

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Java hebt ab

Java aktuell

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



Praxis

Prinzipien des API-Managements, Seite 27

Mobile

Android-App samt JEE-Back-End
in der Cloud bereitstellen, Seite 33

Grails

Enterprise-2.0-Portale, Seite 39

CloudBees und Travis CI

Cloud-hosted Continuous Integration, Seite 58



iJUG
Verbund



25.-26. März 2014
im Phantasialand
Brühl bei Köln

Zwei Tage lang das JavaLand besiedeln



Die Konferenz der Java-Community!

- Seien Sie mit dabei, wenn die neue Konferenz zum Zentrum der deutschen Java-Szene wird!
- Wissenstransfer, Networking und gute Gespräche treffen auf spannende Abenteuer, Spaß und Action.
- Vom Einsteiger bis zum Experten haben alle die Gelegenheit, zwei Tage im JVM-Kosmos zu leben.



Fried Saacke
Vorsitzender ijUG e.V.

"Die im ijUG verbundenen Mitglieder von derzeit 18 Java User Groups aus Deutschland, Österreich und der Schweiz engagieren sich gemeinsam für die Gestaltung der Konferenz. Damit wird JavaLand die führende Java-Konferenz für die Java Community."

www.javaLand.eu

Präsentiert von:



 Heise Zeitschriften Verlag

Community Partner:





Wolfgang Taschner
Chefredakteur Java aktuell



Zwei Tage lang das JavaLand besiedeln

Eine neue Konferenz für die Java-Community: Das „JavaLand 2014“ findet vom 25. bis zum 26. März 2014 statt. Das Außergewöhnliche an dem Community-Event ist neben Wissenstransfer, Networking und guten Gesprächen die Location. Den Teilnehmern steht an diesen beiden Tagen das Phantasialand in Brühl bei Köln exklusiv zur Verfügung – Spaß und Action inbegriffen.

JavaLand 2014 bietet damit einen neuartigen Rahmen zum Lernen, Erfahren und Austauschen. Sowohl Java-Einsteiger als auch Java-Experten haben die Gelegenheit, zwei Tage gemeinsam ihren eigenen Kosmos zu erleben. Ein besonderes Erlebnis ist die Übernachtung im Hotel Matamba oder Ling Bao – und damit das Wohnen im JavaLand. Einmal eingereist, braucht man die Konferenz nicht mehr zu verlassen. Das Zimmerangebot ist jedoch stark begrenzt. Nur wer jetzt bucht, hat eine Chance auf eines der begehrten Zimmer.

Das JavaLand ist jedoch weit mehr, denn die Konferenz versteht sich als „Event der Java-Community für die Java-Community“. Dafür wurden die im IJUG e.V. vertretenen Java-Usergroups zur Mitarbeit eingeladen, um die Konferenz mitzugestalten. Mehr als 400 Vorträge hat die Java-Community eingereicht. Neben vielen Referenten aus dem deutschsprachigen Raum haben sich auch internationale Speaker beworben. Ein Team aus neun Java-Experten stellt jetzt ein attraktives Programm zusammen, das im November veröffentlicht wird.

Alle aktuellen Informationen sind auf www.javaland.eu zu finden. Dort können sich auch die Teilnehmer registrieren. Bis Ende Januar 2014 gibt es noch den sehr günstigen Frühbucher-Rabatt.

Auch die Java aktuell ist im JavaLand 2014 präsent. Ich lade alle Leserinnen und Leser herzlich in unser „offenes Redaktionsbüro“ ein, um über Inhalte und Ausrichtung der Zeitschrift zu reden. Wer möchte, kann sich auch mit einem Beitrag an der geplanten JavaLand-Zeitung beteiligen, die am ersten Tag vor Ort entstehen und am zweiten Tag bereits an alle Teilnehmer verteilt werden soll.

Ich werde Sie in der nächsten Ausgabe über das Programm und die weiteren Aktionen im JavaLand auf dem Laufenden halten und freue mich, Sie im Phantasialand persönlich zu treffen.

Ihr

Trainings für Java / Java EE

- Java Grundlagen- und Expertenkurse
- Java EE: Web-Entwicklung & EJB
- JSF, JPA, Spring, Struts
- Eclipse, Open Source
- IBM WebSphere, Portal und RAD
- Host-Grundlagen für Java Entwickler

Wissen wie's geht

Unsere Schulungen können gerne auf Ihre individuellen Anforderungen angepasst und erweitert werden.

Weitere Themen und Informationen zu unserem Schulungs- und Beratungsangebot finden Sie unter www.aformatik.de

aformatik.[®]

aformatik Training & Consulting GmbH & Co. KG
Tilsiter Str. 6 | 71065 Sindelfingen | 07031 238070

www.aformatik.de



Java macht wieder richtig Spaß:
Neuigkeiten von der JavaOne, Seite 8

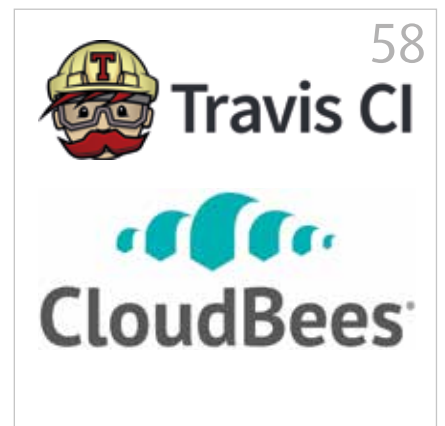


Interview mit Mark Little über das Wachstum
und die Komplexität von Java EE 7, Seite 30

3	Editorial	27	Prinzipien des API-Managements <i>Jochen Traunecker und Tobias Unger</i>	46	Contexts und Dependency Injection – der lange Weg zum Standard <i>Dirk Mahler</i>
5	Das Java-Tagebuch <i>Andreas Badelt, Leiter der DOAG SIG Java</i>	30	„Das Wachstum und die Komplexität von Java EE 7 sind nichts Ungewöhn- liches ...“ <i>Interview mit Mark Little</i>	50	Das neue Release ADF Mobile 1.1 <i>Jürgen Menge</i>
8	Java macht wieder richtig Spaß <i>Wolfgang Taschner</i>	33	Eine Android-App samt JEE-Back-End generieren und in der Cloud bereit- stellen <i>Marcus Munzert</i>	52	Einfach skalieren <i>Leon Rosenberg</i>
9	Oracle WebLogic Server 12c – Zuver- lässigkeit, Fehlertoleranz, Skalierbar- keit und Performance <i>Sylvie Lübeck</i>	39	Enterprise-2.0-Portale mit Grails – geht das? <i>Manuel Breiffeld und Tobias Kraft</i>	58	Cloud-hosted Continuous Integration mit CloudBees und Travis CI <i>Sebastian Herbermann und Sebastian Laag</i>
14	Solr und Elasticsearch – Lucene on Steroids <i>Florian Hopf</i>	44	Wo und warum der Einsatz von JavaFX sinnvoll ist <i>Björn Müller</i>	62	Überraschungen und Grundlagen bei der nebenläufigen Programmierung in Java <i>Christian Kumpke</i>
20	Portabilität von Java-Implementie- rungen in der Praxis <i>Thomas Niedergesäß und Burkhard Seck</i>			13	Inserenten
				66	Impressum



JavaFX oder eine HTML5-basierte Technologie:
Wo und warum der Einsatz von JavaFX sinnvoll ist, Seite 44



Cloud-hosted Continuous Integration mit
CloudBees und Travis CI, Seite 58

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 dritten Quartal 2013.

15. Juli 2013

Java-EE-7-Tutorials online

Die Java-Tutorials für EE 7 sind komplett online. Das Tutorial-Team hat mit „Your First Cup“ auch eine neue Sample-Applikation entwickelt.

<http://docs.oracle.com/javasee>

31. Juli 2013

Was Sie schon immer über TCK-Lizenzen wissen wollten ...

Im letzten Java-Tagebuch wurde die kostenlose Bereitstellung von Test Compatibility Kits an die Eclipse-Projekte „EclipseLink“ und „Virgo“ erwähnt. Markus Eisele, unter anderem Mitglied der Java EE 7 Expert Group, hat sich jetzt einmal die Mühe gemacht, die Lizenzsituation im JCP insgesamt zu analysieren und ebenfalls, was genau die TCKs für die genannten Projekte bedeuten. Unter anderem berichtet er, dass von den achtundzwanzig JSRs für Java EE nur drei ein frei verfügbares TCK haben. Alle anderen seien bei Oracle unter Verschluss und müssen von Firmen, die ihre Application-Server als kompatibel zertifizieren wollen, zu einem vermutlich nicht unerheblichen Preis erworben werden. Dabei haben „Non Profit“-Projekte über einen speziellen Antrag eine Chance auf eine kostenfreie Lizenz. Die Lizenzvergabe an EclipseLink, das sowieso fast ausschließlich von Oracle-Mitarbeitern entwickelt wird, sei im Übrigen mehr ein lizenzrechtlicher Trick gewesen, um der existierenden Referenz-Implementierung das TCK auch offiziell zur Verfügung stellen zu können. Zwar werden die Lizenzen für TCKs mit der Überarbeitung der JCP-Regeln strenger geregelt, um mehr Transparenz und Fairness gegenüber potenziellen Lizenzneh-

mern zu erreichen. Aber das gilt eben nur für neue JSRs, sodass die aktuelle Situation noch eine Weile bestehen bleiben wird.

<http://blog.eisele.net/2013/07/twotcks-for-eclipse-what-is-really-in.html>

15. August 2013

Lambda-Expressions testen

Lambda-Expressions sind „das“ Feature in Java SE 8. Auf der Suche nach einer Einführung, um die Möglichkeiten und die Mächtigkeit auszutesten, gibt es einen guten Überblick mit vielen Code-Beispielen, nicht nur für Eclipse-Nutzer, im Oracle Technet.

<http://www.oracle.com/technetwork/articles/java/lambda-1984522.html>

15. August 2013

Sicherheitslücke: Bitcoins gestohlen

Das Thema „Sicherheit“ verfolgt alle, auch Java-Entwickler, immer mehr – siehe zum Beispiel die Diskussionen über Applets in den vergangenen Monaten. Trotzdem ist es für viele Entwickler immer noch ein lästiges Randthema, mit dem sich andere auseinandersetzen sollen. Anders ist die jüngste Sicherheitslücke nicht zu erklären: Diesmal ist das mobile Betriebssystem Android betroffen, das die Java Cryptography Architecture für Verschlüsselung nutzt. In der jüngeren Vergangenheit waren bereits Bitcoins im Wert von mehreren Tausend Dollar gestohlen worden. Inzwischen ist die Ursache identifiziert: Mehrere Bitcoin-Wallets arbeiten mit einer mangelhaften, also nicht ausreichend zufälligen Initialisierung des Zufallszahlengenerators, sodass dieser nur eine deutlich beschränkte Menge an Zahlenfolgen generiert. Diese durch „brute force“-Angriffe zu erraten,

ist damit sehr leicht. Vermutlich sind nicht nur Bitcoin-Wallets von dieser Schwachstelle betroffen, sondern auch viele andere Apps. Aber hier sind die Konsequenzen sehr direkt und endgültig: Einmal gestohlen, gibt es keine Möglichkeit für die Nutzer, die Bitcoins zurückzuerhalten.

<http://www.heise.de/security/meldung/Androids-Verschlüsselung-angreifbar-1936181.html>

20. August 2013

Das JavaLand öffnet Ende März 2014 seine Tore

Heute wurde mit JavaLand eine neue Java-Konferenz angekündigt, die eine Lücke im deutschsprachigen Raum ausfüllt: eine mehrtägige, überregionale Konferenz von der Community für die Community. Die Konferenz wird von der DOAG und dem Heise-Verlag organisiert, in enger Kooperation mit den im iJUG vertretenen Java User Groups. Um dem JavaLand einen würdigen Rahmen zu verleihen, wurde das Phantasialand in der Nähe von Köln ausgewählt, das neben Themenpark-Attraktionen auch ein hervorragendes Konferenzzentrum hat. Geboten wird neben den klassischen Präsentationen auch ein großes Rahmenprogramm, das auf viel Interaktion der Teilnehmer untereinander sowie mit den anwesenden namhaften Java-Größen setzt. Wir hoffen auf zwei spannende und energiegeladene Tage!

<http://www.javaland.eu>

26. August 2013

Zweiter Java-EE-7-Application-Server zertifiziert:

JEUS 8

JEUS 8, die neueste Version des Application-Servers von TmaxSoft, ist als „Java EE

7 kompatibel“ zertifiziert worden. Damit bildet er eine bislang noch exklusive Gruppe zusammen mit GlassFish 4 (der zugleich auch für das abgespeckte Web-Profil zertifiziert ist). Weitere Application-Server wie WildFly (vielen sicher noch eher als JBoss AS bekannt) werden aber wohl bis Ende des Jahres folgen.
<http://www.tmaxsoft.com/product/productView.do>

30. August 2013

JPA vs. Hibernate?

Hibernate-Architekt Emmanuel Bernard plaudert im Interview mit „The Server Side“ über „JPA 2.0 EntityManager vs. the Hibernate Session: Which one to use?“. Wer erwartet hat, dass hier schwarz und weiß gemalt wird, sieht sich getäuscht: „Wir wünschten, wir hätten nur ein API“, lautet eine zentrale Aussage. Und: „Wir empfehlen, den Entity Manager zu nutzen.“ Man kann ja von da aus bei Bedarf immer noch auf die Hibernate-Session zugreifen („unwrap“) ... Allen am Thema Interessierten sei die URL mit Podcast des Interviews empfohlen.
https://blogs.oracle.com/theaquarium/entry/jpa_vs_hibernate

2. September 2013

Schleichendes Ende für Jigsaw?

Wird das für Java-Anwendungen und die JVM selbst geplante, aber immer wieder verschobene Modul-System „Jigsaw“ still und leise bis zur Unkenntlichkeit reduziert? Diesmal ist es kein Blog-Post, sondern nur eine E-Mail auf dem OpenJDK-Verteiler von Oracle-Plattform-Chefarchitekt Mark Reinhold, die einen weiteren Abstieg für Jigsaw andeutet. Es geht um Bedenken hinsichtlich Abwärts-Inkompatibilitäten durch einen separaten „Module Mode“ sowie um die „Einsparung“ der Auflösung von Abhängigkeiten (die laut Reinhold den bekannten Build Tools überlassen bleiben sollte). Es sei nur ein weiterer Prototyp und keine Aussage über das endgültige Ergebnis des Projekts „Jigsaw“. Aber natürlich ist die Stoßrichtung klar. Markus Eisele betrachtet die Aussagen in seinem Heise-Blog im Kontext der jüngeren Entwicklun-

gen, etwa bezüglich der neuen Compact Profiles in Java 8. Sein Fazit: Oracle braucht Jigsaw nicht mehr, um das vordringliche Ziel „schlanke JVMs für mobile Endgeräte“ zu erreichen. Wenn er Recht hat, wird Jigsaw ein schöner Plan bleiben.
<http://www.heise.de/developer/artikel/Modularisierung-von-Java-zweiter-und-letzter-Versuch-1946408.html>

5. September 2013

Und täglich grüßt das Thema „Applet-Sicherheit“

Oh je! Applets und Sicherheit werden wohl so schnell nicht aus den Schlagzeilen verschwinden. Die jüngste Entdeckung bezieht sich auf die Signatur für Applets (unsigned Applets werden ja inzwischen gar nicht mehr ohne Weiteres gestartet). Diese Signatur umfasst nicht alle wesentlichen Meta-Informationen zum Applet. So lässt sich beispielsweise der Name des Applet-Publishers nachträglich ändern, ohne dass die Signatur davon betroffen ist. Damit könnte ein Schadcode-Entwickler dem Nutzer anstelle der eigenen obskuren Firmenbezeichnung etwa den etwas vertrauenswürdigeren Namen „Oracle Corporation“ vorgaukeln.
<http://www.heise.de/security/meldung/Signierte-Java-Applets-unter-falscher-Flagge-1950043.html>

9. September 2013

JDK 8 Developer Preview

Mark Reinhold, Oracle Chef-Architekt für die Java-Plattform, hat die Freigabe der Developer Preview für das JDK 8 verkündet. Damit sind nun alle Entwickler aufgefordert, sich einmal die neuesten Features wie Lambda-Konstrukte sowie die neue JavaScript-Engine „Nashorn“ anzuschauen und das JDK möglichst viel zu testen. Insbesondere Open-Source-Projekte machen hoffentlich reichlich Gebrauch davon. Kommerzielle Projekte werden es wohl sowieso tun, wenn sie die neue Version direkt unterstützen wollen, damit Probleme, wie sie zum Beispiel mit Apache Lucene und dem Optimierer beim SE-7-Launch aufgetreten sind, nicht wieder zu spät entdeckt werden.
<http://mreinhold.org/blog/jdk8-preview>

10. September 2013

Mission Control in Hotspot JVM!

Das von JRockit bekannte Feature „Mission Control“, und der darin enthaltene „Flight Recorder“ sind in Oracle JDK 7 Update 40 enthalten. Dies ist Teil der vor langem angekündigten Strategie des Zusammenwachsens der beiden JVMs. Damit erreicht die Hotspot JVM nun auch das Niveau von JRockit, was die Mächtigkeit der Werkzeuge für Administratoren angeht, oder auch für Entwickler, die genau verstehen wollen oder müssen, was in ihrer JVM vorgeht. Mission Control war in der Vorgängerversion bereits für Kunden mit einem bestehenden Oracle-Support-Vertrag verfügbar, nun können aber alle davon profitieren. Ein Überblick darüber, was alles möglich ist, ist aus den Release Notes zu Update 40 heraus verlinkt. Eine gute Einführung in den Flugrecorder gibt folgende URL.
<http://hirt.se/blog/?p=370>

21. September 2013

Java ME 8 Early Access

Rechtzeitig zur JavaOne 2013 ist Java ME 8 in einer Early-Access-Version freigegeben worden. Viele Verbesserungen betreffen ME selbst, etwa „IPv6“- und „TLS1.2“-Unterstützung im Generic-Connection-Framework und das neue Embedded Profile, das unter anderem eine deutliche Reduzierung des Memory Footprint mit sich bringt. Moment: Das Thema hatten wir doch vor knapp drei Wochen erst unter dem Stichwort „Jigsaw“... Daneben ist dieses Release aber auch ein wichtiger Schritt für das Zusammenwachsen mit SE: Die neue Version ist mit dem Sprachumfang von SE 8 synchron, wenn auch (logischerweise) nicht alle SE-8-APIs unterstützt sind. Das Ausbauen der Fähigkeiten von ME ist für Oracle auch sehr wichtig. Jahrelang hat die Firma auf die Milliarden von „Feature Phones“ (aus der Marketingsprache übersetzt in die Telefone ohne besondere Features) als Umsatzbringer für Java ME gesetzt. Der Markt wird in absehbarer Zeit aber einbrechen, Smartphones sind auch im unteren Preis-Segment auf dem Vormarsch.
<http://terrencebarr.wordpress.com/2013/09/21/java-me-8-early-access-released-try-it-out>

22. September 2013

Projekt Avatar ist freigegeben – als OSS

Das auf der JavaOne 2011 angekündigte und im Jahr 2012 als Demo vorgestellte Projekt Avatar hat nun wohl die nötige Reife erlangt. Heute wurde es als Open Source freigegeben. Das auf JavaScript setzende Framework zur Entwicklung von Web-Anwendungen besteht aus drei Teilen: Unterstützung serverseitiger JavaScript-Services (basierend auf Node), Nutzung von Java-EE-Services aus den JavaScript-Services und einem clientseitigen Framework, das HTML5-Komponenten an beliebige serverseitige Services binden kann (sprachunabhängig). Client- und serverseitige Teile können entsprechend unabhängig voneinander genutzt werden. Für die Client-Seite tritt Avatar damit als weitere Alternative aus dem Hause Oracle neben JavaServer Faces und JavaFX. Mit der Verzahnung von JavaScript und Java EE liegt es voll im Trend, der auch auf der JavaOne 2013 zu sehen war. https://blogs.oracle.com/theaquarium/entry/project_avatar_is_open_source

26. September 2013

JavaOne 2013 beendet

Die JavaOne 2013 ist beendet – was hat sie der Community gebracht? Große Ankündigungen gab es dieses Jahr nicht, das war auch nicht zu erwarten. Ein immer wiederkehrendes Motto, gerade auch in den Keynotes, war das „Internet of things“. Der Begriff ist nun schon ziemlich alt – für IT-Verhältnisse –, scheint aber für Oracle so langsam wirklich umsatzrelevant zu werden. Passend dazu gab es natürlich die Vorstellung von Java ME 8 und dem Zusammenwachsen von SE und ME. Um Java auf Embedded-Geräten weiterzubringen, hat Oracle das „Java Platform Integrator Program“ gestartet. Damit werden Hersteller an der Portierung und Anpassung von Java auf spezielle Plattformen beteiligt – etwas weniger Kontrolle für Oracle, dafür aber mehr Skalierbarkeit. „Embedded“ auf dem Raspberry Pi liegt voll im Trend. Hierzu gab es ein dreitägiges Lab („Java Embedded Challenge“), in dem die Teilnehmer verschiedenste Anwendungsgebiete ausprobieren konnten, sowie viele Demos

auch in den Keynotes, darunter ein selbstgebautes Tablet und ein Schach-Roboter. Java SE 9 war natürlich auch ein Thema: Multimandanten-Fähigkeit soll Einzug in SE halten, Self-Tuning-Fähigkeiten sollen Administratoren das Leben leichter machen und einige andere Details mehr. Was mit „Jigsaw“ wird, vermag momentan niemand zu sagen – oder darf es nicht.

27. September 2013

Java jetzt im Raspberry-Pi-Image enthalten

Auf der JavaOne 2013 gibt es ja wieder spektakuläre Demos mit Java Embedded auf dem Raspberry Pi. Fast zwangsläufig hat nun das Raspberry-Team Java (genauer gesagt: das Oracle JDK 7) in sein Repository aufgenommen, sodass zukünftige Raspberry-Images direkt mit Java-Unterstützung kommen und keine manuellen Schritte mehr nötig sind. <http://www.heise.de/hardware-hacks/meldung/Oracle-Java-auf-dem-Raspberry-Pi-1968397.html>

29. September 2013

Dreißig Zeichen in Hadoop verarbeitet

Eine kleine Anekdote aus einer JavaOne-Session: Die Verarbeitung eines einzelnen Datensatzes mit dreißig Zeichen in einer extrem skalierbaren Hadoop-Umgebung, die einfache Berechnungen auf den Datensätzen durchführt, dauert: nein, nicht eine Nanosekunde oder Ähnliches, sondern vierunddreißig Sekunden. Das Ganze diente natürlich nur dazu, der Frage „How Big is Big?“ auf den Grund zu gehen und zu demonstrieren, dass man nicht mit Kanonen auf Spatzen schießen sollte. Vierunddreißig Sekunden waren die Zeit, die für die Initialisierung des Clusters erforderlich war. <https://weblogs.java.net/blog/editor/archive/2013/09/26/javaone-2013-impressions-4-how-utilize-hadoop-process-30-characters-just-34-seconds-and-how-do-mu>

10. Oktober 2013

JavaOne-Sessions teilweise online

Die meisten Sessions der JavaOne 2013 sind bislang nicht als Audio- oder Video-

Aufzeichnung verfügbar, aber wer nicht zur JavaOne konnte, kann sich zumindest die Keynotes im Video ansehen. Darüber hinaus gelangt man über „oracle.com/javaone“ („Search Sessions“) an eine suchbare Auflistung und kann über die Session-Details in den meisten Fällen auch die Folien abrufen. <http://medianetwork.oracle.com/media/channels>

September 2013

Gewinner der JavaOne Community Challenge stehen fest

Ende Juli wurde die Java-Community dazu aufgerufen, einen Slogan für die „JavaOne Community Challenge“ einzureichen. Die zehn vielversprechendsten Slogans standen auf java.net zur Abstimmung. Gewonnen hat Daniel Ellmerer mit dem Slogan: „Java: Collecting New Garbage Since 1995“. Die vier nächstplatzierten waren Wojciech Buras mit „One Java to Rule Them All“, Pang TeckChun mit „The World Runs on Java Technology“, Gerrit Grunwald mit „Java Is the Community“ und Eric M. Smith mit „Java: From the First Cup to the Last Line of Code“ <https://www.java.net/javatheme>

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.

Java macht wieder richtig Spaß

Wolfgang Taschner,
Chefredakteur Java aktuell

Im Rahmen der JavaOne 2013 gab es keine großen Neuigkeiten bezüglich Java – dafür ist der Java-Prozess zu transparent. Die kommenden Features sind deshalb bereits weitgehend bekannt. Dennoch tut sich einiges, um die Sprache einfacher, performanter und attraktiver für die kommenden Jahrzehnte zu machen.

Der Slogan aus dem letzten Jahr ist geblieben: „Make the Future Java“. Peter Utschneider, Oracle Vice President, Java Product Management, sieht darin die Herausforderungen, die die Industrie mithilfe von Java gemeistert hat, um Innovationen hervorzubringen. Er zählte zu Beginn seiner Keynote erst mal eine Reihe von Superlativen auf: Neun Millionen Entwickler sorgen weltweit dafür, dass Java auf allen Blu-ray Disc Playern, auf drei Milliarden Mobiltelefonen, auf 97 Prozent aller PCs in den Unternehmen sowie auf 125 Millionen Fernsehgeräten läuft. Java ist damit die attraktivste Software-Plattform.

Eine der größten Herausforderungen der kommenden Jahre ist die Gestaltung des „Internets der Dinge“, bei dem es darum geht, dass zunehmend in unserem Leben intelligente Gegenstände verwendet werden, die nützliche Informationen sammeln, aggregieren und letztendlich unser Leben erleichtern. Oracle hat dafür ein Programm namens „Oracle Java Platform Integrator“ aufgesetzt, damit alle



Kommunikation unter freiem Himmel in der eigens für den Verkehr gesperrten Mason Street

Hardware-Hersteller in der Lage sind, Java schnell und unkompliziert auf ihrer Plattform zu integrieren. Weitere bedeutende Aufgaben sind die Bereiche „Big Data“ und „Cloud“.

Ausblick auf Java SE8

Peter Utschneider ging anschließend nochmals die Fakten durch, die ebenfalls im März 2014 mit Java SE8 zu erwarten sind. An oberster Stelle stehen die Lambda-Expressions, die mit einer neuen Syntax Java-Code besser lesbar und wartbar machen sowie durch Parallel-Ausführung eine bessere Performance bringen. Das Projekt „Nashorn“ ist eine neue hochperformante JavaScript-Engine. Unter den rund 50 neuen Features von Java SE8 ist noch das Date&Time-API hervorzuheben. Eine Demo zeigte eindrucksvoll, wie heute schon alle Varianten von Client-Technologien (HTML5, JavaFX 8 3D und Java ME8) erfolgreich mit Java EE 7 zusammen-

arbeiten. SE8-Spezifikation und von JDK8 kümmern. Neben Oracle tragen weitere Unternehmen, Wissenschaftler und Einzelpersonen zur Weiterentwicklung von Java bei, darunter Freescale, Linaro und Square.

Auch das erst kürzlich vorgestellte Java EE kam nochmals zur Sprache, vorgetragen von Cameron Purdy, Oracle Vice President, Cloud Application Foundation. Er stellte das schon fast in Vergessenheit geratene Projekt „Avatar“ vor, das die Nutzung von JavaScript in HTML5-Anwendungen sowohl auf dem Client als auch auf dem Server wesentlich vereinfachen wird.

Mark Reinhold, Oracle Chefarchitekt der Java Platform Group, zeigte zum Abschluss nochmals auf, wie sich das gesamte Java-Biotop wieder zu einer einheitlichen Plattform entwickelt – getreu dem ursprünglichen Motto „write once, run anywhere“ oder, wie es der Java-Erfinder James Gosling einst nannte, „The Feel of Java“.

Oracle WebLogic Server 12c – Zuverlässigkeit, Fehlertoleranz, Skalierbarkeit und Performance

Sylvie Lübeck, Oracle Deutschland B.V. und Co. KG

Im Rahmen der WebLogic-12c-Artikelserie geht es diesmal um den Aufbau einer hochverfügbaren Umgebung – zuverlässig, fehlertolerant, skalierbar und performant für alle Anwendungen, was auch mit der Abkürzung „RASP“ (reliable, available, scalable, performant) bezeichnet wird. Basis bilden die Cluster-Technologie des WebLogic Servers, die Session-Replikation und das optimierte Zusammenspiel mit der Oracle-Datenbank.

Spielen für eine Anwendung der Betrieb in einer hochverfügbaren Architektur und die Skalierbarkeit eine wichtige Rolle, ist der WebLogic Server genau die richtige Wahl. Schon aus der Historie heraus werden Architektur-Konzepte wie der Betrieb im Cluster unterstützt. Mit WebLogic Server 12.1.2 sind sogenannte „dynamische Cluster“ sowie die Integration mit den letzten Neuerungen der Oracle-Datenbank 12c hinzugekommen. Die Grundbegriffe des WebLogic Servers sind im Artikel „WebLogic-Grundlagen – die feinen Unterschiede“ in der letzten Ausgabe beschrieben [1].

Die Basis – der WebLogic Cluster

Ein Cluster besteht aus mehreren Managed Servern, die alle auf der gleichen WebLogic-Server-Version und dem gleichen Patch-Level basieren. Alle Server gehören immer genau zu ein und derselben Domäne, wobei eine Domäne durchaus mehrere Cluster beinhalten kann. Dem Client gegenüber erscheint ein Cluster wie ein einzelner Server mit den bereitgestellten Anwendungen.

Mit der WebLogic-Server-Cluster-Architektur werden die Anforderungen hinsichtlich Skalierbarkeit und Hochverfügbarkeit unterstützt. Ist eine erhöhte Last zu erwarten, kann der Administrator dem Cluster weitere Managed Server hinzufügen. Die neuen Server werden auf einer vorhandenen oder einer neuen „Machine“ konfiguriert und gestartet. In der Regel sind die Anwendungen auf dem kompletten Cluster bereitgestellt. Dadurch kann bei Ausfall eines Managed Servers ein anderer Server im Cluster den Betrieb der Anwendung

übernehmen. Bei einem Hardware-Ausfall lassen sich Managed Server auf einen anderen physikalischen Server migrieren. Der Umzug wird vom Administrator über die Administrations-Konsole oder per Kommandozeile durchgeführt.

Die meisten Anwendungen werden homogen auf dem Cluster bereitgestellt und bei einem Ausfall transparent abgefangen. Im Gegensatz dazu werden für „Java Messaging Services“ (JMS) und „Java Transaction API“ (JTA) Dienste die Lastverteilung und Hochverfügbarkeit durch Redundanz erreicht. Der JMS-Server wird auf dem Cluster konfiguriert, der wiederum je Managed Server eine dedizierte Queue zur Punkt-zu-Punkt-Kommunikation verwaltet. Diese sind zu einer „distributed Queue“ zusammengefasst. JMS-Topics lassen sich nach dem „publish subscribe“-Mechanismus als „replicated distributed“ oder als „partitioned distributed“ Topic konfigurieren. Die JMS-Nachrichten können in einem „persistent store“ auf dem Dateisystem oder in der Datenbank gesichert werden. Insgesamt hat sich die Konfiguration von JMS im Cluster durch Elastic JMS ab 12.1.2 erheblich vereinfacht.

Die verfügbaren Server, Dienste und Anwendungen sind im clusterweiten „Java Naming and Directory Interface“-Baum (JNDI) eingetragen. Jeder Managed Server hält seine eigene lokale Kopie des JNDI-Baums vor, der bei Veränderungen entsprechend aktualisiert wird.

Die Managed Server innerhalb eines Clusters unterhalten sich miteinander, um möglichst schnell den Ausfall eines Servers mitzubekommen. Seit WebLogic 11g

kommt standardmäßig das „Unicast“-Kommunikations-Modell zum Einsatz, da es weniger Netzwerkverkehr produziert und einfacher zu konfigurieren ist. Zudem kann „Unicast“ unabhängig von der Netzwerk-Topologie (Local Area Network (LAN), Metropolitan Area Network (MAN) oder Wide Area Network (WAN)) ohne spezielle Konfiguration betrieben werden, da die Kommunikation auf TCP/IP-Sockets basiert.

Alternativ wird weiterhin das „Multicast“-Kommunikations-Modell unterstützt, bei dem der „heartbeat“ an alle Managed Server des Clusters gesendet wird. Bei „Unicast“ werden die Managed Server des Clusters in Gruppen von jeweils maximal zehn Servern organisiert und je Gruppe ein Master definiert. Dieser erhält von den Managed Servern, die seiner Gruppe angehören, alle zehn Sekunden einen „heartbeat“. Alle fünfzehn Sekunden wird geprüft, ob ein „heartbeat“ fehlt, was zur Folge hätte, dass der Managed Server aus der Cluster-Member-Liste entfernt wird. Da die Definition der Cluster-Gruppen auf Basis der alphanumerisch sortierten Server-Namen erfolgt, empfiehlt es sich, den Namen nach einer einfachen Nomenklatur wie „server1“, „server2“, ..., „serverN“ durchzuführen.

Fällt der Master selbst aus, wird automatisch der zweitälteste Server innerhalb der Gruppe als neuer Master festgelegt und alle weiteren Master über die Änderung informiert. Dadurch ist es möglich, sehr effizient auch im laufenden Betrieb zusätzliche Server hinzuzufügen. Dies kann durch „server cloning“ erfolgen. Dabei sollte möglichst vermieden werden,

dass eine neue Gruppe entsteht und diese Änderung allen Gruppen-Mastern mitgeteilt werden muss, indem man beispielsweise bei bisher vierzehn Managed Server – einer Gruppe mit zehn und einer mit vier Servern – maximal sechs hinzufügt. Immer wenn ein neuer Zehner angebrochen wird, erfolgt automatisch eine Reorganisation, was im laufenden Betrieb etwas dauert.

Bei der Reorganisation werden die Server wieder nach alphanumerischer Sortierung in Zehner-Gruppen eingeteilt. Der erste Server wird als Master Server definiert. Zu guter Letzt informieren sich die Server untereinander, wer Master Server ist. Dies würde etwa erfolgen, wenn in dem eben beschriebenen Beispiel acht weitere Managed Server dem Cluster hinzugefügt werden. Diese Einschränkung gilt nicht für dynamische Cluster, Cluster im heruntergefahrenen Zustand und Multicast Cluster.

Fällt ein Managed Server aus, ist die Frage, ob die Anwendung nahtlos weiterlaufen kann. Zum einen muss geklärt werden, welcher Server die Anfrage weiterverarbeitet, und zum anderen, ob – und wenn ja, wie – der Zustand einer laufenden Session beziehungsweise Transaktion dem neuen Managed Server bekannt gemacht wird. Hier kommen die WebLogic-Server-Funktionen „Session State Replikation“ und „Application Continuity“ mit ins Spiel, die weiter unten beschrieben sind.

Nach dem ersten Client-Aufruf wird entschieden, welcher Server als Primär-beziehungsweise Sekundär-Server dient. Für Servlets und JSPs ist diese Information je nach Konfiguration im Client-Cookie oder in der URL festgehalten. Bei EJBs und RMI-Objekten erfolgt der Eintrag in den „replica-aware stubs“. Fällt jetzt innerhalb des Clusters ein Managed Server aus, werden die Rollen der verfügbaren Managed Server für die betroffenen Client-Anfragen reorganisiert. Um eine Konkurrenz-Situation auf Netzwerk-Ebene zu vermeiden, empfiehlt es sich, sich einen eigenen „Network Channel“ für die Inter-Cluster-Kommunikation zu konfigurieren.

In WebLogic 12.1.2 ist das Konzept dynamischer Cluster neu eingeführt worden. Die Konfiguration erfolgt über die „Administration Server Console“, das WebLogic-Scripting-Tool (WLST) oder Enterprise Manager Cloud Control. Alle Managed Server des dynamischen Clusters basieren auf ein und demselben Server-Template, das die Grundlage zur Generierung der Managed-Server-Konfiguration wie Server-Name, Port und Listen-Adresse enthält. Auf dieser Basis lässt sich die Anzahl der Managed Server eines dynamischen Clusters sehr einfach erhöhen oder verringern.

Bei der Lastverteilung geht es um eine Verteilung der anfallenden Prozesse, die je nach Algorithmus gleichmäßig (round-ro-

bin), gewichtet oder zufällig erfolgt. Auch hier ist die Voraussetzung, dass mindestens eine Kopie der Objekte beziehungsweise Anwendungen auf einem anderen Server zur Verfügung steht. Technisch weiß der WebLogic Server durch die Interkommunikation per Unicast, welche Server verfügbar sind, und über „IP sockets“ und JNDI-Baum, welche Objekte beziehungsweise Anwendungen wo zur Verfügung stehen.

Für die Verteilung von HTTP- (Servlets und JSPs) und von REST-Anfragen innerhalb eines Clusters ist ein „Loadbalancer“ erforderlich. Für die Test- und Entwicklungs-Umgebung kann das mitgelieferte ProxyServlet verwendet werden. Dieses wird im Rahmen der Domain-Konfiguration auf einem zusätzlichen Managed Server installiert. Alternativ kann auch ein Software-Loadbalancer zum Einsatz kommen. WebLogic liefert ein Plug-in, das neben dem Oracle-HTTP-Server für die gängigsten HTTP-Server zertifiziert ist (siehe WebServer-Zertifizierung in [2]). Die Anfragen werden nach dem „round robin“-Algorithmus auf die Server verteilt. In einer Produktiv-Umgebung empfiehlt sich die Verwendung eines Hardware-Loadbalancer, der in der Regel zur Verteilung der Anfragen Last-Informationen der einzelnen Knoten berücksichtigt. Insgesamt ist der Loadbalancer sowohl zur Verteilung der

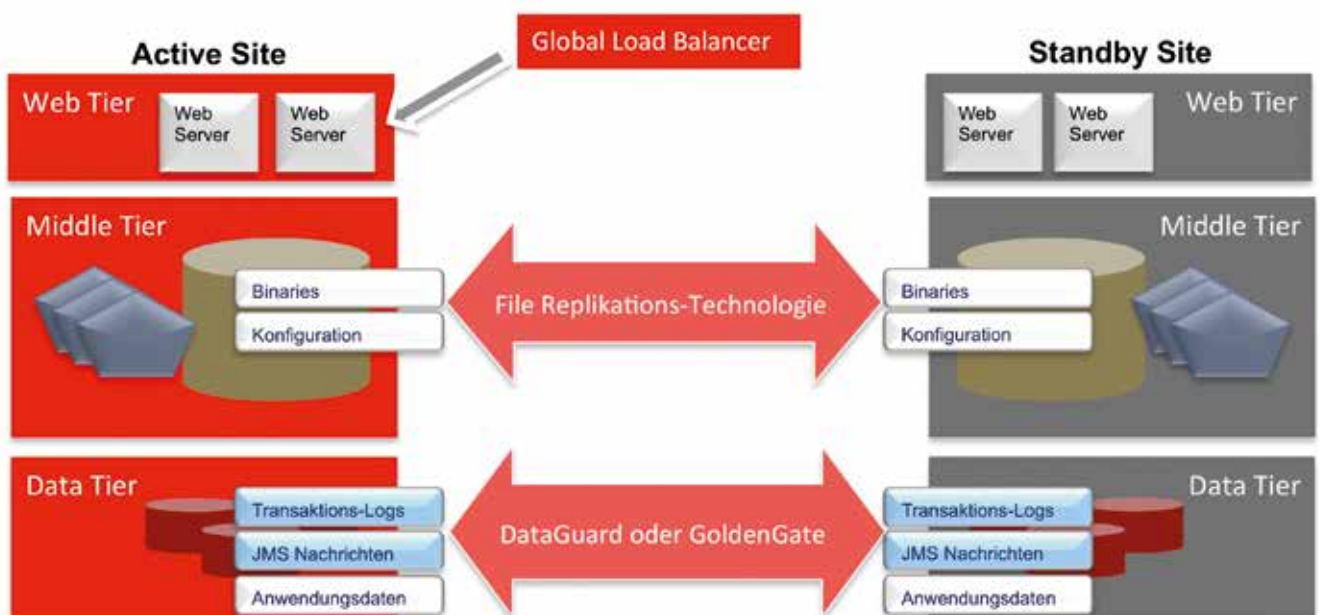


Abbildung 1: WebLogic Server und Maximum Availability Architecture

Last als auch für das Umleiten im Falle eines Ausfalls zuständig.

Üblicherweise werden WebLogic-Server-Cluster im Active-Active-Mode betrieben. Das bedeutet, dass zwei oder mehr Managed Server im Cluster sind, die die gleichen Anwendungen bereitstellen. Bei einem Ausfall kann ein zweiter Server im Cluster die Anfragen übernehmen. Dabei sollte berücksichtigt werden, dass die Managed Server mindestens auf zwei voneinander physisch getrennten Maschinen laufen. Alternativ ist auch eine Active-Passive-Variante möglich. Dies wird beim WebLogic Server mit der „Whole Server Migration“-Funktion oder durch eine Umgebung realisiert, die im „Standby“-Modus bereitsteht. Die weiteren Ebenen, die im übergeordneten Konzept der „Maximum Availability Architecture“ zu berücksichtigen sind, sind in [Abbildung 1](#) dargestellt.

Application Failover durch Session-Replikation

Als Nächstes beschäftigen wir uns damit, was passiert, wenn eine Applikations-Komponente ausfällt. Die Idee von „Application Failover“ ist, dass eine Kopie der Applikations-Komponente den laufenden Prozess übernimmt und bis zum Ende ausführt. Der WebLogic Server hält zu jeder Zeit im JNDI-Baum fest, wo welche Objekte zur Verfügung stehen.

Zusätzlich muss möglichst der Zustand des Prozesses bekannt sein und überge-

ben werden, um nahtlos die Ausführung der Anwendungskomponente sicherzustellen. Abgesehen von der Verwendung von Client-Cookies oder URL-Rewriting, um den Primär- und Sekundär-Server parat zu haben, bietet der WebLogic Server die Möglichkeit, den Session-Zustand zu replizieren. Dabei werden alle clusterfähigen Objekte wie Servlets, JSPs, EJBs, RMI-Objekte und JMS-Ziele berücksichtigt. Die technische Umsetzung der Session-Replikation ist je nach Objekt-Typ unterschiedlich.

