Im Grunde könnten auch die mathematischen Operationen auf „BigInteger" und „BigDecimal" als interne DSLs betrachtet werden. *Abbildung 2* zeigt, dass fast alle Sprachen auf eine Meta-Grammatik zurückgeführt werden können. Eine solche Grammatik lässt sich relativ trivial als interne

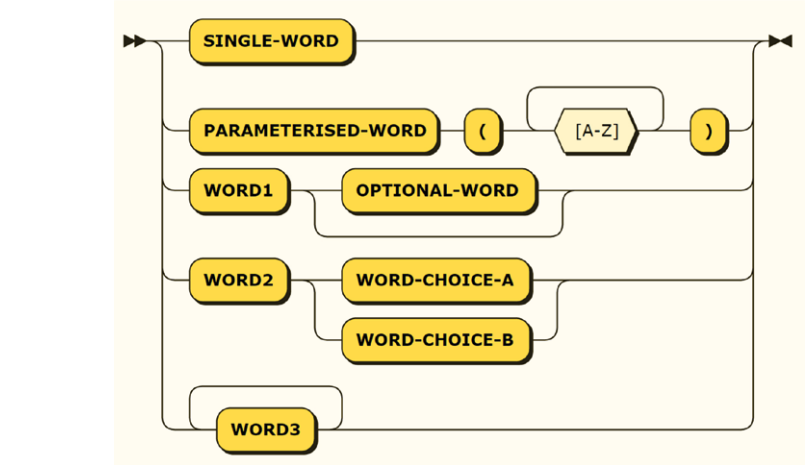


Abbildung 2: Meta-Grammatik

```
|        TEXT | VOTES |       RANK | PERCENT |
|-------------|-------|------------|---------|
|        jOOQ |  1383 |          1 |    32 % |
|   Hibernate |  1029 |          2 |    23 % |
| EclipseLink |   881 |          3 |    20 % |
|        JDBC |   533 |          4 |    12 % |
| Spring JDBC |   451 |          5 |    10 % |
```

Listing 6

```
SELECT   p.text,
         p.votes,
         DENSE_RANK() OVER (ORDER BY p.votes DESC) AS "rank",
         LPAD(
            (p.votes * 100 / SUM(p.votes) OVER ()) || ' %',
            4, ' '
         ) AS "percent"
FROM     poll_options p
WHERE    p.poll_id = 12
ORDER BY p.votes DESC
```

Listing 7

```
select (p.TEXT,
  p.VOTES,
     denseRank().over().orderBy(p.VOTES.desc()).as("rank2),
     lpad(
        p.VOTES.mul(100).div(sum(p.VOTES).over()).concat(" %"),
        4, " "
     ).as("percent"))
.from(POLL_OPTIONS.as("p"))
.where (p.POLL_ID.eq(12))
.orderBy(p.VOTES.desc());
```

Listing 8

```
select (p.TEXT,
     p.VOTES,
     denseRank() over() orderBy(p.VOTES desc) as "rank",
     lpad(
        (p.VOTES * 100) / (sum(p.VOTES) over()) || " %",
        4, " "
     ) as "percent")
from (POLL_OPTIONS as "p")
where (p.POLL_ID === 12)
orderBy (p.VOTES desc)
```

Listing 9

```
// Einstiegsschnittstelle für die DSL, könnte auch über statische Methoden
// implementiert werden
interface Start {
  End singleWord();
  End parameterisedWord(String parameter);
  Intermediate1 word1();
  Intermediate2 word2();
  Intermediate3 word3();
}

// Ausstiegsschnittstelle, könnte auch Methoden wie execute() enthalten
interface End {
  void end();
}

// Zwischenschritt, welcher die Ausstiegsschnittstelle erweitert, sowie auch
// von optionalWord() zurückgibt. Dadurch wird dieser Schritt optional
interface Intermediate1 extends End {
  End optionalWord();
}

// Zwischenschritt, der mehrere Möglichkeiten anbietet
interface Intermediate2 {
  End wordChoiceA();
  End wordChoiceB();
}

// Zwischenschritt, der sich selber zurückgibt beim Aufruf auf word3(), was
// Wiederholungen ermöglicht. Wiederholungen können jederzeit beendet werden
// da die Ausstiegsschnittstelle implementiert wird.
interface Intermediate3 extends End {
  Intermediate3 word3();
}
```

Listing 10

```
SELECT   customer.first_name, customer.last_name
FROM     customer
UNION
SELECT   staff.first_name, staff.last_name
FROM     staff
```

Listing 11

```
select (CUSTOMER.FIRST_NAME, CUSTOMER.LAST_NAME)
.from (CUSTOMER)
.union (
     select (STAFF.FIRST_NAME, STAFF.LAST_NAME)
     .from  (STAFF)
)
```

Listing 12

```
select (CUSTOMER.FIRST_NAME, CUSTOMER.LAST_NAME)
.from (CUSTOMER)
.union (
// ^^^^^ hier entsteht ein Fehler. Der Parameter ist nicht vom Typ
// Select<? extends Record2<String, String>>, sondern vom Typ
// Select<Record1<String>>
     select (STAFF.FIRST_NAME)
     .from  (STAFF)
```

Listing 13

DSL implementieren *(siehe Listing 10)*. jOOQ geht hier noch einiges weiter und verwendet auch andere Java-Sprachtools wie Varargs, Methoden-Überladen und sehr viele Generics.

Starke Typsicherheit

Wer viel in SQL oder gar in PL/SQL, in Transact-SQL oder in anderen prozeduralen Dialekten programmiert, dem ist geläufig, dass SQL eine sehr typsichere Sprache ist. Mittels „SELECT"-Befehlen lassen sich Ad-hoc-Tabellenausdrücke erstellen, deren Reihentyp sehr wohldefiniert ist.

Moderne Sprachen wie Scala oder C# kennen ebenfalls generisch typisierte Tupel, also namenlose Datenstrukturen mit „1-n"-Feldern unterschiedlichen Typs. Das Beispiel in *Listing 11* veranschaulicht diese Typsicherheit anhand einer „UNION"-Klausel und vereint zwei Tupel-Mengen in einer einzigen Menge. *Listing 12* zeigt, wie der Befehl in jOOQ aussieht.

Dabei enthält der Typ, den der „select()"-Aufruf zurückgibt, bereits alle benötigten Typ-Informationen, um später bei „union()" den entsprechenden Typ-Check zu machen: Ein Typ, der mit „Record2<String, String>" parametrisiert ist. Die „select()"-Methode erwartet nun einen Parameter vom Typ „Select<? extends Record2<String, String>>". Dies führt dazu, dass der Java-Compiler – im Unwissen was er gerade verarbeitet – folgende SQL-Befehle mit Fehlermeldungen verwirft: „Falsche Anzahl von Feldern in der SELECT-Klausel" *(siehe Listing 13)* und „Falsche Typen der Felder in der SELECT-Klausel" *(siehe Listing 14)*.

Typsicheres „IN"-Prädikat

Die Typsicherheit kann in allem möglichen SQL-Befehlen oder Klauseln zur Anwendung kommen, auch bei einem „IN"-Prädikat *(siehe Listing 15)*. In diesem Beispiel erwartet die „in()"-Methode wiederum einen Parameter vom Typ „select<? extends Record1<String>>", sodass die Anwendung einen Fehler bei der Kompilierung generiert *(siehe Listing 16)*.

Typsicherer „INSERT"-Befehl

Auch der „INSERT"-Befehl ist insofern typsicher, dass die Anzahl und die Typen der Werte in der „INSERT"-Liste der Anzahl und den Typen der Werte in der „VALUES"-Klausel entsprechen müssen *(siehe Listing 17)*. *Listing 18* zeigt den Befehl, der nicht kompiliert wird.

Datenbank zuerst

Eingangs wurde die Geschichte von Java und SQL erörtert. Dabei hat sich gezeigt, dass JPA

```
select (CUSTOMER.FIRST_NAME, CUSTOMER.LAST_NAME)
.from (CUSTOMER)
.union (
// ^^^^^ hier entsteht ein Fehler. Der Parameter ist nicht vom Typ
// Select<? extends Record2<String, String>>, sondern vom Typ
// Select<Record2<String, Date>>
     select (STAFF.FIRST_NAME, STAFF.DATE_OF_BIRTH)
     .from  (STAFF)
)
```

Listing 14

```
select ()
.from (CUSTOMER)
.where (CUSTOMER.FIRST_NAME.in(
  select (STAFF.FIRST_NAME).from (STAFF)
))
```

Listing 15

```
select ()
.from (CUSTOMER)
.where (CUSTOMER.FIRST_NAME.in(
// hier entsteht ein Fehler ^^^. Der Parameter ist nicht vom Typ
// Select<? extends Record1<String>>, sondern vom Typ
// Select<Record2<String, String>>
  select (STAFF.FIRST_NAME, STAFF.LAST_NAME).from (STAFF)
))
```

Listing 16

```
insertInto (CUSTOMER, CUSTOMER.FIRST_NAME, CUSTOMER.LAST_NAME)
.values    (          "John"             , "Doe"             )
```

Listing 17

```
insertInto (CUSTOMER, CUSTOMER.FIRST_NAME, CUSTOMER.last_name)
.values    (          "John"                                  )
// ^^^^^^^^^ Falsche Anzahl an Parametern. Erwartet wurden: String, String
```

Listing 18

```
// Alle Tabellenreferenzen werden hier aufgelistet:
class Tables {
    Customers CUSTOMER = new Customer();
    Staff STAFF = new Staff();
}

// Alle Tabellen werden durch individuelle Klassen repräsentiert,
// jeweils mit Spalten drin:
class Customer {
    final Field<String> FIRST_NAME = ...
    final Field<String> LAST_NAME = ...
}
```

Listing 19

```
// Der CustomerRecord wurde ebenfalls vom Code Generator generiert:
CustomerRecord customer =
DSL.using(configuration)
  .selectFrom(CUSTOMER)
  .where(CUSTOMER.ID.eq(3))
  .fetchOne();

// Ändern und speichern des Kundenrecords.
customer.setAddress(newAddress);
customer.store();
```

Listing 20

einen ganz anderen Weg gegangen ist als SQL – einen Java-zentrischen Weg. Ein typischer Anwendungsfall für JPA entsteht, wenn die Java-Applikation im Vordergrund steht und die Persistenz-Schicht im Hintergrund. Die Java-Applikation könnte auch in ganz andere Persistenz-Anbieter schreiben wie NoSQL-Datenbanken, File-Systeme, LDAP etc.

In vielen komplexeren und langlebigen System-Architekturen steht aber die Datenbank im Mittelpunkt, und es gibt zum Beispiel verschiedene Applikationen (Java wie Nicht-Java), die auf eine solche Datenbank zugreifen. So. könnten diese Applikationen Web-Applikationen, Administrations-Konsolen, ETL-Skripte, Monitoring-Tools etc. sein.

Komplexe Systeme entwickeln sich oft um die Datenbank herum, da die Daten alle anderen System-Teilnehmer überleben. Noch heute werden COBOL- oder Delphi-Applikationen durch Java-Applikationen ersetzt – die Datenbank aber bleibt. In diesen Szenarien eignet sich SQL und damit jOOQ besonders gut. Entsprechend wurde jOOQ mit einer „Datenbank zuerst"-Philosophie entwickelt, die sich neben den bisher aufgeführten SQL-Funktionalitäten auch noch zeigt.

Der Code-Generator

Die hier gezeigten Beispiele verwenden generierten Quellcode, der von jOOQ aus den Meta-Informationen der Datenbank bereitgestellt wird. So entsteht also für jede Tabelle eine Klasse und für jede Spalte ein Attribut in der Klasse. Ein vereinfachtes Beispiel würde wie in *Listing 19* aussehen. Der Code Generator kann dann fix im Entwicklungsprozess eingebunden und zum Beispiel nach einer Flyway-Datenbank-Migration sofort angestoßen werden. Die generierten Meta-Informationen gewährleisten, dass jede Änderung am Datenbank-Schema sich sofort in Compiler-Fehler niederschlägt, etwa bei einer Umbenennung oder Entfernung einer Spalte.

