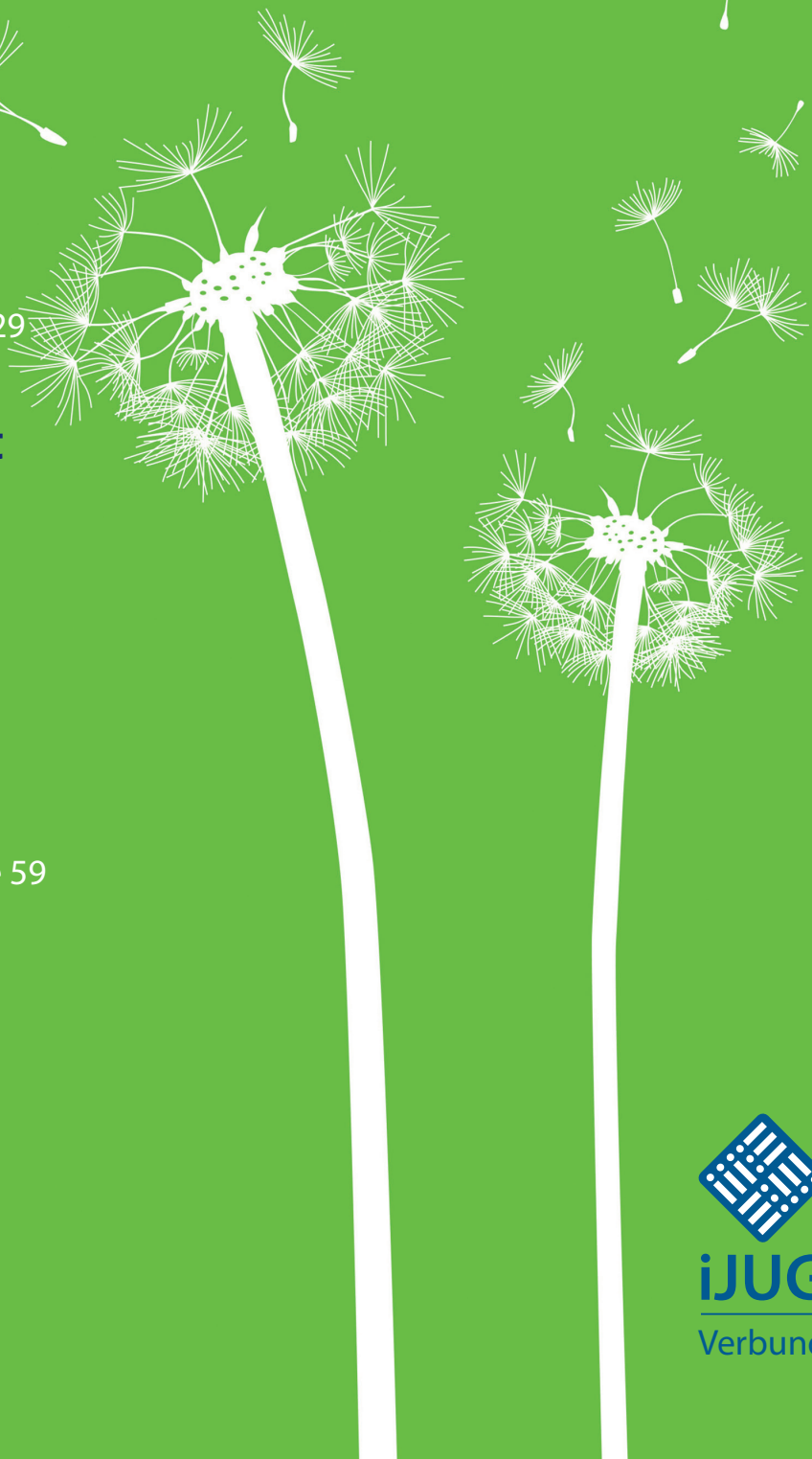
2-2013 | Sommer | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Java aktuell

Java verbreitet sich überall



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

iJUG Verbund

Wolfgang Taschner
Chefredakteur Java aktuell

Java deinstallieren?

Unter dem Namen „CVE-2013-0422" ging die Zero-Day-Sicherheitslücke im Januar 2013 in die Geschichte ein. Wieder einmal hat uns Oracle gezeigt, wie man es nicht machen sollte. Wieder einmal vergingen Tage, bis sich der Gralshüter von Java überhaupt dazu äußerte, und wieder einmal erlebten wir in diesen Tagen einen Wildwuchs aus Halbwahrheiten. Von „Java deaktivieren" war die Rede, an anderer Stelle sogar von „Java deinstallieren". Kein Online-Medium wusste richtig Bescheid, fast alle schrieben wenig Konkretes voneinander ab. So wussten die meisten PC-Anwender nicht, wie sie nachschauen konnten, ob Java überhaupt in ihrem Browser aktiviert war, wofür sie Java im Allgemeinen benötigten und wie sie mit dem Java-Applet auf ihrem Browser umgehen sollten.

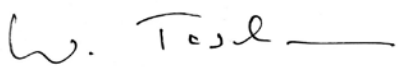
Oracle hat – wie schon bei der letzten Zero-Day-Sicherheitslücke im Herbst 2012 – die Wirkung in der Öffentlichkeit vollkommen verkannt. Im Gegensatz zu der überschaubaren Anzahl von Kunden, die es bei ähnlichen Problemen in der Oracle-Datenbank im Dunkeln lässt, hat es das amerikanische Unternehmen hier mit erheblich mehr Anwendern zu tun. Jeder PC-Benutzer, der im Internet surft, benutzt einen Browser und ist damit potenziell von der Sicherheitslücke betroffen. Bei all diesen Menschen keimt es jetzt im Hinterkopf: „Java = Sicherheitsproblem". Ähnlich wie Microsoft vor etwa zehn Jahren mit der Sicherheitsproblematik um Windows ist Oracle heute auf dem besten Weg, Java für lange Zeit in Misskredit zu bringen und damit den Erfolg einer guten und etablierten Technologie zu verspielen.

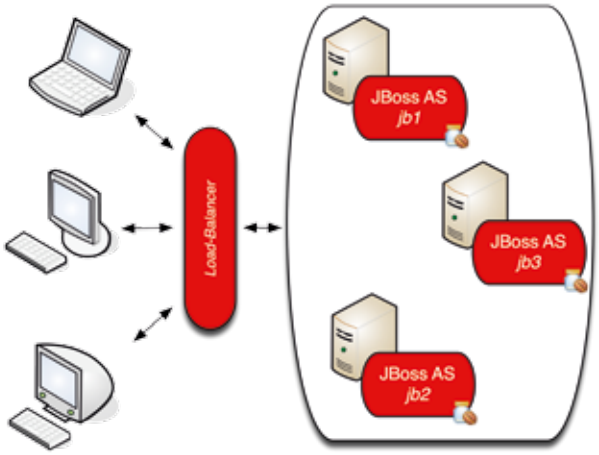
Immerhin haben die Entwickler bei Oracle übers Wochenende einen Fix entwickelt. Zudem hat dieses Release 11 automatisch in Release 10 einen Mechanismus aktiviert, der alle unsignierten Applets auf „unsicher" schaltet und zum Updaten aufruft. Wer jetzt auf diesem Update-Stand ist, kann die ganze Sache gelassener angehen. Allerdings stellt sich mir die Frage, warum Oracle nicht selbst in der Lage ist, solche Sicherheitslücken bereits vorab zu identifizieren und zu beheben.

Der iJUG hat auf die Situation reagiert und die Webseite http://www.ijug.eu/go/sicherheit eingerichtet, um alle Java-Entwickler und -Anwender bei künftigen Sicherheitslücken schnell und kompetent zu informieren.

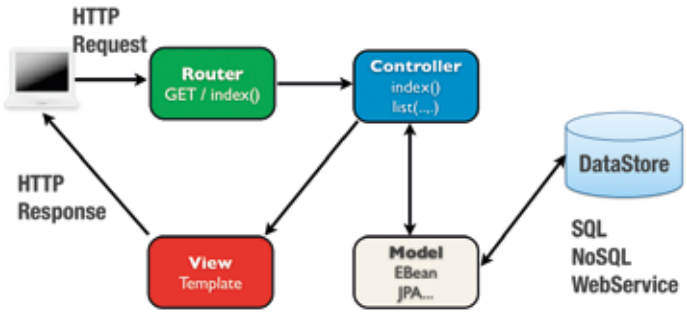
In diesem Sinne hoffe ich auf mehr Kommunikation, Transparenz und Proaktivität bei Oracle und wünsche Ihnen, dass Sie sich nach all dem Hin und Her wieder voll auf Ihre Java-Projekte konzentrieren können.

Ihr





Lastverteilung zum Erreichen der Hochverfügbarkeit, Seite 20



Architektur von Play 2 zur Entwicklung von Web-Apps, Seite 32

Build-Kreislauf beim Continuous Integration, Seite 59

Das Java-Tagebuch

Andreas Badelt, Leiter der DOAG SIG Java

Das Java-Tagebuch gibt einen Überblick über die wichtigsten Geschehnisse rund um Java – in komprimierter Form und chronologisch geordnet. Der vorliegende Teil widmet sich den Ereignissen im vierten Quartal 2012.

8. Oktober 2012

Apache TomEE 1.5 freigegeben

Apache hat seinen auf Tomcat und OpenEJB basierenden Application-Server TomEE in der Version 1.5 freigegeben. Neben der Java-EE-6-Web-Profile-zertifizierten Basis-Version und der um JAX-RS, JAX-WS, JCA und JMS erweiterten großen Version TomEE+ gibt es jetzt eine Zwischen-Version TomEE JAXRS, die genau das zusätzlich enthält, was der Name suggeriert. Ansonsten gibt es viele Bug-Fixes und kleinere bis mittelgroße Verbesserungen, insbesondere im Bereich „DB Connection Pools", sowie Updates der enthaltenen Komponenten.
Nachtrag: Die Unterstützung für TomEE durch andere Projekte und Hersteller wächst. Nur zwei Tage nach der Freigabe von TomEE 1.5 ist eine neue Version IntelliJ IDEA 12 „Early Access" mit TomEE-Server-Integration herausgekommen.
https://blogs.apache.org/openejb/entry/apache_tomee_1_5_released

12. Oktober 2012

JavaFX hat eine neue Community-Seite

Die neue Seite auf Java.net ist die perfekte erste Anlaufstelle für alle News, Artikel, Videos, Blogs und Tweets rund um Java. Die Verwalter der Seite, Gerrit Grunwald, Rajmahendra Hedge und Jim Weaver, wünschen sich Informationen über alle geplanten JavaFX-Sessions.
http://community.java.net/community/javafx

17. Oktober 2012

Java Community Process: Neuwahlen

Die Neuwahlen zum in Zukunft einheitlichen Executive Committee des JCP haben begonnen. Die Kandidaten für die vier ratifizierten Sitze sind Cinterion, Credit Suisse, Fujitsu und HP – sie müssen nur noch von den JCP Members in der Wahl „abgenickt" werden. Um die zwei zu besetzenden „freien" Sitze bewerben sich gleich neun Kandidaten: Die London Java Community (sie tritt zur Wiederwahl an) sowie Cisco Systems, CloudBees, Giuseppe Dell'Abate, Liferay, North Sixty-One, Software AG, Zero Turnaround und eine weitere Usergroup, die MoroccoJUG. Wie es inzwischen gute Tradition ist, stellen die Kandidaten sich schriftlich sowie in den nächsten Tagen per Online-Konferenz („Meet the EC Candidates") der Community vor. Bis zum 29. Oktober 2012 können dann alle eingetragenen Mitglieder des JCP abstimmen.
https://blogs.oracle.com/java/entry/jcp_elections_jug_candidates

22. Oktober 2012

Oracle gibt neue Version von ADF Mobile heraus

Oracle hat eine neue Version von ADF Mobile herausgegeben, mit der ADF-Anwendungen mittels spezieller Container nativ auf iOS- und Android-basierten Geräten ausgeführt werden und vollen Nutzen aus den jeweiligen Clients ziehen können. Aufgrund der Lizenzkosten ist das eher etwas für Firmen, die sowieso schon ADF im Einsatz haben, dann ist es jedoch eine sehr interessante Option. Eine Demo für mobile Endgeräte ist unter http://jdevadf.oracle.com/amx und unter https://blogs.oracle.com/mobile/entry/adf_mobile_released verfügbar.

24. Oktober 2012

JCache Early Draft Review gestartet

Nur wenig mehr als eine Dekade ist vergangen – und schon wird das Early Draft Review für den JSR 107 „JCache" gestartet. Einen Preis für Geschwindigkeit wird dieser JSR nicht mehr erhalten, aber immerhin wird er jetzt wohl von einer 2011 neu zusammengestellten Expert Group zu Ende geführt – statt ewig unvollendet zu bleiben.
Einen hoffentlich baldigen Abschluss wird auch der JSR 347 „Data Grids for the Java Platform" erhalten, der dann sauber darauf aufsetzen kann.
Eine kurze Zusammenfassung darüber steht unter http://blog.eisele.net/2012/10/java-temporary-caching-api-test-driving.html

30. Oktober 2012

Wahlen zum Executive Committee abgeschlossen

Keine Überraschungen bei den Wahlen zum neuen gemeinsamen Executive Committee des Java Community Process. Alle vier von Oracle nominierten Kandidaten für die „ratified seats" wurden bestätigt: Credit Suisse, Fujitsu Limited, Hewlett-Packard und die Cinterion Wireless Modules GmbH.
Bei den Wahlen zu den freien Sitzen wurden die London Java Community wiedergewählt sowie CloudBees, die vor einem Jahr noch gegen Azur Systems den Kürzeren gezogen hatten. Die beiden setzten sich unter anderem gegen Zero Turnaround und die MoroccoJUG durch.
https://blogs.oracle.com/jcp/entry/2012_ec_election_results

10. November 2012

Java EE 7 Community-Umfrage

Die Expert Group für Java EE 7 hat eine Umfrage ins Netz gestellt, um die Community in die Feature-Planung einzubeziehen, an der sich alle Interessierten anonym beteiligen können; eine sehr gute Idee, um

Hemmschwellen abzubauen und direktes Feedback zu erhalten.
https://blogs.oracle.com/theaquarium/entry/wanted_now_your_feedback_on

13. November 2012

Nur noch ein JCP Executive Committee

Der JSR 355 „JCP Executive Committee Merge" ist abgeschlossen. Die beiden Executive Committees zu Java SE/EE und Java ME sind nun zu einem einzigen zusammengewachsen – dasselbe passiert hoffentlich auch mit den Technologien.
https://blogs.oracle.com/jcp/entry/welcome_to_the_newly_merged

19. November 2012

Java ME wird weiter mit SE zusammengeführt

Terrence Barr, Produktmanager bei Oracle für Java Embedded, findet den Executive-Committee-Zusammenschluss offensichtlich auch gut. In seinem Blog weist er auf zwei neue JSRs hin, die Java ME nicht nur modernisieren, sondern auch ME und SE einander näher bringen sollen: JSR 360 „Connected Limited Device Configuration 8" und der darauf aufbauende JSR 361 „Java ME Embedded Profile". Auf der JCP-Seite von JSR 360 heißt es: „The Java ME CLDC specification is to be updated with VM, Java Language, and libraries and features to be aligned with Java SE 8 while retaining the focus on small mobile and embedded target devices." Durch die Angleichung sollen als Features unter anderem Generis, Annotations und „try with resources" zur Verfügung stehen.
http://terrencebarr.wordpress.com/2012/11/18/jsr-360-and-jsr-361-a-big-leap-for-java-me-8/

24. November 2012

Patrick Curran wirbt in Deutschland für den JCP

Patrick Curran, Chairman des Java Community Process (JCP), hat in Deutschland für den JCP geworben. Bei seinem Auftritt auf der DOAG 2012 Konferenz in Nürnberg berichtete er über die laufende Umstrukturierung des JCP und warb gleichzeitig für die aktive Mitarbeit der Community. Einen Tag später war er zur Mitgliederversammlung des Interessenverbunds der Java User Groups e.V. (iJUG) eingeladen, bei der ebenfalls die aktive Zusammenarbeit insbesondere der Expert Groups mit den JUGs diskutiert wurde.
http://www.ijug.eu/home-ijug/aktuelle-news/article/der-java-community-process-soll-schlanker-agiler-und-transparenter-werden.html

4. Dezember 2012

Öffentliche Updates für Java SE 6 gibt es bis Februar 2013

Nach Protesten aus der Community, an denen sich auch der iJUG beteiligt hatte, war der freie Support für Java SE 6 in die Verlängerung gegangen. Am 19. Februar 2013 soll nun endgültig Schluss sein. Alle Nutzer, die weder kommerziellen Support von Oracle noch Spaß am Risiko haben, sollten spätestens dann Java SE 7 nutzen.
http://blogs.oracle.com/java/entry/end_of_public_updates_for

7. Dezember 2012

Ein Nashorn im OpenJDK

Wie von Oracle angekündigt, ist die JavaScript Engine Nashorn („a lightweight high performance JavaScript on a native JVM") dem OpenJDK als neues Teilprojekt angeboten und von den Mitgliedern nun einstimmig angenommen worden. Die von Oracle zur Verfügung gestellte Code-Basis soll bereits den ECMAScript test262 zu 100 Prozent bestehen, aber noch hinsichtlich Performance und Stabilität Verbesserungsbedarf haben.
http://mail.openjdk.java.net/pipermail/announce/2012-December/000140.html

14. Dezember 2012

Community-Umfrage zu Java EE 7

Die offene Umfrage zu Java EE 7 ist beendet und die Resultate sind unter https://blogs.oracle.com/reza/entry/java_ee_7_survey_results veröffentlicht. Über 1.100 Java-Fans haben sich beteiligt und sind sich in den meisten Fragen recht einig. Die Reaktionen und Diskussionen der Expert Group sind auch im Netz einzusehen (siehe Link) – ganz im Sinne des neuen JCP-Mottos „Transparenz". Die Expert Group scheint mit dem Feedback recht glücklich zu sein und begibt sich an die Umsetzung der Community-Wünsche, wenn auch wohl nicht alles sofort mit Java EE 7 umgesetzt werden kann, wie die übergreifende Nutzung der „@Stereotype"-Annotation.

Java.net-Umfrage zu Lambda Expressions

Und noch eine Umfrage, diesmal auf java.net zu Lambda Expressions („aka Closures"). Die kommen zwar erst mit Java SE 8, aber das Ziel der Umfrage war, herauszufinden, wie viele Entwickler sich dafür interessieren oder sie sogar schon ausprobieren. Die Beteiligung war mit 332 Stimmen allerdings eher mager. Von diesen entfielen immerhin 35 Prozent auf „Ich habe das Pre-Release heruntergeladen und mit Lambda Expressions experimentiert", weitere 22 Prozent verfolgen zumindest die Entwicklung, während 20 Prozent sich damit zufriedengeben, nach Fertigstellung von Java SE 8 einen genaueren Blick darauf zu werfen. Die erstaunlichen 35 Prozent Experimentierfreudigen könnten aber vielleicht daher kommen, dass sich hauptsächlich Fans der Lambda Expressions überhaupt für die Umfrage interessiert haben.
http://weblogs.java.net/blog/editor/archive/2012/12/12/poll-result-java-developers-are-following-java-8-lambda-expressions-progress

Java SE 7u10: Mehr Kontrolle über Sicherheits-Einstellungen

Oracle hat Update 10 von Java SE 7 herausgebracht, ein Sicherheits-Update. Mit dieser Version kann das Browser-Plug-in getrennt deaktiviert werden. Außerdem lassen sich je nach Bedarf verschiedene Sicherheitsstufen für das Ausführen von unsignierten Applets, Web-Start- und JavaFX-Applikationen einstellen. Darüber hinaus wird automatisch geprüft, ob die Java-Version noch aktuell ist und damit als hinreichend sicher angenommen wird. Falls diese Prüfung nicht möglich ist (etwa wegen fehlenden Netzwerk-Zugriffs), dient dazu ein in den Build hart verdrahtetes „Haltbarkeitsdatum". Ist die Version nicht mehr aktuell, gibt es in entsprechenden Situationen wie dem Laden unsignierter Applets spezielle Warnungen. Alles in allem also

einige Neuerungen, um Java im Browser – das immer mehr Aufmerksamkeit von kriminellen Hackern erhält – wieder sicherer zu machen.
https://blogs.oracle.com/henrik/entry/oracle_jdk_7u10_released_with

17. Dezember 2012

Zuwachs für die EE-6-Familie
Laut einer Mitteilung der SAP AG ist für NetWeaver Cloud die Kompatibilität mit dem Java EE 6 Web Profile nachgewiesen worden – ein weiterer Application-Server in der inzwischen schon recht großen Familie, der zwar damit wie einige andere nur eine Untermenge der vollständigen Java-EE-6-Spezifikation unterstützt, was aber für eine großen Teil der Enterprise-Anwendungen ausreichen sollte.
http://www.sap.com/corporate-en/news.epx?PressID=20122

18. Dezember 2012

Adopt a Java EE 7 JSR
Kein Tagebuch in der Java aktuell ohne Erwähnung des „adopt aJSR"-Programms. Damit niemand die Gelegenheit zur Mitarbeit verpasst (etwa weil die richtige Idee fehlt, um loszulegen), gibt es auf den GlassFish-Community-Seiten (siehe http://glassfish.java.net/adoptajsr) jetzt eine detaillierte Auflistung der Java-EE-7-JSRs und Informationen darüber, wo genau welcher JSR noch Unterstützung benötigt. Für diejenigen, die sich im Detail für Ursprung, Zweck und Nutzen des Programms interessieren, hat Markus Eisele unter http://blog.eisele.net/2012/12/article-about-adopt-jsr-program.html einen umfangreichen Artikel geschrieben.

31. Dezember 2012

Java SE 8 für ARM – und Tipps für Raspberry-Pi-Bastler
Gute Neuigkeiten für Bastler und Embedded-Fans: Seit Kurzem ist ein Preview des OpenJDK 8 inklusive JavaFX für Linux-/ARM-Rechner verfügbar, also zum Beispiel für den 35-Dollar-Computer Raspberry Pi (siehe http://jdk8.java.net/fxarmpreview). Mark Heckler von Oracle gibt in seinem Blog unter http://blogs.oracle.com/java-jungle/entry/creating_a_portable_java_javafx einige Tipps zur Einrichtung des Pi.

9. Januar 2013

How Do I Get My Java Video on the YouTube/Java Channel?
Der YouTube Java Channel wächst und wächst. Wer dort schon immer sein eigenes Java-Video veröffentlichen wollte, findet die entsprechenden Schritte im Oracle Java Blog unter https://blogs.oracle.com/java/entry/how_do_i_get_my. Die Vorgabe lautet allerdings: „keine Werbung". Das würde bedeuten, dass mein absoluter Lieblingsfilm zu Java keine Chance hätte – der „Java 4-Ever Trailer" der norwegischen JavaZone. Aber er ist ohnehin schon auf YouTube zu finden.

10. Januar 2013

OW2 JOnAS ist für das Java EE 6 Web Profile zertifiziert
Mit OW2 JOnAS kommt ein weiterer Application-Server dazu, der das schlanke Web Profile von Java EE 6 unterstützt. Insgesamt sind es nun schon acht – und damit fast so viele wie bei der Obermenge „Full Profile".
https://blogs.oracle.com/theaquarium/entry/ow2_jonas_java_ee_6

13. Januar 2013

Kritische Sicherheitslücken mit SE 7 Update 11 behoben
Die Ruhe nach der Freigabe von Java SE 7 Update 10 währte nicht lange. Am 9. und 10. Januar wurden im Internet Meldungen über eine neu entdeckte kritische Sicherheitslücke verbreitet, sowie dazu bereits existierende „zero-day exploits", ein Schadcode, der in der Regel von professionellen Hackern für kriminelle Zwecke verkauft oder versteigert wird – je schneller, desto höher der Preis, wobei am „Tag null" natürlich das Maximum herausspringt. Erste Angriffe auf Browser mit aktivem Java-Plug-in über präparierte Internet-Seiten wurden gemeldet. Nach einigen Tagen Unruhe im Netz hat Oracle nun das außerplanmäßige Update 11 herausgegeben. Neben dem Stopfen der entdeckten Lücken werden auch die mit Update 10 eingeführten Sicherheitsstufen genutzt – das Standard-Level wird von „medium" auf „hoch" gesetzt. Damit wird unsignierter Java-Code nur noch nach Bestätigung durch den Nutzer vom Browser-Plug-in geladen, selbst wenn die Java-Version aktuell ist. Der iJUG hat über das Thema berichtet (siehe http://www.ijug.eu/home-ijug/aktuelle-news/article/sicherheitsluecke-behoben-oracle-veroeffentlicht-ausserplanmaessiges-update.html). Eine sehr gute Erklärung, wie genau die Sicherheitslücken ausgenutzt werden, findet sich unter http://timboudreau.com/blog/The_Java_Security_Exploit_in_(Mostly)_Plain_English/read.

18. Januar 2013

Update 11 hat nicht alle Sicherheitslücken geschlossen
Nur wenige Tage nach der Herausgabe des außerplanmäßigen Sicherheits-Updates 11 für Java SE 7 werden bereits neue Lücken gemeldet – außerdem ist eine bekannte Lücke offensichtlich nicht geschlossen worden. Anscheinend gibt es bei Oracle keine Pläne, kurzfristig ein weiteres Update nachzuschieben, vor dem regulären „Critical Patch Update"-Termin Mitte Februar 2013. Ob man sich wirklich darauf verlassen kann, dass Nutzer die Warnungen vor unsignierten Applets beherzigen, statt sie in bekannter Windows-Manier wegzuklicken? Ich möchte gar nicht wissen, wie viele Pop-ups mit haarsträubenden Meldungen ich im Laufe der Jahre bereits genervt – und ungelesen – bestätigt habe. Es bleibt spannend …

24. Januar 2013

Wie sicher ist Java?
Die jüngsten Diskussionen über Sicherheitslücken haben Spuren hinterlassen. Passend dazu gab es auf DZone die Umfrage „Wie sicher ist Java?" Von großer Panik in der Community und Flucht aus Java ist nichts zu spüren, das Thema wird offensichtlich von den meisten Teilnehmern sachlich betrachtet – aber natürlich kommt Oracle bei der Umfrage nicht besonders gut weg. Der Konzern hat al-

lerdings in Bezug auf Kommunikation gelernt: Oracles Java-Evangelist und Blogger Reza Rahman (obwohl eher im Java-EE-Bereich angesiedelt) meldet sich in der Diskussion zu Wort. Und wenige Tage später, am 24. Januar 2013, findet eine Telefonkonferenz von Verantwortlichen für Java Security und das OpenJDK mit den Java User Groups statt („JUG Leaders Call"). Der Link zum Mitschnitt wird auf der Dzone-Seite von Reza Rahman gepostet. Die Oracle-Verantwortlichen können bei einigen Fragen nur ausweichende Antworten geben, teilweise, weil sie nach eigener Aussage durch Verträge mit Partnern gebunden sind (Stichwort „Installier"), teilweise vermutlich, weil sie sich nicht zu sensiblen Details der Sicherheitslücken äußern dürfen. Generell wird aber offen über das Thema „Java und Sicherheit" diskutiert, wobei Kommunikations-Strategien einen großen Raum einnehmen.
http://java.dzone.com/articles/weekly-poll-how-safe-java

Andreas Badelt ist 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.

UPDATE An dieser Stelle erhalten Sie in jeder Ausgabe ein Update über das Geschehen in der Java-Community

Oracle JDeveloper 12*c* – ein Ausblick

Frank Nimphius, ORACLE Deutschland B.V. & Co. KG

Oracle JDeveloper und das integrierte Application Development Framework (ADF) sind derzeit in den Versionen 10g (10.1.3.5), 11g R1 (11.1.1.6) und 11g R2 (11.1.2.3) produktiv verfügbar. Mit Oracle Fusion Middleware 12c (FMW), das in diesem Jahr erwartet wird, kommt mit JDeveloper 12c ein neues Release der Entwicklungsumgebung (IDE) für Oracle Java, Java EE und Service Oriented Architecture (SOA) auf den Markt. Dieser Artikel zeigt vorab einige ausgewählte Neuerungen innerhalb des Oracle JDeveloper und ADF.

JDeveloper ist die strategische Entwicklungsumgebung für Java-, Java-EE-, SOA- und ADF-Anwendungen innerhalb der Oracle Fusion Middleware. In Bezug auf das Application Development Framework kommt dem JDeveloper eine besondere Bedeutung zu, da mit jedem Release der IDE auch eine neue Version des Oracle ADF veröffentlicht wird.

Ein Blick zurück und nach vorn

Wie erwähnt, sind derzeit drei Versionen des JDeveloper produktiv im Einsatz. JDeveloper 10*g* (10.1.3.5) unterstützt den Oracle Application Server und OC4J als Laufzeitumgebung. Bei dieser Version des JDeveloper kann man durchaus von „legacy" sprechen, da sich seit dem Erscheinen dieser Versionen einige Technologien sehr verändert haben. JDeveloper 10*g* wird heute in der Regel zur Pflege existierender JSF- und Struts-Anwendungen eingesetzt oder um das Upgrade nach JDeveloper 11*g* vorzubereiten.

JDeveloper 11*g* R1 (11.1.1.x) ist die Version der IDE und ADF, die von Oracle Fusion Applications verwendet wird. Zu jeder Version des JDeveloper existiert ein Fusion-Middleware-Release (FMW) mit der gleichen Versionsnummer, sodass man den vollen FMW-Produkt-Umfang (WebCenter, BI, UCM, SOA etc.) innerhalb seiner Anwendung nutzen kann. JDeveloper 11*g* R1 und FMW 11*g* sind extrem robuste Produkte, die uns noch eine ganze Weile und über Patch Sets hinweg begleiten werden.

JDeveloper 11*g* R2 (11.1.2.x) ist ein reines JDeveloper- und ADF-Release ohne begleitendes FMW-Release. Damit geschriebene Anwendungen sind – nach Aufspielen eines speziellen Patch – auf WLS 10.3.6 oder 10.3.5 ausführbar, können aber nicht mit anderen FMW-Produkten zusammen installiert werden. Oracle JDeveloper 11*g* R2 wird in Projekten eingesetzt, die neueste Technologien benötigen, zum Beispiel ADF Mobile oder JSF 2, oder die direkt nach Verfügbarkeit auf JDeveloper 12*c* und FMW 12*c* migriert werden sollen. JDeveloper 12*c* ist in diesem Release-Zyklus das nächste Major-Release der Oracle-Java-, Java-EE- und SOA-DIE mit ADF und Oracle FMW.

JDeveloper 12*c* im Überblick

JDeveloper 12*c* und Oracle ADF unterstützen Java EE 6, JSE 7 sowie HTML 5 und Cascading Style Sheets 3 (CSS). Das zur Laufzeit generierte HTML und CSS vieler ADF-Faces und JavaServer-Faces-Komponenten werden dabei ebenfalls auf HTML 5 und CSS3 mit drei Zielen umgestellt:

- Verbesserung der Performance durch reduzierte Download-Größe
- Optimierung für mobile Endgeräte wie Tablet PCs

- Optimierung in Bezug auf die Erstellung reaktiver Benutzeroberflächen (responsive design)

Ein weiteres großes Thema bezüglich Java EE 6 ist die Unterstützung neuerer Standards wie EJB 3.1, JSF 2.1, Context and Dependency Injection (CDI), RESTful Services (JAX-RS), Bean Validation und Servlet 3.0 und die WebLogic-Server-12*c*-Unterstützung durch die Oracle IDE und ADF.

Neben neuen User-Interface-Komponenten in ADF Faces sind im ADF-Umfeld als grundlegende Neuerungen der Support von RESTful-Web-Services seitens des URL Data Control auf der Client-Seite und ADF Business Components (ADF BC) als Service-Provider zu nennen.

Dank einer Besonderheit des Oracle-JDeveloper-Release-Managements, nach der Patch-Sets nicht nur Bug-Fixes beinhalten, sondern auch neue Funktionalitäten, sind einige der für 12*c* geplanten Neuerungen wie zum Beispiel die Unterstützung von Touch Screens bereits in JDeveloper 11*g* enthalten. Der Artikel wird daher nachfolgend immer wieder auf Backports dieser Art eingehen.

Wichtige Neuerungen in der IDE

Insgesamt wird die IDE in JDeveloper 12*c* schneller sein und einen komplett neuen und frischen Eindruck auf den Anwender machen. Dafür sorgen ein neues Look-and-Feel der Entwicklungsumgebung, optimierte Menü-Strukturen sowie eine OSGi-basierte Neuentwicklung der JDeveloper-Extension-Plattform, die Plug-ins erst dann lädt, wenn der Entwickler sie benötigt (siehe Abbildung 1).

Langjährige JDeveloper-Entwickler werden außerdem feststellen, dass der neue Code-Editor sowie die neue visuelle Web-Entwicklungsumgebung an NetBeans angelehnt sind. Mit JDeveloper 12*c* wird die Zusammenarbeit zwischen den beiden IDEs verstärkt. Um Gerüchten und Fehl-Interpretationen vorzubeugen: Beide IDEs bleiben in ihrer jetzigen Form erhalten.

REST-Support

Representational-State-Transfer-Services (REST) sind HTTP-basierte Web-Services, die im JDeveloper 12*c* als JAX-RS-Services mithilfe eines Wizard deklarativ entwickelt werden (siehe Abbildung 2). Dieser Wizard erleichtert das Mapping der Methoden eines Java-Objekts auf die HTTP-Requests „POST", „PUT", „GET" und „DELETE" eines Remote Service Clients.

JDeveloper 12*c* erleichtert auch die Entwicklung von Java-basierten REST-Clients. Java-Entwickler erzeugen JAX-RS-basierte REST-Clients deklarativ aus der Referenz zu einer Web-Application-Description-Language-Source (WADL) heraus. WADL ist in diesem Zusammenhang mit „WSDL für SOAP"-Services zu vergleichen. Änderungen sind an dieser Stelle allerdings vorbehalten, da WADL im Gegensatz zu WSDL kein Standard ist.

Das JDeveloper-Extension-Framework

Eine Eigenschaft aller Java- und Java-EE-Entwicklungsumgebungen auf dem

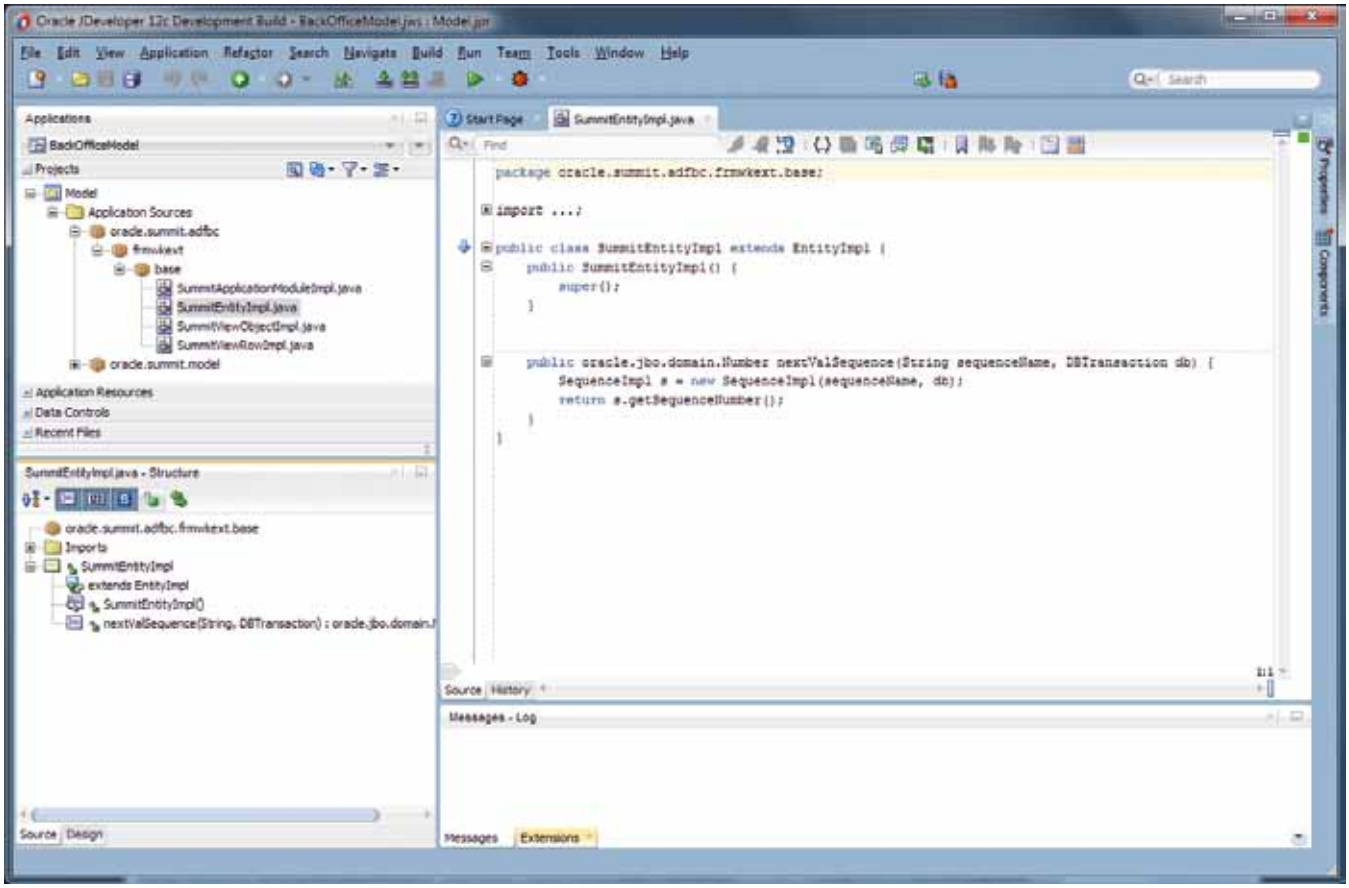


Abbildung 1: Das Look-and-Feel der JDeveloper-12c-IDE

Markt ist die Möglichkeit, dass Entwickler die IDE um eigene Funktionalitäten, die sogenannten „Plug-ins", erweitern.