Bei Servlets und JSPs kann der HTTP-Zustand pro Client im Arbeitsspeicher, im Filesystem oder in der Datenbank festgehalten sein. Die standardmäßig verwendete Variante bei einem WebLogic-Cluster ist dessen Speicherung im Arbeitsspeicher. Dies gilt auch für REST-Anfragen. Dazu müssen diese in Form serialisierbarer Objekte vorliegen. Als Sekundär-Server wird ein Server aus dem Cluster verwendet, der vorzugsweise auf einer anderen Maschine läuft. Der Administrator kann durch die Definition von Replikations-Gruppen die bevorzugten Sekundär-Server konfigurieren.

Beim ersten Aufruf eines Servlets oder von JSPs über den Software-Loadbalancer (via „HttpClusterServlet“) wird entschieden, welcher Server die Rolle des Primär- beziehungsweise Sekundär-Servers übernimmt. Diese Information wird auf Client-Seite in einem Cookie oder alterna-

tiv über „URL-Rewriting“ festgehalten. Zugleich werden alle HTTP-Session-Zustände auf dem Sekundär-Server, im Beispiel von [Abbildung 2](#), auf dem Server „B“ repliziert. Fällt der Server „A“ aus, wird die Anfrage an den Server „B“ umgeleitet und der Client kann transparent weiterarbeiten. Bei einem Hardware-Loadbalancer erfolgt die Umleitung im Failover-Fall über den definierten Algorithmus eventuell auf den Server „C“. Über die Information im Cookie beziehungsweise der URL ist bekannt, dass der HTTP-Session-Zustand auf dem Server „B“ liegt und dem neuen Primär-Server „C“ bereitgestellt werden kann.

Wird aus Gründen der Hochverfügbarkeit an einem zweiten Standort über Metropolitan Area Network (MAN) oder Wide Area Network (WAN) ein Cluster bereitgehalten, muss für den Fehlerfall ein globaler Loadbalancer die Verteilung der Anfragen übernehmen (siehe [Abbildung 3](#)). Dabei ist es wichtig, dass die Session-ID mitprotokolliert wird, damit Anfragen innerhalb derselben Session wieder an den bisherigen Server geleitet werden.

Für die Session-State-Replikation über MAN oder WAN muss ein Netzwerk-Channel konfiguriert sein. Dabei kann, je nach Netzwerk-Latenz, eine synchrone oder asynchrone Session-State-Replikation zum Einsatz kommen. Im WAN wird in der Regel eine asynchrone Replikation aufgesetzt, bei der der Session-Zustand in der Datenbank festgehalten ist. Weitere Einzelheiten

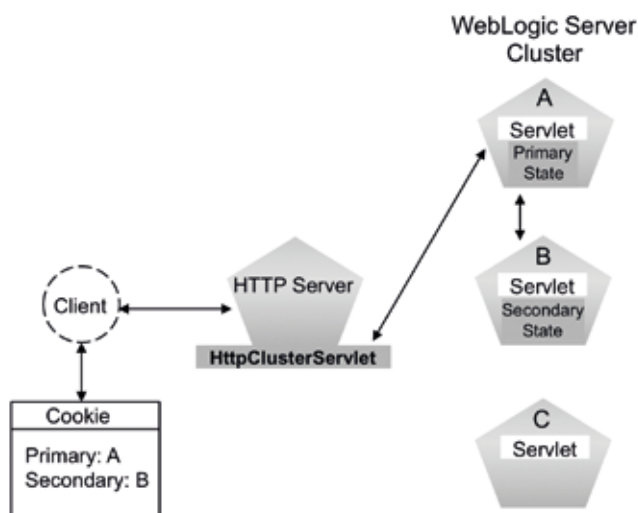


Abbildung 2: HTTP-Session-Replikation mit Software-Loadbalancer/ HTTP Proxy

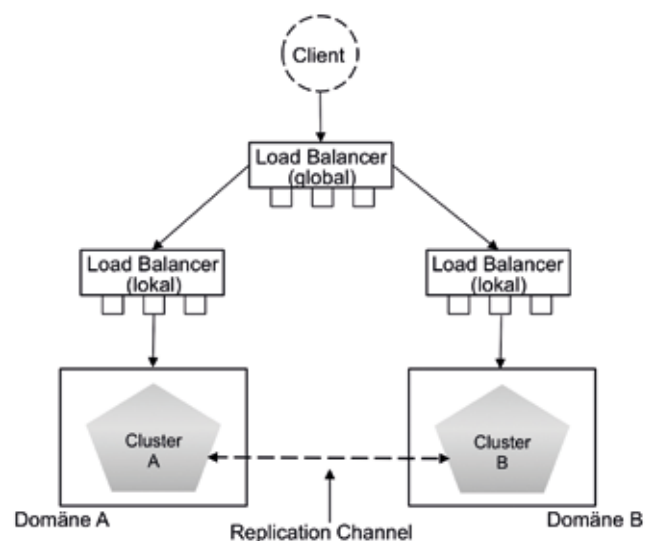


Abbildung 3: Cross-Cluster-Replikation

sind in der Dokumentation [3] in Kapitel 6 näher beschrieben.

„Coherence*Web“ sollte in folgenden Fällen als Alternative zur WebLogic-HTTP-Session-Replikation in Betracht gezogen werden, wenn ...

- ... die Anwendung mit großen HTTP-Session-State-Objekten arbeitet
- ... die Anwendung durch die Speicherung von HTTP-Session-Objekten an die Speichergrenzen stößt („JVM Heap Size“)
- ... der Session-Zustand anwendungsübergreifend zur Verfügung stehen soll
- ... das Backend-System durch Speicherung des HTTP-Session-Zustands in Coherence entlastet werden soll

Bei Enterprise Java Beans und RMI-Objekten erfolgen die Failover-Funktionen über die „replica-aware stubs“. Geht der Aufruf zu einem Dienst, der gerade nicht verfügbar ist, bemerkt der „Stub“ dies und leitet den Aufruf zu einer anderen EJB- beziehungsweise RMI-Objekt-Kopie, auch „replica“ genannt, weiter.

Für „Pinned Services“ wie JMS- und JTA-Transaction-Recovery-Dienste, deren Instanz jeweils nur auf einem Server zur Verfügung steht, bietet der WebLogic Server die Erkennung eines Ausfalls und wahlweise die automatische oder manuelle Service-Migration. JTA Transaktionen

Recovery wird mit WebLogic durch die Sicherung von Transaktions-Logs (TLOG) in der Datenbank realisiert. Der Zustand wird damit konsistent zur Anwendung gesichert und kann auch über Datenbank-Replikationsmechanismen in ein zweites Rechenzentrum gespiegelt werden.

Application Continuity

Bricht eine Verbindung ab, sind dennoch sowohl Datenbank als auch Netzwerk weiterhin verfügbar wäre es von Vorteil, auf einer neuen Verbindung weiterarbeiten zu können. Steht im Hintergrund ein Real Application Cluster (RAC), ist die Chance relativ hoch, dass selbst bei Ausfall der Datenbank-Instanz eine neue Verbindung zu einer der verfügbaren RAC-Instanzen aufgebaut werden kann. Jetzt bleibt nur die Frage, wie die bisherige Arbeit nahtlos weitergeführt werden kann. Woher weiß die neue JDBC-Verbindung, was bisher gelaufen ist? Hier kommt die Funktion „Application Continuity“ der neuen Datenbank 12c zum Tragen, die auch „replay“ genannt wird. Damit ist gemeint, dass alle Aktionen, die bisher ausgeführt wurden, auf der neuen JDBC-Verbindung wiederholt werden. Da sich inzwischen auch die Daten eventuell verändert haben, müssen nicht nur die Aktionen, sondern auch die Daten mitprotokolliert werden. So kann über die neue JDBC-Verbindung per „replay“ alles bis zum Zeitpunkt des Abbruchs wiederhergestellt

werden. Transaction Guard, ein zuverlässiges Protokoll und Werkzeug der Oracle-Datenbank 12c, wird dabei verwendet, um sicherzustellen, dass eine abgeschlossene Transaktion nicht erneut ausgeführt wird.

Voraussetzung für die Nutzung dieser Funktion sind die 12c-Datenbank- und 12c-JDBC-Treiber. Der datenbankseitige Service wird unter Verwendung der Optionen „failover_type=TRANSACTION“ und „commit_outcome=true“ angelegt. Seitens WebLogic ist lediglich die Verwendung des „replay“-JDBC-Treibers erforderlich: „oracle.jdbc.replay.OracleDataSourceImpl“ anstelle von „oracle.jdbc.OracleDriver“.

In der Anwendung sind keine Anpassungen erforderlich. An den Client kommunizierte Fehler werden durch diese Funktion minimiert, aber nicht gänzlich ausgeschlossen. Daher sollten diese weiterhin in der Anwendung abgefangen werden. Das White Paper zum Thema „Application Continuity“ [4] liefert weitere Details über diese neue Funktionalität.

Zusammenspiel mit Oracle RAC

„Active Gridlink for RAC“ bildet das Bindeglied zwischen WebLogic Server und Oracle-RAC, das Lastverteilung, Skalierbarkeit und Hochverfügbarkeit für JAVA-EE-Anwendungen bestmöglich unterstützt. Bei Ausfall einer RAC-Instanz erhält der Universal Connection Pool (UCP) eine „Fast Application Notification“-Nachricht (FAN).

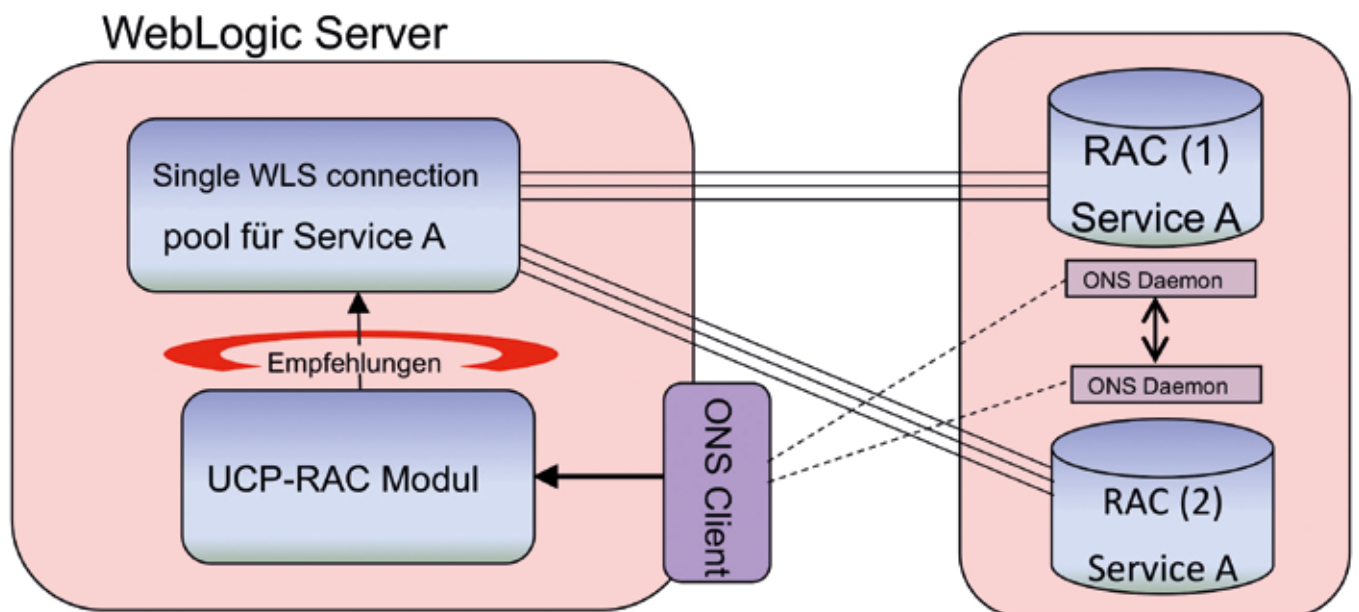


Abbildung 4: RAC-ONS-Kommunikation mit WebLogic Server

Die Anwendung bleibt verfügbar und wird über eine andere Verbindung aus dem Connection Pool an eine der verfügbaren Instanzen im RAC umgeleitet. Dieser Vorgang wird als „Fast Application Failover“ bezeichnet.

Nicht abgeschlossene SQL-Transaktionen der abgebrochenen Verbindungen müssen durch die ohnehin zu implementierende Fehlerbehandlung seitens der Anwendung abgefangen werden. Fehlerhafte Verbindungen werden automatisch aus dem Connection Pool entfernt. Dadurch ist sichergestellt, dass der Connection Pool nur gültige oder reservierte Verbindungen vorhält, ohne diese gegen die Datenbank testen zu müssen.

Ist zu einem späteren Zeitpunkt die RAC-Instanz wieder verfügbar, wird dies automatisch erkannt. Auch neue Instanzen, die etwa aus Gründen der Skalierbarkeit hinzukommen, werden automatisch eingebunden. So ist eine optimale Ressourcen-Ausnutzung des Gesamtsystems sichergestellt. Ein Administrator muss dabei nicht aktiv in den Prozess eingreifen, da diese Information via „Oracle Notification Service“ (ONS) dem WebLogic Server mitgeteilt wird und der UCP automatisch zwischen den verfügbaren RAC-Knoten ausbalanciert wird (siehe Abbildung 4).

Zur optimalen Lastverteilung dient die Funktion „Runtime Connection Load Balancing (RCLB)“. WebLogic erhält über FAN-Ereignisse die Information, wie die Last der einzelnen Server aktuell ist, und kann die Verteilung abhängig von der verfügbaren Kapazität in Bezug auf CPU, Verfügbarkeit und Antwortzeit vornehmen.

Seitens WebLogic wird eine GridLink-Datasource angelegt, die alle verfügbaren Server oder die „Single Client Access Name“-Adresse (SCAN) des RACs nutzt. Darüber hinaus werden der zu verwendende Service, der Port des Remote-Listeners und der „Oracle Notification Service“-Port (ONS) konfiguriert. Ab WebLogic 12.1.2 wird im Zusammenspiel mit der Oracle-Datenbank 12c der ONS-Listener-Port automatisch über den JDBC-Treiber an WebLogic bekanntgegeben.

Bei Web-Sessions und globalen Transaktionen gibt es jeweils die Möglichkeit, eine Affinitäts-Regel zu definieren. Die Affinität zum RAC-Server wird beim ersten Aufruf festgelegt. Bei einer Web-Session (Session

Affinity) bedeutet das, dass weitere Aufrufe im Rahmen ein- und derselben Web-Session über die gleiche JDBC-Verbindung an den gleichen Server geleitet werden. Im Falle einer globalen Transaktion (XA Affinity) ist sichergestellt, dass alle Aufrufe innerhalb der globalen Transaktion an den gleichen RAC-Knoten gehen.

Das waren die grundlegenden Funktionen und Neuerungen, die beim Aufbau einer RASP-Umgebung mit WebLogic eine Rolle spielen. Weitere Aspekte, die im Bereich Hochverfügbarkeit entscheidend sind, werden sehr gut in den Dokumenten unter [5] und [6] besprochen.

Hinweis: Der Artikel in der nächsten Ausgabe wird sich mit der Entwicklung unternehmenskritischer Anwendungen für Oracle WebLogic Server befassen.

Links

- [1] <http://www.doag.org/home/aktuelle-news/article/weblogic-grundlagen-die-feinen-unterschiede.html>
- [2] <http://www.oracle.com/technetwork/middleware/ias/downloads/fusion-certification-100350.html>
- [3] <http://docs.oracle.com/middleware/1212/wls/CLUST>
- [4] <http://www.oracle.com/technetwork/database/database-cloud/private/application-continuity-wp-12c-1966213.pdf>
- [5] <http://www.oracle.com/technetwork/database/features/availability/fusion-middleware-maa-155387.html>
- [6] <http://docs.oracle.com/middleware/1212/core/ASHIA/fmwha.htm>

Sylvie Lübeck

Sylvie.Luebeck@oracle.com



Sylvie Lübeck arbeitet bei der ORACLE Deutschland B.V. & Co. KG in der Abteilung „Business Unit Server Technologies – Fusion Middleware“, die deutschlandweit die Middleware-Themen technisch und vertriebsunterstützend verantwortet. Ihr Schwerpunkt ist der Oracle WebLogic Server.

Oracle verändert seine GlassFish-Strategie

GlassFish-Produktmanager John Clinigan hat im GlassFish Blog „Aquarium“ angekündigt, dass es mit GlassFish 4.0 keine kommerzielle Version von GlassFish geben wird. Neue und bestehende Kunden erhalten gemäß der «Lifetime Support Policy» von Oracle weiterhin Support für die kommerziellen Editionen von GlassFish Server 2.1.x und 3.1.x.

Nach der Übernahme von Sun hatte Oracle zwei Applikationsserver: WebLogic als rein kommerziellen High-End-Server und GlassFish als kostenlosen Open-Source-Applikationsserver mit Upgrade-Pfad auf eine kommerzielle Variante. Oracle ist nun anderen Herstellern wie IBM im Java-EE-Umfeld gefolgt, nur noch einen kommerziellen Applikationsserver anzubieten. Kunden können somit den Oracle WebLogic Server als kommerzielle Alternative nutzen.

Oracle investiert weiterhin in die Open-Source-Edition von GlassFish als entwicklerfreundliche Distribution und in die Zukunft von Java EE durch Bereitstellung der Referenz-Implementierung. Zudem fordert Oracle explizit die Community zur Teilnahme im JCP auf, etwa durch das Adopt-a-JSR-Programm und GlassFish Contributions.

Siehe auch https://blogs.oracle.com/brunoborges/entry/6_facts_about_glassfish_announcement, <https://glassfish.java.net/adoptajsr/> und <https://glassfish.java.net/contribute.html>.

Unsere Inserenten

aformatik Training und Consulting GmbH & Co. KG, www.aformatik.de	S. 3
Netpioneer GmbH www.netpioneer.de	S. 65
TEAM GmbH www.team-pb.de/	S. 31
DOAG e.V. www.doag.org	U2
DOAG e.V. www.doag.org	U4

Solr und ElasticSearch – Lucene on Steroids

Florian Hopf, Freiberuflicher Softwareentwickler

Lucene ist der De-facto-Standard für die Implementierung von Suchlösungen auf der JVM. Statt Lucene direkt zu nutzen, werden vermehrt die darauf aufbauenden Suchserver Apache Solr und ElasticSearch eingesetzt. Der Artikel stellt diese vor.

In der letzten Ausgabe haben wir gesehen, wie man mit der Bibliothek Lucene das Grundgerüst einer flexiblen Suchlösung implementiert. Lucene kümmert sich um die Speicherung und das Auslesen von Textinhalten aus einem invertierten Index. Das Verhalten der Suche wird maßgeblich durch den Analyzing-Prozess beeinflusst. Vorgefertigte oder eigene Analyzer überführen dabei die Textinhalte in die im Index gespeicherten Terme. Die verfügbare Query-Syntax ist zur Suche im Index nutzbar.

Beim Einsatz von Lucene können sich allerdings eventuell noch einige Schwierigkeiten ergeben. Wer bereits die eine oder andere Anwendung auf Basis von Lucene geschrieben hat, weiß, dass derselbe Code oft immer wieder geschrieben werden muss. Außerdem bietet Lucene von Haus aus keinen Mechanismus an, um mit sehr großen Datenmengen umzugehen oder einen Index aus Gründen der Lastverteilung oder Ausfallsicherheit redundant auszulagern. Schließlich gibt es oft auch noch den Wunsch, die Funktionalität von Lucene in einer nicht auf der JVM laufenden Sprache zu verwenden. Für alle diese Probleme besteht die Lösung darin, einen auf Lucene aufsetzenden Suchserver einzusetzen.

Dieser Artikel stellt die beiden Open-Source-Lösungen Apache Solr und ElasticSearch vor. Als Beispiel-Anwendung dient uns wieder die Indizierung von Vortragsbeschreibungen mit ihren Meta-Informationen wie „Sprecher“ oder „Datum“. Im ersten Teil werfen wir einen Blick auf den Platzhirsch Apache Solr, bevor wir uns danach anschauen, was der Newcomer ElasticSearch anders macht.

Apache Solr

Im Gegensatz zu Lucene ist Solr keine Bibliothek, sondern eine Anwendung, die die

Funktionalität von Lucene per HTTP zur Verfügung stellt. Solr kann entweder als Web Application Archive (WAR) in einen Servlet-Container eingesetzt oder direkt per „embedded Jetty“ gestartet werden. Die eigene Anwendung greift dann zum Indizieren und Suchen per HTTP auf Solr zu (siehe [Abbildung 1](#)). Zum Austausch können verschiedene Formate wie XML, JSON, CSV oder bei einer Java-Anwendung ein spezielles binäres Format zum Einsatz kommen. Es existieren zahlreiche Client-Implementierungen für unterschiedliche Programmiersprachen; für Java wird SolrJ direkt vom Solr-Team weiterentwickelt.

Das Schema der in Solr indizierten Daten muss vor der Verwendung in der Konfigurationsdatei „schema.xml“ hinterlegt sein. Dabei werden den Feldern der Dokumente Typen zugewiesen, wobei sowohl primitive Typen als auch selbst konfigurierte möglich sind. Für eigene Typen lässt sich eine Analyzer-Kette für das Indizieren und die Suche hinterlegen. Bei jedem Zugriff auf dieses Feld wird dann automatisch der konfigurierte Analyzer verwendet.

[Listing 1](#) zeigt einige Felder unserer Beispiel-Dokumente und die Konfiguration des Typs für deutschen Text. Wie von Lucene bekannt, werden am Feld die Attri-

bute „indexed“ und „stored“ gesetzt, um zu bestimmen, ob die Analyzer-Kette durchlaufen werden muss und ob die Originalwerte bei der Ausgabe der Suchergebnisse verfügbar sein sollen.

Nach Konfiguration unseres Schemas und einem Neustart können wir, wie in [Listing 2](#) angegeben, Daten indizieren. Der Zugriff auf Solr über SolrJ erfolgt über Implementierungen des Interface „SolrServer“, das alle notwendigen Methoden zur Verfügung stellt. Vergleichbar mit der Lucene-Implementierung werden Document-Instanzen erzeugt, denen unsere Felder hinzugefügt werden. Es sind jedoch jetzt weniger Informationen anzugeben. Ob ein Feld „analyzed“ ist oder gespeichert wird, ist nun direkt in der Solr-Konfiguration hinterlegt. Auch müssen wir an keiner Stelle im Client den Analyzer angeben, dieser wird durch den Feldtyp automatisch bestimmt. Dadurch sind für Anpassungen an der Suchlogik oft nur Änderungen an der Konfiguration in Solr notwendig.

Neben dem Schema sind für die Suche noch weitere anwendungsspezifische Konfigurationen notwendig. In „solrconfig.xml“ können neben diversen Settings wie Lucene-Einstellungen oder Caching-Optionen auch die zentralen „RequestHand-

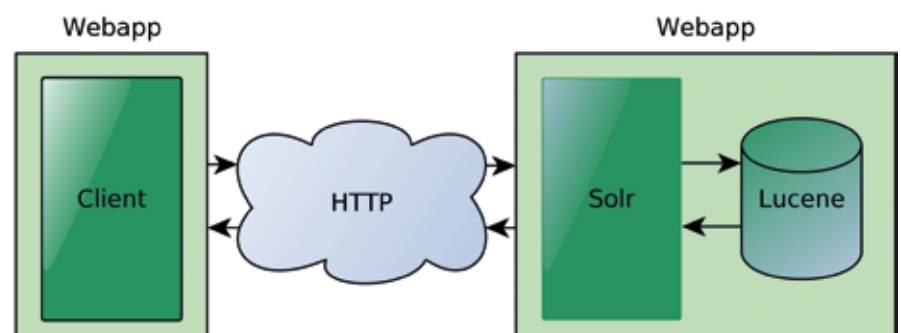


Abbildung 1: Kommunikation mit Solr

```

<fields>
  <field name="title" type="text_de" indexed="true" stored="true"/>
  <field name="category" type="string" indexed="true" stored="true" multiValued="true"
omitNorms="true"/>
  <field name="date" type="date" indexed="true" stored="true"/>
  <field name="speaker" type="string" indexed="true" stored="true" multiValued="true"/>
  <field name="content" type="text_de" indexed="true" stored="true"/>
</fields>
<types>
  <fieldType name="text_de" class="solr.TextField" positionIncrementGap="100">
    <analyzer>
      <tokenizer class="solr.StandardTokenizerFactory"/>
      <filter class="solr.LowerCaseFilterFactory"/>
      <filter class="solr.GermanNormalizationFilterFactory"/>
      <filter class="solr.GermanLightStemFilterFactory"/>
    </analyzer>
  </fieldType>
  [...]
</types>

```

Listing 1: Solr-Feldkonfiguration in schema.xml

ler“ und „SearchComponents“ konfiguriert werden. SearchComponents bieten jeweils unterschiedliche Aspekte einer Suche an, beispielsweise „QueryComponent“ für die eigentliche Kernfunktionalität der Suche oder „HighlighterComponent“ zum Hervorheben von Treffern im Text.

Die eigentliche Funktionalität nach außen stellen die „RequestHandler“ bereit, die die „SearchComponents“ verwenden und einen HTTP-Endpunkt zur Verfügung stellen. Häufig verwendete „SearchComponents“ sind bei Verwendung des für das Suchen vorgesehenen „SearchHandler“ schon registriert. Die von den „SearchComponents“ erwarteten Parameter lassen sich durch Default-Werte vordefinieren. Je nach Konfiguration und Bedarf können diese Werte aber auch durch übergebene Parameter beim Aufruf durch den Client überschrieben werden (siehe Listing 3).

Oftmals genügen die von Solr mitgelieferten Komponenten völlig aus, bei Bedarf lassen sich jedoch auch eigene Komponenten einbinden. Nicht nur an dieser Stelle ist Solr leicht erweiterbar; so gut wie alle enthaltenen Komponenten sind durch eigene Implementierungen ersetzbar, ohne Anpassungen im Kern vornehmen zu müssen.

Da der Zugriff über HTTP erfolgt, kann man mit einfachen Tools die korrekte Funk-

tionsweise testen. Der in Listing 4 angegebene Aufruf mit dem Kommandozeilen-Werkzeug „curl“ fordert beispielsweise die Ergebnisse zu dem Suchbegriff „lucene“ auf dem „RequestHandler“ namens „jug“ im JSON-Format an, wobei der „row“-Parameter, der die Anzahl der Suchergebnisse bestimmt, mit einem abweichenden Wert überschrieben wird.

Durch die Nutzung von Standard-Protokollen lassen sich Probleme auch auf Produktiv-Systemen leicht identifizieren. Für tieferegehende Analysen bietet Solr eine Administrations-Oberfläche an, in der unter anderem Statistiken über die Daten im Index abgerufen werden können und der Analyzing-Prozess mit Beispieldaten durchgeführt wird (siehe Abbildung 2).

Aus einer Java-Anwendung wird, wie in Listing 5 dargestellt, gesucht. Auch hier sind wieder deutlich weniger Informationen als in der Lucene-Implementierung notwendig. Der Client muss lediglich den Suchstring und den zu verwendenden „RequestHandler“ übergeben. Die Konfiguration des Analyzing ist komplett von der Anwendung in Solr gewandert.

Mit wenigen Zeilen Code und Konfiguration haben wir bereits die wichtigsten Grundlagen für eine Suche geschaffen. Solr kümmert sich um die Interna des Zugriffs auf Lucene. Auch wenn uns dies bereits viel Arbeit im Vergleich zu der direkten Nutzung von Lucene abnimmt, gibt es noch weitere Vorteile. So lassen sich fortschrittliche Such-Features fast rein konfi-

```

SolrServer server = new HttpSolrServer("http://localhost:8080");

SolrInputDocument document = new SolrInputDocument();
document.addField("title", "Suchen und Finden mit Lucene und Solr");
document.addField("speaker", "Florian Hopf");
document.addField("date", "2012-07-03T22:00:00Z");

server.add(document);
server.commit();

```

Listing 2: Indizieren von Daten über SolrJ

```

<requestHandler name="/jug" class="solr.SearchHandler">
  <lst name="defaults">
    <int name="rows">10</int>
    <str name="q.op">AND</str>
    <str name="q.alt">*:*</str>
    <str name="defType">edismax</str>
    <str name="qf">content title</str>
  </lst>
</requestHandler>

```

Listing 3: Konfiguration eines „RequestHandler“ in „solrconfig.xml“

gurativ hinzufügen. Einer der Faktoren für den enormen Erfolg von Solr ist beispielsweise die „Facettierung“, die in Solr früher als in Lucene verfügbar war. Dabei können Suchergebnisse dynamisch zu Kategorien zusammengefasst werden, die der Nutzer zur Eingrenzung der Suchergebnisse nehmen kann. Besonders bei großen Datenmengen ist damit das Auffinden von Inhalten enorm erleichtert, da man durch die Suchergebnisse geführt wird.

Listing 6 zeigt, wie die Facettierung über Java-Code angefordert und ausgelesen werden kann. Meist sind die Ergebnisse der Facettierung als Links neben der Suchergebnisliste angezeigt. Durch einen Klick werden die Treffer dann automatisch anhand einer übergebenen Filter-Query

eingeschränkt und damit die Anzahl nur auf Dokumente mit dieser Eigenschaft reduziert.

Obwohl schon früh eine Skalierung von Solr über Master-Slave-Setups möglich war, wurde dieser Aspekt mit Solr 4 neu angegangen. Seit diesem Release ist es möglich, echte Solr-Cluster zu betreiben, die ausfallsicher in Bezug auf das Indizieren und die Suche sind. Zur Verwaltung des Cluster-States wird das aus dem Hadoop-Framework stammende Apache ZooKeeper verwendet.

Wie wir gesehen haben, ist es mithilfe von Solr auch ohne tiefgehende Lucene-Kenntnisse möglich, fortschrittliche Such-Anwendungen zu bauen. Die mitgelieferten Komponenten bieten neben

dem vorgestellten Faceting sehr viele Möglichkeiten, um die Suche mit weiteren Funktionalitäten anzureichern. Häufig verwendet werden neben dem schon erwähnten Highlighting beispielsweise „MoreLikeThisComponent“ zur Suche nach ähnlichen Dokumenten, „Suggester“ für Autovervollständigungen von Termen aus dem Index oder externen Quellen und „SpellcheckComponent“ zur Korrektur von Rechtschreibfehlern im Suchbegriff. Ebenfalls erwähnenswert ist die Indizierung von Geo-Koordinaten über Längen- und Breitengrade, die entweder zur Sortierung (sortiere nach Ergebnis in der Nähe) oder zur Filterung (zeige nur Ergebnisse im Umkreis von 100 km) verwendet werden kann.

ElasticSearch

Das seit 2010 ebenfalls als Open Source verfügbare ElasticSearch ähnelt Solr auf den ersten Blick. Beides sind fertige Anwendungen, die Lucene benutzen und die Funktionalität über eine HTTP-Schicht abstrahieren. Beide bieten zudem die Möglichkeit, auch fortgeschrittene Suchfunktionalität direkt zu nutzen.

Im Detail macht ElasticSearch jedoch einiges anders. Im Gegensatz zu Solr ist es keine auf JEE basierende Web-, sondern eine Stand-alone-Anwendung. Die Netzwerk-Funktionalität ist über das asynchrone Netzwerk-Framework „Netty“ im-

Term	Count
und	717
die	113
der	82
ein	32
von	21
mit	6
in	
vortrag	
zu	
auf	

Abbildung 2: Solr-Administrations-Oberfläche


```
curl "http://localhost:8080/solr/jug?q=.lucene&rows=1&wt=json
&indent=on"
{
  "responseHeader": {
    "status":0,
    "QTime":0},
  "response":{"numFound":3,"start":0,"docs":[
    {
      "title":"Suchen und Finden mit Lucene und Solr",
      "date":"2012-07-03T22:00:00Z",
      "speaker":["Florian Hopf"],
      [...]
    }
  ]}
}
```

Listing 4: Ausführen einer Suche über „curl“ in Solr

```
SolrQuery solrQuery = new SolrQuery("suche");
solrQuery.setQueryType("/jug");

QueryResponse response = server.query(solrQuery);
assertEquals(1, response.getResults().size());

SolrDocument result = response.getResults().get(0);
assertEquals("Suchen und Finden mit Lucene und Solr", re-
sult.get("title"));
assertEquals("Florian Hopf", result.
getFirstValue("speaker"));
```

Listing 5: Ausführen einer Suche mit SolrJ

```
SolrQuery query = ...;
solrQuery.setFacet(true);
solrQuery.addFacetField("speaker");

QueryResponse response = server.query(solrQuery);

List<FacetField.Count> speakerFacet = response.
getFacetField("speaker").getValues();
assertEquals(1, speakerFacet.get(0).getCount());
assertEquals("Florian Hopf", speakerFacet.get(0).getName());
```

Listing 6: Facettierung über SolrJ

```
curl -XPOST ,http://localhost:9200/jug/talk/' -d ,{
  "speaker" : "Florian Hopf",
  "date" : "2012-07-04T19:15:00",
  "title" : "Suchen und Finden mit Lucene und Solr"
}'
{"ok":true,"_index":"jug","_type":"talk",
"_id":"CeltdivQRGSvLY_dBZv1jw","_version":1}
```

Listing 7: Indizierung von Daten in Elasticsearch

plementiert. Zur Kommunikation kommt ein echtes REST-API zum Einsatz, als Austausch- und internes Format dient JSON. In Bezug auf den Datenaustausch gibt es also nicht die Vielfalt an Möglichkeiten wie bei Solr. Elasticsearch benötigt allerdings keine Konfiguration vorab, nach dem Download ist es sofort einsatzbereit. Der Einstieg fällt dadurch sehr leicht. Beispielsweise lassen sich mit dem in Listing 7 angegebenen Curl-Kommando Daten ohne jegliche Konfiguration direkt nach dem Download indizieren.

ElasticSearch teilt die indizierten Daten grundsätzlich in Indizes und Typen auf. In unserem Beispiel werden der Index „jug“ und der Typ „talk“ verwendet. Existiert der Index noch nicht, wird dieser automatisch angelegt. Die zu indizierenden Daten werden über eine JSON-Struktur im Request-Body übergeben.

Der Typ bestimmt das Schema der Dokumente. Erfolgt vorab keine Konfiguration, wird das Schema aus dem ersten indizierten Dokument abgeleitet. Für unseren Fall werden für den Talk-Titel der Default-Analyzer verwendet und das Datum anhand des Formats erkannt und speziell abgelegt. Das Schema eines Typs ist über den „_mapping“-Endpunkt einsehbar, der an die URL angehängt werden kann.

Da das Analyzing oft eine anwendungsspezifische Konfiguration benötigt, kann man sich mit diesem automatisierten Prozess oft nicht zufriedengeben und will stattdessen selbst Anpassungen vornehmen. Eine nachträgliche Änderung eines einmal angelegten Schemas ist nur eingeschränkt möglich. Neue Felder können problemlos hinzugefügt, das Verhalten vorhandener Felder kann allerdings nicht mehr verändert werden. Deshalb gilt auch für Elasticsearch, dass man sich vorab Gedanken über die Ablage seiner Daten machen sollte.

In Listing 8 wird unser zuvor erstellter Index erst gelöscht, bevor er mit einem zu unseren Daten passenden Mapping neu erstellt wird. Dadurch wird das Feld „title“ mit dem von uns benötigten Analyzer für deutsche Texte verarbeitet.

Um auf den Daten zu suchen, ist das in Listing 9 angegebene „curl“-Kommando möglich. Der „_search-Handler“ akzeptiert den Query-Parameter „q“, dem ein Wert in der Lucene-Query-Syntax über-

geben werden kann. Ist kein Feldname angegeben, wird auf einem speziellen Feld „_all“ gesucht, das per Default alle Feldinhalte des Dokuments enthält. Für die spätere Ausgabe speichert Elasticsearch standardmäßig die kompletten indizierten Daten im Feld „_source“. Deshalb muss man sich nicht sofort überlegen, welche Felder man als gespeichert ablegen will; der Originalinhalt ist immer verfügbar. Bei großen Datenmengen sollte jedoch darüber nachgedacht werden, dieses Feature zu deaktivieren, um Speicherplatz zu sparen.

Wie schon bei Solr gesehen, wollen wir eventuell noch weitere Features wie die Facettierung aktivieren oder auch komplexere Queries formulieren. In Elasticsearch arbeitet man deshalb nur selten mit Query-Parametern, meist wird stattdessen die sogenannte „Query-DSL“ verwendet. Dabei handelt es sich um JSON-Strukturen, die einzelne Aspekte einer Abfrage abbilden. Listing 10 zeigt beispielsweise eine Abfrage, bei der zusätzlich die Facettierung auf ein Feld angefordert wird.

Die Verwendung einer ausdrucksstarken Sprache mag an dieser Stelle wie eine unnötige Aufblähung des Request wirken. In der Praxis ist es allerdings so, dass dies zu deutlich leichter verständlichen Beispielen führt und die Wartbarkeit einer Anwendung verbessern kann.

Wenn der Client in Java implementiert ist, muss man sich nicht zwingend selbst mit JSON auseinandersetzen. Der integrierte Java-Client stellt über Builder-Objekte eine der Query-DSL ähnliche Sicht auf

```
curl -XDELETE http://localhost:9200/jug/
curl -XPOST localhost:9200/jug -d '{
  "mappings" : {
    "talk" : {
      "properties" : {
        "title" : {
          "type" : "string",
          "analyzer" : "german"
        }
      }
    }
  }
}'
```

Listing 8: Löschen und Erstellen eines Elasticsearch-Index mit Mapping-Informationen

```
curl -XGET ,http://localhost:9200/jug/talk/_
search?q=title:suche'
{...},
"hits":{
  "total":1,"max_score":0.054244425,
  "hits":[
    ...
    "_score":0.054244425,
    "_source" : {
      "speaker" : "Florian Hopf",
      "date" : "2012-07-04T19:15:00",
      "title": "Suchen und Finden mit Lucene und Solr"
    }
  ]
}
```

Listing 9: Suchen in Elasticsearch per „curl“

die Abfrage dar. Listing 11 zeigt das vorher in „curl“ formulierte Beispiel in Java, wobei

einige Methoden statisch importiert wurden. Interessanterweise spricht der Java-

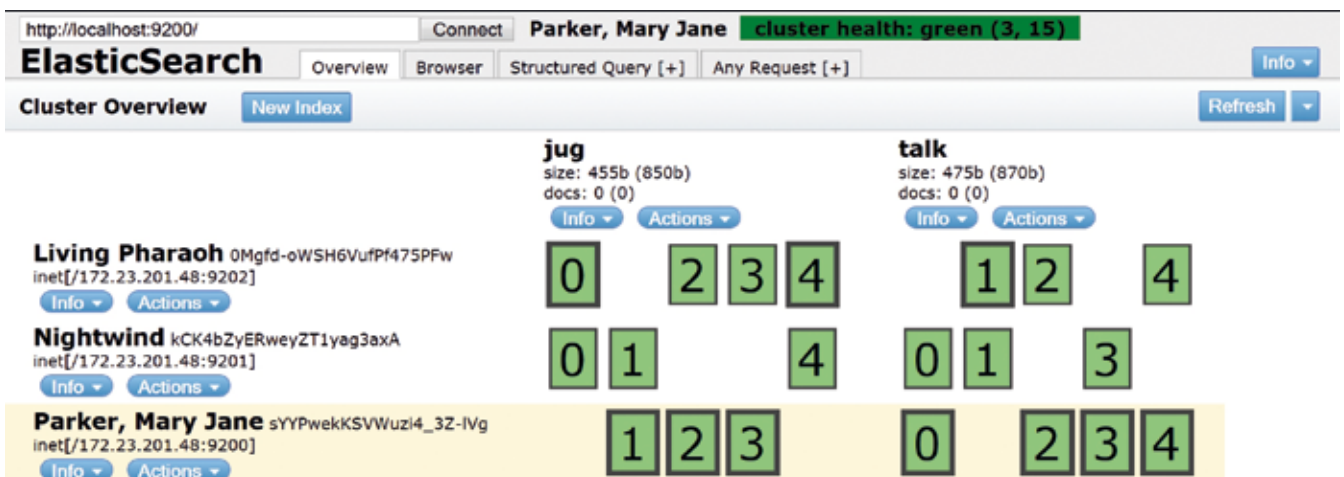


Abbildung 3: Verteilung eines Clusters, visualisiert in „elasticsearch-head“

```
curl -XGET 'http://localhost:9200/jug/talk/_search' -d '{
  "query" : {
    "query_string" : {
      "query" : "suche"
    }
  },
  "facets" : {
    "tags" : {
      "terms" : {
        "field" : "speaker"
      }
    }
  }
}'
```

Listing 10: Suchen in Elasticsearch per Query-DSL

Client in der Default-Konfiguration auch nicht per REST-API mit dem Elasticsearch-Server, sondern hängt sich als Knoten in den Cluster ein.

Generell ist das Thema „Clustering und Verteilung in Elasticsearch“ für den Benutzer sehr einfach zu verwalten. Knoten im selben Netzwerk verbinden sich anhand eines konfigurierten Namens mit allen Knoten desselben Cluster-Namens. Für die Verteilung relevant sind „Shards“ und „Replicas“, die für jeden logischen Index konfiguriert werden können. „Shard“ ist ein Lucene-Index, in den ein Teil der Indexdaten abgelegt wird. „Replica“ ist die Kopie eines Shard, die auf einem weiteren Knoten abgelegt wird. Wie wichtig der Verteilungsgedanke bei Elasticsearch ist, sieht man auch an den vorkonfigurierten Werten: Wenn nicht anders angegeben, wird jeder logische Index auf fünf Shards verteilt. Somit ist selbst die Nutzung von nur einem Knoten mit einem logischen

Index schon eine verteilte Suche auf Lucene-Ebene.

Shards und Replicas werden je nach vorhandenen Knoten automatisch von Elasticsearch verteilt. Tritt ein neuer Knoten dem Cluster bei, werden die Shards umorganisiert, damit eine möglichst gute Lastverteilung erreicht wird. Der Benutzer kann jedoch alle diese Mechanismen selbst beeinflussen; beispielsweise kann konfiguriert werden, dass bestimmte Daten nur auf festgelegten Knoten abgelegt sind. **Abbildung 3** zeigt einen Screenshot des Elasticsearch-Plug-ins „elasticsearch-head“, mit dem unter anderem der aktuelle Zustand eines Clusters visualisiert werden kann. Unsere fünf Shards sind mit jeweils einem Replica auf drei Knoten verteilt.