Die Performance

Je mehr sich die Datenbank in den Mittelpunkt einer Architektur verschiebt, desto wichtiger wird der Faktor „Performance" auf dieser Datenbank. Mit JPA ist es praktisch unmöglich, die generierten SQL-Befehle zu optimieren – nicht zuletzt wegen der „N+1"-Problematik, die sich beim Navigieren des Objekt-Graphen nicht verhindern lässt. SQL – und damit jOOQ – beeinflussen die Performance sehr feingranular.

Active Records

Die meisten Schreib-Operationen sind triviale „INSERT"- oder „UPDATE"-Befehle. Was mit JPA sehr einfach ist, funktioniert auch auf einer weniger komplexen Ebene mit sogenannten „Active Records", also Records, die eine Datenbank-Verbindung aufrechterhalten und alle Änderungen wieder zurückspeichern können. *Listing 20* zeigt dazu ein Beispiel. Mit diesem API ist auch einfaches optimistisches Locking auf den Records möglich.

```
-- MySQL, H2, HSQLDB, Postgres und SQLite
SELECT * FROM BOOK LIMIT 1 OFFSET 2

-- CUBRID unterstützt eine MySQL Variante
der LIMIT .. OFFSET Klausel
SELECT * FROM BOOK LIMIT 2, 1

-- Derby, SQL Server 2012, Oracle 12c,
und der SQL:2008 Standard
SELECT * FROM BOOK OFFSET 2 ROWS FETCH
NEXT 1 ROWS ONLY

-- Informix hat SKIP .. FIRST Support
SELECT SKIP 2 FIRST 1 * FROM BOOK

-- Ingres (fast der SQL:2008 Standard)
SELECT * FROM BOOK OFFSET 2 FETCH FIRST 1
ROWS ONLY

-- Firebird
SELECT * FROM BOOK ROWS 2 TO 3

-- Sybase SQL Anywhere
SELECT TOP 1 ROWS START AT 3 * FROM BOOK

-- DB2 (fast der SQL:2008 Standard, ohne
OFFSET)
SELECT * FROM BOOK FETCH FIRST 1 ROWS
ONLY

-- Sybase ASE, SQL Server 2008 (ohne
OFFSET)
SELECT TOP 1 * FROM BOOK
```

Listing 21

```
-- DB2 (mit OFFSET), SQL Server 2008 (mit
OFFSET)
SELECT * FROM (
  SELECT BOOK.*,
    ROW_NUMBER() OVER (ORDER BY ID ASC)
AS RN
  FROM BOOK
) AS X
WHERE RN > 1
AND RN <= 3

-- DB2 (mit OFFSET), SQL Server 2008 (mit
OFFSET)
SELECT * FROM (
  SELECT DISTINCT BOOK.ID, BOOK.TITLE
    DENSE_RANK() OVER (ORDER BY ID ASC,
TITLE ASC) AS RN
  FROM BOOK
) AS X
WHERE RN > 1
AND RN <= 3

-- Oracle 11g und älter
SELECT *
FROM (
  SELECT b.*, ROWNUM RN
  FROM (
    SELECT *
    FROM BOOK
    ORDER BY ID ASC
  ) b
  WHERE ROWNUM <= 3
)
WHERE RN > 1
```

Listing 22

Gespeicherte Prozeduren

Nicht immer steht Java im Vordergrund einer Applikation. Um Performance zu erreichen oder um ein sauberes API zwischen der Datenbank und den Client-Applikationen umzusetzen, werden oft gespeicherte Prozeduren verwendet. Der jOOQ Code Generator generiert auch hier Klassen mit Methoden, die den Prozeduren entsprechen, um mit wenig Programmcode entsprechende Aufrufe machen zu können. Gespeicherte Funktionen können darüber hinaus auch direkt und typsicher in SQL-Befehlen verwendet werden, egal ob sie einen einzelnen Spaltenwert oder eine Tabelle zurückgeben.

SQL-Transformation und Standardisierung

Der jOOQ-Programmierer baut keine SQL-Zeichenketten mehr, sondern direkt einen Abstract Syntax Tree (AST) in Java. Dieser kann nicht nur während der Kompilierung verwendet werden, sondern auch zur Laufzeit, um etwa zu entscheiden, in welcher exakten Form ein SQL-Befehl wirklich an die Datenbank ausgegeben werden soll. Der Entwickler kann einfach „selectFrom(BOOK).limit(1).offset(2)" schreiben und jOOQ generiert das richtige SQL, je nach Datenbank-Dialekt, etwa in Datenbanken, die irgendeine Form von „limit .. offset" unterstützen *(siehe Listing 21)*, aber auch in den anderen Datenbanken *(siehe Listing 22)*. Insbesondere bei den letzteren Beispielen handelt es sich um die Art „Infrastruktur-Code", die kein Entwickler wirklich schreiben möchte (oder auf Anhieb richtig hinbekommt).

Fazit

Interne domänen-spezifische Sprachen können in Java zu sehr viel Typsicherheit führen, fast so viel wie die externe Sprache selbst implementiert. Im Fall von SQL – einer sehr typsicheren Sprache – ist dies besonders interessant.

jOOQ wurde konzipiert, um an der Java/SQL-Schnittstelle möglichst viel der bestehenden kognitiven Reibung bei Java-Entwicklern zu entfernen – vor allem bei den Entwicklern, die SQL in Java einbetten möchten, sodass sich der Java-Code genauso wie SQL-Code anfühlt. Gleichzeitig wurde jOOQ so konzipiert, dass möglichst viel Typsicherheit bereits bei der Kompilierung erstellt werden kann, sowohl in Java wie auch in Scala, Groovy etc., was die Fehleranfälligkeit reduziert und somit die Qualität und Produktivität erhöht..

jOOQ ist frei verfügbar unter der Apache Lizenz 2.0 in Verwendung mit Open-Source-Datenbanken oder unter einer kommerziellen Lizenz in Verwendung mit kommerziellen Datenbanken.

Weiterführende Informationen

1. Die jOOQ-Dokumentationsseite: http://www.jooq.org/learn
2. Java Fluent API Designer Crash-Kurs: http://blog.jooq.org/2012/01/05/the-java-fluent-api-designer-crash-course/
3. Ein Webinar mit Arun Gupta von Red Hat über jOOQ und JavaEE: https://www.youtube.com/watch?v=ZQ2Y5Z0ju3c
4. Eine Präsentation über jOOQ an der GeeCON Krakau: http://vimeo.com/99526433

Lukas Eder
lukas.eder@datageekery.com

Lukas Eder ist der Gründer und Geschäftsführer der Data Geekery GmbH, die Firma hinter jOOQ. Er ist aktiv an der Entwicklung von jOOQ beteiligt und berät Kunden zu verschiedenen Themen an der Java- und SQL-Schnittstelle sowie auch bei der PL/SQL-Entwicklung.

JBoss vs. WebLogic Server – ein Duell auf Augenhöhe?

Manfred Huber, WWK Lebensversicherung a. G.

Oracle-Kunden betreiben ihre Anwendungen auf dem WebLogic Server. Unter Kostengesichtspunkten tragen sie sich mit dem Gedanken, einen Teil oder vielleicht sogar alle Ihre Anwendungen auf JBoss zu migrieren. Welche Punkte gilt es hierbei zu beachten? Kann JBoss unter den ungleichen Startbedingungen den Vorsprung des WebLogic Servers überhaupt einholen?

Mit der Oracle-Ankündigung, den Support für GlassFish einzustellen, verbleibt der JBoss im JEE-Applikationsserver-Bereich als einzige echte Open-Source-Alternative zum Oracle WebLogic Server (OWLS). Viele Oracle-Kunden stellen sich die Frage, ob sie nicht ihre Applikationen künftig auf JBoss betreiben sollten.

Aus technischer Sicht hat JBoss mit EAP 6 einiges an Boden gut gemacht, insbesondere auch durch die Implementierung des Domänen-Modells. Auch wenn manches, wie etwa die Administrations-Konsole, noch etwas rustikal erscheint, ist EAP 6 sicherlich ein Schritt nach vorne. Schwerpunkt dieses Artikels sind aber nicht die technischen Betrachtungen, wie man sie ja bereits im Netz und der einschlägigen Literatur findet. Vielmehr stehen hier Überlegungen zu den Kosten eines Parallelbetriebs von WebLogic Server und JBoss EAP 6 beziehungsweise einer Ablösung von WebLogic Server durch JBoss EAP 6 im Vordergrund.

Als Oracle-Kunde startet man nicht auf der grünen Wiese, sondern hat bereits WebLogic Server im Einsatz. Dann genügt es natürlich nicht, einfach nur die Kosten für Lizenzen und Wartung gegenüberzustellen. Eine sinnvolle Betrachtung muss neben den technischen Aspekten auch alle durch einen Parallelbetrieb/Ablösung entstehenden Kosten betrachten. Auch die Aufwände für Schulung/Einarbeitung, für die Neu-Implementierung beziehungsweise die Umstellung von Verfahren und Skripten sowie für die Migration der Applikationen etc. sind nicht zu unterschätzen.

Es stellt sich die Frage, ob JBoss in Anbetracht all dieser Aspekte für geschäftskritische Applikationen eine echte Alternative zum WebLogic Server darstellt? Zur Beantwortung dieser Frage werden nach dem Vergleich der Lizenzbestimmungen und Lizenzpreise die zu beachtenden Aspekte anhand eines fiktiven Beispiels aufgezeigt und zwei mögliche Szenarien beschrieben:

- *Szenario 1*
  Es werden nur die Lizenz- beziehungsweise Subskriptionspreise verglichen
- *Szenario 2*
  Neben den Lizenz- beziehungsweise Subskriptionspreisen werden auch die Kosten für die Umstellung einbezogen

Den Euro-Beträgen in *Abbildung 1* liegt für den WebLogic Server die öffentlich im Netz verfügbare „Oracle Technology Global Price List" mit Stand vom 7. August 2014 zugrunde. Die unrabattierten Listenpreise in Dollar wurden im September in Euro umgerechnet. Für JBoss EAP stellte die Firma RedHat

Edition	Bruttopreise NUP			Bruttopreise Prozessor		
	Kauf	Wartung	Gesamt	Kauf	Wartung	Gesamt
OWLS SE	184 €	41 €	225 €	9.195 €	2.023 €	11.218 €
OWLS EE	459 €	101 €	560 €	22.987 €	5.057 €	28.044 €

Abbildung 1: Listenpreise für Oracle WebLogic Server

Supportmodell	Supportpreis (netto)	Supportpreis (brutto)
Standard (Support während der Geschäftszeiten) 16 Core	4.960,00 €	5.902,40 €
Standard inkl. Management und Monitoring (= JON) 16 Core	6.400,00 €	7.616,00 €
Standard (Support während der Geschäftszeiten) 64 Core	18.000,00 €	21.420,00 €
Standard inkl. Management und Monitoring (= JON) 64 Core	23.600,00 €	28.084,00 €
Premium (Support 24 x 7) 16 Core	7.200,00 €	8.568,00 €
Premium inkl. Management und Monitoring (= JON) 16 Core	9.600,00 €	11.424,00 €
Premium (Support 24 x 7) 64 Core	25.600,00 €	30.464,00 €
Premium inkl. Management und Monitoring (= JON) 64 Core	33.600,00 €	39.984,00 €

Abbildung 2: Listenpreise für JBoss EAP

freundlicherweise im September 2014 eine aktuelle Preisliste in Euro zur Verfügung *(siehe Abbildung 2)*.

Lizenz-Bestimmungen

Bei den Lizenzen gibt es zwischen RedHat und Oracle durchaus Unterschiede. Während man bei Oracle zwischen verschiedenen Editionen (Standard Edition, Enterprise Edition und Suite) und zwei Lizenzmetriken (Named User Plus oder Prozessor) wählen kann, leistet RedHat nur für die EAP-Version Support und bietet als einzige Metrik die Lizenzierung nach Cores an.

Gravierend sind die Unterschiede vor allem für Kunden, die VMware als Virtualisierungslösung einsetzen. Während man bei Oracle alle Server im ESX-Cluster komplett lizenzieren muss, hat man bei RedHat die Wahl, ob man nur einzelne virtuelle Server oder die physikalischen Server lizenziert, je nachdem, was für den Kunden vorteilhafter ist.