JDeveloper 12*c* bietet dafür ein neues OSGi-basiertes Extension-Development-Kit sowie einen Wizard für den Start der Plug-in-Entwicklung. Eigene Plug-ins können dabei ohne vorhergehendes Deployment im JDeveloper getestet werden, was den Zeitaufwand für das Testen minimiert.

Maven

In JDeveloper 12*c* ist Maven nun endlich vollständig angekommen und als Build- und Dependency-Management-System fest integriert. Entwickler können zukünftig beim Erstellen eines neuen JDeveloper-Workspace entscheiden, ob das Projekt mit Ant, Maven oder dem JDeveloper-eigenen Build-System kompiliert und „deployt" werden soll. JDeveloper 12*c* unterstützt die folgenden Maven-Funktionalitäten:

- Maven 3
- Erstellung des Maven Project Object Model (POM) für JDeveloper-Projekte und -Workspaces
- Import von POM
- POM-Konfiguration und Syntax-Hilfe
- Laufzeit-Ziele (Run Goals)
- Verwaltung von Maven-Repositories
- Repository-Authentifizierung
- Flat-Editor zur deklarativen Konfiguration

Es wird kein externes Maven-Repository geben, in dem Oracle die Library-Abhängigkeiten für eine bestimmte Fusion-Middleware-Version hinterlegt. Stattdessen ist ein Script geplant, mit dem Kunden ein eigenes lokales Maven-Repository für ein bestimmtes Fusion-Middleware Release erstellen können.

„OJDeploy" ist ein Deployment-Tool innerhalb des JDeveloper und in Bezug auf den Umgang mit den in ADF vorhandenen Metadaten sowie auf die Fertigstellung abhängiger Dateien optimiert. Zur Unterstützung des Software-Build- und Deployment-Prozesses durch Continuous Integration (CI) ist bei JDeveloper 12*c* OJDeploy als Plug-in für Maven verfügbar. Maven kann im Zusammenspiel mit dem Hudson-CI-Tool dann zur automatischen ADF-Build-Generierung verwendet werden.

Neuerungen in Oracle ADF

Die wesentlicheren, nicht visuellen Änderungen innerhalb von ADF gibt es beim URL Data Control sowie dem JavaBean Data Control (POJO). Das URL Data Control ist nun voll REST-fähig und unterstützt URI-Aufrufe über die HTTP-Methoden „POST", „GET", „PUT" und „DELETE". Der Entwickler registriert dabei bestimmte URIs im Data Control und wählt die gewünschte HTTP-Methode für den Aufruf. Das URL Data Control ermöglicht es, User-Interfaces basierend auf RESTful-Services deklarativ für Web- und mobile Endgeräte zu erstellen.

In JDeveloper 12*c* ist das JavaBean Data Control deutlich verschlankt und kommt zukünftig ohne XML aus. XML wird erst generiert, wenn Entwickler das Data Control anpassen, zum Beispiel um UI-Hints (Tool-Tipps und Prompts) und Validierungen zu konfigurieren. Eine weitere Neuerung im JavaBean Data Control ist die Möglichkeit, ähnlich wie in ADF Business Components, die List-of-Value Listen für Accessor-Attribute zentral zu definieren. Somit erleichtert sich die Programmierung von List-of-Values auch für Enterprise Java-

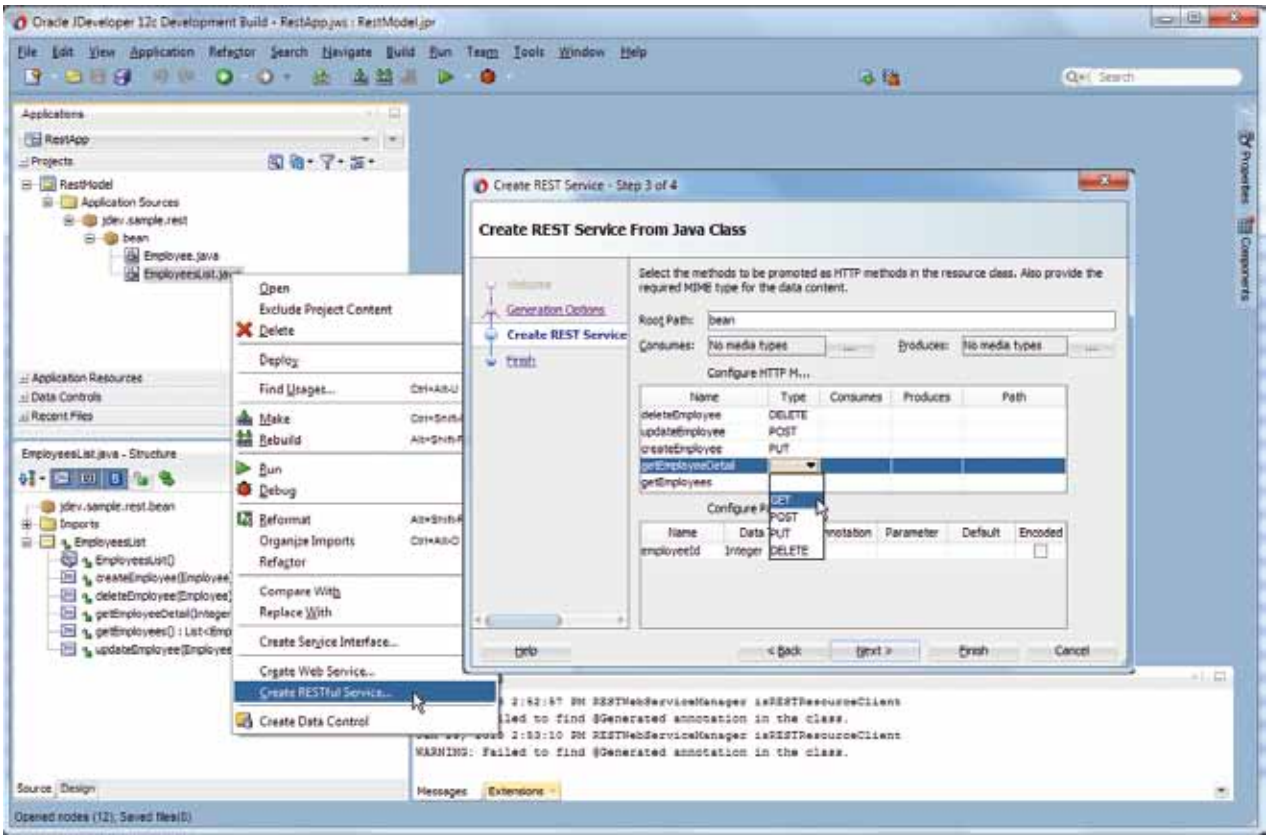


Abbildung 2: Erstellen POJO-basierter REST-Services

Beans Business Models. „Named Criteria", ein Konzept aus dem ADF-Business-Components-Umfeld, erlauben es Entwicklern, Query-Filter vergleichbar zu „Where Clauses" in SQL zu definieren und über das Data Control Panel deklarativ zur Erstellung von Suchmasken zu verwenden.

Hinweis: Einige in diesem Abschnitt erwähnten Funktionalitäten des URL- und JavaBean Data Control sind bereits als Backport in JDeveloper 11*g* R2 verfügbar.

Neuerungen in ADF Business Components

„ADF Business Components" (BC), ein Oracle-eigenes Business-Service- und Persistenz-Framework, ist intern bei Oracle Fusion Applications und extern bei Oracle-Kunden großflächig im Einsatz. Eine wesentliche Veränderung in ADF BC ist der Wechsel der Standard-Daten-Typen weg vom „oracle.jbo.domain"-Package hin zu „Java Extended for Oracle". Mit diesem Wechsel, der zum Beispiel „oracle.jbo.domain.Number" durch „Integer" oder „BigDecimal" ersetzt, ist es wesentlich leichter, ADF BC als Web-Service zu deployen oder Web-Services in ADF BC zu integrieren. Die bisherigen Oracle-Domain-Typen sind dabei weiterhin unterstützt.

Die ADF-Business-Component-Editor-Dialoge sind durch sogenannte „Flat-Editors" ersetzt worden, die einen schnelleren Zugriff auf Konfigurationen von Entitäten, Views, Objekten und Applikations-Modulen ermöglichen.

Innerhalb von ADF BC können Validierungen mittels Groovy programmiert werden. Dazu bietet JDeveloper 12*c* einen Groovy-Debugger, der die Suche nach möglichen Programmierfehlern vereinfacht.

ADF BC als RESTful-Service-Provider

Wiederverwendung ist eine große Stärke des ADF-BC-Frameworks innerhalb von ADF. Dabei können sowohl einzelne Objekte über spezielle ADF-Libraries zwischen Entwicklungsprojekten ausgetauscht als auch komplette Modelle (Geschäftslogik und Persistenz-Definition) zur Verwendung in Web-Anwendungen oder als Web-Service deployt werden. Um ein ADF-Business-Components-Model als Service zu deployen, genügt es, wenn der Anwendungsentwickler die View-Objekte, View-Criteria und Methoden auswählt, die über einen JAX-WS-Service oder einen direkten Zugriff aus BPEL heraus aufrufbar sein sollen.

Mit JDeveloper 12*c* wird es möglich sein, ADF BC als RESTful-Services neben SOAP-Services zu deployen, ohne zusätzlich programmieren zu müssen. Einmal geschriebene Geschäftslogik kann somit im Web und in einer SOA-Umgebung als SOAP- und REST-Service verwendet werden (siehe Abbildung 3).

Remote Task Flows

Task Flows in ADF erlauben es, die JSF-Navigation einer Anwendung modular zu definieren, aufzurufen, als Library zu deployen und dadurch wiederzuverwenden. In JDeveloper 12*c* können Task Flows nun auch stand-alone auf einen Remote-Server deployt und über eine Laufzeitreferenz in ADF-Anwendungen dynamisch eingebunden werden. Diese Funktionalität kommt dem Einsatz von WSRP-Portlets nahe. An dieser Stelle sei aber vermerkt, dass es sich bei Remote Task Flows nicht um Portlets handeln. Der Vorteil von Remote Task Flows liegt in der Anwendungs-Adminis-

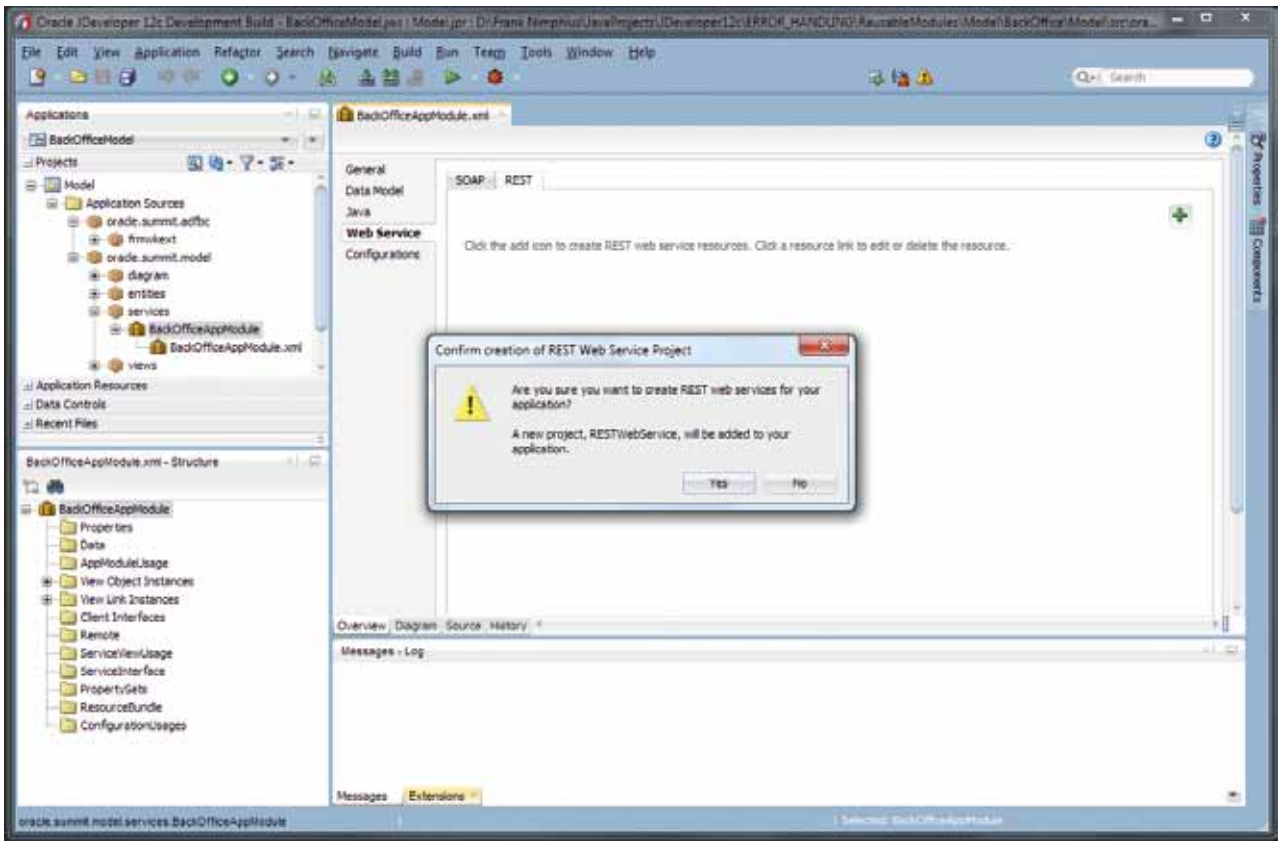


Abbildung 3: REST-Service-Entwicklung für ein bestehendes ADF-BC-Modul

tration und -Verwaltung, da nunmehr einzelne Anwendungs-Module separat gewartet und deployt werden können.

Neuerungen in ADF Faces und JSF-2-Unterstützung

Auch für ADF Faces gilt, dass einige der neuen 12*c*-Funktionen wie Tablet- und Content-Delivery-Network-Unterstützung (CDN) bereits in Patch-Sets bestehender Versionen (in diesem Fall Oracle JDeveloper 11*g* R1 11.1.1.6) als Backport verfügbar gemacht wurden.

Ein Augenmerk in der Weiterentwicklung von ADF Faces gilt dessen Verwendung zur Erstellung von öffentlichen Webseiten. Das zeigt sich vor allem in Funktionen wie „Pretty URL", einer verbesserten Suchmaschinen-Unterstützung, Reduktion der verwendeten CSS- und JavaScript-Ressourcen sowie dem Einsatz von HTML 5 in den Graphic- und User-Interface-Komponenten.

ADF Faces in JDeveloper 12*c* basiert auf JavaServer Faces 2 und bietet somit – ähnlich wie JDeveloper 11*g* R2 – Facelets als Alternative zur Verwendung von JSPX-Dokumenten als View Declaration Language an. Auch wenn eine Migration von JSPX nach Facelets nicht zwingend und somit optional ist, bietet JDeveloper 12*c* einen Migrations-Assistenten für diesen Wechsel an.

Neue ADF-Faces-Komponenten

Nachfolgend ein Überblick über die geplanten Änderungen, ohne groß ins Detail zu gehen. Die Neuerungen in ADF Faces sind:

- Eine neue „PanelGridLayout"-Komponente zur Erstellung von Spalten- und Zeilen-orientierten Layouts ähnlich der HTML-Tabelle. PanelGridLayout ist in das ADF-Faces-Geometry-Management integriert, sodass sich Änderungen der Anzeigegröße des Browsers direkt auf das Grid auswirken. Im Vergleich zu bestehenden Layout-Containern in ADF Faces ist PanelGridLayout schlanker in Bezug auf das zur Laufzeit generierte HTML.
- HTML-5-basierter Datei-Upload, der es erlaubt, mehrere Dateien gleichzeitig zu laden. Die Animation der jeweiligen Upload-Vorgänge kann man als „schick" bezeichnen.
- HTML 5 „place-holder-label" erlauben es, Prompts in ein Textfeld zu schreiben, die dann entfernt werden, sobald der Anwender in das Feld klickt und beginnt, es zu editieren.
- Die verschiedenen Typen von „Button" und „Link" werden zu einer Komponente „af:button" und „af:link" zusammengeführt. Die bestehenden Komponenten bleiben dabei weiterhin gültig.
- Eine Code-Editor-Komponente erlaubt es, Script-Code innerhalb einer ADF-Faces-Seite anzuzeigen und zu editieren.
- Page Templates können künftig als Kopie eines bestehenden Template erstellt werden. Im Falle des Dynamic Tab Shell Template (UI Shell) würden alle Page-Sourcen in das neue Template kopiert und die verwendeten ADF-Bindings ebenfalls dem Template angehängt.
- Der ADF-Faces-Skin-Editor wurde erheblich verbessert und unterstützt im Ansatz einen Live-View-WYSIWYG-Mode sowie die Möglichkeit, Komponenten-Images nicht nur zu generieren, sondern auch in der Helligkeit, dem Kontrast und der Sättigung zu verändern.
- Im Bereich der Daten-Visualisierung sind einige neue Graph-Typen inklusive eines „m:n"-Social-Network-Diagrammer hinzugekommen. Alle Graph-Typen, die zuvor noch Flash verwendet haben, benutzen nun ausschließlich HTML 5. Für Browser, die HTML 5 nicht unterstützen, erfolgt ein „graceful fallback" auf die Verwendung von Flash.

ADF Mobile

ADF Mobile ist eine auf Oracle ADF basierende, hybride Entwicklungsumgebung für die deklarative Entwicklung von Anwendungen für Apple-iOS- und Android-basierte mobile Endgeräte mit der Option, weitere Plattformen zukünftig zu unterstützen. ADF-Mobile-Anwendungen werden mittels Java, JavaScript, HTML und spezieller XML-UI-Komponenten entwickelt und per Deployment nativ auf dem mobilen Endgerät installiert. Über Java und JavaScript, aber auch deklarativ über ein spezielles „ADF Device Datacontrol", haben Anwendungsentwickler Zugriff auf Gerätefunktionen wie Kamera, Mail, Adressbuch und SMS. Der externe Datenzugriff in ADF Mobile erfolgt über SOAP- und REST-Services sowie eine lokale SQLite-Datenbank für den Offline-Betrieb.

ADF Mobile wurde für JDeveloper 12*c* geplant und entwickelt, ist allerdings bereits in einer ersten Produkt-Version im Oracle JDeveloper 11*g* R2 (11.1.2.3) verfügbar. Da ADF Mobile als Extension entwickelt wurde (OSGi-basiert), ist die Weiterentwicklung nicht an Major Releases und Patch-Sets gebunden.

Fazit

Der JDeveloper ist Oracles Aushängeschild für die Entwicklung unternehmensweiter Java-EE- und SOA-Anwendungen. Mit JDeveloper 12c erscheint ein neues Release der IDE- und des integrierten ADF-Entwicklungs-Frameworks. Dieser Artikel hat einige der Neuerungen in Kurzform vorgestellt und auf etwaige Backports zu bestehenden JDeveloper-Versionen verwiesen.

Hinweis: Da sich der Oracle JDeveloper 12*c* noch in der Entwicklung befindet, kann es vorkommen, dass Funktionen, die in diesem Artikel vorgestellt wurden, anders oder gar verspätet im Produkt erscheinen. Zudem ist es dem Autor nicht möglich, an dieser Stelle konkrete Angaben zum Erscheinungsdatum von JDeveloper 12*c* zu machen. Das amerikanische Produktrecht verlangt diese Hinweise.

Links

- ADF Code Corner: http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html
- ADF Architecture Square: http://www.oracle.com/technetwork/developer-tools/adf/learnmore/adfarchitect-1639592.html
- OTN Harvest blog: https://blogs.oracle.com/jdevotnharvest

Frank Nimphius
frank.nimphius@oracle.com

Frank Nimphius ist als Senior Principal Product Manager für Oracle innerhalb des Produkt-Managements für Entwicklungswerkzeuge tätig. Im Rahmen seiner Tätigkeit für Oracle ist er spezialisiert auf die Evangelisierung und Weiterentwicklung der JDeveloper- und ADF-Produkte.

Asynchrone Datenabfragen mit DataFX

Hendrik Ebbers, Java User Group Dortmund

Mit JavaFX 2.x hat Oracle ein mächtiges neues API für Desktop-Anwendungen veröffentlicht. Das ursprünglich als Skriptsprache veröffentlichte JavaFX wurde komplett in Java umgeschrieben und wird seit Java 7 Update 6 standardmäßig mit Java ausgeliefert. Das Framework liefert eine Fülle an UI-Komponenten und Features, die alle modernen Ansprüche an User Interfaces wie Animationen, Multitouch, CSS-Styling, Charts etc. bedienen.

Mit JavaFX ist es möglich, moderne Rich Clients für eine wachsende Zahl an Endgeräten wie Desktop-PCs, Tablets oder Embedded-Devices zu entwickeln. So wurde beispielsweise vor Kurzem eine erste JDK-/JavaFX-Preview für das Raspberry Pi veröffentlicht (siehe http://www.guigarage.com/2012/12/my-first-steps-with-javafx-on-raspberry-pi/). DataFX, das API, das dieser Artikel vorstellt, erweitert JavaFX um einige nützliche Funktionen in den Bereichen der Daten-Visualisierung und -Beschaffung.

DataFX setzt sich aktuell aus zwei verschiedenen Bestandteilen zusammen. Der erste bietet verschiedenste Implementierungen für List-, Table- und Tree-Zellen. Ein Beispiel dafür ist die „MoneyTableCell", mit der sich Zahlen mit einer Währungs-Formatierung direkt in einer Tabelle darstellen lassen. Dank CSS-Unterstützung kann man sogar verschiedene Styles für positive und negative Beträge definieren. In diesem Artikel wird das Hauptaugenmerk auf den zweiten Teil von DataFX gelegt. Dieser unterstützt Entwickler mit einem „DataSource & DataReader"-API, durch das beliebiger Content asynchron zur Nutzung in JavaFX Controls abgerufen werden kann.

DataFX liegt seit Ende 2011 in der Version 1.0 vor und kann auf der Projektseite (siehe http://www.javafxdata.org) heruntergeladen oder per Maven/Gradle zu einem Projekt hinzugefügt werden. Listing 1 zeigt die Maven Dependency.

```
<dependency>
                    <groupId>org.javafxdata</groupId>
                    <artifactId>datafx-core</artifactId>
                    <version>1.0</version>
</dependency>
```

Listing 1

Multi-Threading in Desktop-Anwendungen

Ähnlich dem Event Dispatch Thread in Swing gibt es in JavaFX auch einen Thread, der für die Aktualisierung der GUI zuständig ist. Darin sind das komplette Layouten und Neuzeichnen der Benutzeroberfläche sowie sämtliche Benutzer-Interaktionen behandelt. Damit dürfen allerdings – wie bereits in Swing – alle Variablen, die an die Oberflächenkontrollen gebunden sind, nur in diesem Thread bearbeitet werden. Der Inhalt beziehungsweise das Datenmodell einer Tabelle darf also nicht außerhalb des JavaFX-Application-Thread geändert werden. Viele Business-Anwendungen beziehen ihre Daten jedoch von Backend-Systemen, Datenbanken oder Web-Services. Die Abfrage dieser Systeme kann einige Zeit in Anspruch nehmen und würde im schlimmsten Fall den JavaFX-Thread komplett blockieren, sodass ein Neuzeichnen der Oberfläche oder das Reagieren auf Benutzereingaben nur noch mit einer extremen Zeitverzögerung durchgeführt wird. Listing 2 zeigt ein Negativbeispiel, in dem durch einen Buttonklick die Oberfläche für längere Zeit komplett einfriert.

```
Button myButton = new Button("click me");
    myButton.setOnAction(new EventHandler<ActionEvent>() {

                    @Override
                    public void handle(ActionEvent arg0) {
                            String result = longDurationRequest();
                            myTextfield.setText(result);
                    }
            });
```

Listing 2

DataFX bietet nun die Möglichkeit, genau solche zeitaufwändigen Aufgaben in einen Hintergrund-Thread zu verlagern und nach einem erfolgreichen Laden der Daten diese im JavaFX-Thread bereitzustellen. Das Ganze ist im Grunde mit dem durch Java 6 eingeführten „SwingWorker" vergleichbar. Durch Nutzung des Property-API von JavaFX und dessen Binding-Features können Ergebnisse zusätzlich viel einfacher an der Oberfläche dargestellt werden.

iTunes-Suche mit DataFX

Anhand einer Web-Service-Abfrage ist die Nutzung von DataFX hier beispielhaft dargestellt. Als Basis dient das „iTunes Search"-

API von Apple, das, wie die meisten Web-Services heutzutage, REST nutzt und Daten im JSON-Format liefert. Den Web-Service kann man über die URL „https://itunes.apple.com/search?term=Nirvana“ selbst ausprobieren. In diesem Fall werden 50 in iTunes erhältliche Medien (Videos, Songs etc.) mit dem Schlagwort „Nirvana“ gesucht und zurückgeliefert. Listing 3 zeigt die JSON-Antwort des Service als Beispiel.

Zum Auslesen von Daten bietet DataFX das Interface „org.javafxdata.datasources.reader.DataSourceReader“ und verschiedene Implementierungen an. Als konkrete Implementierung wird für das Beispiel die Ableitung „org.javafxdata.datasources.reader.RestRequest“ genutzt, die das Ergebnis eines HTTP-Request als Rohdaten zurückliefert. Dieser kann einfach über einen Builder erstellt und parametrisiert werden (siehe Listing 4).

Durch den Builder kann der Endpoint komplett definiert werden. Betrachtet man die Dokumentation des iTunes-API unter „http://www.apple.com/itunes/affiliates/resources/documentation/itunes-store-web-service-search-api.html” genauer, findet man noch eine Fülle an Parametern zur besseren Definition der Suche. Im ersten Aufruf weiter oben wird zum Beispiel der „term“-Parameter in der URL angegeben. Dieser könnte ebenfalls direkt durch den Builder definiert werden. Zusätzlich wird noch angegeben, dass der Request JSON-Daten liefert (siehe Listing 5).

Um die Daten nun in Java-Objekte umzuwandeln, ist eine Klasse nötig, die das Datenmodell beschreibt. Da DataFX „Jackson” als JSON-Parser nutzt, kann diese Klasse mit Jackson-Annotations versehen werden. Die hier genutzte Klasse soll den Namen, den Preis und ein Vorschaubild zu iTunes Content kapseln (siehe Listing 6).

Um die rohen JSON-Daten zu parsen und für JavaFX bereitzustellen, ist eine „DataSource“ notwendig. DataSources dienen in DataFX dazu, Daten asynchron in einem Hintergrund-Thread zu laden. Für das konkrete Beispiel wird die DataFX Klasse „ObjectDataSource“ genutzt, die XML- oder JSON-Daten in Java-Objekte umwandelt.

Auch für diese Klasse gibt es wieder einen Builder, mit dem sich einfach eine konfigurierte Instanz erstellen lässt. Dem Builder wird der „DataSourceReader“ übergeben. Zusätzlich gibt es noch eine Besonderheit in den eingehenden JSON-Daten. Solange, wie der JSON-Input aus einem einzelnen Element oder aus einem Array von gleichen Elementen besteht, ist es klar, dass diese Elemente geparst werden sollen.

```
{
 “resultCount“:50,
 “results“: [
{“trackName“:“Smells Like Teen Spirit“,
“artworkUrl100“:“http://a190.phobos.apple.com/us/r1000/032/Features/4b/c4/cb/
dj.mjhyndcl.100x100-75.jpg“,
...},
...]
}
```

Listing 3

```
RestRequest request = new RestRequestBuilder(“http://itunes.apple.com“).
path(“search“).build();
```

Listing 4

```
RestRequest request = new RestRequestBuilder(“http://itunes.apple.com“).
path(“search“).format(Format.JSON).queryParam(“term“,“Nirvana“).
queryParam(“media“,“music“).build();
```

Listing 5

```
@JsonIgnoreProperties(ignoreUnknown = true)
public class ITunesMedia implements Serializable {
        private static final long serialVersionUID = 1L;
        private String artworkUrl100;
        private String trackName;
        private double trackPrice;
        public ITunesMedia() {}
        public String getArtworkUrl100() {
                return artworkUrl100;
        }
        public void setArtworkUrl100(String artworkUrl100) {
                this.artworkUrl100 = artworkUrl100;
        }
        public String getTrackName() {
                return trackName;
        }
        public void setTrackName(String trackName) {
                this.trackName = trackName;
        }
        public double getTrackPrice() {
                return trackPrice;
        }
        public void setTrackPrice(double trackPrice) {
                this.trackPrice = trackPrice;
        }
}
```

Listing 6

In dem hier genutzten Beispiel möchten wir allerdings nur einen Teil der Daten, und zwar das „results"-Array, parsen. Für diesen Zweck kann man dem Builder der „ObjectDataSource" das gewünschte Tag übergeben. Zuletzt muss nur noch eine „ObservableList" als Ergebnisliste übergeben werden. Dies ist eine mit JavaFX neu eingeführte Liste, deren Zustandsänderungen durch Listener überwacht werden können. Übergibt man diese Liste einer „ListView" oder „TableView" als Datenmodell, passt sich die Oberfläche automatisch an, sobald Daten in der Liste geändert werden. Sollte der Builder eine nicht leere Liste erhalten, werden alle neu erstellten Objekte ans Ende der Liste gehängt (siehe Listing 7).

Der Aufruf der „retrieve()"-Methode startet die Datenbeschaffung im Hintergrund. Die erhaltenen Daten werden dann im JavaFX-Application-Thread automatisch der Ergebnisliste hinzugefügt. Um die Daten nun in einer grafischen Oberfläche anzuzeigen, nutzen wir eine Tabelle, der wir die „ObservableList" als Datenmodell übergeben. Zusätzlich müssen noch die Tabellenspalten definiert werden. Hier wollen wir den Namen, das Vorschaubild und den Preis jeweils in einer eigenen Spalte anzeigen.

Wie man in Abbildung 1 sehen kann, werden momentan noch die Standardzellen-Implementierungen der JavaFX-Tabelle genutzt. Diese stellen die Werte einfach über ein „toString()" dar. Beim Namen ist dies absolut in Ordnung. Allerdings soll auch das Bild angezeigt werden und die Preisangabe in einer vernünftigen Formatierung erscheinen. Für Letzteres bietet DataFX die bereits in der Einleitung angesprochene „MoneyTableCell". Zellen zur Darstellung von Bildern werden mit dem nächsten Release von DataFX folgen. Listing 8 zeigt eine einfache Implementierung.

Über Cell-Factories sind die beiden Zellen-Implementierungen nun zur Darstellung der Daten in der Tabelle nutzbar. Die Erstellung der einzelnen Instanzen und das Setzen der Daten in die Zelle erfolgt automatisch durch JavaFX. Erweitert man die Anwendung noch um ein einfaches Suchfeld, kann man mit wenigen Zeilen Code eine einfache iTunes-Suche in JavaFX realisieren. Unter http://www.doag.org/go/java_aktuell/ebbers steht der komplette Sourcecode zum Download bereit.

Zusätzlich zu dem bisher besprochenen Code wird im Beispiel noch die JavaFX-GUI erstellt. Hierzu werden mehrere Builder-Klassen genutzt, durch die man schnell ein einfaches User-Interface aufbauen kann. Die Datenabfrage ist in einem „EventHandler" realisiert, der jedes Mal ausgeführt wird, sobald der Button eine Aktion feuert.

Abbildung 2 zeigt anhand eines Web-Service, wie einfach es ist, ein JavaFX-Programm zu erstellen, das mittels DataFX problemlos Daten aus einer externen Quelle visualisieren kann. Das fertige Programm ist als Maven-Projekt unter „https://github.com/guigarage/DataFX-iTunes-Demo" zu finden.

Weitere DataFX-Features

Neben den im Beispiel vorgestellten Funktionen bietet das DataSource-API von DataFX noch einige weitere Features, die hier kurz angeschnitten werden sollen.

Da die „retrieve()"-Methode der DataSource ein Objekt vom Typ „javafx.concurrent.Worker" zurückgibt, kann man zusätzlich die im Hintergrund laufende Datenbeschaffung überwachen oder abbrechen. Der Worker arbeitet intern mit JavaFX-Properties, an die jederzeit „Change-

Abbildung 1: Suchergebnisse in JavaFX

Abbildung 2: Die fertige iTunes-Suche

```
ObservableList<ITunesMedia> items = FXCollections
                                    .<ITunesMedia> observableArrayList();
ObjectDataSource<ITunesMedia> dataSource = ObjectDataSourceBuilder.<ITunesMedia>create().
dataSourceReader(request).resultList(items).itemTag("results").itemClass(ITunesMedia.class).build();
dataSource.retrieve();
```

Listing 7

```
public class ImageTableCell extends TableCell<ITunesMedia, String> {

        public ImageTableCell() {
                final ImageView view = new ImageView();
                setGraphic(view);

                itemProperty().addListener(new ChangeListener<String>() {

                        @Override
                        public void changed(ObservableValue<? extends String> observableValue,
                                        String oldValue, String newValue) {
                                try {
                                        if (newValue != null) {
                                                view.setImage(new Image(newValue, true));
                                        } else {
                                                view.setImage(null);
                                        }
                                } catch (Exception e) {
                                        view.setImage(null);
                                        e.printStackTrace();
                                }
                        }
                });
        }
}
```

Listing 8

```
dataSource.retrieve().stateProperty().addListener(new ChangeListener<State>() {
                        @Override
                        public void changed(ObservableValue<? extends State> observableValue,
                                        State oldValue, State newValue) {
                                System.out.println("Current State:" + newValue);
                        }
                });
```

Listing 9

Listener" registriert werden können (siehe Listing 9).

Mit den bereits vorgestellten Klassen „ObjectDataSource" und „RestRequest" ist es auch sehr einfach möglich, Daten an einen REST-Endpunkt zu senden. Hierzu kann für den Request zum Beispiel „POST" als HTTP-Methode ausgewählt und der DataSource explizit mitgeteilt werden, dass es kein Result gibt. Der HTTP-POST wird automatisch im Hintergrund durchgeführt und blockiert nicht den JavaFX-Application-Thread. Zusätzlich lassen sich neben den Query-Parametern auch Form- oder Request-Parameter übergeben. So kann man beispielsweise das Senden einer HTML-Form emulieren. Auch „Oauth" wird durch Setzen eines Key/Secret in den Grundzügen unterstützt. Die jeweiligen Builder-Klassen besitzen Methoden zur Konfiguration aller dieser Parameter. Neben der hier vorgestellten „ObjectDataSource" zur Erstellung von Java-Objekten aus XML- oder JSON-Daten gibt es in der Version 1.0 von DataFX noch eine „JdbcDataSource" sowie eine „CSVDataSource", die Java-Objekte aus Datenbank-Abfragen beziehungsweise CSV-Dateien erstellen können.

Fazit und Ausblick

In der aktuellen Version bietet DataFX viele Hilfestellungen, um Daten aus verschiedenen Quellen für eine JavaFX-Anwendung bereitzustellen. Dem Entwickler wird sämtliche Multi-Threading-Programmierung abgenommen, sodass er sich voll auf User-Interface und Datenmodell konzentrieren kann.

Für die nächste Version von DataFX wird das DataSource-API aktuell deutlich weiterentwickelt. Alle „Reader" und „DataSources" werden auf gemeinsame Interfaces und Basis-Klassen migriert, wodurch eine generische Programmierung einfacher fällt. Zusätzlich sollen sich die „DataSources" besser konfigurieren lassen. So kann man beispielsweise die Executor-Instanzen angeben, in denen die Hintergrund-Tasks ablaufen sollen. All diese Erweiterungen sollen die Nutzung und Integration von DataFX in großen Business-Anwendungen deutlich einfacher und attraktiver gestalten. Dazu kommen neue, konkrete Implementierungen von „Readern" und „DataSources", um Bilder und andere Daten im Hintergrund zu laden. Neue Zellen, wie die bereits angesprochene Zelle für Grafiken, und Utility-Klassen werden die nächste Version abrunden.

Hendrik Ebbers
hendrik.ebbers@web.de

Hendrik Ebbers ist Leiter der Java User Group Dortmund und beschäftigt sich größtenteils mit Rich-Client-Programmierung und Middelware. Sein aktuelles Hauptaugenmerk liegt auf JavaFX. Neben DataFX arbeitet er noch an weiteren Open-Source-Projekten wie JFXtras (siehe http://jfxtras.org) mit. Seine aktuellen Arbeiten und Forschungsergebnisse veröffentlicht er regelmäßig auf seinem eigenen Blog www.guigarage.com

Stolperfallen bei der Software-Architektur

Frank Pientka, MATERNA GmbH

Software wird heute agil entwickelt und stark an den geschäftlichen Nutzen ausgerichtet. Entscheidend für den Erfolg eines Software-Projekts ist ein pragmatischer und qualitätsorientierter Architektur-Entwurf, denn die negativen Auswirkungen falscher Entscheidungen können sich später als aufwändig und teuer erweisen. Der Artikel zeigt Schritte und Vorgehensweisen, damit der Architektur-Entwurf gelingt.

Über den Erfolg eines Produkts entscheiden Merkmale wie Qualität und Funktionsumfang. Bei der Definition der Anforderungen sollten daher die Erwartungen des Anwenders im Vordergrund stehen. Oft lassen sich Qualitätsmerkmale aus den Geschäftszielen ableiten, die das zu entwickelnde Produkt unterstützt. Außerdem sollte man funktionale und nicht-funktionale Merkmale frühzeitig unterscheiden und priorisieren. Ein wichtiger Grundsatz der Anforderungsanalyse besteht darin, die Problembeschreibung („Was?", „Warum?", „Wieso?") von der Lösung („Wie?", „Womit?") zu trennen.