Die Qual der Wahl

Beide Produkte eignen sich gut für die meisten Anwendungsfälle und stellen eine zukunftssichere Wahl dar. Änderun-

gen in Lucene fließen in beide Projekte schnell ein; in Solr, da es als Unterprojekt von Lucene von denselben Committern bearbeitet wird, und in Elasticsearch, weil die dahinterstehende Firma mehrere Lucene-Committer beschäftigt. Die Projekte werden sich in nächster Zeit weiter annähern, was den Umfang der Features betrifft. Solr hat bereits Ideen von Elasticsearch übernommen und Elasticsearch hat bezüglich der Such-Features aufgeholt. Noch ist es so, dass zu Solr mehr Wissen in Form von Büchern oder Erfahrungen in Unternehmen verfügbar ist. Der schnelle Einstieg in Elasticsearch kann diesen Vorteil allerdings wieder relativieren.

Links

- <http://lucene.apache.org/solr>
- <http://elasticsearch.org>
- <https://github.com/fhopf/lucene-solr-talk>

Florian Hopf
mail@florian-hopf.de



Florian Hopf arbeitet als freiberuflicher Software-Entwickler mit den Schwerpunkten „Content Management“ und „Suchlösungen“ in Karlsruhe. Er setzt Lucene und Solr seit Jahren in unterschiedlichen Projekten ein und ist einer der Organisatoren der Java User Group Karlsruhe.

```
Client esClient = nodeBuilder().client(true).node().client();
QueryBuilder query = queryString("suche");
TermsFacetBuilder facet = termsFacet("speaker").field("speaker");
SearchResponse response = esClient.prepareSearch("jug")
    .addFacet(facet)
    .setQuery(query)
    .execute().actionGet();
assertEquals(1, response.getHits().getTotalHits());
```

Listing 11: Suchen mit dem Elasticsearch-Java-Client

Portabilität von Java-Implementierungen in der Praxis

Thomas Niedergesäß und Burkhard Seck, Tricept Informationssysteme AG

„Write once, run anywhere!“, dieser legendäre Slogan begleitet die Programmiersprache Java, seitdem sie das Licht der IT-Welt erblickt hat. Ihre Portabilität war und ist zweifellos einer der Gründe für den enormen Erfolg. Nicht umsonst wird Java zurzeit auf Milliarden von Geräten eingesetzt, und das in den unterschiedlichsten Bereichen. Aber ist das mit der Portabilität wirklich immer so einfach? Wann ist eine Anwendung oder Implementierung portierbar? Reicht es wirklich, auf Java zu setzen, und Betriebs-Plattformen und Einsatzgebiete sind anschließend egal?

Seit mehr als zehn Jahren sind die Autoren in die Entwicklung und den produktiven Betrieb eines Kontoführungs- und Buchungssystems in einem heterogenen System-Umfeld involviert und setzen dabei voll auf Java-Technologien. Ausgehend von den konkreten Erfahrungen, die sie in diesem Projekt sammeln konnten, wird sich dieser Artikel etwas grundsätzlicher mit dem Thema „Portabilität“ befassen.

Die Autoren beabsichtigen nicht, eine universelle Lösung für alle Portabilitätsprobleme vorzustellen. Sie können auch nicht im Rahmen dieses Artikels das Thema in all seinen Facetten behandeln. Vielmehr möchten sie zum Nachdenken anregen und das Bewusstsein für die Problematik schärfen. Dazu zeigen sie anhand einiger Beispiele aus der Praxis auf, wo Probleme und Fallstricke lauern, wenn man als Entwickler seinen Quellcode portabel halten möchte, und betrachten einige Aspekte, die die Portabilität einer Java-Anwendung beeinflussen.

Definition des Begriffs „Portabilität“

Schlägt man bei Wikipedia nach, ist „Portabilität“ oder „Portierbarkeit“ definiert als der Grad der Plattform-Unabhängigkeit eines Computer-Programms. Portabilität ist jedoch kein Maß für die Lauffähigkeit eines Programms auf der Zielformatplattform. Ein Portabilitätsgrad von mehr 99 Prozent bedeutet nicht, dass das portierte Programm tatsächlich nutzbar ist und wie gewünscht seine Arbeit verrichtet. Es bedeutet nur, dass eine Portierung im Vergleich zu einer Neuentwicklung deutlich weniger Aufwand erfordert.

Diese theoretische Betrachtungsweise vernachlässigt aber, dass in der Praxis und somit im eigentlichen Kundengeschäft die Lauffähigkeit eines Programms von essenzieller Bedeutung ist, wenn man von einer Plattform auf eine andere wechseln will. Aus diesem Grund fassen wir für uns den Portabilitätsbegriff etwas weiter.

Wir verstehen unter Portabilität neben der oben beschriebenen, eher technischen Definition von Portabilität genauso auch ein Maß für die fachliche Nutzbarkeit des Sourcecodes auf unterschiedlichen Plattformen. In realen Kundenprojekten stehen vor allen Dingen konkrete Geschäftsanforderungen und deren korrekte Umsetzung im Vordergrund. Wir differenzieren zwischen technischer und fachlicher Portabilität. Es geht einerseits darum, wie einfach es ist, technisch eine Java-Klasse oder auch -Anwendung auf unterschiedlichen Plattformen zu verwenden, und andererseits, wie einfach es ist, dieses auch vor einem fachlichen Hintergrund zu tun.

Portabilität aus praktischer Sicht

Java hatte schon immer den Anspruch, portabel zu sein, und hat dies in seiner Entwicklungsgeschichte auch immer wieder erfolgreich beweisen können. Seit es Java gibt, ist es zumindest aus technischer Sicht vergleichsweise einfach, auf eine andere Plattform zu wechseln. Dabei heißt Plattform-Wechsel nicht unbedingt nur Betriebssystem-Wechsel, sondern auch Wechsel auf andere Java-Versionen, auf virtuelle Maschinen anderer Hersteller oder Wechsel von einem JEE-Application-Server auf einen anderen.

Entsprechend groß ist die Erwartungshaltung von Anwendern und Entwicklern, egal ob es sich dabei um die Java Standard oder die Enterprise Edition handelt. Man verlässt sich darauf, dass eine Java-Anwendung grundsätzlich portabel ist. So ist es nicht verwunderlich, dass man bei Kunden immer wieder Aussagen zu hören bekommt wie: „Java läuft doch überall“, „Testen auf der Entwicklungsplattform reicht aus, den Anwendungsserver testen wir im Integrationstest mit“, oder auch Aussagen von Vorgesetzten: „Wenn wir Java einsetzen, können wir bei der technischen Ausbildung unserer Entwickler sparen.“

Wenn das alles so einfach ist, warum machen wir uns dann hier überhaupt Gedanken zum Thema „Portabilität“? Offenbar gibt es dort doch kaum Probleme.

Um in die Problematik weiter einzutauchen, werfen wir einen Blick darauf, wie wir heutzutage in der Praxis Software entwickeln. Gerade im Java-Bereich ist man einem permanenten Wandel unterworfen. Technologische Ansätze veralten in rasender Geschwindigkeit und werden ständig durch neue Ansätze abgelöst. Die Zahl neuer Technologien, die sich dauerhaft etablieren, ist dabei im Vergleich zur absoluten Anzahl neuer Ideen eher gering. Bei jeder Weiterentwicklung steht man gerade in kleineren Projekten im Prinzip immer vor der Frage, ob man nicht gleich die alten Ansätze über Bord werfen und auf aktuellere Technologien setzen soll.

Ermöglicht werden diese raschen Technologie-Sprünge durch den Einsatz immer leistungsfähigerer Frameworks, Klassen-Bibliotheken und Tools zur Generierung von

Java-Sourcecode. Viele Standardaufgaben kann ein Entwickler heute nach kurzer Einarbeitung und ohne größeres Verständnis der technischen Hintergründe übernehmen. Oft beschränkt sich seine Tätigkeit dabei auf die Konfiguration eines gerade etablierten Tools. Man betrachte nur den Aufwand, den man vor zehn Jahren treiben musste, um aus Java ein einfaches SQL-Statement Richtung Datenbank abzusetzen. Wie leicht ist es dagegen heute, mit einer Datenbank zu kommunizieren? Der Einsatz dieser Werkzeuge führt jedoch häufig dazu, dass bei den Entwicklern das Wissen um die technischen Hintergründe verblasst. Durch die Einführung weiterer Abstraktionsebenen geht das Bewusstsein um die Zusammenhänge im System immer mehr verloren. Wozu soll man sich auch Wissen aneignen, das man nicht zur Erfüllung seiner Aufgaben braucht?

Dazu kommt, dass die Lebenszyklen so mancher Software oder Technologie inzwischen so kurz sind, dass es sich auf den ersten Blick kaum lohnt, Aufwand in so etwas wie die Portabilität der Quellen zu stecken.

Betrachtet man Software-Entwicklung auf kurze Sicht, so ist es in erster Linie wichtig, dass der Java-Sourcecode fachlich und technisch korrekt abläuft und Antwortzeiten, Speicherverbrauch und Performance akzeptabel sind. Portabilität heißt dabei in der Regel, dass der Quellcode auf allen Plattformen lauffähig ist, auf denen er zum aktuellen Zeitpunkt und in naher Zukunft zum Einsatz kommt. Natürlich gibt es auch Anwendungen mit relativ kurzer Lebensdauer und geringem Funktionsumfang, bei denen Portabilität einen größeren Stellenwert einnimmt. Man denke nur an Mobile Apps, die auf einer Vielzahl unterschiedlichster Endgeräte laufen sollen. Dafür portablen Code zu schreiben, gleicht oft einer Sisyphus-Arbeit.

Bewegt man sich in einer Umgebung, in der die Entwicklungs- und Lebenszyklen deutlich länger sind, so spielt die Portabilität von vornherein eine größere Rolle. Ein Beispiel dafür sind heterogene Systemumgebungen im Banken- und Versicherungsbereich, in denen oft neue Ansätze mit seit Jahrzehnten bewährten Technologien ein hocheffizient funktionierendes System bilden müssen. Dort kann man aufgrund der Vielzahl der fachlichen und technischen Anforderungen nicht einfach einmal alles neu entwickeln. Es reicht nicht aus, dass



Abbildung 1: Beispiel Massenzahlungsverkehr – ein typischer Batch-Ablauf stellt ganz andere Anforderungen an den Sourcecode

der Java-Quellcode funktioniert, er muss auch in fünf oder vielleicht zehn Jahren noch funktionieren (siehe Abbildung 1). Das bedeutet ganz konkret, der Entwickler muss sich beim Design seiner Anwendung schon Gedanken darüber machen, wie er dies trotz der rasanten Entwicklung im Java-Bereich sicherstellen kann.

Die Etablierung von Java als Programmiersprache am Mainframe war und ist für viele Java-Entwickler in diesem Zusammenhang eine echte Herausforderung. Muss man sich doch plötzlich mit einer Plattform beschäftigen, auf der es durchaus vorkommen kann, dass eine Anwendung zwanzig Jahre lang ohne wesentliche Änderungen läuft – der totale Gegensatz zur schnelllebigen Java-Welt.

Große Systemumgebungen bestehen in der Regel aus vielen unterschiedlichen Einzelsystemen. Wer möchte da nicht die Möglichkeiten von Java nutzen, portablen Sourcecode produzieren und so Synergien bei der Entwicklung ausschöpfen? Dass unter solchen Voraussetzungen die Portabilität der Quellen einen ganz anderen Stellenwert besitzt als bei der Entwicklung von quasi „Wegwerf-Anwendungen“, leuchtet ein. Das heißt aber nicht, dass dort generell portabler Code entsteht. Wir dürfen nicht vergessen, dass viele Entwickler ihre Karriere meist in kleinen Umgebungen beginnen und das Bewusstsein für Portabilität einfach nicht so ausgeprägt ist.

Wer immer in und für Windows-Umgebungen entwickelt, der blendet Anforderungen, die eine Linux- oder Großrechner-Umgebung an den Sourcecode stellt, irgendwann aus, selbst wenn er zuvor darauf aufmerksam gemacht wurde, dass es da Unterschiede gibt. Generell haben die Autoren die Beobachtung gemacht, dass Sourcecode in der Regel immer nur so portabel ist, wie er sein muss. Ist man gezwungen, für fünf unterschiedliche Umgebungen zu entwickeln, so läuft der Sourcecode auf diesen fünf Umgebungen. Mit einiger Wahrscheinlichkeit tut er dies auch in anderen Umgebungen, obwohl man sich da nicht wirklich sicher sein kann. Trotzdem verlässt man sich darauf, wenn selbstverständliche technische Voraussetzungen wie Kompatibilität der JVM-Version oder die Verfügbarkeit verwendeter Bibliotheken auf der anderen Plattform erfüllt sind. In vielen Fällen funktioniert das sogar. Da liegt wahrscheinlich auch eine der Fallen, in die man bei der täglichen Arbeit mit Java tappen kann: Die relativ zuverlässige Portabilität eines Java-Programms im technischen Bereich führt dazu, dass man andere Portabilitätsaspekte ebenfalls geistig ausblendet.

Das Projekt-Umfeld

Die Bank-Plattform, für die die Autoren entwickeln, umfasst eine Beraterplatz-Anwendung in klassischer Dreischichten-Archi-

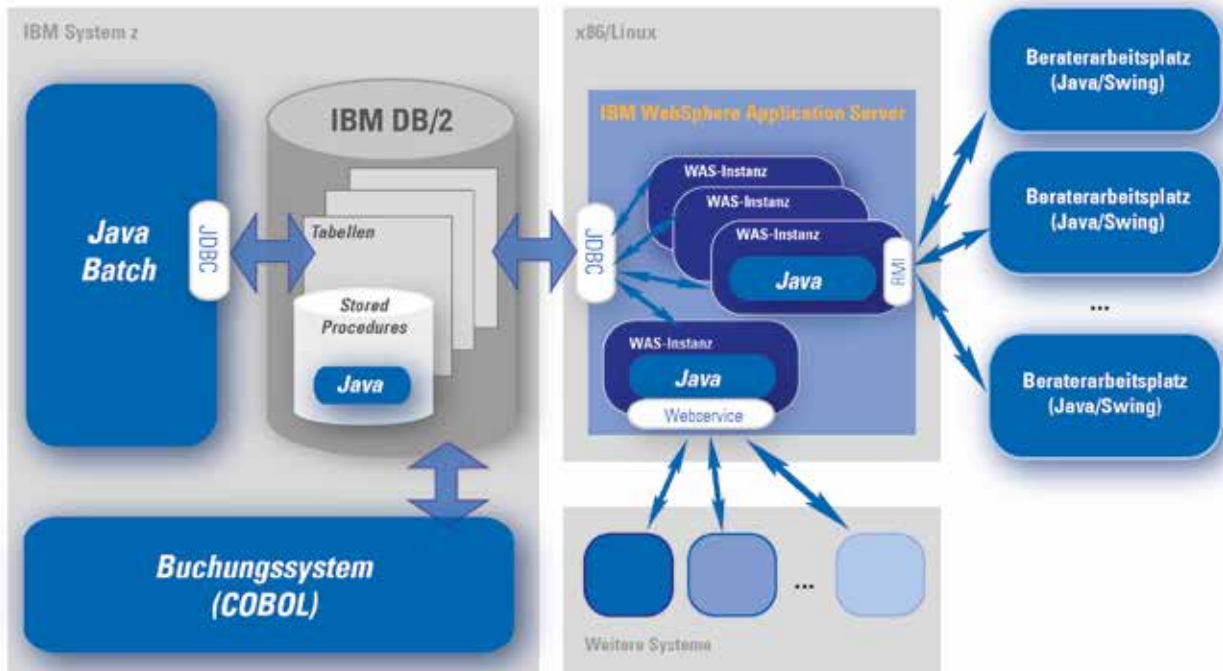


Abbildung 2: Das Projekt-Umfeld

tektur mit einer Online-Server-Anwendung auf einem JEE-Application-Server-Cluster (Linux), die über RMI Java-Swing-Clients (Windows) bedient (siehe Abbildung 2). Der Server kommuniziert mit einer DB/2-Datenbank am Mainframe und einem dort ebenfalls angesiedelten COBOL-Buchungssystem.

Weitere Systeme greifen über Web-Services auf den Online-Server zu und nutzen die dort implementierte Fachlichkeit. Der Swing-Client ist als Rich-Client mit minimalem fachlichen Codeanteil implementiert. Alle Konsumenten des JEE-Servers arbeiten statusbehaftet. Am Mainframe werden unter z/OS in COBOL und in Java implementierte Batchprozesse ausgeführt.

Hinzu kommen noch DB/2 Stored Procedures, die ebenfalls in Java implementiert sind, und ein unter z/OS laufender Java-Socket-Server für COBOL-nach-Java-Aufrufe. Das System ist seit 2005 produktiv im Einsatz und verwendet, Stand heute, Java 7. Begonnen wurde das Projekt unter Visual Age for Java mit der Java-Version 1.2.1., hat also schon einige Portierungen und Migrationen hinter sich.

Für alle Systeme wird die gleiche Quellcode-Basis verwendet, um die Java-Implementierung zusammenzuhalten. Entwickelt wird unter Eclipse mit einer Stan-

dard-VM (Windows). Das gilt auch für die Java-Batchprozesse. Sämtliche Entwicklertests erfolgen auf der Entwicklungs-Plattform. Nur bei besonders Performance-kritischen Implementierungen für den Batch werden auch Tests auf dem Mainframe als Ziel-Plattform durchgeführt.

Tests gegen den Application Server spielen in der Entwicklung keine Rolle. Unser System lässt sich in einen speziellen Instanzenmodus schalten, um bei der Entwicklung auf den Application-Server verzichten zu können. Client- und Server-Teil laufen dann zusammen in einer virtuellen Maschine, wobei durch selbst implementierte Mechanismen sichergestellt wird, dass es später zu keinen Problemen bei der Serialisierbarkeit kommt.

Oberstes Ziel ist es, für alle Java-Komponenten denselben fachlichen Quellcode zu verwenden. Zeichnen sich während der Entwicklung bereits Performance-Probleme ab, entscheidet man sich bewusst gegen eine Wiederverwendung bestehenden Sourcecodes. Dann kommen besondere Implementierungen zum Einsatz – zum Glück ist dies nur sehr selten notwendig. Während der gesamten Entwicklung erfolgen automatisierte Regressionstests, auch für die GUI.

Anhand einer Auswahl von Beispielfällen aus der Praxis wird nun verdeutlicht,

wo das Thema „Portabilität“ Einfluss auf die Entwicklung von Java-Sourcen haben kann. Bei einigen dieser Beispiele denkt man sicherlich: „Das ist doch selbstverständlich!“ Die Autoren haben jedoch die Erfahrung gemacht, dass gerade diese Selbstverständlichkeiten dann im Projektalltag doch wenig beachtet wurden. Die Beispiele kommen sowohl aus dem technischen als auch aus dem eher fachlichen Bereich.

Aufwärts- und Abwärts-Kompatibilität zwischen einzelnen Java-Versionen

Die Anforderungen für eine Aufwärts- beziehungsweise Abwärts-Kompatibilität sind einfach. Der Bytecode muss kompatibel sein und wenn man das nicht gewährleisten kann, muss der Sourcecode kompatibel sein. Befolgt man in heterogenen Systemen, in denen mehrere Java-Versionen parallel in Betrieb sind, diese einfachen Regeln nicht, kommt man schnell an den Punkt, an dem eine Implementierung nicht auf allen Systemen eingesetzt werden kann. Bewusst geworden ist das vielen zum ersten Mal in größerem Umfang bei der Einführung von Java 5, aufgrund der vielen damals neu eingeführten Sprachkonstrukte. „Generics“ und „foreach“-Schleifen wurden eingeführt, „enum“ wurde Schlüsselwort und war

```

public class TestMillis {

    public static void main(String[] args) {

        // Ermittlung der Auflösungsgenauigkeit
        // von java.lang.System.currentTimeMillis

        int anzahl = 10000000;
        long[] werte = new long[anzahl];

        for (int i = 0; i < anzahl; i++) {
            werte[i] = System.currentTimeMillis();
        }

        long vergleich = werte[0];

        for (int i = 1; i < anzahl; i++) {
            if (vergleich != werte[i]) {
                System.out.println(werte[i] + " " +
(werte[i] - vergleich) + " " + i);

                vergleich = werte[i];
            }
        }
    }
}

```

Listing 1: Ein einfacher Algorithmus zur Bestimmung der Auflösungsgenauigkeit von „System.currentTimeMillis()“

plötzlich als Variablen-Name ungeeignet, „Autoboxing“ bot ganz neue Möglichkeiten und neue Pakete wie „java.util.concurrent“ wurden verfügbar.

Auch externe Bibliotheken sind davon betroffen. Funktionieren sie auch auf der neuen Plattform? Man muss nämlich nicht nur den eigenen Code auf Kompatibilität prüfen, sondern das Gesamtsystem. Was ist mit Workarounds, die man irgendwann einmal geschrieben hat, um Bugs in Basis-klassen zu beheben? Sind diese beim Wechsel auf eine aktuellere Java-Version, in der die Bugs behoben sind, wieder ausgebaut worden? Funktionieren Workarounds überhaupt noch?

Einsatz externer Bibliotheken

Externe Bibliotheken können nicht nur in Bezug auf die Aufwärts- oder Abwärts-kompatibilität die Portabilität beeinflussen. Man sollte sich immer fragen, ob diese auch auf allen eingesetzten Plattformen zur Verfügung stehen und ob die Funktio-

nalität auf allen Plattformen wirklich gleich ist. Auf Mainframe-Systemen können die Security-Einstellungen den Einsatz einer externen Bibliothek unmöglich machen, weil diese Bibliothek beispielsweise JNI-Aktionen ausführen möchte, für die der Benutzer nicht berechtigt ist.

Will man die Portabilität längerfristig sicherstellen, hat es sich als hilfreich erwiesen, diese Bibliotheken auch im Quellcode vorzuhalten, sollte dieser verfügbar sein. So hat man im Zweifelsfall immer noch die Möglichkeit, die Bibliothek selbst zu migrieren und zu kompilieren. Gerade Open-Source-Bibliotheken und -Projekte im Java-Bereich sind gern nach wenigen Jahren wieder von der Bildfläche verschwunden. Was macht man dann mit seiner eigenen Implementierung?

Dabei ist aber zu bedenken, dass Bibliotheken oft auch wieder von anderen Bibliotheken abhängen. Irgendwo ist eine Grenze, an der man sich entscheiden muss, ob eine Migration noch sinnvoll ist

oder der Umstieg auf eine andere Bibliothek nicht besser wäre. Verwenden die eingesetzten Bibliotheken irgendwo JNI, kann eine Migration zudem auch schnell zu einer hochkomplexen Angelegenheit werden.

Verwendung von Start-Parametern und Zugriff auf System-Eigenschaften einer JVM

Optimierungen durch Eingriff in die Start-Parameter der virtuellen Maschine können zu Inkompatibilitäten führen. Das sind Parameter, die durch die „-D“-Option gesteuert werden, zum Beispiel „-Djdk.map.althashing.treshold“. Hier sollte man sich immer fragen, warum man so etwas verwendet, wie man es dokumentiert, was man beachten muss, wenn man auf eine JVM eines anderen Herstellers wechselt, und wie man Testfälle für diese Parameter definieren kann. Genauso problematisch können das Auslesen und die Verwendung von System-Properties sein. Wie portabel ist beispielsweise „System.getProperty(“com.ibm.vm.bitmode”);“?

Hardware-Abhängigkeiten trotz JVM

Die Grund-Idee jeder JVM ist es, Hardware-Unterschiede transparent zu machen und so eine größtmögliche Portabilität der Java-Sourcen zu ermöglichen. Es gibt aber einzelne Stellen, an denen trotzdem Hardware-Unterschiede zutage treten können. Ein Beispiel dafür ist die Methode „System.currentTimeMillis()“, die den Unterschied zwischen der aktuellen Uhrzeit und dem Zeitpunkt „Mitternacht am 1. Januar 1970“ (UTC) in Millisekunden zurückliefert (siehe Listing 1). Verwendet wird dieser Long-Wert dann beispielsweise für Zeitmessungen. Wie genau ist denn dieser Wert nun, beziehungsweise wie kompatibel und wovon ist er abhängig?

Für unterschiedliche PC-Architekturen haben die Autoren testweise ermittelt, dass die Auflösungsgenauigkeit des Funktions-ergebnisses ungefähr zwischen zehn und fünfzehn Millisekunden liegt. Grund dafür ist die Auflösungsgenauigkeit des eingesetzten Hardware-Timers der physischen Maschine. In diesem Intervall werden also immer gleiche Ergebnisse geliefert.

Ein Klassiker in Mainframe-Umgebungen ist es, Time Stamps für Primärschlüssel in Datenbank-Tabellen zu verwenden. Dass diese also nicht unbedingt eindeutig sind, wenn man sie aus Java heraus erzeugt,

fällt in der Entwicklung und in einfachen Tests oft gar nicht auf. Erst wenn man auf eine deutlich leistungsfähigere Maschine wechselt oder viele Time Stamps gleichzeitig anfordert, bekommt man Probleme mit doppelt erzeugten Primärschlüsseln.

Zu ähnlichen Problemen kann auch die Funktion „System.nanoTime()“ führen. Diese Funktion liefert vom genauesten System-Zeitgeber die Zeit. Sie hat keinen Bezugspunkt zu einem Datum. Hier zeigt sich allerdings, dass sich das ermittelte Ergebnis sogar in der Anzahl der Stellen unterscheiden kann. So werden unter Windows Server 2008 R2 zum Beispiel fünfzehn Stellen ermittelt und unter Windows 7 nur vierzehn. Wenn man diesen Wert nun in einer Datenbankspalte mit vierzehn fest definierten Stellen verwendet, wird das in der Entwicklung unter Windows 7 problemlos funktionieren, auf dem Produktivsystem unter einem anderen Betriebssystem wahrscheinlich nicht mehr.

Unterschiede zwischen Server- und Stand-alone-Anwendungen

Server-Anwendungen waren schon immer eine Klasse für sich. Generell musste sich ein Java-Entwickler bei deren Programmierung deutlich mehr Gedanken um Nebenläufigkeit, Synchronisation und Thread-Sicherheit machen oder um Hauptspeicher-Verbrauch, Caching und Initialisierungsmechanismen als beim Entwickeln einer Stand-alone-Anwendung, bei der man sich die Ressourcen nicht mit anderen teilen muss. So kommt es durchaus vor, dass eine Java-Klasse, die für eine klassische Stand-alone-Anwendung entwickelt wurde, auf einem Server nicht einsetzbar ist, weil sie beispielsweise durch Caching den Hauptspeicher so belastet, dass der Server nicht mehr reagiert.

Zusätzlich muss man sich fragen, wie portabel die eigene Server-Anwendung eigentlich ist? Verwendet man nur den JEE-Standard oder benutzt man herstellerspezifische Mechanismen, Klassen oder Annotations? Das Laufzeitverhalten unterschiedlicher Server bezüglich des Umgangs mit Data Sources, Timeouts oder Locking auf Datenbank-Tabellen kann sich eklatant unterscheiden. Kann ich meine Anwendung dann noch einfach portieren? Verhält sich unter Umständen der Test-Container in der Entwicklungsumgebung ganz anders als mein Produktiv-Server?

Welche Aussagekraft haben dann noch meine Entwicklertests?

Laufzeitverhalten auf unterschiedlichen Plattformen

Die Systemumgebung kann das Laufzeitverhalten stark beeinflussen und so für

Einschränkungen in der Portabilität einer Anwendung sorgen. Wenn man nach einem Umgebungswechsel mit der Datenbank plötzlich „remote“ statt „lokal“ kommuniziert oder auf einmal nur noch ein Zehntel der Netzwerk-Bandbreite zur Verfügung steht, darf man sich nicht wundern, wenn sich die

```
public class Kontonummer {

    private Iban iban;
    private Bankleitzahl blz;
    private String laenderKennzeichen;
    private String ktoArt;
    private Datum gueltigAb;
    private Datum gueltigBis;
    private Datum loeschDatum;
    private Long vereinbarungsNr;
    private Long stammVereinbarungsNr;
    private Integer ktoProduktNr;
    private Long ktoVereinbarungsNr;
    private String intKontierung;
    private String verwendungsZweck;
    private Iban nachfolgeIban;
    private Iban nachfolgeIbanArt;
    private Datum zurordnungsDatum;
    private Long stapelNrRM;
    private Character sperreMrk;
    private String status;
    private String ktoTyp;
    private Integer zustand;
    private Short optimSperre;

    // ...
}
```

Listing 2: Kontonummern-Klasse vollständig

```
public class KontonummerKompakt {

    private Iban iban;
    private Bankleitzahl blz;
    private String laenderKennzeichen;
    private Long vereinbarungsNr;
    private String ktoArt;
    private Datum loeschDatum;
    private Integer zustand;
    private Short optimSperre;

    // ...
}
```

Listing 3: Kontonummern-Klasse kompakt

Laufzeiten verschlechtern. Überall dort, wo man mit anderen Prozessen kommuniziert, auf Dateien, Datenbanken oder Netzwerke zugreift und sich nicht darauf verlassen kann, dass diese Ressourcen optimal angebunden sind, muss man erhöhten Aufwand in Entwicklung und Test stecken, um einen Wechsel der Systemumgebung ohne Performance-Überraschungen zu ermöglichen.

Man sollte sich immer vor Augen halten, dass sich das Entwicklungssystem in der Regel ganz anders verhält als das Produktivsystem. Gerade wenn man am PC für den Mainframe entwickelt, kann das Laufzeitverhalten beider Systeme enorm voneinander abweichen. Bewährt hat sich in solchen Fällen, immer vom schlechtesten Fall auszugehen. Eine Anwendung, die mit einer „remote“ angebundenen Datenbank performant läuft, wird garantiert nicht schlechter, wenn die Datenbank in einer anderen Umgebung „lokal“ zur Verfügung steht.

Verlässliche Aussagen zu Performance und Laufzeitverhalten liefert grundsätzlich immer nur die Ziel-Plattform, deshalb sollte man Performance-Untersuchungen nach Möglichkeit immer dort ausführen.

GUI und Portabilität

Im GUI-Bereich stoßen Portabilitätsbemühungen schnell an ihre Grenzen. Zwar sehen Swing und Java FX auf allen Plattformen gleich aus. Aber das Look and Feel unterschiedlichster Plattformen in einem Modell nachzubilden, ist quasi unmöglich. Dazu weichen die Bedienkonzepte zu stark voneinander ab. Die Themen „Mobile“ und „Windows 8.1 mit Gestensteuerung“ verschärfen dies noch. Portabilität ist hier aktuell nur dadurch zu erreichen, dass man viele Kompromisse eingeht.

Auch GUI-Designer/-Editoren sind ein schönes Beispiel dafür, dass es unterschiedliche Arten von Portabilität gibt. Der Aufwand, das grafische Entwurfstool für die Oberflächen und damit oft auch die eigentliche Entwicklungsumgebung zu wechseln, ist bei einer größeren Anzahl von Oberflächen enorm.

Unterschiede in den Laufzeit-Anforderungen

Nicht jede Java-Implementierung ist auf jeder Plattform sinnvoll und effektiv. Trotz ähnlicher fachlicher Anforderungen kann es passieren, dass man sich bewusst für zwei unterschiedliche Implementierungen

entscheiden muss, weil anders die Laufzeit-Anforderungen nicht erfüllt werden können. Als Beispiel sei hier eine Kontonummern-Tabelle aus einem Kontoführungs- und Buchungs-System angeführt. Diese besitzt zweiundzwanzig Spalten mit Hunderttausenden von Einträgen.

Für die Online-Anwendung auf dem Application-Server wurde die Tabelle „1:1“ auf ein Objekt mit zweiundzwanzig Attributen abgebildet. Instanzen der Kontonummern-Klasse werden eher selten verwendet, meist bei der Eröffnung eines neuen Kontos. Es spielte also keine Rolle, wie groß das Objekt im Speicher ist. Einige Zeit später sollte ein Java-Batchprozess implementiert werden, der nachts die Kontonummern reorganisiert und freigewordene Kontonummern so kennzeichnet, dass sie wiederverwendet werden können.

Die erste Implementierung verwendete dazu die Kontonummern-Klasse aus der Online-Anwendung. Ergebnis: eine inakzeptable Laufzeit durch viele Garbage Collects, hoher Hauptspeicher-Verbrauch, teilweise Abbrüche durch Speicher-Überlauf. Die Größe des Objekts spielte plötzlich eine Rolle, weil der Batchprozess anders als die Online-Anwendung massenhaft Kontonummern instanziiert. Für den Batch wurde deshalb die Implementierung entsprechend angepasst. Die Kontonummern-Klasse existiert jetzt noch in einer zweiten Variante mit nur acht Attributen, und zwar genau denen, die fachlich für den Batchprozess benötigt werden. Allein das führte schon zu einer massiven Reduktion des Hauptspeicherbedarfs und damit dazu, dass der Batchprozess jetzt im geforderten Zeitfenster läuft (siehe Listing 2 und 3).

Gerade in der Batchverarbeitung unter Java kann es nötig sein, ganz bewusst auf Objektorientierung zu verzichten, da sich ansonsten die Anforderungen an Verarbeitungsgeschwindigkeit und Speicherbedarf überhaupt nicht erfüllen lassen.

Eine beliebige Datenbank-Tabelle besitzt zum Beispiel eine Spalte „Zustand“. Nun sollen alle Datensätze, deren Zustandsattribut gleich „1“ ist, auf den Zustandswert „3“ geändert werden. Objektorientiert würde man nun alle relevanten Datensätze als Objekt-Repräsentation im Hauptspeicher instanziiert, ändern und zurückschreiben. Das wird bei sehr großen Datenmengen jedoch nicht mehr funktionieren. Viel perfor-

manter wäre es, ein entsprechend ausformuliertes Update-SQL direkt abzusetzen, was aber in keiner Weise mehr objektorientiert ist. Das wäre im Batch im Zweifelsfall aber die Lösung der Wahl. Performance-Anforderungen können also Spezial-Implementierungen notwendig machen.

An den letzten beiden Beispielen aus dem Spannungsfeld „Batch und Online“ lässt sich noch einmal verdeutlichen, wo der Unterschied zwischen technischer und fachlicher Portabilität liegt. Rein technisch gesehen kann man das Objekt mit den zweiundzwanzig Attributen auch im Batch benutzen. Man muss nichts anpassen oder konfigurieren. Nur in der Praxis führt die Implementierung dazu, dass die Antwortzeiten viel zu groß sind, wodurch die Implementierung für diesen speziellen Batchfall einfach unbrauchbar ist. Das Gleiche gilt für die Änderung des einzelnen Zustandsattributs. Prinzipiell kann man die objektorientierte Lösung auch in der Massen-Datenverarbeitung einsetzen. Sie erfordert dann nur massenhaft Zeit.

Caching, statische Variablen, Pseudo-Konstanten und Portabilität

Zum Abschluss betrachten wir noch einen komplexeren Aspekt beim Thema „Portabilität“: Welche Auswirkungen kann es haben, wenn man in seiner Implementierung aus Performance-Gründen einzelne Werte oder Wertemengen in statischen Variablen im Cache hält? Es gibt eine ganze Reihe von Fragen, die man sich dabei stellen muss.

Zuerst: Was merkt man sich überhaupt? Nicht alles ist in jeder Umgebung sinnvoll. Muss man im Batch Tausende von Produkteinstellungen in den Cache einlesen, wenn man nur zwei oder drei davon braucht? In einer Online-Anwendung kann das durchaus sinnvoll sein. Wie wird der Cache initialisiert? Automatisiert oder manuell? Lädt man die Daten „lazy“ oder alle auf einmal? Jede Umgebung kann da ihre eigenen Anforderungen haben.

Wenn die Daten dann im Hauptspeicher vorliegen: Muss der Zugriff synchronisiert werden, weil in bestimmten Umgebungen mehr als einer zugreifen kann? Funktioniert die Initialisierung überhaupt, wenn mehrere zugreifen können?

Wie sieht es mit der Lebensdauer aus? Ist ein Refresh der Daten erforderlich, weil sie sich ändern können und das fachliche

Auswirkungen hat? Wenn ja, wann? Muss man den Refresh synchronisieren?

Zum Abschluss: Wie vermeidet man Speicherlecks? Wer räumt die Daten auf, wenn sie nicht mehr benötigt werden? Arbeitet der Cache „stateful“ oder „stateless“?

Gerade wenn man seine Implementierung in mehreren unterschiedlichen Umgebungen einsetzen will, sollte man diese Fragen nicht einfach unter den Tisch fallen lassen.

Zum Thema „Lebensdauer“ gibt es ein schönes Beispiel aus der Praxis der Autoren, das verdeutlicht, was passieren kann, wenn man sich nicht bewusst ist, wie das Gesamtsystem arbeitet: Eine Online-Anwendung holt sich vom Mainframe das aktuelle Buchungsdatum, wenn es fachlich benötigt wird. Da sich dieser Wert über den Tag nicht ändert und der Zugriff in der Online-Anwendung zu einer spürbaren Verlängerung der Antwortzeiten führt, kam ein Entwickler auf die Idee, das Buchungsdatum beim ersten Zugriff in einer „static“-Variablen abzulegen. Bei allen folgenden Zugriffen wird der Wert dann aus dieser Variablen gelesen. Die Folge waren gewaltige Performance-Steigerungen beim Test aus der Entwicklungsumgebung heraus.

Irgendwann zog diese Implementierung in die Integrations-Testumgebung auf einen App-Server um, der nicht jeden Tag neu gestartet wird. Zu Anfang war noch alles in Ordnung. Aber nach einem Tag waren im Integrationstest Buchungen nicht mehr durchführbar, weil das Buchungsdatum seit Mitternacht vor dem Tagesdatum lag und entsprechende Prüfungen fehlschlugen. Keiner wusste, warum.

Irgendwann wurde der Server aus anderen Gründen neu gestartet. Danach lief alles wieder ordnungsgemäß. Bis zum nächsten Tag. Alles nur, weil ein Entwickler von falschen Annahmen ausging, was die Lebensdauer seiner Variablen anging. Im Batch hätte die Implementierung übrigens tadellos funktioniert, weil dort sichergestellt ist, dass kein Batchprozess über den Tageswechsel hinaus läuft (siehe Listing 4).

Fazit

Die Erfahrung zeigt, dass die Portabilität einer Anwendung in der Praxis nicht nur am Sourcecode festgemacht werden kann, sondern auch fachliche Aspekte einen großen Einfluss auf die Portabilität haben. Technisch gesehen sind Java-Anwendungen vergleichsweise leicht zu portieren.

Oft muss man sich um die technische Portabilität gar keine Gedanken machen.

Die Portabilitäts-Problematik erstreckt sich beim Einsatz von Java deshalb oft größtenteils auf nichttechnische Faktoren. Diese sind aber kein spezielles Java-Problem, sondern ein Problem für jede Portierung, egal in welcher Programmiersprache und in welcher Technologie. Dass diese Probleme häufig kaum wahrgenommen werden, liegt daran, dass das Wissen um die technischen Grundlagen immer mehr verblasst und in vielen Fällen die Portierbarkeit von Java-Sourcecode gar nicht als Problem gesehen wird.

Portabel zu entwickeln, heißt immer auch, Kompromisse zu machen. Am deutlichsten wird das vielleicht bei der Entwicklung von Oberflächen für unterschiedliche Plattformen.

In der Praxis wird nur dann wirklich portabel entwickelt, wenn man auch dazu gezwungen ist. Bleibt man immer nur auf einer Plattform, schleichen sich früher oder später Dinge ein, die die Portabilität einschränken – oft unbewusst. Es zeigt sich jedoch auch, dass dieser Effekt deutlich geringer ist, wenn man schon in Entwicklung und Test auf unterschiedlichen Plattformen

```
public class BuchungsManager {
    // ...

    private static Datum AKTUELLES_BUCHUNGS_DATUM = null;

    // ...

    public static Datum getaktuellesBuchungsDatum() {
        if ((AKTUELLES_BUCHUNGS_DATUM == null) || (AKTUELLES_BUCHUNGS_DATUM < Datum.heute()))
        {
            // Datum erstmals holen bzw. refreshen, wenn ein Tageswechsel stattgefunden hat
            AKTUELLES_BUCHUNGS_DATUM = BuchungsManager.getaktuellesBuchungsDatumFromMainframe();
        }

        return AKTUELLES_BUCHUNGS_DATUM;
    }

    // ...
}
```

Listing 4: Buchungsdatum richtig cachen

arbeitet. Gerade die gemeinsame Verwendung von Fachfunktionen und -objekten im Online- und Batch-Bereich in unserem Referenzprojekt führte zu einer ganz anderen Sichtweise, was Portabilität angeht.

Was man nicht vergessen darf: Während des Lebenszyklus einer Software kann man nicht immer alles bedenken. Woher will man zu Projektbeginn wissen, wie die eigenen Implementierungen in Zukunft genutzt werden oder wie sich die eingesetzte Programmiersprache weiterentwickeln wird? Deshalb wird absolute Portabilität immer ein Wunschtraum bleiben. Man kann allerdings den Aufwand beeinflussen, den man treiben muss, um auf eine andere Plattform zu wechseln. Und da nimmt einem Java schon eine ganze Menge Arbeit ab.

Thomas Niedergesäß
thomas.niedergesaess@tricept.de



Thomas Niedergesäß ist bei der Tricept Informationssysteme AG als Chief Architect Java Solutions verantwortlich für die strategisch-technologische Ausrichtung des Java-Bereichs. Als Spezialist für Java-Enterprise-Anwendungen berät und coacht er seit mehr als zehn Jahren namhafte Kunden vor Ort.

Burkhard Seck
burkhard.seck@tricept.de



Als Head of Java Solutions koordiniert Burkhard Seck die Aktivitäten der Tricept Informationssysteme AG im Java-Umfeld. Er verfügt über mehr als ein Jahrzehnt an Erfahrung als Entwickler und Architekt von Enterprise-Java-Anwendungen und berät deutschlandweit große Kunden, vornehmlich aus dem Finanzsektor.

Prinzipien des API-Managements

Jochen Traunecker und Tobias Unger, gridsolut GmbH & Co. KG

APIs sind in aller Munde und mit dem WSO2 API Manager steht eine passende Open-Source-Lösung zum Management von APIs bereit. Um das API-Management besser einordnen zu können, müssen APIs von Services abgegrenzt und ihre jeweiligen Lebenszyklen genauer betrachtet werden.