RedHat bietet verschiedene Support-Modelle (5 x 8 und 7 x 24; mit und ohne Management) an, die auch miteinander kombiniert werden können (beispielsweise Standard-Support für die Test-Umgebungen und Premium-Support für die Produktions-Umgebungen), wohingegen Oracle generell nur ein Support-Modell anbietet.

Last but not least ist es bei RedHat egal, welcher Prozessor-Typ in der Hardware verbaut ist, währenddessen bei Oracle bei jedem Tausch der Hardware ein Blick in die „Core Factor Table" anzuraten ist. Ansonsten besteht die latente Gefahr, mit einer neuen Hardware aufgrund eines höheren Core-Faktors plötzlich nicht mehr richtig lizenziert zu sein.

Vergleich der Lizenz-Preise

Bei den Lizenz-Preisen (siehe oben) ist festzustellen, dass RedHat mit den Subscriptionen eine Art „Mietmodell" anbietet. Aufgrund der sich im Einsatz befindlichen Instanzen kann der Kunde jährlich neu entscheiden, welche Subscription er abschließt. Dieses „Mietmodell" kennt Oracle hingegen nicht. Jede Lizenz muss zwingend gekauft werden. Support wird bei Oracle gegen eine jährliche Gebühr in Höhe von 22 Prozent des Kaufpreises angeboten. *Tabelle 1* fasst die wesentlichen Punkte nochmals zusammen.

Die Möbelhauskette „Maier & Töchter"

In der fiktiven Firma „Maier & Töchter" laufen alle wichtigen Anwendungen auf der Oracle WebLogic Server Enterprise Edition. Die Geschäftsleitung hat sich vor einigen Jahren in Abstimmung mit ihrem IT-Leiter für den Kauf von Prozessor-Lizenzen entschieden. Neben den internen Anwendungen im LAN wird durch die firmeninterne IT auch der Internet-Auftritt der Möbelhauskette einschließlich Online-Shop betrieben. Die entsprechende Infrastruktur steht in der DMZ. Alle OWLS-Instanzen werden auf virtuellen Linux-Servern (VMware) betrieben *(siehe Abbildung 3)*.

Die jährlichen Support-Zahlungen an Oracle betragen für 96 Cores beziehungsweise 48 Prozessor-Lizenzen 242.745,15 Euro. Der Seniorchef M. Maier hat auf einer IT-Konferenz einen Vortrag zum Thema „Open Source" gehört und sich auf einem Messestand bei der Firma RedHat über die Lizenz-Preise erkundigt.

Da die jährlichen Kosten für den JBoss nur bei rund einem Fünftel der Kosten des WebLogic Server liegen, hat Herr Maier seinen IT-Leiter gebeten, ein entsprechendes Projekt aufzusetzen. Der für den Applikationsserver-Bereich zuständige IT-Mitarbeiter hat seine Anforderungen in einem Storyboard zusammengefasst und ein Red-

JBoss Enterprise Application Platform (JBoss EAP)	Oracle WebLogic Server Enterprise Edition (OWLS EE)
Lizenzierung nach Cores	Lizenzierung nach Cores (Prozessoren) oder Named-User-Plus (NUP)
Lizenzierung virtueller Server entweder mit der zugeordneten Core-Anzahl oder der physikalischen Cores	Lizenzierung virtueller Server mit der zugeordneten Core-Anzahl nur bei Einsatz von Oracle VM. Bei VMWare ist immer der gesamte ESX-Cluster zu lizenzieren
Entwicklung mit 25 User je 16 Core frei (auf Server, Laptop und Workstations)	Entwicklung ist voll zu lizenzieren
Subscription jährlich neu abschließbar	Lizenzen müssen einmalig gekauft werden
Verschiedene Subscriptionsmodelle (kombinierbar s. u.)	Nur ein einziges Supportmodell
Keine Unterscheidung nach Prozessortyp	Unterscheidung nach Prozessortyp

Tabelle 1: Lizenz-Bestimmungen JBoss EAP vs. OWLS EE

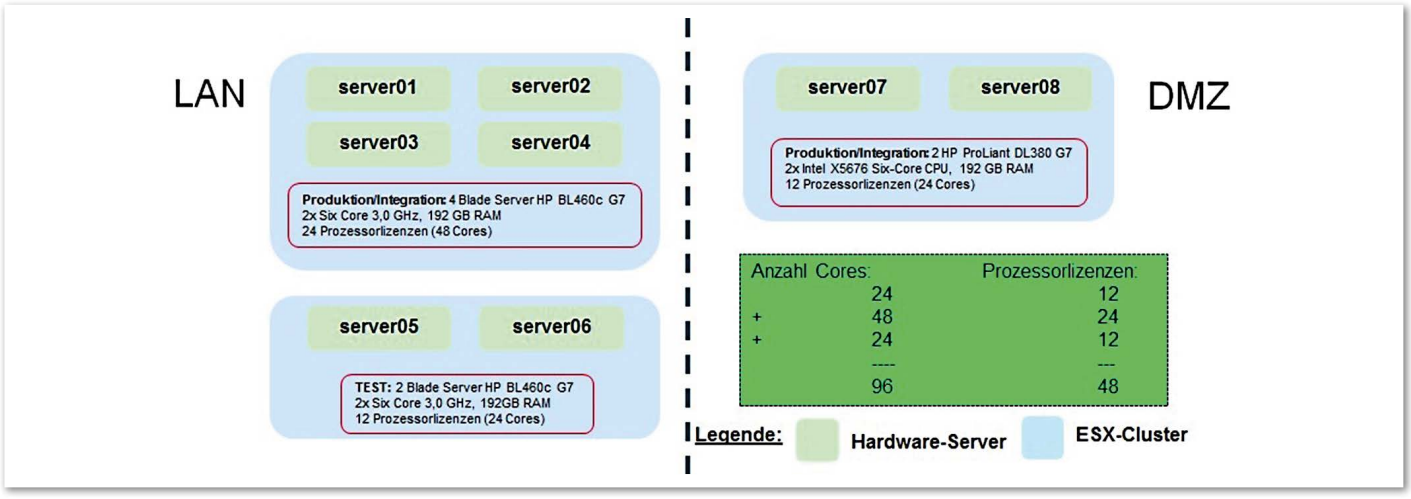


Abbildung 3: OWLS-Landschaft der Firma Maier

Lizenzmetrik	Anzahl	Supportpreis (brutto)
Standard (Support während der Geschäftszeiten) 16 Core	1	5.902,40 €
Premium inkl. Management und Monitoring (= JON) 16 Core	1	11.424,00 €
Premium inkl. Management und Monitoring (= JON) 64 Core	1	39.984,00 €
Gesamt	1	57.310,40 €

Abbildung 4: Subscriptionen JBoss EAP

Hat-Mitarbeiter war einige Tage bei der Firma Maier vor Ort, um die in dem Storyboard definierten Fälle umzusetzen, darunter:

- *Installation/Konfiguration*
  Möglichst automatisierte Installation und Konfiguration einer Domäne bestehend aus einem Domänen-Controller sowie zwei Applikationsserver-Instanzen in einem Cluster
- *Parallelbetrieb*
  Parallelbetrieb von zwei Domänen auf derselben Infrastruktur mit unterschiedlichen Versionen
- *Deployment*
  Das automatisierte Deployment inklusive automatisiertes Stoppen und Starten des Domänen-Controllers, einzelner Applikationsserver-Instanzen sowie des gesamten Clusters
- *Failover*
  Demonstration, dass die Applikation auch nach dem Ausfall einer der beiden Applikationsserver-Instanzen erreichbar ist und die Sitzungsdaten erhalten bleiben
- *Authentifizierung*
  Konfiguration der Domänen, dass die Authentifizierung von Benutzern gegen das Active Directory möglich ist
- *Überwachung/Berichtswesen*
  Erläutern eines möglichen Monitoring und Reporting
- *Entwicklungsumgebung*
  Einbindung in Eclipse, Debugging etc.

Da JBoss alle technischen Anforderungen erfüllen konnte, erarbeitet der IT-Leiter im zweiten Schritt den Projektplan für die Migration und wirft nochmals einen Blick auf die Kosten. Da für die Testumgebung die Standard-Subscription genügt, können die 96 Cores abgedeckt werden *(siehe Abbildung 4)*.

Szenario 1: Betrachtung der Lizenz- beziehungsweise Subscriptionskosten

Betrachtet man ausschließlich die Lizenz- und Subscriptionskosten ist JBoss natürlich viel günstiger als WebLogic Server. Selbst wenn die Anschaffungskosten außer Acht bleiben, lassen sich bezogen auf die Infrastruktur der Firma Maier im Zehn-Jahres-Vergleich ohne weiteres knapp zwei Millionen Euro einsparen *(siehe Abbildung 5)*. Diese Zahl lässt sich aber nur erreichen, wenn zum Support-Ende-Termin von Oracle die gesamte Umgebung auf JBoss umgestellt wird und dafür keinerlei Aufwände angesetzt werden. Da dies mit Sicherheit so nicht realistisch ist, diskutiert der IT-Leiter mit seinem Administrator und kommt zu nachfolgendem Ergebnis.

Szenario 2: Betrachtung weiterer Aspekte

Neben den Lizenz- und Subscriptionskosten müssen u. a. noch folgende Aspekte für die Planung berücksichtigt werden:

	JBoss EAP	OWLS EE	Ersparnis
Kosten (1. Jahr)	57.310,40 €	242.745,15 €	185.434,75 €
Kosten (2 Jahre)	114.620,80 €	485.490,30 €	370.869,50 €
Kosten (3 Jahre)	171.931,20 €	728.235,45 €	556.304,25 €
Kosten (5 Jahre)	286.552,00 €	1.213.725,74 €	927.173,74 €
Kosten (10 Jahre)	573.104,00 €	2.427.451,49 €	1.854.347,49 €

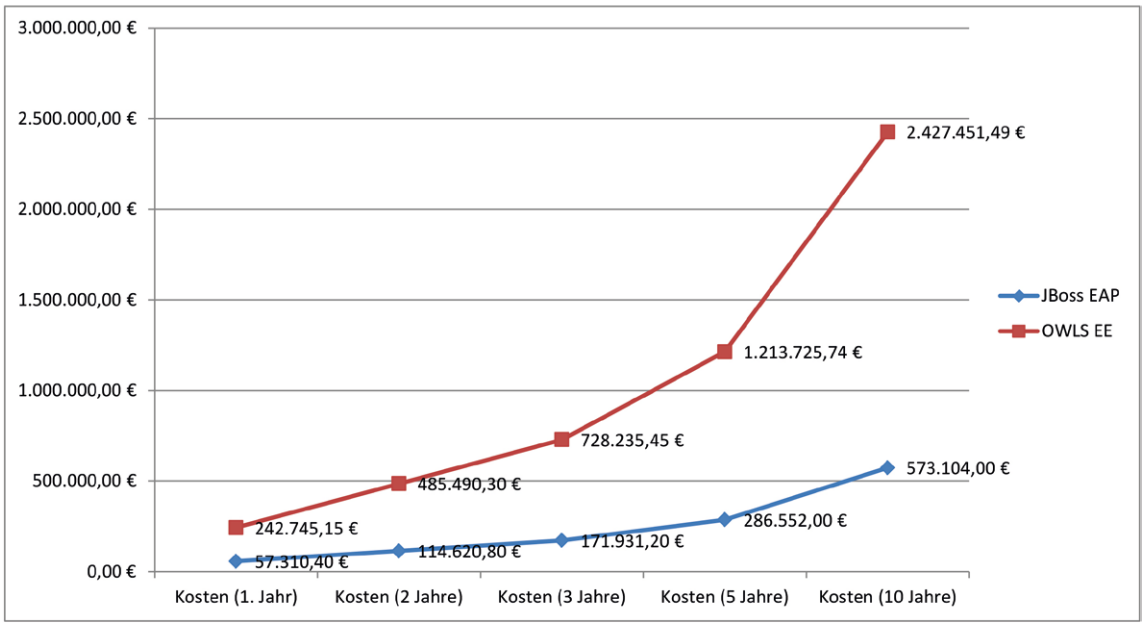


Abbildung 5: Kostenvergleich Szenario1

- Gemäß der vor einigen Jahren erlassenen IT-Dokumentations-Richtlinie sind unter anderem ein Architektur-Whitepaper, ein Administrations-Handbuch, eine System-Beschreibung sowie diverse Handbücher zur Installation und zum Betrieb der neuen Software zu erstellen
- Die Administratoren und Entwickler müssen auf JBoss geschult werden
- Für die Entwickler sind entsprechende Handbücher zu erstellen
- Regelungen für das Patchen der Umgebungen sind festzulegen
- Im Helpdesk-Tool und in der „cmdb" sind die entsprechenden Einträge zu erstellen
- Da sich die Firma Maier vor rund zwei Jahren gemäß ISO 27001 auf Basis von IT-Grundschutz hat zertifizieren lassen, ist auch ein entsprechendes Sicherheitskonzept zu erstellen. Außerdem muss der entsprechende Baustein im BSI-Grundschutztool bearbeitet werden.
- Für die Zusammenarbeit mit den anderen Abteilungen sind entsprechende Operational Level Agreements (OLAs) zu erstellen
- Weiterhin muss auch an das Monitoring gedacht werden. Bisher erfolgt die Überwachung der WebLogic-Server-Instanzen mit dem Oracle Enterprise Manager. Hier ist auch für den JBoss eine Lösung erforderlich.
- Schließlich muss auch für die Migration ein entsprechender Aufwand eingeplant werden