Oft ist es gar nicht so einfach herauszufinden, was der Kunde möchte und was er wirklich braucht, um die von ihm beschriebenen Anforderungen zu erfüllen. Das fängt damit an, dass das Entwicklungsteam Einschränkungen oder Rahmenbedingungen für den Einsatz der Software gern mit dem Hinweis übersieht, diese Fakten seien doch allgemein bekannt. Dabei bilden genau diese Punkte – gemeinsam mit anderen Parametern wie den Qualitätsmerkmalen – das Gerüst einer Software-Architektur.

Definition der Qualität

Im Allgemeinen ist ein Qualitätsmerkmal eine Eigenschaft, die zur Unterscheidung von Produkten, Bausteinen oder Herstellungsprozessen in qualitativer (subjektiver) oder quantitativer (messbarer) Hinsicht herangezogen werden kann. Erst mit einem konkret beschriebenen Qualitätsszenario ist es möglich, das Erreichen der Qualitätsziele zu messen und eventuelle Korrekturmaßnahmen einzuleiten. Oft wird die Wechselwirkung zwischen den Qualitätsmerkmalen ausgeblendet, da nicht immer alle Ziele in gleichen Maßen und zu jeder Zeit erreicht werden können.

Aus den nicht-funktionalen Anforderungen können Entwickler übergeordnete Qualitätsziele ableiten, wie sie zum Beispiel im Standard ISO/IEC 9126 definiert sind. Diese Anforderungen sind unter dem Akronym „FURPPS" zusammengefasst, das für „Functionality" (Funktionalität), „Usability" (Bedienbarkeit), „Reliability" (Zuverlässigkeit), „Performance" (Effizienz), „Portability" (Übertragbarkeit) und „Supportability" (Wart-/Änderbarkeit) steht. Eine solche Kategorisierung der Anforderungen hilft beim systematischen Erfassen und Priorisieren aller Rahmenbedingungen und Merkmale für ein neues Produkt (siehe Abbildung 1). Doch viele IT-Organisationen bleiben bei diesen allgemeinen Qualitätskategorien stehen, anstatt ihr Produkt konkret mit spezifischen und messbaren Qualitätsszenarien zu beschreiben.

Durch die agilen Vorgehensweisen in der Software-Entwicklung steht der Kunde

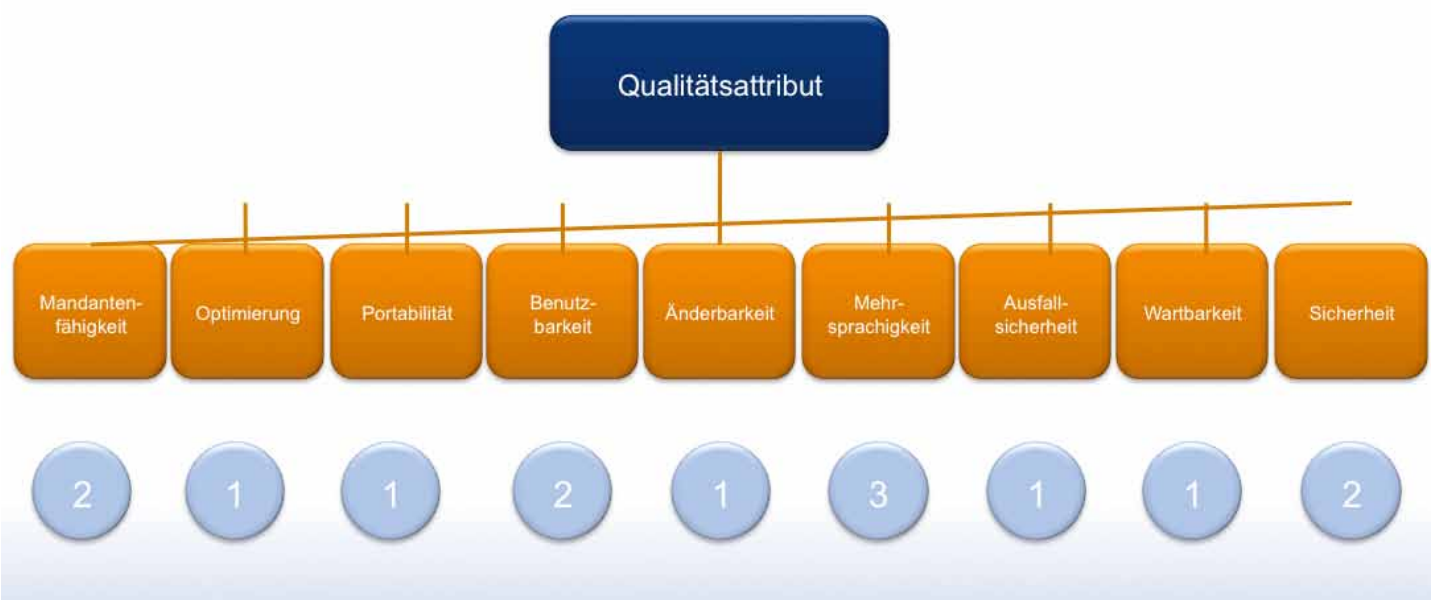


Abbildung 1: Priorisierter Qualitätsbaum

mit seinen Anforderungen im Vordergrund und die Funktionalität des Software-Produkts ist im Mittelpunkt der Aktivitäten. Dies hat zur Folge, dass die eigentlichen Qualitätsmerkmale gern in Vergessenheit geraten. Hier ist es Pflicht, mindestens die Basis- und Leistungsmerkmale zu erfüllen und Merkmale zu vermeiden, die zur Ablehnung des Produkts durch den Kunden führen können.

Fehlen bestimmte Qualitätsmerkmale später ganz oder sind diese nur mangelhaft umgesetzt, so kann das für den Erfolg des Software-Produkts kritisch sein. Falsch eingeschätzte Merkmale können sich so zu teuren Produktmängeln entwickeln oder zu erheblichem Mehraufwand in der Entwicklung führen. Da die Qualitätsmerkmale von Anfang an in der Architektur der Software berücksichtigt werden müssen, ist ein externes Review oftmals eine sinnvolle Investition.

Trennung von Form und Inhalt

Neben den fachlichen Anwendungsfällen sind auch die Qualitätsanforderungen mit Qualitätsszenarien zu beschreiben, um diese so besser (be-)greifbar und umsetzbar zu machen. Für beide Anforderungsarten wird zunächst die äußere logische Form mit Subsystemen festgelegt, die man dann auf die physischen Komponenten abbildet. Dazu ist kein (geniales) großes Design erforderlich, es sollen lediglich die großen Dinge möglichst früh feststehen.

Erst wenn die groben Elemente des zu entwerfenden Systems mit allen seinen Schnittstellen festgelegt sind, sollte man damit beginnen, dessen innere Struktur zu beschreiben. Die Architektur geht hier von außen nach innen. Dabei ist für den späteren Änderungsaufwand zu beachten, dass Formänderungen teurer sind als Strukturänderungen. Das Verhalten der Software sollte nach Möglichkeit einfach, ohne viel Zusatzaufwand anpassbar sein.

Die frühe Festlegung auf Subsysteme erleichtert einerseits die Planung und andererseits hilft der „äußere Rahmen“ auch bei der Ausgestaltung der „inneren Struktur“. Beim Umfang des Architekturentwurfs gilt auch hier der Design-Grundsatz des bekannten Bauhaus-Direktors Mies van der Rohe, dass weniger oft mehr ist.

Landkarten und Diagramme zur besseren Orientierung

Die Architektur-Dokumentation unterstützt beim Entwurf des Systems und macht die Architektur sowohl nachvollziehbar als auch bewertbar. Sie ist sowohl zentrales Konstruktions- als auch Kommunikationsmittel für alle Projektbeteiligten. Da sie viele Adressaten hat, ist es wichtig, diese bei der Erstellung zu berücksichtigen. Dabei hat es sich bewährt, unterschiedliche Sichtweisen für die jeweiligen Leser anzubieten. Deshalb ist der Leserkreis der Dokumentation vorher festzulegen und getroffene Architektur-Entscheidungen sind mit Annahme und Begründung festzuhalten. Für einen einheitlichen Entwurf ist wichtig, dass für dasselbe Problem immer dieselbe einheitliche Lösung zum Einsatz kommt.

Es gibt unterschiedliche Landkarten; dabei unterscheiden sich diese nicht nur vom Detaillierungsgrad her, sondern auch in der Art der Darstellung. So stellt ein U-Bahn-Plan nicht die korrekte Entfernung, sondern die genau Abfolge der Stationen in den Vordergrund. Ähnlich ist es mit der Mercator-Projektion des Erdglobus. Es ist ein Unterschied, ob man eine Weltreise auf dem Globus oder eine Wanderung in unbekanntem Gelände machen möchte. Deshalb muss es auch für die Software unterschiedliche Sichten und Darstellungen geben, die je nach Leser- und Nutzerkreis Details unterschiedlich darstellen.

Der Betrieb möchte beispielsweise genau wissen, auf welchen Systemen die Software installiert ist und über welche Ports und Protokolle diese miteinander kommunizieren. Einen Software-Entwickler interessieren eher die Details und was das Ganze zusammenhält, also wo er etwas verändern oder anpassen muss. Neben einer rein statischen Sicht wird dabei eine dynamische Laufzeitsicht mit den ausgetauschten Nachrichten oder Objekten immer wichtiger. Solche Informationen helfen nicht nur bei der Umsetzung, sondern erleichtern auch das systematische Testen.

Architektur-Stolperfallen

Da eine gute Dokumentation wichtiger Bestandteil einer Bewertung der Architektur ist, sind die Aktualität und die Konsistenz der Inhalte ein wichtiges Qualitätsmerkmal. Elementare Architektur-Entscheidungen sollen nachvollziehbar sein.

Zur Erstellung einer Architektur-Dokumentation hilft eine einheitliche Vorlage. Nur so kann erreicht werden, dass die Inhalte immer an der gleichen Stelle festgehalten sind und an die Beantwortung wichtiger Fragestellungen gedacht wird. Darüber hinaus erleichtert eine gute Dokumentation die Umsetzung und Weiterentwicklung des Systems. Deswegen ist diese keine lästige Pflicht, sondern ein notwendiges Mittel, um die Kommunikation und Entscheidungen in einem Softwareprojekt zu erleichtern. Gerade nicht konsequent dokumentierte, kommunizierte und gemanagte Architektur-Entscheidungen können zu einer hohen technischen Schuldenlast führen.

Vorab zu klären ist die Frage, wer wann wie viel und womit dokumentiert. Wenn mehrere Beteiligte an einem Dokument schreiben, ist es hilfreich, ein Kollaborationssystem wie ein Wiki mit Export-Möglichkeit nach PDF anzubieten. Die Bilder sollten möglichst mit wenigen, einfach zu bedienenden Werkzeugen erstellt sein, damit man diese einfach anpassen kann. Es ist hilfreich, sich bei der Verwendung einer einheitlichen Notation wie UML oder BPMN auf eine Untermenge und eine bestimmte Version zu beschränken, da die Lesbarkeit Vorrang vor der methodischen Korrektheit hat.

Eine der ersten Fallen ist die Abgrenzung, was und in welchem Umfang zu dokumentieren ist. Hier liegt die Würze in der Kürze, wobei hier immer die Balance zwischen „abstrakt“ und „konkret“ gefunden werden muss. Dabei sind Redundanzen und Lücken zu vermeiden, sodass es sinnvoll ist, das Dokument von Zeit zu Zeit redaktionell zu bearbeiten. Wichtig sind Randbedingungen und das Festhalten der angestrebten Leserzielgruppe. Zunächst werden die Produktvision mit wenigen Sätzen beschrieben und der Systemkontext grafisch dargestellt.

Über sieben Brücken muss eine gute Dokumentation gehen

In ihrem Buch „Documenting Software Architectures: Views and Beyond” stellen die Autoren Clements, Bachmann und Bass sieben Regeln für eine gute Architektur-Dokumentation auf:

1. Schreibe aus Sicht des Lesers
2. Vermeide unnötige Wiederholungen

Abbildung 2: ATAM-Inhalte

3. Vermeide Mehrdeutigkeiten, erkläre deine Notation
4. Verwende eine Standardstrukturierung
5. Halte Begründungen für Entscheidungen fest
6. Halte die Dokumentation aktuell, aber auch nicht zu aktuell
7. Überprüfe die Dokumentation auf ihre Gebrauchstauglichkeit

Beim Schreiben der Dokumentation macht hier nur die Übung den Meister. Gerade wegen der verschiedenen Adressaten der Dokumentation muss sowohl wegen der Breite als auch wegen der Tiefe der behandelten Themen oft ein Kompromiss gefunden werden, damit „wesentlich und aktuell" mehr ist als „allgemein und unklar". Das Erstellen einer zentralen, aktuellen und gut strukturierten Architektur-Dokumentation erleichtert die Kommunikation aller Projektbeteiligten und hilft, Entscheidungen fundierter zu treffen.

Entscheidungen und Kompromisse festhalten

Eine gute Software-Architektur muss nachvollziehbar und konsistent beschrieben sein. Da ein Software-Produkt viele Beteiligte hat, sollten die wichtigsten Stakeholder-Gruppen eigene Sichten erhalten. So benötigt der IT-Betrieb eine andere Sichtweise als der Entwickler oder die Fachabteilung.

Neben der Strukturierung der Systembestandteile und ihrer Beziehungen sollte man bei den Entwurfsentscheidungen die möglichen Auswirkungen auf die angestrebten Qualitätsmerkmale berücksichtigen. Dazu gehören zunächst das Erkennen kritischer Herausforderungen und positiven Ergebnisse sowie das Festhalten der getroffenen Kompromisse. Die am Software Engineering Institut (SEI) der Carnegie Mellon Universität (CMU) entwickelte „Architecture Tradeoff Analysis Method" (ATAM) hilft, Architektur-Entscheidungen hinsichtlich der Auswirkung auf die verschiedenen Qualitätsmerkmale zu bewerten (siehe Abbildung 2).

Deswegen sollte möglichst früh der Qualitätsbaum des Produkts mit seinen Merkmalen definiert werden, der an den Ästen im Laufe des Projekts immer detailreicher wird. Die priorisierten Qualitätsmerkmale werden den Architektur-Entscheidungen gegenübergestellt, um mögliche Risiken erkennen zu können. Zusätzlich werden empfindliche Stellen innerhalb der Architektur deutlich.

Die späteren Anforderungen und die dafür getroffenen Architektur-Entscheidungen sollten gegenüber diesen Qualitätsmerkmalen geprüft und auf mögliche Risiken untersucht werden. Ganz wichtig ist es, die eingegangenen Kompromisse, sogenannte „Trade-off-Points", festzuhalten, bei deren Design-Entscheidung zwei oder mehrere Qualitätsmerkmale wechselseitig betroffen sind.

Meinungen, Schätzungen und Bewertungen

Ein Workshop läuft nach einem definierten Muster ab, wobei Umfang, Inhalt und Untersuchungsschwerpunkte am Anfang mit dem Kunden auf dessen Bedürfnisse abgestimmt werden. Erfahrene Software-Architekten führen das Projekt-Review nach der ATAM-Methode in mehreren Phasen durch. Falls nötig, ergänzen Spezialisten für bestimmte Fachgebiete das Review-Team. Schwerpunkte des Beratungsangebots bilden Workshops und Interviews mit den Schlüsselpersonen des Entwicklungsprojekts, die Analyse von Projektergebnissen, Prozessen oder Dokumenten sowie eine umfangreiche Vor- und Nachbereitung.

Während der einzelnen Phasen stimmt man Zwischenergebnisse und die weiteren Schritte ab. Somit ist das Review-Vorgehen für den Kunden transparent und die Ergebnisse sind frühzeitig erkennbar. Ab-

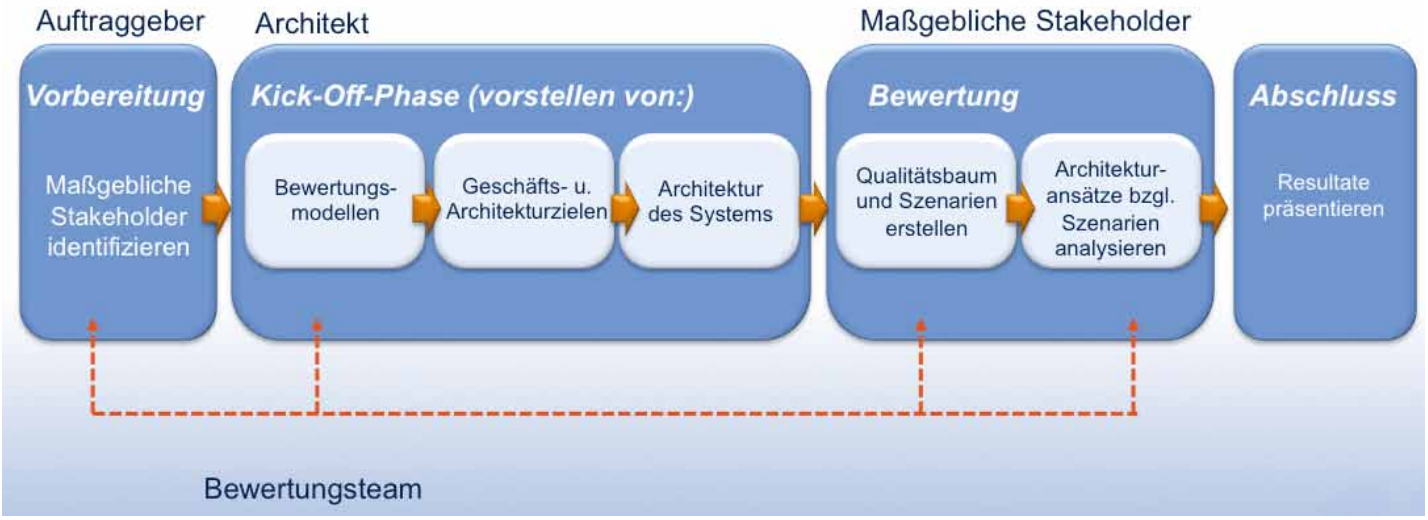


Abbildung 3: Review-Ablauf mit der Architecture Tradeoff Analysis Method

geschlossen wird die Architekturanalyse vor dem Entscheidungsgremium mit einer Abschlusspräsentation und der Übergabe des Review-Berichts. Oft lassen sich hier schon die ersten Maßnahmen festlegen, sodass deren Umsetzung in einem nachfolgenden Review überprüfbar ist.

Selten gibt es nur eine Ursache für eine schlechte oder schlecht umgesetzte Architektur-Dokumentation. Deswegen sind eine Priorisierung und die Kontrolle der Korrekturmaßnahmen mindestens genauso wichtig wie die Schwachstellen-Analyse.

Ein Architektur-Review ist somit eine effiziente Möglichkeit, die Risiken einer Software-Architektur rechtzeitig zu ermitteln, bevor ungeplante Projektänderungen das veranschlagte Budget sprengen. Außerdem macht das Angebot die getroffenen Design-Entscheidungen transparenter. Auch hier unterstützt die ATAM-Methode (siehe Abbildung 3).

Exkurs aus der Projektküche

Am Anfang eines Projekt-Reviews stand die Anforderung, eine bestehende, über mehrere Jahre entwickelte Web-Anwendung, einen B2B-Shop, zu analysieren und auf Optimierungspotenzial hin zu untersuchen. In einem Einstiegsworkshop mit der Geschäftsleitung wurden zunächst die aktuellen und zukünftigen Businessziele definiert, an der sich eine zukünftige Shop-Architektur messen lassen soll. Danach wurden mit den Produktverantwortlichen, wie dem Projektleiter und dem Architekten des Shop-Systems, die angestrebten Qualitätsziele als Baum visualisiert und priorisiert. Zu jedem Qualitätsziel kamen mehrere Unterziele mit dazugehörigen Szenarien hinzu.

Erst danach wurden Dokumentation und Code daraufhin begutachtet, inwiefern diese die identifizierten Ziele unterstützen oder welche Maßnahmen nötig sind, um diese besser zu erreichen.

Für die Bewertung des Codes und des Designs sowie für die Erstellung von aussagekräftigen Metriken haben sich die Auswertung des Commit-Protokolls des Versionswerkzeugs und die Berichte der statischen Codeanalyse bewährt.

Über Einzelinterviews wurden die Informationen vertieft, die Befunde verifiziert sowie die Ergebnisse und weiteren Maßnahmen in einer Zwischenpräsentation abgestimmt. Es war wichtig, dass alle Beteiligten immer eine weitgehende Einsicht in den Stand und die weiteren Schritte hatten. Da es bei der Umsetzung der Maßnahmen wichtig war, alle Beteiligten mit ins Boot zu holen, wurden unterstützende Maßnahmen wie begleitende Reviews und Coachings durchgeführt. So konnte der nachhaltige Erfolg der Umsetzung der Review-Ergebnisse gewährleistet werden.

Man sollte die Architektur der Software und ihre Dokumentation auch regelmäßig extern überprüfen lassen. Da Fehler frühzeitig erkannt und entsprechende Maßnahmen rechtzeitig eingeleitet werden können, ist das Geld hier gut investiert.

Fazit

Das Zitat von Winston Churchill „Es ist ein großer Vorteil im Leben, die Fehler, aus denen man lernen kann, möglichst früh zu begehen", gilt auch für die Software-Entwicklung. Je früher im Entwicklungsprozess Fehler erkannt werden, desto einfacher lassen sich diese beseitigen beziehungsweise geeignete Gegenmaßnahmen einleiten.

Da die Qualitätsmerkmale von Anfang an berücksichtigt werden müssen, ist ein frühes und regelmäßiges externes Review sein Geld wert. Nur so können teure Fehlentscheidungen rechtzeitig vermieden werden. Spätere potenzielle Stolpersteine können durch das Frühwarnsystems des Reviews oder eine „zweite unabhängige Meinung" erkannt und geeignete Gegensteuermaßnahmen eingeleitet werden. Die Aussage von Henry Ford „Qualität ist, wenn der Kunde zurückkommt und nicht das Produkt", gilt noch immer.

Frank Pientka
frank.pientka@materna.de

Frank Pientka ist Dipl.-Informatiker und als Senior Software Architect bei der MATERNA GmbH in Dortmund tätig und Gründungsmitglied des iSAQBs. Seit über zehn Jahren unterstützt er Firmen bei der Umsetzung tragfähiger Software-Architekturen. Dabei führt er externe Reviews und Architektur-Bewertungen als Gutachter durch.

Hochverfügbarkeit mit dem JBoss AS 7

Heinz Wilming und Immanuel Sims, akquinet AG

Die Gruppierung mehrerer Server zu einem Cluster ist eine wichtige Eigenschaft eines Java-EE-Servers für den Betrieb kritischer Geschäftsanwendungen. Dies gewährleistet zum einen eine hohe Verfügbarkeit und zum anderen die Skalierbarkeit der Anwendung bei wachsender Last.

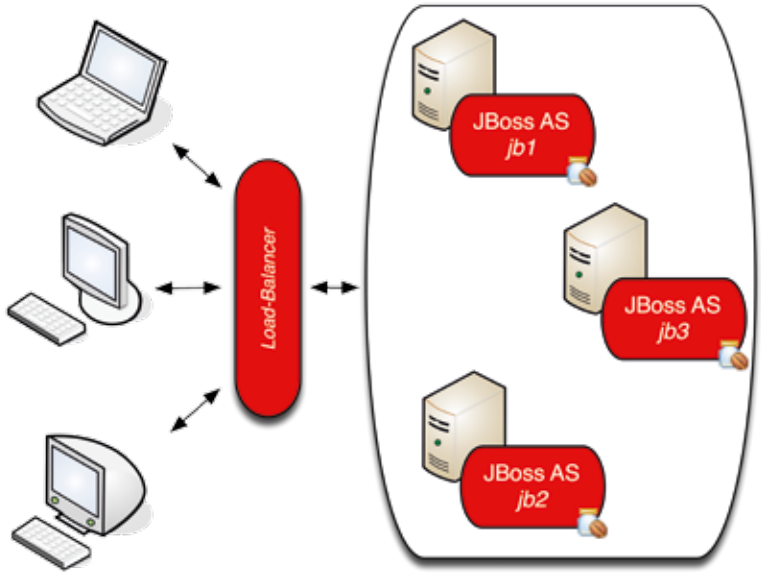


Abbildung 1: Einfache Cluster-Topologie mit Lastverteilung

Der Artikel gibt einen Überblick über die Clustering-Funktionalitäten des JBoss Application Server (AS) in der Version 7 und zeigt anhand einer Beispielanwendung, wie man eine Java-EE-Anwendung clusterfähig implementiert und einen einfachen Cluster betreibt. Er bietet darüber hinaus einen Ausblick auf die Nutzung des Domain Mode zum dynamischen Skalieren und zum Verwalten einer Cluster-Umgebung.

Einführung

Da verschiedene Ausfälle, etwa bei der Stromversorgung oder der Hardware, auf längere Zeit nicht vermeidbar sind, ist es meist notwendig, Software redundant auszulegen, damit bei einer Störung das System in bestimmtem Umfang verfügbar bleibt. JBoss AS 7 bietet hier die Möglichkeit, verschiedene Server zu einem Cluster zu kombinieren und die interne Server-Topologie vor den Klienten zu verbergen.

Hochverfügbarkeit einer Anwendung kann dabei in unterschiedlichen Ausprägungen erreicht werden. Für eine Web-Anwendung wäre es beispielsweise möglich, die Anfragen über einen Load-Balancer an die einzelnen Server-Instanzen des Clusters zu verteilen (siehe Abbildung 1).

Bei dieser Topologie werden die Server unabhängig voneinander betrieben und haben keinerlei Kenntnis davon, dass sie gemeinsam in einem Cluster arbeiten. Das setzt voraus, dass der eingesetzte Load-Balancer Anfragen eines Klienten immer an den gleichen Server weiterleitet und erst bei einer Störung auf einen anderen ausweicht. Das ist notwendig, da die Sitzungsdaten eines Klienten nicht zwischen den beteiligten Servern repliziert werden. Die Assoziation der Sitzung eines Klienten mit dem jeweiligen Server wird als „sticky session" bezeichnet. Stürzt ein Server ab, gehen alle transienten Daten verloren. Der Klient wird allerdings mit der nächsten Anfrage auf einen anderen Server umgeleitet (Failover) und könnte auf Basis eines gemeinsamen persistenten Datenbestands weiterarbeiten. Dieser Ansatz lässt sich relativ leicht umsetzen und ist effizient. Jedoch können mitunter wichtige Daten verlorengehen. Für einen Webshop würde ein verlorener Warenkorb bedeuten, dass der Kunde verärgert ist und die ausgewählten Artikel nicht kaufen wird.

Damit auch transiente Sitzungsdaten nach einem Ausfall für die Klienten verfügbar sind, müssen sie auf anderen Servern des Clusters verfügbar sein. Dazu sind die Sitzungsdaten beispielsweise einer HTTP-Session oder einer zustandsbehafteten EJB-Komponente über einen Cache zwischen den jeweiligen Server-Instanzen zu verteilen (siehe Abbildung 2). Bei diesem Aufbau müssen die Server-Instanzen eines Clusters untereinander kommunizieren und Kenntnis über die Cluster-Topologie haben, um die jeweiligen Sitzungsdaten zu replizieren. Die Ausfallsicherheit wird hierbei zulasten der Skalierbarkeit erhöht, da jeder Server neben seinen eigenen Daten aus Redundanzgründen auch die Sitzungsdaten anderer Server des Clusters vorhält.

Verwendete Technologien

Um Hochverfügbarkeit zu unterstützen, müssen die Daten innerhalb des Clusters repliziert werden, damit die Sitzungsdaten

auf mehreren Server-Instanzen redundant vorgehalten sind. Bei einem Server-Ausfall kann der Klient dann auf einen anderen Server des Clusters umgeleitet werden, ohne dass dieser hiervon Kenntnis nimmt. Im JBoss AS 7 wird dazu der verteilte Cache Infinispan [1] als Fundament für die Replikation der Sitzungsdaten verwendet. Innerhalb einer Java-EE-Anwendung sind transiente Zustände auf mehreren Ebenen zu verwalten, beispielsweise die Daten einer HTTP-Session oder der Zustand einer EJB-Komponente. Dementsprechend enthält der JBoss AS 7 vier vorkonfigurierte sogenannte „Cache-Container" für die Replikation der Daten:

- *web*
  Replikation von HTTP-Session-Daten
- *sfsb*
  Replikation der Verbindungs- und Sitzungsdaten von Stateful-Session-Beans
- *hibernate*
  Second-Level-Cache für JPA/Hibernate
- *cluster*
  Replikation allgemeiner Objekte in einem Cluster

Um tatsächlich Daten zwischen den einzelnen Server-Instanzen zu übertragen, benutzt Infinispan „JGroups" [2] als grundlegendes Subsystem. Dies ist ein Framework für die Kommunikation innerhalb einer Gruppe von verteilten Knoten. Es stellt Operationen zur Verfügung, um neue Knoten zu einem Cluster hinzuzufügen, Knoten explizit zu entfernen sowie fehlerhafte Knoten zu erkennen und automatisch auszusortieren. JGroups unterstützt verschiedene Netzwerk-Protokolle, um den Anforderungen des jeweiligen Netzes gerecht zu werden.

JBoss AS 7 beinhaltet zwei vorkonfigurierte Protokoll-Stacks für eine verlässliche Kommunikation zwischen den Server-Instanzen des Clusters: einen UDP-basierten (Standard) und einen TCP-basierten Protokoll-Stack. Beide Stacks verwenden IP-Multicasts zum Auffinden anderer Server-Instanzen. Zum Verteilen von Informationen hingegen verwendet lediglich der UDP-basierte Stack IP-Multicasts.

Lastverteilung und Failover

Beispielsweise die Apache-Module „mod_jk" [3] oder „mod_cluster" [4] setzen Lastverteilung und Failover für eine Web-Anwendung um. Die „mod_cluster"-Module bieten in Verbindung mit JBoss AS 7 einige Vorteile, da dieser ein Subsystem für die Integration von „mod_cluster" enthält. Dieses bietet unter anderem die Möglichkeit, verschiedene Metriken für eine intelligente Lastverteilung der Anfragen auf Seiten des HTTP-Servers mit einzubeziehen. Die Lastkennzahl wird serverseitig berechnet und berücksichtigt Faktoren wie zum Beispiel die Anzahl aktiver HTTP-Sessions, die CPU-Auslastung oder den Speicherverbrauch des Servers. Ein weiterer Vorteil ist, dass die Server-Instanzen im Gegensatz zu „mod_jk" dynamisch über IP-Multicast erkannt und eingebunden werden.

Lastverteilung und Failover für entfernte EJB-Klienten werden von der JBoss-EJB-Client-Bibliothek [5] unterstützt. Die Cluster-Topologie wird einem entfernten EJB-Klienten nach einer initialen Verbindung mitgeteilt. Weitere EJB-Aufrufe sind ab diesem Zeitpunkt bereits ohne zusätzliche Konfiguration standardmäßig per Zufalls-Algorithmus an die Server-Instanzen verteilt. Werden die Sitzungsdaten einer zustandsbehafteten EJB-Komponente repliziert, erfolgt im Fehlerfall ein transparentes Failover. Zur Erhöhung der Ausfallsicherheit sollten jedoch mehrere anfängliche Verbindungen konfiguriert sein, da ansonsten bei einem Ausfall der initialen Server-Instanz die Cluster-Topologie nicht übertragen werden kann.

Cluster-fähige Anwendung

Damit eine Java-EE-Anwendung in einem Cluster betrieben werden kann, muss sie entsprechend konfiguriert sein. Erst dann

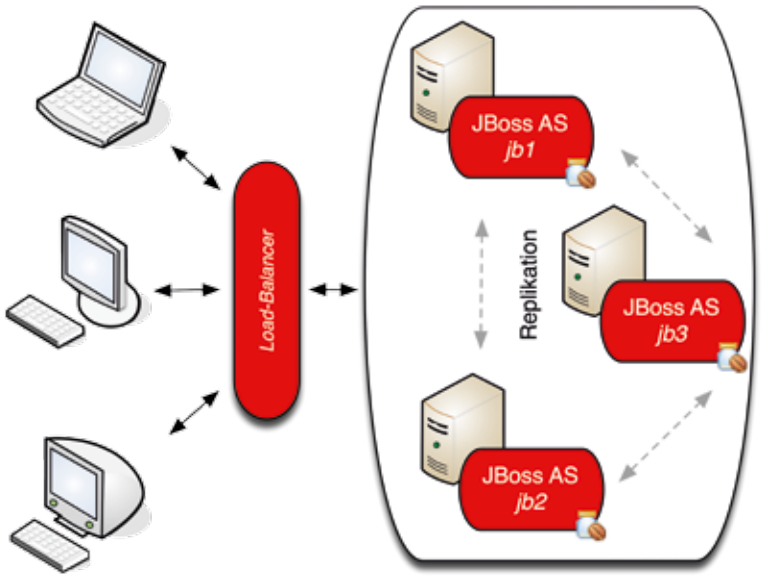


Abbildung 2: Cluster-Topologie mit Failover und Lastverteilung

```
@org.jboss.ejb3.annotation.Clustered
@Stateful
public class ClusteredStatefulBean {
  ...
}
```

Listing 1

```
<web-app>
  <distributable/>
</web-app>
```

Listing 2

stehen die Dienste des Applikationsservers für den Betrieb eines Clusters, wie die Cache-Container zur Replikation der Daten oder die Gruppenkommunikation über JGroups, der Anwendung zur Verfügung.

Enterprise-JavaBeans sind die Kern-Komponenten einer Java-EE-Anwendung. EJB-Komponenten bieten unter anderem deklaratives Transaktions- und Sicherheits-Management. Um eine EJB-Komponente in einem Cluster zu betreiben, muss dieses dem Server durch eine Annotation oder per XML-Deployment-Deskriptor mitgeteilt werden. Listing 1 zeigt, wie beispielsweise eine Stateful-Session-Bean aussieht.

Es ist jedoch zu beachten, dass bei einer Stateful-Session-Bean die Replikation mit einem zusätzlichen Aufwand verbunden ist. Da bei der Replikation die Komponente serialisiert werden muss, wird sie zuvor passiviert. Das bedeutet, dass die Komponente vor jedem Replikationsvorgang passiviert und entsprechend vor dem nächsten Aufruf wieder aktiviert werden muss. Dies übernimmt der Applikationsserver. Der Entwickler sollte sich der hinzugefügten Komplexität bewusst sein, da die Performanz beeinträchtigt wird.

Auch Stateless-Session-Beans können mit den Annotation „@Clustered" versehen sein. Es ist zwar kein Zustand im Cluster zu replizieren, jedoch können dann die Aufrufe entfernter Klienten dynamisch auf den gesamten Cluster verteilt werden.

Bei Web-Anwendungen muss ebenso kenntlich sein, dass die HTTP-Sessions repliziert werden sollen. Das wird dem Applikationsserver durch das Element „<distributable/>" in der Datei „web.xml" mitgeteilt (siehe Listing 2).

Der Domain Mode

Der Domain Mode ist eine Betriebsart des JBoss AS 7. Er ermöglicht eine zentralisierte Verwaltung mehrerer Server-Instanzen. Das umfasst die Konfiguration, die Bereitstellung von Anwendungen sowie das Starten und Stoppen von Server-Instanzen. Dazu werden die einzelnen Server jeweils einer Servergruppe zugeordnet.

Alle Server einer Servergruppe unterliegen denselben Konfigurations-Richtlinien und betreiben die gleichen Anwendungen. Die Konfigurationen können bei Bedarf auch mit individuellen, umgebungsspezifischen Eigenschaften parametrisiert sein. Da innerhalb einer Domäne mehrere Server-Instanzen verwaltet werden können und auch zur Laufzeit Server-Instanzen einer Servergruppe hinzugefügt und entfernt werden, ist der Domain Mode ideal zur Verwaltung eines Clusters geeignet.

Eine Domäne kann aus mehreren physischen oder virtuellen Hosts bestehen. Jeder Host benötigt einen Host-Controller, der den Lebenszyklus der Server-Instanzen verwaltet. Die gesamte Domäne wird von einem Domain-Controller verwaltet und stellt die Schnittstelle für Verwaltungsaufgaben zur Verfügung (siehe Abbildung 3).

Die Controller und die Server-Instanzen von jedem Host sind separate JVM-Prozesse. Es gibt zusätzlich noch einen Prozess-Controller. Dieser ist ebenfalls ein separater JVM-Prozess, der dafür verantwortlich ist, den Lebenszyklus des Host-Controller-Prozesses zu überwachen. Aus Gründen der Ausfallsicherheit und um die Verwaltungsaufgaben von den Server-Instanzen der Domäne zu trennen, ist es empfehlenswert, den Domain-Controller auf einem gesonderten Host zu betreiben.

Für die Konfiguration einer Domäne sind die beiden Dateien „domain.xml" und „host.xml" aus dem Verzeichnis „$JBOSS_HOME/domain/configuration" relevant.

„domain.xml" enthält die Konfigurationsrichtlinien der Domäne. Der Domain-Controller verwaltet die Datei zentral und der Host-Controller verteilt die Konfiguration an die Server-Instanzen. Die Konfiguration enthält die Servergruppen, verschiedene Profile, Socket-Binding-Groups sowie die Zuordnung von jeweils einem Konfigurationsprofil und einer Socket-Binding-Group zu einer Servergruppe.