Mittlerweile haben Service-orientierte Architekturen (SOA) den typischen Hype-Cycle durchschritten und sind auf dem Plateau der Produktivität angekommen [1]. Zur Implementierung einer SOA stehen heute ein breites Angebot an ausgereiften Middleware-Produkten diverser kommerzieller Hersteller sowie eine Vielzahl an Open-Source-Projekten zur Verfügung. Die Integration lose gekoppelter Services kann sich dabei auf bewährte und dokumentierte Musterlösungen (Enterprise Integration Patterns [2]) stützen. Trotzdem scheint die tatsächliche Wiederverwendung von entwickelten Services gerade innerhalb von Unternehmen noch nicht wirklich gelebt zu werden. Dies steht im Gegensatz zu allgemein verfügbaren und sehr populären Services, wie sie zum Beispiel durch Twitter angeboten werden.

Bei einer genaueren Betrachtung fällt auf, dass in diesem Kontext häufig nicht von

einem Service gesprochen wird, sondern von einer Programmier-Schnittstelle (engl. Application Programming Interface (API)).

Abbildung 1 zeigt ein typisches Beziehungsgeflecht zwischen API-Nutzern (Clients), APIs und durch APIs exponierte Services. Dargestellt ist das klassische Überweisungsszenario einer Bank. Eine Überweisung bedingt dabei drei Schritte: Prüfung auf Missbrauch, Abbuchung des Überweisungsbetrags auf Seiten des Auftraggebers sowie Gutschrift auf Seiten des Empfängers, wobei Fehlerbehandlungen und Transaktions-Management in diesem Beispiel nicht betrachtet werden.

Zwei Dienste (Express Transfer Service und Standard Transfer Service) kombinieren nun die drei beschriebenen Schritte (Fraud-Check-, Debit- und Credit-Service) und exponieren selbst einen Überweisungsdienst, der durch eine Schnittstellen-Definition (Transfer Interface) spezifiziert

wird. Die beiden Dienste selbst unterscheiden sich durch Dienstgüte-Zusagen (engl. Service Level Agreement (SLA)).

Beide Transfer-Services werden mittelbar von Bank-Anwendungen wie Online-Banking (Web Banking Application) oder Mobile Banking (Mobile Banking Application) aufgerufen: Das Banking-API-Gateway exponiert durch ein API die beiden Transfer-Services. Dabei ist durch das Gateway sichergestellt, dass nur autorisierte Anwendungen das API aufrufen können. Daneben werden der API-Aufruf überwacht und die den SLAs entsprechenden Dienste ausgewählt.

Service und API-Lebenszyklen

Das Beispiel des Banking-API (siehe Abbildung 1) verdeutlicht die unterschiedlichen Rollen eines Service und eines API: Der Service deckt fachliche Aspekte ab und das API ist für die Beziehung zwischen

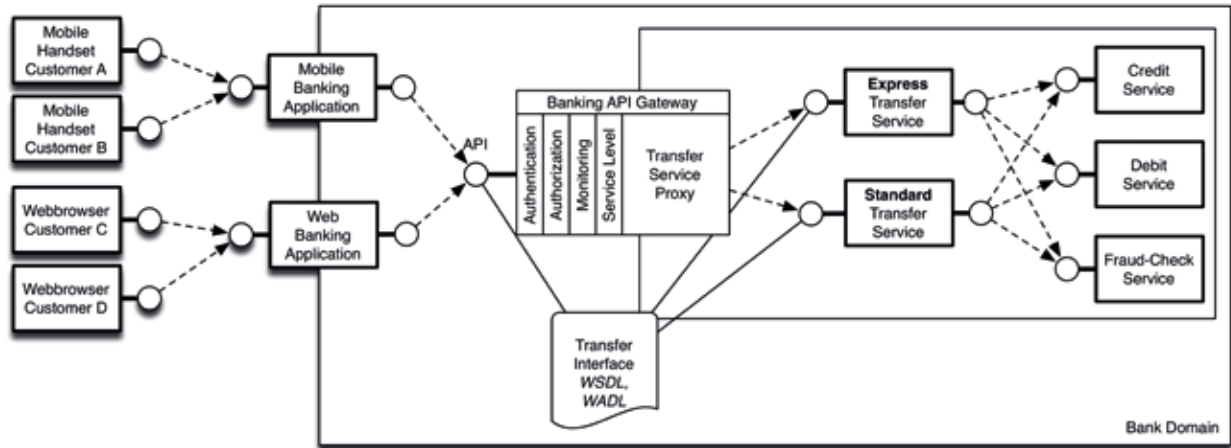


Abbildung 1: Beispiel eines Banking-API

Service-Konsument und Service-Anbieter verantwortlich.

Ein API und ein Service bilden zusammen eine Einheit, durchlaufen jedoch individuelle Lebenszyklen (siehe Abbildung 2). Typischerweise wird ein Service spezifiziert, entwickelt und nach erfolgreichem Testen freigegeben. Mit Freigabe kann ein Service ausgerollt werden und in Produktion gehen. Anschließend kann darauf aufbauend ein API erzeugt und veröffentlicht werden. In einem weiteren Schritt wird das API des Lebenszyklus als überholt (deprecated) markiert und schließlich zurückgezogen. Sobald ein API als überholt markiert ist, können sich keine neuen Klienten darauf registrieren. Wurde das API zurückgezogen, werden keine Aufrufe mehr an den Service durchgereicht und dieser kann entfernt beziehungsweise durch eine neue Version ersetzt werden.

API-Management

Die Implementierung eines API-Managements kann durch die in Abbildung 3 skizzierten Komponenten unterstützt werden. „Runtime“ stellt die technologische Infrastruktur zum Betrieb des API-Gateways und der Services bereit. Dort erfolgen diverse Policies wie Autorisierung, SLAs und Logging. Ein API-Entwickler nutzt einen „API-Publisher“ zum Entwickeln, Veröffentlichlichen, Überwachen und Verwalten von APIs. Ein API-Konsument bedient sich wiederum eines „API-Stores“, um APIs zu suchen, deren Dokumentation zu sondieren und sich schließlich als Konsument auf einem API zu registrieren (abonnieren). Mit dem Abonnieren eines API kann dieses evaluiert, getestet und schließlich produktiv genutzt werden. Eingebettet sind diese drei Komponenten zum API-Management

in eine „Monitoring- und Auswertungs-Infrastruktur“.

Alle vier Komponenten bilden die Basis für ein API-Management, das durch die Entkopplung der API-Entwickler und API-Konsumenten unterstützt wird. Ein API-Entwickler kann jederzeit im API-Store abfragen, welche Konsumenten ein API nutzen und wie häufig das API aufgerufen wurde. Hilfsmittel wie Diskussionsforen, E-Mail-Listen, Bug-Tracker und Wikis unterstützen den Dialog zwischen API-Entwicklern und -Konsumenten.

Ist ein API als überholt markiert, kann der API-Store alle darauf registrierten API-Konsumenten aktiv informieren und diese sind in der Lage, entsprechend zu reagieren. API-Konsumenten können selbst wiederum im API-Store APIs bewerten und kommentieren und damit anderen potenziellen API-Konsumenten Hinweise geben.

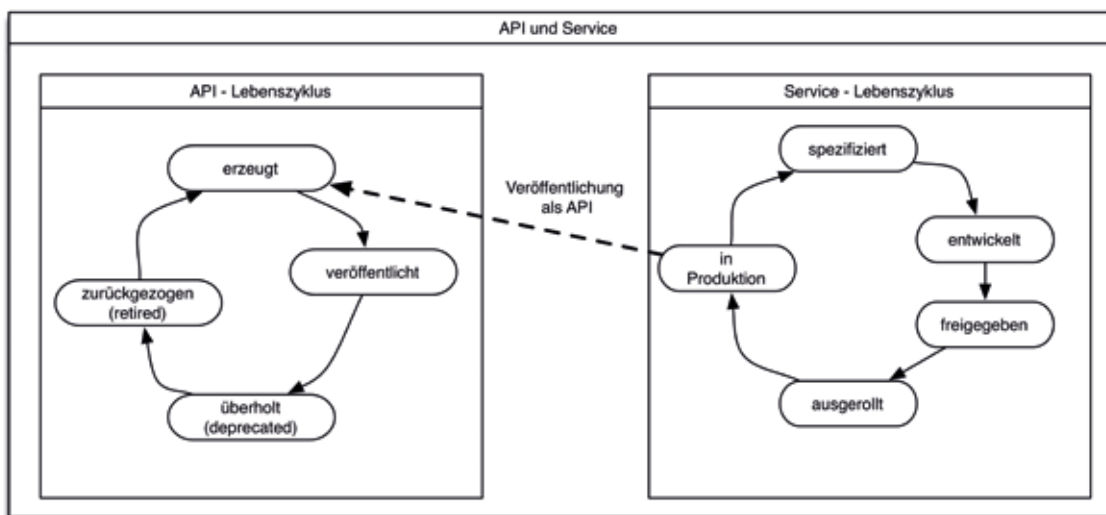


Abbildung 2: API und Service-Lebenszyklen

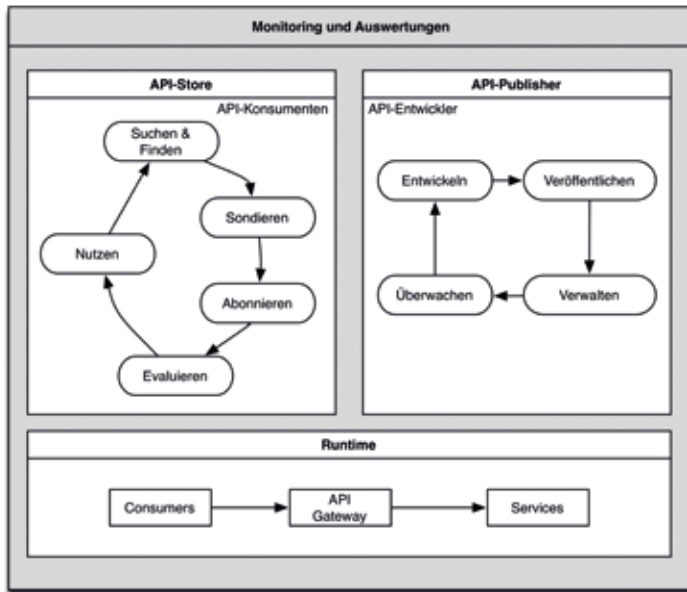


Abbildung 3: API-Management-Komponenten

Die Monitoring- und Auswertungs-Infrastruktur ist beim API-Management in vielfacher Weise nutzbar. Neben der Überwachung eines API lässt sich die Drosselung (Throttling) für jeden Konsumenten individuell einstellen. Darüber hinaus sind vielfältige Monetarisierungsmodelle möglich.

Der WSO2 API Manager

Aufbauend auf OSGi-Komponenten der WSO2-Middleware-Plattform bietet sich

der WSO2 API Manager als 100-prozentige Open-Source-Lösung (Apache 2.0 Lizenz) zum API-Management an. Die in Abbildung 3 schematisch dargestellten Komponenten einer API-Management-Plattform sind in dieser Open-Source-Lösung realisiert. So stehen ein API-Publisher und ein API-Store zur Verfügung, die sowohl von Entwicklern durch eine Web-Oberfläche als auch programmatisch durch Web-Services genutzt werden können. Die API-Management-

Runtime wird durch den WSO2 Enterprise Service Bus (ESB) bereitgestellt. Das Monitoring übernimmt der WSO2 Business Activity Monitor (BAM). Zur Authentifizierung und Autorisierung dient der WSO2 Identity Server, der in diesem Fall OAuth 2.0 nutzt.

Abbildung 4 zeigt die Web-Oberfläche des API-Publishers mit einem einfachen Demo-API. Um das Beispiel einfach zu halten, wird ein öffentlich frei zugänglicher Dienst aufgerufen, der auf einen HTTP-GET-Request die IP-Adresse des Aufrufenden zurückgibt. Im Unternehmens-Umfeld könnten analog zum hier gezeigten Beispiel interne JAX-WS- oder JAX-RS-Dienste eingebunden sein. Es wird deutlich, dass lediglich eine aktive Subskription (ein User) vorliegt und das API in der Version 1.0.0 im Zustand „published“ ist.

Für Konsumenten bietet der API-Store wiederum die Möglichkeit, sich für dieses API zu registrieren (siehe Abbildung 5). Dabei erhält der Konsument einen Consumer Key mit Consumer Secret und zum schnellen Ausprobieren lassen sich Access Token direkt erzeugen. Ein schlichter CURL-Aufruf bestätigt, dass alles wie gewünscht zusammenspielt (siehe Listing 1).

Da der WSO2 API Manager ein breites Spektrum an Protokollen und Serialisierungen unterstützt (SOAP, REST, JSON, XML etc.), sollten die meisten Anwendungsfälle abgedeckt sein. Eine Auflistung und Beschreibung aller Features steht auf der Projektseite [3].

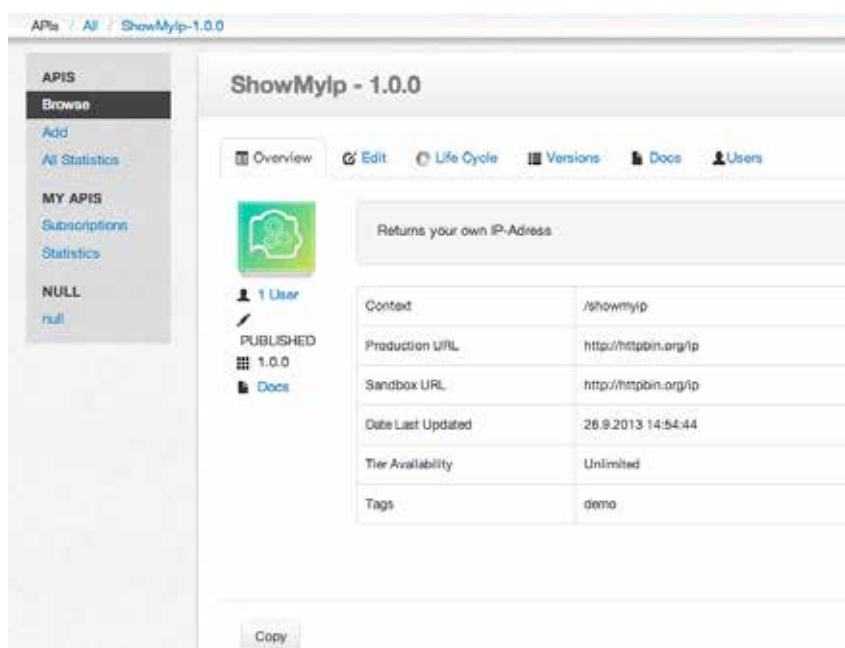


Abbildung 4: WSO2 – API-Publisher

Fazit

Durch die Einführung eines API-Managements mit API-Store, API-Publisher, Runtime und Monitoring steht in größeren SOA-Landschaften ein mächtiges Werkzeug zur Beherrschung der Gesamt-Landschaft zur Verfügung. Aber auch vermeintlich harmlose Aufgaben wie die Bereitstellung von Daten für mobile Anwendungen als REST-Services können durch ein API-Management wie dem WSO2 API Manager unterstützt werden. Die eigenen Entwickler können sich auf die Bereitstellung der geschäftlich relevanten Daten konzentrieren und müssen die Authentifizierung, Autorisierung und Überwachung der Aufrufenden nicht ausprogrammieren.

Die Einführung eines API-Managements befreit jedoch nicht von der fachlichen Architektur-Arbeit, denn die Prinzipien des Domain Driven Designs [4] gelten

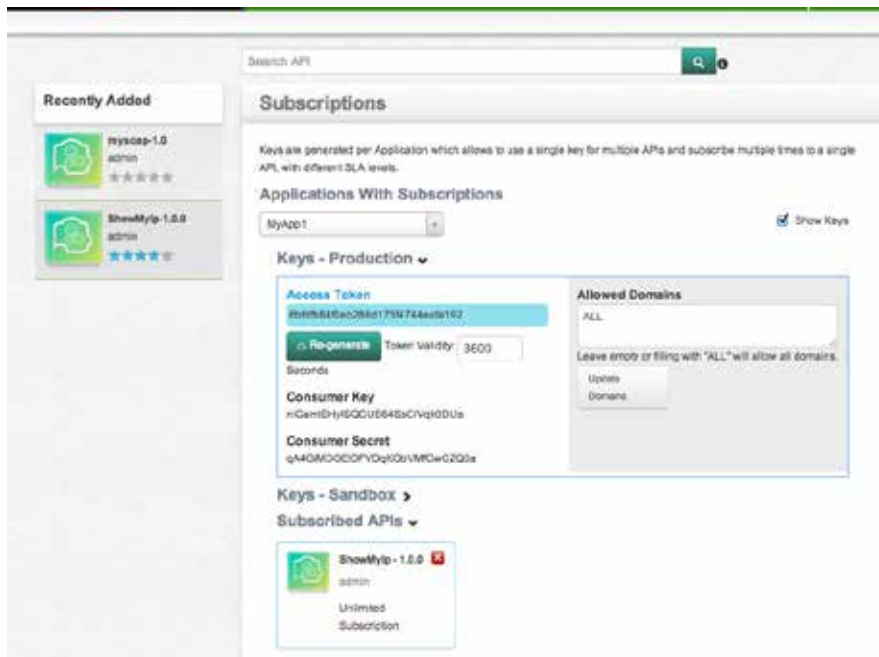


Abbildung 5: WSO2 – API-Store

```
curl --header „Authorization: Bearer 8bfdfb84f0eb288d175f-4744eafa192“ http://10.234.0.7:8280/showmyip/1.0.0 //Antwort
{
  "origin": "109.193.77.44"
}
```

Listing 1

auch in einer SOA. Ein API-Store an sich kann die Wiederverwendbarkeit von Services nicht garantieren. Doch unterstützt das API-Management die kontinuierliche Weiterentwicklung einer SOA-Landschaft dahingehend, dass Änderungen an Services

(neue Versionen bereitstellen, alte Version zurückziehen) beherrschbar bleiben.

Links

- [1] <http://www.infoq.com/news/2012/08/Gartner-Hype-Cycle-2012>
- [2] <http://www.eaipatterns.com>

- [3] <http://wso2.com/products/api-manager>
- [4] Eric Evans: Domain Driven Design, Addison-Wesley 2004, ISBN 0-321-12521-5

Jochen Traunecker

jochen.traunecker@gridsolut.de



Jochen Traunecker (Dipl.-Ing.sc.agr, Dipl.-Inf.) ist Gründer und geschäftsführender Gesellschafter der GRIDSOLUT und kann auf viele Jahre Praxis im Bereich des Software-Engineering zurückblicken.

Tobias Unger

tobias.unger@gridsolut.de



Tobias Unger (Dipl.-Inf.) verfügt über mehrjährige Erfahrung in den Bereichen „BPM“ und „Java Enterprise“ sowohl im praktischen wie auch auf dem wissenschaftlichen Gebiet, insbesondere bei der Prozessausführung und Aufgabenverwaltung. Er ist Autor zahlreicher Publikationen und als Sprecher auf Konferenzen aktiv.

„Das Wachstum und die Komplexität von Java EE 7 sind nichts Ungewöhnliches ...“

Interview mit Mark Little, Vice President Middleware Engineering bei Red Hat, über Java EE 7 und den Java Community Process. Ursprünglich bekannt durch sein Betriebssystem Red Hat Enterprise Linux, kaufte Red Hat im Jahr 2006 JBoss und erweiterte mit der Enterprise Middleware sein Lösungsportfolio.

Red Hat hat sich sehr intensiv an der Entwicklung von Java EE 7 beteiligt. Können Sie etwas zu Ihren Beiträgen zu Contexts and Dependency Injection (CDI) 1.1, der Bean Vali-

dation und der Entwicklung einer Reihe weiterer Java Specification Requests (JSRs) sagen?

Mark Little: Wir verantworteten die CDI-

und Bean-Validation-Updates, da wir bereits bei Java EE 6 diese Projekte leiteten. Darüber hinaus waren wir bei anderen JSRs wie Java Transaction API (JTA), Java EE

Connector Architecture (JCA) und dem Update des Java Message Service (JMS) aktiv. Auf die eine oder andere Weise beteiligte sich Red Hat bei allen JSRs, die bei Java EE 7 aktualisiert wurden. Selbst dann, wenn wir nicht selbst für ein Projekt zuständig sind, legen wir großen Wert darauf, dass die Anliegen unserer Kunden und der Open-Source-Community in die Weiterentwicklung von Java EE einfließen.

Welchen Stellenwert haben diese JSRs für Java EE 7 und welche Erfahrungen haben Sie bei der Verbesserung und Weiterentwicklung gesammelt?

Mark Little: Unserer Meinung nach ist CDI die wichtigste Erweiterung der Java-EE-Architektur der letzten Jahre. Das betone ich nicht nur, weil Red Hat die Projektleitung innehatte, denn im Grunde genommen war es Teamwork. Ich sage dies, weil wir von vielen Herstellern hören, dass es lange Zeit sehr aufwändig war, Applikationen mit dem Standard-Java-EE-Stack zu entwickeln. Mit CDI geht das jetzt deutlich einfacher. Anno-

tationen sind eine wichtige Bereicherung der Sprache und werden auf einer Vielzahl von Gebieten eingesetzt. Schon seit der Verfügbarkeit von Java EE 6 ist dieser Trend erkennbar. Wir sehen immer mehr Entwickler, die bislang Java EE 6 nicht auf dem Radar hatten, sich heute aber dessen funktionale Optionen genauer ansehen, um damit produktiver Enterprise-Applikationen erstellen zu können. Und in diesem Zusammenhang erwähnen sie immer wieder CDI. Vor diesem Hintergrund war es klar, dass wir das Projekt der CDI-Aktualisierung in Java EE vorantreiben wollten, da es einige Aspekte gab, die wir in der CDI-Version von Java EE 6 noch nicht einbauen konnten. Darüber hinaus erhielten wir auch Feedback von Anwendern, die CDI mit Java EE 6 nutzten, das wir berücksichtigen wollten. Wir orientierten uns im Projektablauf und im Vorgehen an den bei Java EE 6 bewährten Abläufen. All unsere Prozesse sind offen – wir haben eine offene Mailingliste, alle Beteiligten sehen, was gerade diskutiert wird und es gibt ein Open Issue Tracking. Die Drafts durchlaufen einen umfangreichen Revisionsprozess.

Welchen Einfluss hat Java EE 6 Web Profile auf die Technologien von Red Hat? Was ändert sich bei Java EE 7 hinsichtlich neuer APIs wie WebSocket und JSON-P?

Mark Little: Das Wachstum und die Komplexität von Java EE 7 sind nichts Ungewöhnliches. Wenn man CORBA, DCE und andere Standards betrachtet, kann man ähnliche Phänomene erkennen. Java EE 6 reagierte bereits auf diese Entwicklung mit der Einführung des Web Profile – im Grunde genommen eine vereinfachte Form des Full Profile. Was Red Hat betrifft, haben wir früher unserer Community eine eigene Profile-Version bereitgestellt. Es gab keinen Standard, auf dem wir aufbauen konnten, aber wir waren in der Lage, den Stack zu optimieren. Wollte ein Kunde beispielsweise keine Web Services, ließen wir die einfach weg. Ich denke, es war daher eine gute Sache, Web Profile als Standard einzuführen. Das Feedback, das wir daraufhin erhielten, war sehr positiv. Es gibt viele Entwickler, die sich jetzt Web Profile genauer anschauen und dann möglicherweise dorthin upgraden. Der Grund ist



TEAM - Ihr Partner für innovative IT-Lösungen

Als Oracle Platinum Partner bieten wir ein umfassendes Dienstleistungsspektrum rund um die Oracle-Technologien.

- ADF Entwicklung Best Practices
- Von Forms zu Java
- Workshop und Coaching
- Oracle WLS/GlassFish Server Installation/Administration
- Individualentwicklung
- TEAM ADF-Tools





Mark Little, Vice President Middleware Engineering bei Red Hat

einfach: Das Full Profile verfügt über einige Funktionen, die in Web Profile nicht vorhanden sind, von Entwicklern aber benötigt werden. Es ist auf jeden Fall eine gute Möglichkeit, mehr Anwender zu haben – und aus der Sicht von Red Hat, mehr neue Kunden zu gewinnen. Mit Java EE 6 gab es eine Reihe neuer APIs, etwa JAX-RS und CDI, die wegen ihrer Vorteile von Entwicklern positiv aufgenommen wurden. Bei Java EE 7 sind weitere APIs wie JSON-P und WebSocket hinzugekommen. Beide tragen dazu bei, dass der Java-Enterprise-Stack in den Bereichen „Cloud“ und „Mobile“ eine Vorreiterrolle einnimmt.

Eine Vielzahl von Cloud-bezogenen Features, die ursprünglich mit Java EE 7 verfügbar sein sollten, wurde auf Java 8 verschoben. Wie kam es dazu? Wie wird Java EE 7 heute in Cloud-Umgebungen eingesetzt und wie wird sich die Technologie in künftigen Releases weiterentwickeln?

Mark Little: Aus Gesprächen in den Communities und mit unseren Kunden wissen wir, dass Anwender Applikationen aus ihrer internen Infrastruktur in eine andere verlagern wollen, ohne dass sie die Applikationen neu implementieren müssen. Die beste Möglichkeit ist, dafür zu sorgen, dass die bisherigen Plattformen, die auf der eigenen Hardware liefen, auch in der Cloud zur Verfügung stehen. Wir haben daher von Anfang an darauf geachtet, dass unsere Implementierungen als Service auf der Infrastruktur laufen und damit als Platform as a Service. Als Java EE 6 kam, ergab sich eine einfachere Möglichkeit

zur Realisierung. Mit den Profiles von Java EE 6 stand Kunden, die ihre Applikationen in der Cloud betreiben wollen, ein standardisiertes Web Profile und eine standardisierte Full-Profile-Plattform zur Verfügung. Wir haben im Jahr 2011 unsere eigene Platform as a Service vorgestellt, die anfangs auf einem Pre-Release unseres Java-EE-6-kompatiblen Application Server basierte. Dann folgte das vollständig kompatible Release in JBoss Enterprise Application Platform (EAP) 6, mit der wir unsere Platform-as-a-Service-Lösung OpenShift ankündigten. Während der Entwicklung von Java EE 7 entstand eine Reihe neuer JSRs, die sich speziell mit dem Einsatz von Java EE in der Cloud befassen. Aber wie gesagt: Auch Java EE 6 ist bereits sehr gut für die Cloud geeignet. Es gibt Bereiche, in denen Verbesserungen wünschenswert sind, beispielsweise in puncto Modularität und Multi-Tenancy. Aber ich denke, es war richtig, die Realisierung dieser Features aufzuschieben, damit wir Java EE 7 rechtzeitig fertigstellen konnten. Die angesprochenen Features kommen im nächsten Release und deshalb wird sich Java EE 8 noch besser für die Erstellung von Cloud-Applikationen eignen.

Wie wird JBoss Developer Studio IDE die neuen Möglichkeiten von Java EE 7 aufgreifen?

Mark Little: Wir wollen mit JDBS Entwickler bei ihrer Arbeit begleiten, damit wir schnell sehen, wo es Probleme gibt. Wenn beispielsweise etwas bei CDI nicht so läuft, wie es sollte, erhalten wir über JDBS rasch Feedback von unseren Entwicklern. Es ist daher enorm wichtig, dass wir neue Features möglichst schnell in die IDE einbauen, damit Entwickler produktiver werden.

Wenn wir einmal das Gesamtbild ihres JCP-Engagements betrachten, wie lösen Sie Interessenkonflikte, die sich aus der Zusammenarbeit mit anderen Herstellern ergeben?

Mark Little: Java und Java EE haben für große Teile unserer Branche einen sehr hohen Stellenwert. Es ist daher in unser aller Interesse, bei der Weiterentwicklung zusammenzuarbeiten. Da gibt es keinen Unterschied zur Arbeit in anderen Standardgremien wie OASIS oder W3C. Immer wieder müssen offene Punkte geklärt werden und nicht jeder kann seine Position vollständig durchsetzen; man muss kompromissfähig sein.

Wie kann JCP die Java Community besser unterstützen? Was ist zu tun, damit sich mehr Entwickler und Unternehmen beteiligen?

Mark Little: Die verschiedenen Java User Groups, beispielsweise rund um Adopt-a-JRS, leisten sehr gute Arbeit. Alles, was wir bei Red Hat machen, wird durch die Open Source Community beeinflusst. Wir erhalten viel Feedback zu Vorschlägen, die in verschiedene JSRs eingehen, bevor sie tatsächlich formal dokumentiert werden. Auf diese Art und Weise sind unsere Communities an der Arbeit des Executive Committee beteiligt.

Auf der JavaOne 2012 haben Sie und Cameron Purdy einen Vortrag gehalten, der erläuterte, wie Nicht-Sprachen in einer virtuellen Maschine arbeiten – beispielsweise, wie ein Ruby-Entwickler den Java-EE-Stack benutzt und dabei Features wie Java Message Service (JMS) oder Enterprise Java Beans (EJBs) einsetzt. Was bedeutet das für eine größer werdende Java-Community oder ein Ecosystem?

Mark Little: In den letzten Jahren haben sich Sprachen wie JRuby, Clojure und Scala, die in der JVM laufen, immer stärker verbreitet. Auch wir entwickeln eine eigene Sprache namens „Ceylon“. Wir arbeiten mit diesen Communities zusammen und ermutigen Entwickler, die eine hochperformante Messaging-, Transaktions- oder Datenbank-Lösung benötigen, dass sie das Rad nicht neu erfinden, da es bereits eine Lösung für ihre Zwecke in Form des Java-EE-Stack gibt. Wir versuchen daher genau herauszufinden, was sie von einer vorhandenen API, beispielsweise für das Messaging, erwarten, entwickeln mithilfe der Community eine Lösung und unterstützen diese dann mit unserem gesamten oder zumindest einem Teil unseres Stacks. Ruby-Entwickler definieren das API und stellen die Verbindung des API mit unserer JMS-Implementierung bereit. Wir wollen, dass diese neuen Sprach-Communities nicht die gleichen Fehler machen wie wir vor zwanzig Jahren. Für sie ist es einfacher, das zu verwenden, was vorhanden ist, und dann zu verbessern.

Hinweis: Das Interview führte Janice J. Heiss, Java Acquisitions Editor bei Oracle und Technologie-Redakteurin für das Java Magazine. Ins Deutsche übersetzt von Red Hat.

Eine Android-App samt JEE-Back-End generieren und in der Cloud bereitstellen

Marcus Munzert, Generative Software GmbH

Android ist zurzeit das meistgenutzte mobile Betriebssystem. In vielen Fällen, vor allem auch bei Business-Anwendungen und bei Anwendungen, die bisher klassischen Embedded-Geräten vorbehalten waren, bietet eine native Entwicklung mit Java Vorteile. Und die Java Enterprise Edition (JEE) hat alles, um ein passendes, leistungsfähiges Back-End an den Start zu bringen. Der Artikel beschreibt eine Software-Architektur für dieses Szenario und zeigt, wie man durch Code-Generierung schnell und zuverlässig eine entsprechende Anwendung entwickeln kann.

Eine der Herausforderungen bei der Entwicklung einer Android-Anwendung, die mit Servern kommuniziert, ist ein effektiver, effizienter und zuverlässiger Datenaustausch zwischen Gerät und Server. Häufig wird der Architektur-Stil „Representational State Transfer“ (REST) genutzt. Als Client-Server-Protokoll kommt meist HTTP(S) zum Einsatz.

Während beim Austausch-Datenformat früher der Schwerpunkt auf XML lag, hat inzwischen „JavaScript Object Notation“ (JSON) die Nase vorn. Dienste wie YouTube, Twitter und Box bieten in ihren neuesten API-Versionen ausschließlich JSON als Datenformat an. All diese Zutaten bilden die Grundlage der Architektur, die in [Abbildung 1](#) schematisch dargestellt ist.

Aktuell widmet sich übrigens das Forschungsprojekt „mohito“ [1] der Problematik, Daten über mehrere Plattformen hinweg zur Verfügung zu stellen und zu synchronisieren. Dort wird, ebenso wie im weiter unten beschriebenen Verfahren, auf den Einsatz von domänenspezifischen Sprachen (DSL) und modellgetriebener Software-Entwicklung gesetzt. Doch zunächst die Architektur.

REST-API mit JEE

Bei der Gestaltung eines REST-API ist darauf zu achten, dass die fünf notwendigen REST-Eigenschaften berücksichtigt sind [2]:

- Einheitliches Interface – unabhängig von einem Client, der es nutzt

- Zustandslosigkeit; ein Server-Aufruf hängt also nicht von vorherigen Aufrufen ab
- Möglichkeit der Zwischenspeicherung der Rückgabe-Daten in einem Cache zwischen einem Client und dem eigentlichen Server
- Klare Trennung der Aufgaben von Client und Server
- Ein in Schichten aufgeteiltes System

Es ist hierbei lediglich ein organisatorischer Unterschied, ob das REST-API öffentlich verfügbar sein soll oder ob es ausschließlich von Clients genutzt wird, die zusammen mit dem API unter ein und derselben

Kontrolle liegen. Im letzteren Fall sind leichter Änderungen am API durchführbar, da neuere Versionen der Clients gleichzeitig mit dem neuen, serverseitigen API geplant und veröffentlicht werden können. Auch das in diesem Artikel vorgestellte, generierte API eignet sich für beide Fälle. Es kann also auch von weiteren, nicht generierten Clients genutzt werden.

JAX-RS [3] ist Teil von JEE und bietet alles, um ein REST-API zu implementieren. Eine sogenannte „Resource“ bildet den Einstiegspunkt in das REST-API. Von hier aus gibt es zwei Möglichkeiten, wie auf Business-Logik und die Persistenzschicht zugegriffen werden kann. Entweder wer-

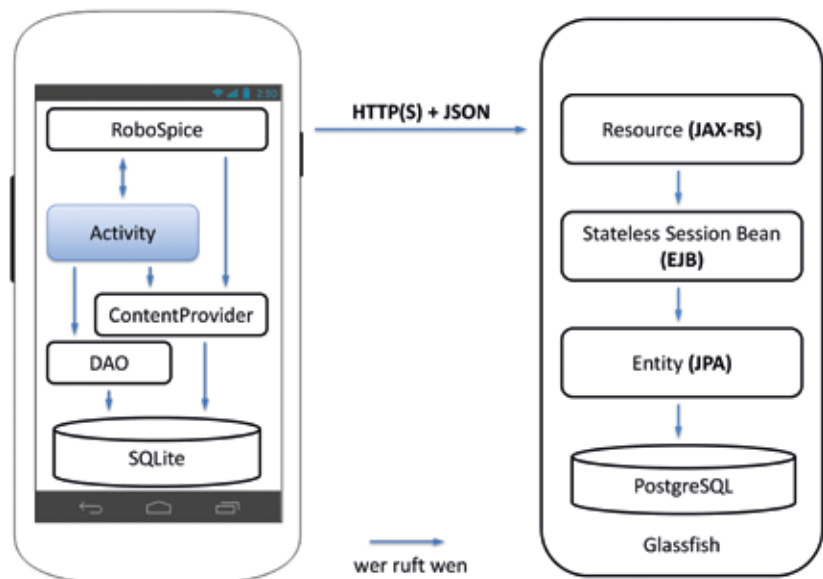


Abbildung 1: Schema der Software-Architektur

den in einem EAR-File bereitgestellte Session Beans angesprochen oder die Resource ist selbst eine Session Bean. Im letzteren Fall würden typischerweise alle JEE-Komponenten in ein WAR-File gepackt, was seit JEE 6 möglich ist [4]. Dadurch entfällt die Notwendigkeit, ein EAR-File zu erstellen. Die beiden Möglichkeiten sind in **Abbildung 2** anhand eines auch im weiteren Verlauf des Artikels verwendeten Beispiels zu sehen.

Es ist zu beachten, dass sich Stateful-Session-Beans oder das Zwischenspeichern von Daten in der HTTP-Session über Request-Grenzen hinweg aufgrund der Zustandslosigkeit des REST-API nicht eignen. So wird auch keine Session-ID in der HTTP-Session hinterlegt. Wenn eine Authentifizierung notwendig ist, wird mit jedem Request vom Client ein zuvor ausgehandeltes Authentication Token mitgeliefert. Dieses Token identifiziert den Client und aus ihm werden dessen Zugriffsrechte abgeleitet [5].

Wenn man davon ausgeht, dass die Persistenzschicht in der JEE-Anwendung mithilfe von JPA umgesetzt ist, gibt es bereits eine einfache Art und Weise, die Entitäten einer Persistence-Unit automatisch direkt

über ein REST-API ansprechbar zu machen: JPA-RS [6]. JPA-RS gehört (noch) nicht zum JEE-Standard, sondern zu EclipseLink. Einen konzeptionell ähnlichen, jedoch flexibleren und weiter gehenden Ansatz stellt die im Verlauf des Artikels beschriebene Codegenerierung dar. Aber zuerst einmal werfen wir noch einen Blick auf die Architektur des Android-Clients.

Der Android-Client

Der Android-Client muss für den Datenaustausch mit dem Back-End einige Randbedingungen berücksichtigen:

- Lebenszyklus einer Android-Anwendung (Activity Lifecycle)
- Offline-Fähigkeit
- Häufigkeit und Zeitpunkt der Daten-Synchronisation mit dem Back-End

Je nach Anforderungen an eine Anwendung werden die Punkte „Offline-Fähigkeit“ und „Daten-Synchronisation mit dem Back-End“ unterschiedlich gelöst. Es gibt mindestens vier Synchronisations-Muster, die auch gleichzeitig in ein und derselben Anwendung eingesetzt werden können:

- *Keine Synchronisation*
Daten werden bei Bedarf über das REST-API geholt und lokal nicht gespeichert
- *Synchronisation on Demand*
Der Anwender synchronisiert die Daten explizit, zum Beispiel durch Drücken eines Synchronize-Buttons
- *Synchronisation während der Benutzung*
Dies ist eine Art Lazy-Loading. Daten werden während der Nutzung der Anwendung auf dem mobilen Gerät gespeichert
- *Synchronisation durch Server-Push*
Ein Server informiert Anwendungen mithilfe des asynchronen Messaging-Verfahrens „Google Cloud Messaging for Android“ (GCM) [7] über Veränderungen im Datenbestand auf einem Server. Die Anwendung synchronisiert daraufhin im Hintergrund stillschweigend die lokalen Daten mit den Daten vom Server.

Außer für den Fall, dass überhaupt keine Daten-Synchronisation gefordert ist, werden die Daten lokal – wie auf mobilen Geräten üblich – in einer SQLite-Datenbank gespeichert. Darin werden auch Daten

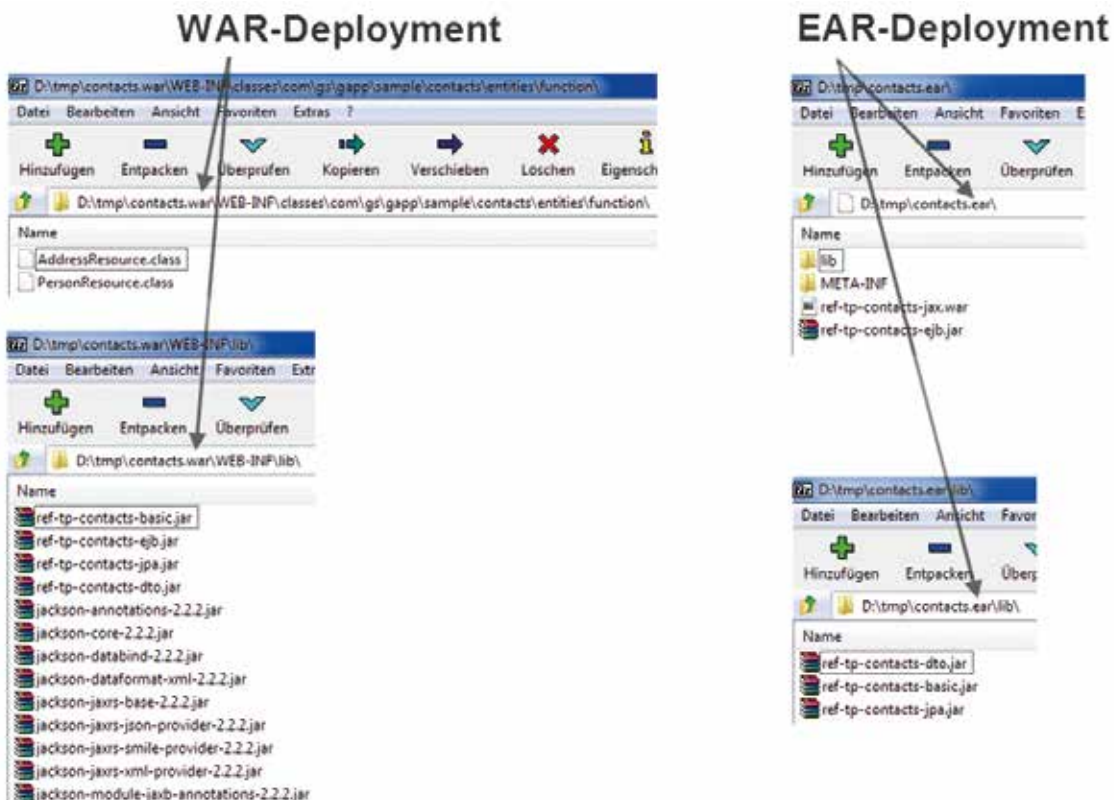


Abbildung 2: WAR- versus EAR-Deployment

aufbewahrt, die vom Benutzer auf dem Gerät neu erfasst oder modifiziert werden. Vom Benutzer gelöschte Daten werden zunächst nicht physisch vom Gerät entfernt, sondern nur als „zu löschen“ markiert. Erst, wenn solche Daten auf dem Server gelöscht worden sind, werden sie auch physisch vom Gerät entfernt.

Zugriffe von der Android-Anwendung auf das Back-End müssen asynchron erfolgen, damit das User-Interface stets gut bedienbar ist. Werden das im Android-API angebotene AsyncTask [8] oder das Loader-Framework [9] dazu verwendet, kann das zur Folge haben, dass gerade in Ausführung befindliche Server-Zugriffe wiederholt ausgeführt werden müssen oder dass die zurückgelieferten Daten im User-Interface nicht oder zumindest nicht sofort angezeigt werden können.

Um nicht in diese Probleme hineinzulaufen, verwenden wir in der Android-Anwendung stattdessen sogenannte „Local Services“. Ein Local Service ist ein Android-Service, der ausschließlich von der Anwendung selbst aufgerufen werden kann. Mit der Open-Source-Bibliothek „RoboSpice“ [10] wird eine starke Vereinfachung der Nutzung von solchen Local Services erreicht.