Aufgrund dieser Überlegungen kommt der IT-Leiter zu dem Schluss, dass eine Komplett-Ablösung des WebLogic Server durch JBoss mindestens drei Jahre in Anspruch nimmt. Den Aufwand beziffert er mit insgesamt 300.000 Euro. Da die neue Plattform sukzessive aufgebaut wird, reicht im ersten Jahr ein 16-Core-Band-Standard-Subscription von RedHat aus. Im zweiten Jahr kommen die Kosten für ein weiteres 16-Core-Band Premium-Subscription hinzu. Und erst im dritten Jahr müssen alle 96 Cores über entsprechende Core-Bänder abgedeckt werden.

Der Parallelbetrieb und der Umstellungsaufwand haben zur Folge, dass die Gesamtkosten erst einmal ansteigen. Eine Ersparnis tritt frühestens nach vier Jahren ein und auch nur unter der Voraussetzung, dass im dritten Jahr die komplette Umstellung auf JBoss gelingt. Nur wenn also alles so wie geplant umsetzbar ist, könnte die Firma Maier im Zehn-Jahres-Vergleich knapp eine Million Euro einsparen *(siehe Abbildung 6)*.

Sollte sich der Supportstrom ab dem vierten Jahr zum Beispiel nur um 50 Prozent verringern lassen, so verringert sich die Ersparnis nach zehn Jahren auf 170.000 Euro *(siehe Abbildung 7)*.

Fazit

Aus technischer Sicht wird man wohl jede JEE6-Applikation vom WebLogic Server auf JBoss migrieren können. Beim Vergleich der Kosten darf man aber nicht den Fehler machen, nur die Wartungs- (Oracle) mit den Subscriptionskosten (RedHat) zu vergleichen. Der Aufbau einer neuen Plattform und die Migration der Anwendungen sind ebenfalls mit Kosten verbunden, die unbedingt berücksichtigt werden müssen.

Je nachdem, wie hoch diese Kosten angesetzt werden, fällt die Ersparnis höher

	JBoss EAP	Umstellungsaufwand	OWLS EE	Gesamt	Ersparnis
Kosten (1. Jahr)	5.902,40 €	200.000,00 €	242.745,15 €	448.647,55 €	-205.902,40 €
Kosten (2 Jahre)	17.326,40 €	50.000,00 €	485.490,30 €	552.816,70 €	-67.326,40 €
Kosten (3 Jahre)	57.310,40 €	50.000,00 €	728.235,45 €	835.545,85 €	-107.310,40 €
Kosten (4 Jahre)	114.620,80 €		970.980,60 €	950.166,65 €	20.813,95 €
Kosten (5 Jahre)	171.931,20 €		1.213.725,74 €	1.122.097,85 €	91.627,90 €
Kosten (10 Jahre)	458.483,20 €		2.427.451,49 €	1.408.649,85 €	1.018.801,64 €

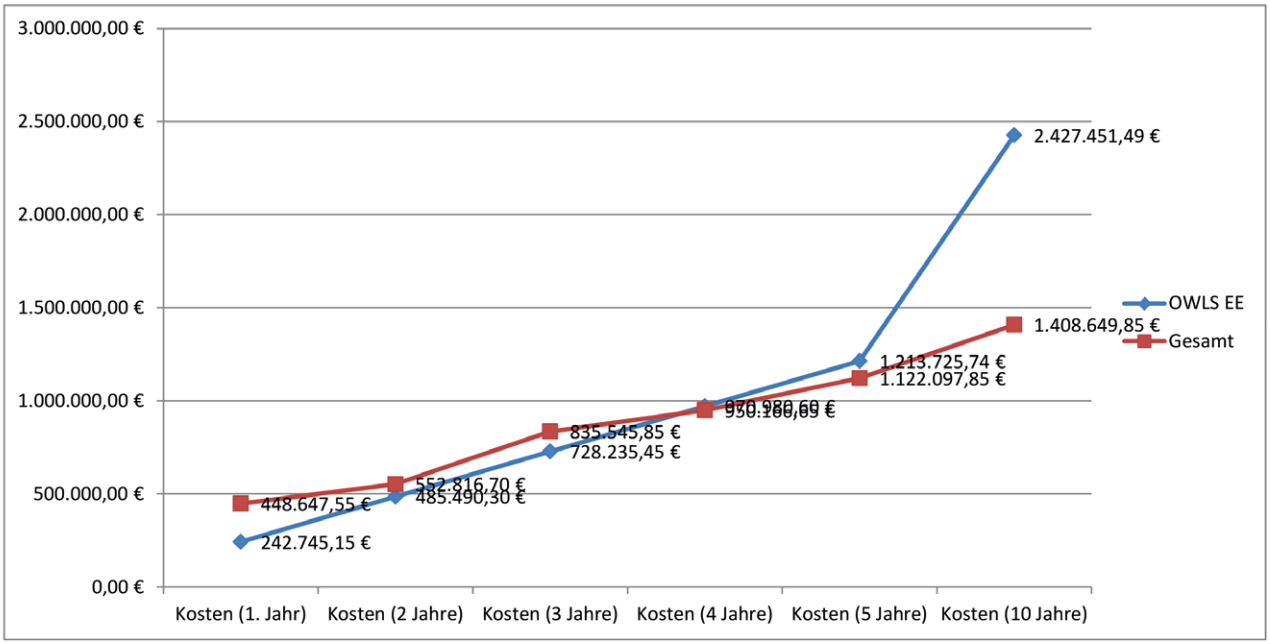


Abbildung 6: Kostenvergleich Szenario2

	JBoss EAP	Umstellungsaufwand	OWLS EE	Gesamt	Ersparnis
Kosten (1. Jahr)	5.902,40 €	200.000,00 €	242.745,15 €	448.647,55 €	-205.902,40 €
Kosten (2 Jahre)	17.326,40 €	50.000,00 €	485.490,30 €	552.816,70 €	-67.326,40 €
Kosten (3 Jahre)	57.310,40 €	50.000,00 €	728.235,45 €	835.545,85 €	-107.310,40 €
Kosten (4 Jahre)	114.620,80 €	121.372,57 €	970.980,60 €	1.071.539,22 €	-100.558,63 €
Kosten (5 Jahre)	171.931,20 €	121.372,57 €	1.213.725,74 €	1.364.843,00 €	-151.117,25 €
Kosten (10 Jahre)	458.483,20 €	606.862,87 €	2.427.451,49 €	2.258.257,87 €	169.193,62 €

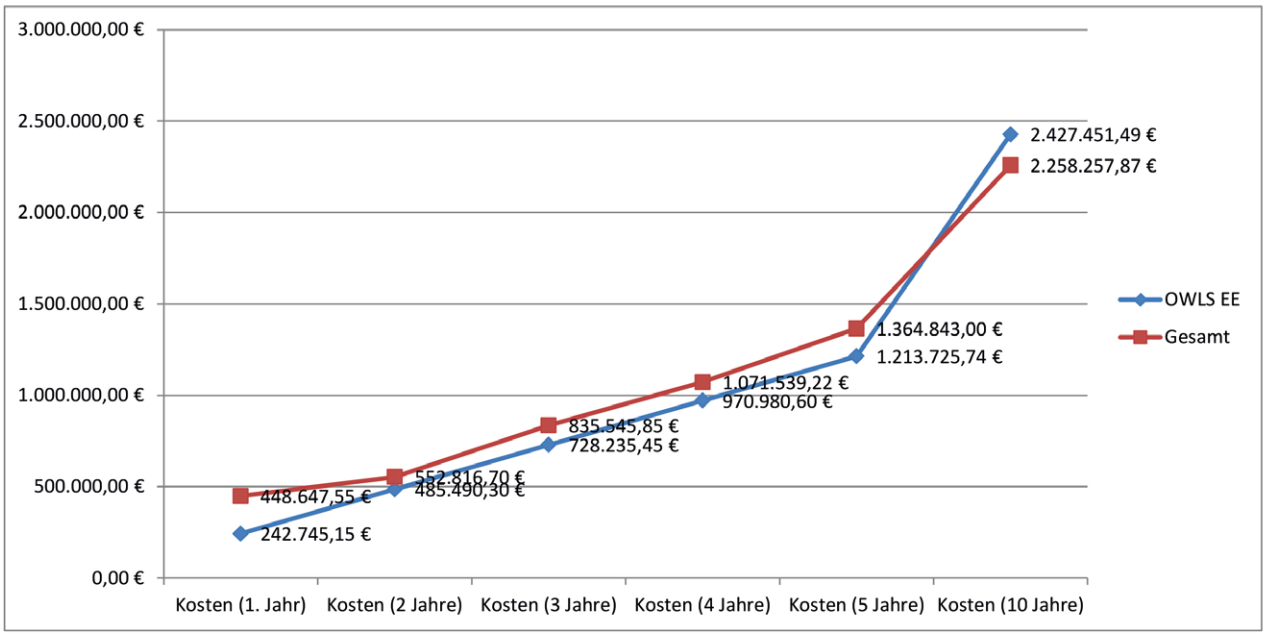


Abbildung 7: Kostenvergleich Szenario2a

oder niedriger aus. Deshalb ist es vielleicht zielführender, neben den reinen Kostengesichtspunkten auch folgende, weitere Aspekte zu berücksichtigen:

- Wie sind die Lizenzregelungen bezüglich VMware?
- Sind unterschiedliche Support-Modelle sinnvoll?
- Ist die jährliche Vertragslaufzeit interessant?
- Soll künftig eine Dual-Vendor-Strategie verfolgt werden?
- Welches Know-how besitzen die Mitarbeiter?
- Wie hoch ist der Aufwand für den laufenden Betrieb?

Aus diesen wenigen Fragen ist bereits ersichtlich, dass beim Neuaufbau einer Applikation-Server-Plattform sehr viele Aspekte zu berücksichtigen sind. Von daher kann die Frage, ob sich eine Migration vom WebLogic Server auf JBoss lohnt, nicht generell mit „ja" oder „nein" beantwortet werden. Jeder Einzelfall muss anhand der Daten im jeweiligen Unternehmen diskutiert und von Fall zu Fall auch unter Beachtung der Unternehmensstrategie bewertet werden.