Host- und Domain-Controller sind jeweils über die Datei „host.xml" konfiguriert. Die Konfiguration umfasst, welche Server auf einem lokalen Host existieren, wie der Domain-Controller erreicht wird und welcher Host-Controller die Rolle des Domain-Controllers übernimmt.

Um einen Domain-Controller aufzusetzen, richtet man als Erstes eine JBoss-AS-7-Distribution ein und legt dort einen Management-Benutzer an. Dieser dient den Host-Controllern zur Authentifizierung und kann mithilfe des Skripts add-user.sh angelegt werden. Wichtig ist, dass der Benutzer dem „ManagementRealm" zugeordnet wird und zur Anmeldung an einem

```
<domain xmlns="urn:jboss:domain:1.3">
  ...
  <server-groups>
    <server-group name="cluster-ha" profile="ha">
      <socket-binding-group ref="ha-sockets" />
    </server-group>
  </server-groups>
</domain>
```

Listing 3

```
<host name="master" xmlns="urn:jboss:domain:1.3">
  ...
  <domain-controller>
    <local/>
  </domain-controller>
  ...
</host>
```

Listing 4

```
./domain.sh \
  --host-config=host-master.xml \
  -Djboss.bind.address.management=10.0.0.1
```

Listing 5

```
<host name="server1" xmlns="urn:jboss:domain:1.3">
  ...
</host>
```

Listing 6

```
<host name="server1" xmlns="urn:jboss:domain:1.3">
  ...
  <domain-controller>
    <remote host="${jboss.domain.master.address}"
port="9999" username="domainadmin" security-
realm="ManagementRealm"/>
  </domain-controller>
  ...
</host>
```

Listing 7

```
...
  <server-identities>
    <secret value="c2VjcmV0"/>
  </server-identities>
...
```

Listing 8

```
...
  <servers>
    <server name="server-one" group="cluster-ha" />
  </servers>
...
```

Listing 9

```
./domain.sh \
  --host-config=host-slave.xml \
  -Djboss.domain.master.address=10.0.0.1 \
  -Djboss.bind.address=10.0.0.2 \
  -Djboss.bind.address.management=10.0.0.2
```

Listing 10

```
./jboss-cli.sh --connect 10.0.0.1:9999
[domain@10.0.0.1:9999 /] deploy ./cluster-example-1.0-SNAPSHOT.war --server-groups=cluster-ha
```

Listing 11

Für den Domain-Controller kann die vorkonfigurierte Datei „host-master.xml" benutzt werden. Diese beinhaltet die notwendige Konfiguration für den Host, wie zum Beispiel das physikalische Network-Binding der Management-Schnittstellen oder die JVM-Konfiguration. Mit dieser Konfiguration wird ein Host- zum Domain-Controller (siehe Listing 4). Dieser kann mit dem Skript „domain.sh" aus dem „bin"-Verzeichnis gestartet werden (siehe Listing 5).

Dem Domain-Controller können nun beliebig viele Hosts hinzugefügt werden. In unserem Beispiel kommt die vorkonfigurierte Datei „slave-host.xml" für die Konfiguration des Host-Controllers zur Anwendung. Diese Datei ist ebenfalls im Verzeichnis der Domain-Konfiguration der JBoss-AS-7-Distribution enthalten. Es muss ein eindeutiger Name für jeden Host in der Domäne gewählt werden, damit es nicht zu Konflikten kommt. Andernfalls ist der Default-Wert der Hostname des Servers (siehe Listing 6). Die Konfiguration in Listing 7 spezifiziert, wie der Domain-Controller erreichbar ist. Der Benutzername ist der zuvor erstellte Management-Benutzer. Außerdem ist das Base64-kodierte Passwort bekanntzugeben (siehe Listing 8).

Als letzter Schritt können nun die tatsächlichen Server-Instanzen hinzugefügt werden (siehe Listing 9). Dabei wird eine Server-Instanz immer genau einer Servergruppe der Domäne zugeordnet. Ein Host wird ebenfalls mit dem Skript „domain.sh" aus dem bin-Verzeichnis gestartet (siehe Listing 10).

Eine Beispiel-Anwendung

Die Anwendung unter [6] ist mit dem Kommando „mvn package" konstruiert und das resultierende Web-Archiv im Applikations-Server bereitgestellt. Die Anwendung besteht aus einer Stateful-Session-Bean, die bei jedem Aufruf einen Zähler inkremen-

Domain-Controller berechtigt ist. Das Passwort des neuen Benutzers wird zusätzlich Base64-kodiert ausgegeben. Dieser Wert ist später für die Authentifizierung des Host-Controllers erforderlich.

Der JBoss AS 7 ist bereits für den Betrieb eines Clusters vorkonfiguriert und bietet entsprechende Konfigurations-Profile an. Für kleinere Cluster-Topologien genügt es zunächst, am Ende der „domain.xml"-Datei eine Servergruppe zu konfigurieren (siehe Listing 3).

Jede Servergruppe benötigt einen eindeutigen Namen und muss einem Profil der Domain-Konfiguration zugeordnet werden. Die Standard-Konfiguration enthält vier vorkonfigurierte Profile:

- *default*
  Java-EE-Web-Profil plus einige Erweiterungen wie RESTful-Web-Services
- *full*
  Java-EE-Full-Profil und alle Server-Fähigkeiten ohne Clustering
- *ha*
  Standard-Profil mit Clustering-Fähigkeiten
- *full-ha*
  Vollständiges Profil mit Clustering-Fähigkeiten

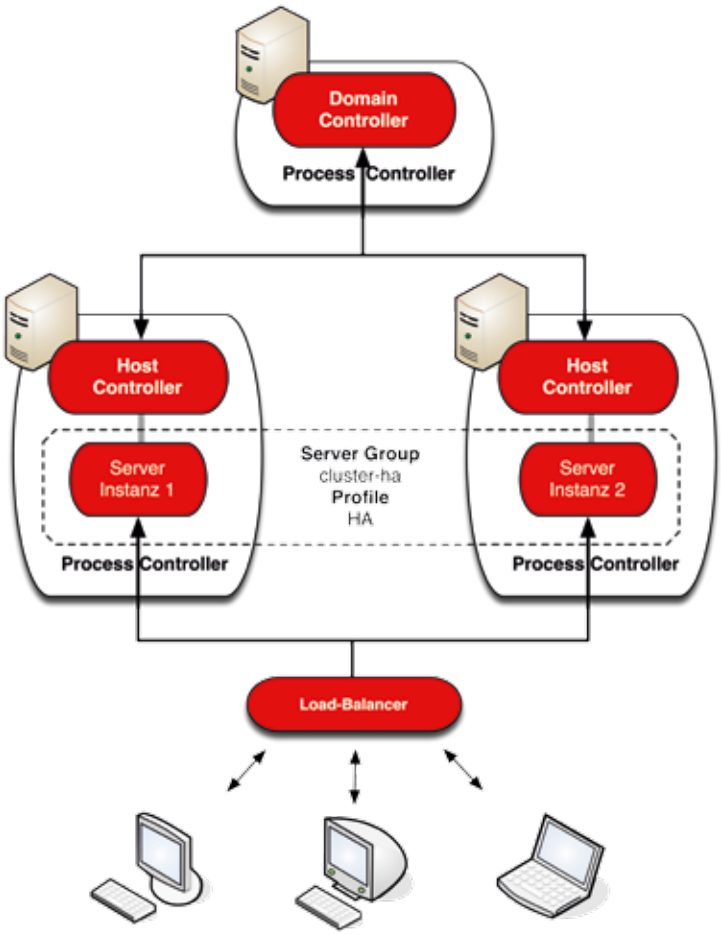


Abbildung 3: Verwaltung einer Cluster-Topologie im Domain Mode

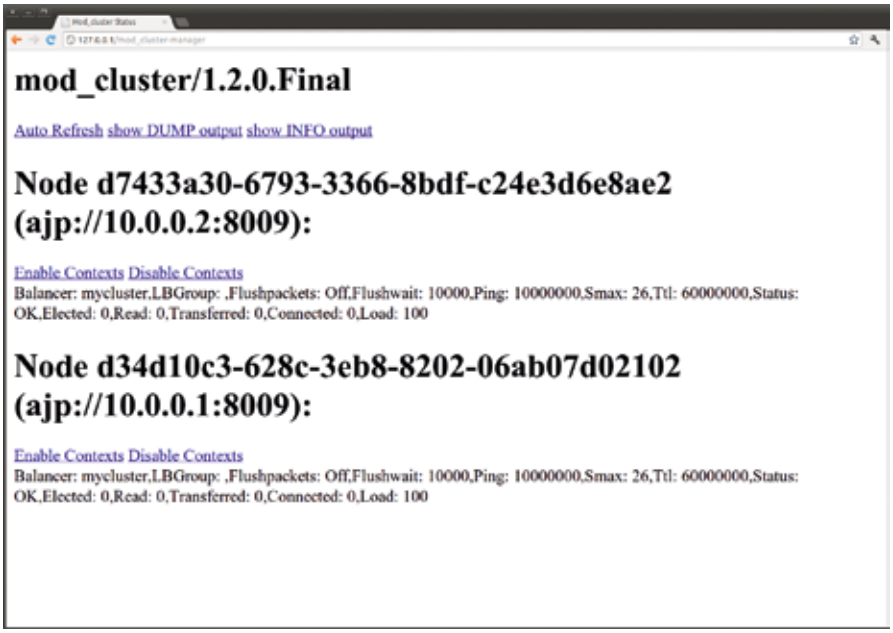


Abbildung 4: Die Management-Konsole von „mod_cluster“

tiert, und aus einer Stateless-Session-Bean, die den Namen des ausführenden Servers zurückgibt. Beide Session-Beans können über eine JSF-Seite aufgerufen werden.

Auf der „mod_cluster“-Projektseite [4] lässt sich eine Binär-Distribution des Apache-Web-Servers mit „mod_cluster“ herunterladen. Diese ist vorkonfiguriert und muss lediglich mit „/opt/jboss/httpd/sbin/apache-ctl start“ im „root“-Verzeichnis entpackt und gestartet werden. Wenn der Cluster bereits gestartet ist, sind die registrierten Server unter „http://127.0.0.1:6666/mod_cluster_manager“ sichtbar (siehe Abbildung 4).

Die Anwendung kann entweder über die Web-Konsole des Domain-Controllers oder über das Command Line Interface (CLI) bereitgestellt werden. Das CLI wird per Skript aus dem bin-Verzeichnis des JBoss AS gestartet (siehe Listing 11). Die Anwendung ist danach über „http://127.0.0.1:6666/cluster“ verfügbar. Die Anfragen werden über den Cluster verteilt und bei einem Ausfall erfolgt ein Failover.

Fazit

JBoss AS 7 bietet ausgereifte Technologien, um kritische Java-EE-Anwendungen hochverfügbar auf Ebene der Middleware zu betreiben. Die enthaltenen Profile sind bereits für kleinere Cluster-Topologien entsprechend konfiguriert. Der Domain Mode des Applikationsservers verwaltet komfortabel mehrere Server-Instanzen. Jedoch birgt ein Cluster eine enorme Komplexität, die Detailkenntnisse der grundlegenden Technologien „JGroups“ und „Infinispan“ erfordert. Für geschäftskritische Anwendungen ist daher der Einsatz der offiziellen Produkte von Red Hat zu empfehlen. Diese sind ebenfalls Open-Source-Software, basierend auf den Community-Produkten.

Darüber hinaus ist zu beachten, dass eine hohe Verfügbarkeit die Skalierbarkeit einer Anwendung einschränken kann. In der nächsten Ausgabe werden wir auf dieses Spannungsfeld eingehen und Cluster-Topologien diskutieren, die auch die Anforderungen an Rechenleistung und Verfügbarkeit gleichermaßen berücksichtigen.

Weiterführende Links

[1] http://www.jboss.org/infinispan
[2] http://www.jgroups.org
[3] http://tomcat.apache.org/connectors-doc
[4] http://www.jboss.org/mod_cluster
[5] https://github.com/jbossas/jboss-ejb-client
[6] https://github.com/akquinet/jbosscc-as7-examples/tree/master/cluster-example

Heinz Wilming
heinz.wilming@akquinet.de

Heinz Wilming ist Leiter des JBoss Competence Center der akquinet AG. Er beschäftigt sich dort mit Technologien und Architekturen für verteilte Anwendungen. Sein Fokus liegt dabei auf der Java-EE-Plattform und den Open-Source-Technologien.

Immanuel Sims
immanuel.sims@akquinet.de

Immanuel Sims ist als Werkstudent bei der akquinet AG tätig. Er studiert Informatik an der Humboldt Universität in Berlin und hat sich intensiv mit dem Thema „Clustering“ im Kontext des JBoss AS 7 beschäftigt.

Analyse unstrukturierter Datenquellen anhand frei definierbarer Algorithmen – ein Framework

Prof. Dr.-Ing. Martin Schulten und B. Sc. Christoph Seidel, Westfälische Hochschule

Herkömmliche Internet-Suchmaschinen helfen zwar beim Auffinden von Webseiten, doch sie leisten nur wenig mehr als die Feststellung, ob bestimmte Schlüsselwörter auf einer Seite vorkommen oder nicht. Bewertungskriterien sind für alle Suchen vom Betreiber pauschalisiert definiert und gelten als gut gehütete Geheimnisse. Dieser Artikel beschreibt ein Framework, mit dessen Hilfe man einen eigenen Suchserver aufsetzen kann und bei dem man die Bewertung von Treffern selbst definiert. Die flexibel gestaltete Architektur erlaubt die Analyse prinzipiell von beliebigen Datenquellen wie beispielsweise auch lokalen Datenbanken oder sozialen Netzwerken.

Jeder kennt das Problem aus dem Web: Google, Bing und andere Suchmaschinen bringen in der Regel sehr viele Ergebnisse für einen Suchlauf hervor. Die Qualität einer Suchmaschine zeigt sich jedoch nicht in der hohen Anzahl der Treffer, sondern in der Fähigkeit, die Relevanz von Treffern zu bewerten. Dafür haben Anbieter eine Fülle von Algorithmen entwickelt, die sie mehr oder weniger erfolgreich kombinieren, um ein für den Nutzer zufriedenstellendes Ranking zu erzeugen. Abgesehen von einigen pauschalen Einstellungen sowie Sortier- und Filter-Funktionen hat der Anwender keinen Einfluss darauf. Zur Extraktion der wirklich interessanten Daten ist diese Vorgehensweise oftmals nicht ausreichend.

Es gibt zwei Ursachen für diese Problematik: Einerseits kann die Suchmaschine nicht wissen, was man eigentlich genau finden und anhand welcher Kriterien man „gute" von „schlechten" Treffern unterscheiden will. Daher schert sie alle Nutzer über einen Kamm (Pauschalisierung).

Außerdem ist es in vielen Fällen so, dass Informationen unstrukturiert beziehungsweise nach falschen Gesichtspunkten strukturiert vorliegen. So ist bei Webseiten meist der Inhalt anhand des gewünschten Layouts und nicht nach inhaltlichen Gesichtspunkten strukturiert. Es findet auch keine Verschlagwortung von Inhalten statt. Somit ist es äußerst schwierig, die gebotenen Inhalte automatisiert zu analysieren (fehlende Semantik): Ist das gefundene Textfragment eine Überschrift oder eine Produktbezeichnung? Handelt es sich bei der gefundenen Zahl um den Preis oder das Gewicht eines Produkts?

Insbesondere bei hohen Anforderungen an die Qualität der gelieferten Treffer sowie bei automatisierten Analysen zeigen sich schnell die Grenzen: Was ist, wenn man zum Beispiel nur die Seiten finden will, auf denen der Begriff „A" in der Überschrift und der Begriff „B" im ersten Satz danach steht? Wenn man die Treffer nach geometrischem Abstand der Suchbegriffe auf der Seite zueinander sortiert sehen möchte? Wenn man nur die Seiten sehen will, auf denen in der Nähe eines bestimmten Begriffs eine Zahl zwischen „23" und „42" steht, die Zahl eine Preisangabe ist und sie außerdem im Browser nicht oberhalb des Suchbegriffs selbst dargestellt wird?

Eigenen Framework entwickelt

An der fehlenden semantischen Beschreibung von Webseiten können wir zumindest kurzfristig nichts ändern. Wir können aber sehr wohl die Pauschalisierung von Analysen umgehen und eigene Mechanismen implementieren. Zur Extraktion relevanter Daten aus unstrukturierten Quellen braucht man Algorithmen, die auf den jeweiligen Suchauftrag spezialisiert sind. An der Westfälischen Hochschule am Campus Bocholt ist deshalb ein Framework entstanden, mit dessen Hilfe man ein System mit selbst entwickelten Algorithmen zur Analyse von Daten aufsetzen und so eine sehr spezifische Ergebnis-Sortierung beziehungsweise -Selektion erreichen kann. Der Ablauf der Analyse lässt sich folgendermaßen beschreiben: Man bedient sich zunächst einer Datenquelle wie beispielsweise einer Trefferliste einer herkömmlichen Suchmaschine, eines sozialen Netzwerks oder einer Datenbank, die man üblicherweise mittels eines API ansprechen kann. Hiermit haben wir einen Datenpool, der zunächst nicht oder nur grob vorverarbeitet ist. Der Datenpool kann durchaus aber auch aus vernetzten Kameras oder einem Sensor-Netzwerk bestehen – die Art der Daten ist prinzipiell für das Framework gleichgültig und beeinflusst nur die Wahl des letztlich verwendeten Analyse-Algorithmus.

Nun kann ein eigener Algorithmus, bei Bedarf auch mehrere Algorithmen, eine Detail-Analyse dieses Datenpools durchführen. Ein solcher Algorithmus führt dabei ein Scoring durch, um zu kennzeichnen, inwiefern die Seite für einen „guten" oder „schlechten" Treffer gehalten wird. Die so erhaltene Liste kann dann beispielsweise sortiert nach den Scores als HTML-Seite ausgegeben werden.

Je nach Anzahl der Treffer, der Komplexität der eigenen Analyse-Algorithmen und natürlich der genutzten Hardware kann ein solcher Analyse-Auftrag durchaus mehrere Stunden in Anspruch nehmen. Unter anderem spielte dieser Umstand eine Rolle bei der Technologie-Auswahl und beim Design der Architektur.

Die Plattform

Da das System möglichst plattformunabhängig lauffähig sein sollte, war schon vor der Entwicklung des ersten Prototyps klar,

dass Java eingesetzt werden sollte. Die erste Variante wurde somit als Server-Anwendung auf Basis von JSP/Servlet unter Tomcat implementiert. Hier wurden Erfahrungen über das Laufzeitverhalten eines relativ monolithisch implementierten Systems gesammelt. Unter anderem zeigte sich, dass man die Analyse-Algorithmen in der nächsten Implementierung vom Rest des Systems entkoppeln muss, um durch Verteilung dieser Module auf mehrere Server eine akzeptable Performance erreichen zu können. Eine starke Modularisierung soll die Erweiterbarkeit und Skalierbarkeit des verteilten Systems sicherstellen. Die Ausführung der Analyse-Prozesse sollte auf mehrere Server beispielsweise in einer Cloud verteilt werden können. In der zweiten Version wurde das verteilte System auf Basis einer EJB-Architektur implementiert.

Architektur und Ablauf

Um eine lose Kopplung zwischen der Oberfläche und der eigentlichen Anwendungslogik zu erreichen, fiel die Wahl auf eine dreischichtige Architektur (siehe Abbildung 1). Als Technik für das Frontend werden JSP und JavaScript benutzt (Schicht 1). Mittels JavaScript können asynchrone Aufrufe an ein Servlet (Schicht 2) geschickt werden, sodass eine dynamisch wachsende Trefferliste schon während der laufenden Analyse sichtbar ist. Das Servlet arbeitet als Schnittstelle zum Backend und erzeugt aus den vom Nutzer eingegebenen Such-Parametern Auftragsobjekte („Jobs“, siehe Abbildung 2), die vom Backend (Schicht 3) entgegengenommen und verarbeitet werden können.

Ein Job enthält Identifikationsdaten, eine Liste von Datenquellen, deren Typ jeweils gekennzeichnet ist, sowie die Liste von Parsern, die die Datenquellen durchsuchen sollen. Jeder Parser, der in der Liste steht, ist zudem mit den vom Anwender eingegebenen, spezifischen Suchparametern konfiguriert. Anhand einer XML-Konfiguration (siehe Listing 1), die die Konfigurations-Objekte jedes Parsers in einem definierten Schema bereitstellen müssen, wird eine Eingabemaske auf der Web-Oberfläche generiert. Während der Erstellung des Jobs werden dabei die XML-Konfigurationen mit den eingegebenen Werten angepasst und zurück an die jeweiligen Parser gegeben, damit diese sich passend für den Job konfigurieren können. So muss die eigentliche Bedeutung der Parameter des Parsers im Framework nicht bekannt sein. Jeder Parser muss lediglich selbst wissen, was die einzelnen Werte zu bedeuten haben.

Das Backend wird über eine EJB („Job Control“) angesprochen. Sie bietet Methoden, um Jobs zu starten. Die Parser, die die Implementierung von Analyse-Algorithmen darstellen, können verteilt im Netzwerk arbeiten und sind ebenso als EJBs umgesetzt. Sie werden durch das Modul „Job Executor“ gesteuert, das in einem eigenen Thread läuft. Dieses startet wiederum für jeden Parser einen Thread mit einer „Parser Connection“, die die Schnittstelle zu einem optional entfernt ausgeführten Parser darstellt. Dem Framework müssen zuvor nur IP-Adresse und JNDI-Name mitgeteilt werden. Die Verteilung der Parser im Netz ermöglicht den Zugriff auch auf nicht-öffentliche Quellen, sofern dies gewünscht wird. Somit ist das System auch zur Suche in Unternehmens-Datenbanken, im Intranet oder in anderen, nur eingeschränkt erreichbaren Datenquellen einsetzbar.

So können alle Parser gleichzeitig laufen, wobei jeder Parser für sich die Liste der Quellen sequentiell abarbeitet. Hat ein Parser eine Quelle untersucht und eine Liste

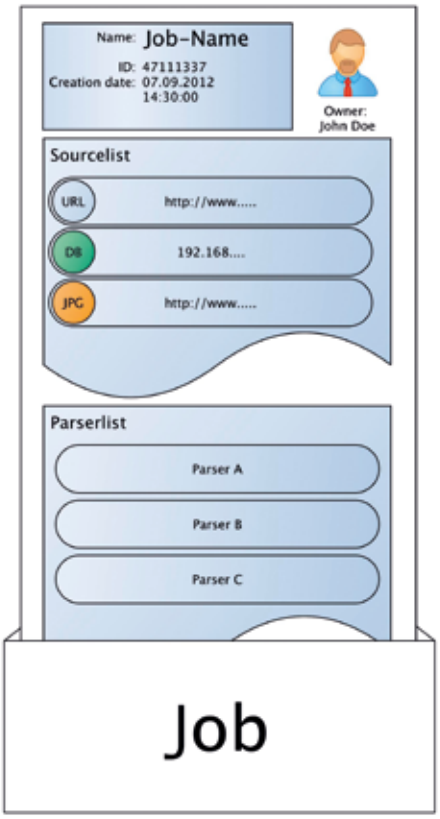


Abbildung 1: Übersicht über die Architektur

```
<parser>
<name>ExampleParser</name>
<config>
<param key="searchString">
<value type="String">Stecknadel</value>
<description>Der Begriff, nach dem der
Parser sucht.</description>
</param>
<param key="searchInteger">
<value type="Integer">42</value>
<description>Die Zahl, nach der der Parser
sucht.</description>
</param>
</config>
</parser>
```

Listing 1: Beispiel zur Konfiguration eines Parsers

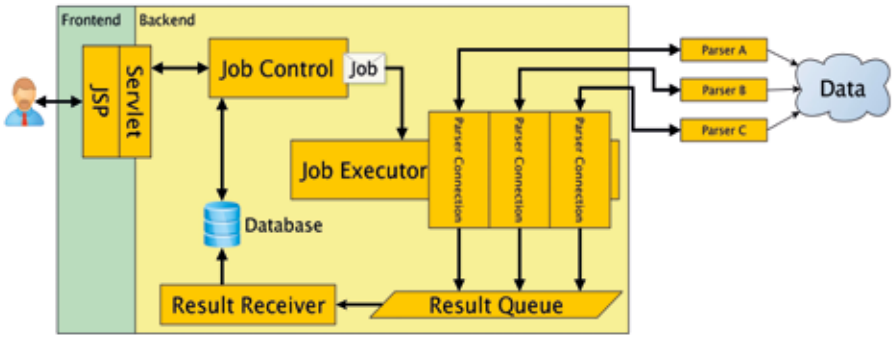


Abbildung 2: Aufbau eines Job-Objekts

von Ergebnissen zurückgegeben, wird diese vom Verbindungs-Objekt zur Weiterverarbeitung in die „Result Queue" gelegt. Die Ergebnisse können dabei positive Rückmeldungen, aber auch Fehlermeldungen oder Warnungen enthalten. Die Message-Driven-Bean „Result Receiver" empfängt und bearbeitet diese Objekte. Rückmeldungen werden mittels Hibernate persistiert und können vom Benutzer als Ergebnis des Jobs abgerufen werden.

Integration eines eigenen Parsers

Zur Erstellung und Verwendung eines eigenen Analyse-Algorithmus muss man lediglich ein eigenes Parser-Modul implementieren und seine Konfiguration vornehmen. Hierzu sind folgende Schritte erforderlich:

1. In Eclipse ein EJB-Projekt erstellen und eine Bibliothek einbinden
2. Stateful Bean für den Parser erzeugen (das Listing hierfür steht unter http://www.doag.org/go/java_aktuell/schulten)
3. Stateful Bean für die Konfiguration erzeugen (einfache Schnittstelle zur XML-Konfiguration)
4. Projekt starten und Parser über die Web-Oberfläche des Systems mit Namen, IP-Adresse, JNDI-Name für Parser und Konfiguration registrieren

Da die Parameter, die ein Algorithmus zur Analyse verwenden, unterschiedlich sein können, müssen jeweils entsprechende Konfigurationsmethoden spezifisch implementiert werden. Die Übergabe der Konfiguration erfolgt dabei in Form von XML-Dokumenten.

Fazit und Ausblick

Der Entwicklungsschritt von einer nur auf JSPs und Servlets basierenden Anwendung hin zu einer verteilten Architektur auf Basis von EJBs hat die gewünschten Effekte in Bezug auf eine Flexibilisierung und mögliche Leistungssteigerung bewirkt. Die einfach gehaltene Plug-in-Schnittstelle für eigene Parser ermöglicht den Einsatz des Analyse-Systems als spannendes Arbeitsfeld für Studierende der Studiengänge „Informatik. Softwaresysteme" und „Verteilte Systeme". Gleichzeitig ist der Einsatz des Systems in künftigen Projekten mit Partnern aus der Wirtschaft angedacht.

Im nächsten Entwicklungsschritt ist die Implementierung eines Job Schedulers geplant, der identische Suchläufe in regelmäßigen Zeitabständen wiederholt.

Prof. Dr.-Ing. Martin Schulten
martin.schulten@w-hs.de

Martin Schulten ist Professor für Informationstechnik an der Westfälischen Hochschule, Campus Bocholt. Er lehrt und forscht dort im Bereich Internetanwendungen und Verteilte Systeme. Schwerpunkte seiner Arbeit bilden Web-Mining- und Analysewerkzeuge sowie Integrative Technologien.

B.Sc. Christoph Seidel
christoph.seidel@w-hs.de

Christoph Seidel bereitet nach dem Bachelor-Studium in Informationstechnik derzeit den Abschluss des Master-Studiums Verteilte Systeme an der Westfälischen Hochschule, Campus Bocholt, vor. Seine Interessenschwerpunkte liegen allgemein im Bereich Systementwurf und der Architektur verteilter Systeme.

Android goes Gradle

Heiko Maaß, Namics Deutschland GmbH

Aktuell stellt Google für das Bauen von Android-Applikationen ein Eclipse-Plug-in (ADT) und eine Sammlung von Ant-Skripten zur Verfügung. Für das erfolgreiche Apache-Maven-Build-System gibt es dagegen keine offizielle Unterstützung. Die Wahrscheinlichkeit dafür ist auch stark gesunken, da Google derzeit ein neues Build-System auf Basis von Gradle implementiert. Dieser Artikel stellt eine frühe Betaversion vor.

Betrachten wir zunächst die Basis des neuen Build-Systems: Gradle wird unter der Apache-2.0-Lizenz veröffentlicht [1] und vom Unternehmen Gradleware vorangetrieben. Gradle erhebt den Anspruch, die Flexibilität beziehungsweise Skriptbarkeit von Apache Ant mit einer deklarativen Build-Konfiguration zu vereinen. Dazu nutzt Gradle die Script-Sprache Groovy, um eine eigene Domain Specific Language (DSL) für die Build-Konfiguration zu ermöglichen.

Der Vorteil einer DSL gegenüber einem XML-basierten Ansatz entsteht durch die verbesserte Lesbarkeit beziehungsweise Ausdrucksfähigkeit sowie durch die nahtlose Integration von projektspezifischen Erweiterungen. Ein Gradle-Build besteht aus einer Reihe sogenannter „Tasks", die einzelne, in sich geschlossene Arbeitsschritte wie das Kompilieren von Klassen oder das Ausführen von Unit-Tests, abbilden.

In Abhängigkeit vom zu bauenden Projekt kann Gradle Repositories sowohl von Apache Maven als auch von Apache Ivy auflösen. Im Gegensatz zu Maven ist ein formales Repository jedoch nicht zwingend: Abhängigkeiten können auch direkt aus dem Dateisystem referenziert werden. Plug-ins erweitern die Funktionalität eines Gradle-Build-Skripts. Genau das macht momentan Google: Die Tools-Entwickler konzentrieren die Funktionalität für das Bauen von Android-Projekten in zwei Gradle-Plug-ins:

- „android" für eine ausführbare Android App
- „android-library" für eine Android-Bibliothek, die in mehrere Apps eingebettet werden kann

Zum Zeitpunkt der Artikelerstellung ist die Version 0.2 aktuell, die zusammen mit dem Android-Tool-Preview „15 rc7" ausprobiert werden kann. Um diese Versionen herunterladen zu können, muss der Preview-Channel im Android SDK Manager aktiviert sein [2].

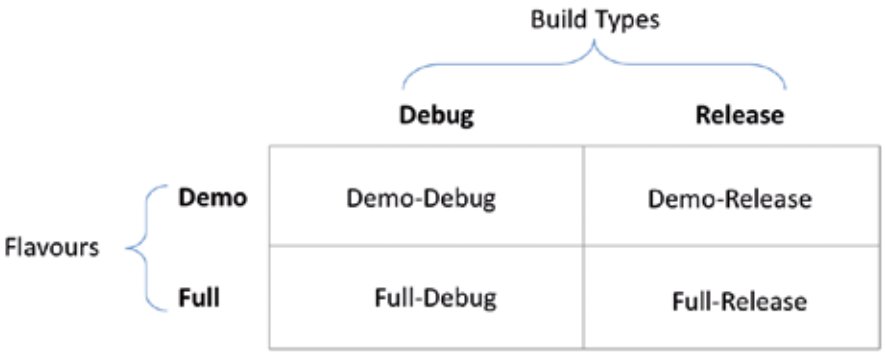


Abbildung 1: Product Flavours und Build Types

Unterschiedliche App-Ausprägungen durch Product Flavours und Build Types

Eine der wichtigsten neuen Möglichkeiten des neuen Build-Systems ist das Konzept der „Product Flavours". Durch sie können mehrere Ausprägungen einer Applikation erzeugt werden, die sich inhaltlich unterscheiden. Ein häufiger Anwendungsfall dafür ist die Aufspaltung einer Applikation in eine Demo- und eine Vollversion.

Für alle Product Flavours möchte man typischerweise im Rahmen des Entwicklungsprozesses sowohl Debug- als auch Release Builds bauen. Diese zusätzliche, orthogonale Dimension nennt Google „Build Types". Damit können für das Testen relevante Unterschiede (wie erweitertes Logging oder Debug-Ansichten) strukturiert in das Projekt eingeflochten werden. Abbildung 1 zeigt eine einfache Build-Matrix, bestehend aus jeweils zwei Build Types und Product Flavours.

Das Android-Plug-in baut immer eine Kombination aus Build Type und Product Flavour, beispielsweise „Demo-Release". Damit das neue Konzept funktioniert, hat Google die Verzeichnisstruktur eines Android-Projekts umgestellt: Gemeinsam verwendete Dateien werden im „src/main"-Verzeichnis abgelegt. Für jeden Product Flavour und jeden Build Type gibt es parallel zum „main"-Verzeichnis ein eigenes Verzeichnis für die Ablage der ausprägungsspezifischen Klassen und Ressourcen (beispielsweise „src/main/demo" und „src/main/release"). Beim Bau der jeweiligen Ausprägung (wie „Demo-Release") werden die ausprägungsspezifischen Dateien mit den gemeinsam verwendeten Dateien vermischt. Dabei kommen – abhängig vom Dateityp – unterschiedliche Strategien vor:

- „Klassen“ aus dem Product Flavour beziehungsweise aus dem Build Type werden zusammen mit den Klassen aus „src/main“ in einem Classpath vereint. Das Überlagern von Klassen ist nicht vorgesehen; sollte ein Product Flavour oder ein Build Type einen gleichen qualifizierten Klassennamen wie in „src/main“ verwenden, wird der Compiler den Build-Prozess mit der Fehlermeldung „duplicate class“ abbrechen.
- Ressourcen werden ebenfalls vereint, bei Konflikten überschreibt die spezifische Version jedoch die Version aus „src/main“. Sollte es Konflikte zwischen Build Type und Product Flavour geben, gewinnt die Datei aus dem Build Type.
- „AndroidManifest.xml“ wird zusammengeführt („gemergt“), dabei werden gegebenenfalls bereits definierte Attribute in der „src/main“-Version überschrieben.

Abbildung 2 zeigt eine beispielhafte Verzeichnis-Hierarchie mit jeweils zwei Build Types und Product Flavours.

Einbinden des Android-Plug-ins

Betrachten wir nun, wie der Build konfiguriert wird. Gemäß Konvention erwartet Gradle das Build-Skript in einer Datei namens „build.gradle“. In dieser Datei wird das Projekt mithilfe der Groovy DSL beschrieben. Damit die Android-spezifischen Tasks aufgerufen werden können, muss das Android-Plug-in zunächst als Classpath-Abhängigkeit bekannt gemacht werden. Gradle sieht dafür den sogenannten „buildscript-Block“ vor (siehe Listing 1).

```
buildscript {
  repositories {
    mavenCentral()
  }
  dependencies {
    classpath 'com.android.tools.
build:gradle:0.2'
  }
}
apply plugin: 'android'
```

Listing1: Registrierung des Android-Plug-ins

Sowohl das Android- als auch das Android-Library-Plug-in befinden sich im gleichen Artefakt, das wir direkt vom Maven Central Repository beziehen. Die Maven-Koordinaten („groupId“, „artifactId“ und „version“) des Artefakts geben wir (jeweils mit Doppelpunkten getrennt) innerhalb des „dependencies“-Blocks an. Mit dem Aufruf des „apply“-Befehls wird das Android-Plug-in für das Build-Skript schlussendlich registriert.

Grundlegende Projekteinstellungen

Dank der Registrierung können wir nun den Android-Block nutzen, in dem sämtliche Android-spezifischen Projekt-Einstellungen vorgenommen werden. Im Rahmen der Umstellung auf Gradle hat Google aufgeräumt und vormals verstreute Konfigurationseinstellungen im Android-Block zusammengeführt. Beispielsweise mussten die minimal zu unterstützende SDK-Version („minSdkVersion“) und die Ziel-SDK-Version („targetSdkVersion“) vorher in der Datei „AndroidManifest.xml“ festgelegt werden. Auch die „project.properties“-Datei ist nun hinfällig: Das „target“, also die Konfiguration der Android-Library für den Kompilier-Vorgang, wird nun ebenfalls im Android-Block definiert (siehe Listing 2).

```
// ... (siehe Listing 1)
apply plugin: 'android'
  android {
    target = "Google Inc.:Google APIs:15"
    defaultConfig {
      versionCode = 103
      versionName = "1.0.3-SNAPSHOT"
      minSdkVersion = 9
      targetSdkVersion = 15
    }
  }
```

Listing 2: Grundlegende Projekt-Einstellungen

Konfiguration der Product Flavours und Build Types

Product Flavours und Build Types werden in gleichnamigen Blöcken innerhalb des Android-Blocks beschrieben. Beispielsweise können wir hier unterschiedliche Android-Paketnamen und Paket-Suffixe vergeben, sodass alle Ausprägungen der Applikation auf einem Gerät parallel installiert werden können. Der Paketname dient als „Identifier“ einer Applikation und muss eindeutig sein.

Auf die aus dem bisherigen Android-Build-System bekannte, automatisch generierte „BuildConfig“-Klasse können wir nun ebenfalls stärkeren Einfluss nehmen:

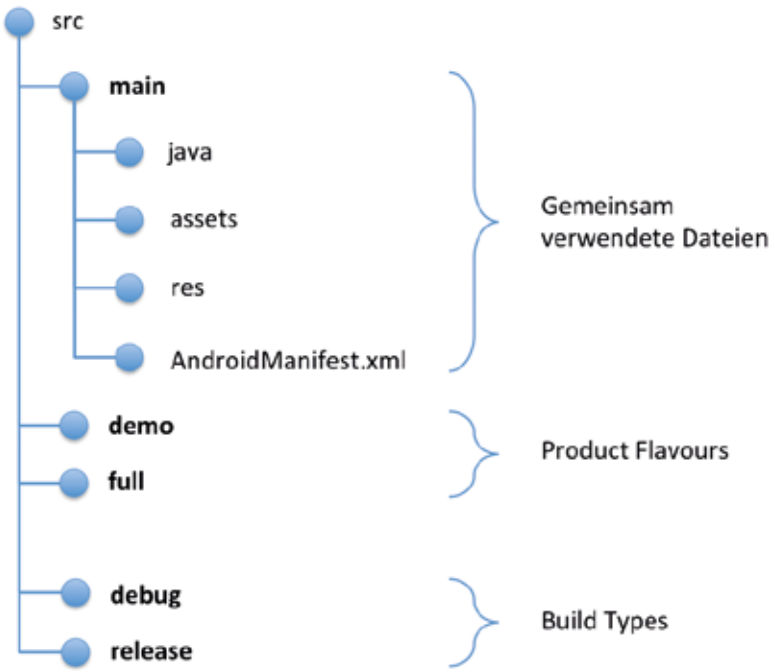


Abbildung 2: Die neue Verzeichnisstruktur

Es ist nun möglich, zusätzliche Code-Schnipsel an die „BuildConfig"-Klasse zu heften, um beispielsweise ausprägungsabhängige, statische Variable zu definieren.