Textuelle Modellierung

Eine Anwendung, die der bisher beschriebenen Architektur folgt, lässt sich zu großen Teilen aus einem kompakten, einfachen Modell generieren. Zur Modellierung werden textuelle DSLs zur Beschreibung von Datenstrukturen und User-Interfaces eingesetzt. Es genügt im ersten Schritt, die Datenstrukturen zu modellieren. Daraus lassen sich die Persistenz-Schicht, das REST-API, die lokale Datenhaltung auf dem mobilen Gerät (SQLite) und Client-Code zum Zugriff auf das API ableiten. Durch die zusätzliche Modellierung des User-Interface kann eine komplette Anwendung generiert werden. Zur Veranschaulichung ist in Listing 1 beispielhaft ein Modell mit zwei Entitäten gezeigt. Damit soll eine einfache Anwendung zur Verwaltung von Kontakten erstellt werden.

Die Modellierung des User-Interface gerät etwas umfangreicher. Eine detaillierte Beschreibung der User-Interface-DSL würde an dieser Stelle zu weit führen. Wir richten den Blick lediglich auf einen kleinen Ausschnitt des User-Interface-Modells,

```
namespace com.gs.gapp.sample.contacts.entities;
import com.gs.gapp.sample.contacts.ContactsTypeAliases;
module ContactsPersistence kind = Persistence, Basic, Rest;
set TypeFilters = ContactTypeAliases;

enumeration Gender {
    entry MALE;
    entry FEMALE;
}

entity BaseEntity {
    set Storable = false;

    field pk : PrimaryKey {
        set Id = true;
    }
}

entity Person extends BaseEntity {
    field firstName : Text;
    field lastName : Text;
    field gender : Gender;
    field dateOfBirth : Birthday;

    field addresses : Address {
        set CollectionType = List;
        set LazyLoading = true;
        set REST-SerializeToIds = true;
    }
}

entity Address extends BaseEntity {
    field street : Text;
    field houseNumber : Text;
    field zipCode : Text;
    field city : Text;
}
```

Listing 1: Modell der Datenstrukturen

das in Listing 2 zu sehen ist: die Verbindung zwischen der modellierten Anwendung und dem Modell, das die Datenstrukturen des REST-API beschreibt. Hier wird zwischen „local Storage“ und „remote Storage“ unterschieden. „local Storage“ bestimmt die Struktur der SQLite-Datenbank und „remote Storage“ legt fest, wie die Daten über das REST-API zu erwarten sind. In unserem Fall sind die Strukturen identisch, was es uns einfach macht, Code zu generieren, der Daten zwischen REST-API und SQLite-Datenbank austauscht. Würden hier unterschiedliche Datenstrukturen angegeben,

müsste man das Mapping zwischen SQLite-Datenbank und REST-API manuell codieren.

Generisches Modellierungswerkzeug

Das Besondere des zur Modellierung verwendeten Werkzeugs „gApp Modeling Suite“ [11] ist seine einfache und universelle Einsetzbarkeit und Erweiterbarkeit. Sowohl dessen Editor als auch dessen Parser sind generisch implementiert. Alle Modell-Dateien erhalten die Endung „gapp“. Die Modellierung mit einer weiteren DSL wird jeweils durch die Installation eines Eclipse-Plug-ins ermöglicht, ohne einen weiteren

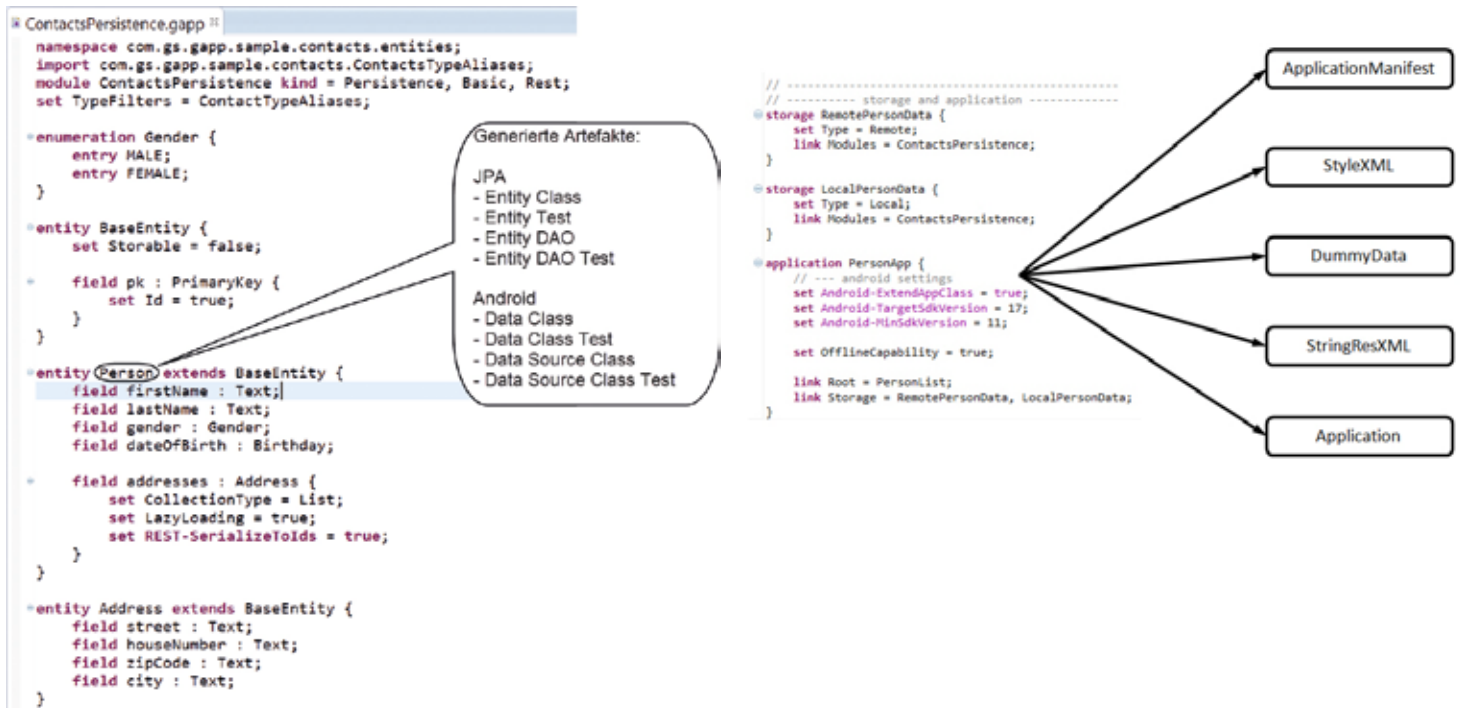


Abbildung 3: Beispiele von Modell-Elementen und dafür generierten Dateien

Editor oder Parser zu installieren. Ein solches Eclipse-Plug-in enthält lediglich eine DSL-Definition. Diese Flexibilität ermöglicht es auch, in ein eigentlich plattformunabhängiges Modell, wie in Listing 2 gezeigt, plattformspezifische Annotationen einzufügen. Listing 3 zeigt beispielhaft ein mit demselben Modellierungswerkzeug erstelltes Modell unter Verwendung der Function DSL zur Modellierung von Komponenten, in denen nach der Generierung Business-Logik von Hand programmiert werden kann. Mithilfe dieser DSL lassen sich unter anderem individuelle REST-Services und EJBs generieren.

Code-Generierung

Die textuellen Modelle für die Datenstrukturen und das User-Interface dienen als Eingabe für den Code-Generator. Dieser ist auf Basis der Virtual-Developer-Plattform entwickelt worden [12]. Er legt die Dateien in mehrere verschiedene Zielprojekte ab. Abbildung 3 zeigt beispielhaft, welche Dateien für welche Modell-Elemente durch den Generator erzeugt werden.

Der Generator selbst steht auf einem entfernten Virtual-Developer-Server als Service bereit. In der IDE werkelt lediglich ein leichtgewichtiges Plug-in, der sogenannte „Virtual Developer Cloud Connector“. Dadurch können auf einem Virtual-Developer-Server bereitgestellte Code-Generatoren die volle

Server-Power nutzen und von beliebigen IDEs aus aufgerufen werden (Eclipse, NetBeans, IntelliJ, Visual Studio, Xcode). Über diesen einen Cloud-Connector sind sämt-

liche auf einem Server verfügbaren Generatoren aufrufbar. Als Nebeneffekt sind für Generatoren eventuelle lokale Installationsprobleme von vornherein ausgeschlossen.

```
// -----
// ----- storage and application -----
storage RemotePersonData {
    set Type = Remote;
    link Modules = ContactsPersistence;
}

storage LocalPersonData {
    set Type = Local;
    link Modules = ContactsPersistence;
}

application PersonApp {
    // --- android settings
    set Android-ExtendAppClass = true;
    set Android-TargetSdkVersion = 17;
    set Android-MinSdkVersion = 11;

    set OfflineCapability = true;

    link Root = PersonList;
    link Storage = RemotePersonData, LocalPersonData;
}
```

Listing 2: Modellausschnitt Datenstrukturen im Client

```

namespace com.gs.gapp.sample.contacts.function;
import com.gs.gapp.sample.contacts.ContactTypeAliases;
import com.gs.gapp.sample.contacts.entities.ContactsPersistence;
module ContactsFunction kind = Function, Basic;
set TypeFilters = ContactTypeAliases;

type BirthdaySearchParameters {
    field gender : Gender;
    field startDate : Birthday;
    field endDate : Birthday;
}

function getPersonsForBirthday {

    in searchParameters : BirthdaySearchParameters;

    out persons : Person {
        set CollectionType = Set;
    }
}
    
```

Listing 3: Modell für Business-Logik-Komponenten

Abbildung 4 zeigt den Zusammenhang zwischen den einzelnen Virtual-Developer-Software-Komponenten.

Ein Code-Generator wird in purem Java gegen das Virtual-Developer-API entwickelt und mittels OSGi modular aufgebaut. Mit der Modularisierung wird das Modell mit den Datenstrukturen nicht direkt in einem einzelnen Schritt in Sourcecode für JAX-RS umgewandelt. Die Generierung führt zuerst mehrere Modell-zu-Modell-Transformationen aus. Als letzten Schritt erzeugt der Generator dann den JAX-RS-Sourcecode. [Abbildung 5](#) zeigt die logische Struktur des gesamten Generators. Diese bekommt ein Anwender des Generators nicht zu sehen. Nur der Hersteller des Generators arbeitet mit solchen Strukturen, über die er Generatoren konfektioniert.

Mithilfe von vom Hersteller anders konfektionierten Generatoren kann man zu einem späteren Zeitpunkt die Datenstruktur des REST-API von der Datenstruktur der Persistenzschicht trennen und trotzdem weiter effektiv Code-Generierung einsetzen. Dieselbe Entkoppelung ist für die Datenhaltung auf dem Client möglich.

Bereitstellung des Back-Ends in der Cloud

In der heutigen Zeit stehen verschiedene Möglichkeiten zur Auswahl, ein JEE-Back-End

für Testzwecke und für den Produktiv-Betrieb mit geringem Aufwand über einen PaaS-Anbieter in der Cloud bereitzustellen. Dazu zählen Jelastic, CloudBees, Heroku, Amazon Beanstalk, Cloud Foundry, OpenShift, Rackspace und CloudControl. Interessant ist auch das Angebot von Engine Yard [13], das in Kürze ebenfalls ein PaaS-Angebot für Java bereitstellen wird. Engine Yard hat seit einem Jahr eine Partnerschaft mit Oracle und es ist beabsichtigt, die PaaS-Angebote der beiden Firmen miteinander zu verknüpfen.

Beispielhaft sei hier das Vorgehen für die Bereitstellung des Back-Ends mit dem PaaS-Dienst Jelastic skizziert [14]. Eine der Besonderheiten an Jelastic ist, dass es nicht ein Betreiber von eigenen Rechenzentren ist. Stattdessen handelt es sich hierbei um einen Software, die vom gleichnamigen Unternehmen mit Sitz in Palo Alto entwickelt wird. Unabhängige Internet-Service-Provider nutzen diese Software, um damit eine PaaS anbieten zu können. Dies ermöglicht es, sich selbst auszusuchen, in welchem Land sich das Rechenzentrum befinden soll, bei dem man seine Anwendungen betreibt. Die zweite Besonderheit ist, dass man sich nicht einfach einen dedizierten oder virtuellen Server mietet, sondern verbrauchsabhängige Kosten anfallen (CPU, Memory, Disk Space, Traffic). Bis jetzt werden Java, PHP und Ruby als Plattformen angeboten. In Deutschland bietet die dogado Internet GmbH den Jelastic-Service an.

Mit Jelastic kann man ein WAR- oder EAR-File direkt in einen zuvor mit wenigen Klicks bereitgestellten GlassFish JEE Application Server deployen. Dies geht entweder über die webbasierte Administrations-Oberfläche oder über ein Jelastic Plug-in, das es für Eclipse, IDEA, Maven und NetBeans gibt. [Abbildung 6](#) zeigt, wie die webbasierte Administrations-Oberfläche aussieht und an welcher Stelle das zuvor hochgeladene EAR-File in den Applikationsserver deploy wird. Mit dem Jelastic-IDE-Plug-in geht es noch schneller, da Hochladen und Deployment als eine zusammenhängende Aktion durchgeführt wird.

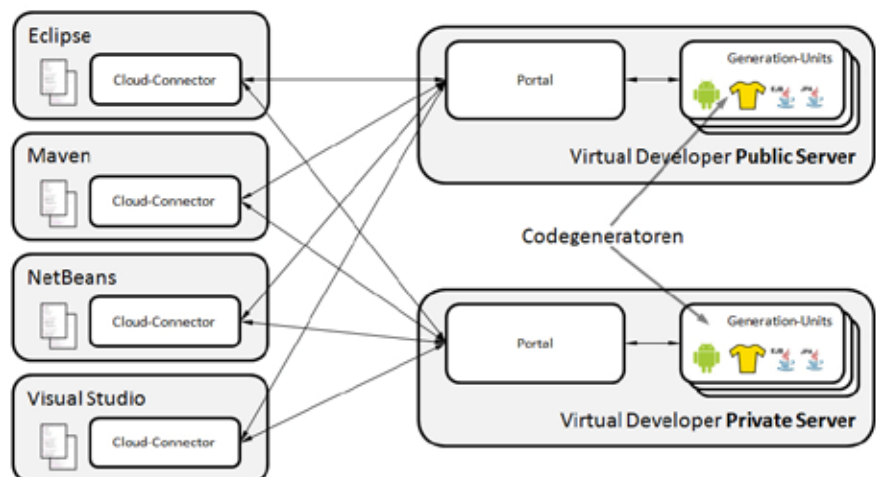


Abbildung 4: Virtual-Developer-Software-Komponenten

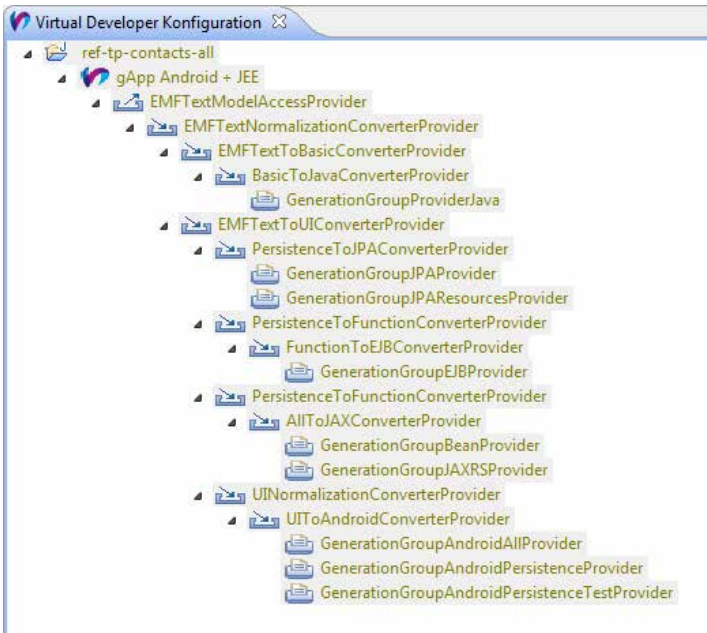


Abbildung 5: Struktur des Code-Generators

An dieser Stelle sei auch darauf hingewiesen, dass die Nutzung eines PaaS-Anbieters nicht für jeden Fall die optimale Lösung darstellt. Zu beachten ist beispielsweise, dass über eine PaaS nicht immer die Versionen von Software-Komponenten angeboten werden, die man für die eigene Anwendung benötigt. Letztendlich verliert man durch den Einsatz von PaaS-Lösungen bis zu einem gewissen Grad die Kontrolle über die Umgebung, auf der die eigene Anwendung laufen soll. Es muss jeder für sich entscheiden, ob dieser Nachteil durch die vielen Vorteile aufgewogen wird.

Fazit

Mithilfe von JEE, leichtgewichtiger Modellierung und Code-Generierung sowie der Nutzung von PaaS-Diensten für Java ist es heutzutage möglich, native mobile Client-Server-Anwendungen in kürzester Zeit zu entwerfen, zu entwickeln, zu testen und in Produktion zu nehmen. Um diese Agilität zu erreichen, müssen keine Kompromisse bei der Qualität der Software-Architektur

und des Quellcodes eingegangen werden. Zudem eröffnet die Verwendung von plattformunabhängigen Modellen die große Chance, kosteneffizient native mobile Anwendungen für mehrere Plattformen liefern zu können, und zwar ohne dass die verschiedenen Implementierungen, was den Funktionsumfang angeht, divergieren.

Links

- [1] Mohito: <http://www.mohito-projekt.de>
- [2] REST Constraints: http://whatisrest.com/rest_constraints/index

- [3] JAX-RS: <http://docs.oracle.com/javaee/6/tutorial/doc/giepu.html>
- [4] WAR vs. EAR: <http://www.munzandmore.com/2012/ora/ejb-31-stateless-session-bean-dependency-injection-asynchronous-methods>
- [5] Authentication: <http://vinaysahni.com/best-practices-for-a-pragmatic-restful-api#authentication>
- [6] JPA-RS: <http://wiki.eclipse.org/EclipseLink/Development/2.4.0/JPA-RS>
- [7] GCM: <http://developer.android.com/google/gcm>
- [8] AsyncTask: <http://developer.android.com/reference/android/os/AsyncTask.html>
- [9] Loaders: <http://developer.android.com/guide/components/loaders.html>
- [10] RoboSpice: <http://github.com/octo-online/robo-spice>
- [11] gApp: <http://generative-software.de/gapp>
- [12] Virtual Developer: <http://virtual-developer.com>
- [13] Engine Yard: <http://engineyard.com>
- [14] Jelastic: <http://jelastic.com>

Marcus Munzert

marcus.munzert@generative-software.com



Marcus Munzert ist geschäftsführender Gesellschafter der 2007 gegründeten Generative Software GmbH. Er entwickelt seit 1998 mit Java und setzt seit 2002 mit Begeisterung Methoden der modellgetriebenen Software-Entwicklung ein.

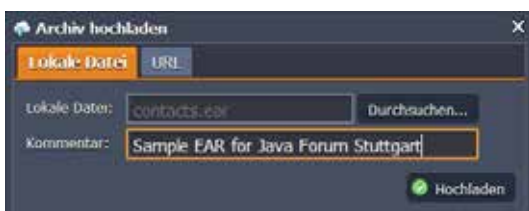


Abbildung 6: Jelastic-Administrations-Oberfläche

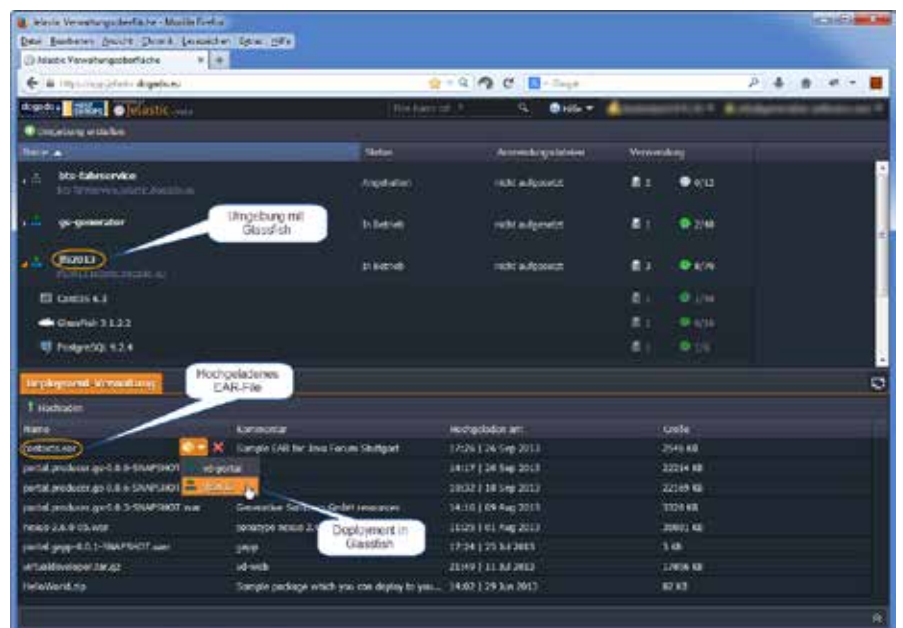


Abbildung 7: Jelastic-Administrations-Oberfläche

Enterprise-2.0-Portale mit Grails – geht das?

Manuel Breitfeld und Tobias Kraft, exensio GmbH

Das Grails-Framework wurde in der Enterprise-Java-Community lange Zeit nur als Tool für kleine Projekte angesehen. Dieser Artikel zeigt, dass selbst große und komplexe Projekte damit möglich sind.

Portale sind mit einem Flugzeug-Cockpit vergleichbar und bilden eine aggregierte Sicht auf eine Vielzahl unterschiedlicher Informationstöpfe. Neben Produkten von IBM und Oracle ist vor allem Microsoft Sharepoint im Markt allgegenwärtig. Im Open-Source-Bereich ist Liferay am bekanntesten.

In ihren Projekten haben die Autoren jedoch immer wieder festgestellt, dass diese Portal-Frameworks außerordentlich komplex sind und enorm viel Zeit für das Anpassen der mächtigen Funktionsvielfalt erforderlich war. Aus diesem Grund sind sie anders vorgegangen. Ihr Portal-Framework verfügt über solide Basis-Funktionen wie die Integration von Backend-Systemen, Enterprise Search, CMS, DMS, Wiki, Blog etc. Allerdings sind sie vom Funktionsumfang her nicht mit denen der großen Hersteller vergleichbar. Die Basis wird durch das Grails-Framework in den Kunden-Projekten erweitert mit dem Vorteil, dass ressourcenschonende, benutzerfreundliche und performante Lösungen entstehen.

Portale

Die Hauptkomponenten eines Portals sind Portlets, von denen mehrere auf einer Portal-Seite platziert werden können. Ein Portlet selbst kann auch mehrfach auf Seiten eingesetzt sein. Dies erfordert, dass Portlets parametrierbar sind und unabhängig voneinander funktionieren. Weitere Kern-Anforderungen sind neben den eingangs schon erwähnten Basis-Funktionalitäten die Personalisierung und das Single Sign-on.

Im Java-Umfeld gibt es mit den JSRs 168 und 286 zwei verabschiedete Standards, die hauptsächlich die Schnittstellen zwischen Portalen und Portlets sowie das Portlet-Programmier-Modell definieren. Der OASIS-Standard „Web Services für Re-

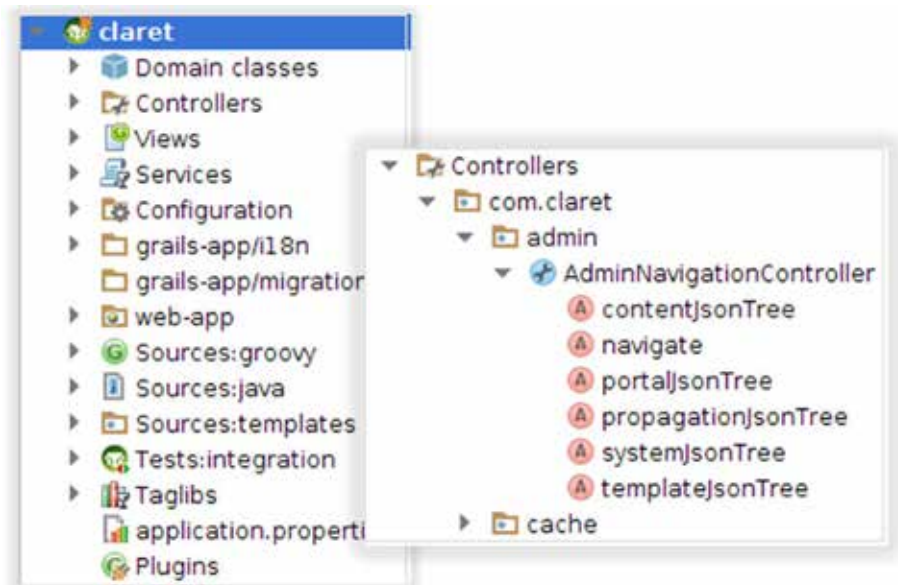


Abbildung 1

„mote Portlets“ (WSRP) definiert Standards zur unabhängigen, Portal-übergreifenden Bereitstellung von Portlets. Im Bereich der Interoperabilität zwischen verschiedenen Portalen, also dem Zugriff auf Portlets eines anderen Portal-Containers, wurde leider die Erfahrung gemacht, dass dies in den meisten Fällen trotz der Standardisierung nicht funktioniert.

Portalen eilt, nicht ganz zu Unrecht, der Ruf voraus, schwergewichtig zu sein. Dies spiegelt sich auch im Entwicklungszyklus wider, angefangen bei der Installation der Entwicklungsumgebung bis zu den langen Startzeiten des Servers.

Grails als Basis-Technologie

Das Open-Source-Projekt Grails startete bereits im Jahre 2005 und liegt mittlerweile in der Version 2.3 vor. Grails wurde wesentlich inspiriert durch Ruby on Rails und hat das Ziel, die Erstellung von Web-Appli-

kationen auf der Java-Plattform zu vereinfachen. Als Basis dienen bewährte Technologien aus dem Java-Umfeld wie Hibernate und Spring. Dies vereinfacht den Einstieg für Java-Entwickler in die Grails-Welt.

Wesentliche Paradigmen für die Entwicklung sind die beiden Prinzipien „Don't repeat yourself“ (DRY) und „Convention over Configuration“ (CoC). Dies erspart jede Menge Glue Code und der Umsetzungsweg ist vordefiniert. Es gilt jedoch: Der Standard ist vorkonfiguriert, bei Bedarf kann aber davon abgewichen werden, sodass die Flexibilität nicht verloren geht.

Beispielsweise ist in jeder Klasse das Logging verfügbar, ohne dass eine Variable instanziiert werden muss. Es lässt sich einfach über den Befehl `log.info()` aufrufen. Auch der in [Abbildung 1](#) dargestellte Aufbau eines Grails-Projekts spiegelt das CoC-Prinzip wider. Für jede Schicht wie Controller, Views oder auch Services gibt es ein

```
def names = ["Spring", "Grails", "Hibernate", "Groovy",
"JEE"]

def shortNames = names.findAll { it.size() <= 6 }
assert shortNames == ["Spring", "Grails", "Groovy", "JEE"]

def namesWithG = names.grep( ~ /.*(?i)g./ ).sort()
assert namesWithG == ["Grails", "Groovy", "Spring"]
```

Listing 1

eigenes Verzeichnis, in dem die Dateien abgelegt sind. Ein weiteres Beispiel sind Controller-Klassen, die alle mit dem gleichnamigen Wort enden. Die Actions des in [Abbildung 1](#) aufgeführten Controllers sind in einer View beispielsweise einfach über „adminNavigation.templateJsonTree()“ ansprechbar.

Schnelles Entwickeln mit Grails

Groovy ist eine nicht typisierte, auf der JVM aufbauende Sprache und stellt die Basis von Grails dar. Mit Groovy können Anforderungen oftmals einfacher und mit weniger Code ausgedrückt werden als in Java. Insbesondere die Closures, das Pendant zu den mit Java 8 eingeführten Lambda-Ausdrücken, sind sehr mächtig. [Listing 1](#) vermittelt einen kurzen Eindruck anhand von Beispielen für die Verarbeitungen von Collections.

Schnell erfolgt in Grails der Start nach dem Motto „Runterladen und loslegen“. Außer einer vorinstallierten JVM wird nichts weiter benötigt; es kann nach dem Entpacken der Grails-Distribution gestartet werden. Mithilfe von Scaffolding, bei dem aus Domain-Klassen die Views und Controller generiert werden, lassen sich schnell erste Ergebnisse erzielen, um diese mit Kunden besprechen zu können. Mit Scaffolding generierte Web-Applikationen unterstützen die kompletten CRUD-Funktionalitäten.

Grails besitzt ein Plug-in-System, mit dem zusätzliche Erweiterungen verfügbar gemacht werden. Aktuell stehen mehr als 500 Plug-ins bereit, die Unterstützung für diverse Bereiche wie nicht funktionale Anforderungen (wie Security) oder Oberflächen-Erweiterungen bieten. Dies spart beim Entwickeln eine Menge Zeit, und selbst wenn ein Plug-in nicht alle Anforde-

rungen unterstützt, kann es zumindest als Basis für eigene Erweiterungen dienen.

Als letzter Aspekt zum Aufzeigen der einfachen Entwicklung wird die Datenbank-Zugriffsschicht GORM herangezogen. Der Standard ist in Grails Hibernate, allerdings existieren auch Plug-ins, um NoSQL-Datenbanken wie MongoDB über GORM anzubinden. Der Zugriff erfolgt direkt über die Domain-Klassen ([siehe Listing 2](#)). Insbesondere die aufgeführten Dynamic-Finder sind ein sehr komfortabler Mechanismus für Abfragen. Hier kann dynamisch ohne weitere Konfiguration oder Programmierung auf Attribute eine Domain-Klasse oder auch Beziehung gefiltert werden.

Integration mit Groovy und Grails

Eine wesentliche Anforderung an Portale ist die Integration von Dritt-Systemen. Da die bereitgestellten Schnittstellen der einzubindenden Systeme stark variieren können, ist hier eine entsprechende Flexibilität gefordert.

In vielen Fällen erfolgt die Integration über ein abgestimmtes Datenaustausch-Format wie CSV-Dateien, XML oder JSON. Groovy bietet hier von Haus aus Funktio-

nalitäten für eine unkomplizierte Verarbeitung dieser Formate. [Listing 3](#) zeigt, wie mit wenigen Zeilen Code ein JSON-String eingelesen und weiterverarbeitet werden kann.

Weitere Szenarien für einen direkten Zugriff auf Drittsysteme sind die Integration über folgende Wege:

- Java-Schnittstellen (EJB)
- SAP-Integration (SAP JCo)
- Datenbanken

Durch die enge Verzahnung mit Java lassen sich solche Anwendungsfälle über die JEE-Standards oder durch Einbinden entsprechender Dritt-Bibliotheken realisieren. Liegen komplexere Integrations-Szenarien vor, bietet sich der Einsatz eines Integration Frameworks wie Apache Camel oder Spring Integration an. Für Apache Camel gibt es, wie in Grails üblich, ein Plug-in und durch die starke Assoziation von Grails zu Spring lässt sich auch Spring Integration einbinden.

Grails für Portale

Welche Gründe sprechen nun für den Einsatz des Grails-Frameworks als Basis zur Umsetzung von Enterprise-Portalen? Neue Anforderungen können dank Groovy und der Grails-Prinzipien schnell implementiert werden. Insbesondere bei der Integration von Drittsystemen kann Groovy seine Stärken in Punkto Einfachheit ausspielen.

Die Fähigkeit zur Umsetzung von Enterprise-Applikationen hat Grails bei den Autoren schon in einigen Groß-Projekten bewiesen. Hier wurden Größenordnungen von 200 Domain-Klassen und Datenbanken mit bis zu 20 GB realisiert. Auch bei den nichtfunktionalen Anforderungen für

```
// CRUD Operationen
Portlet.delete()
Portlet.save(flush:true)
// dynamic finder
Portlet.findAllByPageAndPortal('Home', portalname)
// where query
PortletInst.where{
    year(published) == 2013 && right != 'public'
}
```

Listing 2

Enterprise-Applikationen wie Verfügbarkeit, Ausfallsicherheit oder Security-Aspekten kann Grails durch die Nähe zu Java seine Stärken ausspielen.

Ein komplexes Unterfangen ist es, Portale zu testen. Die Gründe sind das Einbinden von Drittsystemen oder auch die Platzierung mehrerer Portlets auf einer Seite, die gegebenenfalls noch personalisiert sind. In Grails werden mit Unit-, Integrations- und funktionalen Tests die bekannten Test-Typen unterstützt. Standardmäßig ist JUnit als Test-Framework konfiguriert, allerdings wird über ein Plug-in auch das Spock-Framework [1] unterstützt. Mit Spock werden Testfälle mit einer DSL (Domain Specific Language) nach dem „Gegeben Wenn Dann Prinzip“ definiert. Ein Test-Fall mit Spock ist in Listing 4 dargestellt, der unter anderem zeigt, wie einfach mehrere Test-Durchläufe über die Parameter-Tabelle möglich sind.

Für funktionale Tests bietet sich Geb [2] an, das wiederum auch über den Plug-in-Mechanismus installiert wird. Es setzt auf den Selenium-Treibern auf, wodurch eine breite Browser-Unterstützung gewährleistet ist. In einer an JQuery angelehnten Notation können einfach Elemente selektiert und Events ausgelöst werden.

Elementar ist in der heutigen Zeit über Continuous Integration das automatisierte und regelmäßige Durchführen der Tests. Mit Jenkins und Hudson ist der Aufbau einer entsprechenden Infrastruktur für Grails problemlos möglich und es sind auch entsprechende Werkzeuge zum Messen der Code-Qualität und der Test-Abdeckung vorhanden.

Umsetzung von Portal-Modulen

In ihrem Portal haben die Autoren bereits viele Module umgesetzt, die eine solide Grundlage für den Einsatz in Kunden-Projekten bieten. Neben Basis-Funktionalitäten wie einem übergreifenden Tagging, einer Enterprise-Suche, einem CMS und einem DMS gehören mittlerweile auch diverse Web-2.0-Funktionalitäten zu einem Enterprise-2.0-Portal. Entscheidend ist bei allen Modulen die Integration, die so nahtlos wie möglich sein sollte. Nachfolgend sind die umgesetzten Portal-Module im Detail vorgestellt.

Tagging: Die Verschlagwortung von Inhalten, das sogenannte „Tagging“, ist mitt-

```
import groovy.json.JsonSlurper
def jsonText = '''{
  "message": {
    "header": { "from": "bud@spencer.com" },
    "body": "Hello Java Aktuell."
  }
}'''

def mail = new JsonSlurper().parseText(jsonText)
assert mail.message.body == "Hello Java Aktuell."
assert mail.message.header.from.size() > 0
```

Listing 3

```
import spock.lang.Specification
class PortalServiceSpecification extends Specification {

  def "Return total number of pages"() {
    setup: 'Create PortalService instance with pages'
    PortalService pService = new PortalService(pageList)
    expect: 'count pages'
    expectedCount == pService.count()
    where:
      pageList      | expectedCount
      []             | 0
      ['Home', 'Search'] | 2
      ['Home']      | 1
  }
}
```

Listing 4

ID	Tag ID	Entity
1	6	com.claret.MicroMessage-10
2	5	com.claret.MicroMessage-10
3	6	com.claret.BlogPost-1
4	2	com.claret.BlogPost-3
5	3	com.claret.Wiki-2

Tabelle 1

lerweile im Intra- und Internet nicht mehr wegzudenken. Es gibt kaum eine Plattform, bei der der Nutzer nicht anhand von Tags suchen beziehungsweise der Inhaltsanbieter nicht Elemente mithilfe von Tags kategorisieren kann.

Tags spielen in Portalen eine große Rolle, denn alle populären Inhaltstypen innerhalb Content-getriebener Portale werden typischerweise mit Schlagworten

versehen: Sämtliche Artefakte eines CMS/ DMS wie beispielsweise News, Artikel und Dokumente werden Tags zugeordnet (siehe Abbildung 2). Auch und vor allem im Bereich der Web-2.0-Funktionalitäten existieren diverse Inhalte, die verschlagwortet werden. Dies sind unter anderem Blog- und Wiki- Beiträge sowie Kurznachrichten à la Twitter. Darüber hinaus sind Anwendungsfälle wie „Expert Finder“, „People Fin-

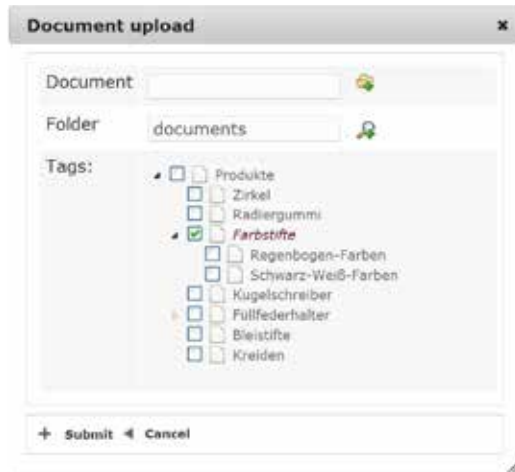


Abbildung 2

der“ oder andere Netzwerke prädestiniert, um Personen zu taggen.

Das von den Autoren auf Basis des Taggable-Plug-ins [3] entwickelte Tagging-Modul kommt ohne direkte Abhängigkeit im Datenmodell aus, indem es in einer separaten Datenbank-Tabelle die Beziehung zwischen Tag und Entität (codiert über den absoluten Pfad inklusive ID) verwaltet (siehe Tabelle 1). Dabei bietet es die folgenden Funktionalitäten:

- **Funktionsübergreifendes Tagging**
Dieselben Tags können verschiedenen Entitäten zugewiesen werden
- **Hierarchie**
Wie [Abbildung 2](#) zeigt, lassen sich Tags hierarchisch aufbauen



Abbildung 3

- **Gültigkeitsbereiche**
Für unterschiedliche Entitäten können Sub-Bäume aus der Tag-Hierarchie ausgewählt werden – eine Art Kategorisierung. Dies ist besonders dann sinnvoll, wenn sich der Blog mit anderen Themen als die News beschäftigt. Der Ablauf des Tagging ist gleich, die Tags an sich sind verschieden.

Enterprise-Suche: Die Suche ist ebenso allgegenwärtig in Applikationen wie das Tagging. In einem Portal nimmt die Suche einen besonderen Stellenwert ein – sie stellt Ergebnisse aus unterschiedlichen fachlichen Komponenten dar. So kann die Suche nach „Grails“ in unserem Portal beispielsweise unterschiedlichste Ergebnistypen liefern: Personen, die sich damit auskennen, oder Dokumente, News, Blog- sowie Wiki-Beiträge, bei denen das Wort im Text oder als Tag vorkommt. Die Liste ist beliebig fortsetzbar.

Die von den Autoren implementierte Enterprise-Suche basiert auf Apache Solr und Elasticsearch (beide Varianten sind verfügbar) und setzt auf dem Solr- [4] beziehungsweise Elasticsearch-Plug-in [5] auf. Dabei verwenden sie den Funktionsumfang, den die entsprechenden Such-Server bieten, und können dadurch unter anderem Highlighting, Autocomplete, Fuzzy Search und Sortierung anbieten. Die Ergebnisse sind, abhängig von ihrem Typ, unterschiedlich dargestellt: Für eine Person können ein Profilbild und Kontaktdaten angezeigt werden, bei Dokumenten bietet sich die Anzeige einer Vorschau an.

Content/Document Management System: Typischerweise wird in jedem Portal eine Art editierbarer Inhalt dargestellt. Das in diesem Portal enthaltene Content/Document Management System (CMS/DMS) unterstützt dabei sowohl unstrukturierten Inhalt auf Basis eines WYSIWYG-Editors als auch strukturierten und auf Templates basierenden Inhalt. Neben dieser Kernfunktion sind folgende Funktionen enthalten:

- **Context Navigation**
Neben der Portal-Navigation ist es bei umfangreichen Inhalten sinnvoll, innerhalb des dargestellten Inhalts navigieren zu können. Typischerweise stellt dabei ein Portlet die Hauptseite eines

Kapitels als Inhalt dar und diese Seite verfügt beispielsweise im linken Bereich über eine eigene, hierarchische Navigation, um auf die zugehörigen Unterseiten zu verweisen.

- **Workflow**
Ein dem Vier-Augen-Prinzip folgender Freigabe-Mechanismus stellt sicher, dass Inhalte erst freigegeben werden müssen, bevor sie im Portal dargestellt werden.
- **Versionierung**
Änderungen werden versioniert. Wurde fälschlicherweise Inhalt freigegeben, lässt sich eine frühere Revision wiederherstellen.
- **Inline-Editing**
Um es dem Editor einfach zu ermöglichen, gewisse Inhalte zu ändern, kann der Inhalt direkt dort bearbeitet werden, wo er auch im Portal dargestellt ist. Das mitunter mühsame Suchen in der separaten CMS-Struktur entfällt damit.

Die Entwicklung dieser Funktionen greift nicht auf bereits existierende Plug-ins oder Content-Management-Systeme zurück, sondern ist eine Eigenentwicklung. Die Integration von etablierten Systemen wie Adobe CQ5 oder TYPO3 ist dennoch möglich und in Kundenprojekten schon erfolgreich durchgeführt worden.

Web-2.0-Funktionalitäten: Da auch im Enterprise-Umfeld der Einsatz von Web-2.0-Funktionalitäten immer stärker zunimmt, bietet das Portal einige entsprechende Module. Die folgende Liste gibt einen Überblick:

- Blog
- Wiki
- Kurznachrichten
- Activity Stream
- Teilen, Kommentieren und Bewerten (Voting) von Inhalten
- Integration anderer sozialer Dienste wie Twitter

Technisch bauen die meisten dieser Module auf die bereits vorgestellten Funktionalitäten auf. Die Grundlage des Activity Streams (siehe [Abbildung 3](#)) ist beispielsweise die Enterprise-Suche: Da die Suche alle Inhalte des Portals kennt, kann sie auch zurückmelden, wer zuletzt einen Kommen-

tar geschrieben, eine Bewertung abgegeben oder eine Kurznachrichte an den aktuell eingeloggtten Benutzer gerichtet hat – die perfekte Datenbasis für den Activity Stream.