Weitere Informationen

JBoss:

1. http://www.jboss.org/infinispan
2. http://en.wikipedia.org/wiki/Consistent_hashing
3. https://docs.jboss.org/author/display/ISPN/Clustering+modes#Clusteringmodes-InvalidationMode
4. http://www.jgroups.org/manual-3.x/html/index.html
5. http://www.jgroups.org/perf/perf2006/Report.html

WebLogic Server:

1. http://docs.oracle.com/middleware/1212/wls/CLUST
2. http://www.oracle.com/technetwork/database/database-cloud/private/application-continuity-wp-12c-1966213.pdf
3. http://www.oracle.com/technetwork/database/features/availability/fusion-middleware-maa-155387.html
4. http://docs.oracle.com/middleware/1212/core/ASHIA/fmwha.htm

Manfred Huber
manfred.huber@wwk.de

Manfred Huber ist seit mehr als zehn Jahren bei der WWK Lebensversicherung a. G. in München als Gruppenleiter tätig. In seinen Verantwortungsbereich fällt unter anderem der Bereich "Applikationsserver", wo derzeit mehr als zweihundert Instanzen des WebLogic Servers betrieben werden. Als Projektleiter für das Projekt „Open Source Applikationsserver" hat er sich in den vergangenen Jahren intensiv mit Alternativen zum WebLogic Server beschäftigt und war maßgeblich an der Entscheidung zugunsten JBoss beteiligt.

http://ja.ijug.eu/15/2/13

PDF-Dokumente automatisiert testen

Carsten Siedentop, pdfunit.com

Viele PDF-Dokumente sind heutzutage das Ergebnis fachlicher Abläufe. Verträge, monatliche Rechnungen aber auch Kataloge und andere Druckstücke entstehen durch individuelle Programme, die nur selten fehlerfrei sind. Da es manchmal schwierig ist, diese Programme zu testen, wäre es hilfreich, wenigstens die Korrektheit der erzeugten PDF-Dokumente zu überprüfen – nicht manuell, sondern automatisiert.

Die Anzahl der verfügbaren APIs für automatische Tests ist überschaubar. Als erstes sei PDFBox *(siehe „https://pdfbox.apache.org")* genannt, ein API für den Zugriff auf verschiedene Inhalte von PDF-Dokumenten. Es ist kein Test-Framework, bietet aber die Möglichkeit, eigene „assert()"-Methoden zu schreiben. Die aktuelle Version ist PDFBox 1.8.8 von Dezember 2014.

Zweitens gibt es das Projekt „jPdfUnit" *(siehe „http://jpdfunit.sourceforge.net")*. Es ist ein Aufsatz auf PDFBox und bietet „assert()"-Methoden für den Vergleich von Texten. Das Projekt ist inaktiv, momentan ist die Version jPdfUnit 1.2 von Dezember 2011 aktuell.

Als Drittes gibt es PDFUnit, ein Testwerkzeug, das entstand, weil die zuvor genannten APIs die benötigte Funktionalität nicht abdecken konnten. Die aktuelle Version ist 2014.06, die nächste ist für April 2015 angekündigt. Der Autor ist an der Entwicklung von PDFUnit beteiligt. Die nachfolgenden Abschnitte beschreiben die verfügbaren Funktionen und deren Verwendung.

Texte überprüfen

Das erste Code-Beispiel *(siehe Listing 1)* zeigt Validierungsmethoden auf erwartete Texte auf der ersten Seite des zu testenden PDF-Dokuments. Es wird der normale JUnit-Report erzeugt.

Die komplette API ist flüssig gebaut. Der Einstieg für alle Tests ist die statische Methode „AssertThat.document(pdfUnderTest)...". Alle umgangssprachlich benannten Methoden sind typisiert, sodass Eclipse nur die erlaubten Methoden als Line-Completion anzeigt *(siehe Abbildung 1)*. Insgesamt stehen zwölf Methoden zur Verfügung, um die Anwesenheit oder Abwesenheit von Text zu überprüfen. *Listing 2* zeigt einige davon.

```
@Test
public void hasContentOnFirstPage() {
  String filename = "documentUnderTest.pdf";
  AssertThat.document(filename)
     .hasContent(ON_FIRST_PAGE)
     .startingWith("Lorem")
     .containing("ipsum")
     .endingWith("est laborum.");
}
```

Listing 1: Text auf der ersten Seite

```
// Überprüfung der An- und Abwesenheit von Text, :
.hasContent(..).containing(.., WhitespaceProcessing)
.hasContent(..).endingWith(..)
.hasContent(..).matchingComplete(.., WhitespaceProcessing)
.hasContent(..).matchingRegex(..)
.hasContent(..).notContaining(.., WhitespaceProcessing)
.hasContent(..).notMatchingRegex(..)
.hasContent(..).startingWith(..)
// und weitere Methoden
```

Listing 2: Vergleichsmethoden für Text

Bei einer automatischen Dokumenten-Erstellung können Zeilenumbrüche nicht vorhergesehen werden. Deshalb bieten manche „assert"-Methoden die Möglichkeit, die Art der Whitespace-Behandlung von außen mitzugeben („ignore", „normalize", „keep"). Bei Tests auf mehrseitigen Dokumenten muss es eine Möglichkeit geben, Seiten zu benennen. In *Listing 1* ist der Test auf die erste Seite beschränkt. *Listing 3* zeigt weitere nützliche Konstanten für die Seitenauswahl. Darüber hinaus können mit „PagesToUse.getPages(1, 2, 3)" individuelle Seiten definiert werden *(siehe Listing 5)*.

Tests auf Seitenausschnitte begrenzen

Es kann notwendig sein, Tests auf Ausschnitte einer Seite zu beschränken. Soll beispielsweise Text nur in der Fußzeile überprüft werden, wird ein Rechteck mit der passenden Größe für diesen Seitenausschnitt definiert. Dieser Ausschnitt ist durch vier Eigenschaften definiert. Es gibt die x/y-Position für die linke obere Ecke des Ausschnitts auf der PDF-Seite sowie die Breite und Höhe des Ausschnitts. Die Standard-Einheit sind „Millimeter", es können aber auch „Zentimeter", „Inches" und „DPI72" angegeben werden *(siehe Listing 4)*.

Zwei PDF-Dokumente vergleichen

Textteile, Seitenzahlen, Bilder, Schriften oder auch das Layout ganzer Seiten oder von Sei-

```
ON_ANY_PAGE,   ON_EACH_PAGE
ON_EVEN_PAGES, ON_ODD_PAGES
ON_FIRST_PAGE, ON_LAST_PAGE
```

Listing 3: Einschränkungen von Tests auf definierten Seiten

tenausschnitten eines Testdokuments können mit entsprechenden Teilen eines Master-Dokuments verglichen werden. *Listing 5* zeigt den Vergleich der jeweils zweiten Seite zweier PDF-Dokumente als gerenderte Images. Sind die beiden Images nicht gleich, wird zusätzlich zur Fehlermeldung noch ein Differenzbild erstellt *(siehe Abbildung 2)*.

```
@Test
public void hasContentOnFirstPage_InClippingArea() {
  String filename = "documentUnderTest.pdf";
  double upperLeftX  = 17.6; // in millimeter
  double upperLeftY  = 45.8;
  double w        = 60.0; // width
  double h        = 8.8; // height
  ClippingArea inClippingArea = new
        ClippingArea(upperLeftX, upperLeftY, w, h);
  AssertThat.document(filename)
        .hasContent(ON_FIRST_PAGE, inClippingArea)
        .startingWith("Lorem")
        .containing("ipsum")
        .endingWith("est laborum.");
}
```

Listing 4: Text innerhalb eines Seitenausschnitts

```
@Test
public void comparePDFWithMasterAsRenderedImages_Page2() {
  String filenameTest = "documentUnderTest.pdf";
  String filenameMaster = "master.pdf";
  PagesToUse ON_PAGE_2 = PagesToUse.getPage(2);
  AssertThat.document(filenameTest)
     .and(filenameMaster)
     .haveSameAppearance(ON_PAGE_2);
}
```

Listing 5: Vergleich von Test-PDF und Master als Images

```
@Test
public void containsLogo_OnAllPagesAfter1() {
  String filename = „documentUnderTest.pdf";
  String imageFileName = "images/myCompanyLogo.png";
  File imageFile = new File(imageFileName);
  AssertThat.document(filename)
     .containsImage(imageFile, OnEveryPage.after(1));
}
```

Listing 6: Logo auf jeder Seite ab Seite 2

Wenn zwei Unternehmen fusionieren, ändert sich häufig das Logo auf den Rechnungen. Das ist kein Programmier-Problem. Dennoch geht manchmal etwas schief, und so sollte einfach getestet werden, ob es wirklich geklappt hat *(siehe Listing 6)*.

Unsichtbare Teile im PDF

Die bisherigen Tests könnten alle manuell ausgeführt werden, auch wenn das zeitaufwändig und fehleranfällig wäre. Unsichtbare Inhalte hingegen, wie Schriften, Lesezeichen, JavaScript, Formulare und XMP-Daten eines PDF-Dokuments lassen sich ausschließlich auf elektronischem Weg testen. So überprüft der Test in *Listing 7*, ob Feldnamen in Formularen eindeutig sind. *Listing 8* zeigt, wie man die Existenz von JavaScript innerhalb eines PDF-Dokuments überprüft.

XMP-Daten validieren

Ein sensibler Inhalt in PDF-Dokumenten sind XMP-Daten. „XMP" steht für „Extensible Metadata Platform" und ist ein Standard, um Metadaten in digitale Medien einzubetten. Häufig wird ein PDF-Dokument mit XMP-Daten angereichert, um diese zu einem späteren Zeitpunkt wieder auszuwerten. Insofern macht es Sinn, wenn die XMP-Daten auch richtig sind. Für die Validierung bietet PDFUnit die Funktion „hasXMPDate()" mit sehr flexiblen Nachfolgefunktionen, in denen XPath mit all seiner Mächtigkeit ausgereizt werden kann. Namespaces werden selbst-

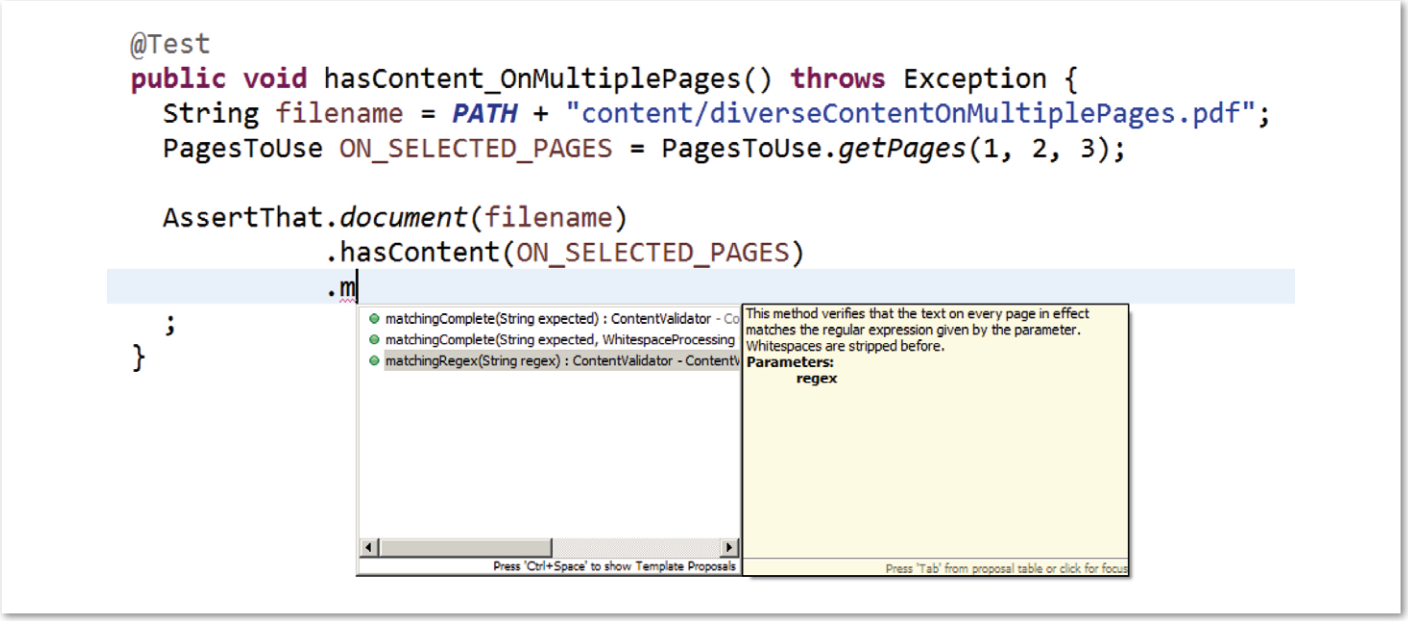


Abbildung 1: Unterstützung im Eclipse

```
@Test
public void hasFields_WithoutDuplicateNames() {
  String filename = "documentUnderTest.pdf";
  AssertThat.document(filename)
      .hasFields()
      .allWithoutDuplicateNames();
}
```

Listing 7: Formular-Felder ohne doppelte Namen

```
@Test
public void hasJavaScript() {
  String filename = "documentUnderTest.pdf";
  String scriptFile = "javascript/fieldValidations.js";
  InputStream scriptFile = new FileInputStream(scriptFile);
  AssertThat.document(filename)
      .hasJavaScript()
      .containing(scriptFile);
}
```

Listing 8: JavaScript in PDF überprüfen

```
@Test
public void hasXMPData_WithNodeAndValue() {
  String filename = "documentUnderTest.pdf";
  XMLNode nodeCreateDate =
  new XMLNode("xmp:CreateDate", "2011-02-08T15:04:19+01:00");
  XMLNode nodeModifyDate =
  new XMLNode("xmp:ModifyDate", "2011-02-08T15:04:19+01:00");

  AssertThat.document(filename)
      .hasXMPData()
      .withNode(nodeCreateDate)
      .withNode(nodeModifyDate);
}

@Test
public void hasXMPData_XPathExpression() {
  String filename = "ocumentUnderTest.pdf";
  DefaultNamespace ns = new
  DefaultNamespace("http://purl.org/dc/elements/1.1/");
  XPathExpression expr =
  new XPathExpression("//foo:format = 'application/pdf'", ns);

  AssertThat.document(filename)
      .hasXMPData()
      .matchingXPath(expr);
}
```

Listing 9: Zwei Tests auf XMP-Daten

verständlich unterstützt. *Listing 9* zeigt zwei Beispiele.