In Listing 3 definieren wir die Product Flavours „demo" und „full". Die statische Variable „MAX_ITEMS" in der „BuildConfig"-Klasse soll sich bei beiden Ausprägungen unterscheiden. Zusätzlich soll bei Debug-Ausprägungen das Suffix „.debug" an den Paketnamen angehängt werden, sodass sie nicht mit den Release-Ausprägungen kollidieren.

Anhand dieser Konfiguration erzeugt das Android-Plug-in automatisch für jede Ausprägung die jeweiligen Installations-Tasks. Der Befehl „gradle tasks" gibt alle Tasks auf der Konsole aus (siehe Abbildung 3). Mit „gradle installFullRelease" wird beispielsweise die „Full-Release"-Version auf einem angeschlossenen Android-Gerät installiert.

```
Install tasks
---------------
installDemoDebug - Installs the Debug build for flavor Demo
installDemoRelease - Installs the Release build for flavor Demo
installDemoTest - Installs the Test build for the DemoDebug build
installFullDebug - Installs the Debug build for flavor Full
installFullRelease - Installs the Release build for flavor Full
installFullTest - Installs the Test build for the FullDebug build
uninstallAll - Uninstall all applications.
uninstallDemoDebug - Uninstalls the Debug build for flavor Demo
uninstallDemoRelease - Uninstalls the Release build for flavor Demo
uninstallDemoTest - Uninstalls the Test build for the DemoDebug build
uninstallFullDebug - Uninstalls the Debug build for flavor Full
uninstallFullRelease - Uninstalls the Release build for flavor Full
uninstallFullTest - Uninstalls the Test build for the FullDebug build
```

Abbildung 3: Automatisch generierte Gradle-Tasks

```
productFlavors {
  demo {
    packageName = "com.namics.tests.flavor.demo"
    buildConfig "public final static int MAX_ITEMS = 10;"
  }
  full {
    packageName = "com.namics.tests.flavor.full"
    buildConfig "public final static int MAX_ITEMS = 999;"
  }
}

buildTypes {
  debug {
    packageNameSuffix = ".debug"
  }
}
```

Listing 3: Konfiguration der beiden Product Flavours

Blick unter die Haube

Der Quelltext des Artefakts „com.android.tools.build:gradle:0.2" [3] zeigt, dass sowohl das Android- als auch das Android-Library-Plug-in nicht von Gradles mitgeliefertem Java-Plug-in ableiten, sondern direkt das generische „org.gradle.api.Plugin"-Interface implementieren. Sämtliche Gradle-unabhängigen Kompilier- und Signierprozesse hat Google in das Artefakt „com.android.tools.build:builder:0.2" ausgelagert.

Diese Abstraktion sorgt jedoch für einen kleinen Einschnitt in der Flexibilität: Der Austausch von Android-Klassen im Classpath, der beispielsweise für Unit-Tests-Frameworks wie Robolectric [4] notwendig ist, ist nach Wissen des Autors nicht per Konfiguration möglich.

Fazit und Ausblick

Das neue Build-System bringt Erleichterung in den Entwickler-Alltag. Das Bauen unterschiedlicher Ausprägungen einer App wird dank der Product Flavours und Build Types stark vereinfacht und strukturiert. Mit der Entscheidung von Google für Gradle als Basis des neuen Build-Systems dürfte Gradle zusätzliche Aufmerksamkeit erhalten und gleichzeitig an Bedeutung in der Java-Welt gewinnen. Noch fehlen allerdings einige wichtige Features: Beispielsweise ist noch keine IDE-Unterstützung implementiert. Zudem gibt es noch keine Aussage darüber, bis wann Googles neues Build-System fertig ist.

Quellen

[1] https://github.com/gradle/gradle
[2] http://tools.android.com/preview-channel
[3] http://search.maven.org/remotecontent?filepath=com/android/tools/build/gradle/0.2/gradle-0.2-sources.jar
[4] http://pivotal.github.com/robolectric/

Heiko Maaß
heiko.maass@namics.com

Heiko Maaß hat Wirtschaftsinformatik an der HTWG in Konstanz studiert und arbeitet als Senior Software Engineer bei der Namics Deutschland GmbH mit Fokus auf mobilen Applikationen, die einen echten Mehrwert schaffen.

Web-Apps mit „Play!“ entwickeln – nichts leichter als das!

Andreas Koop, enpit consulting OHG

Das Open-Source-Play-Framework stellt eine ernstzunehmende Konkurrenz für etablierte Java-Web-Frameworks dar. Nicht zuletzt die Übernahme von Play 2 in den Typesafe-Stack der Scala-Macher bringt frischen Wind in den Dunstkreis von Struts, JSF, Tapestry, Wicket und Grails.

Das Play-Framework verzichtet auf Servlets und bricht auch sonst mit weitverbreiteten Konventionen im Java-EE-Umfeld, um die Produktivität und Leichtgewichtigkeit in den Vordergrund zu rücken. Die Architektur basiert auf dem MVC-Pattern [1] und orientiert sich stark an Ruby on Rails und Django. Für nahezu alle essenziellen Bereiche einer Web-Anwendung bietet das Framework eine von Haus aus integrierte Lösung, die Konfigurationen, OR-Mapping, Library-Dependency-Management, Datenbank-Evolutionsskripte, Unit-, Integrations- und UI-Tests, Security, Caching, Validierung und sogar Scheduling umfasst.

Als Ablauf-Umgebung fungiert ein eingebetteter JBoss-Netty-Server. Darüber hinaus können Play-Anwendungen als Web Application Archive (WAR) paketiert und auf einem Java EE Web-Container installiert werden. Ein besonderes Alleinstellungsmerkmal von Play! ist, dass Anwendungen sowohl mit Java als auch mit Scala entwickelt werden. Hier ist man also ebenfalls voll im Trend der polyglotten Programmierung.

Architektur und Technologien

Play 2 bezeichnet sich selbst als „Web framework for a new era“ und setzt damit die Messlatte sehr hoch an. Grund für diese Annahme ist das eigens entwickelte und auf ein asynchrones Request-Verarbeitungsmodell ausgerichtete Streaming-API. Ein Request blockiert keinen Thread, sondern gibt diesen frei, falls während der Request-Verarbeitung auf I/O oder etwa auf einen Web-Service-Call gewartet wird. Damit ist Play! für hochperformante und -skalierbare Anwendungen geradezu prädestiniert. Abbildung 1 veranschaulicht die wesentlichen Architekturmerkmale.

Alle eingehenden HTTP-Anfragen landen im Router und werden von da aus an die entsprechende Controller-Action weitergeleitet. In der jeweiligen Controller-Action werden anhand der übergebenen Parameter notwendige Modell-Objekte geladen und/oder Business-Operationen ausgeführt. Anschließend erfolgt der Aufbau der gewünschten (HTML)-View, die nach erfolgreicher Ausführung als Antwort zurück an den Client gesendet wird.

Der Technologie-Stack von Play 2 baut auf der Programmiersprache Scala und dem Event-getriebenen Middleware-Framework Akka [2] auf – einem Nachrichten-orientierten Programmiermodell für nebenläufige Software (Stichwort: Actor-Model-Konzept). Als Build- und Dependency-Management-Werkzeug fungiert Simple Build Tool (sbt, [9]), das fester Bestandteil in Scala-Projekten ist und in Play 2 eine interaktive Konsole zur Verfügung stellt, um Play-Anwendungen zu erstellen, zu starten, zu testen sowie Projektdateien für Eclipse, IDEA oder NetBeans zu erstellen.

Wie in der Einleitung erwähnt, basiert die Request-Verarbeitung nicht auf Servlets, sondern auf einem Streaming-API (siehe Iteratee I/O, [3]) – zumindest in der Java-Welt ein recht neues Konzept. Es stellt die Grundlage für asynchrone, zustandslose Kommunikation dar und ist die Basis für die in Play implementierten Features Comet, WebSockets und Server-Sent-Events. Abbildung 2 zeigt die essenziellen Komponenten von Play! und deren Zusammenspiel im Überblick.

Play-Anwendung erstellen und ausführen

Eine Besonderheit von Play ist, dass Anwendungen sowohl in Java als auch in

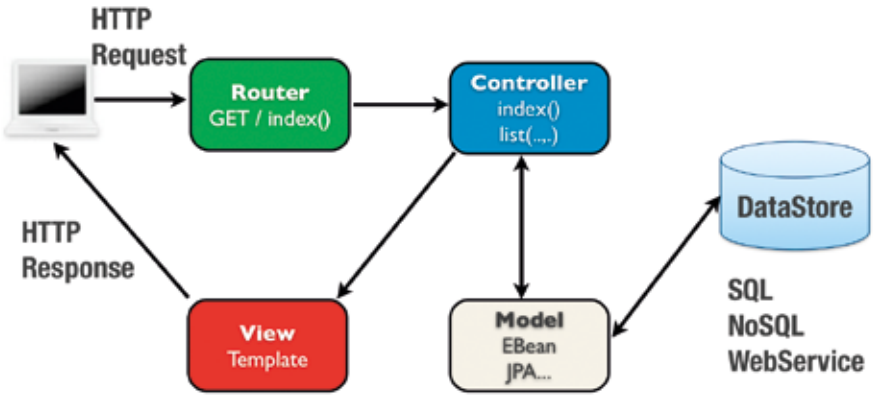


Abbildung 1: MVC-Architektur von Play 2

```
$ play new myApp
What is the application name?
> myApp

Which template do you want to use for this new application?

  1 - Create a simple Scala application
  2 - Create a simple Java application
  3 - Create an empty project

> 2

OK, application myApp is created.

Have fun!
```

Listing 1: Eine neue Play-Anwendung erstellen

```
db.default.driver=org.h2.Driver
db.default.url="jdbc:h2:mem:play"

#You can expose this datasource via JNDI if needed (Useful for JPA)
db.default.jndiName=DefaultDS
jpa.default=defaultPersistenceUnit
```

Listing 2: JPA in „conf/application.conf" aktivieren

```
val appDependencies = Seq(
    // Add your project dependencies here,
    "org.hibernate" % "hibernate-entitymanager" % "3.6.9.Final"
)
```

Listing 3: Auszug „Dependencies" in „project/Build.scala"

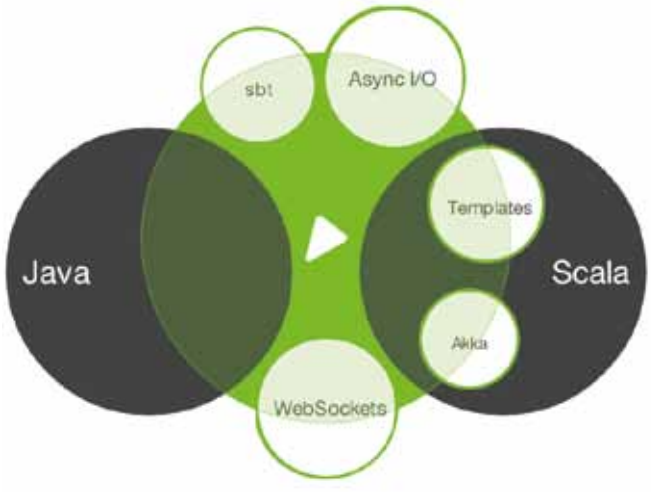


Abbildung 2: Technologie-Auszug aus Play 2 (Quelle: www.playframework.org)

Scala programmiert werden können. Dies zeigt sich unmittelbar bei Erstellung einer neuen Play-Anwendung, was im Übrigen durch Ausführung eines einzigen Befehls auf der Kommandozeile erledigt wird (siehe Listing 1).

Bei der Erstellung einer neuen Anwendung werden per Konvention alle notwendigen Dateien und Verzeichnisse erstellt, sodass im Anschluss die Anwendung per „play run" unmittelbar über den Built-in-Server auf dem Default-Port 9000 unter http://localhost:9000 erreichbar ist. Abbildung 3 zeigt die Struktur einer Play-Anwendung.

Der Entwickler findet sich sehr schnell zurecht. Man muss sich keine Gedanken über die Ablage-Struktur für Modell- und Controller-Klassen, Views, Konfigurationsdateien sowie Public-Ressourcen wie Bild-, CSS- und JavaScript-Dateien machen. Über die letzten Jahre hat sich eine geordnete Struktur etabliert. Sogar die Logdatei und die Build-Ausgabe haben im Entwicklungsmodus einen wohldefinierten Platz.

Model, O/R-Mapping mit JPA und Validierung

Bevor eine Anwendung gebaut wird, ist in der Regel ein entsprechendes Fachklassen-Modell erforderlich, das ein Abbild der jeweiligen Fach-Domäne enthält. Entitäten und deren Beziehungen werden über ein geeignetes O/R-Mapping in der Datenbank gespeichert. Als Persistenz-Framework setzt Play standardmäßig auf Ebean [4] bei Java- und Anorm [5] bei Scala-Anwendungen. Da dies eher Exoten unter den O/R-Frameworks sind, befindet sich in Play erfreulicherweise auch Unterstützung für JPA, das einfach per Konfiguration aktiviert wird (siehe Listing 2). Als JPA-Provider kommt Hibernate zum Einsatz. Es muss nur die notwendige Dependency in „project/Build.scala" definiert sein (siehe Listing 3).

Die eigentlichen Entitäten sind einfache Java-Klassen (POJOs) mit den erforderlichen Annotationen für das Mapping auf die entsprechende Datenbank-Struktur – nichts Neues also. Einzig der Zugriff auf den EntityManager erfolgt über ein von Play zur Verfügung gestelltes API. Mit „play.db.jpa.JPA.em()" kann auf den aktuellen Entity-Manager zugegriffen werden. Eine Besonderheit ist, dass sämtliche Attribute als „public" deklariert sind – leichtgewichtig und einfach gemäß Play!-Ideologie.

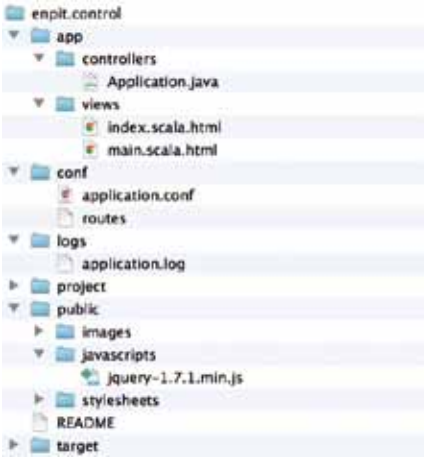


Abbildung 3: Struktur und Basis-Dateien einer Play-Anwendung

Was die Validierung angeht, setzt Play auf die standardisierte Bean-Validation (JSR-303), bietet allerdings ähnlich zu JPA gleich etwas mehr Komfort durch eigene Annotationen. (siehe Listing 4).

Transaktions-Handling

Die an JPA-Entitäten definierten Klassen-Methoden werden gemäß Play-Best-Practice in den Controller-Action-Methoden aufgerufen. Um diese Business-Operationen Transaktions-sicher auszuführen, existiert eine entsprechende Annotation „@play.db.jpa.Transactional", die an die gewünschte Java-Methode gesetzt wird (siehe Listing 5). Das war es auch schon – wiederum sehr überschaubar und einfach.

Typsichere Routen und Controller-Actions

Wie im Architektur-Abschnitt dargestellt (siehe Abbildung 1) landet ein eingehender Request im Router. Aus Entwicklersicht ist dies eine Konfigurationsdatei, die eingehende HTTP-Anfragen an die gewünschte Controller-Action weiterleitet. Obwohl die Routen-Konfigurationsdatei wie eine einfache Textdatei aussieht, handelt es sich um eine interne Scala-DSL [6]. Aufgrund dessen werden Fehler zur Übersetzungszeit festgestellt und nicht erst zur Ausführungszeit. Listing 6 zeigt beispielhaft den Aufbau einer „conf/routes"-Datei.

Die Routing-Datei lässt sich sehr einfach lesen. HTTP-Methoden (GET, POST, PUT, DELETE und HEAD) in Kombination mit einer URI, die auch Platzhalter enthalten kann (wie „/contacts/:id"), werden auf eine Con-

```
@Entity
@SequenceGenerator(name = "contact_seq", sequenceName = "contact_seq")
public class Contact {
   @Id
   @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "con-
tact_seq")
   public Long id;
..
   @Constraints.Required
   public String lastname;

   @Constraints.Email
   public String email;
   @Formats.DateTime(pattern="dd.MM.yyyy")
   public Date birthday;
   @ManyToOne(cascade = CascadeType.MERGE)
   public Company company;
   public static Contact findById(Long id) {
      return JPA.em().find(Contact.class, id);
   }
..
}
```

Listing 4: JPA-Entität mit Validierung und Persistenz-Methoden

```
@Transactional
 public static Result createContact() {
   ...
 }
 //Bei ausschließlich lesendem Datenbank-Zugriff
 @Transactional(readOnly=true)
 public static Result listContacts() {
   List<Contact> contacts = Contact.findAll();
   ...
 }
```

Listing 5: Annotations-basierte Transaktions-Steuerung

```
# Home page
GET  /              controllers.Application.index()
# Create contact
POST /contacts/new   controllers.Contacts.create()
# Get contact
GET /contacts/:id    controllers.Contacts.show(id: Long)
# List contacts
GET  /contacts       controllers.Contacts.contacts(f ?= "", o ?= "asc", s ?= "lastname")
```

Listing 6: Typsicheres Routing per interner Scala-DSL

```
public static Result show(Long id) {
   Contact contact = Contact.findById(id);

   return ok( views.html.show.render( contact ) );
}
```

Listing 7: Controller-Action zur Anzeige eines Kontakts

```
1: @(contact: Contact)
2: @main(„enpit.office - Contact“) {
3:   <h2>@contact.firstname @contact.lastname</h2>
4:   Email:   @contact.email <br />
5:   Birthday: @contact.birthday.format(„dd.MM.yyyy“) <br />
6:   ...
7: }
```

Listing 8: View-Template als Scala-Funktion

```
@(title: String)(content: Html)
<!DOCTYPE html>
<html>
  <head><title>@title</title>
    ...
  </head>
<body>
  ...
  @content
</body></html>
```

Listing 9: Ein Main-View-Template definiert das Seiten-Layout

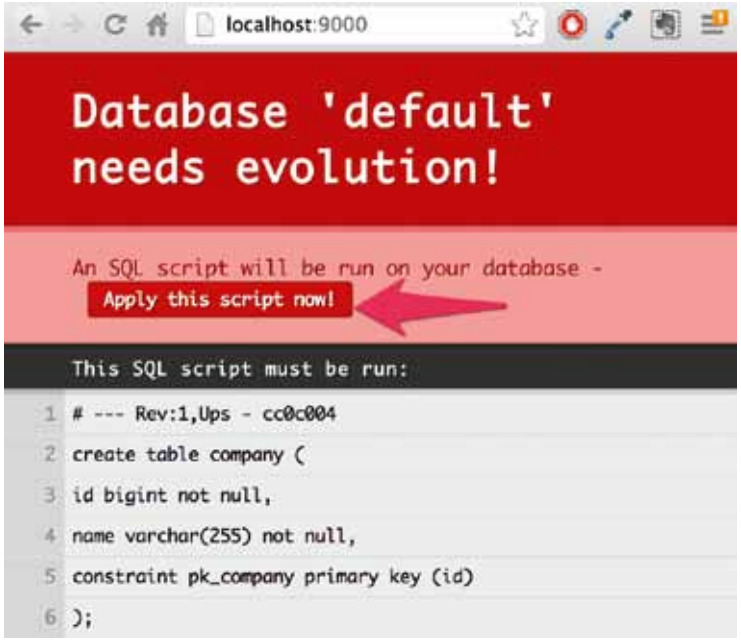


Abbildung 4: SQL-Skripte aus dem Browser heraus ausführen

troller-Action abgebildet. Falls die Methode nicht existiert, meldet Play bereits zur Übersetzungszeit einen Fehler. Optionale URL-Parameter können darüber hinaus modelliert und mit Default-Werten belegt werden.

Wie im Abschnitt zur Transaktions-Steuerung angedeutet, sind Controller-Actions Klassen-Methoden mit dem Rückgabetyp „play.mvc.Result“. Ein Result beinhaltet den HTTP-Statuscode, -Header-Informationen sowie den Body-Inhalt. Neben HTML kann beliebiger Content an den Client gesendet werden, beispielsweise XML oder JSON. Dazu ein Beispiel (siehe Listing 7).

Zunächst wird anhand der übergebenen Parameter die gewünschte Business-Operation ausgeführt und anschließend das Result-Objekt mithilfe einer von Play! bereitgestellten Methode zusammengebaut. Der Result-Body besteht aus dem gerenderten Ergebnis aus der Ausführung des „show-View“-Templates. Neben „ok(..)“, was einem „HTTP 200“ entspricht, gibt es folgende Hilfsmethoden, um das gewünschte Ergebnis an den Client zu senden: „notFound()“, „badRequest(..)“ und „internalServerError(..)“ sowie einen beliebigen Statuscode mittels „status(488, “Spezial Antwort“)“.

Typsichere View-Templates

Der aufmerksame Leser wird es sicherlich schon bemerkt haben: Die in Play 2 verwendeten View-Templates sind nichts anderes als (Scala-)Funktionen, die direkt in einer Controller-Action aufgerufen werden. Im Beispiel (siehe Listing 7) ist „views.html.show.render (contact)“ diejenige Stelle, die das View-Template ausführt. Per Konvention wird hier das Template evaluiert, das unter „/views/html/show.scala.html“ liegt. Sollte der Ablageort nicht der Konvention entsprechen, erfolgt auch hier zur Übersetzungszeit eine Fehlermeldung.

Da View-Templates schlichtweg Funktionen sind, beginnen sie mit der Liste formaler Parameter. Die Funktion selbst ist dafür verantwortlich, mithilfe der übergebenen Parameter den gewünschten HTML-Code zu generieren. Durch die Umsetzung der Templates in Scala können beliebige Scala-Funktionen und -Bibliotheken eingebunden sein. Umfangreiche Views lassen sich in wiederverwendbare, kleinere Kompositions-Bausteine zerlegen. Scala-Anweisungen werden mit einem „@“ eingeleitet. Listing 8 zeigt ein einfaches Play-View-Template.

Zeile 2 demonstriert eine Komposition von View-Templates. Damit nicht jede View den HTML-Header und das Seiten-Layout definieren muss, wird ein „main“-Template zwecks Wiederverwendung eingebunden. Es erwartet einen Parameter – hier: Seitentitel – sowie einen HTML-Body. Listing 9 zeigt den Aufbau des Main-Templates.

Datenbank-Evolution

Bevor eine Play-Anwendung aufgerufen und getestet werden kann, sind in der Regel ein entsprechendes Datenbank-Modell sowie Testdaten erforderlich. Play!

Abbildung 5: Compiler-Fehler werden im Browser angezeigt

enthält standardmäßig die schlanke H2-InMemory-Datenbank [7] und erlaubt es somit, in sehr kurzer Zeit eine Anwendung lokal testen zu können. Selbstverständlich werden auf Wunsch jedoch alle per JDBC ansprechbaren Datenbanken unterstützt. Die Verwendung von H2 als Default ermöglicht jedoch den Einstieg in Play! ungemein.

Die Datenbank wird dabei bei jedem Neustart der Anwendung von Grund auf neu erstellt und mit Testdaten befüllt. Um diesen Vorgang möglichst Entwickler-freundlich zu gestalten, werden im Entwicklungs-Modus (Play! unterstützt mehrere „Projekt-Stages“) alle in „conf/evolutions/default“ abgelegten SQL-Skripte ausgeführt. Die Reihenfolge wird dabei direkt aus dem Dateinamen abgeleitet, also „1.sql“, „2.sql“ etc. Die SQL-Skripte selbst erlauben es, durch die semantischen Kommentare „# --- !Ups“ und „# --- !Downs“ einzelne SQL-Anweisungen beim Hoch- beziehungsweise Herunterfahren der Play-Anwendung zu verorten.

Für die Ausführung der SQL-Skripte ist kein dediziertes Kommando notwendig, es sind nur SQL-Skripte zu speichern und die Anwendung über den Browser aufzurufen. Falls das Datenbank-Schema nicht existiert, wird dem Entwickler die Ausführung der DB-Evolutionsskripte angeboten (siehe Abbildung 4).

Per Klick auf „Apply this script now!“ werden sämtliche (über alle SQL-Skripte hinweg) unter „Ups!“ gestellten SQL-Anweisungen ausgeführt. Anschließend wird die ursprünglich angeforderte URL erneut geladen. Im Normalfall wird die gewünschte Seite mit den Testdaten angezeigt.

Produktivität mit Freuden dank „Fix & Reload“

Es wurde in diesem Artikel bereits viel über die Typ-Sicherheit gesprochen. Die Art der Umsetzung in Play! ist einzigartig gelöst, indem jegliche Compiler-Fehler direkt im Browser angezeigt sind. Während der Entwicklung ist es also kaum notwendig, auf die Server-seitigen Logdateien zu schauen.

Um die Produktivität noch weiter zu erhöhen, kommt das sogenannte „Triggered-Compilation“-Konzept zur Anwendung, das durch „play ~compile“ oder „play ~run“ eingeleitet wird. Es sorgt dafür, dass Router, sämtliche Java- oder Scala-Klassen sowie View-Templates automatisch beim Speichern kompiliert werden. Der Vorteil

Abbildung 6: Beispielseite auf Basis von Twitter-Bootstrap

besteht darin, dass im Erfolgsfall nichts mehr kompiliert werden muss, sobald das nächste Mal über den Browser auf die Anwendung zugegriffen wird. Der Entwicklungszyklus und damit die Produktivität wirken noch effektiver, als es bei Play ohnehin schon ist. Abbildung 5 zeigt, wie Compiler-Fehler im Browser dargestellt sind. Nach Behebung des Fehlers und Aktualisierung der Seite im Browser ist das Ergebnis zu sehen, in diesem Beispiel eine auf Basis von Twitter-Bootstrap [8] erstellte Kontakt-Übersichtsseite (Abbildung 6).

Durch die SBT-basierte [9] Play-Console ist die Entwicklung mit einem simplen Texteditor möglich. Den besten Komfort bietet derzeit allerdings die integrierte Entwicklungsumgebung IDEA 12 [10] von JetBrains Inc. Code-Assistenten in View-Templates, in der Routing-Konfiguration sowie für die Integration der Play-Console in die IDE erhöhen nochmals enorm die Produktivität.

Fazit und Ausblick

Die Architektur von Play! ist sehr durchdacht und für zustandslose Echtzeit- und hochskalierbare Web-Anwendungen ausgelegt. In großen Teilen sind die verwendeten Konzepte mit denen von Request-basierten Web-Frameworks wie Grails vergleichbar.

Das Play-Framework erfährt unter anderem aufgrund des wirklich umfangreichen und kompletten Stacks einen deutlich höheren Hype, nicht zuletzt durch die Unterstützung der Scala-Macher von Typesafe Inc.

Wer nah an CSS3, JavaScript oder HTML5 bleiben und ganz nebenbei noch neue Programmiersprachen (Scala) erlernen möchte, der ist beim Play-Framework genau richtig. Die exzellenten Möglichkeiten, neueste Techniken auf verblüffend einfache und geniale Art und Weise einzusetzen, rufen nach Jahren der Java-EE-Entwicklung längst vergessene Programmierfreuden hervor. Da das Play-Framework wirklich umfangreich ist, gibt es in der nächsten Ausgabe einen zweiten Teil, der folgende Punkte beleuchtet:

- Session-Handling
- Security/Social-Sign-on
- Realtime-Streaming mit WebSockets
- Internationalisierung
- WebJars, CoffeeScript und LESS
- Deployment auf der Cloud-Plattform „Heroku“

Weitere Informationen

- Homepage: http://www.playframework.org
- Play-Framework Community:
  - GitHub: https://github.com/playframework/play
  - Google Group: https://groups.google.com/forum/?hl=de&fromgroups#!forum/play-framework
  - Google Plus Community: https://plus.google.com/106233335814246022600
  - Stackoverflow: http://stackoverflow.com/tags/playframework
- DOAG 2012 Vortrag zum Thema „Web-Apps mit Play! entwickeln“: http://de.slideshare.net/enpit/wepapps-mit-play-nichts-leichter-als-das-15297423

Quellen

[1] Model View Controller: http://de.wikipedia.org/wiki/Model_View_Controller
[2] Akka: http://akka.io
[3] Iteratee I/O: http://www.haskell.org/haskellwiki/Iteratee_I/O
[4] EBean, Java O/R-Mapping: http://www.avaje.org/
[5] Anorm, Scala O/R-Mapping: https://github.com/playframework/Play20/wiki/ScalaAnorm
[6] Domain Specific Language (DSL): http://www.scala-lang.org/node/1403
[7] H2 Java Database: http://www.h2database.com
[8] Twitter-Bootstrap: http://twitter.github.com/bootstrap
[9] SBT - Simple Build Tool: http://www.scala-sbt.org
[10] Play-Support in Jetbrains IDEA 12: http://www.jetbrains.com/idea/features/play_framework.html

Andreas Koop
andreas.koop@enpit.de

Dipl.-Inf. Andreas Koop ist Geschäftsführer des Beratungsunternehmens enpit consulting OHG. Sein beruflicher Schwerpunkt liegt in der Beratung, dem Training und der Architektur für Oracle-ADF- und WebLogic-Server-Projekte. Daneben beschäftigt er sich mit der Entwicklung von mobilen Apps auf Basis von HTML5 sowie Oracle ADF Mobile und evaluiert aktuelle Web-Technologien und -Frameworks.

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; Robert Szilinski, esentri consulting GmbH

Chefin von Dienst (CvD):
Carmen Al-Youssef,
office@ijug.eu

Titel, Gestaltung und Satz:
Claudia Wagner
DOAG Dienstleistungen GmbH
Foto Titelseite: Fotolia ©herzform

Anzeigen:
CrossMarkeTeam, Ralf Rutkat,
Doris Budwill
redaktion@ijug.eu

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

Linked-Data-Praxis: Daten bereitstellen und verwerten

Angelo Veltens, http://datenwissen.de

Ein Artikel in der letzten Ausgabe hat in die Grundlagen von Linked Data eingeführt und gezeigt, wie daraus ein Web aus Daten entstehen kann. Linked Data ist jedoch keine bloße Theorie: Mit wenigen Zeilen Code kann eine Anwendung Teil dieses Daten-Webs werden. Der Beitrag zeigt, wie man Linked Data veröffentlichen und in seinen Anwendungen verwerten kann.

Wer zum ersten Mal von Linked Data hört, ist nicht selten von der Idee fasziniert, hält sie aber zugleich für utopisch und noch nicht Praxis-relevant. Dabei ist die Veröffentlichung und Verwertung von Linked Data nicht schwieriger als die von JSON- oder XML-Dokumenten. Durch die zusätzliche Semantik und weltweite Verlinkbarkeit der Daten wird die Verwertung von Informationen aus einer Vielzahl von heterogenen Systemen sogar einfacher.

Um dies zu verdeutlichen, werden wir in diesem Artikel auf die Hotel-Bewertungs-Plattform zurückkommen, die in der letzten Ausgabe als Beispiel diente. Wir werden exemplarisch einen Hotel-Service entwickeln, der Informationen über Hotels als Linked Data publiziert, sowie eine Hotel-Rating-Seite, die diese Informationen mit Bewertungen der Nutzer verknüpft. Wir verwenden dazu das Web-Framework „Grails“ [1] zusammen mit der RDF-Bibliothek „groovyrdf“ [2], die auf Apache Jena [3] aufsetzt.

Das Problem

Werfen wir zunächst nochmals einen Blick auf einen typischen JSON-Webservice. Ein Service, der Informationen über Hotels als JSON bereitstellt, könnte folgenden Datensatz ausliefern (siehe Listing 1).

```
{
  id: 42,
  name: 'Hotel Fiktiva'
  city: 'Berlin'
}
```

Listing 1

Es mag relativ leicht sein, diesen Service anzubinden, dabei koppelt man jedoch seine Anwendung mit eben diesem Service. Die Anwendung muss wissen, über welche URIs sie Daten abrufen kann und in welchem Format die Daten ausgeliefert werden. Man muss fest implementieren, dass der Name eines Hotels unter dem Schlüssel „name“ zu finden ist und der Name des Ortes unter „city“. Möchte man weitere Informationen über diesen Ort einbinden, muss man einen entsprechenden Web-Service finden und die Zeichenkette „Berlin“ irgendwie mit den dort zu findenden Daten verknüpfen. Die Daten haben keinen „Kontext“, keinen Bezug zu anderen Datensätzen und keine Semantik.

Das mag bei nur wenigen anzubindenden Diensten einer in sich geschlossenen System-Landschaft praktikabel sein, bei der alle zu nutzenden Systeme bekannt sind. Mit Linked Data jedoch kann der gesamte Datenbestand des World Wide Web erschlossen werden.

Linked Data veröffentlichen

Wie also könnte ein Hotel-Service aussehen, der Linked Data bereitstellt? Eins vorweg: Man muss nicht auf JSON oder XML verzichten, nur weil man Linked Data anbietet. Dank Content-Negotiation kann beides nebeneinander harmonieren. Gemäß den Linked-Data-Prinzipien benötigen wir zuerst eine URI als Name für unsere Dinge, also die Hotels. Für die Wahl passender URIs gibt es bereits eine Reihe von Patterns [4]. In erster Linie ist es aber wichtig, dass wir zwischen einem Hotel an sich und dem Dokument über dieses Hotel unterscheiden. Für unsere Beispielanwendung sind die URIs nach folgendem Muster aufgebaut: „http://hotels.datenwissen.de/hotel/<ID>#it“. <ID> ist dabei eine interne ID aus der verwendeten relationalen Datenbank. Sie ist für Anwendungen, die unsere Daten nutzen wollen, irrelevant und wird hier nur verwendet, um jedem Hotel eine eindeutige URI zu geben. Wie in der letzten Ausgabe beschrieben, verwenden wir den Fragment-Identifier „#it“, um das Hotel vom Dokument zu unterscheiden, das durch folgende URI identifiziert wird: „http://hotels.datenwissen.de/hotel/<ID>“. In der UrlMappings.groovy von Grails lässt sich leicht ein Resource-Mapping anlegen, das diese Vorgaben abbildet (siehe Listing 2).

```
static mappings = {
  "/hotel/$id" (resource:"hotel")
}
```

Listing 2

Da der Fragment-Identifier vom Client automatisch abgeschnitten wird, brauchen wir uns diesbezüglich um nichts weiter zu kümmern. Ein Client wird immer nur das Dokument (ohne „#it“) anfordern. Ein Hotel können wir schließlich nicht über HTTP übermitteln; wir sind auf Dokumente mit Informationen zu Hotels beschränkt. Wir werden jedoch die URI des Hotels (mit „#it“) verwenden, um in diesen Dokumenten Aussagen über das Hotel zu treffen. Mithilfe von Content-Negotiation können wir das angeforderte Dokument in einer Vielzahl von Formaten ausliefern. Für das

Auge könnten wir die Informationen zum Beispiel als HTML aufbereiten, gleichzeitig ist es aber auch möglich, RDF und JSON bereitzustellen.

Für unser Beispiel beschränken wir uns auf JSON und RDF in der Turtle-Syntax. Das oben beschriebene URL-Mapping sorgt dafür, dass die show()-Action des HotelController aufgerufen wird, wenn ein GET-Request den Server erreicht (siehe Listing 3).

In Zeile 2 laden wir zunächst das Hotel mit der angeforderten ID aus der Datenbank. Wird kein solches gefunden, liefern wir den Status-Code 404 (Not Found) zurück. Im „withFormat-Block" unterscheiden wir anschließend nach dem vom Client gewünschten Format. „json" ist bereits von Grails auf die MIME-Types „text/json" und „application/json" gemappt; „ttl" müssen wir selbst durch einen zusätzlichen Eintrag in der Config.groovy auf „text/turtle" mappen (siehe Listing 4).

Zu beachten ist außerdem, dass „grails.mime.use.accept.header" auf „true" gesetzt ist, damit Grails den Accept-Header auswertet. Unsere Anwendung ist nun in der Lage, sowohl JSON als auch Turtle-RDF unter der gleichen URI bereitzustellen – je nach Vorzügen des anfragenden Clients. Wir müssen im nächsten Schritt lediglich ein RDF-Dokument aus den Daten unserer Datenbank erzeugen.

Die Bibliothek „groovyrdf" ermöglicht es, RDF-Daten in einer an Turtle angelehnten Syntax zur Laufzeit zu erzeugen. Sie baut auf dem Jena-Framework auf, ist dabei weniger mächtig, dafür aber leichter zu verstehen. RDF lässt sich mit wenigen Code-Zeilen erzeugen oder auslesen. In einem Grails-Service erzeugen wir nun einige RDF-Tripel mit Aussagen zum gewünschten Hotel (siehe Listing 5).