Personalisierung und Berechtigungen:

Eine Hauptfunktionalität von Portalen ist die personalisierte Sicht auf Inhalte je nach Benutzer. In Grails bietet Spring Security dafür eine gute Grundlage. Die Implementierung eines hierarchischen Rollenkonzepts und die Anbindung an Datenquellen wie Active Directory, Datenbank oder CMS bietet im Portal folgende Möglichkeiten, auf Benutzerrollen zugeschnittene Inhalte zu liefern:

- *Enterprise-Suche*
Die Suche liefert nur Resultate, die für den Benutzer bestimmt sind. Ist der Benutzer einer Rolle zugeordnet, die bestimmte Inhalte des CMS nicht sehen darf, werden diese auch von der Suche nicht angezeigt.
- *Portalseiten und Portlets*
Benutzergruppen können Portalseiten und Portlets zugewiesen werden, sodass diese dem Benutzer nicht angezeigt werden. Das kann aufgrund von Personalisierung eingestellt werden (der Benutzer braucht diese Inhalte nicht oder das Portal wird übersichtlicher) oder wegen eines klassischen Zugriffsschutzes (der Benutzer darf diese Inhalte nicht sehen).
- *Inhalt und Context-Navigation*
Inhalte aus dem CMS unterliegen Berechtigungsregeln und werden einem Benutzer abhängig von der Rolle ausgegeben. Entsprechend der Berechtigungen können Benutzer daher auch unterschiedliche Einträge innerhalb der Context-Navigation vorfinden.

Das Portal anpassen

Die Anpassungen an einem Portal teilen sich in zwei Bereiche: Anpassungen, die zur Laufzeit gemacht werden, und solche, die ein neues Deployment benötigen. In diesem Portal erfolgt die Definition von Navigation und Portalseiten zur Laufzeit. Ein Administrator kann also Portalseiten neu anlegen, diese entsprechend berechnen und Portlets hinzufügen. Das Portal bietet dafür Templates, die ein gewisses Layout vorgeben.

Aus dem Fundus an Portlets lässt sich eine neue Funktionalität zur Laufzeit in das Portal einbringen. Dabei bietet dieses Portal die Methode der „Portlet Preferences“ an. Somit kann eine Portlet-Instanz Eigenschaften erhalten – ein Portlet kann sich also je nach Konfiguration auf unterschiedlichen Portalseiten anders verhalten. So ist es beispielsweise möglich, dem Activity Stream Portlet über die Eigenschaft „hits“ vorzugeben, wie viele Einträge angezeigt werden sollen.

Die Umsetzung neuer Anforderungen fällt in die Kategorie der Änderungen, die ein neues Deployment erfordern. Dies ist denkbar einfach – jeder Grails-Controller ist dank des Portal-Frameworks automatisch ein Portlet. Es sind also lediglich folgende Schritte nötig:

1. Funktionalität in Grails umsetzen (Controller, View, Service etc.)
2. Neues Portlet im Admin-Backend definieren (zur Laufzeit)
3. Portlet auf die gewünschte Seite ziehen (zur Laufzeit)

Der bequeme Umstand, dass jeder Controller gleichzeitig ein Portlet ist, führt auch dazu, dass bereits in anderen Projekten entwickelte Funktionalitäten ohne Modifikation im Portal als Portlet eingesetzt werden können.

Fazit

In der Praxis hat sich gezeigt, dass die Entwicklung mit Grails schnell ist und auch Entwickler ohne Grails-Kenntnisse sich schnell zurechtfinden. Dabei ist sowohl Grails an sich als auch das von uns entwickelte Portal Enterprise-fähig und in der Praxis in mehreren Kundenprojekten erprobt. Tatsächlich setzen die Autoren Grails mittlerweile in fast 80 Prozent ihrer Projekte ein.

Das Portal-Framework hat die Komplexität enorm reduziert. Das Framework bietet die Kern-Funktionalitäten eines Portals „out of the box“. Individuelle Kundenwünsche können im Anschluss schnell entwickelt und als Portlet im Portal eingebunden werden.

Referenzen

- [1] <http://docs.spockframework.org/en/latest>
- [2] <http://www.gebish.org>
- [3] <http://grails.org/plugin/taggable>
- [4] <http://www.grails.org/plugin/solr>
- [5] <http://grails.org/plugin/elasticsearch>

Manuel Breitfeld
manuel.breitfeld@exensio.de



Manuel Breitfeld ist nach seinem Studium der Softwaretechnik seit dem Jahr 2010 als Consultant für die exensio GmbH tätig. Sein Tätigkeitsfeld umfasst sowohl Entwicklungsprojekte mit Enterprise-Portalen und Web-Applikationen als auch Beratungsprojekte im Usability- und Intranet-Bereich.

Tobias Kraft
tobias.kraft@exensio.de



Tobias Kraft war nach seinem Studium des Wirtschaftsingenieurwesens in mehreren Software- und IT-Beratungshäusern als Consultant und Software-Architekt tätig (darunter sd&m – heute Cap Gemini). Seit dem Jahr 2009 beschäftigt er sich bei der exensio GmbH mit der Architektur und Umsetzung von Enterprise-Portalen sowie Web-Applikationen basierend auf Java-Technologien und dem Grails-Framework.

NetBeans IDE 7.4 ab sofort verfügbar

Oracle gibt die Verfügbarkeit von NetBeans Integrated Development Environment (IDE) 7.4 bekannt. Sie unterstützt die Entwicklung von HTML5- und JavaScript-Benutzeroberflächen on-device. Für die Anwendungsentwicklung für Java EE und PHP stehen fortschrittliche HTML5-Entwicklungs-Features zur Verfügung. NetBeans IDE 7.4 steht unter <https://netbeans.org> zum Download zur Verfügung.

Wo und warum der Einsatz von JavaFX sinnvoll ist

Björn Müller, CaptainCasa GmbH

Die CaptainCasa-Community hat gerade ihr Rich-Client-Framework „Enterprise Client“ von Java Swing auf JavaFX umgestellt. Natürlich spielte die Frage, ob JavaFX das richtige Framework ist oder ob nicht doch eine HTML5-basierte Technologie vorzuziehen sei, in den Diskussionen vor der Umstellung eine wichtige Rolle. Der Artikel gibt die Kernüberlegungen wieder, warum die Entscheidung für JavaFX ausfiel.

JavaFX ist da. Inhaltlich gesehen ist es ab Release 2.0 „gelingen“, im aktuellen Release 2.2 „absolut brauchbar“ und seit dem ganz offiziellen Release 3.0 „strategisch verwendbar“. JavaFX ist ein neues, besseres Swing. Mit vielen neuen Möglichkeiten im Bereich der grafischen Gestaltung von Oberflächen (CSS Styling, Animationen, Multitouch etc.) und mit einer einfach zugänglichen Art, diese mit Java als Programmiersprache zu entwickeln (Scene Graph etc.).

JavaFX ist eine Rendering-Umgebung. Nicht weniger, aber auch nicht mehr. Wie man diese Umgebung nun konkret beispielsweise an eine Anwendungslogik anschließt, ist mit JavaFX offen. Das ist auch gut so, da die Szenarien, wie es unterhalb der Oberfläche aussieht, durchaus mannigfaltig sind.

JavaFX steht auf den gängigen Desktop-Betriebssystemen zur Verfügung (Windows, Linux, MacOS) und macht derzeit deutliche Schritte in Richtung „iOS“ und „Android“. Der Traum eines Java-UI-Entwicklers, eine UI-Technologie für die verschiedenen gängigen Devices verwenden zu können, rückt zunehmend in die Nähe – Aussagen der Java-Community hinsichtlich offizieller Verfügbarkeit gibt es allerdings derzeit leider noch nicht. Es existiert auch noch die Verfügbarkeit von JavaFX auf Geräten à la Raspberry Pi. Das ist sicherlich hochinteressant, aber für die meisten noch ein spezieller Markt.

Positionierung gegenüber HTML5

Es stellt sich nun ganz konkret die Frage, warum man denn nun ein gewisses Vorhaben unbedingt in JavaFX umsetzen möchte und nicht in einer HTML5-basierten Technologie. HTML5 ist ganz klar das, was man von wissenden oder unwissenden Managern,

Architekten oder wem auch immer als bitte zu übernehmende Oberflächen-Architektur serviert bekommt. Zugegebenermaßen nicht ohne Grund, hinter HTML5 steht ja durchaus eine Reihe von sinnvollen Aussagen wie „Zero Installation“ oder „Läuft auf allen Browsern und Devices“.

Mit HTML5 lässt sich alles machen, was an Oberflächen denkbar ist – die technischen Grenzen sind nach oben weit offen. Vorsichtig sein sollte man mit Aussagen wie „Das geht nicht mit HTML“. Durch einfache Google-Recherche wird jeder Chef irgendwelche Artikel finden, in denen irgendjemand beschreibt, wie man es doch hinbekommt. Es gibt also keine eindeutige Linie, bei der man sagen kann: „Bis dahin HTML5, darüber hinaus nativere Technologien wie JavaFX“. Zudem gibt es ganz klar eine Menge Szenarien, in denen nur HTML5 Sinn ergibt:

- Anwendungen, die sich an anonyme Endbenutzer richten. Hier sticht ganz klar der „Zero-Deployment“-Gedanke alles andere aus. Ein Shop, der nicht direkt im Browser da ist, wird nicht genutzt werden.
- Anwendungen, in denen das HTML-Design eine große Rolle spielt, wenn der Entwicklungsprozess also darauf beruht, dass HTML-Designer die Seiten gestalten, um sie danach mit Code anzureichern – so schrecklich die Ergebnisse am Ende teilweise sind. Auch hier wird es schwerfallen, überzeugend mit JavaFX zu argumentieren.
- Wenn in einem Szenario nur sehr simple Oberflächen vorkommen, dann will auch hier der Benutzer keine Extra-Installation am Client vornehmen müssen, um diese Oberflächen zum Laufen zu bringen.

Langfristigkeit von Vorhaben

Die Laufzeit von Anwendungen kann sehr unterschiedlich sein. Entsprechend wichtig ist die Wahl der verwendeten UI-Technologie. Das Browser-/HTML5-Technologie-Umfeld ist geprägt durch eine Unmenge an Frameworks einerseits und durch regelmäßig wechselnde Hypes andererseits.

Die Ursache hierfür ist, dass auf Basis von HTML5 und JavaScript eine effiziente Erstellung von Oberflächen nur schwer möglich ist. Folglich braucht es abstrahierende Frameworks, die die große Komplexität auf der einen und konkrete Unzulänglichkeiten wie Browser-Kompatibilitätsprobleme auf der anderen Seite verbergen. Die Abstraktionen reichen von simplen HTML5-Template-Ansätzen über mächtige JavaScript-Bibliotheken bis hin zu Google-Web-Toolkit-Ansätzen (GWT), in denen JavaScript nur noch das Kompilat einer Java-Entwicklung ist.

Viele Anwendungen sind sehr langfristig getaktet. Angenommen, man entwickelt eine Anwendung im Bereich der Logistik zur Verwaltung von Lagerbeständen. Startet man heute mit der Entwicklung, beginnt in ein bis zwei Jahren das Rollout an einige Kunden. Der Vertrieb wird das Produkt dann mindestens fünf Jahre lang verkaufen und der Endkunde letztendlich das Produkt zehn Jahre lang nutzen wollen.

Ganz schnell kommen da Zeiträume von deutlich mehr als zehn Jahren als Lebenszyklus zusammen. Diese langen Zeiträume passen nur schwer zu der kurzfristigeren Taktung von Web-Technologien. Oder andersherum: Bei solch langen Zeiträumen sticht das Vertrauen in eine explizit installierte Software die Vorteile einer Zero-Installation geradezu aus. JavaFX besitzt diese langfristige Positionierung:

- Es ist unabhängig vom Browser; die wesentliche Abhängigkeit der Installation besteht zum Betriebssystem
- Es hat als Java-Standard ein eindeutiges und langfristiges Commitment seitens Oracle und der Java-Community
- Es steht in der Tradition von Java Swing, das einen Lebenszyklus von nun mittlerweile fünfzehn Jahren aufweisen kann

Die Kosten eines UI-Framework-Wechsels innerhalb des Lebenszyklus einer Anwendung sind sehr hoch. Die langfristige Stabilität von Web-Frameworks ist zumindest fragwürdig – hier ergibt die Positionierung von JavaFX deutlich Sinn.

„Zero Installation“ hat seinen Preis

Wie oben erwähnt, ist „Zero Installation“ ein wichtiges Argument für die Wahl eines HTML5-basierten UI-Frameworks. Aber, „Zero Installation“ hat einen Preis: Der Entwickler der Anwendung muss sich mit Browser-Inkompatibilitäten beschäftigen – sowohl zur Entwicklungszeit eines Produkts als auch während der nachgelagerten Wartungszeit. „Zero Installation“ bedeutet, dass letztlich der Endnutzer mitteilt, in welcher Browser-Umgebung er erwartet, dass die Software funktioniert. „Bring your own Browser“, heißt hier die Devise.

Dieser Aufwand ist hoch – trotz aller Frameworks, die man oberhalb von HTML5 einsetzt. Es sind oft die kleinen Inkompatibilitäten, die große Aufwände erzeugen unter dem Motto „Bei der Maske gibt es unter IE ein paar Pixel zu viel, die unter Firefox nicht da sind“. Je nach Wahl des HTML-Frameworks beginnt nun die Suche – mit mehr oder weniger Support durch den Framework-Hersteller. Bei einer Anwendung, bei der „Zero Installation“ keine zentrale Rolle spielt, sollte man also gründlich überlegen, ob man die damit einhergehenden Aufwände überhaupt rechtfertigen kann.

Viele Anwendungen richten sich an Benutzergruppen wie „Sachbearbeiter“ oder „professionelle Anwender“. Hier sind in der Regel eine explizite Installation am Client und somit auch die Nutzung von JavaFX am Client recht problemlos möglich. JavaFX bietet hier über sein neues Bundle-Konzept neue, App-ähnliche Installationsweisen an, in denen die Java-Umgebung nicht mehr vorausgesetzt, sondern Bestandteil des Bundles ist. Die Installation

geschieht durch einfachen Aufruf der App und erfordert beispielsweise keine Eingabe eines Administrator-Kennworts.

Effizienz der Entwicklung

„Alles ist in HTML5 möglich“, so die Aussage, die getroffen wurde. Nun, natürlich schließt sich die Folgefrage an: „Zu welchem Preis?“. JavaFX steht bezüglich der Entwicklungseffizienz auf einem anderen Fundament als HTML5:

- JavaFX basiert auf Java – einer etablierten Programmiersprache mit etablierten Konzepten wie Interfaces, Vererbung, Typsicherheit.
- JavaFX stellt Komponenten in Form einer direkt erweiterbaren Bibliothek zur Verfügung. Instanzen der Komponenten werden direkt in einem Komponentenbaum zusammengestellt (Scene Graph) und können dort als Ganzes oder in Teilen manipuliert werden (Größenskalierung etc.).
- Über Java gibt es eine riesige Auswahl an professionellen Werkzeugen zur Unterstützung der Entwicklung (Debugging, Profiling, Tracing etc.).
- Die Performance von JavaFX und Java ist in der Regel kein Thema; die Abarbeitungsgeschwindigkeit ist einfach hoch genug. Dies gilt insbesondere für das Vornehmen von Änderungen im Komponentenbaum (Scene Graph). Irgendwelche Tricksereien, die im HTML5-Bereich leider üblich sind (Beispiel: innerHTML-Update ist wesentlich schneller als direkte DOM-Manipulation), sind nicht vonnöten.

Die Grenzen dessen, was man mit JavaFX auf einfache und normale Weise machen kann, sind deutlich höher als die Grenzen in einer vergleichbaren HTML5-/JavaScript-Umgebung.

Eine konkrete, pauschale Quantifizierung dieser Aussage ist natürlich schwierig. Helfen kann hier allein der Kontakt zu konkreten JavaFX-Projekten – oder das Anfertigen von entsprechenden Prototypen im kleineren Umfeld, einmal auf Basis von JavaFX und einmal auf Basis einer HTML5-basierten Umsetzung.

Ganz klar ist auch: Wenn bereits Entwicklungserfahrung im Bereich der Swing-Entwicklung vorliegt, ist ein Umstieg des Entwicklers von Swing auf JavaFX recht

einfach, da die grundlegenden Konzepte der Entwicklung ähnlich sind.

Positionierung von JavaFX

Hier ist es sinnvoll, JavaFX zu positionieren und sich explizit gegen HTML5 zu entscheiden:

- Bei langfristig konzipierten Anwendungen; hier ist es vor allem die Framework-Sicherheit, die eine Rolle spielt.
- Bei Anwendungen mit einem gewissen Umfang; nur diese haben für den Benutzer auch einen gewissen Mehrwert, der eine (wenn auch App-basierte) Installation am Client rechtfertigt.
- Bei Anwendungen mit normalen bis komplexen Interaktionen. Wenn es etwas mehr sein soll, dann spielt JavaFX seinen Effizienzvorsprung aus. Hier ist insbesondere die komplette Nichtbeachtung von Browser-Kompatibilitätsproblemen sehr angenehm.

Diese Betrachtungsweise trifft bei vielen Geschäftsanwendungen zu – egal, ob sie als Standardprodukt oder als Projektlösung konzipiert sind:

- Durchweg langfristig ausgelegt
- In der Regel viele einfache Masken (Feld-Button-Dialoge etc.), aber auch einige komplexe Dialoge (Reporting, Prozessvisualisierung, etc.)
- Hohe Anforderung an Interaktionsmöglichkeiten einerseits (bis hin zu „alles per Keyboard“) und an die Performance andererseits
- Bedienung durch Sachbearbeiter, bei denen die Stabilität einer Installation wichtiger ist als „Zero Deployment“
- Eine große Entwicklungseffizienz ist erforderlich, da die Größen der Entwicklungsgruppen in der Regel recht überschaubar sind

Diese Geschäftsanwendungen sind in vielen Fällen noch gar nicht in der Browser-/HTML-Welt angekommen, sondern basieren noch auf Technologien der späten 1990er-Jahre und sind beispielsweise in Java Swing, in C++ oder auch noch in Delphi codiert. Betreffende Unternehmen, die den Versuch gestartet haben, ihre Anwendung auf HTML(5) zu portieren, haben in vielen Fällen bereits enorme Kosten in entsprechende Techno-

logie-Evaluationen gesteckt, um schließlich herauszufinden, dass die Komplexität und die Risiken einer Übertragung nach HTML5 zu groß sind und gleichzeitig die Vorteile („Zero Deployment“) im konkreten Einsatzgebiet nicht signifikant genug sind.

„Pro JavaFX“ ist keine exklusive Entscheidung

Eine Entscheidung für JavaFX ist auch damit verbunden, dass in der Software-Architektur der Anwendung trotzdem Raum für gewisse HTML(5)-basierte Front-Ends ist. Von daher sollte man nur dort auf JavaFX setzen, wo es sinnvoll ist, und von Anfang an überlegen, wie typische HTML-Szenarien parallel integriert werden können. In jeder Anwendung gibt es die operativen Szenarien, bei denen eine Implementierung mit JavaFX sinnvoll ist. Es existiert auch eine Reihe von Szenarien, bei denen parallel dazu HTML(5)-Dialoge zur Verfügung gestellt werden müssen: User Self Service, Informationssystem (auch an externe Nutzer) oder einfache Bewilligungsmasken als Bestandteil von Workflows.

Die Entscheidung, für bestimmte Szenarien JavaFX einzusetzen, ist eine wichtige Entscheidung. Aber sie ist noch keine Architektur-Definition. Die folgenden Fragen sind prinzipiell unabhängig von JavaFX, aber für die spätere Ausgestaltung

des Dialogteils der Anwendung von zentraler Bedeutung:

- Wo läuft welcher Teil der Anwendung? Auf dem Server oder auf dem Client?
- Wie autark ist der Client? Sind Phasen ohne ein Netzwerk möglich oder setzt er eine konstante Verbindung voraus?
- Gibt es ein eher Server-zentrisches oder Client-zentrisches Programmier-Modell, wenn es um die konkrete Bereitstellung von Dialogen für den Anwender geht?

Gut ist, dass JavaFX hier keine Schranken vorgibt. Ob man die Anwendung bewusst als „Fat Client“ oder als „Thin Client“ erstellt, spielt bei JavaFX letztendlich (und glücklicherweise) keine Rolle.

Fazit

JavaFX kommt spät in den UI-Technologie-Markt, in dem sich vor allem HTML5-basierte Technologien tummeln. Der strategische Einsatz von JavaFX muss deswegen für ein Szenario gut begründet und kann kein Selbstzweck sein. Das Positive aus JavaFX-Sicht: Es gibt diese guten Gründe und es gibt entsprechend gute Szenarien, in denen der Einsatz sinnvoll ist.

JavaFX ist nur indirekt eine HTML5-Konkurrenz, da es naturgemäß dort, wo

HTML5 stark ist („Zero-Deployment“-Szenarien), nicht viel entgegengesetzt kann. Aber für die vielen Anwendungen, deren Oberfläche bislang nativ codiert wurde (C++, Java Swing, Delphi etc.), ist JavaFX eine ernsthafte Technologie, die auf jeden Fall in Erwägung gezogen werden sollte.

Björn Müller

bjoern.mueller@captaincasa.com



Björn Müller arbeitete zunächst zehn Jahre in der Anwendungsentwicklung, Basisentwicklung und Architekturentwicklung für SAP. 2001 erfolgte die Gründung der Casabac Technologies GmbH als Pionier im Bereich von Rich Internet Applications auf Basis von Client-Scripting (AJAX). 2007 gründete er die Captain-Casa Community, eine Verbindung mittelständischer, deutschsprachiger Softwarehäuser mit dem Fokus auf Rich Client Frameworks für umfangreiche, langlebige Anwendungssysteme.

Contexts und Dependency Injection – der lange Weg zum Standard

Dirk Mahler, buschmais GbR

Die JSRs 299 (Contexts und Dependency Injection, CDI) und 330 (Dependency Injection, DI) sind bereits Ende 2009 als Teil von Java EE 6 veröffentlicht worden. Implementierungen stehen – eingebettet in Application-Servern oder als Bibliotheken – ebenfalls seit geraumer Zeit zur Verfügung. Es ist aber zu beobachten, dass sich Entwickler nur schwer mit CDI anfreunden können oder es nur halbherzig einsetzen. In einer Reihe von Artikeln werden die zu lösenden Probleme und angebotenen Lösungskonzepte näher erläutert.

Nicht selten ist zu erleben, dass im Zusammenhang mit CDI von „unverständlicher Magie“ gesprochen wird. Es lassen sich verschiedene Ursachen für das Zustandekommen dieser Aussage finden:

Die Spanne reicht dabei vom fehlenden Wissen über den Standard, bis hin zum hohen Abstraktionsgrad und der damit zwangsläufig verbundenen „Angst vor Kontrollverlust“.

Nachfolgend wird daher ein problemorientierter Einstieg in das Thema aufgezeigt, um den Sprung über möglicherweise vorhandene Schwellen zu erleichtern. Den Auftakt bietet ein kleiner, keinen

Anspruch auf Vollständigkeit erhebender Streifzug durch die Beziehungshistorie von Objekt-Instanzen.

Handwerk: Kopplung

Im Jahr 2002 erschien der Artikel „Java’s new Considered Harmful“ (siehe „<http://drdobbs.com/184405016>“). Er beschrieb zwei Probleme, die durch die Verwendung des Java-Schlüsselwortes „new“ entstehen: Speicher-Allokation und fehlende Polymorphie. Zugegebenermaßen hat der erste Aspekt heutzutage an Relevanz verloren, der zweite ist jedoch von umso größerer Bedeutung.

Das Konstrukt in Listing 1 beschreibt die „Abhängigkeit“ eines Konsumenten („ClientImpl“) von einem Dienst („ServiceImpl“) und erzeugt bereits zur Compile-Zeit eine feste „Kopplung“ zwischen beiden: Es können zur Laufzeit ausschließlich Instanzen des konkreten Typs „ServiceImpl“ verwendet werden, obwohl gegebenenfalls in verschiedenen Kontexten die Nutzung eines dazu zuweisungskompatiblen Typs (also einer Ableitung) sinnvoll beziehungsweise sogar notwendig sein kann. Ein populäres Beispiel dafür ist die Verwendung von Mocks in Unit-Tests, denkbar sind aber auch Implementierungen von Remote-Diensten, die auf unterschiedlichen Technologien basieren (SOAP, REST, Remote-EJB, etc.).

Industrialisierung: Fabriken

Zur Auflösung einer derartig festen Kopplung wurde im genannten Artikel bereits der Einsatz sogenannter „Factories“ empfohlen: Die Erzeugung benötigter Instanzen wird aus dem jeweiligen Konsumenten ausgelagert und kann so vom vorhandenen Kontext abhängig gemacht werden, der im Zweifelsfall erst zum Zeitpunkt der Assemblierung – also des Zusammenbaus und der Auslieferung der Anwendung – bekannt ist. In einer Test-Umgebung kann also ein Mock oder Simulator anstelle einer im Produktionssystem verwendeten Implementierung zur Verfügung gestellt werden.

Ein logischer und empfehlenswerter, aber nicht zwingend notwendiger Schritt ist in diesem Zusammenhang die Beschreibung der Abhängigkeit als Schnittstelle, in Java ausgedrückt über ein Interface („IService“). Dies zeigt Listing 2. Kurz zusam-

```
ClientImpl.java
public class ClientImpl {
    private ServiceImpl service = new ServiceImpl();
    ...
}
```

Listing 1

```
IService.java:
public interface IService {
    void doSomething();
}
ClientImpl.java:
public class ClientImpl {
    private IService service = (IService) ServiceFactory.
getService(IService.class);
    ...
}

RestServiceImpl.java:
public class RestServiceImpl implements IService {
    ...
}
```

Listing 2

mengefasst, besteht die entscheidende Idee also darin, die Erzeugung und Bereitstellung von Abhängigkeiten aus dem Anwendungscode in technische Infrastruktur auszulagern.

Indirekt kann so eine weitere Form der Kopplung aufgehoben werden, die nicht unmittelbar aus dem Code ersichtlich ist: Besitzt der benötigte Dienst einen inneren Zustand (wie Cache), darf sein „Lebenszyklus“ – also Erzeugung und Zerstörung – nicht an den seiner Konsumenten („ClientImpl“) gekoppelt sein. In Listing 1 wird die Service-Instanz für jede „ClientImpl“-Instanz neu erzeugt – die Realisierung eines effizienten Cache-Service wäre auf diesem Wege so gut wie unmöglich. In Listing 2 kann die verwendete Factory eine Entscheidung darüber treffen, ob eine bereits bestehende Instanz des Service wiederverwendet werden kann. Sie muss hierfür jedoch über die notwendigen Informationen bezüglich der Eigenschaften des Service verfügen.

Durch den Einsatz der Factories eröffnet sich eine weitere interessante Möglichkeit: Anstelle der Original-Instanz des Dienstes

kann ein Stellvertreter-Objekt („Proxy“) ausgeliefert werden, das ebenfalls das angeforderte Interface implementiert und Methoden-Aufrufe an die Dienst-Instanz delegiert, diese aber beeinflusst. Ein Anwendungsfall hierfür ist die Behandlung sogenannter „Querschnittsaspekte“ („Cross Cutting Concerns“) wie Logging von Methoden-Ein- beziehungsweise Austritten oder Beginn und Abschluss von Transaktionen vor beziehungsweise nach einem Methoden-Aufruf (siehe Listing 3).

Wiederkehrende und oftmals technische Aspekte können damit elegant aus dem Anwendungscode ausgelagert werden. Dieser Ansatz erhöht nicht nur die Lesbarkeit, sondern sorgt auch dafür, dass redundante Implementierungen vermieden werden können und in allen Teilen der Anwendung gleichartig funktionieren. Aufgrund ihrer auf Schnittstellen basierenden Natur sind Proxys kaskadierbar, vor dem Methodenaufruf an der Dienstinstanz kann also eine ganze Kette von Proxy-Instanzen durchlaufen werden, die unterschiedliche Aspekte (Transaktionen, Logging, Sicherheit etc.) abbilden – die Ak-

```

TransactionProxy.java:
public class TransactionProxyImpl implements IService {

    private IService delegate;

    @Override
    public void doSomethingA() {
        beginTransaction();
        delegate.doSomethingA();
        commitTransaction();
    }

    @Override
    public void doSomethingB() {
        beginTransaction();
        delegate.doSomethingB();
        commitTransaction();
    }
    ...
}

```

Listing 3

tivierung und Reihenfolge wird dabei im Normalfall durch Konfiguration bestimmt.

Für den effizienten Einsatz dieser Lösung muss noch ein Problem gelöst werden, das in Listing 3 deutlich sichtbar ist: Für jedes Interface („IServiceA“, „IServiceB“, etc.), für das eine Factory Dienst-Instanzen erzeugen sollte, müsste theoretisch bereits zum Entwicklungszeitpunkt eine entsprechende Proxy-Implementierung (wie ServiceATransactionProxy, ServiceBTransac-

tionProxy) zur Verfügung gestellt werden, die entsprechend dem implementierten Dienst-Interface die jeweils erzwungenen Methoden enthält, die in ihrer Umsetzung jedoch bis auf den Aufruf der entsprechenden Methode des Delegate komplett identisch sind.

Gelöst wird dies mit einer generischen Implementierung des Proxy, der erst zum Zeitpunkt seiner Instanziierung (also zur Laufzeit) den Typ des gewünschten Inter-

face annimmt, Methoden-Aufrufe an diesem Interface abfängt („Interceptor“) und die von ihm bereitgestellte Funktionalität vor oder nach dem eigentlichen Methodenaufruf einfügt (bildlich gesprochen „herumwickelt“). Dieses Konzept wird als „dynamischer Proxy“ bezeichnet und typischerweise über die vorhandene technische Infrastruktur zur Verfügung gestellt – im vorliegenden Fall also durch die Factory.

Automatisierung: Frameworks

Die Verlagerung technischer Aspekte in die Zuständigkeit von Factories hatte die Konsequenz, dass deren Implementierungen recht hohe Komplexitäten bei stetig wiederkehrenden Mustern aufwiesen. Darüber hinaus war der Anwendungscode nach wie vor durchsetzt mit Konstrukten, die dem Auffinden der jeweiligen Factory und dem Holen der Service-Instanzen dienten. Diese beiden Umstände führten zum Entstehen generischer Frameworks, die auf das sogenannte „Inversion of Control“ (IoC) setzten, heutzutage besser bekannt unter dem Namen „Dependency Injection“ (DI).

Der prominenteste Vertreter seiner Art war lange Zeit das Spring-Framework. Es lagert die Deklaration der Abhängigkeiten zwischen Instanzen in eine oder mehrere XML-Dateien aus. Die Implementierungen der Klassen müssen dabei festgelegten Konventionen folgen und werden als „Beans“ bezeichnet. Anhand der deskriptiven

```

ClientImpl.java:
public class ClientImpl {
    private IService service;

    public void setService(IService service) {
        this.service = service;
    }
    ...
}

spring.xml:
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" ...>
    <bean id="client" class="com.buschmais.client.impl.ClientImpl" >
        <property name="service" refId="serviceBean" />
    </bean>
    <bean id="serviceBean" class="com.buschmais.service.impl.ServiceImpl" />
</beans>

```

Listing 4

Beschreibungen übernimmt das Framework die Erzeugung benötigter Instanzen und die „Injizierung“ benötigter Abhängigkeiten, beispielsweise über Konstruktoren oder Set-Methoden. Es agiert damit als eine generische Factory, die zugleich die Kontrolle über den Lebenszyklus der Anwendung und ihrer Komponenten übernimmt. Die Initialisierung einer Spring-basierten Applikation erfolgt im Normalfall nicht mehr in der „main()“-Methode einer Anwendungsklasse, sondern durch das Framework. Letzteres kann daher auch als „Container“ aufgefasst werden.

In Listing 4 ist zu erkennen, dass durch den oder die XML-Deskriptoren folgende Informationen für den Container bereitgestellt werden:

- Zu verwaltende Beans und deren Namen („clientBean“, „serviceBean“)
- Abhängigkeiten zwischen den Beans in Form zu setzender „Properties“ (Getter/Setter) und der jeweils referenzierten Bean-Namen

Daneben können auch technische Aspekte, wie das bereits erwähnte Transaktions-Management, deklariert sein. Diese sind über dynamische Proxys realisiert, die bei Bedarf vom Framework in die Beans injiziert werden.

Das Spring-Framework wurde populär, weil es eine im Vergleich zu EJB-1.x/2.x-Containern leistungsfähige, gleichzeitig aber leichtgewichtige und flexible technische Infrastruktur bereitstellte. Die bessere Lesbarkeit des Java-Codes wurde jedoch – insbesondere in größeren Anwendungen – um den Preis schwer wartbarer XML-Deskriptoren erkaufte.

Standardisierung: EJB 3, DI und CDI

Das im Jahr 2004 erschienene Java 5 wartete mit einer Neuerung auf, die sich gravierend auf Container und die mit ihnen verbundenen Programmiermodelle auswirken sollte: „Annotationen“. Deutlich sichtbar wurde dies mit dem Sprung der Spezifikation von Enterprise Java Beans (EJB) von 2.1 nach 3.0 im Rahmen von Java EE 5 (siehe Listing 5).

Durch den Container zu verwalten- de Beans sowie ihr Lebenszyklus werden anhand von Annotationen („@Stateless“) identifiziert, ebenso die zu injizierenden

```
EjbServiceImpl.java:
@Stateless
public class EjbServiceImpl implements IService {
    ...
}
ClientImpl.java:
@Stateless
public class ClientImpl implements IClient {
    @EJB
    private IService service;
    ...
}
```

Listing 5

Abhängigkeiten („@EJB“, „@Resource“). Es entfallen zumindest größtenteils die bisher notwendigen XML-Deskriptoren.

Leider war die Annotations-basierte Dependency Injection im Java-EE-5-Universum nur für die vergleichsweise schwergewichtigen und daher nicht für alle Anwendungsfälle sinnvollen EJBs spezifiziert. So mussten im benachbarten Umfeld von Java Server Faces 1.x Abhängigkeiten zwischen den Managed Beans nach wie vor in XML-Deskriptoren gepflegt werden, sodass sich hier wiederum alternative Frameworks (wie JBoss Seam, Spring) etablierten, die entsprechende Annotationen zur Verfügung stellten.

Es war daher konsequent und notwendig, diese Lücke mit dem Erscheinen von Java EE 6 zu schließen. Nach einigem Hin und Her zwischen verschiedenen Fronten im Java Community Process entstanden daraufhin zwei Java Specification Requests (JSRs): „Dependency Injection“ (DI, JSR330) und darauf aufbauend „Contexts and Dependency Injection“ (CDI, JSR299), die oftmals der Einfachheit halber schlicht unter CDI zusammengefasst werden. Sie greifen die geschilderten Erkenntnisse auf und verfeinern diese zu einem eleganten Programmiermodell.

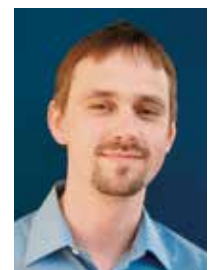
Aus der Geschichte lernen

Wichtig zum Verständnis von CDI sind die in dem geschichtlichen Abriss skizzierten Probleme und Lösungsansätze. Die Ziele des Standards sollen an dieser Stelle daher noch einmal überblicksartig zusammengefasst werden:

- Die Verwaltung von Abhängigkeiten zwischen Konsumenten und Diensten ...
- ... zur Aufhebung von Kopplungen auf den Ebenen von Typen und Lebenszyklen ...
- ... durch die Auslagerung in eine Container-Infrastruktur ...
- ... bei minimaler Durchsetzung des Anwendungs-Codes mit technischen Konstrukten ...
- ... und Unterstützung bei der Auslagerung von Querschnittsaspekten ...
- ... sowie Integration mit benachbarten Technologien.

In der nächsten Ausgabe wird auf die Umsetzung grundlegender Konzepte durch CDI eingegangen werden. Der Artikel widmet sich dabei den Fragen der Deklaration von Beans, deren Lebenszyklen sowie typischerer Injizierung und skizziert die Integration mit Enterprise Java Beans beziehungsweise Java Server Faces.

Dirk Mahler
dirk.mahler@buschmais.com



Das neue Release ADF Mobile 1.1

Jürgen Menge, ORACLE Deutschland B.V. & Co. KG

Seit Freigabe von Oracle ADF Mobile im Oktober 2012 wurden zahlreiche Artikel, Vorträge und Anwendungsbeispiele veröffentlicht. ADF Mobile ist ein Framework zur Entwicklung hybrider Applikationen für mobile Endgeräte [1]. Als Entwicklungsumgebung (IDE) kommt der Oracle JDeveloper zum Einsatz.

Im Mai 2013 erschien zusammen mit dem JDeveloper 11.1.2.4 ein erstes Update von ADF Mobile (interne Versionsnummer 11.1.2.4.39.64.36). Kurz danach wurde ein Patch Set (11.1.2.4.39.64.39) veröffentlicht, um einige kurzfristig aufgetretene Probleme zu beseitigen. Durch die Funktion „Check-for-Updates“ im JDeveloper ist sichergestellt, dass die aktuellste Version installiert wird. Der Artikel stellt die neue Funktionalität vor, die mit ADF Mobile 1.1 zur Verfügung steht.

Umstellung von PhoneGap 1.0 auf Apache Cordova 2.2.

Für die Integration gerätespezifischer Funktionen hat ADF Mobile bisher das Framework PhoneGap 1.0 verwendet. Inzwischen hat Adobe (nach Kauf des Unternehmens Nitobe) den Quellcode von PhoneGap an die Apache Foundation übergeben, bei der das Projekt unter dem Namen „Cordova“ weitergeführt wird. Mit der Umstellung auf Cordova 2.2 trägt Oracle dieser Umstellung Rechnung und ermöglicht zugleich weitergehende Möglichkeiten, von Cordova bereitgestellte Plug-ins zukünftig zu integrieren.

Push Notification

Eine häufige Anforderung besteht darin, dass Applikationen Benachrichtigungen vom Server erhalten, wenn ein bestimmtes Ereignis eingetreten ist. So könnte etwa innerhalb eines Geschäftsprozesses eine Anfrage gestellt worden sein, die vom Manager zu bestätigen oder abzulehnen ist. Dazu ist serverseitig ein Push-Mechanismus erforderlich. Clientseitig muss das Betriebssystem die Benachrichtigung entgegennehmen und diese an die betreffende Applikation weiterleiten.

ADF Mobile nutzt die von Apple (Apple Push Notification) beziehungsweise Google (Google Cloud Messaging) bereitgestellten Services, um Benachrichtigungen an das Gerät zu senden. Eine ADF-Mobile-Applikation kann sich für diese Benachrichtigung registrieren („Methode onOpen()“) und erhält die empfangenen Nachrichten vom Betriebssystem. In der Methode „onMessage()“ wird festgelegt, wie die Applikation mit der Nachricht umgehen soll. [Abbildung 1](#) zeigt den Ablauf des Prozesses und das Zusammenspiel der beteiligten Komponenten.

Badging

Diese Funktionalität hängt mit der Push Notification zusammen und ist nur für Geräte auf Basis von iOS verfügbar. Ein kleiner roter Zähler, der das Icon der Applikation überlagert, zeigt die Anzahl der eingegangenen Nachrichten an.

Anzeige von Dateien

Das generische Device Data Control enthält eine neue Methode „displayFile(String, String)“. Mit dieser Methode können Dateien (wie PDF, Word, Excel) als Teil der Applikation angezeigt werden. Auf iOS-Geräten wird der QuickLook Preview aufgerufen, auf Android-Geräten kommt die externe Applikation zum Einsatz, die für den angegebenen MIME-Type registriert wurde ([siehe Abbildung 2](#)).

Application Archive Packaging

Diese Funktionalität ist für Lösungsanbieter (ISV) interessant, die ihre mobile Lösung an Kunden verkaufen, die diese Lösung erweitern und mit eigenen Zertifikaten signieren wollen. Mit dem neuen Release können mobile Applikationen in

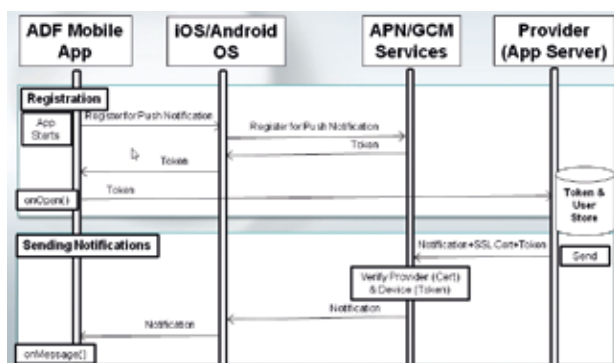


Abbildung 1: ADF Mobile Push Notification

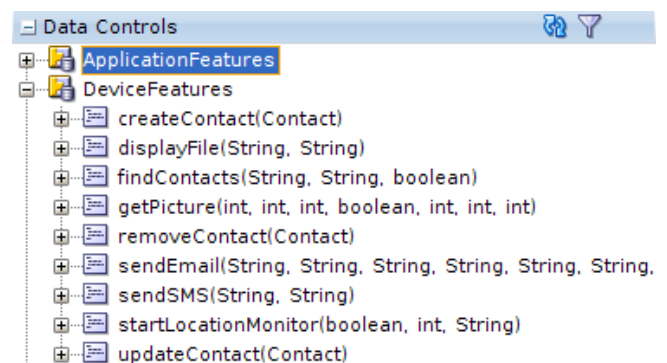


Abbildung 2: Device Data Control

ein „Mobile Application Archiv“ (.maa) gepackt und ausgeliefert werden. Es enthält den Programmcode nur in kompilierter Form. Nachträglich können dem Archiv aber Erweiterungen oder Zertifikate hinzugefügt werden.

Aktualisierung der Mobile SDK

Für ADF Mobile 1.1 wurden die aktuellen Versionen der für die Entwicklung benötigten plattform-spezifischen SDK zertifiziert:

- Apple Xcode 4.5, als OS wird Lion oder höher benötigt
- Android 4.2

Performance-Optimierungen und Beseitigung von Fehlern

Im neuen Release wurden Optimierungen der Performance durchgeführt und verschiedene Fehler beseitigt. Eine Liste der behobenen Fehler ist in den Release-Notes zu finden [2].

Erweiterte Layout-Möglichkeiten

Für die Gestaltung mobiler Seiten (AMX Pages) stehen zusätzliche Komponenten zur Verfügung (siehe Abbildungen 3 und 4):

- Rating Gauge
- Dial Gauge
- Zusätzliche Charts

AMX-Komponenten und das Framework unterstützen im Release 1.1 auch Sprachen mit unterschiedlicher Schreibrichtung („right to left“).