Nützliche kleine Tools

Es ist zwar schön, dass es Tests auf unsichtbare Inhalte gibt, es gibt aber noch ein prinzipielles Problem mit den Inhalten: Wenn Fehler auftauchen, will man erkennen, welche Inhalte ein PDF-Dokument tatsächlich hat. Zur Beantwortung dieser Frage bietet PDFUnit viele kleine Tools, *Listing 10* zeigt eine kleine Auswahl. Die Namen der Klassen sprechen für sich. Es sind Konsolenprogramme, die die notwendigen Daten als Parameter erwarten *(siehe Listing 11)*.

PDFUnit auch als XML-API

Damit auch Nicht-Java-Entwickler ihre PDF-Dokumente testen können, gibt es PDFUnit auch als XML-Schnittstelle. Sämtliche Funktionen der Java-Version stehen dabei über XML zur Verfügung. Die XML-Schnittstelle kommt mit einer XML-Schema-Definition, sodass ein XML-Editor die Erstellung der Tests gut unterstützen kann *(siehe Abbildung 3)*. Die Ausführung der Tests erzeugt denselben JUnit-Report wie normale Unit-Tests. *Listing 12* zeigt das Java-Beispiel aus *Listing 4* in seiner XML-Form.

Die Nachteile

PDFUnit ist nicht kostenlos. Entwickler sind es gewohnt, mit kostenlosen Produkten umzugehen, andererseits haben sie alle für Betriebssystem, Datenbank und Internetzugang bezahlt. In die Entwicklung von PDF-

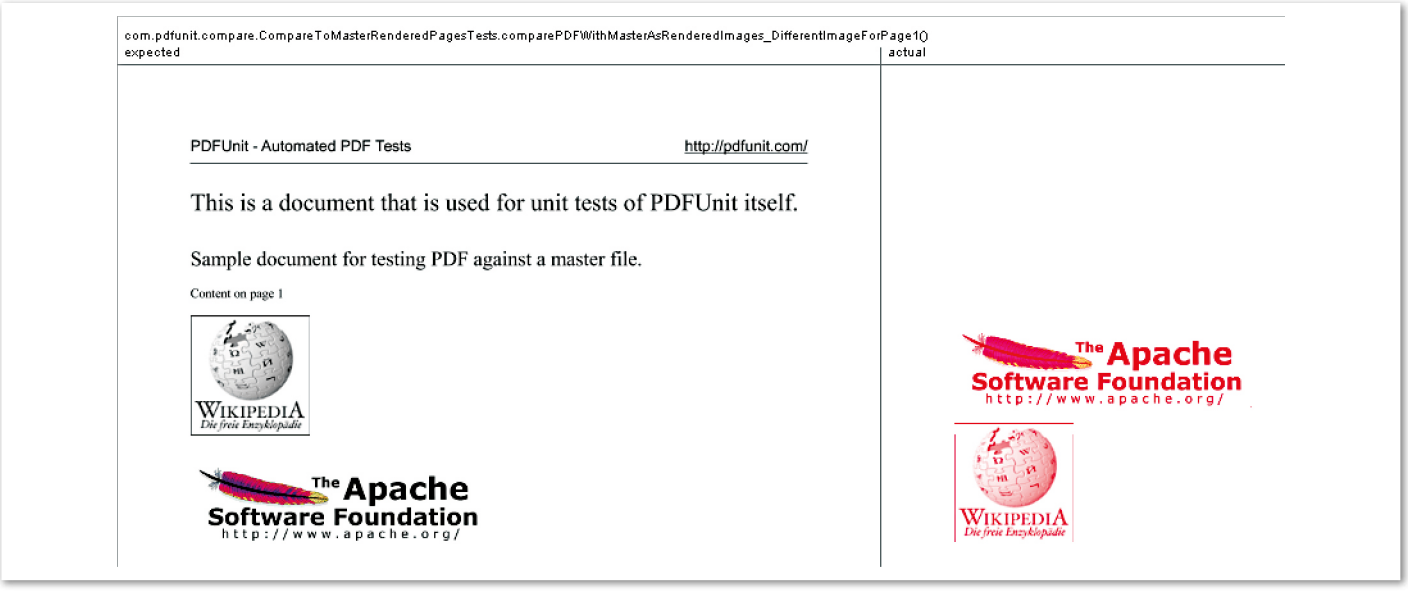


Abbildung 2: Diff-Image beim Seitenvergleich als gerenderte Seiten

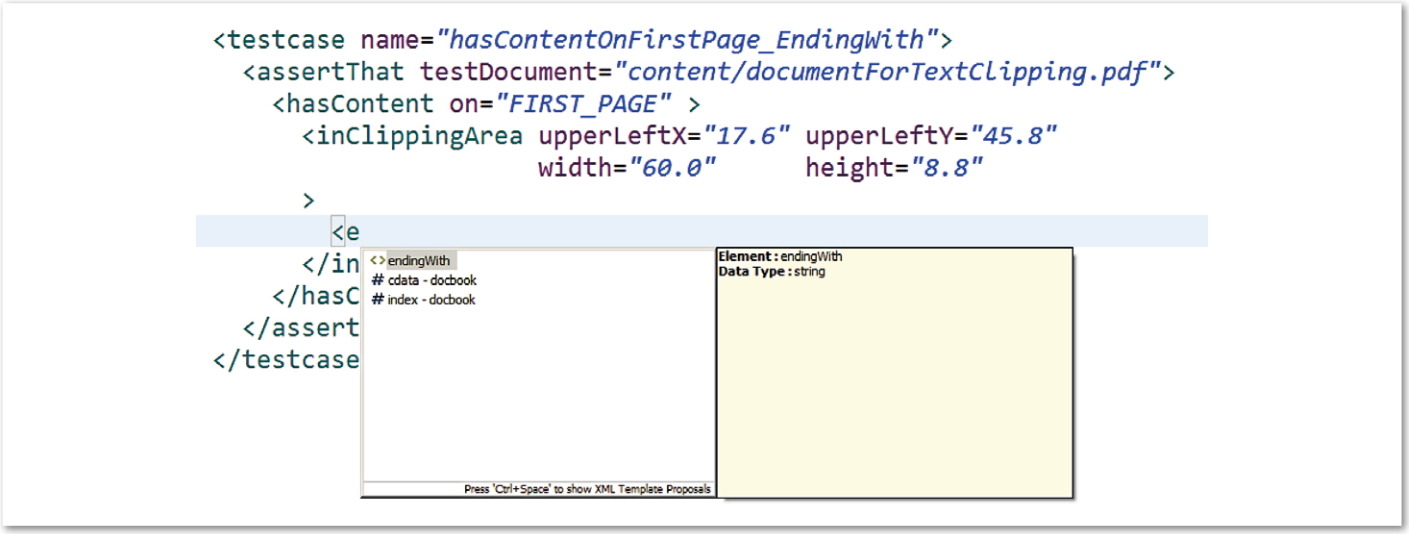


Abbildung 3: XML-Schnittstelle von PDFUnit im Eclipse-Editor

```
ExtractEmbeddedFiles
ExtractFieldsInfo
ExtractImages
ExtractJavaScript
ExtractSignaturesInfo
ExtractXMPData
RenderPdfClippingAreaToImage
```

Listing 10: Nützliche Tools rund um PDF (Auswahl)

```
set TOOL=com.pdfunit.tools.ExtractImages
set OUT_DIR=./tmp
set PASSWD=
set IN_FILE=documentUnderTest.pdf
java %TOOL%  %IN_FILE%  %OUT_DIR%  %PASSWD%
```

Listing 11: Extraktion von Bildern aus einem PDF-Dokument

```
<testcase name="hasContent_OnFirstPage_InClippingArea">
  <assertThat testDocument="documentUnderTest.pdf">
     <hasContent on="FIRST_PAGE" >
        <inClippingArea upperLeftX="17.6"
              upperLeftY="45.8"
              width="60.0"
              height="8.8"
        >
           <startingWith>Lorem</startingWith>
           <containing>ipsum</containing>
           <endingWith>est laborum.</endingWith>
        </inClippingArea>
     </hasContent>
  </assertThat>
</testcase>
```

Listing 12: Beispiel für PDFUnit-XML

Unit sind zwei Personenjahre an Entwicklungszeit geflossen. Solange kann niemand umsonst arbeiten. Immerhin sind während der Entwicklung nützliche Informationen an verschiedene Open-Source-Projekte zurückgeflossen.

Zukünftige Entwicklung

Neue Funktionen entstehen überwiegend durch Kundenanfragen. Parallel dazu wird das Projekt intern auf Java-8 (Streams) umgestellt, ohne dass dadurch die Schnittstelle verändert wird. Solche Änderungen können aufgrund der hohen Testabdeckung von PDFUnit problemlos durchgeführt werden. Das nächste Release erscheint im April 2015. Dann wird auch die Perl-API verfügbar sein, die ebenfalls zurzeit entwickelt wird. Ausführliche Dokumentationen und ein Beispielprojekt für Eclipse stehen unter „www.pdfunit.com“ zur Verfügung.

Fazit

Programme mögen kompliziert sein, aber die von Ihnen erzeugten PDF-Dokumente gehen an Kunden. Es ist einfach geworden, Dokumente zu testen – sie sollten fehlerfrei sein.

Carsten Siedentop
c.siedentop@pdfunit.com

Carsten Siedentop ist seit 23 Jahren freiberuflich als Anwendungsentwickler in Software-Projekten (Java, COBOL) tätig. Zusätzlich gibt er sein Wissen seit 18 Jahren als Trainer weiter. Im Leben jenseits des Computers nimmt die Musik einen großen Raum ein, unter anderem ist er Sänger an der Kölner Oper.

http://ja.ijug.eu/15/2/15

Unbekannte Kostbarkeiten des SDK

Heute: Bestimmung des Aufrufers

Bernd Müller, Ostfalia

Das Java SDK enthält eine Reihe von Features, die wenig bekannt sind. Wären sie bekannt und würden sie verwendet, könnten Entwickler viel Arbeit und manchmal sogar zusätzliche Frameworks einsparen. Wir stellen in dieser Reihe derartige Features des SDK vor: die unbekannten Kostbarkeiten.

Manchmal wäre die Information, wer eine Methode aufgerufen hat, sehr interessant, etwa bei der Erzeugung von Logging-Meldungen. Java besitzt keine einfache und intuitiv zugängliche Möglichkeit, dies zu realisieren. Man kann jedoch den Stack-Trace des aktuellen Threads verwenden, um an die gewünschte Information zu gelangen.