In Zeile 3 instanziieren wir dazu einen „JenaRdfBuilder" und in Zeile 4 generieren wir dynamisch eine URI für das gewünschte Hotel – basierend auf der URI des Hotel-Dokuments, ergänzt um „#it". In den darauffolgenden Zeilen werden innerhalb der geschweiften Klammern vier Aussagen zu dieser URI (also dem Hotel) getroffen. Dazu werden einige tatsächlich existierende Ontologien verwendet: Die Accommodation Ontology (siehe „http://purl.org/acco/ns#") bietet ein Vokabular für Hotels, Ferienwohnungen und ähnliche Unterkünfte. Sie ist eine Ergänzung zur eCommerce-Ontologie „GoodRelations" (siehe „http://purl.org/goodrelations/v1#"), die hier auch für den Namen des Hotels verwendet wird. Um den Standort zu beschreiben, greifen wir auf die „Friend-of-a-Friend"-Ontologie (FOAF) zurück, die, ihrem Namen zum Trotz, über „based_near" den Standort aller möglichen „Spatial Things" beschreiben kann.

Eine Aussage erfolgt im einfachsten Fall durch die Angabe des Prädikat-URI, gefolgt vom gewünschten Wert. Die Werte beziehen wir hier direkt aus dem Hotel-Objekt, zum Beispiel „hotel.name". Hinter „hotel.basedNear" verbirgt sich jedoch kein Objekt-Literal, sondern ein URI, auf den wir verlinken möchten. Daher wird bei „based_near" nochmals ein Klammerpaar geöffnet, um eine „Sub-Ressource" mit dem in „hotel.basedNear" enthaltenen URI zu erzeugen. Das leere Klammerpaar dahinter sagt aus, dass wir an dieser Stelle keine Aussagen zu dieser verlinkten Ressource treffen. Ein Client kann jedoch den URI abrufen, um weitere Informationen zu erhalten.

Statt einen eigenen „Geo-Service" zu entwickeln, können wir einfach auf bestehende Dienste verlinken. „geonames.org" stellt bereits Linked Data für eine Vielzahl von geografischen Orten zur Verfügung. Die Stadt Berlin ist dort durch den URI „http://sws.geonames.org/2950159/" identifiziert. Listing 6 zeigt das RDF-Dokument, das durch unseren Hotel-Service generiert wird. Um unseren Service zu vervollständigen, müssen wir die RDF-Daten im Controller als Dokument zurückliefern (siehe Listing 7).

Über unseren Service erzeugen wir die RDF-Daten und schreiben sie anschließend über die „write()"-Methode im Turtle-Format in einen „StringWriter". Über die Grails-„render()"-Methode senden wir den resultierenden String mit korrektem Content-Type zurück an den Client.

Der Dienst ist tatsächlich unter „http://hotels.datenwissen.de" online. Um eine Liste der gespeicherten Hotels zu erhalten, ruft man einfach diesen URI ab und folgt dann den Links zu weiterführenden Daten der Hotels. Auch der Quellcode ist online zugänglich [5]. Es empfiehlt sich, damit zu experimentieren.

Linked Data konsumieren

In einer weiteren Anwendung werden nun die Daten des Hotel-Service konsumiert.

```
def show(long id) {
  def hotel = Hotel.get(id)
  if (!hotel) {
    render(status: 404)
    return
  }
  withFormat {
    ttl { /* render text/turtle */ }
    json { render hotel as JSON }
  }
}
```

Listing 3

```
grails.mime.types = [
    // [...]
    json:          ['application/json','text/json'],
    ttl:           'text/turtle' // Neu
]
```

Listing 4

```
RdfData hotelToRdf(Hotel hotel) {
  def serverUrl = grailsApplication.config.grails.serverURL
  new JenaRdfBuilder().build {
    "${serverURL}/hotel/${hotel.id}#it" {
      a 'http://purl.org/acco/ns#Hotel'
      'http://purl.org/goodrelations/v1#name' hotel.name
      'http://purl.org/acco/ns#numberOfRooms' hotel.roomCount
      'http://xmlns.com/foaf/0.1/based_near' {
        "$hotel.basedNear" {}
      }
    }
  }
}
```

Listing 5

```
<http://hotels.datenwissen.de/hotel/1#it>
  a <http://purl.org/acco/ns#Hotel> ;
  <http://purl.org/acco/ns#numberOfRooms> "200" ;
  <http://purl.org/goodrelations/v1#name> "Hotel Fiktiva" ;
  <http://xmlns.com/foaf/0.1/based_near>
    <http://sws.geonames.org/2950159/> .
```

Listing 6

```
RdfData rdfData = rdfService.hotelToRdf(hotel)
StringWriter out = new StringWriter()
rdfData.write(out, RdfDataFormat.TURTLE)
render(contentType: 'text/turtle', text: out.toString())
```

Listing 7

```
def index() {
  def rdfLoader = new JenaRdfLoader ()
  RdfData rdfData = rdfLoader.load(
    'http://hotels.datenwissen.de/'
  )
  List<RdfResource> hotelResources = rdfData.listSub-
jects(
    'http://purl.org/acco/ns#Hotel'
  )
  List hotels = hotelResources.collect { res ->
    [
      uri: res.uri,
      name: res("http://purl.org/goodrelations/v1#name")
    ]
  }
  return [hotels: hotels]
}
```

Listing 8

```
def gr = new RdfNamespace('http://purl.org/goodrela-
tions/v1#')
String name = res(gr.name)
```

Listing 9

```
beans = {
  rdfLoader (JenaRdfLoader) {}
  gr (RdfNamespace, 'http://purl.org/goodrelations/v1#')
  acco (RdfNamespace, 'http://purl.org/acco/ns#')
  geo (RdfNamespace, 'http://www.geonames.org/ontol-
ogy#')
  foaf (RdfNamespace, 'http://xmlns.com/foaf/0.1/')
}
```

Listing 10

```
<g:each in="${hotels}" var="hotel">
    <g:link controller="rating" action="show"
      params="[uri:hotel.uri]">
        ${hotel.name}
    </g:link>
</g:each>
```

Listing 11

```
def hotelResource = rdfLoader.loadResource(uri)
String name = hotelResource (gr.name)
int numberOfRooms = hotelResource (acco.numberOf-
Rooms)
String cityUri = hotelResource(foaf.basedNear).uri
RdfResource cityResource = rdfLoader.
loadResource(cityUri)
String cityName = cityResource(geo.name)
```

Listing 12

Diese Hotel-Rating-Web-Anwendung soll es den Besuchern ermöglichen, Hotels zu bewerten, ohne dass die Anwendung selbst Daten über Hotels speichert. Um das Beispiel einfach zu halten, werden wir zunächst nur den oben gezeigten Hotel-Service abfragen. Darauf aufbauend ist es jedoch leicht, weitere Teile des „Web of Data" zu erschließen.

Unter „http://hotels.datenwissen.de" stellt der Hotel-Service eine Liste der verfügbaren Hotels zur Verfügung. Für die Hauptseite der Hotel-Rating-Anwendung fragen wir diese Liste ab, um aus den Daten eine Auswahl der verfügbaren Hotels zu rendern (siehe Listing 8).

In den Zeilen 3 bis 5 werden über eine „RdfLoader"-Instanz RDF-Daten vom Hotel-Service abgefragt. Um Content-Negotiation kümmert sich der RdfLoader selbst, wir müssen uns nicht um unterschiedliche RDF-Syntaxen kümmern und nichts selbst parsen. Stattdessen erhalten wir ein „RdfData"-Objekt, aus dem wir Aussagen über die vorhandenen Ressourcen auslesen können. Dies erfolgt in den Zeilen 6 bis 8. Über „listSubjects()" erhalten wir alle Ressourcen des Typs „http://purl.org/acco/ns#Hotel", also alle Hotels, die uns der Service lieferte. Die Daten, die uns interessieren, URI und Name der Hotels, geben wir als Map zurück, sodass sie für das Rendern der Groovy Server Page (GSP) verwendet werden können. Über „res.uri" bekommen wir den URI der Ressource „res" und über den Aufruf „res(String)" kommen wir an die Informationen heran, die eine Ressource unter dem übergebenen Prädikat bereitstellt. „res(‚http://purl.org/goodrelations/v1#name')" liefert uns somit den Namen des Hotels.

Statt mit langen URIs zu hantieren, lassen sich auch Namespaces deklarieren. Legen wir einen Namespace namens „gr" für die Ontologie „http://purl.org/goodrelations/v1#" an, können wir anschließend über die Kurzschreibweise „gr.name" auf die gleiche Eigenschaft zugreifen (siehe Listing 9). Um weder den RdfLoader noch die Namespaces ständig instanziieren zu müssen, können wir sie bequem als Spring-Beans in der Datei „resources.groovy" hinterlegen (siehe Listing 10). In der GSP-Seite rendern wir für jedes Hotel einen Link zu „/rating/show" mit dem Hotel-URI als Parameter (siehe Listing 11). Abbildung 1 zeigt das Ergebnis.

Klickt ein Nutzer auf einen der Links, wird die „show()"-Action des RatingController aufgerufen. Sie nutzt den übergebenen URI, um weitere Informationen zum Hotel nachzuladen (siehe Listing 12).

Über „loadResource()" laden wir gezielt Daten zu der durch den URI identifizierten Ressource, also das gerade angeforderte Hotel. Die Attribute „gr.name" und „gr.numberOfRooms" werden auf die oben vorgestellte Weise abgefragt. Interessant wird es wieder ab Zeile 4: Hinter dem Prädikat „foaf.basedNear" verbirgt sich kein Objekt-Literal, sondern eine weitere Ressource – nämlich der Ort, an dem sich das Hotel befindet. Wir fragen daher dessen URI ab und nutzen ihn, um mit dem rdfLoader Daten über den Ort zu laden. Dabei kann es uns egal sein, welcher Service sich hinter diesem URI verbirgt, solange er wieder Linked Data bereitstellt. Um den Namen des Orts zu erhalten, greifen wir auf die Eigenschaft „geo.name" zu. Die gewonnenen Daten können wir wieder an die GSP liefern, um sie zu rendern.

Hotels bewerten

Das Bewerten von Hotels hat anschließend nicht mehr viel mit Linked Data zu tun. In einer „save()"-Action erzeugen wir

Abbildung 1: Auflistung der vom Hotel-Service gelieferten Hotels

lediglich ein neues HotelRating-Objekt mit den vom User eingegebenen Daten: „new HotelRating(ratedHotelUri: uri, rating: rating, comment: comment).save()“. Zu beachten ist hier lediglich, dass wir den URI des bewerteten Hotels speichern, um festzuhalten, worauf sich die Bewertung bezieht. Um alle Bewertungen zu einem Hotel zu finden, suchen wir nach all jenen, die sich auf den URI des Hotels beziehen: „HotelRating.findAllByRatedHotelUri(uri)“.

Alles Weitere ist reine Grails-Funktionalität, auf die an dieser Stelle nicht näher eingegangen wird. Abbildung 2 zeigt die fertige Seite mit den vom Hotel-Service abgefragten Informationen und den von der Anwendung selbst gespeicherten Bewertungen. Auch der Quellcode der Hotel-Rating-Anwendung ist online frei zugänglich [6]. Eine laufende Instanz ist unter http://hotel-rating.datenwissen.de dargestellt.

Das Beispiel weitergedacht

Anhand eines konkreten Beispiels haben wir nun sowohl publiziert als auch konsumiert und uns bei der Abfrage von Hotels auf einen ganz konkreten Service beschränkt. Dies muss jedoch nicht sein. Zum Beispiel könnten wir eine semantische Suchmaschine nutzen, um alle Ressourcen vom Typ „http://purl.org/acco/ns#Hotel“ im Web of Data zu finden und zur Bewertung anzubieten. Die dadurch entstehende Heterogenität der RDF-Aussagen muss allerdings entsprechend in der Implementierung berücksichtigt werden. Nicht alle Hotel-Ressourcen im Web of Data werden exakt dem gleichen Aufbau folgen. Im Gegensatz zu den Key-Value-Paaren eines JSON-Datensatzes sind bei RDF aber auch die Prädikate durch URIs identifiziert und durch strukturierte Daten beschrieben. So lässt sich zum Beispiel ausdrücken, dass zwei Prädikate das Gleiche bedeuten. Erfahrungsgemäß setzen sich bestimmte Ontologien als De-facto-Standards durch, wie die FOAF-Ontologie für die Beschreibung von Personen und deren Beziehungen zueinander.

Eine Anwendung, die die Daten aus dem ganzen Web abfragt, ohne genau zu wissen, aus welchen Diensten sie stammen, sollte dennoch tolerant gegenüber dem Aufbau der Ressourcen sein. Sie sollte auf Daten, die nicht in der erwarteten Form ausgedrückt werden, im Zweifel verzichten können oder die Ressource verwerfen. Dies ist jedoch ein kleiner Verlust im Verhältnis zu dem riesigen Datenpool, der uns zur Verfügung steht, wenn wir das Web als globale und dezentral verwaltete Datenbank nutzen.

Weiterhin ist es erstrebenswert, dass Dienste, die Linked Data konsumieren, auch selbst wieder Linked Data veröffentlichen. Die Hotel-Rating-Anwendung könnte die Bewertungen ebenfalls als Linked Data bereitstellen und mit den Hotel-URIs verlinken. Auf diese Weise können später alle Bewertungen zu einem Hotel im Web abgefragt werden. Es könnte viele verschiedene Bewertungsplattformen geben und dennoch kann eine Suche alle verfügbaren Bewertungen einbeziehen, unabhängig davon, auf welcher Plattform sie abgegeben wurden.

Abbildung 2: Anzeige eines Hotels und seiner Bewertungen

Social Web of Data

Was unserem Hotel-Rating-Dienst noch fehlt, ist eine Authentifizierung der Nutzer. Prinzipiell können auch Personen über einen URI identifiziert und Informationen mit ihnen verlinkt werden. Wenn sich ein Nutzer über seinen URI authentisiert, könnten seine Bewertungen und Kommentare mit ihm verlinkt werden, ohne dass unser Dienst eine eigene Benutzerverwaltung bereitstellen muss und auch ohne dass der Nutzer bei einem zentralen Dienst wie Facebook angemeldet sein muss. Erfreulicherweise ist genau dies mit dem WebID-Protokoll [7] möglich. In der nächsten Ausgabe werden wir uns dieses Protokoll genauer ansehen.

Fazit

Das Bereitstellen und Verwerten von Linked Data ist nicht schwieriger als der Umgang mit XML-Dokumenten. Die Verlinkung der Daten macht es leicht, Dienstunabhängig auf Daten zuzugreifen. Statt über proprietäre APIs an einzelne Dienste anzudocken, behandeln wir Daten wie den Rest des Webs: Wir greifen über URIs auf sie zu und folgen den in ihnen enthaltenen Links zu weiteren Daten. Linked Data eignet sich daher besonders beim Zusammenführen von Daten aus einer Vielzahl heterogener Systeme.

Referenzen

[1] http://grails.org
[2] http://datenwissen.de/groovyrdf
[3] http://jena.apache.org
[4] http://patterns.dataincubator.org/book
[5] https://github.com/angelo-v/hotels
[6] https://github.com/angelo-v/hotel-rating
[7] http://webid.info

Angelo Veltens
angelo.veltens
@online.de

Angelo Veltens studierte Angewandte Informatik an der Dualen Hochschule Baden-Württemberg Karlsruhe und befasste sich in Studienarbeiten und Abschlussarbeit mit Linked Data und semantischem Wissensmanagement. Heute ist er als Software-Entwickler in Braunschweig tätig, arbeitet in seiner Freizeit am „Social Web of Data“ und referiert auf Konferenzen zu diesem Thema.

Vagrant: Continuous Delivery ganz einfach

Sebastian Laag, adesso AG

Das Einrichten von Test-Servern oder Entwicklungsrechnern ist häufig extrem zeitaufwändig und problematisch. Virtualisierung ist eine Lösung, um Server dynamisch bereitzustellen und Applikationen zu testen. Vagrant stellt mit leicht konfigurierbaren virtuellen Maschinen (VM) eine Lösung zur Verfügung, um Entwicklungsumgebungen oder ganze Infrastrukturen aufzubauen.

Vagrant nutzt zur Erstellung und Installation von VMs das quelloffene Virtualisierungstool VirtualBox von Oracle. Durch die Konfigurationsmanager „Chef" und „Puppet" kann vor jedem Start sowohl die VM als auch die auf der VM installierte Anwendung aktualisiert werden.

Abbildung 1 zeigt den Zusammenhang zwischen „Vagrant", „VirtualBox" und „Chef" zur Installation von Anwendungen wie Java oder Tomcat auf einer VM. Dank dieser leichtgewichtigen und portablen Lösung kann die Produktivität in der Entwicklung deutlich erhöht werden, da der manuelle Aufbau von komplexen Test-Umgebungen entfällt.

Außerdem wird so jede Änderung an der Infrastruktur nachvollziehbar und kann später auch auf andere Umgebungen übertragen werden. Vagrant eignet sich nicht nur zum Aufbau von Infrastrukturen, sondern kann auch im Rahmen von automatisierten Tests verwendet werden. Für diesen Fall wurde das Vagrant-Plug-in für den Continuous-Integration-Server Jenkins entwickelt. Es bietet eine Möglichkeit an, VMs mit Vagrant im Rahmen von Continuous Integration für automatisierte Tests bereitzustellen. Die virtuellen Instanzen können auf die Tests zugeschnitten werden. Das Vagrant-Plug-in stellt verschiedene konfigurierbare Schritte für einen Testlauf zur Verfügung.

Dieser Artikel beschreibt, wie Vagrant einzelne Entwickler, Teams oder ganze Unternehmen bei unterschiedlichsten Projekten unterstützen kann. Zudem werden das Vagrant-Plug-in für Jenkins vorgestellt und seine Möglichkeiten im Bereich von automatisierten Tests erläutert.

Die Entstehung von Vagrant

Im Jahr 2010 initiierten Mitchell Hashimoto und John Bender Vagrant. Das Projekt basiert auf Oracles quelloffenem Virtualisierungstool VirtualBox und verwendet Chef und Puppet zur Konfiguration der virtuellen Maschinen. Im März 2012 wurde die erste stabile Version 1.0 veröffentlicht [1]. Mittlerweile existiert Vagrant in Version 1.0.6, das einige Stabilitätsverbesserungen beinhaltet. Heute wird Vagrant von vielen Organisationen wie Nokia, at&t und Mozilla eingesetzt. Durch den vielfachen Einsatz in Unternehmen haben sich neue Anforderungen an Vagrant ergeben. Um diese umzusetzen, hat der Initiator des Projekts mittlerweile ein eigenes Unternehmen gegründet [2].

Für die Verwendung von Vagrant sind nur die aktuellste Version von Vagrant sowie eine VirtualBox-Installation [3] erforderlich. Vagrant stellt ein Kommandozeilen-Tool zur Verfügung, über das die virtuellen Maschinen heruntergeladen, gestartet, gestoppt und konfiguriert werden können. Die Konfiguration erfolgt über sogenannte Vagrant-Files. Mit diesen wird die vollständige Konfiguration der VMs festgelegt. Für den Anfang wird eine mit VirtualBox erstellte „Box" benötigt. In dieser müssen das grundlegende Betriebssystem sowie die „VirtualBox Guest Additions" installiert werden. Diese werden beispielsweise für Funktionen wie geteilte Verzeichnisse und Port Forwarding benötigt. Zusätzlich erfordert die Box einen SSH-Server mit Key-basiertem Zugang für den Vagrant-Benutzer und Ruby, sowie RubyGems für die Installation von Chef und Puppet. Diese sind zudem für die Bereitstellung der virtuellen Maschine notwendig.

Falls der Aufwand einer eigens eingerichteten Box zu hoch ist, existieren be-

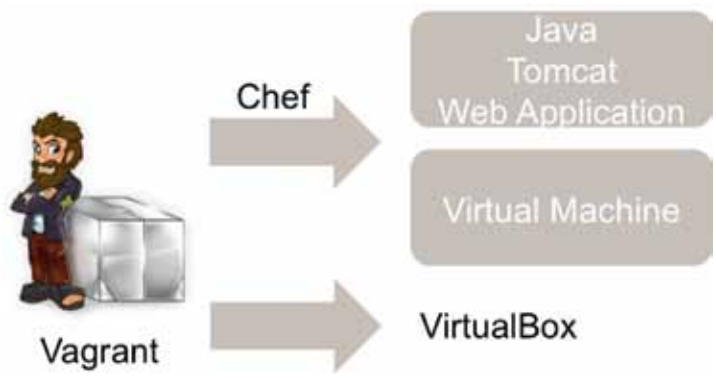


Abbildung 1: Die Zusammenhänge

reits von Vagrant bereitgestellte Boxen, die verwendet werden können [4]. Mit dem Tool „veewee“ lassen sich ebenfalls Vagrant-Box-Images selbst mit Tool-Unterstützung erstellen [5]. Auf Basis einer solchen Box kann dann die Installation der vollständigen Infrastruktur mithilfe von Chef oder Puppet automatisiert erfolgen.

Ein Beispiel-Projekt

Das nachfolgende Beispiel verdeutlicht den Einsatz von Vagrant und Chef zur Konfiguration einer VM. Dabei werden auf der VM ein Tomcat-Server installiert und eine Demo-Anwendung bereitgestellt. Mit den auf „www.vagrantup.com“ zur Verfügung stehenden Box-Images kann der Start einer VM mit einem einfachen Vagrant-File mit dieser Konfiguration erfolgen (siehe Listing 1).

Mittels „config.vm.box“ wird der Name der VM festgelegt. Die „box_url“ ist optional und wird nur verwendet, falls auf dem Rechner kein Box-Image mit dem angegebenen Namen gefunden werden kann – dann wird es nämlich von dieser URL heruntergeladen. Ein solches Image kann bei Vagrant auch über die Kommandozeile definiert werden.

Der Abschnitt „config.vm.provision :chef_solo do |chef|“ definiert, dass die Anwendungen mithilfe von Chef installiert und konfiguriert werden sollen. Dieser verwaltet die Installationsanweisungen in sogenannten „Cookbooks“. Daher wird mit „chef.cookbooks_path“ der Pfad definiert, an dem die Kochbücher abgelegt sind. Die Chef Community stellt bereits eine Vielzahl von Kochbüchern zur Verfügung, die auf die eigenen Bedürfnisse angepasst werden können [6]. Jedes besitzt ein eigenes Unterverzeichnis, das mehrere Dateien enthält, die die Konfigurationen beinhalten.

Diese Kochbücher können durch Rollen konfiguriert beziehungsweise zu einer Komposition zusammenzufügt werden. Das ist nützlich, um einen Server, der aus mehreren Kochbüchern besteht, zusammenzustellen – also beispielsweise Tomcat, Java und eine Web-Anwendung. Zu diesem Zweck werden JSON-Dateien geschrieben, die die geänderten Einstellungen festlegen. Listing 2 zeigt eine solche Rolle.

Diese Rolle legt fest, welche Web-Applikation in der VM abgelegt werden soll. Mit der sogenannten „run_list“ werden die auszuführenden Chef-Kochbücher definiert. Die Reihenfolge ist dabei entscheidend. In diesem Fall werden der Ubuntu-Installationsassistent „apt-get“ aktualisiert und die Demoanwendung „demo.war“ in das Deployment-Verzeichnis des Tomcat kopiert. Um auf die in der VM installierte Anwendung zuzugreifen, ist die Einrichtung einer Port-Weiterleitung notwendig (siehe Listing 3).

Mit dieser Zeile wird Port 8080 der VM auf den lokalen Port 18080 weitergeleitet. Über die URL „http://localhost:18080/demo“ kann nun die Startseite der Demo-Anwendung aus der Host-Umgebung erreicht werden. Die bis hierhin vorgestellte Konfiguration lässt sich beliebig erweitern.

„Hello World“ – und nun?

Anhand des Beispiels wird deutlich, dass es extrem einfach ist, eine Vagrant-Konfiguration aufzubauen. Diese kann nun der Startpunkt sein, um noch weitere Programme mittels Chef zu konfigurieren oder das Vagrant-File zu einer Multi-VM-Umgebung auszubauen und so ganze Infrastrukturen entwerfen. Zu diesem Zweck deklariert man für jede VM innerhalb des Vagrant-Files einen eigenen Präfix (siehe Listing 4).

Die Konfiguration der einzelnen VMs erfolgt nun innerhalb des „config.vm.define“-Abschnitts. Hier können nun auch die einzelnen Chef- oder Puppet-Einstellungen erfolgen. Um die Kommunikation zwischen den VMs zu ermöglichen, wird ein sogenanntes „Host-only-Netzwerk“ verwendet. Nur dieses unterstützt eine direkte Kommunikation (siehe Listing 5).

Im Idealfall wird das Vagrant-File zusammen mit den Chef-Kochbüchern oder Puppet-Konfigurationen versioniert, sodass für einzelne Entwicklungsteams oder das gesamte Unternehmen Änderungen sofort zur Verfügung stehen.

Anwendung von Vagrant

Die bisher erstellte Konfiguration lässt sich über das mitgelieferte Kommandozeilen-Tool steuern. Dazu wird in das Verzeichnis des Vagrant-Files navigiert und der Befehl „vagrant up“ ausgeführt. Dieser startet sämtliche innerhalb des Vagrant-Files

```
Vagrant::Config.run do |config|
  config.vm.box_url="http://files.vagrantup.com/lucid32.
box"
  config.vm.box = "lucid32"
  config.vm.provision :chef_solo do |chef|
    chef.cookbooks_path = ["cookbooks"]
    chef.roles_path = ["roles"]
    chef.add_role("tomcatserver")
  end
end
```

Listing 1

```
  "name":"tomcatserver",
  "description":"",
  "json_class":"Chef::Role",
  "default_attributes": {
  },
  "override_attributes": {
    "webapp": {
      "webapp":"demo.war"
    }
  },
  "chef_type":"role",
  "run_list": [
    "recipe[apt]",
    "recipe[java]",
    "recipe[tomcat]",
    "recipe[webapp]"
  ],
  "env_run_lists": {
  }
}
```

Listing 2

```
Vagrant::Config.run do |config|
  ...
  config.vm.forward_port 8080, 18080
  ...
End
```

Listing 3

```
...
  config.vm.define :lucid32 do |lucid32_config|
    lucid32_config.vm.box = "lucid32"
  end

  config.vm.define :lucid64 do |lucid64_config|
    lucid64_config.vm.box = "lucid64"
  end
...
```

Listing 4

konfigurierten virtuellen Maschinen. Wird nach dem Befehl noch der Name der virtuellen Maschine, zum Beispiel „lucid32“, angehängt, so wird nur diese gestartet. Haben sich nach dem Start der VMs Änderungen an der Konfiguration ergeben, können diese mit dem Befehl „vagrant provision“ übernommen werden. Dabei ist kein Neustart der VMs erforderlich. Chef erkennt sogar, welche Passagen der Konfiguration verändert wurden, sodass bei jedem „provision“-Aufruf nur die geänderten Teile übernommen werden. Ist die Arbeit beendet, werden die VMs mittels „vagrant halt“ heruntergefahren. VMs, die nicht mehr benötigt werden, lassen sich mit „vagrant destroy“ entfernen. Alle verfügbaren Befehle können durch Ausführen des Befehls „vagrant“ angezeigt werden. Bei Bedarf kann man sogar eigene Vagrant-Kommandos definieren [7].

Für Vagrant existiert zusätzlich ein Java-API, das die Erstellung von Vagrant-Files und die Verwendung des Kommandozeilen-Tools mithilfe von Java-Code ermöglicht. Es existiert bisher allerdings nur in einer experimentellen Version [8]. Eine Möglichkeit, ein solches API zu benutzen, ist die Verwendung von Vagrant zur Unterstützung von selbst geschriebenen Tests. Dabei wird innerhalb des Tests die notwendige virtuelle Maschine durch Java-Code konfiguriert, für den Test bereitgestellt und innerhalb des Tests genutzt.

Im Rahmen von automatisierten Tests, etwa mit Jenkins, kann das Vagrant-Jenkins-Plug in [9] genutzt werden. Damit lässt sich für einen Jenkins-Job ein Vagrant-File festlegen, das das Plug-in ausführt und somit die virtuelle Maschine bereitstellt. Diese kann innerhalb des Test-Laufs genutzt und umkonfiguriert werden. Nach Test-Abschluss wird sie vom Plug-in automatisch heruntergefahren und alle verwendeten Ressourcen werden wieder freigegeben. Bei der Nutzung des Plug-ins ist zu beachten, dass es JRuby nutzt, das aktuell keine Windows-7-64-Bit-Unterstützung anbietet. Weiterhin ist zu bedenken, dass sich das Plug-in noch in einer frühen Version befindet und noch einige Probleme mit sich bringt [10].

```
...
  config.vm.define :lucid32 do |lucid32_config|
    lucid32_config.vm.box = “lucid32“
    lucid32_config.vm.network :hostonly,“192.168.1.10“
, :netmask => “255.255.255.0“
  end
...
```

Listing 5

Fazit

Der Vorteil von Vagrant liegt auf der Hand. Für den Fall, dass man ein frei zugängiges Box-Image nutzt oder innerhalb der Organisation ein Verzeichnis pflegt, in dem auch eigene Box-Images abgelegt werden können, hat jeder neue Entwickler sofort die Möglichkeit, nach Bedarf eine Test-Umgebung zu erzeugen. Änderungen an den Konfigurationen der Box-Images können jederzeit vorgenommen und dem gesamten Team oder dem ganzen Unternehmen zur Verfügung gestellt werden. Somit ist sichergestellt, dass jeder Entwickler die gleiche Entwicklungsumgebung verwendet. Dafür muss allerdings auch das jeweilige Vagrant-Projekt unter Versionsverwaltung gestellt werden, sodass jeder Entwickler Änderungen sofort bemerkt und sich seine Entwicklungsumgebung neu erzeugen kann.

Da sich mit Vagrant nicht nur Entwicklungsumgebungen für Entwickler konfigurieren lassen, sondern auch gesamte Infrastrukturen, ist es auch denkbar, auf diese Weise gemeinsam genutzte Dev-, Test- und Produktions-Umgebungen zu verwalten.

Konfigurationen können also zentral in Zusammenarbeit von Entwicklung und Betrieb (DevOps) gepflegt und dann auf allen Umgebungen genutzt werden. Das vermeidet Differenzen zwischen Dev-, Test- und Produktions-Umgebung und die daraus resultierenden Fehler.

Die entwickelten Chef- oder Puppet-Skripte sind auch ohne Vagrant nutzbar, um beliebige virtuelle oder physische Rechner entsprechend zu konfigurieren. Dabei ist es auch möglich, sehr einfach Cluster aufzubauen und diese zentralisiert mit neuen Versionen der Software oder der Installations-Anweisungen zu versorgen.

Die genannten Szenarios werden bereits in vielen Unternehmen umgesetzt. Durch die vielfache Verwendung von Vagrant haben sich weitere Änderungswünsche ergeben, die in zukünftigen Versionen umgesetzt werden. So ist unter anderem geplant, nicht nur die Virtualisierung mittels VirtualBox zu unterstützen, sondern potenziell jedes mögliche Virtualisierungs-Tool mit Vagrant verwenden zu können. Eine weitere Neuerung, die in Version 1.1.x Einzug in Vagrant erhalten wird, ist ein neues Plug-in-System, mit dem jeder Entwickler Erweiterungen für Vagrant entwickeln kann. Doch auch ohne diese Neuerungen ist Vagrant schon jetzt bereit für den produktiven Einsatz. Ein solches Projekt wird auf Bitbucket vom Autor zur freien Verfügung gestellt [11].

Weitere Informationen

[1] http://www.golem.de/news/virtuelle-entwicklungsumgebung-vagrant-1-0-veroeffentlicht-1203-90316.html
[2] http://www.hashicorp.com/blog/announcing-hashicorp.html
[3] http://downloads.vagrantup.com
[4] http://www.vagrantbox.es
[5] https://github.com/jedi4ever/veewee
[6] http://community.opscode.com/cookbooks
[7] http://vagrantup.com/v1/docs/extending/commands.html
[8] https://github.com/guigarage/vagrant-binding
[9] https://wiki.jenkins-ci.org/display/JENKINS/Vagrant+Plugin
[10] https://github.com/rtyler/vagrant-plugin/issues
[11] https://bitbucket.org/laag/vagrant-chef

Sebastian Laag
sebastian.laag@adesso.de

Sebastian Laag (Dipl. Inf. Univ.) ist als Software-Engineer bei der adesso AG in Dortmund tätig und arbeitet dort als Entwickler an Java-basierten Web-Anwendungen.

Cobol und Java: Zwei Sprachen kommen sich näher

Rolf Becking, Micro Focus GmbH

Mit Java und Cobol treffen zwei unterschiedliche Sprachkonzepte aufeinander, die auf den ersten Blick schwer zu vereinbaren sind. Dabei ist die Kombination beider Sprachen für viele Anwendungsfälle hochinteressant. Jetzt gibt es einen Compiler, der Cobol zu Java-Byte-Code kompilieren kann und damit die Ausführung von Cobol-Programmen in der JVM ermöglicht. Damit ist eine nahtlose Integration der beiden so unterschiedlichen Programmiersprachen realisierbar.

Java ist heute als Programmiersprache erste Wahl. Ein klar definiertes, objektorientiertes Konzept, eine große Auswahl an Frameworks, die dem Programmierer viel Arbeit abnehmen, eine Menge Tools, die ihm das Leben erleichtern, sowie die herausragenden Stärken in der Internet-Programmierung haben Java zu einer Sprache gemacht, auf die die meisten Unternehmen heute nicht mehr verzichten wollen und können. Aber diese Unternehmen entwickeln nicht nur neue Software, sie haben auch eine mehr oder weniger große Anzahl an Bestandsanwendungen – häufig in Cobol –, die gepflegt werden wollen.

Die Ansätze, mit denen man versucht, dem damit verbundenen Wartungsaufwand Herr zu werden, sind ganz unterschiedlich. Manche Unternehmen gehen sogar so weit, dass sie versuchen, ihre Altapplikationen in Java von Grund auf neu zu programmieren. Eine Vorgehensweise, die nicht immer erfolgreich ist und die sich in vielen Fällen allein aus wirtschaftlichen Überlegungen von vornherein verbietet. Denn eine Anwendung, die über zwanzig Jahre gewachsen ist, in der vielleicht Hunderte von Mann-Jahren an Entwicklungsaufwand stecken, lässt sich nicht im Vorbeigehen neu programmieren. Große Java-Projekte dieser Art erstrecken sich daher meist über viele Jahre und sind in vielen Fällen auch nach einem Jahrzehnt noch nicht abgeschlossen.

Natürlich ist nicht jede Bestandsanwendung in Cobol codiert, diese Sprache nimmt jedoch aufgrund ihrer Verbreitung eine Sonderstellung ein. In der Vergangenheit wurden fast alle bedeutenden Applikationen in Großunternehmen wie Banken, Versicherungen, Finanzbehörden, Pensionsfonds und vielen anderen in Cobol entwickelt, implementiert und auf Großrechnern betrieben. Diese Anwendungen sind mitunter 20, 30 oder in Teilen sogar 40 Jahre alt und in dieser Zeit sind sie ständig gewachsen, immer größer und komplexer geworden.

Die Unternehmen, die solche Anwendungen betreiben, haben sich nicht selten eine Java-Strategie auf die Fahnen geschrieben. Natürlich kommen sie nicht ohne einen Internet-Auftritt aus, natürlich wollen sie Web-Anwendungen mit Daten ihrer Kern-Systeme betreiben und natürlich wollen sie nun auch Mobile Devices nutzen. So verwundert es nicht, dass die Verbindung moderner, in Java implementierter Anwendungsteile mit der guten alten Geschäftslogik heute zu den Standard-Anforderungen im Umgang mit Cobol gehört.

Verbindung von Java und Cobol

Für die Verbindung von Java mit irgendeiner anderen Sprache stellt sich grundsätzlich immer folgendes Problem: Java läuft in einer virtuellen Maschine (JVM). Will man von dort Programme aufrufen, die zu Maschinencode, dem sogenannten „nativen Code" (DLLs unter Windows oder Shared Objects/Libraries unter Unix/Linux), kompiliert und gelinkt sind, so bietet die JVM dafür eine Schnittstelle an, das „Java Native Interface" (JNI). Diese ist nicht nur kompliziert aufgebaut, sie hat einen großen Haken: Beim Aufruf wird die JVM verlassen und ihr damit jegliche Kontrolle entzogen. Das kann besonders kritisch werden, wenn das native Programm wegen eines Fehlers abstürzt. Die Folge ist, dass die JVM ebenfalls automatisch endet. Aus diesem Grund verbietet sich die Verwendung des JNI in Web- oder Application-Servern.

Selbst wenn man sich nicht innerhalb eines solchen Servers bewegt, ist diese Art der Verbindung nicht einfach zu handhaben. Ein Cobol-Programm muss nämlich an die Gegebenheiten der Java-Welt angepasst werden. Die Java Runtime arbeitet grundsätzlich „multi-threaded", wofür ein Cobol-Programm normalerweise nicht vorgesehen ist. Daneben gibt es das Problem der Inkompatibilität der Datentypen – ein String in Java ist beispielsweise nicht gleich einem String in der Cobol-Welt. Diese Probleme können technisch zwar gelöst werden, sind aber nicht einfach in den Griff zu bekommen.

Der umgekehrte Aufruf von Cobol zu Java hat mit den gleichen Problemen zu kämpfen. Der Aufruf als solcher lässt sich in Cobol, das mittlerweile ja objektorientierte Spracherweiterungen umfasst, allerdings einfacher erstellen. Zudem muss das Cobol-Runtime-System in der Lage sein, die gewünschte JVM zu laden. Die Anbieter von Cobol-Compilern haben in der Regel für diese Themen einen jeweils proprietären Ansatz gewählt, um dem Programmierer das Codieren des JNI-API zu ersparen.