Unterstützung für iPhone 5 und iPad mini

Das neue Release bringt die von Apple für den AppStore geforderte Unterstützung des iPhone-5-Formfaktors und des neuen iPad mini.

Migration von ADF-Mobile-Applikationen 1.0

Bestehende ADF Mobile-Applikationen der Version 1.0 müssen auf 1.1 migriert werden. In Abhängigkeit von den verwendeten Funktionen sind manuelle Schritte notwendig, die in den Release Notes [2] im Abschnitt Migration beschrieben sind.

Weitere Informationen

Ein Video [3] gibt einen guten Überblick über die neuen Funktionen von ADF Mobile

1.1. Eine ausführliche Beschreibung findet man in der aktualisierten Version der Dokumentation [4].

Mit ADF Academy wurde ein neues interaktives Medium eingeführt, um ADF-Technologien anschaulich zu vermitteln. Die erste Folge [5] behandelt ADF Mobile, allerdings noch auf Basis der Version 1.0.

Ausblick

Es gibt bereits weitreichende Pläne für kommende Versionen, darunter:

- Die bessere Nutzung von RESTful Web Services
- Verbesserte Integration von mobile Security
- Die Offline Datenspeicherung
- Eine bessere Integration der Cordova Plug-ins

Ergänzung vom 10. Oktober 2013

Vor wenigen Tagen wurde ein Patch zu ADF Mobile 1.1 veröffentlicht. Die exakte Versionsnummer lautet 11.1.2.4.39.64.51. Der Patch kann über die Check-for-Update-Funktion des JDevelopers 11.1.2.4 installiert werden. Er enthält neben einer Reihe von bereinigten Fehlern (siehe <http://www.oracle.com/technetwork/developer-tools/jdev/documentation/11-124-rn-1942343.html#11.1.2.4.0ShermanUpdate4-ReleaseNotesREPOSITORY-ADFMobile>) auch zwei neue Funktionen (siehe http://docs.oracle.com/cd/E37975_01/doc.111240/e24475/define_features.htm#CIHIFGEC):

- *URL Scheme Support*
Bei der Definition einer ADF Mobile-Applikation lässt sich eine URL angeben. Andere Applikationen können über diese die ADF Mobile-Applikation aufrufen und beim Aufruf Parameter übergeben
- *Custom SOAP Header Support*
Bei Verwendung des Web Service Data Controls lassen sich eigene SOAP-Header definieren. Dies kann zum Beispiel sinnvoll sein, um eine bestimmte Variante der Authentifizierung zu implementieren

Literatur

- [1] Oracle ADF Mobile Overview and FAQ: <http://www.oracle.com/technetwork/developer-tools/adf/learnmore/adfmobilefaq-1866697.pdf>
- [2] Oracle ADF 11.1.2.4 Release Notes: <http://www.oracle.com/technetwork/developer-tools/jdev/documentation/11-124-rn-1942343.html>



Abbildung 3: Rating Gauge



Abbildung 4: Dial Gauge

- [3] What's New. ADF Mobile 11.1.2.4.0 Video: http://www.oracle.com/goto/newsletters/qtr/adf/0513/download_ADFMobile.html?msgid=3-8370815808
- [4] Mobile Developer's Guide for Oracle ADF: http://docs.oracle.com/cd/E37975_01/doc.111240/e24475/toc.htm
- [5] ADF Academy - ADF Mobile: http://download.oracle.com/otn_hosted_doc/jdeveloper/academy/Developing%20Applications%20with%20ADF%20Mobile/player.html

Dr. Jürgen Menge
juergen.menge@oracle.com



Dr. Jürgen Menge hat bei Oracle zunächst sechs Jahre in der Schulung als Trainer für die Entwicklungs-Werkzeuge Oracle Designer und Oracle Forms gearbeitet, um dann als technischer Berater in den Lizenzvertrieb zu wechseln. Seine fachlichen Schwerpunkte sind sowohl die klassischen Entwicklungs-Werkzeuge als auch die neuen, Standard-basierten Tools (JDeveloper/ADF, BI Publisher etc.).

Einfach skalieren

Leon Rosenberg, anotheria solutions GmbH

Von B2C-Portalen wird vor allem eins erwartet – zu skalieren. Leider wird Skalierbarkeit zu oft zur Technologie-Frage erklärt – die Auswahl einer passenden „Hype-Technologie“ würde genügen. Dass dem nicht so ist, belegt spätestens der Live-Betrieb. Anstatt die Technologie-Keule zu schwingen, zeigt der Artikel, wie sich mit intelligenter Architektur und dem Verzicht auf das Datenmodell ein hoch skalierendes, hoch performantes Portal entwickeln lässt. Neben den grundsätzlichen Konzepten sind konkrete Skalierungs-Szenarien und Lösungen dargestellt.

In den dunklen Anfangszeiten des Internets – also Ende des letzten Jahrtausends – wurde die Frage der richtigen Architektur eines Systems zur Datenbank-Wahl degradiert. Beauftragte der CEO eines Unternehmens den Bau eines neuen Portals, stellte sich das Entwicklerteam die Frage, ob man eine Oracle-Enterprise-Lizenz benötige oder ob eine einfache Lizenz ausreichend sei. Ganz wilde Kerle brachten sogar Poet oder Versant (oder andere objektorientierte Datenbanken) ins Spiel. Anschließend wurden Daten-Modelle erstellt, die in den meisten Fällen Datenbank-Modelle waren, bevor man sich überhaupt die Frage stellte, was genau das neue System leisten und wie es funktionieren soll.

Obwohl wir nun, gut zehn Jahre später, um eine ganze Reihe interessanter Entwicklungen in der Software-Entwicklung weiter sind, läuft es heute ganz ähnlich ab, nur dass es nicht um Oracle vs. Informix geht, sondern darum, ob man Mongo, Hadoop oder Elasticsearch nimmt. Zweifelsohne sind das tolle und sehr nützliche Technologien, jedoch sollte deren Wahl nicht vor der Architektur-Entscheidung stehen. Mit anderen Worten: Die Technologie sollte der Architektur dienen, indem sie bestimmte Aufgaben innerhalb dieser Architektur übernimmt. Die Aufgaben sollten allerdings von der Architektur und den Anforderungen diktiert werden.

Der „Technology first“-Ansatz, dem man häufig in der Software-Entwicklung begegnet, ist für die weniger technisch versierten Entscheider zwar attraktiv, führt jedoch selten zu der wirklich guten Lösung für das konkrete Problem. Wenn ein Startup-Unternehmen Mongo, Bootstrap,

ElasticSearch, Rails etc. nutzt, kann es damit bestimmt auch seine Probleme lösen und wenn nicht, dann kann ihm keiner vorwerfen, er hätte nicht die fortschrittlichste Technologie benutzt. Der Autor vertritt hingegen den entgegengesetzten Ansatz: „Architecture first“. Das bedeutet, dass in erster Linie das konkrete Problem architektonisch gelöst wird und die Technologie nur ein Mittel zur Umsetzung der Architektur ist. Das bedeutet auch, dass die Technologie nur ein Teil der Lösung ist, und zwar nur dann, wenn sie auch zur Lösung beiträgt.

Hierzu ein Beispiel: Jahrelang wurden relationale Datenbanken zur Lösung sämtlicher Probleme – auch für Internet-Portale – herangezogen und jahrelang hatte man damit Probleme, diese dann zu skalieren, sobald das Schema zu komplex wurde. Aus dieser Not heraus wurde die Nachfolge-Generation der RDBMS-Systeme, die NoSQL-Datenbank, geboren. Ob es sich hier um wirkliche Neuentwicklungen oder um die Wiederbelebung ganz alter Ideen handelt, ist hierbei irrelevant.

Der Erfolg der NoSQL-Datenbanken basiert zum Großteil darauf, dass sie die Schwäche der SQL-Datenbanken, nämlich die „Joins“, erkannt haben und sie erst gar nicht unterstützen. Nur wenn man seine Architektur so baut, dass man keine „Joins“ braucht, sind die SQL-Datenbanken kein Skalierungsproblem mehr.

Was eine Architektur wirklich ist

Bevor wir darüber sprechen, wie man die richtige Architektur findet, welche die üblichen nicht funktionalen Anforderungen wie Flexibilität, Skalierung und Manage-

barkeit erfüllt, müssen wir uns die Frage stellen, was eine Architektur überhaupt ist. Die Meinungen darüber gehen weit auseinander. Einige sehen darin eine sehr abstrakte Art der Anforderungsbeschreibung, andere wiederum sagen, die Einteilung des Codes in Packages und das Strukturieren dieser Packages sei die Architektur.

Auf der Seite <http://www.sei.cmu.edu/architecture/start/glossary/community.cfm> findet man eine große Auswahl verschiedener Definitionen. Der Autor findet diese am passendsten: „Software-Architektur ist die Struktur eines Systems, die aus Komponenten, den extern sichtbaren Eigenschaften dieser Komponenten und den Beziehungen zwischen ihnen besteht.“

Bei der Software-Architektur geht es also um die Komponenten eines Systems und darum, wie diese miteinander kommunizieren. Nun will man keine universelle Architektur bauen, also nicht nach dem heiligen Gral der Software-Entwicklung suchen, sondern sich um jene hochskalierbaren B2C-Portale kümmern, bei denen vor allem die Benutzer jede Menge an Zeit verbringen, sei es in Online-Shops, Vergleichsportalen oder sozialen Netzwerken in all ihren Ausprägungen. Diese Portale haben aus Sichtweise der Software-Architektur einige Gemeinsamkeiten:

- Sie haben alle eine hohe „read/write“-Ratio – die Menge der Lese-Zugriffe ist in der Regel deutlich höher als die der Schreibzugriffe (bis zu 90 Prozent)
- Sie haben mehrere klar trennbare Funktionen (wie Nachrichten-System oder Profil-Ansicht)



Abbildung 1: Services richtig schneiden

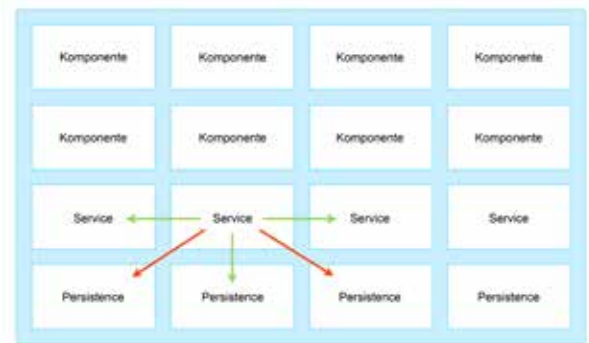


Abbildung 2: Erlaubte (grüne) und verbotene (rote) Kommunikationswege

- Sie unterliegen Peaks, die ein Vielfaches der Normal-Last zu bestimmten Tages-, Wochen- oder Jahreszeiten verursachen
- Sie verändern sich ständig und sehr schnell, sowohl was den Code als auch was den Content angeht

Grundsätzliche Architektur-Prinzipien

Eine sehr populäre Software-Architektur beim Bau solcher Portale ist die Service-orientierte Architektur (SOA). Sie ist etwas in Verruf gekommen durch die Popularität der Web-Services, einer Architektur, die mit SOA wenig zu tun hat, aber gerne mit ihr verwechselt wird. SOA ist viel älter und reifer als Web-Services und bietet – richtig angewandt – Antwort auf viele Skalierungsfragen.

Wenn wir zurück an unsere Architektur-Definition denken, sagt SOA, dass unsere Komponenten Services oder Clients sind, wobei eine konkrete Komponente sowohl das eine als auch das andere gleichzeitig sein kann, und dass die extern sichtbaren Eigenschaften dieser Komponenten die Interfaces sind, die jene Komponenten zur Verfügung stellen. Dabei gibt es zwischen den Komponenten zwei Arten von Beziehungen und folglich auch zwei Arten von Kommunikation:

- *Direkte oder synchrone Kommunikation* Methodenaufrufe, also Inanspruchnahmen eines Service durch einen Client
- *Indirekte oder asynchrone Kommunikation* Benachrichtigungen über Zustandsänderungen (Events), die eine Komponente, ohne zu wissen, ob sie Empfänger hat, überallhin ausstrahlt

Die direkte Kommunikation ähnelt einem Telefonanruf bei einem telefonischen Be-

stellungsservice, während die indirekte Kommunikation mehr an einen Börsenticker erinnert, der unabhängig davon läuft, ob ihn nun jemand liest oder nicht. Sowohl die „abfragbaren Methoden“ als auch die „ablauschbaren Daten“ stellen, Software-architektonisch gesehen, Interfaces dar, also Mittel, über die man mit der Komponente kommunizieren kann.

Isolation bis in die Datenbank

Ein weiteres sehr nützliches Prinzip von SOA ist die Isolation der einzelnen Komponenten voneinander. Das bedeutet unter anderem, dass eine Komponente die Alleinherrschaft über ihre Daten hat und niemand diese Daten verändern darf, ohne dass die Komponente darüber zumindest in Kenntnis gesetzt wurde. Idealerweise soll die Komponente die Datenmanipulation selbst durchführen.

Service-orientierte Architekturen haben viele Vorteile, aber der Wichtigste ist, dass sie kein globales Datenmodell besitzen. Die Interfaces einer Komponente sind alles, was über die Komponente nach außen bekannt ist. Das Innenleben jeder Komponente ist ihr selbst überlassen und nach außen hin unsichtbar.

Manchen fällt es schwer, diesen freiwilligen Verzicht auf ein Datenmodell zu akzeptieren. Mit einer schnellen SQL-Abfrage könnte man eine komplexe Recherche schnell durchführen.

Es stimmt, dass eine Verknüpfung der Daten verschiedener Services auf der Datenbank-Ebene Vorteile für Nachforschungen, Data Mining und statistische Auswertungen bietet. Aber niemand sagt, dass diese Verknüpfungen auf dem Produktionssystem existieren müssen.

Wenn man Daten für statistische Auswertungen braucht, so kann man diese Daten regelmäßig aus dem Produktionssystem überführen und dabei so viele Verknüpfungen herstellen wie nötig. Das Produktionssystem selbst sollte aber von solchen Dritt-Verwendungen freigehalten werden, die in der Regel die Performance deutlich verringern. Der Verzicht auf ein gemeinsames Datenmodell bedeutet für die Systemarchitektur Folgendes:

- Anstelle eines Daten-Modells existiert ein Service-Modell. Man könnte dazu auch Enterprise-Modell sagen, wenn dieses Wort nicht schon mehrfach für alles Mögliche herhalten musste. Das Service-Modell besteht aus den Services in dem System, den Artefakten, die sie verwalten und den Beziehungen zwischen den Artefakten.
- Jeder Service und jede Komponente sind völlig frei in der Wahl ihrer Persistenz. Der Service, der hochstrukturierte Daten hat, kann sie in einer SQL-Datenbank ablegen; derjenige, der eher Blobs verwaltet, seien es Bilder oder Texte, kann das in einer für diese Zwecke passenden Persistenz tun.
- Services werden innerhalb des Systems unterschiedlich belastet. Steigt in einer Datenbank-orientierten Zwei-Tier-Applikation immer die Last auf das Gesamtsystem an, ist bei SOA leicht zu identifizieren, wo die Last genau ansteigt. Das erlaubt es, die Last an genau dieser Stelle effizient durch Optimieren oder Skalieren zu reduzieren.

Damit aber die Vorteile einer SOA sich in vollen Zügen entfalten können, müssen

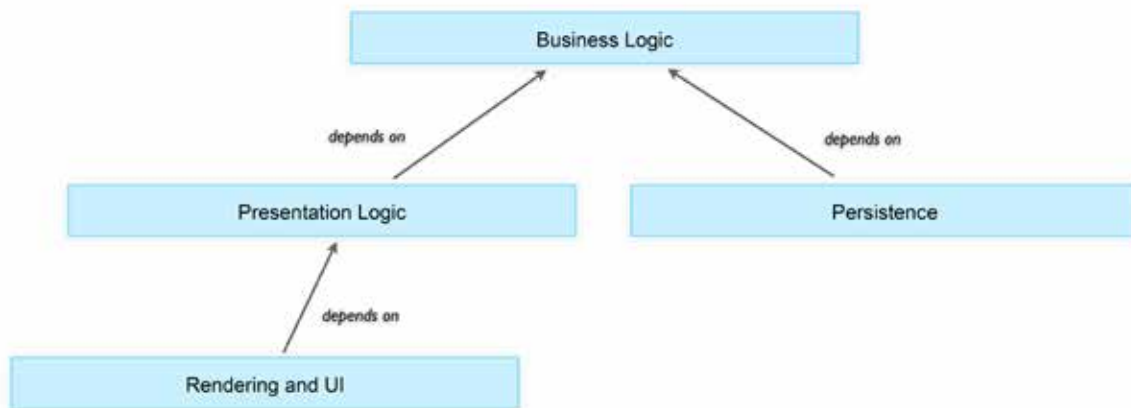


Abbildung 3: Abhängigkeiten zwischen den Layern

die Services richtig geschnitten sein. Große, monströse Services werden leicht zu Applikationen innerhalb der Applikation und zeigen all die Probleme auf, die man nicht haben wollte. Sind die Services hingegen zu klein, ist der Kommunikations-Overhead zu groß, um sinnvoll skalieren zu können. Die richtige Servicegröße findet man am einfachsten, indem man sich zwei Paradigmen der Software-Entwicklung bedient: „Design by Responsibility“ und „Layer Isolation“. Die Layer Isolation legt den grundsätzlichen Verantwortungsbereich des Service fest: Was ist ein Service (Business Logic) und was nicht (etwa Presentation Logic). Mithilfe des Design by Responsibility können wir eine fachliche Trennung vornehmen, die uns hilft, den Verantwortungsbereich des Service eng zu halten (siehe Abbildung 1). Wenn man die Services identifiziert hat, muss man sich Gedanken darüber machen, wie sie miteinander kommunizieren sollen und dürfen (siehe Abbildung 2).

Ein globales Daten-Modell verhindert

Das übliche Layer-Modell ist vertikal. Oben liegt die Präsentation und unten die Persistenz. Folglich beginnen die Entwickler mit der Persistenz und entwerfen ein Daten-Modell. Tatsächlich muss man bei SOA aber in der Mitte anfangen (siehe Abbildung 3).

Auf diese Weise gelingt es uns, ein wirkliches Service-Modell zu entwerfen, bei dem jede Persistenz nur die Bedürfnisse eines Service erfüllt und auf ihn eingestellt ist. Das führt zu einer strikten Isolation einzelner Persistenzen, die wir für die Skalierung brauchen (siehe Abbildung 4).

Ein weiteres Paradigma, das man unbedingt befolgen sollte, heißt „KISS“, „Keep It Simple & Stupid“. Auf Software-Architekturen angewandt, muss eine Architektur nur die Komponenten enthalten, die absolut notwendig sind. Alles, was keinen Mehrwert bietet, und dazu können auch die eingangs erwähnten Hype-Technologien gehören, muss wegbleiben. Mit anderen Worten: Alles, was keine Miete zahlt, muss raus, denn jede Technologie hat Wartungskosten, die nur gerechtfertigt sind, wenn die Technologie auch auf das Ergebnis einzahlt.

Gute Architektur erlaubt gezieltes Tuning

Hat man seine Services fertig entworfen und entwickelt, muss man sich die Lastfrage stellen. Oft ist es erst im Live-Betrieb wirklich klar, wie viel Last auf eine Komponente zukommt. Natürlich ist es schön, einen aussagekräftigen Lasttest zu haben, es ist aber schwer, diesen zu entwerfen und umzusetzen, ohne das reale Verhalten der Benutzer zu kennen. Doch wenn man das Benutzerverhalten kennt, ist man bereits im Live-Betrieb. Wie auch immer, es ist überhaupt nicht schlimm, wenn man das Tuning erst im wirklichen Betrieb vornimmt, denn den berühmten Satz über „premature optimization“ kennt jeder.

Wichtig ist allerdings, jede Komponente zu überwachen, um die Engstellen rechtzeitig, am besten noch vor dem Überladen, zu erkennen und entsprechend reagieren zu können. Hat man eine Engstelle identifiziert, ergeben sich mehrere Reaktionsmöglichkeiten. Der Artikel geht auf zwei davon, „Caches“ und „Loadbalancing“, detaillierter ein.

Caches

Wie eingangs erwähnt, reden wir vor allem über B2C-Portale. Diese haben eine gemeinsame Eigenschaft, nämlich einen hohen prozentuellen Anteil von Lese-Zugriffen an der Gesamt-Zugriffszahl. Das spricht dafür, mit Caches zu arbeiten. In der Tat sind Caches ein extrem wirksames Werkzeug, das allerdings auch seine Tücken hat.

Es gibt mehrere Arten von Caches, die zwei Hauptarten sind Query- und Objekt-Caches. Query-Caches, auch „Method-Caches“ genannt, sind Caches, die meistens von außen eingesetzt werden. Man speichert also Ergebnisse von Methoden (Queries) und hofft, dass gleiche Eingaben auch immer dieselben Ausgaben erzeugen. Wird die gleiche Abfrage mehrmals ausgeführt, kann man ab dem zweiten Request die komplexe und zeitaufwändige Verarbeitung auslassen und direkt das Ergebnis der vorigen Operation zurückgeben. Der Vorteil von Query-Caches ist, dass sie meistens von der konkreten Architektur und Domäne unabhängig, also als fertiges Produkt leicht integrierbar sind. Dafür haben sie aber eine ganze Reihe von Nachteilen:

- Sie sind ineffektiv bei großen Interfaces, die mehrere Methoden bieten, um an eine Information (also ein Objekt) zu kommen, denn es wird der Pfad und nicht das Ziel im Cache gehalten.
- Sie sind größenproportional zu der Menge der möglichen Anfragen und nicht der möglichen Antworten, daher schwer zu planen und übergewichtig.
- Sie sind nicht elegant.

Bei den Objekt-Caches sieht die Sache ganz anders aus. Sie sind schwerer einzubauen, dafür jedoch um einiges effektiver. Die ideale Implementierung eines Objekt-Cache ist eine Collection (ob „Map“ oder „Liste“, kommt auf die Anforderungen an), die sämtliche von diesem Service verwalteten Objekte in ihrer Objekt-Form enthält. Idealerweise kann jede Lese-Abfrage mithilfe des Cache vom Service beantwortet werden, ohne dass sich dieser an eine externe Persistenz (also etwa die Datenbank) wenden muss. Eine solche 100-prozentige „Cachebarkeit“ ist oft nicht zu erreichen. Dort, wo sie erreicht werden kann, lohnt sie sich allerdings immer. Das beste Beispiel für einen 100-prozentigen Cache sind Benutzerdaten. Sie sind in der Regel sehr klein und daher leicht zu cachen (User-Id, E-Mail, Name, Registrierungszeit), werden jedoch permanent an vielen Stellen der Applikation gebraucht.

Cachen, was nicht existiert

Eine Sonderform des Objekt-Cache sind die sogenannten „Null- oder Negativ-Caches“. Meistens muss man einen Objekt-Cache erst zur Laufzeit befüllen, um etwa einen Kaltstart der Anwendung oder das Erstellen neuer Objekte (vor allem auch bei mehreren verteilten Instanzen desselben Service) zu ermöglichen. Das bedeutet, dass eine Anfrage nach einem bestimmten Objekt durch den Cache auf die Persistenz durchschlagen kann.

Was passiert aber, wenn der Service permanent nach nicht existierenden Objekten gefragt wird? Im schlimmsten Fall erzeugen diese Anfragen direkten Datenbank-Traffic, führen zur Überladung der Datenbank und der mühevoll aufgebaute Cache ist weitgehend ineffektiv. Negativ-Caches adressieren genau dieses Problem, indem man in ihnen vermerkt, welche Objekte man bereits erfolglos abzufragen versucht hat, und man sich das beim nächsten Request erspart.

Cachen, was sich ändert

Eine andere Unter-Kategorie von Caches sind Expiry-Caches. Dabei nimmt man an, dass ein Objekt oder seine Bestandteile über eine gewisse Zeit einen bestimmten Zustand behalten. Ein typisches Beispiel ist das Profil eines Benutzers bei einem Dating-Dienst. Ob der Benutzer „A“ die Än-

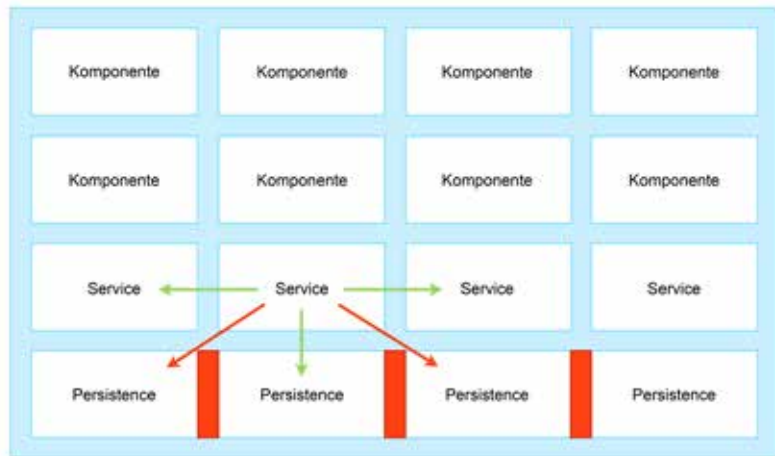


Abbildung 4: Persistenzen isolieren

derungen von Benutzer „B“ an seinem Profil in 5, 30 oder 60 Sekunden sieht, ist für den Benutzer „A“ nicht wichtig. Man kann also den Zustand des Profils von Benutzer „B“ für eine – menschlich gesehen – sehr kurze, für das System aber lange Zeit einfrieren und mit dieser Objektversion arbeiten. Auf diese Weise verhindert man, dass viele Anfragen, die sehr wahrscheinlich sowieso das gleiche Ergebnis gebracht hätten (wie oft ändert jemand schon sein Profil), nicht verarbeitet werden müssen. Diese Caching-Technik ist vor allem clientseitig sehr beliebt, um Netzwerk-Traffic zu sparen.

Ein anderes Beispiel, in dem ein Expiry-Cache sinnvoll sein kann, ist folgendes: Wenn während der Verarbeitung einer Anfrage davon auszugehen ist, dass das gleiche Objekt mehrmals angefragt wird, jedoch von so unterschiedlichen Stellen im Code, dass ein Zwischenspeichern des Objekts als Variable unmöglich oder unpraktisch ist. „Thread-Local“ ist ein beliebtes Mittel, um solche Zwischenspeicher zu realisieren.

Grundsätzlich geht es bei den Expiry-Caches um sogenannte „Trade offs“. Die Geschwindigkeit der Änderungssichtbarkeit wird gegen die Performance eingetauscht. Bei der Skalierung der Applikation geht es meistens darum, Trade offs zu erzeugen und Ressourcen, die im Überfluss vorhanden sind, gegen die knappen einzutauschen: „RAM vs. CPU“ oder „Network bei Caches“ sind weitere Beispiele.

Lokale Optimierungspotenziale sind begrenzt

Caches sind ein gutes Mittel, um die Performance einer einzelnen Komponente oder

eines Service zu optimieren, aber irgendwann ist jede Optimierung am Ende. Das ist der letzte Moment, an dem man sich die Frage stellen sollte, wie man mehrere Instanzen einer Komponente betreiben kann. Anders ausgedrückt: Wie skaliert man seine Architektur. Unterschiedlich Node-Typen lassen sich unterschiedlich gut skalieren. Dabei gilt die Regel, dass, je weiter vorn und näher am Kunden eine Komponente ist, desto leichter sie skaliert werden kann.

Performance ist aber nicht der einzige Grund, um die grundsätzliche Skalierungsfähigkeit anzustreben. Die Verfügbarkeit eines Systems hängt auch zu großen Teilen davon ab, ob man in der Lage ist, Komponenten mehrfach parallel zu betreiben, sodass der Ausfall eines System-Teils problemlos kompensiert werden kann. Die Kür ist natürlich, das System elastisch betreiben zu können und die Ressourcen-Verwendung dem echten Traffic dynamisch anpassen zu können.

Die Presentation-Ebene, zu der die Web-Apps gehören, die in einem Web-Server oder Servlet-Container laufen und die für das Generieren des Mark-up (HTML, XML oder JSON), also für die Präsentation der Seite, verantwortlich sind, lässt sich in der Regel sehr leicht skalieren. Solange die einzelnen Web-Server entweder komplett „stateless“ sind oder ihr State wiederherstellbar ist, also aus Caches besteht, oder ihr State ausschließlich einem User zugeordnet ist und der User immer wieder auf dem gleichen Web-Server landet, können neue Web-Server einfach dazugestellt und wieder entfernt werden. Interessanter wird

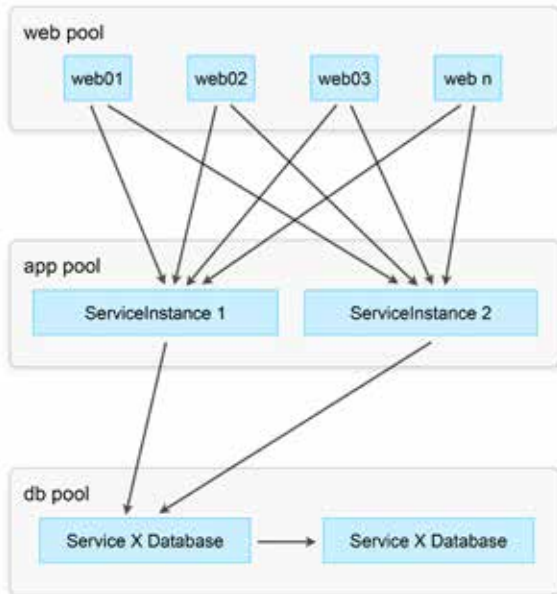


Abbildung 5: Aufrufe auf die Services nach Round-Robin verteilen

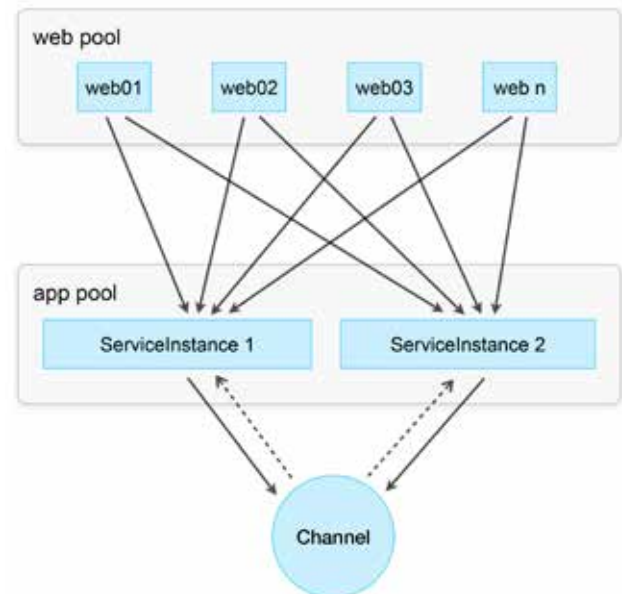


Abbildung 6: Round-Robin mit State-Synchronisation

es auf der Application-Ebene, in der die Services laufen. Bevor wir dazu übergehen, ein Blick auf die Ebene dahinter, die Datenbanken.

Die traurige Wahrheit, was das Skalieren der Datenbank betrifft, besteht darin, dass es nicht funktioniert. Mögen uns die Vertreter diverser Datenbank-Hersteller ständig vom Neuen versuchen zu überzeugen, dass ihre neuesten Features skalieren können – sie tun es nicht, wenn es darauf ankommt. Damit keine Missverständnisse aufkommen: Es gibt viele gute Gründe, eine Datenbank in einem Cluster oder einer Master/Slave-Replikation zu betreiben, aber Performance gehört nicht dazu. Der Hauptgrund, warum Applikationen so schlecht über die Datenbanken skalieren können, liegt darin, dass die Hauptaufgabe der Datenbanken darin besteht, Daten verlässlich zu speichern; Daten wieder auslesen ist etwas, das sie viel schlechter beherrschen. Wenn wir also nicht über die Datenbank skalieren können, müssen wir dies über die Application-Ebene tun. Es gibt viele Gründe, warum das gut funktioniert:

- Dort ist das meiste Wissen über die Applikation gesammelt. Wenn man also skalieren möchte, tut man es mit dem Wissen über die Applikation und ihre Benutzung.
- Dort kann man mit den Mitteln von Programmiersprachen arbeiten, was sehr viel mächtiger ist, als mit den Mitteln,

die einem auf der Datenbank-Ebene zur Verfügung stehen

Skalierungsstrategien

Zum Skalieren der Services stehen mehrere Strategien zur Verfügung. Zuerst einmal ist es wichtig zu definieren, was der Zustand eines Service, also sein State ist. Der State einer Service-Instanz ist die Menge an Informationen, die nur sie kennt und die sie von anderen Instanzen unterscheidet. Eine Instanz eines Service ist meistens eine JavaVM, in der eine Kopie des Service läuft.

Die Informationen, die den State ausmachen, sind meistens die Daten im Cache. Verwaltet ein Service keine Daten, ist er komplett „stateless“. Möchte man die Last auf einen Service reduzieren, versucht man, mehrere Instanzen davon zu betreiben. Dabei gibt es verschiedene Strategien, wie man die Last auf die einzelnen Instanzen verteilt. Die einfachste heißt „Round-Robin“. Dabei spricht jeder Client mit jeder Service-Instanz. Die Service-Instanzen werden abwechselnd genutzt. Das funktioniert gut, solange die Services „stateless“ sind und einfache Aufgaben abarbeiten (etwa E-Mails verschicken, siehe [Abbildung 5](#)).

Sofern die Service-Instanzen States besitzen, müssen sie diese untereinander synchronisieren, zum Beispiel indem sie die Änderungen in den States über einen Event-Channel oder ein anderes Publisher/

Subscriber-Modell austauschen (siehe [Abbildung 6](#)). Dabei kann jede Instanz jede Änderung, die sie auf einem Daten-Objekt vornimmt, anderen Instanzen mitteilen (gestrichelte Linie), sodass diese die Änderung auf ihren States auch durchführen können und so der gemeinsame Service State über alle Instanzen erhalten bleibt.

Diese Methode funktioniert gut bei geringem Traffic, bekommt aber Probleme, wenn die Menge der parallel durchgeführten Änderungen so groß ist, dass Konflikte durch gleichzeitiges Verändern eines Objekts auf mehreren Instanzen wahrscheinlich sind. Um den entgegenzutreten, kann man „Routing“ einführen. Darunter versteht man in dem Fall, dass die Service-Instanz kontextsensitiv ausgewählt wird. Der Kontext können in diesem Fall der Client, die Operation oder die Daten sein.

Routing über die Daten oder Sharding ist das mächtigste Instrument. Darunter versteht man einen Routing-Mechanismus, der eine Service-Instanz aufgrund von Operations-Parametern, also den Daten, auswählt. Idealerweise hat man einen primären Parameter wie eine User-Id, den man leicht in eine numerische Form umwandeln kann, sodass man eine Division mit Rest durchführen kann. Dividiert man durch die Anzahl der verfügbaren Instanzen, so bekommt die erste Instanz alle Anfragen mit „rest=0“, die nächste mit „rest=1“ und so weiter (siehe [Abbildung 7](#)).

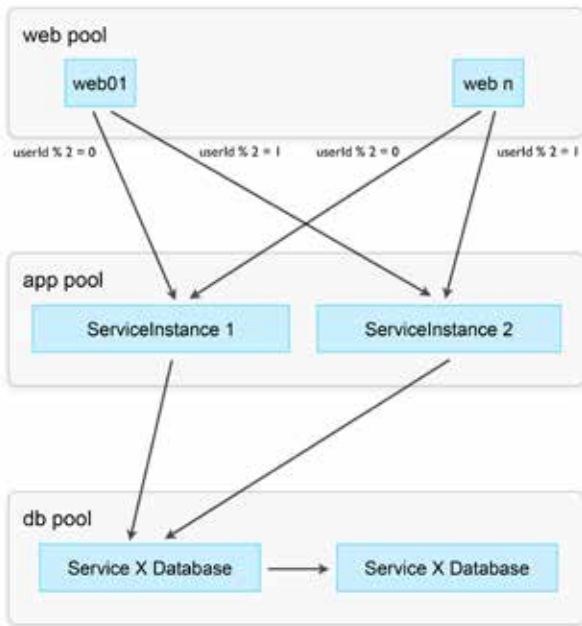


Abbildung 7: Sharding über Modul

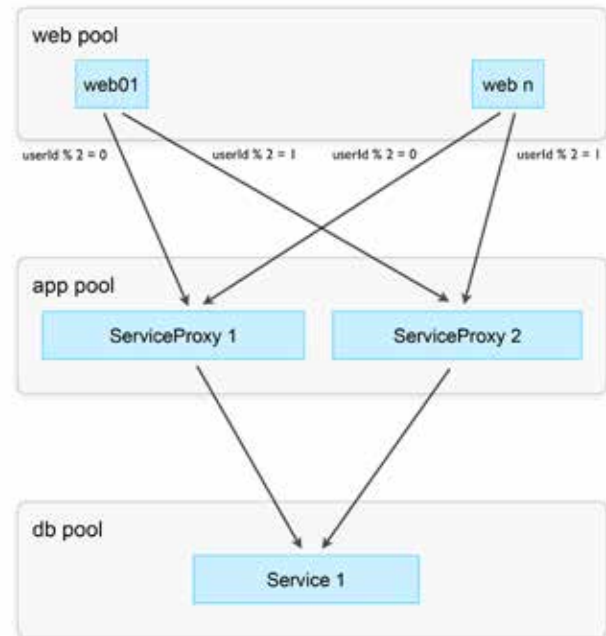


Abbildung 8: Service-Proxies

Diese Methode hat einen sehr nützlichen Nebeneffekt: Da alle Anfragen zum gleichen Benutzer immer auf derselben Instanz landen, wird eine Fragmentierung der Daten erreicht. Das bedeutet kleinere Caches, bei Bedarf gewollt fragmentierte Datenbanken und weiteres Optimierungspotenzial.

Natürlich kann man bei entsprechender Middleware die verschiedenen Skalierungsstrategien kombinieren, indem man beispielsweise die Service-Instanzen über Sharding zu Gruppen zusammenfasst, innerhalb derer man wiederum Round-Robin fährt. Diese weiterführenden Strategien zu diskutieren, würde aber den Rahmen des Artikels sprengen.

Manchmal können die Daten aber nicht „ge-sharded“ werden, vor allem dann, wenn eine Operation zwei Datensätze gleichzeitig modifiziert. Das wohl klassische Beispiel hierfür ist die Zustellung einer Nachricht von Benutzer „A“ zu Benutzer „B“, bei dem sowohl die Mailbox von Benutzer „A“ als auch die von Benutzer „B“ beschrieben ist.

Es ist unmöglich, eine Verteilung zu finden, bei der sowohl die Mailbox von Benutzer „A“ als auch die von Benutzer „B“ immer garantiert auf derselben Service-Instanz ist. Trotzdem gibt es auch hier eine Vielzahl von möglichen Strategien. Die wohl einfachste ist, den Service implizit aufzuteilen, idealerweise durch die Middleware und

ohne dass der Client etwas davon merkt (siehe Abbildung 8).

Die Proxies sind dafür da, Lese-Operationen noch vor dem eigentlichen Master-Service zu verarbeiten, und da wir sehr viel mehr Lese- als Schreib-Operation haben, den Master-Service zu entlasten. Die Lese-Operationen finden meistens im Kontext eines Benutzers statt, können also wie gewohnt „ge-sharded“ werden. Die verbleibenden Schreib-Operationen schlagen über die Proxies auf den Master durch, da aber sehr viel Last bereits in den Proxies abgearbeitet wurde, können die Schreib-Operationen weniger Schaden anrichten.

Fazit

Der Artikel zeigt, wie man die richtige Architektur findet und welche Mittel zur Verfügung stehen, um diese Architektur zu skalieren. Dabei stellt sich heraus, dass die Zeit, die man in die Wahl und die Einhaltung von Architektur-Paradigmen investiert, eine Investition ist, die sich, wenn es drauf ankommt, mehrfach auszahlt.

Slave-Proxies, Round-Robin-Service-Instanzen und Null-Caches sind Techniken, an die man sicherlich nicht gleich am Anfang der Entwicklung eines neuen Portals denkt. Es ist auch unnötig, sie auf Verdacht einzubauen; wichtig ist, sie zu kennen und eine Architektur zu haben, die den Einsatz solcher Tools erlaubt.

Nicht jedes Portal muss skalieren – aber wenn man es will, muss man es schnell können. Sich dabei ausschließlich auf Technologie zu verlassen, bedeutet, seine Handlungsfreiheit abzugeben und, wenn die Technologiewahl sich als falsch erwiesen hatte, gelegentlich die Scherben seines Systems aufsammeln zu müssen. Hat man im Gegensatz dazu auf die richtige Architektur gesetzt, Prinzipien und Paradigmen aufgestellt und sie befolgt, wird man mit Sicherheit auf jede neue Herausforderung eine Antwort finden können.

Leon Rosenberg
 leon@leon-rosenberg.net
 twitter: @dvayanu



Leon Rosenberg widmet sich als Software-Architekt seit mehr als zehn Jahren der Entwicklung von hochskalierenden Portalen, darunter Friendscout24, Parship und Allyouneed. Außerdem engagiert er sich bei Open-Source-Projekten im Bereich der Verteilung und des Application Management.

Cloud-hosted Continuous Integration mit CloudBees und Travis CI

Sebastian Herbermann und Sebastian Laag, adesso AG

Continuous-Integration-Server (CI) werden heutzutage von vielen Projekten genutzt, um Qualitätsstandards der eigenen Software dauerhaft sicherzustellen. Seine Einrichtung und Konfiguration ist jedoch aufwändig und erfordert entsprechende Ressourcen. Cloud-hosted CI-Lösungen bieten eine einfache Möglichkeit, CI für eigene Projekte zu nutzen.



Continuous Integration ist ein Standard-Werkzeug, um die Qualität der eigenen Software zu überwachen. Dazu führt ein CI-Server die automatisierten Unit-Tests aus, erstellt aktuelle Binary und fügt diese zu einer Anwendung zusammen. Mit Jenkins [1], Hudson [2] und Bamboo [3] gibt es erprobte Software, um eigene CI-Server zu betreiben. Der Aufwand bei der Einrichtung und Pflege dieser Server kann eine Hürde darstellen, sofern die Ressourcen und das Know-how nicht im eigenen Haus zur Verfügung stehen. Diese Problematik lösen Cloud-hosted CI-Anbieter wie CloudBees [4] und Travis CI [5]. Dieser Artikel stellt diese Plattformen vor und geht dabei auf deren Möglichkeiten und Einschränkungen ein.