Motivation

Generell ist es für das Gesamtverständnis eines Systems wichtig zu wissen, welche Methode „A" welche Methode „B" aufruft, beziehungsweise alternativ, welche Methode „B" durch welche Methode „A" aufgerufen wird. Statische Analyse-Werkzeuge und damit insbesondere moderne IDEs unterstützen bei der Beantwortung beider Fragen. Die jeweilige Antwort basiert aber ausschließlich auf einer statischen Analyse und liefert für die zweite Fragestellung immer nur die potentiellen Kandidaten eines Aufrufs. Will man zur Laufzeit wissen, wer der Aufrufer einer bestimmten Methode ist, muss man etwas tiefer in die Trickkiste greifen.

Stack-Traces

Die Klassen „Error" und „Exception", beides Unterklassen von „Throwable", werden verwendet, um Ausnahmesituationen bei der Programmausführung anzuzeigen. Dazu wird eine Momentaufnahme des Aufruf-Stacks des ausführenden Threads in ein Array von Instanzen der Klasse „StackTraceElement" kopiert und in „Throwable" geschrieben. Die Momentaufnahme erfolgt zu dem Zeitpunkt, an dem die Ausnahmesituation aufgetreten ist. Über die Methode „getStackTrace()" des „Throwable" kann dann auf diesen Stack-Trace zugegriffen werden.

Um an einen Stack-Trace und damit eine Repräsentation des aktuellen Aufruf-Stacks zu kommen, muss aber nicht notwendigerweise ein „Error" oder eine „Exception" erzeugt werden. Die Klasse „Thread" besitzt ebenfalls eine Methode „getStackTrace()", die das Gewünschte liefert. Dazu das entsprechende Zitat aus dem Java-Doc der Methode „getStackTrace()" *[1]*: „Returns an array of stack trace elements representing the stack dump of this thread. This method will return a zero-length array if this thread has not started, has started but has not yet been scheduled to run by the system, or has terminated. If the returned array is of non-zero length then the first element of the array represents the top of the stack, which is the most recent method invocation in the sequence. The last element of the array represents the bottom of the stack, which is the least recent method invocation in the sequence. Some virtual machines may, under some circumstances, omit one or more stack frames from the stack trace. In the extreme case, a virtual machine that has no stack trace information concerning this thread is permitted to return a zero-length array from this method."

Das Beispiel in *Listing 1* macht sich dies zunutze. In der Main-Methode der Klasse „Caller" wird die Methode „foo()" der Klasse „Callee" aufgerufen. In dieser erfolgt der Zugriff auf den aktuellen Stack-Trace mit der Methode „getStackTrace()". Der Rückgabewert ist ein Array von „StackTraceElement"-Objekten. Wie dem Java-Doc zu entnehmen, repräsentiert das erste Array-Element den obersten Eintrag des Stacks und das letzte Array-Element den ersten Methodenaufruf in dieser Aufruf-Folge. *Listing 2* zeigt die Ausgabe des Beispiels. Die Zeilennummern entsprechen nicht dem Code

```
public class Caller {

  public static void main(String[] args) {
     new Callee().foo();
  }


}

public class Callee {

  public void foo() {

     StackTraceElement[] stackTraceElements =
        Thread.currentThread().getStackTrace();
     System.out.println("0: " + stackTraceElements[0]);
     System.out.println("1: " + stackTraceElements[1]);
     System.out.println("2: " + stackTraceElements[2]);
  }

}
```

Listing 1

```
0: java.lang.Thread.getStackTrace(Thread.java:1552)
1: de.pdbm.Callee.foo(Callee.java:6)
2: de.pdbm.Caller.main(Caller.java:6)
```

Listing 2

aus *Listing 1*, da dort die Import-Anweisungen weggelassen wurden. Im Beispiel enthält das Array auch nur die dargestellten drei Elemente. In einem etwas realistischeren Programm ist das Array deutlich größer. Der interessante Wert ist im dritten Array-Element (Index 2) gespeichert. Will man beispielsweise für das Logging entsprechende Informationen aufbereiten, so können die folgenden Methoden der Klasse „StackTraceElement" verwendet werden:

- String getClassName()
- String getFileName()
- String getMethodName()
- int getLineNumber()

Stack-Traces revisited

Das eigentliche Ziel ist erreicht, nämlich die Bestimmung des Aufrufers einer bestimmten Methode. Es soll aber nicht unerwähnt bleiben, dass das dargestellte Verfahren einen gewissen Aufwand bedeutet und somit bei großzügiger Verwendung zu Performance-Problemen führen kann. Generell ist die Erzeugung eines (großen) Stack-Traces mit deutlichem Aufwand verbunden *[2]*, da der ausführende Thread unterbrochen, das Array erzeugt und unter Verwendung vieler String-Konvertierungsmethoden befüllt werden muss.

Wie fast immer bei teuren Operationen, bietet die VM auch hier Kommandozeilen-Optionen für das Fein-Tuning an. So kann beispielsweise die maximale Größe des Strack-Traces, also die Länge des Arrays, mit „-XX:MaxJavaStackTraceDepth=<value>" auf den Wert „value" beschränkt werden. Der Default ist „1024". „-XX:-StackTraceInThrowable" unterbindet die Erzeugung des Stack-Trace ganz, es wird also ein Array mit null Elementen erzeugt. Dasselbe gilt ebenfalls für die erste Option mit einem Wert von „0". In beiden Fällen liefert das Beispielprogramm eine „ArrayIndexOutOfBoundsException".

Fazit

Java kennt keinen direkten Weg, um in einer Methode den Aufrufer dieser Methode zur Laufzeit zu bestimmen. Mit einem explizit erzeugten Stack-Trace innerhalb der Methode ist dies jedoch leicht möglich. Da der damit verbundene Aufwand relativ hoch ist, empfiehlt sich dieses Vorgehen jedoch nur in Ausnahmefällen.

Literatur/Videos

[1] http://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html

[2] Scott Oaks, Java Performance – The Definitive Guide, O'Reilly, 2014

Bernd Müller

bernd.mueller@ostfalia.de

Bernd Müller ist Professor für Software-Technik an der Ostfalia (Hochschule Braunschweig/Wolfenbüttel). Er ist Autor mehrere Java-EE-Bücher, Sprecher auf nationalen und internationalen Konferenzen und engagiert sich in der JUG Ostfalen sowie im iJUG.

http://ja.ijug.eu/15/2/16

DEVOXX 4KIDS in Deutschland

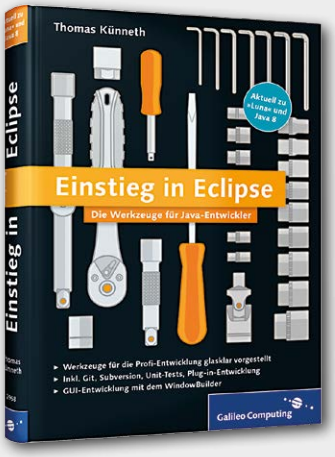
Am 6. Juni 2015 wird es in Hamburg eine Devoxx4Kids geben. Dahinter stehen weltweit organisierte Veranstaltungen, bei denen Kinder Computerspiele entwickeln, Roboter programmieren und eine Einführung in Elektronik bekommen. Ziel ist es zu zeigen, dass es möglich ist, kreativ mit dem Computer zu arbeiten. Im Vorfeld der Veranstaltung sind Quadrocopter, Tinkerforge und NAOs geplant.

Bereits am 5. Mai 2015 findet in Hamburg bei der JUG HH ein Vorstellungsevent mit dem Karlsruher Devoxx4Kids-Team statt.

Weitere Informationen unter „http://www.devoxx4kids.org/deutschland"

Einstieg in Eclipse

Gelesen von Daniel Grycman



„Effiziente Java-Entwicklung mit Eclipse" verspricht der Autor Thomas Künneth auf der Buch-Rückseite. Das Buch richtet sich an Eclipse-Einsteiger und setzt Java-Kenntnisse voraus, die auch notwendig sind. Der Fokus liegt dabei auf dem Handling von Eclipse und nicht wirklich auf dem Ansatz einer effizienten Entwicklung von Anwendungen mit Eclipse. Positiv bleibt aber anzumerken, dass Thomas Künneth viel Erfahrung mit Eclipse besitzt.

Auf dem Titel rühmt sich das Buch „aktuell zu Luna und Java 8" zu sein, was dem Autor aber nur in Bezug auf die Aktualisierung der Abbildung und im Unterkapitel „Ein neuer Look" einigermaßen gelingt. Leider lässt er jeglichen Hinweis über die Möglichkeiten des Java-Eclipse-Compilers im Hinblick auf Lambdas und sonstige Erweiterungen vollkommen aus. Gerade hier hätte der Autor anhand entsprechender Code-Beispiele eine effiziente Entwicklung mit Java aufzeigen können. Ebenso ist nicht nachvollziehbar, dass gerade mit dem Zusatz „Java 8" die Erstellung einer grafischen Benutzeroberfläche dem Leser anhand von Swing nahegebracht wird. Ein Einstieg in JavaFX wäre an dieser Stelle logischer gewesen. Beim zweiten Teil „Fortgeschrittene Techniken" wurde leider auch auf eine kurze Einführung von Maven im Zusammenhang mit Eclipse komplett verzichtet, was sich gerade im Hinblick auf die Verwendung von Ant im letzten Kapitel des ersten Teils angeboten hätte.

Beim Lesen bekommt man leider fortlaufend den Eindruck, dass es sich bei dieser fünften, aktualisierten Auflage um eine sehr schnelle und in Teilen sehr unsaubere Überarbeitung handelt. Im Buch finden sich vermeidbare Tippfehler, beispielsweise „ALAX" statt „AJAX" im Inhaltsverzeichnis und der entsprechenden Kapitelüberschrift.

Fazit: Das Buch ist lediglich für wirkliche Einsteiger in die Entwicklungsumgebung Eclipse empfehlenswert, aber auch nur, um einen Überblick über die Bedienung der IDE zu erhalten.

Titel:	Einstieg in Eclipse – Die Werkzeuge für Java-Entwickler (Aktuell zu Luna und Java 8)
Autor:	Thomas Künneth
Auflage:	5., aktualisierte Auflage 2014
Verlag:	Galileo Computing
Umfang:	400 Seiten
Preis:	39,90 Euro eBook (downloadbar) im Preis enthalten
ISBN:	978-3-8362-2958-6

Daniel Grycman
daniel.grycman@bilsteingroup.com

Java – Der Grundkurs

Gelesen von Oliver B. Fischer

Im Galileo-Computing-Verlag legt Michael Kofler mit „Java – Der Grundkurs" sein erstes Java-Buch vor, dem im letzten Jahr bereits ein im Eigenverlag erschienenes eBook vorausging. Wenn sich ein Buch explizit an Anfänger einer Programmiersprache wendet, muss es in der Lage sein, einen zügigen und motivierenden Einstieg zu ermöglichen, zum anderen aber auch einen möglichst breiten und hinreichend tiefen Einblick zu vermitteln. Koflers Java-Grundkurs verfügt über einen sehr strukturierten Aufbau. Beginnend mit dem verpflichtenden „HelloWorld"-Programm, wird der Leser im ersten Kapitel durch die Installation des JDK und die von Eclipse auf den unterschiedlichen Plattformen begleitet und auch die händische Nutzung von „javac" erklärt.

In den nächsten Kapiteln wird der Leser Schritt für Schritt in Variablen, deren Sichtbarkeit, Datentypen und Operatoren sowie die unterschiedlichen Kontrollstrukturen eingeführt. Der Arbeit mit Arrays und Zeichenketten, Datum und Zeitangaben sind ebenfalls einzelne Kapitel gewidmet. Anschließend werden Methoden und Exceptions vorgestellt. Erst zur Mitte des Buchs erfolgt der Einstieg in die objektorientierte Java-Programmierung, die durch Kapitel zu Generics, Lambda-Ausdrücken und Collections ergänzt wird. Zudem gibt es einzelne Kapitel zur Arbeit mit Dateien, JavaDoc und JavaFX.