Aufruf über Protokoll-Schnittstelle

Als Alternative zu JNI-basierten Lösungen bietet sich in der Welt der Web- und Application-Server ein Aufruf über eine Protokoll-Schnittstelle an, die es ermöglicht, mit einem externen Server zu kommunizieren.

Dabei können Web-Services oder die Java Connector Architektur (JCA) zum Einsatz kommen. Compiler-Anbieter wie Micro Focus stellen Techniken bereit, mit denen ein Cobol-Programm einen Web-Service aufrufen kann oder aber selbst als Web- beziehungsweise J2EE-Service in einem Cobol-Server eingesetzt werden kann, sodass es für einen entsprechenden Aufruf zur Verfügung steht.

Auf diese Weise werden die Probleme der JNI zwar umgangen, allerdings hat der Anwender nun zwei Server nebeneinander, einen für Java und einen für Cobol. Diese Koexistenz zweier unterschiedlicher Server stellt grundsätzlich kein Problem dar, kann aber für Anwendungssysteme, die transaktionsorientiert arbeiten, zu einer echten Herausforderung werden. Die beiden Server arbeiten nämlich mit unterschiedlichen Transaktionen, die zwar miteinander koordiniert werden können, aber nicht in einer gemeinsamen Transaktion zusammengefasst werden. Im Anwendungsfall bedeutet dies, dass ein verändernder Datenzugriff des Java-Programms im aufgerufenen Cobol-Programm nicht weiterverarbeitet werden kann, da er noch nicht von der umschließenden Java-Transaktion committet worden ist. Dies mag in vielen Applikationen keine Rolle spielen, hat sich aber bei der Migration von transaktionsorientierten Systemen mit einem generierten Front-end – zum Beispiel dem VisualAge Generator für CICS – als echtes Problem herausgestellt, an dem eine komplette Migration einer solchen Anwendung scheitern kann.

Cobol in der JVM

Mit der Kompilierung von Cobol zu Java-Byte-Code, wie sie Visual Cobol von Micro Focus nun bietet, werden die genannten Probleme aus der Welt geschafft und eine Integration der beiden Programmiersprachen ermöglicht. Zum Hintergrund dieser Integration: Der Cobol-Spezialist Micro Focus hat bereits einige Erfahrungen mit der Generierung unterschiedlicher Code-Formate gesammelt. Dazu gehört zum einen der eigene, plattformunabhängige „.int"-Code, der zu Optimierungszwecken zum proprietären „.gnt"-Code oder dem jeweiligen Objektcode der betreffenden Plattform weiter kompiliert und anschließend gelinkt werden kann, sowie zum anderen die Unterstützung der in der .Net-Welt übliche Common Intermediate Language (CIL), früher Microsoft Intermediate Language (MSIL). Dies ermöglicht die Integration von Cobol-Applikationen in die .Net-Welt und damit die Aufrufbarkeit von Cobol-Programmen aus VB.NET und C# sowie umgekehrt. Die mit Visual Cobol angebotene Kompilierung zum Java-Byte-Code als drittes Format ist da eine logische Konsequenz.

Die Kompilierung zum Java-Byte-Code erlaubt es, jedes klassische Cobol-Programm zu einer Java „.class"-Datei zu kompilieren, die von der JVM ausgeführt werden kann. Damit lassen sich bestehende Cobol-Altanwendungen ohne jede Veränderung als Java-Byte-Code ausführen. Anwendungen vollständig neu in Java zu entwickeln – „rip and replace" –, wird damit überflüssig.

Ob es jedoch überhaupt sinnvoll ist, Programme komplett in Java-Byte-Code zu kompilieren, sei dahingestellt, denn ohne eine Java-Umgebung, in die das jeweilige Programm integriert werden soll, hat der Java-Byte-Code gegenüber dem nativen Code auch keine Vorteile.

Integration in die Java-Welt

Für die Integration in die Java-Welt bietet sich jedoch eine Vielzahl von Möglichkeiten, die hier an einem kleinen Beispiel erläutert werden sollen.

Für die Aufrufbarkeit aus der rein objektorientierten Java-Welt muss das aufzurufende Cobol-Programm eine Klasse und die dazu gehörenden Methoden zur Verfügung stellen (siehe Abbildung 1). Die Cobol-Syntax ist spätestens seit dem ISO-2002-Standard um entsprechende Konstrukte erweitert worden, die Micro Focus als Compiler-Hersteller noch einmal vereinfacht hat. Für die Definition einer Klasse reicht ein „class-id"-Paragraph mit dem Namen der Klasse, zur Definition einer dazu gehörenden Methode der „method-id"-Paragraph mit dem Namen der Methode, wie in Listing 1 dargestellt.

Der Cobol-Code der Methode „CountTo" wird später betrachtet. Zum Vergleich zeigt Listing 2 die Definition einer Java-Klasse und die Implementierung einer Java-Methode.

Die Java-Klasse „JavaOutput" implementiert ein Interface – ein Interface, das erstaunlicherweise im Cobol Code definiert wurde (siehe Listing 3).

```
class-id BizLogic.
working-storage section.
method-id CountTo.
```

Listing 1

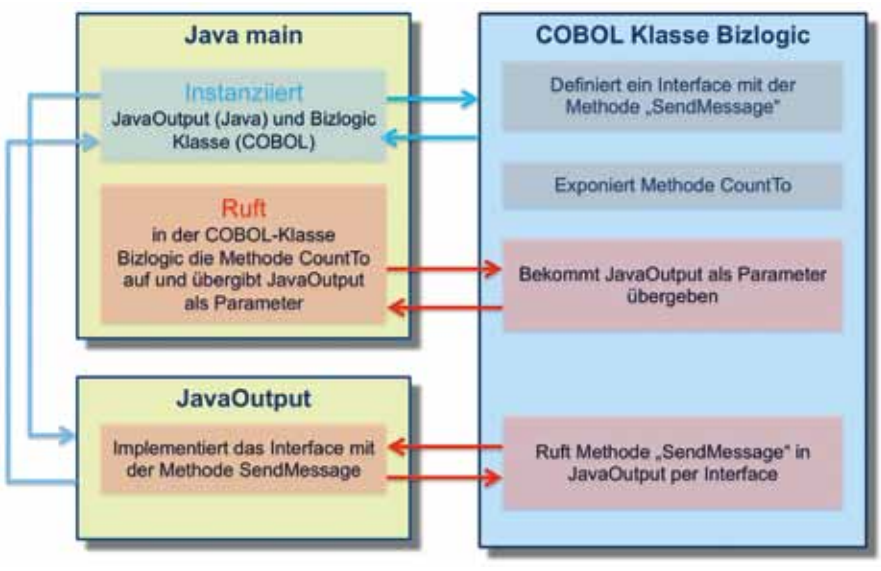


Abbildung 1: Zusammenarbeit von Cobol und Java in einem Programm

Die Syntax „procedure division using by value msg as string" mag hartgesottene Cobol-Programmierer überraschen. Es ist tatsächlich möglich, in Cobol bei einem Parameter anzugeben, wie er übergeben werden soll – Java übergibt alle Parameter als reine Input-Parameter „by value" – und auch noch, welchen Datentyp der jeweilige Parameter hat. „As string" bedeutet dabei, dass es sich um ein Datenfeld vom Java-Datentyp „string" handelt. Eine Beschreibung der Parameter in der Linkage-Section kann Cobol natürlich wie bisher verarbeiten, ist damit aber überflüssig geworden. Um die Konvertierung der Datentypen kümmert sich der Cobol-Compiler beziehungsweise das Cobol-Runtime-System. Ohne dieses kann die Anwendung nicht laufen, denn zwischen Cobol- und Java-Datentypen sind die Unterschiede doch zu groß.

Die oben beschriebene Cobol-Klasse und die Java-Klasse müssen nun noch instanziiert werden. Das erledigt ein kleines Java-Programm (siehe Listing 4).

Wie sich hier zeigt, lässt die Codierung keinerlei Rückschlüsse darauf zu, ob es sich bei den zu instanziierenden Klassen um Java-Klassen oder um Cobol-Klassen handelt. Der Aufruf der Cobol-Methode „CountTo" erfolgt ebenso einfach wie der einer Java-Methode. Die Implementierung dieser Methode in Cobol soll nun ebenfalls näher untersucht werden (siehe Listing 5).

Interessant ist hier die Parameterübergabe aus Java mit „bl.CountTo(10, rcv);", die in Cobol dem Procedure Division Header „procedure division using by value n1 as binary-long, by value callmeback as type ICallback." entspricht. Der Parameter „n1" kommt hier als Java-Datentyp „binary-long" an, ohne jemals in einer Linkage-Section erwähnt worden zu sein. Die internen Cobol-Daten sind in der „Local-Storage"-Section definiert, womit dem „Multi-threaded"-Verhalten der Java-Welt Rechnung getragen wird, denn „Local-Storage-Daten" sind in Cobol „thread-lokal".

Als zweiter Parameter wird „by value callmeback as type IcallBack" übergeben. Dies darf für Cobol als geradezu sensationell bezeichnet werden, denn hier wird ohne Definition in der Linkage-Section ein Parameter übergeben, der als Implementierung des Interface „IcallBack" typisiert wird, also eine Objektreferenz auf die Java-Klasse „JavaOutput" darstellt, die das in Cobol definierte Interface „IcallBack" in Java implementiert.

Zum Abschluss wird noch einmal der Cobol-Code der Methode „CountTo" analysiert. Dabei handelt es sich um ganz gewöhnlichen prozeduralen Cobol-Code, wie er in jedem beliebigen Cobol-Programm existieren könnte. Einzig der Aufruf der Java-Methode SendMessage aus Cobol ist noch eine Bemerkung wert: „invoke callmeback::SendMessage(formatted)".

Da in Cobol der Punkt eine besondere Bedeutung hat, wird auf die Schreibweise „::" ausgewichen. Ansonsten ist die objektorientierte Syntax in Cobol der Java-Syntax so ähnlich geworden, dass sich ein Java-Programmierer ohne Probleme zurechtfinden kann.

Fazit

Das Beispiel zeigt, dass sich Cobol erstaunlich weiterentwickelt hat. Die Ausführung von Cobol-Programmen in der JVM ist heute Realität. Die zur Verfügung gestellte Syntax erlaubt eine komfortable und nahtlose Integration in die Java-Welt. Beide Sprachen können in Visual Cobol for Eclipse gemeinsam programmiert und mit einem Debugger analysiert werden.

Rolf Becking
rolf.becking@microfocus.com

Rolf Becking ist Senior Technical Account Manager bei Micro Focus. Seine Schwerpunkte liegen in den Bereichen „Software-Entwicklung", „Software-Werkzeuge", „Integration verschiedener Programmiersprachen" sowie „Modernisierung und Migration von Anwendungssystemen auf offene Systeme".

```
        public class JavaOutput implements ICallback {
        @Override
        public void SendMessage(String arg0) {
                System.out.println("Java says" +
arg0);
        }
}
```

Listing 2

```
interface-id ICallback.

    method-id SendMessage.
    procedure division using by value msg as string.
    end method.

    end interface.
```

Listing 3

```
        public class JavaCounter
{

        public static void main(String[] args) {

                JavaOutput rcv = new JavaOutput();
                BizLogic bl = new BizLogic();
                bl.CountTo(10, rcv);


        }

}
```

Listing 4

```
        class-id BizLogic.
    working-storage section.

    method-id CountTo.
    local-storage section.
    01  i          binary-long.
    01  formatted  pic zzz9.

    procedure division using by value n1 as binary-long,
                        by value callmeback as type
ICallback.
      perform varying i from 1 by 1 until i > n1
        move i to formatted
        invoke callmeback::SendMessage(formatted)
      end-perform
    end method.

    end class.
```

Listing 5

Continuous Bugfixing in großen Projekten

Jürgen Nicolai, main {GRUPPE}, Stuttgart

Statische Code-Analyse mit Tools wie FindBugs, PMD oder CheckStyle ist das Mittel der Wahl, um Fehler zu finden, noch bevor der Code ausgeführt wird. Diese Konzepte, die bei mittlerer Projektgröße gut funktionieren, lassen sich aber nicht „1:1" auf Großprojekte übertragen. Dieser Artikel stellt Lösungen vor, wie statische Code-Analyse in einem Großprojekt sinnvoll funktioniert.

Listing 1 zeigt einen Code aus der aktuellen IntelliJ-Version. Obwohl der Code kompiliert wird, enthält er einige versteckte Fehler, die der Java-Compiler nicht anzeigt. So fehlt beispielsweise der Methode eine sprechende JavaDoc-Dokumentation und in Zeile 7 ruft sich die Methode selbst wieder auf. Dieser Teil des Programms friert ein – ein offensichtlicher Fehler.

Aufgabe der statischen Code-Analyse ist es, Fehler zu finden, bevor der Code ausgeführt wird. Im Vergleich zu dynamischen Testverfahren, bei denen der Code ausgeführt wird, ist statische Code-Analyse einfacher umzusetzen und kann ab der ersten Zeile Source-Code erfolgen. Sie ist damit eine gute Ergänzung zu Code-Reviews und dynamischen Testverfahren wie JUNIT.

```
1  package com.intellij.openapi.util;
2  public abstract class LazyInitializer<Value,
   Exc extends Throwable>
3 implements
   ThrowableComputable<Value, Exc> {
4   private Value myValue;
5  public Value compute() throws Exc {
6   if (myValue == null) {
7     myValue = compute();
8   }
9    return myValue;
10 }
```

Listing 1

Statische Code-Analyse mit Java

Im Java-Umfeld gibt es eine Reihe guter Open-Source-Werkzeuge wie FindBugs [1], PMD [2] oder CheckStyle [3]. Wenn man die Werkzeuge genauer betrachtet, wird man feststellen, dass sie sich in der Arbeitsweise und vor allem bei der Menge und Art der gefundenen Fehler unterscheiden. So findet FindBugs keinen undokumentierten Code, während CheckStyle in diesem Bereich seine Stärken ausspielt. Umgekehrt zeigt FindBugs Nullpointer-Fehler an, die CheckStyle nicht erkennt.

Tabelle 1 fasst die Eigenschaften der gängigen Werkzeuge zusammen. Der Einsatz der genannten Tools erscheint auf den ersten Blick nicht sehr schwierig, aber in der Praxis tauchen in großen Projekten Probleme auf, die gelöst werden müssen.

Vollständigkeit versus Übersicht

Da sich die Werkzeuge ergänzen, sollte man für eine vollständige statische Code-Analyse eine Kombination aus verschiedenen Tools einsetzen. Mit einem derartigen „Werkzeug-Zoo" werden erstaunlich viele Fehler gefunden. Das erste Problem, das zu bedenken ist, betrifft das Regelwerk (auch „rule-set" genannt), mit dem die Analysen erfolgen. Für jedes Werkzeug ist zu entscheiden, ob eine Regel im Projekt einen Fehler darstellt oder nicht. In der Praxis sollte man nicht die „default-rule-sets" der Tools verwenden, da diese hinsichtlich Menge und nicht in Bezug auf Qualität optimiert sind.

Ein zweites Problem: Jedes Werkzeug erzeugt seinen Report, die Ergebnisse der statischen Code-Analyse sind daher auf mehrere Dokumente verteilt. Entwickler sollten diese Auswertungen täglich analysieren, um Fehler früh zu beheben. Auch die Integration der Tools in die Entwicklungsumgebung ändert nichts an der Tatsache, dass jedes Tool über einen Menüpunkt aktiviert werden muss. Wächst die Code-Basis des Projekts, ist irgendwann der Punkt erreicht, an dem es zu einer Informationsüberflutung der Entwickler kommt, wenn mehrere Werkzeuge im Einsatz sind. Die Folge ist, dass die Tools täglich Ergebnisse produzieren, die Entwickler jedoch gefundene Fehler nicht konsequent und zeitnah verfolgen. Eine Lösung könnte ein Qualitäts-Team darstellen, dessen Aufgabe es ist, alle neuen Feh-

Tool	Basis	Anzahl Regeln (ca.)	Eclipse ANT/ MAVEN	Eigene Regeln	Eigene Reports	Anpassung Falschmeldungen
PMD	Source Code	300	Ja	Ja, mit XPath oder Java	Ja, XSLT	Kommentare//NOPMD Annotations@SuppressWarnings („PMD") externe XML-Konfiguration
FindBugs	Binär Code (.class Files)	400	Ja	Eingeschränkt: sehr komplex	Ja, XSLT	Annotationseigene@SuppressWarnings XML-Konfiguration
Checkstyle	Source Code	130	Ja	Ja, mit Java	Ja, XSLT	Kommentare // CHECKSTYLE:OFF XML-Konfiguration

Tabelle 1: Eigenschaften der Open-Source-Werkzeuge zur statischen Code-Analyse

ler in Zusammenarbeit mit den Entwicklern zu analysieren. Effizienter ist es, den Entwicklern die Ergebnisse der statischen Code-Analyse passend zu präsentieren und sie zu motivieren, neue Fehler schnell zu bewerten und zu beheben.

Ein Hauptproblem der statischen Code-Analyse sind sogenannte „Falschmeldungen" (false-positives). Die Werkzeuge zeigen scheinbare Fehler an, die tatsächlich aber keine sind. Der Entwickler muss dann entscheiden, ob er den Code ändert oder den Fehler ignoriert. Falls es sich um keine echten Fehler handelt, tauchen diese bei der nächsten Analyse wieder auf und generieren unnötige Aufwände für die Entwickler.

Zur Lösung dieses Problems bieten alle Werkzeuge die Möglichkeit, Codestellen oder ganze Klassen von der Analyse auszuschließen. Leider existiert dazu kein einheitliches Verfahren, sodass der Entwickler etwa wissen muss, dass bei einem PMD-Fehler der Kommentar „//NOPMD" anzugeben ist.

Tabelle 1 zeigt in der dritten Spalte die unterschiedlichen Mechanismen. In einem großen Projekt, das mehr als ein Werkzeug für die Analyse verwendet, ist diese Art der Kennzeichnung jedoch nicht praktikabel.

Fehlerflut bei bestehenden Projekten

Ist bei einem bestehenden Projekt die statische Code-Analyse eingeführt, tauchen erfahrungsgemäß sehr viele Fehler auf. Es entstehen ungeplante Aufwände oder die Entwickler resignieren möglicherweise vor der schieren Menge an Unregelmäßigkeiten. Einige Werkzeuge bieten die Möglichkeit der Historisierung. Jeder Fehler erhält einen Zeitstempel. Damit wird es möglich, neu hinzukommende Unregelmäßigkeiten von den Altlasten zu unterscheiden. Dies entschärft das Problem bei Altsystemen, weil man sich auf die neuen Wartungsfehler konzentrieren kann. Besser ist es natürlich, die statische Code-Analyse zu Beginn eines Projekts einzuführen.

In Drei-Mann-Projekten mögen die beschriebenen Probleme noch tolerierbar sein, aber ab einer gewissen Größe sind sie nicht mehr in den Griff zu bekommen. Deshalb sollte man in Großprojekten von einem naiven Einsatz der Werkzeuge „out of the box" absehen.

Wege aus dem Werkzeug-Zoo

Um den Konflikt „Vollständigkeit versus Übersicht" aufzulösen, ist zwar ein „Werkzeug-Zoo" erforderlich, die Ergebnisse müssen jedoch für die Entwickler über eine Nachbearbeitung transparent sein. Dazu werden die Ergebnisse der statischen Code-Analyse mit sogenannten „Aggregatoren" zu einem einzigen Report zusammengefasst. Werkzeuge hierfür sind SONAR [4], health4j [5] oder eingeschränkt auch HUDSON [6] und Jenkins [7]. Abbildung 1 zeigt, wie das Analysis-Collector-Plug-in von Jenkins die Ergebnisse der Werkzeuge zusammenfasst.

Falschmeldungen ausschließen

In der Praxis ist ein einheitlicher Mechanismus erforderlich, um die Fundstellen einmal identifizierter „false positives" zu kennzeichnen, sodass diese in zukünftigen Analysen ignoriert werden. Hier gibt es ohne zusätzliche Tools keine befriedigende Lösung. Deshalb bieten SONAR und health4j einen einheitlichen Kommentar oder Annotationen, um Falschmeldungen für alle verwendeten Tools zu kennzeichnen.

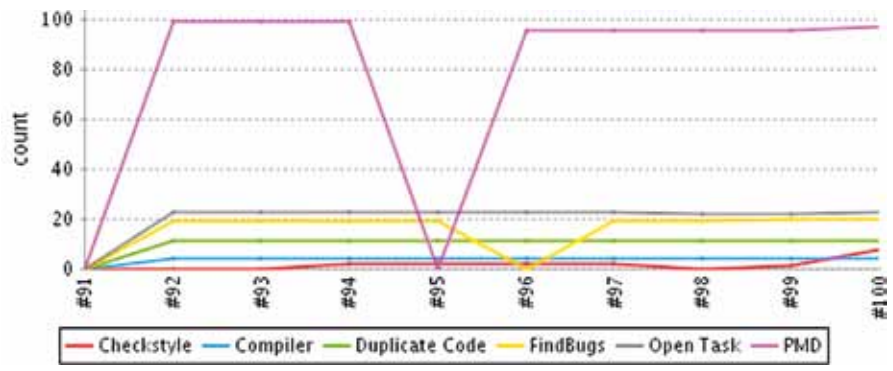


Abbildung 1: Zusammenfassung der Ergebnisse mehrerer Tools mit Jenkins

Der Fehlerflut entgegenwirken

Wer das Projekt „IntelliJ" mit dem Standard-Rule-Set von FindBugs analysiert, bekommt mehr als 2.000 Fehler. Diese Zahl ist beeindruckend, aber auch demotivierend. Niemand kann sich alle diese Unregelmäßigkeiten anschauen. Deshalb gilt: Man beginnt mit einem kleinen, mit den Entwicklern abgestimmten Rule Set, in dem nur schwerwiegende Fehler enthalten sind. Die Menge der Fehler und auch die Aufwände für die Behebung bleiben damit auf einem beherrschbaren Niveau. Dieses minimale Rule Set kann später sukzessive erweitert werden, wenn die statische Code-Analyse etabliert und von den Entwicklern in den täglichen Entwicklungsablauf integriert ist.

Viele Tools zur statischen Codeanalyse liefern sogenannte „Messwerte", die Eigenschaften des Systems beschreiben. Diese Zahlen heißen auch „Software-Metrik". Eine bekannte Metrik, die Auskunft über die Größe und das Wachstum eines Systems gibt, ist beispielsweise „Lines of Code".

Vor lauter Zahlenmaterial und schön bebilderten Auswertungen wird oft vergessen, die Prozesse zur Code-Analyse so zu gestalten, dass die beteiligten Entwickler auf Dauer motiviert werden, das System kontinuierlich zu verbessern. Folgende Punkte sind dabei zu bedenken:

- Immer die Entwickler mit ins Boot holen. Nur wenn sie den Nutzen erkennen und die Auswertungen nicht als „Kontrolle" missverstehen, bringt statische Code-Analyse einen langfristigen Nutzen.
- Die statische Code-Analyse sollte vollautomatisch beim Build erstellt werden. Eine konsequente Ausführung durch die Entwickler kann nicht erzwungen werden.
- Die Anzeige der problematischen Stellen sollte bereits in der Entwicklungsumgebung erfolgen.
- Das Aufzeigen von Unterschieden und Trends ist motivierender als absolute Zahlen: Die Aussage „Wir haben immer noch 2.000 Problemstellen" wirkt demotivierend. Besser sind Auswertungen, die Trends aufzeigen und den Entwicklern ein positives Feedback vermitteln, etwa wenn Fehler behoben wurden.
- Thema „Personalisierung": Die Auswertungen sollten gezielt Fehler aus den

Bereichen anzeigen, die die Entwickler zu verantworten haben.

Ein Großprojekt in der Praxis

Das Software-System besteht aus einem Web- und einem Fat-Client-Teil, dem eine gemeinsame Basis-Architektur zugrunde liegt. An jedem Subsystem arbeiten etwa zehn Teams. Das System wird produktiv betrieben und parallel weiterentwickelt. Hinzu kommen noch Hilfssysteme, die ebenfalls produktiven Charakter haben und mittels statischer Code-Analyse untersucht werden sollen. Abbildung 2 zeigt die Zusammenhänge. Jedes Teilsystem beinhaltet etwa 20.000 Klassen mit zwei Millionen Lines of Code. Die Systeme basieren auf einer OSGi-Architektur, werden über einen automatischen Build-Prozess gebaut und über automatische Tests qualitätsgesichert. Das System ist modular aufgebaut und jedes Team ist für bestimmte Subsysteme zuständig.

Wie bereits beschrieben, verbietet sich der unmittelbare Einsatz des „Werkzeug-Zoos" in derart großen Projekten quasi von selbst. Weitere Anforderungen an die statische Code-Analyse waren:

- Verwendung der führenden Open-Source-Tools zur statischen Code-Analyse
- Historisierung aller Daten, Anzeigen von Veränderungen und Trends
- Möglichst wenige Falschmeldungen
- Integration in die Entwicklungsumgebung und den Build
- Gute Performance: Das Gesamtsystem sollte innerhalb weniger Stunden analysiert werden können
- Personalisierung: Die Entwickler sollten nur die Codestellen prüfen müssen, für die sie auch verantwortlich sind. Darüber hinaus sollten aber auch Gesamt-Sichten zur Code-Qualität möglich sein

Das Problem der Informationsüberflutung durch zu umfangreiche Auswertungen war der Hauptgrund, weshalb sich das Team für eine Eigenentwicklung mit dem Namen „health4j" entschloss und nicht auf Tools wie SONAR zurückgriff. health4j ruft alle gängigen Open-Source-Werkzeuge zur Code-Analyse „out of the box" auf, kombiniert und veredelt die Ergebnisse. Dabei sind alle Auswertungen so personalisiert, dass die Entwickler nur die Analysen ihrer Teilsysteme sehen.