Cloud-hosted Continuous Integration

Soll ein CI-Server in einem Projekt eingesetzt werden, fällt die Wahl meistens auf Jenkins. Dieser lässt sich kostenfrei verwenden, bringt aber in seiner Grundausstattung nur wenige Features mit. Kommen zum Beispiel neben Subversion weitere Versionierungssysteme zum Einsatz, ist diese Funktionalität durch Plug-ins zu ergänzen.

Eine wesentliche Voraussetzung für den Betrieb eines Jenkins CI-Servers ist die Bereitstellung der notwendigen Hardware. Gerade wenn verschiedene Builds oder Testläufe den CI-Server exklusiv nutzen, sind mehrere Server notwendig, um einen Mindestgrad an Parallelität zu ermöglichen und somit Zeit für den Gesamttablauf einzusparen. Werden mehrere Slaves be-

nötigt, kann das abhängig von den internen Beschaffungsvorgängen für Hardware die Etablierung einer CI-Infrastruktur ausbremsen. Außerdem wird das Setup der Umgebung durch solche Slaves oder die benötigten Plug-ins zunehmend komplexer und aufwändiger.

Genau hier kommen die Vorteile einer Cloud-basierten CI-Lösung zum Tragen. Je nach verwendetem Produkt werden – wie in einer Cloud üblich – die benötigten Ressourcen bei Bedarf zugeschaltet oder wieder freigegeben. In diesem Bereich existieren bereits mehrere Anbieter, die solche Dienste zur Verfügung stellen [6]. Darunter gibt es auch kostenfreie Angebote, die sich durch Einschränkungen von den kostenpflichtigen Produkten unterscheiden. Einige Anbieter stellen eine komplette Infrastruktur für den Einsatz von Continuous Delivery zur Verfügung. Andere hingegen ermöglichen lediglich die Ausführung eines Build. Zwei populäre Anbieter von Cloud-basierten CI-Lösungen sind CloudBees und Travis CI, deren Angebote sich deutlich voneinander unterscheiden und im Folgenden vorgestellt werden.

CloudBees-Plattform

Die CloudBees-Plattform bietet umfangreiche Cloud-basierte Möglichkeiten zur Unterstützung der Anwendungsentwicklung. Dazu zählen Angebote wie die Bereitstellung von Datenbanken, Application- oder Continuous-Integration-Servern. Das Angebot setzt sich aus zwei Paketen zusammen. DEV@cloud bietet als CI-Umgebung eine Jenkins-Instanz, die in der Cloud

gehostet wird. RUN@cloud stellt eine Plattform zur Verfügung, um entwickelte Anwendungen zu hosten. Das Angebot richtet sich vor allem an Entwickler von Java-Applikationen. Die Plattform unterstützt ebenso Programmiersprachen wie PHP und Ruby. Auch populäre Frameworks wie Play! oder Ruby on Rails sind im Angebot. Zudem kann der entwickelte Code direkt in den eigenen Versionierungssystemen verwaltet werden. CloudBees bietet somit ein Komplettpaket, um Continuous Delivery für eigene Projekte einzurichten.

Als CI-Server stellt CloudBees eine Jenkins-Installation bereit, die sich frei konfigurieren lässt. Dies bietet ein maximales Maß an Flexibilität, da keine Vorgaben zur Konfiguration der Build-Jobs bestehen. Allerdings kann die Konfiguration von Jenkins sehr komplex und aufwändig werden. CloudBees bietet so die gewohnte CI-Umgebung, bei der der Administrator vor allem durch das Hosting in der Cloud profitiert, da keine eigene Infrastruktur vorgehalten werden muss. Kosten entstehen neben niedrigen Fixkosten vor allem durch die tatsächliche Nutzung, wie es bei Cloud-Produkten üblich ist.

Die Bereitstellung von Jenkins-Instanzen für Kunden bietet den Vorteil, dass in vielen Unternehmen bereits Know-how zu Jenkins existiert und daher die Administration selbst durchgeführt werden kann. Der Aufwand für die Bereitstellung des CI-Servers reduziert sich mit CloudBees auf wenige Minuten, da dieser „on demand“ erzeugt werden kann. Neben der Cloud-hosted CI-Umgebung wird für Unterneh-

men auch ein Enterprise Jenkins zur Verfügung gestellt. Dieses Angebot richtet sich an Unternehmen, die ihre CI-Infrastruktur nicht in der Cloud vorhalten wollen, sondern eigene Hardware verwenden möchten. Von CloudBees gestellte Fachleute können auf dieser Infrastruktur dann die CI-Umgebung installieren, an die Kundenbedürfnisse anpassen und administrieren.

Neben den Enterprise-Kunden können auch Open-Source-Entwickler und kleinere Projekte von der Plattform profitieren, sie richtet sich jedoch hauptsächlich an Enterprise-Kunden. So wird beim kostenlosen Angebot etwa die eigene Jenkins-Instanz bei zu langer Inaktivität in einen Hibernate-Modus versetzt. Beim nächsten Aufruf muss die Instanz erst reaktiviert werden, was mehrere Minuten dauern kann. Dies führt für Gelegenheitsnutzer zu Wartezeiten; Kunden mit intensiver Nutzung haben dieses Problem nicht.

Travis CI

Travis CI bietet im Gegensatz zu CloudBees eine Continuous-Integration-Plattform, die nur für Projekte, die auf GitHub gehostet werden, zur Verfügung steht. Jedes öffentliche Projekt auf GitHub kann das Angebot kostenfrei nutzen. Zur Verwendung muss sich der Benutzer mittels GitHub-Account bei Travis anmelden und kann dort seine Projekte zur kontinuierlichen Integration einrichten. Die Software hinter Travis ist auf GitHub unter einer Open-Source-Lizenz verfügbar [7]. Die Anwendung ist jedoch bislang nicht dazu geeignet, in eigenen Umgebungen installiert zu werden, da hierfür weder ein Softwarepaket noch die Dokumentation bereitgestellt wird. Die Komponenten der Anwendung sind

jedoch als eigene GitHub-Projekte veröffentlicht.

Travis CI richtete sich anfangs vor allem an Open-Source-Entwickler, da die Möglichkeit, private Repositories zu integrieren, erst mit einem kommerziellen Angebot etabliert wurde. Durch die einfache Konfiguration und die enge Integration mit GitHub konnte Travis CI schnell an Popularität gewinnen. Zu dieser Integration gehört die Option, Status-Informationen des letzten Build zum Beispiel auf der GitHub-Seite seines Projekts zu platzieren [8].

Travis bindet auch weitere Teile des GitHub-API an. So werden unter anderem auf GitHub erstellte Pull-Requests eines Projekts automatisch getestet. Hierdurch können die Entwickler einer Software schnell erkennen, ob die Änderungen eines Mitentwicklers problemlos in das eigene Projekt integriert werden können.

Der Build- und Test-Prozess für Travis wird über die Konfigurationsdatei „.travis.yml“ an das eigene Projekt angepasst. Somit lassen sich Build-Jobs mit geringem Aufwand einrichten und in die Cloud auslagern. Allerdings bietet diese Konfigurationsdatei weniger Flexibilität als ein

eigener CI-Server wie im Angebot von CloudBees. Dafür entfällt aber auch die Bereitstellung einer eigenen Umgebung. Im Zusammenspiel mit der Anbindung an GitHub bekommt der Benutzer so die Möglichkeit, sehr schnell Continuous Integration für dort gehostete Projekte einzurichten.

Konfiguration eines Travis-Projekts

In der Konfigurationsdatei „.travis.yml“ kann der Benutzer Angaben zum Build-Tool, zur Programmiersprache, zu Umgebungsvariablen etc. machen [9]. Wann ein Build ausgeführt werden muss, erfährt Travis über das API von GitHub. Builds werden bei neuen Commits in das eigene Repository oder bei Pull-Requests gestartet. Eine Möglichkeit, diesen Prozess selbst anzustoßen, gibt es nicht.

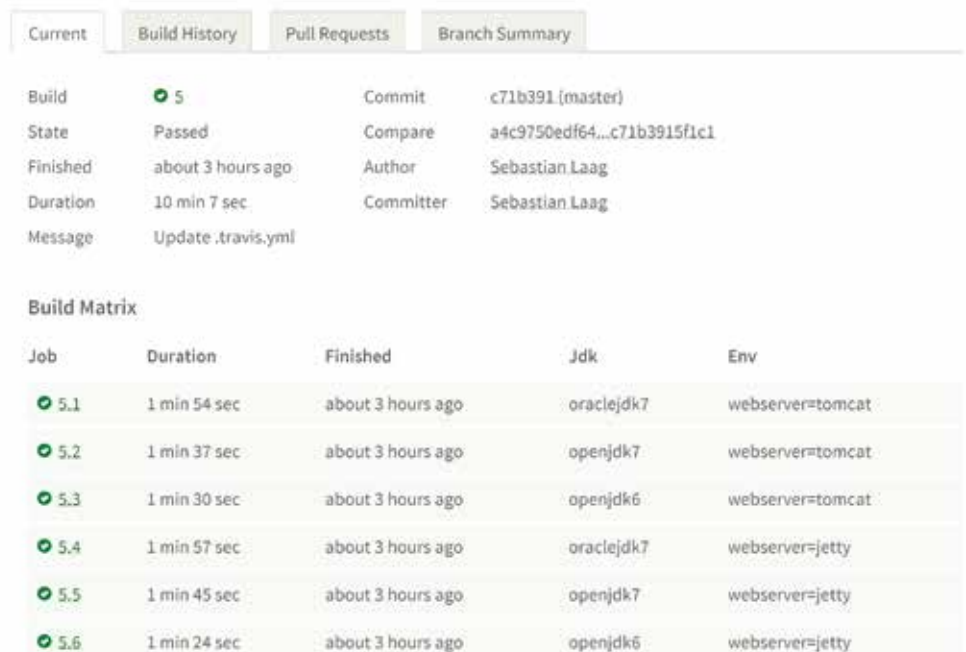
Über die Konfigurationsdatei lässt sich der komplette Build-Prozess definieren. Neben der Angabe zur Programmiersprache kann der Benutzer auch auf Skripte verweisen, die vor oder nach dem Build ausgeführt werden. Ebenso lässt sich das Kommando zum Starten der Testsuite konfigurieren, sofern dies nicht über das Build-

```
language: java
jdk:
  - oraclejdk7
  - openjdk7
  - openjdk6
env:
  - webserver=tomcat
  - webserver=jetty
```

Listing 1: „.travis.yml“-Konfiguration für Java-Projekte

sebastianlaag/occi4java

RESTful OCCi 4 Java



Build	State	Finished	Duration	Message	Commit	Compare	Author	Committer
5	Passed	about 3 hours ago	10 min 7 sec	Update .travis.yml	c71b391 (master)	a4c9750edf64...c71b3915f1c1	Sebastian Laag	Sebastian Laag

Job	Duration	Finished	Jdk	Env
5.1	1 min 54 sec	about 3 hours ago	oraclejdk7	webserver=tomcat
5.2	1 min 37 sec	about 3 hours ago	openjdk7	webserver=tomcat
5.3	1 min 30 sec	about 3 hours ago	openjdk6	webserver=tomcat
5.4	1 min 57 sec	about 3 hours ago	oraclejdk7	webserver=jetty
5.5	1 min 45 sec	about 3 hours ago	openjdk7	webserver=jetty
5.6	1 min 24 sec	about 3 hours ago	openjdk6	webserver=jetty

Abbildung 1: Übersicht der Build-Matrix für einen Java-Build mit Oracle JDK 7 und OpenJDK 7

Open Source	Startup	Small Business	Enterprise
Faire Nutzung	zwei gleichzeitige Jobs	fünf gleichzeitige Jobs	zehn gleichzeitige Jobs
Unbegrenzte Build-Minuten	unbegrenzte Build-Minuten	unbegrenzte Build-Minuten	unbegrenzte Build-Minuten
Unbegrenzte Anzahl Repositories	unbegrenzte Anzahl Repositories	unbegrenzte Anzahl Repositories	unbegrenzte Anzahl Repositories
kostenfrei	129 \$/Monat	249 \$/Monat	489 \$/Monat

Tabella 1: Travis-CI-Preise [12]

Tool geschieht. Sind externe Ressourcen wie Datenbanken, Caches oder anderes erforderlich, stellt Travis im Rahmen der Build-Umgebung viele Dienste direkt zur Verfügung. Dazu zählen MySQL, MongoDB, Redis, Memcached, Cassandra und Elasticsearch.

Bei Travis ist die Konfigurationsdatei im Repository auf GitHub abgelegt und somit bei öffentlichen Projekten für jeden einsehbar. Dies schafft neben Transparenz allerdings auch Sicherheitsprobleme, falls die Build-Umgebung Zugangsdaten oder andere vertrauliche Angaben benötigt. Um dies zu lösen, generiert Travis RSA-Schlüssel zur Verschlüsselung dieser Daten in der Konfigurationsdatei.

Bei der Angabe unterschiedlicher Konfigurationen zur Laufzeit-Umgebung sowie zu den Umgebungs-Variablen und Build-Konfigurationen erstellt Travis automatisch eine Build-Matrix, um alle diese Konfigurationen miteinander zu kombinieren. Bei der Angabe von drei JDKs und vier unterschiedlichen Konfigurationen der Umgebungsvariablen führt Travis daher bereits zwölf Builds aus. Hier weist der Hersteller darauf hin, dass der Benutzer keine großen Matrizen erstellen soll, da die Ressourcen für das kostenfreie Angebot begrenzt sind. Zudem benötigen große Konfigurationen viel Laufzeit, da nur ein Build parallel ausgeführt werden kann. Für die Ausführung mehrerer gleichzeitiger Builds muss der Nutzer die kostenpflichtige Version Travis Pro verwenden.

Java-Projekte mit Travis bauen

Listing 1 zeigt ein einfaches Beispiel zum Bauen eines Java-Projekts. Die dort beschriebene Konfigurationsdatei weist Travis an, das Java-Projekt mit OpenJDK in Version 6 und 7 sowie Oracle JDK in

Version 7 zu bauen. Travis führt daraufhin einen eigenen Build-Job für jedes konfigurierte JDK aus. Das Oracle JDK 6 steht nicht zur Verfügung, da es seit Ende 2012 nicht mehr von Oracle unterstützt wird. Travis rät dazu, das Oracle JDK 7 zu benutzen, da es abwärtskompatibel ist, und ermutigt so zu einem Umstieg auf Java 7.

Das genutzte Build-Tool ermittelt Travis automatisch anhand der Existenz der Dateien „pom.xml“ (Maven) oder „build.gradle“ (Gradle). Wird keine der beiden Dateien gefunden, versucht Travis, das Projekt mit Ant zu bauen. Da bei Ant kein standardisierter Weg zur Installation von Abhängigkeiten existiert, muss dies zusätzlich in der Konfigurationsdatei angegeben werden. Als Standard werden bei Maven die Befehle „mvn install -DskipTests=true und mvn test“ beziehungsweise in Gradle-Projekten „gradle assemble“ und „gradle check“ zur Installation von Abhängigkeiten und zum Ausführen der Test-Suite genutzt.

Abbildung 1 zeigt den Einsatz einer Build-Matrix, die sich für Java-Projekte anbietet, um gegen mehrere JDKs zu testen. Dabei können auch unterschiedliche Einstellungen durch Umgebungsvariable gesetzt sein (siehe Listing 1). Für größere Projekte wird die Einschränkung von einem parallelen Build und einer maximalen Laufzeit von siebzig Minuten schnell spürbar. Diese Einschränkungen gelten jedoch nicht für das kommerzielle Angebot Travis Pro.

Die Zahlungsmodelle im Vergleich

CloudBees und Travis CI verfolgen sehr unterschiedliche Ansätze bei der Gestaltung ihrer Zahlungsmodelle. Travis CI lässt sich bei Verwendung eines öffentlichen GitHub-Repository unbegrenzt häufig kostenfrei nutzen, da es selbst zu einem gro-

ßen Teil durch Crowdfunding und größere Sponsoren finanziert wird [10]. Der Benutzer muss jedoch mit einigen Einschränkungen leben. Es kann nur ein Build gleichzeitig ausgeführt werden und es gibt keine Garantie dafür, wann dieser erfolgt. In eigenen Versuchen hat sich jedoch gezeigt, dass nach maximal fünfzehn Minuten Wartezeit der eigene Job ausgeführt wird. Auf CloudBees kann ebenfalls nur ein Job gleichzeitig ausgeführt werden, allerdings sind nur die ersten dreihundert Minuten Build-Zeit innerhalb eines Monats gratis.

Wer nicht mit den Einschränkungen arbeiten möchte, kann die kostenpflichtigen Angebote der beiden Plattformen nutzen. Dabei bieten auch hier beide unterschiedliche Ansätze an (siehe Tabellen 1 und 2). Während bei Travis bei jedem Angebot die Anzahl der Build-Minuten, die Anzahl der Repositories und die Anzahl der Benutzer unbeschränkt ist, kann bei Bezahlung lediglich die Anzahl gleichzeitiger Jobs beeinflusst werden. Zudem fallen bei Benutzung privater Repositories zusätzliche Kosten bei GitHub an.

CloudBees bietet bei Bezahlung eine unbegrenzte Anzahl gleichzeitiger Jobs an. Die monatlichen Kosten variieren zwischen hundert und zweihundert Dollar, je nach gewähltem Angebot. Zusätzlich kostet jede Stunde Build-Zeit je nach Größe der verwendeten VM 0,106 Dollar pro Stunde, beziehungsweise 0,425 Dollar je Stunde. Die Anzahl der Benutzer und des Repository ist hier allerdings auf maximal fünfzig Benutzer beziehungsweise 50 GB Speicherplatz für das Repository beschränkt. Zudem bietet CloudBees in der Enterprise-Version ein Plug-in an, das die Verfügbarkeit, Sicherheit und Administration großer Installationen verbessert [11]. Die Verfügbarkeit wird durch die Verwen-

	Free	Pro	Enterprise
Monatsrate	-	100 \$	200 \$
m1.small build minutes	300 Minuten/Monat	0,106 \$/Stunde	0,106 \$/Stunde
m1.large build minutes	steht nicht zur Verfügung	0,425 \$/Stunde	0,425 \$/Stunde
Parallele Builds	1	unbegrenzt	unbegrenzt
Benutzer	3	20	50
Repository	2 GB	25 GB	50 GB
CloudBees Enterprise Plug-in	-	-	vorhanden
Externe Slaves	-	-	20\$/Monat/Slave

Tabella 2: CloudBees-Preise [13]

derung von redundanten Jenkins-Installationen erhöht.

Fazit

Bei näherer Betrachtung der bisher vorgestellten Anbieter wird deutlich, dass beide ein sehr unterschiedliches Angebot von Cloud-hosted Continuous Integration zur Verfügung stellen. CloudBees ist mehr auf die Verwendung im Enterprise-Umfeld ausgerichtet, da durch die eigene CI-Umgebung die Konfiguration sowie die Einrichtung des CI-Servers in eigener Hand liegen. Zudem sind durch das Enterprise-Plug-in Verfügbarkeit sowie Sicherheit für Unternehmen gewährleistet. Die Verwendung eigener Slaves bietet die Möglichkeit, diese je nach Bedarf einzurichten und zu verwenden. Das Angebot wird durch RUN@cloud um Komponenten für Continuous Delivery komplettiert.

Travis CI hingegen bietet anstelle einer eigenen CI-Umgebung eine Plattform, für die Jobs durch minimalen Aufwand definiert werden können. Hier hat der Anwender keinen direkten Zugriff auf den verwendeten CI-Server (und darauf, wann ein Build ausgeführt wird) und die feingranulare Konfiguration der Umgebung. Dies muss über die Konfiguration von Builds oder mit ausgelieferten Skripten geschehen. Zudem existiert auch nur ein monatliches Zahlungsmodell, das sich nicht nach der Nutzung richtet, sondern nur Auswirkung auf die gleichzeitig ausführbaren Jobs hat.

Auch sicherheitstechnisch zeigen sich gravierende Unterschiede. Da Travis in der kostenfreien Variante nur öffentliche Git-

Hub-Repositories zulässt, muss der eigene Quellcode offengelegt werden. In den bezahlten Modellen ist der Benutzer auf GitHub als Hosting-Plattform angewiesen. CloudBees-Nutzer können sicherheitskritische Daten beliebig speichern und durch die Verwendung von Jenkins selbstständig konfigurierbar auf diese zugreifen. Das generelle Problem der Speicherung von unternehmenskritischen Daten in der Infrastruktur eines Cloud-Anbieters bleibt jedoch.

Cloud-hosted Continuous Integration erweist sich als interessanter Ansatz, um CI für eigene Projekte schnell einrichten zu können. Die vorgestellten Angebote unterscheiden sich allerdings grundlegend und richten sich an unterschiedliche Benutzergruppen. Ob der jeweilige Anbieter den eigenen Anforderungen entspricht, kann mit den kostenlosen Angeboten getestet werden.

Weitere Informationen

- [1] <http://jenkins-ci.org>
- [2] <http://hudson-ci.org>
- [3] <https://www.atlassian.com/software/bamboo>
- [4] <http://www.cloudbees.com>
- [5] <http://travis-ci.com>
- [6] <http://blog.testnoir.com/ciaas-roundup-11-hosted-continuous-integration-services>
- [7] <https://github.com/travis-ci>
- [8] <http://about.travis-ci.org/docs/user/status-images>
- [9] <http://about.travis-ci.org/docs/user/build-configuration>
- [10] <https://love.travis-ci.org>
- [11] <http://www.cloudbees.com/jenkins-enterprise-by-cloudbees-available-plugins.cb>
- [12] <http://travis-ci.com/plans>
- [13] <http://www.cloudbees.com/platform/pricing/devcloud.cb>

Sebastian Herbermann
sebastian.herbermann@adesso.de



Sebastian Laag
sebastian.laag@adesso.de



Sebastian Herbermann (cand. Dipl. Inf., Univ.) und Sebastian Laag (Dipl. Inf., Univ.) sind beide als Software-Engineer bei der adesso AG in Dortmund tätig und arbeiten dort als Java-Entwickler mit Schwerpunkt auf Web-Anwendungen.

Überraschungen und Grundlagen bei der nebenläufigen Programmierung in Java

Christian Kumpke, Netpioneer GmbH

Java hält seit den Anfängen alles bereit, was für die nebenläufige Programmierung wichtig ist: Threads, die Keywords „synchronized“ und „volatile“ sowie ein Memory Model. Dabei ist „volatile“ wohl das Keyword, um das sich die meisten Mythen ranken. Dieser Artikel zeigt, was dieses Keyword genau bedeutet und welche Auswirkungen es auf die Ausführung eines Programms in der Java Virtual Machine (JVM) und auf die Laufzeit-Optimierungen in deren Just In Time Compiler (JIT) hat.

In den meisten Java-Projekten wird Nebenläufigkeit nicht gezielt eingebaut, sondern ist eher eine Randbedingung, die man immer mal wieder beachten muss. Ein Beispiel aus dem Projekt-Alltag ist ein automatischer Twitter-Mechanismus, der für einen Kunden an ein großes Content Management System (CMS) angeschlossen wird. Dabei sollen die URLs von neu erstellten Artikeln beim Veröffentlichen immer automatisch mit der zugehörigen Überschrift getwittert werden. Dazu prüft eine Schleife regelmäßig auf neue Artikel. Wird ein solcher gefunden, erfolgt eine Reihe von automatisierten Prozessen, unter anderem die Übergabe des Artikels an den Twitter-Service.

Als eine Art „Not-Aus“ soll das Twittern über ein Flag abschaltbar sein. Dieses wird vor dem Aufruf des Twitter-Service überprüft und soll über die Java Management Extensions (JMX) geschaltet werden können. Hier ist die Nebenläufigkeit vielleicht nicht offensichtlich. Allerdings läuft der JMX-Connector, der die JMX-Bean aufruft, in einem eigenen Thread, während unsere Schleife möglicherweise im Haupt-Thread der Anwendung läuft. Damit sind mindestens zwei Threads an der korrekten Funktion des geschilderten Beispiels beteiligt.

Doch wo liegen die Herausforderungen bei nebenläufigen Programmen? Antworten und weitere Details zu dem Thema finden sich unter anderem in [1]. In diesem Artikel wird die Frage nur soweit beantwortet, wie sie für das weitere Verständnis des Inhalts notwendig ist.

Solange sich innerhalb eines Programms jeder Thread unabhängig von den

anderen nur um seine eigenen Aufgaben kümmert, sind keine weiteren Vorkehrungen zu treffen. Vorsicht ist immer dann geboten, wenn Threads auf geteilte Ressourcen zugreifen, etwa auf gemeinsame Variablen.

Wenn aber der eine Thread die Nachrichten zu Twitter schickt und der JMX Connector Thread nur sein eigenes „isTwitterActive“-Flag beschreibt, wäre der geforderte „Not-Aus“ funktionslos. Damit das Ganze funktioniert, brauchen beide Threads das Flag als gemeinsame Variable.

Die Überraschung

Für die weiteren Untersuchungen wird das Szenario auf das im Listing 1 gezeigte einfache Beispiel „SimpleExample“ reduziert. Dabei wird beim Programmstart zunächst ein neuer „Looper“-Thread erstellt und gestartet. Dieser durchläuft nun in der „run“-Methode seine „while“-Schleife so lange, bis das Flag „finish“ auf „true“ steht. Die „main“-Methode legt sich nun für eine Sekunde schlafen, setzt anschließend das Flag „finish“ vom erzeugten „Looper“-Thread auf „true“ und wartet mit „join()“ auf

```
class SimpleExample {
    static class Looper extends Thread {
        boolean finish = false;

        @Override public void run() {
            while (!finish) {
                // do something
            }
        }
    }

    public static void main(String[] args) throws Exception {
        Looper looper = new Looper();
        looper.start();
        Thread.sleep(1000); // wait 1s
        looper.finish = true;
        System.out.println("Wait for Looper to terminate...");
        looper.join();
        System.out.println("Done.");
    }
}
```

Listing 1: SimpleExample.java (zum Download unter [6])

```
while (!finish) {
    System.out.println("finish is " + finish);
}
```

Listing 2: Debug-Ausgabe

```
$ java -XX:+UnlockDiagnosticVMOptions \
  -XX:+PrintAssembly \
  -XX:+DebugNonSafepoints SimpleExample
Loaded disassembler from hsdis-amd64.so
Decoding compiled method 0x00007feccd05fb90:
Code:
[Disassembling for mach='i386:x86-64']
...
0x00007feccd05fd26: test    %eax,0xb6812d4(%rip)
0x00007feccd05fd2c: jmp    0x00007feccd05fd26
...
```

Listing 3: Maschinen-Code

dessen Beendigung. Im Anschluss gibt das Programm noch kurz eine Bestätigung aus und terminiert.

Beim Start des Programms würde man in der Konsole nun zunächst die beiden Zeilen „\$ java SimpleExample“ und „Wait for Looper to terminate...“ zu sehen bekommen und nach etwa einer Sekunde die Ausgabe „Done.“ erwarten. Anschließend sollte sich das Programm beenden.

Die Realität ist jedoch stark vom verwendeten System abhängig. Mit einer aktuellen Java-Version auf einem 64-Bit-System wird das Programm aller Voraussicht nach nicht terminieren. Während die „main“-Methode auf den „Looper“-Thread wartet, wird dieser endlos in seiner „while“-Schleife hängen bleiben. Auf einem 32-Bit-System mit weniger als 2 GB Hauptspeicher wird das Programm wahrscheinlich wie gewünscht funktionieren. Die Ursache dafür wird später noch genauer erläutert.

Der nächste Schritt wäre, das Programm zur Fehlersuche im Debugger zu starten. Sitzt der Breakpoint etwa in Zeile 6 im Kopf der „while“-Schleife, wird der Debugger, nachdem er das Programm angehalten hat, behaupten, dass „finish“ nach der verstrichenen Sekunde tatsächlich den Wert „true“ hat. Lässt man das Programm weiterlaufen, wird es wie gewünscht terminieren. Als Nächstes wäre eine Debug-Ausgabe in die Schleife zu bauen (siehe Listing 2).

Wird das Programm erneut gestartet, bekommt man einige Male die Ausgabe „finish is false“ zu sehen und anschließend beendet das Programm. Damit es auch ohne Debugger und Debug-Ausgabe korrekt funktioniert, muss die Variable „finish“ als „volatile“ deklariert sein: „volatile boolean finish = false;“. Damit ist sichergestellt, dass Änderungen an der Variable durch den einen Thread auch in anderen Threads ankommen.

Der JIT-Compiler

In der ursprünglichen, fehlerhaften Variante des Programms kommt die Änderung am Flag nicht im „Looper“-Thread an. Hier werden oft Prozessor-Caches als Ursache vermutet, die den Wert zwischenspeichern, anstatt ihn in den Hauptspeicher zu schreiben. Allerdings lässt sich das Verhalten auch problemlos auf einem Single-Prozessor-System nachstellen, bei dem alle Threads den gleichen Prozessor-Cache verwenden. Die eigentliche Ursache für das beobachtete Verhalten sind die Optimierungen des JIT-Compilers.

Beim Kompilieren wird ein Java-Programm zunächst in Java-Byte-Code übersetzt und bei der anschließenden Ausführung von der Java Virtual Machine (JVM) anfangs über einen Interpreter abgearbeitet. Die interpretierte Ausführung ist im Gegensatz zu Programmen, die direkt

in Maschinen-Code kompiliert sind, recht zeitaufwändig. Aus diesem Grund überwacht die JVM, welche Stellen im Code sehr oft ausgeführt werden, die sogenannten „Hot Spots“. Diese werden dann vom JIT-Compiler in direkt ausführbaren Maschinen-Code übersetzt. Dabei werden auch zahlreiche Optimierungen vorgenommen, die den aktuellen Zustand des Programms berücksichtigen.

Sind für die Optimierung falsche Annahmen getroffen worden oder springt der Debugger etwa in einen Breakpoint, können diese Optimierungen auch wieder verworfen werden und der Byte Code wird wieder interpretiert, bis die Stelle erneut als Hot Spot erkannt wird.

In „SimpleExample“ aus Listing 1 ist die „while“-Schleife des „Looper“-Thread ein solcher Hot Spot. Beim Übersetzen in Maschinen-Code wird der JIT-Compiler feststellen, dass die Variable „finish“ innerhalb der Schleife nicht verändert wird. Da sie nicht als „volatile“ deklariert ist, muss bei der Optimierung auch nur der aktuelle Thread berücksichtigt werden. Daher kann die Überprüfung von „finish“ während des Schleifendurchlaufs komplett entfallen und Änderungen an der Variablen haben keinen Einfluss mehr auf das Verhalten der „while“-Schleife.

Hält der Debugger das Programm jedoch innerhalb der Schleife an einem Breakpoint an, werden durchgeführte Optimierungen verworfen. Damit wird die Variable wieder überprüft und der „Looper“-Thread wird sich beenden, nachdem „finish“ in der „main“-Methode auf „true“ gesetzt wurde.

Das Java Memory Model

Optimierungen kann und darf der JIT-Compiler nicht willkürlich vornehmen. Die Regeln, innerhalb derer Optimierungen möglich sind, geben die Java Language Specification (JLS) und die Java Virtual Machine Specification (JVMS) [3] vor. Ein entscheidender Teil für die Bedeutung von „volatile“ ist dabei das darin enthaltene Java Memory Model. Mit seinen acht „happens-before“-Regeln legt es die Reihenfolge fest, in der Aktionen innerhalb eines Thread und zwischen verschiedenen Threads auszuführen sind. Eine Aktion kann dabei ungefähr mit einer Anweisung im Java-Programm gleichgesetzt werden.

Abbildung 1 zeigt die acht Regeln des Java Memory Model. Im weiteren Verlauf werden einzelne Regeln, die im aktuellen Beispiel von Bedeutung sind, näher betrachtet. Genauere Erklärungen zu allen Regeln finden sich in [1].

Ohne „volatile“ muss die JVM bei der Ausführung nur die „Program Order Rule“ beachten. Da „finish“ innerhalb der Schleife nicht verändert wird, muss der Wert nur einmal überprüft werden und die Überprüfung innerhalb der Schleife kann entfallen. Ist „finish“ als „volatile“ deklariert, kommt zusätzlich die „Volatile Variable Rule“ ins Spiel. Diese verlangt, dass ein Schreibzugriff auf eine „volatile“-Variable vor dem nächsten Lesezugriff passieren muss.

Zusammen mit der „Program Order Rule“ und der „Transitivity“ bedeutet dies für das „SimpleExample“, dass der Schreibzugriff in Zeile 13 abgeschlossen sein muss, bevor die Variable „finish“ das nächste Mal gelesen wird. Mit anderen Worten: Wird „finish“ in der „main“-Methode auf „true“ gesetzt, muss diese Veränderung im nächsten Schleifendurchlauf im „Looper“-Thread sichtbar sein. Daher verbietet sich in diesem Fall die „Weg“-Optimierung der Überprüfung des Flag innerhalb der Schleife.

Je nach Rechnerarchitektur muss die JVM bei Schreibzugriffen auf „volatile“-Variablen anschließend die Prozessor-Caches leeren (Cache Flush), um den geänderten Wert in den Hauptspeicher zu bringen. Für das hier gezeigte Fehlverhalten bei fehlendem „volatile“ waren Prozessor-Caches aber nicht die Ursache.

Mit der „Monitor Lock Rule“ lässt sich außerdem erklären, warum eine Debug-Ausgabe in Zeile 7 ebenso für ein korrektes Funktionieren des Programms sorgt, ohne dass „finish“ als „volatile“ deklariert wird. „System.out“ ist ein „PrintStream“, dessen Methode „println“ einen „synchronized“-Block enthält. Da sowohl in der „main“-Methode als auch in der Schleife das gleiche „PrintStream“-Objekt verwendet wird, arbeiten beide Threads mit demselben Monitor-Lock. Damit muss folgende Aktionsreihenfolge garantiert sein: In der „main“-Methode wird „finish“ auf „true“ gesetzt und anschließend durch eine Ausgabe auf die Konsole auf den Lock zugegriffen.

Wird jetzt innerhalb der Schleife das nächste Mal der Wert von „finish“ auf die Konsole ausgegeben, wird auf densel-

```
$ java -XX:+UnlockDiagnosticVMOptions \
  -XX:+PrintAssembly \
  -XX:+DebugNonSafepoints \
  SimpleExample
Loaded disassembler from hsdis-amd64.so
Decoding compiled method 0x00007fb65105fb90:
Code:
[Disassembling for mach='i386:x86-64']
...
0x00007fb65105fd30: movzbl 0x68(%rbx),%r11d
0x00007fb65105fd35: test   %eax,0xbbf52c5(%rip)
0x00007fb65105fd3b: test   %r11d,%r11d
0x00007fb65105fd3e: je     0x00007fb65105fd30
...
```

Listing 4: Geänderte Ausgabe

ben Monitor-Lock zugegriffen. Jetzt muss die JVM garantieren, dass der Schreibzugriff auf „finish“ abgeschlossen und beim nächsten Lesen der Variable sichtbar ist, bevor die Ausgabe auf die Konsole erfolgt. Damit wird der „Looper“-Thread die „while“-Schleife beim nächsten Durchlauf verlassen und das Programm terminieren.

Unterschiedliche JVMs

Das Entfernen nicht benötigter Überprüfungen ist nur eine von vielen Optimierungen, die eine moderne JVM und ihr JIT-Compiler zur Laufzeit durchführen. Viele weitere finden sich unter anderem in [2] und [5]. Welche Optimierungen und wie aggressiv diese durchgeführt werden, hängt auch vom umgebenden System ab.

Auf einer 32-Bit-Intel-Architektur stehen beim Java Runtime Environment (JRE) von Oracle zwei verschiedene JVM-Implementierungen zur Verfügung: Client („java -client“) und Server („java -server“). Auf Systemen mit weniger als 2 GB Hauptspeicher wird standardmäßig die Client-VM verwendet. Diese führt die beschriebene Optimierung nicht durch. Daher wird dort auch das fehlerhafte Programm ohne „volatile“ funktionieren. Auf Systemen mit mehr als 2 GB Hauptspeicher oder 64-Bit-Systemen wird standardmäßig die Server-VM verwendet und dort kann man den beschriebenen Fehler beobachten.

Dem JIT-Compiler auf die Finger geschaut

Zuletzt noch ein genauer Blick auf das Ergebnis der Optimierungen des JIT-Com-

piler. Dazu ist das HotSpot Disassembler Plug-in [4] notwendig. Mit dessen Hilfe gibt der JIT-Compiler den generierten Maschinen-Code auf der Konsole aus. Listing 3 zeigt den generierten Maschinen-Code, wobei hier nicht relevante Teile weggelassen wurden.

Zu sehen ist der innere Kern der „while“-Schleife nach der Optimierung. Zunächst ein Vergleich der Speicherstelle „0xb6812d4(%rip)“ mit dem Register „%eax“. Anschließend ein unbedingter Sprung „jmp“ zurück zum Vergleich. Der gezeigte Vergleich, dessen Ergebnis in jedem Fall ignoriert wird, ist nicht der Vergleich mit „finish“, sondern dient der JVM zum Erreichen eines Safepoint für die Garbage Collection [2]. Entscheidend sind der unbedingte Rücksprung zum Vergleich und die daraus resultierende Endlosschleife. Ist „finish“ jetzt korrekt als „volatile“ deklariert, erhält man folgende geänderte Ausgabe (siehe Listing 4).

Zu Beginn wird jetzt der aktuelle Wert von „finish“ in das Register „%r11d“ geladen. Anschließend erfolgt wieder der „%eax“-Vergleich für interne Zwecke. Danach wird „%r11d“ auf „false“ getestet und der anschließende Sprung wird nur ausgeführt, wenn „%r11d“ tatsächlich den Wert „false“ hatte. Damit wird der „Looper“-Thread die Schleife verlassen, sobald „finish“ auf „true“ gesetzt ist.

Fazit

Die nebenläufige Programmierung mit Java ist eine spannende Sache und es

lohnt sich, auf die technischen Grundlagen zu schauen. Der Blick an die Stelle, an der Java Byte Code zu performantem Maschinen-Code wird, fördert interessante Details zutage, die einem als Java-Entwickler normalerweise verborgen bleiben. Dieser Weg ist für die alltägliche Fehlersuche sicher nicht praktikabel, liefert aber einen interessanten Einblick in die Tiefen der JVM und deren Optimierungsmöglichkeiten.

Außerdem wurde die Rolle des Java Memory Model und des Keyword „volatile“ bei der nebenläufigen Programmierung mit Java verdeutlicht. Das mag den einen oder anderen Denkanstoß für die Fehlersuche in nebenläufigen Programmen geben. Wenn etwa der Wert einer Variablen mal wieder nicht der zu sein scheint, der er sein sollte, oder wenn der neue Artikel in unse-

rem Beispiel trotz „Not-Aus“ immer noch getwittert wird.

Literatur

- [1] „Java Concurrency In Practice“, Brian Goetz, Tim Peierls, Joshua Bloch und weiteren, erschienen im Addison Wesley Verlag, ISBN 0-321-34960-1
- [2] „Java Performance“, Charlie Hunt und Binu John, erschienen im Addison Wesley Verlag, ISBN 0-137-14252-8

Links

- [3] Java SE Specifications: <http://docs.oracle.com/javase/specs>
- [4] HotSpot Disassembler Plug-in: <https://kenai.com/projects/base-hsdis>
- [5] HotSpot Internals: <https://wikis.oracle.com/display/HotSpotInternals>
- [6] Downloads zum Artikel: <http://www.netpioneer.de/de/java-aktuell-artikel.html>

Christian Kumpe

christian.kumpe@netpioneer.de



Christian Kumpe studierte Informatik am Karlsruher Institut für Technologie (KIT) und sammelte bereits während seines Studiums als Freelancer Erfahrung in diversen Java-Projekten. Seit dem Jahr 2011 arbeitet er als Software-Entwickler bei der Netpioneer GmbH in Karlsruhe. Seine aktuellen Themenschwerpunkte sind Java-basierte Portale und Internet-Plattformen. Dabei gilt sein Interesse auch den technischen Details der JVM und deren Bedeutung für die nebenläufige Programmierung.



LUST AUF VERÄNDERUNG?

SOFTWAREENTWICKLER

SOFTWAREARCHITEKT

BERATER

PROJEKTLEITER

SCRUM MASTER

WIR SUCHEN DICH!

(VOM EINSTEIGER BIS ZUM PROFI)



Netpioneer GmbH • Ludwig-Erhard-Alle 20 • 76131 Karlsruhe
0721/92060-814 • jobs@netpioneer.de • www.netpioneer.de

Die iJUG-Mitglieder auf einen Blick

Java User Group Deutschland e.V.
<http://www.java.de>

DOAG Deutsche ORACLE-Anwender-
gruppe e. V.
<http://www.doag.org>

Java User Group Stuttgart e.V. (JUGS)
<http://www.jugs.de>

Java User Group Köln
<http://www.jugcologne.eu>

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

Java User Group München (JUGM)
<http://www.jugm.de>

Java User Group Metropolregion
Nürnberg
<http://www.source-knights.com>

Java User Group Ostfalen
<http://www.jug-ostfalen.de>

Java User Group Saxony
<http://www.jugsaxony.org>

Sun User Group Deutschland e.V.
<http://www.sugd.de>

Swiss Oracle User Group (SOUG)
<http://www.soug.ch>

Berlin Expert Days e.V.
<http://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
<https://www.xing.com/net/jughannover>

Java User Group Augsburg
<http://www.jug-augsburg.de>

Java User Group Bremen
<http://www.jugbremen.de>

Java User Group Münster
<http://www.jug-muenster.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.

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, NeptuneLabs;
Dr. Jens Trapp, Google; André Sept,
InterFace AG

Titel, Gestaltung und Satz:

Alexander Kermas,
DOAG Dienstleistungen GmbH
Foto Titel © mcrad/Fotolia.com

Anzeigen:

Simone Fischer
anzeigen@doag.org

Mediadaten und Preise:

<http://www.doag.org/go/mediadaten>

Druck:

Druckerei Rindt GmbH & Co. KG
www.rindt-druck.de

Java aktuell
Magazin der Java-Community



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



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 Rechnung. 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.





30.01.2014
DevCamp
Let's play together

Thema: Moderne Softwareentwicklung
im Oracle-Umfeld