Fazit: Die Frage „Kaufen oder nicht kaufen?" lässt sich nur abhängig von dem eigenen Ziel beantworten. Für einen Crashkurs für die Universität oder den Schnelleinstieg im Eigenstudium enthält es alles Notwendige. Für eine dauerhafte Arbeit mit Java ist es nicht ausreichend. Zur erfolgreichen Arbeit mit Java gehört auch Wissen um dessen Interna.

Titel:	Java – Der Grundkurs
Autor:	Michael Kofler
Verlag:	Galileo Computing
Umfang:	426 Seiten
Preis:	12,90 Euro eBook (downloadbar)
ISBN:	978-3-8362-2923-4

Oliver B. Fischer
o.b.fischer@swe-blog.net

Android-Apps entwickeln

Gelesen von Ulrich Cech

"Android-Apps entwickeln" von Uwe Post erhebt den Anspruch, selbst ohne Programmier-Kenntnisse den Einstieg in die Android-Programmierung und dabei eine Grundeinführung in die Java- und Objektorientierte Programmierung zu vermitteln. Das Buch stellt dies anhand der Entwicklung einer Android-Spiele-App dar.

Der Autor schafft es von Anfang an, Interesse an der Android-Entwicklung und die Motivation auf die Entwicklung eigener Ideen zu wecken, da er schon im ersten Kapitel einen Ausblick gibt, welche Funktionalitäten im Laufe des Buches implementiert werden. Weiter geht es mit einer sehr kurzen Einführung in Java und in die objektorientierte Programmierung. Dass hier tatsächlich nur an der Oberfläche gekratzt werden kann, sollte in einem solchen Werk jedem einleuchten. Der Autor selbst verweist an dieser Stelle auf weiterführende Sekundär-Literatur, die kostenlos im Internet verfügbar beziehungsweise auf der beiliegenden DVD enthalten ist. Er versucht, die Einstiegshürden sehr niedrig zu halten, damit der Leser den Programmcode auch in folgenden Kapiteln zumindest grob nachvollziehen kann. Trotzdem kommt jemand, der noch nie programmiert hat, um eine ausführliche Grundeinführung in die Objekt-Orientierung nicht herum, denn die Einführung im Buch alleine ist nicht ausreichend.

Titel:	Android-Apps entwickeln, Eine Spiele-App von A bis Z, 4. aktualisierte Auflage
Autor:	Uwe Post
Verlag:	Galileo Computing
Umfang:	409 Seiten
Preis:	24,90 Euro
ISBN:	978-3-8362-2790-2

Im Kapitel über die Eclipse-IDE für die Android-Entwicklung geht der Autor auf zentrale Elemente beim Programmieren ein, nämlich Logging und Debugging. Sehr hilfreich ist in diesem Kapitel vor allem das Aufzeigen einiger Eclipse-Probleme in Kombination mit dem Eclipse-ADT-Plug-in, was gerade einem Einsteiger einiges Kopfzerbrechen ersparen wird. Hier sei angemerkt, dass zum aktuellen Zeitpunkt das Eclipse-ADT in den Wartungsmodus versetzt ist und keine neuen Features mehr entwickelt werden. Google setzt dabei auf die IntelliJ-basierte-IDE „Android-Studio". Diese Information stand jedoch zum Erscheinungstermin der 4. Auflage des Buchs noch nicht zur Verfügung.

Die erste Android-App enthält bereits Sprachausgabe. Hier zeigt der Autor, dass komplex aussehende Funktionalitäten sehr einfach implementiert werden können. Spätestens jetzt sollte die Spannung auf die Entwicklung einer eigenen Spiele-App aufgebaut sein. Diese Spannung wird auch bedient, denn innerhalb eines Kapitels mit nicht einmal 200 Zeilen Code schafft es der Autor, eine lauffähige Version eines kleinen Spiels zu programmieren und das nötige Verständnis der beteiligten Basis-Komponenten zu vermitteln. Trotzdem hält das Buch die Programmierlust aufrecht, denn das Spiel wird in den nachfolgenden Kapiteln mit zusätzlichen Elementen (wie Sound, Animation) erweitert.

Die Tatsache, dass es dem Autor gelingt, in einem Einstiegsbuch sogar „Augmented Reality" zu integrieren, ist schon bemerkenswert. Hier zeigt sich gerade wieder seine Fähigkeit, die Dinge auf das absolut Notwendige zu beschränken und dem Leser einfach beizubringen, wie diese Funktionalität zu implementieren sei.

Auf 48 Seiten werden einige Sensoren besprochen und repräsentative Beispiel-Apps entwickelt: ein Schrittzähler für den Erschütterungssensor und eine App zum Aufzeichnen der Wegpunkte eines zurückgelegten Wegs mit anschließender Darstellung dieses Wegs auf einer Karte. Dem Einsteiger wird eine Menge an Features in diesem Buch gezeigt, das Gefühl von Erfolgserlebnissen kann der Autor perfekt vermitteln.

Das Kapitel „Tipps und Tricks" ist namentlich etwas verwirrend, denn hier werden Views, Styles und Themes sowie Dialog-Elemente und die Programmierung von Homescreen-Widgets vorgestellt, was eher unter „GUI-Optimierung" zusammengefasst werden könnte.

Im letzten Kapitel über das Thema „Veröffentlichung einer Android-App" geht der Autor auch auf alternative Android-Markets ein und zeigt die dortigen Chancen. Hier ist anzumerken, dass einer der erwähnten Markets („AndroidPIT-Market") zum Ende des Jahres 2014 schließt, aber auch diese Info existierte bei Erscheinungsdatum des Buches noch nicht. Der Autor stellt die InApp-Kaufoptionen ganz grob anhand eines Google-Beispielcodes aus der offiziellen Dokumentation vor, die für ein grundlegendes Verständnis reicht, aber zur effektiven Implementierung noch zahlreiche Fragen offen lässt.

Fazit: Der Autor hält einen roten Faden durch, um von einer Thematik zur nächsten zu kommen. Er macht keine unlogischen Sprünge, Exkurse oder sonstige den Lesefluss-hemmenden Aktionen. Dass der Verfasser auch Science-Fiction-Autor ist, erklärt den flüssig zu lesenden Text und die hervorragende didaktische Aufbereitung der Thematik. Es macht einfach nur viel Spaß, dieses Buch zu lesen, wodurch auch die Motivation für die Android-Entwicklung aufgebaut wird.

Dieses Buch versteht es, in sehr kompakter Art und Weise die zahlreichen Aspekte und Möglichkeiten der Android-Entwicklung vorzustellen, was es zu einem idealen Einstiegswerk macht. Der Autor gibt an zahlreichen Stellen zusätzliche Hinweise für die Entwicklung von gut strukturiertem und flexiblem Code.

Alles, was für die App-Entwicklung erforderlich ist (Eclipse, JDK etc.), ist auf der beiliegenden DVD. Dass der gesamte Quellcode beziehungsweise die Eclipse-Projekte für jedes Kapitel auf der DVD enthalten sind, sodass der Leser seinen Code am Ende auch noch einmal bei Problemen mit der Referenz abgleichen kann, runden dieses Werk vollständig ab.

Ulrich Cech
uc@cech-home.de

Die iJUG-Mitglieder auf einen Blick

Java User Group Deutschland e.V.
www.java.de

DOAG Deutsche ORACLE-Anwendergruppe e.V.
www.doag.org

Java User Group Stuttgart e.V. (JUGS)
www.jugs.de

Java User Group Köln
www.jugcologne.eu

Java User Group Darmstadt
http://jugda.wordpress.com

Java User Group München (JUGM)
www.jugm.de

Java User Group Metropolregion Nürnberg
www.source-knights.com

Java User Group Ostfalen
www.jug-ostfalen.de

Java User Group Saxony
www.jugsaxony.org

Sun User Group Deutschland e.V.
www.sugd.de

Swiss Oracle User Group (SOUG)
www.soug.ch

Berlin Expert Days e.V.
www.bed-con.org

Java Student User Group Wien
www.jsug.at

Java User Group Karlsruhe
http://jug-karlsruhe.mixxt.de

Java User Group Hannover
www.jug-h.de

Java User Group Augsburg
www.jug-augsburg.de

Java User Group Bremen
www.jugbremen.de

Java User Group Münster
www.jug-muenster.de

Java User Group Hessen
www.jugh.de

Java User Group Dortmund
www.jugdo.de

Java User Group Hamburg
www.jughh.de

Java User Group Berlin-Brandenburg
www.jug-berlin-brandenburg.de

Java User Group Kaiserslautern
www.jug-kl.de

Der iJUG möchte alle Java-Usergroups unter einem Dach vereinen. So können sich alle Java-Usergroups in Deutschland, Österreich und der Schweiz, die sich für den Verbund interessieren und ihm beitreten möchten, gerne unter office@ijug.eu melden.

iJUG Verbund

www.ijug.eu

Impressum

Herausgeber:
Interessenverbund der Java User Groups e.V. (iJUG)
Tempelhofer Weg 64, 12347 Berlin
Tel.: 030 6090 218-15
www.ijug.eu

Verlag:
DOAG Dienstleistungen GmbH
Fried Saacke, Geschäftsführer
info@doag-dienstleistungen.de

Chefredakteur (VisdP):
Wolfgang Taschner, redaktion@ijug.eu

Redaktionsbeirat:
Ronny Kröhne, IBM-Architekt;
Daniel van Ross, FIZ Karlsruhe;
Dr. Jens Trapp, Google;
André Sept, InterFace AG

Titel, Gestaltung und Satz:
Alexander Kermas,
DOAG Dienstleistungen GmbH

Anzeigen:
Simone Fischer
anzeigen@doag.org

Mediadaten und Preise:
http://www.doag.org/go/mediadaten

Druck:
adame Advertising and Media GmbH
www.adame.de

Titelfoto: © gubh83 / Fotolia.com
Foto S. 4: © Alexander Atkishkin / 123rf.com
Foto S. 26: © ruthblack / 123rf.com
Foto S. 63: © alexmit / 123rf.com

Inserentenverzeichnis

www.ijug.eu

JETZT ABO BESTELLEN

Sichern Sie sich 4 Ausgaben für 18 EUR

Für Oracle-Anwender und Interessierte gibt es das Java aktuell Abonnement auch mit zusätzlich sechs Ausgaben im Jahr der Fachzeitschrift DOAG *News* und vier Ausgaben im Jahr Business News zusammen für 70 EUR. Weitere Informationen unter www.doag.org/shop/

FAXEN SIE DAS AUSGEFÜLLTE FORMULAR AN
0700 11 36 24 39

ODER BESTELLEN SIE ONLINE
go.ijug.eu/go/abo

Interessenverbund der Java User Groups e.V.
Tempelhofer Weg 64
12347 Berlin

Java aktuell

+++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN

☐ **Ja**, ich bestelle das Abo Java aktuell – das IJUG-Magazin: 4 Ausgaben zu 18 EUR/Jahr

☐ **Ja**, ich bestelle den kostenfreien Newsletter: Java aktuell – der iJUG-Newsletter

ANSCHRIFT

Name, Vorname

Firma

Abteilung

Straße, Hausnummer

PLZ, Ort

GGF. ABWEICHENDE RECHNUNGSANSCHRIFT

Straße, Hausnummer

PLZ, Ort

E-Mail

Telefonnummer

X

Die allgemeinen Geschäftsbedingungen* erkenne ich an, Datum, Unterschrift

*Allgemeine Geschäftsbedingungen:
Zum Preis von 18 Euro (inkl. MwSt.) pro Kalenderjahr erhalten Sie vier Ausgaben der Zeitschrift "Java aktuell - das iJUG-Magazin" direkt nach Erscheinen per Post zugeschickt. Die Abonnementgebühr wird jeweils im Januar für ein Jahr fällig. Sie erhalten eine entsprechende Rechung. Abonnementverträge, die während eines Jahres beginnen, werden mit 4,90 Euro (inkl. MwSt.) je volles Quartal berechnet. Das Abonnement verlängert sich automatisch um ein weiteres Jahr, wenn es nicht bis zum 31. Oktober eines Jahres schriftlich gekündigt wird. Die Wiederrufsfrist beträgt 14 Tage ab Vertragserklärung in Textform ohne Angabe von Gründen.

upcycling software

DOAG
www.doag.org

DevCamp
29. - 30. April 2015 | Frankfurt a. M.