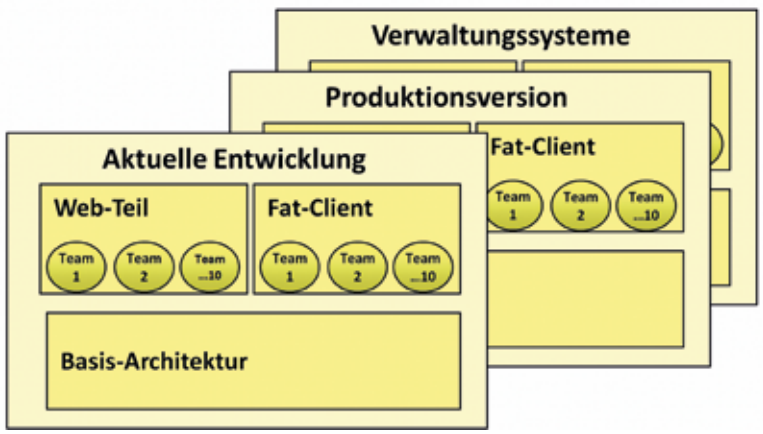


Abbildung 2: Grundlegende Architektur eines Großprojekts

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: JfreeChart
Bundle-SymbolicName: JfreeChart
Bundle-Version: 1.0.0.qualifier
CodeAssignment: org.jfree.chart.*:={scope=chart, team=chartteam}, org.jfree.data.*:={scope=data,
team=datateam}
```

Listing 2: Die „MANIFEST.MF"-Datei mit health4j-Einträgen zur Owner-Zuordnung

Die Waffe gegen riesige Auswertungen ist eine Personalisierungs-Funktion, die die große Informationsmenge des Gesamtsystems intelligent verdichtet. Die Entwickler sollen ihren Code erkennen und ohne große Mühe schnell zur Tat schreiten können. health4j zerlegt die Ergebnisse gemäß vorgegebenen Regeln und erzeugt sogenannte „Sichten" auf das Gesamtsystem. Dabei wäre „Sichten pro Entwickler" ein Ansatz gewesen, der bei einem lang laufenden Projekt jedoch problematisch ist, da sich die Zusammensetzung der Teams relativ häufig ändert. Es ist daher sinnvoller, die Sichten auf Basis der Organisationseinheiten festzulegen. Die Angabe einer „Code-Ownership" definiert, welchem Team der jeweilige Code bei der statischen Code-Analyse zugeordnet werden soll. Technisch wird dies über Angaben in einer externen Datei, eine Code-Annotation oder über die im Java-Umfeld übliche „MANIFEST.MF"-Datei gelöst. Listing 2 zeigt, wie dabei das Code-Owner-Konzept umgesetzt ist (siehe Listing 2).

Die Angabe „Code-Assignment :org.jfree.data.*::{scope:jfree, team:chartteam}" ordnet alle Klassen des „jar"-Moduls, die im Paket „org.jfree.data.*" liegen, dem Team mit dem Namen „datateam" zu. Die Zuweisung „scope=data" legt fest, welchem fachlichen Teil dieser Code zuzuordnen ist. Auf Basis dieser Informationen lassen sich Qualitätsaussagen auf der Team-Ebene und einer fachlichen Ebene treffen. Neben der Aussage „Im Code des Chart-Teams finden sich keine schweren Fehler mehr" können auch Auswertungen über das „Data-Subsystem" gemacht werden, an dem mehrere Teams arbeiten könnten. Das Owner-Konzept erlaubt es also, Team-spezifische Reports zu erstellen, die den Qualitätsverlauf des Team-Codes visualisieren. Abbildung 3 zeigt die Aufteilung eines Gesamtsystems auf Basis von Organisationseinheiten.

Für „chart-team" und „data-team" stellt health4j vollkommen getrennte Auswertungen zur Verfügung. Das schließt auch den JUNIT-Report oder die UML-Diagramme ein. Die Teams haben Zugriff auf JUNIT-Auswertungen, die nur die eigenen Klassen umfassen. Die Teams bewegen sich also in ihrem kleinen Bereich und sind von den restlichen zwei Millionen Codezeilen abgeschirmt.

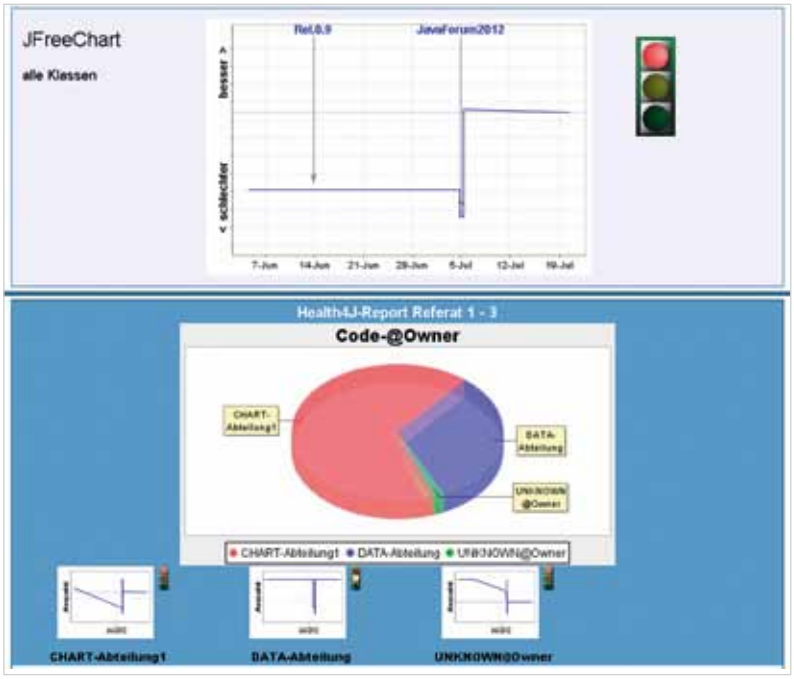


Abbildung 3: Aufteilung eines Gesamtreports durch health4j in zwei Team-Reports

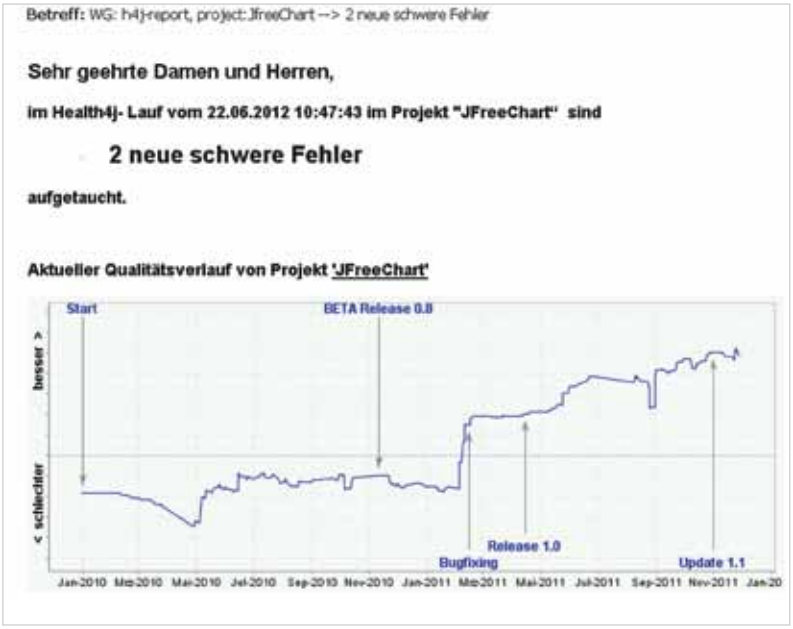


Abbildung 4: E-Mails sollen motivieren, sich die Analyse-Ergebnisse anzuschauen

Falschmeldungen und E-Mail-Benachrichtigungen

Jede Falschmeldung ist ein Ärgernis und mindert die Akzeptanz der statischen Code-Analyse. Aus diesem Grund bietet health4j eine transparente Kennzeichnung von Falschmeldungen über Annotationen oder Kommentare an. Weiterhin kann die Kennzeichnung intelligent automatisiert werden: So führen bestimmte Muster wie der Begriff „ignore" in einem Kommentar in der Umgebung von Fehlern automatisch zu einer Kennzeichnung als Falschmeldung. Durch diese Automatismen lassen sich Hunderte Falschmeldungen von vornherein eliminieren.

Leider ist es im echten, stressigen Entwicklerleben nicht immer so, dass man täglich in aller Ruhe die Meldungen der statischen Code-Analyse betrachtet. Aus diesem Grund versendet health4j E-Mails, wenn im Team-Report neue Unregelmäßigkeiten auftauchen. Auch hier ist es wichtig, die Beteiligten nicht zu überfordern: Die Mails gehen nur an die Code-Owner und werden nur bei offensichtlichen Fehlern versendet (siehe Abbildung 4).

Qualität der Analyse und Ergebnispräsentation

Die Qualität der Analyse lässt sich durch Kombination von Fehlermeldungen und Metriken deutlich verbessern. Beispiele sind:

- Undokumentierter Code wird nur angezeigt, wenn die Funktionen einen gewissen Umfang und Komplexität aufweisen
- Code-Duplikate werden nur angezeigt, wenn sie im gleichen Modul auftreten
- Bei generiertem Code werden bestimmte Regeln automatisch ausgeblendet
- Fehler, die von mehreren Tools gefunden werden, werden nur einmal angezeigt
- JUNIT-Testcode ist von der Analyse ausgeschlossen

Die Ergebnisse sind so präsentiert, dass sich die Qualität des Systems schnell erfassen lässt. Aus diesem Grund wurde von umfangreichen Metriken Abstand genommen. Eine einfache Ampel-Metapher für die Bereiche „Fehler", „Stil", „Dokumentation" und „Offene Punkte" genügt, um einen Handlungsbedarf zu erkennen. Abbildung 5 zeigt einen typischen Übersichtsreport. Besonders wichtig sind der Qualitätsverlauf und die Historisierung der Daten: Die Beteiligten erhalten so schnell positive Rückmeldung, wenn sie die Empfehlungen der statischen Code-Analyse beachten.

Fazit

Statische Code-Analyse ist ein wichtiges und vor allem günstiges Verfahren, um die Code-Qualität zu verbessern. Die verfügbaren Open-Source-Werkzeuge sind ausgereift und finden erstaunlich viele Fehler im Code, ohne dass das System gestartet werden muss. Um statische Code-Analyse in einem großen Projekt erfolgreich einzuführen, bedarf es allerdings einiger Überlegungen. Die reine Tool-Auswahl genügt nicht. Entscheidend sind gut durchdachte

Prozesse rund um die statische Code-Analyse und vor allem das Commitment der Entwickler.

Weiterführende Links

[1] FindBugs: http://findbugs.sourceforge.net/
[2] PMD: http://pmd.sourceforge.net/
[3] CheckStyle: http://checkstyle.sourceforge.net/
[4] SONAR: http://www.sonarsource.org
[5] health4j: http://www.health4j.de
[6] HUDSON: http://hudson-ci.org
[5] JENKINS: http://jenkins-ci.org/

Jürgen Nicolai
j.nicolai@main-gruppe.de

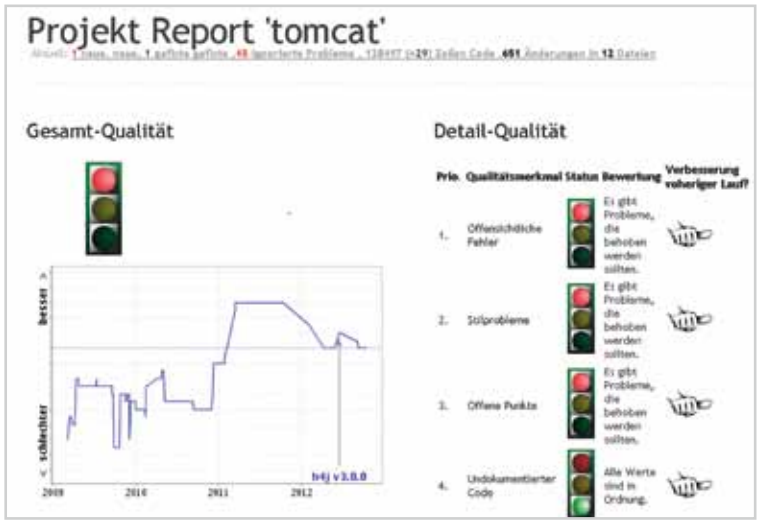


Abbildung 5: Die health4j-Startseite lässt Qualität auf einen Blick erkennen

Jürgen Nicolai ist Geschäftsführer der main {GRUPPE} mit Sitz in Stuttgart. Er ist seit über 15 Jahren im Java-Umfeld als Trainer und Coach tätig. Die main {GRUPPE} unterstützt ihre Kunden mit dem Produkt health4j bei der statischen Code-Analyse sowie beim Test- und Qualitätsmanagement.

„Wir sollten das Oracle-Bashing unterlassen …“

Usergroups bieten vielfältige Möglichkeiten zum Erfahrungsaustausch und zur Wissensvermittlung unter den Java-Entwicklern. Sie sind aber auch ein wichtiges Sprachrohr in der Community und gegenüber Oracle. Wolfgang Taschner, Chefredakteur von Java aktuell, sprach darüber mit Falk Hartmann von der Java User Group Saxony.

Wie bist du zur JUG Saxony gekommen?
Hartmann: Im Jahre 2007 hatte Torsten Rentsch die Idee zur Gründung einer JUG in Dresden. Ich erfuhr über Umwege davon und meldete mich als Mitstreiter. So bin ich von Anfang an dabei.

Wie ist die JUG Saxony organisiert?
Hartmann: Im rechtlichen Sinne sind wir leider noch „unorganisiert“, also eine Gemeinschaft aus einer Handvoll Enthusiasten. Das funktioniert nur, da wir auch an den Enthusiasmus der Sprecher appellieren, erlaubt uns aber andererseits, ohne Mitgliedsbeiträge oder Eintritt auszukommen.

Was zeichnet die JUG Saxony aus?
Hartmann: Ich kann nicht sagen, ob wir uns essenziell von anderen JUGs unterscheiden, aber unsere Besucher sind ein gut vernetztes, wissbegieriges Völkchen. Wir erleben ständig, dass die Veranstaltungen genutzt werden, um auch alte Kollegen und Freunde zu treffen. Zudem bleibt kein Thema undiskutiert; es gibt immer jemanden, der qualifizierte Fragen stellt – zur Freude und manchmal auch zum Leidwesen der Referenten.

Wie viele Veranstaltungen gibt es pro Jahr?
Hartmann: Meist sind es um die zehn Veranstaltungen, die in der Regel donnerstags stattfinden. Manchmal klemmt es etwas bei der Veranstaltungsplanung und wir nehmen spontan Vorträge ins Programm. Die Veranstaltungen finden in der Regel in Dresden, in letzter Zeit aber auch häufiger in Leipzig und ab und zu in Chemnitz statt.

Was motiviert dich besonders, bei der JUG Saxony mitzuarbeiten?
Hartmann: Das sind ganz verschiedene Dinge. Zum einen sind die Macher der JUG Saxony ein eingeschworenes Team, in dem man sich aufeinander verlassen kann, wenn man mal wegen anderweitiger Verpflichtungen mit den Aufgaben in der JUG nicht klarkommt. Andererseits sind auch für mich die Veranstaltungen der JUG nicht nur Weiterbildung, sondern auch eine Möglichkeit, Freunde und ehemalige Arbeitskollegen zu treffen. Ab und zu zeigen wir auch mal einem Referenten die Stadt, wobei sich immer sehr interessante Gespräche ergeben.

Was bedeutet Java für dich?
Hartmann: Java ist mein tägliches Brot. Ich schaue mir auch andere Sprachen an, aber nach fünfzehn Jahren mit Java bin ich in dieser Sprache am fittesten.

Welchen Stellenwert besitzt die Java-Community für dich?
Hartmann: Die Community ist für mich in erster Linie eine Möglichkeit, die aktuellen Trends zu verfolgen. Meist kann man erfolgversprechende Ansätze schon zeitig kennenlernen, wenn sich die Vordenker der Community gegenseitig darauf hinweisen – oft Monate, bevor die ersten mehrseitigen Artikel in den Java-Zeitschriften stehen.

Zur Person: Falk Hartmann
Falk Hartmann arbeitet seit 15 Jahren mit Java. Seine Themenschwerpunkte sind OSGi, XML und Security. Er arbeitet als Principal Software Engineer bei Demandware in Jena.

Was hast du bei der Übernahme von Sun durch Oracle empfunden?
Hartmann: Ich war natürlich gespannt und ein bisschen besorgt, wie es weitergeht. Der Start war dann ja auch sehr holprig, einige gute Leute aus dem Java-Umfeld wechselten den Arbeitgeber, hinzu kam der Streit um Hudson. Mittlerweile sehe ich die Sache gelassen. Es geht voran – wenn auch nicht so schnell, wie wir uns das vielleicht wünschen.

Wie sollte sich Java weiterentwickeln?
Hartmann: Ich wünsche mir immer noch einen echten Komponenten-Ansatz in Java – mein Favorit wäre OSGi, auch wenn das wohl nichts werden wird. Auch die Modularisierung des JRE selbst steht für mich weit oben auf der Wunschliste, gerade weil ich längere Zeit für stark Ressourcenbeschränkte Systeme entwickelt habe, auf denen die Hälfte der Packages unbenutzt den knappen Speicherplatz belegte.

Wie sollte sich die Community gegenüber Oracle verhalten?
Hartmann: Ein bisschen Offenheit gegenüber Oracle wäre meiner Meinung nach angebracht. Oracle tut einiges für Java und die Community, wir haben schon einige Sprecher von Oracle bei uns gehabt. Meiner Meinung nach sollten wir das Oracle-Bashing unterlassen – Oracle ist eine Firma wie viele andere, sehr groß und sicher nicht so schnell wie eine junge Firma; auf der anderen Seite ist es aber auch eine Firma mit exzellenten Mitarbeitern und vielen Ideen.

Falk Hartmann
falk.hartmann@jugsaxony.org
http://www.jugsaxony.org

Pragmatisches Testen mit System

Martin Böhm, SALT Solutions GmbH

Mit jedem Software-Entwicklungsprojekt stehen Team und Projekt-Stakeholder vor der Herausforderung, innerhalb eines festgelegten Budgets anforderungsgerechte und praktikable Tests zu entwerfen. Insbesondere in kleinen Projekten wird zwar der Test-Gedanke postuliert, aber der Weg zu passenden Tests ist in wachsenden Projekten oft steiniger als geplant: Die Tests schlagen fehl, die Entwicklung verzögert sich, Meilensteine werden gerissen. Der Artikel zeigt Wege auf, wie man systematisch, aber unkompliziert zu projektadäquaten Tests gelangt und die Test-Verantwortung gezielt verteilen kann.

„Die Wünsche und Anforderungen des Kunden stehen an erster Stelle. Wer dem Kunden nicht folgt, verschwindet vom Markt." So oder ähnlich lässt sich die Motivationslogik in vielen Projekten beschreiben. Die Anforderungen des Kunden schnell in Projekte zu überführen und umzusetzen, führt allerdings immer wieder zu Frustration, wenn Wunsch und Wirklichkeit aufeinandertreffen. Spätestens dann, wenn bereits große Schritte im Projekt gemacht wurden und der erste Liefertermin naht, stellt sich die Frage: Funktioniert alles so, wie sich der Kunde es wünscht?

Typische Testprobleme

Gerade in den heißen Projekt-Phasen ist guter Rat teuer. Wer sich erst jetzt mit dem Thema „Test" beschäftigt, wird mit „hippen" Test-Reizwörtern überschüttet und von den ultimativen Test-Werkzeugen überfordert. Man tritt langsam, aber sicher in die Spirale typischer Testprobleme ein:

- Der Test wird am Anfang des Projekts zwar postuliert, aber keiner weiß, wie man es richtig macht
- Komplexität und Vorbereitungsdauer für die Test-Aktivitäten werden unterschätzt
- Aufgrund fehlenden Test-Know-hows wird Testen auf einfache, funktionale Aspekte reduziert: „Anwendung an! Funktioniert! Fertig!"
- Es werden hektisch neue Test-Werkzeuge eingesetzt, deren Zweck nur schwer durchschaubar ist
- Es wird etwas Falsches getestet, weil die Anzahl oder der Umfang der Test-Fälle unpassend sind
- Es werden Tester abgestellt, die jedoch schnell überlastet sind, weil die Spezifikation und Durchführung nachträglicher Tests sehr aufwändig ist
- Der Test wird als Kostentreiber empfunden, weil man wirklich schwere und teure Fehler trotzdem zu spät findet
- Testen wird auf die notwendigsten Aktivitäten reduziert, das Test-Management abgeschafft
- Die Folgen unzureichender Tests beziehungsweise unzureichenden Testmanagements werden nicht benannt
- Der Sinn des Testens ist spätestens dann in Frage gestellt, wenn folgende Äußerungen im Projekt kursieren: „Die entscheidenden Fehler findet man sowieso nur im Live-Betrieb!"

Wie kann es dazu kommen? Die Ursache liegt in der noch weit verbreiteten Phasen-Denkweise vieler Projekte. Dabei wird Testen häufig noch im Sinne des Wasserfall-Modells als reine Prüftätigkeit verstanden. Folglich muss der Test immer am Ende der Entwicklung stattfinden.

Test ist aber nicht nur Prüftätigkeit, sondern erfordert ein hohes Maß an Planung der einzelnen Test-Aktivitäten, den Abgleich der Anforderungen mit den Test-Fällen und schließlich auch die Abstimmung der Zeit- und Ressourcen-Pläne mit allen Projekt-Beteiligten. Die Voraussetzungen für projektadäquates Testen werden also bereits in den frühen Phasen eines Projekts geschaffen. Aufgrund der anfänglichen Projekt-Euphorie und des zu Beginn noch etwas verschwommenen Projekt-Scopes werden die frühen Test-Aktivitäten jedoch häufig vergessen.

Falsche Annahmen über Tests

Um sich diesem Problem weiter zu nähern, sind Fehlannahmen über das Testen zu widerlegen. Dazu greift der Autor die Argumentations-Grundlagen dreier typischer Fehlannahmen auf, die man anschließend gezielt aus dem eigenen Projekt-Kontext beliebig zerlegen und widerlegen kann:

- *„Testen ist Selbstzweck und kein Kernziel des Projekts, deshalb will der Kunde es nicht in Rechnung gestellt haben."*
  Argumentationsgrundlage der Fehlannahme: Eine mangelnde Test-Strategie und mangelnde Test-Planung führen zu mangelnden Anforderungen an die Test-Umgebung und die Test-Aktivitäten. Dadurch werden unzureichende Tests entwickelt und die Test-Ziele verfehlt. Um diese zu kompensieren, erhöht man den Zeit- und Ressourcen-Aufwand, wodurch die Test-Effizienz sinkt. Der Projekt-Erfolg ist dadurch gefährdet, denn die Kosten der Test-Aktivitäten steigen. Der Versuch, den Kunden bei aufwandsbudgetierten Projekten an den Kosten zu beteiligen, führt aber dazu, dass der Kunde den aus seiner Sicht überflüssigen Aufwandsposten „Test" wegzudrücken versucht.
- *„Je mehr Tests wir durchführen, desto mehr Fehler spüren wir auf."*
  Argumentationsgrundlage der Fehlannahme: Die Vermutung, dass Fehler gerade dort lauern, wo bereits Fehler gefunden wurden, ist zwar prinzipiell richtig. Allerdings wird bei dieser Argumentation vergessen, dass auch alle Test-Aktivitäten unter Kosten-Nutzen-Aspekten durchgeführt werden müs-

sen. Weil die Nicht-Existenz von Fehlern selbst unter größtem Test-Aufwand nicht bewiesen werden kann, würde die Argumentation bedeuten, die Fehlerfindungs-Kosten ins Unermessliche steigen zu lassen.

- *„Mit dem richtigen Test-Werkzeug und einer geeigneten Test-Methode ist Testen ein Kinderspiel."*
  Argumentationsgrundlage der Fehlannahme: Auch hier werden falsche Annahmen über die Fehlerfindungs-Quoten getroffen. Häufig sind sich die Projektbeteiligten über das einzige und richtige Werkzeug uneinig, und insbesondere fehlendes Test-Know-how im Sinne von „A fool with a tool is still a fool" führt nicht selten zur Überbewertung von Test-Techniken oder gar zum Missbrauch von Test-Werkzeugen, um letztendlich den beabsichtigten Return-on-Investment zu erreichen.

Was Qualität und Testen bedeutet

Nach der Norm DIN EN ISO 9000 definiert sich Qualität als die Eignung eines Produkts oder einer Dienstleistung, bestimmte Anforderungen des Kunden zu erfüllen. Um die Qualität eines Produkts oder einer Dienstleistung zu bestimmen, setzt man Tests als Mess-Methode ein. Damit wird deutlich, dass Testen keine Fehlerfindungs-Methode ist, sondern eine Mittel zur Beantwortung unserer eingangs erwähnten Frage: „Akzeptiert der Kunde das gelieferte Produkt?".

Das Erreichen von Qualitätszielen impliziert also immer das Testen. Dazu folgende Beispiel-Situation: Die Anwendung „X" soll am Tag „Y" bereitgestellt werden. Bisher hat sich niemand im Projekt über den Test und die Bereitstellung des Produkts beim Kunden Gedanken gemacht. Eine Woche vor dem Termin fragt der Projektleiter sein Team: „Können wir das Produkt ausliefern?" Der Build-Manager antwortet: „Ja, ich habe die Anwendung gestartet und sie läuft." Ein Entwickler antwortet hingegen: „Nein, in Schnittstelle ‚A' fehlt ein Parameter ‚S'."

Es zeigt sich, dass beide Aussagen auf unterschiedlichen Messungen basieren, und zwar im Beispiel einmal auf der Messung des Funktionsmerkmals „Anwendungsstart" und zum anderen auf der des Funktionsmerkmals „Schnittstelle ‚A' mit Parameter ‚S'", die den Anforderungen des Entwicklers nicht gerecht wird.

Das Fazit daraus: Implizit testet man immer, aber möglicherweise nicht das Richtige und nicht zum richtigen Zeitpunkt, wenn man Qualität als Gesamtheit der Anforderungen an ein Produkt betrachtet. Was wird damit noch deutlich? Die Bewertung der Qualität eines Produkts kann aus unterschiedlichen Blickwinkeln erfolgen. Beim Testen muss das berücksichtigt werden.

Große und kleine Projekte, Module und Zwischen-Produkte

Nach all der Theorie zurück zum Projektgeschäft. Software-Entwicklungsprojekte könnten nicht unterschiedlicher sein, sodass sie sich darüber hinaus auch noch in ihrem zeitlichen Verlauf ständig verändern. Während am Anfang eher kleine und allgemeine Anforderungen stehen, wächst die Menge der sich konkretisierenden Anforderungen mit dem Projekt-Fortschritt an. Während am Anfang eher anforderungsgetriebene Tests in Form von Unit-Tests oder auf Basis des FIT-Ansatzes entstehen, erfordern große Projekte einen eher systematischen Testansatz, um die schiere Menge an gesetzten Anforderungen mit der tatsächlich bereitgestellten Software gezielt und systematisch zwischen den Projektdienstleistern abzugleichen. Dazu gehört es auch, die Test-Ergebnisse zu bewerten und, wie im obigen Beispiel, die Anforderungen eines Entwicklers an eine bestimmte Schnittstelle mit den Gesamtsystem-Anforderungen abzustimmen.

Aufgrund der großen Menge an Anforderungen bietet sich wie bei vielen Lösungsproblemen die Zerlegung des Gesamtproblems an, also den Test in übersichtliche Teile des Software-Systems zu zerlegen. Unit-Tests, die nur eine Klasse oder Komponente isoliert prüfen, sind ein konkretes und sehr gutes Beispiel dafür. Eine Zerlegung ist auch deshalb sinnvoll, weil die Qualität des Gesamt-Systems in erheblichem Maß von der Qualität seiner Sub-Systeme und Teil-Komponenten abhängt. Entsprechend dieser System-Subsystem-Komponenten-Metapher bietet sich auch für den Software-Test die Anwendung des Teile-und-Herrsche-Prinzips an, um nicht nur einzelne Units, sondern auch Anwendungsteile sowie Sub-Systeme zu testen und schließlich auch das Software-System in seiner Gesamtheit gegenüber den Anforderungen des Kunden und der Projektbeteiligten wie Entwicklern und Wartungsteam zu vermessen.

ISTQB-Teststufen

Zur Anwendung des Teile-und-Herrsche-Prinzips hat das International Software Testing Qualifications Board (ISTQB) ein Teststufenmodell entwickelt, mit dem die Test-Aktivitäten auf unterschiedliche Teile des Gesamtsystems verteilt werden können (siehe Abbildung 1). Das Modell ähnelt in seiner Darstellung sehr stark dem V-Modell, liefert allerdings eine generische Basis für

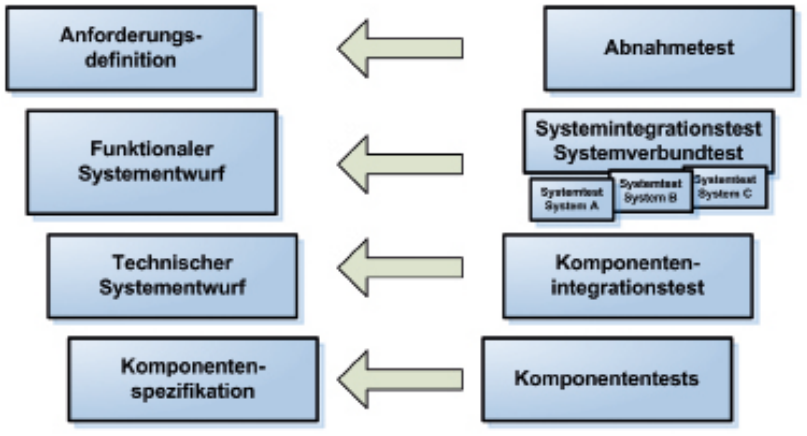


Abbildung 1: Teststufen nach ISTQB

die Einbettung der Test-Aktivitäten in unterschiedlichste Projektvorgehensmodelle.

Die in der Darstellung gezeigten Teststufen stellen eine Form von „Quality Gates“ dar, die sicherstellen sollen, dass eine Teststufe erst dann betreten wird, wenn in der vorherigen Stufe bewiesen wurde, dass die Qualitätsansprüche an Komponente, Teil-System beziehungsweise Software-System erreicht wurden. Darüber hinaus haben die Teststufen folgenden Zweck:

- Die Verantwortung für die Qualität bestimmter Systemteile ist genau abgrenzt; sie bilden ein direktes Mapping der Verantwortung für einzelne Systemteile zu den jeweiligen Projektbeteiligten ab
- Die Test-Verantwortung wird nicht auf eine einzelne Test-Organisation, sondern auf viele Schultern gleichmäßig verteilt
- Durch die klare Verantwortungs-Abgrenzung ist der Umgang mit Tests in Multi-Projekten erheblich leichter
- Mit den Teststufen können einzelne Test-Techniken und -Methoden besser in den Entwicklungsprozess eingeordnet und abgegrenzt werden. Deren Auswahl ist durch die Teststufen bereits grob eingegrenzt
- Die Anwendung von Teststufen resultiert in einem gewissen Zwang zur Qualitätssicherung von Teil-Komponenten

Wichtig ist allerdings, dass die Teststufen keine Test-Methoden oder -Techniken implizieren. Erst nach der Analyse der Anforderungen und Test-Objekte sind Test-Techniken auswählbar. So sind beispielsweise Performance-Tests auf allen Stufen möglich. In der Teststufe „Komponententest“ könnte man für die Prüfung von Performance-Anforderungen Laufzeit-Messungen an Methoden-Aufrufen durchführen, während man in der Teststufe „Abnahmetest“ die Zeit misst, die ein durchschnittlicher Anwender für die Durchführung ausgewählter Anwendungsfälle benötigt. Damit wird deutlich, dass mit den Teststufen nur qualitative Abhängigkeiten manifestiert werden, und zwar im Sinne von: „Sichere erst die Qualität der Einzelteile, bevor die Qualität der Gesamtheit geprüft wird.“

Regressionstests

Die Teststufen können beliebig oft durchlaufen werden. Wurden beispielsweise im Systemtest im System „A“ Fehler erkannt, müssen nach deren Behebung sowohl der Komponententest als auch der Komponenten-Integrationstest erneut durchlaufen werden, um sicherzustellen, dass die Komponenten des Systems „A“ auch weiterhin der Spezifikation und dem technischen System-Entwurf entsprechen. Auf diese Weise wird im Sinne der Kosten-Nutzen-Argumentation sichergestellt, dass erst dann eine Teststufe betreten wird, wenn die dafür notwendige Qualität der Systemteile sichergestellt ist (siehe Abbildung 2 und Tabelle 1).

In der Praxis bedeutet das, dass vor allem in den frühen Phasen des Projekts häufig nur der Komponenten-Test erfolgreich passiert wird, während man aufgrund noch unausgereifter Schnittstellen an den Quality Gates zum Komponenten-Integrationstest oder Systemtest häufig noch scheitert. Mit diesem Vorgehen wird bereits implizit eine simple Möglichkeit der Test-Steuerung genutzt, und zwar der Einsatz von Regressionstests, um rechtzeitig Abweichungen und Fehler in Komponenten und Sub-Systemen festzustellen.

Bestandsaufnahme über Vorgehen, Prozesse und Test-Infrastruktur

Um einen Teststufen-Ansatz einzuführen, sind gar keine großen Schritte notwendig. Häufig reicht bereits eine Analyse der aktuellen Test-Landschaft aus, um einzelne

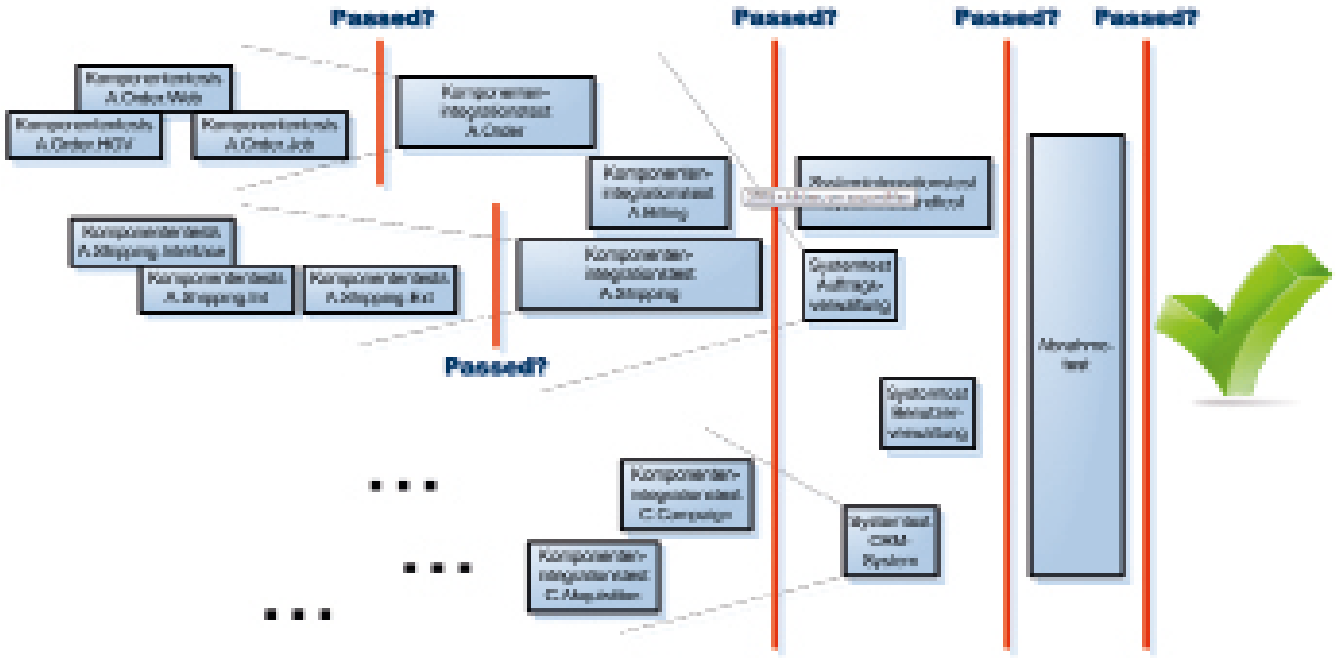


Abbildung 2: Qualitative Abhängigkeiten der Teststufen

Test-Aktivitäten in die Teststufen einordnen zu können und sich ein erstes Bild zu machen. Eine beispielhafte Bestandsaufnahme könnte so aussehen:

- System und Komponenten werden täglich um 16 Uhr gebaut
- Die Entwickler entwickeln Unit-Tests und führen diese lokal aus (Problem: Diese schlagen aber häufig fehl, weil Abhängigkeiten zu einer schlecht gepflegten Datenbank bestehen)
- Automatische Regressionstests werden selten durchgeführt (Problem: Der Build-Manager hat sie häufig deaktiviert)
- A.Shipping hat die meisten Unit-Tests (Problem: Es werden dort allerdings selten Abweichungen detektiert)
- Ein Code-Review wird durchgeführt (Problem: Die Behebung der Abweichungen findet aber nur selten statt)
- Der Kunde testet, sobald die Software aufs Produktivsystem ausgeliefert wurde (Problem: Er findet jedoch kritische Fehler häufig erst nach langer Produktivzeit)

Hier wird deutlich, dass zwar unterschiedliche Test-Techniken und -Methoden eingesetzt werden, diese aber offenbar in keinem qualitativen Zusammenhang zueinander stehen. Erst mit der Verknüpfung der Qualitäts-Aspekte durch Teststufen können für die Prüfung von Komponenten und Gesamtsystem gezielt passende Test-Methoden und -Techniken ermittelt werden, mit denen die Anforderungen auf der jeweiligen Teststufe sichergestellt sind.

Problem-Analyse und Behebung

Im Beispiel sind bereits einige Probleme in Klammern notiert. Für die Feststellung und Behebung bieten sich zahlreiche Methoden an, darunter das auf Test-Prozesse spezialisierte Test Process Improvement (TPI). Allerdings sind viele dieser Methoden sehr aufwändig, weil sie eine gewisse Erfahrung erfordern oder in Form von Assessments einen erheblichen Ressourcen-, Zeit- und Personal-Bedarf verursachen, der in laufenden, zeitkritischen Projekten schwer zu leisten ist.

Mit dem Teststufen-Ansatz ist die Test-Umgebung bereits separiert und die Verantwortung auf unterschiedliche Personen aufgeteilt. Die Problem-Behebung könnte also innerhalb der einzelnen Teststufen von den einzelnen Verantwortlichen selbstständig gelöst werden. Hier bietet sich die Fünf-Warum-Frage-Technik an, weil diese selbst von testunerfahrenen Projektmitgliedern sehr einfach und problemadäquat eingesetzt werden kann. Dabei werden mithilfe naiver Fragetech-

Spezifikation („linker Ast“)	Ergebnisse (i. w. S. Testobjekte)	Teststufe („rechter Ast“)	Ziel	Bedeutung (nach ISTQB Glossar)	Zielgruppe, Entscheider	was neben der Software noch so abfällt…	Beispiele
Anforderungen, User Stories	HW+SW+ (Daten, z. B. aus Migration)	Abnahmetest	„Endgültige Vertrauenshöhe des Kunden in das Produkt bzw. das System messen.“	Formales Testen hinsichtlich der Benutzeranforderungen und -bedürfnisse bzw. der Geschäftsprozesse. Wird durchgeführt, um einem Auftraggeber oder einer bevollmächtigten Instanz die Entscheidung auf der Basis der Abnahmekriterien zu ermöglichen, ob ein System anzunehmen ist oder nicht.	Auftraggeber, Anwender, Produktmanager	Anwendungssystem; Benutzerdokumentation; Verfahrensanweisungen	Durchführung der Geschäftsprozesse am System; Check der Testdurchführungsprotokolle und der HW-Requirements; Revision der Projektdurchführung
Schnittstellen- und Betriebsspezifikation	HW+SW	Systemintegrations-test	„Qualität der Prozessabwicklung mit externen Systemen prüfen.“	Testen der Integration von Systemen und Paketen; Testen der Schnittstellen zu einer externen Organisation; Testen eines Systemverbunds; u. a. auch zur Zertifizierung	Auftraggeber, Anwender, Produktmanager, IT-Betrieb, externe Organisationen	Schnittstellendokumentation; Betriebsspezifikation	Test von Schnittstellen zu externen Systemen; EDI-Schnittstellen; SAP-Schnittstellen; Web-services; Hardware
funktionaler Systementwurf	SW	Systemtest	„Äußere Qualität des Systems messen.“	Testen eines integrierten Systems, um sicherzustellen, dass es spezifizierte Systemanforderungen erfüllt.	Produktmanager, IT-Betrieb, techn. Projektleiter	SW-System mit Installations- und Betriebshandbuch; Installationsskripte; Testtreiber;	Klassischer GUI- Klicktest; Klassischer Performancetest
technischer Systementwurf	SW-Module	Komponentenintegrationstest	„Qualität der Komponenten aus Sicht benachbarter Komponenten messen.“	Testen wird durchgeführt mit dem Ziel, Fehlerzustände in den Schnittstellen und dem Zusammenwirken der integrierten Komponenten aufzudecken.	Entwickler, SW-Architekt	JAR/WAR/EAR; JavaDoc-ZIP; Testtreiber	Pseudounittest (z. B. Unittest mit Persistenz, Unittests über HTTP); EAR-Deploy; Spring- oder OSGi-Container starten
Komponentenspezifikation	SW-Komponenten	Komponententest	„Innere Qualität der Komponente aus technischer Sicht messen.“	Testen einer (einzelnen) Komponente.	Entwickler	Sourcecode inkl. JavaDoc; Entwicklungsrichtlinien, XMLs und Konfigurationsfiles	Isolierter Unittest; Checkstyle; Findbugs

Tabelle 1: Bedeutung der Teststufen

nik Problemursachen analysiert, und zwar durch folgendes exemplarische Frage-Antwort-Spiel:

- Warum werden keine Systemtests entwickelt?

→ Weil wir keine Zeit für die Testfall-Erstellung haben.

- Warum haben wir dafür keine Zeit?

→ Weil der Kunde das nicht bezahlt.

- Warum bezahlt der Kunde nicht?

→ Weil er es nicht beauftragt hat.

- Warum hat es der Kunde nicht beauftragt?

→ Weil er den Zweck der Systemtests nicht verstanden hat.

- Warum hat der Kunde den Zweck nicht verstanden?

→ Weil Systemtests nicht als Teil des Projekts beauftragt wurden.

- Warum hat es der Kunde nicht beauftragt? etc.

Ziel der Fragetechnik ist es, möglichst schnell die wesentlichen Kern-Ursachen für Test-Probleme so einzukreisen, dass der laufende Projektbetrieb so wenig wie möglich gestört wird. In Untersuchungen hat sich nämlich herausgestellt, dass nach etwa fünf Fragen die Kern-Ursachen bereits ausreichend aufgedeckt sind und man sich mit weiteren Fragen quasi im Kreis dreht. Die Lösung der Probleme obliegt dann nicht irgendeinem Testverantwortlichen, sondern genau denjenigen, die die höchste Expertise im Problembereich besitzen (siehe Tabelle 1).

Ad-hoc testen

Eine oft anzutreffende Vorgehensweise sind Ad-hoc-Tests, also Tests, bei denen wie im obigen Beispiel der Build-Manager einen provisorischen Test des Anwendungsstarts ohne Planung und Dokumentation auf Basis eigener Erfahrungen durchführt. Warum sind gerade solche Vorgehensweisen so beliebt?

Betrachtet man beispielsweise den Prozess zur Implementierung von Unit-Tests, stellt man fest, dass diese im Zuge der Implementierung sehr konkret entwickelt werden und durch die Nähe zum Code beziehungsweise zu den Code-Anforderungen der Zyklus „Anforderung → Entwicklung → Test → Test → Ergebnis“ sehr kurz ist. Im Gegensatz dazu ist die Spezifikation von Systemtest-Fällen sehr abstrakt. Der Zeitraum von der Anforderungs-Formulierung bis zum tatsächlichen Test-Feedback ist im Vergleich zu den Komponenten- beziehungsweise Entwickler-Tests eher lang. Aus diesem Dilemma haben sich agile Entwicklungsmodelle wie „Scrum“, „Prototyping“ oder „Extreme Programming“ entwickelt, weil sich kleine Feedback-Zyklen für alle Projektbeteiligten einfach besser anfühlen. Wie kann man aber selbst bei großen und bei langen Entwicklungszyklen sicherstellen, dass das Feedback ausreichend ist, und wie erreicht man in den höheren Teststufen Integrationstest, Systemtest und Abnahmetest einen ähnlichen Effekt? Hier hilft leider nur diszipliniertes Vorgehen:

- Anforderungen konsequent festhalten und pflegen
- Testbarkeit früh sicherstellen und verbessern
- Test-Beteiligte früh einbinden und in den Test-Prozess integrieren

Auch hier sollte man nicht nur an die Tester denken, denn auch Entwickler, Projektleiter, IT-Betrieb und Anwender haben im Sinne der Teststufen die Verantwortung, ihre Test-Aufgaben zu erledigen.

Eine gewisse Unterstützung zur Integration der Teststufen bieten CI-Umgebungen wie „CruiseControl“ oder „Jenkins“ an. So kann man etwa die Teststufen „Komponententest“ und „Integrationstest“ mit Maven realisieren, indem die Maven-Phasen „test“ und „integration-test“ mit entsprechenden Plug-ins bestückt und innerhalb einer CI-Umgebung mit „CruiseControl“ oder „Jenkins“ ausgeführt werden.

Die Stufen „Systemtest“ und „System-Integrationstest“ lassen sich damit genauso realisieren, indem Job-Abhängigkeiten zwischen den einzelnen CI-Projekten eingerichtet sind. Sofern die Tests bestimmter Projekte erfolgreich durchgeführt wurden, lassen sich durch Job-Trigger die entsprechenden System-Integrationstests in den jeweiligen Zeitfenstern anstoßen. Selbst manuelle Tests lassen sich auf diese Weise integrieren. So kann die CI-Umgebung nach erfolgreichem Deployment (auch das erfolgreiche Deployment ist im Sinne der Qualitätssicherung ein Test) eine Mail an die Tester absenden und über den erfolgreichen Abschluss der vorgelagerten Teststufen informieren. Die Tester können dann den manuellen System- oder System-Integrationstest einleiten. Der erfolgreiche Abschluss der Tests kann mit einer Reply-Mail an das CI-System durch den Test-Verantwortlichen quittiert und die nächste Teststufe angetriggert werden.

Fazit

Der Schlüssel zum Test-Erfolg liegt darin, die Wünsche und Anforderungen aller Projektbeteiligten zu berücksichtigen und in den Projektalltag zu integrieren:

- Qualität und Test sind ganzheitlich über Kunde, Anbieter, Projektleitung, Entwicklungs- und Wartungsteam sowie IT-Betrieb zu betrachten
- Den oft ersehnten idealen Tester oder Test-Automaten gibt es nicht
- Selbst ein Test-Outsourcing entbindet die Projektbeteiligten nicht von ihrer Qualitäts- und Test-Verantwortung
- Die Verteilung der Test-Verantwortung auf mehrere Schultern reduziert das Risiko unzureichender Tests auf anderen Teststufen erheblich
- Die Test-Ergebnisse und -Erkenntnisse lassen sich durch Teststufen zwischen den einzelnen Test-Verantwortlichen so miteinander verknüpfen, dass bereits frühzeitig Erkenntnisse über die Qualität des Gesamtsystems abschätzbar sind
- Test-Verbesserungen in laufenden Projekten können durch den Teststufen-Ansatz gezielt und vor allem unabhängig voneinander bereits durch kleine Schritte der jeweiligen Spezialisten vorangetrieben werden

Weitere Informationen und Links

- http://www.istqb.org
- http://jenkins-ci.org
- http://hudson-ci.org
- http://maven.apache.org

Martin Böhm
martin.boehm.business@gmx.de

Drillinge – bei der Geburt getrennt. Wie PL/SQL, Apex und Continuous Integration wieder zusammenfinden

Markus Heinisch, Trivadis GmbH

In vielen Technologien gehört Continuous Integration (CI) zum „State of the Art" professioneller Anwendungsentwicklung. Integrations-Probleme können damit zeitnah entdeckt und behoben werden – und nicht erst kurz vor einem Meilenstein. Fehler werden durch die Ausführung von Unit-Tests sofort sichtbar, zudem ist die konstante Verfügbarkeit eines lauffähigen Standes für Demo-, Test- oder Vertriebszwecke gewährleistet.

Die professionelle Anwendungsentwicklung bringt herausragende Applikationen hervor, die dem Nutzer einen hohen Mehrwert liefern. Jedoch sind jedem Entwickler eine Reihe von Risiken und Hürden bewusst, bis die Applikation fertig entwickelt ist. Dazu gehört die Zusammenarbeit von Entwicklungsteams an einem System. Die Software wird von vielen Kollegen entwickelt und muss an den dafür vorgesehenen Schnittstellen zusammenpassen. Aus Erfahrung weiß jeder Entwickler, dass die Integration der Arbeitsergebnisse aller Entwickler aufwändig ist und zu fehlerhafter Software führen kann.

In der Vergangenheit wurde der Zusammenbau der Software aus ihren Komponenten zu einem System als sehr arbeitsintensiv erkannt. Folgerichtig wurde beispielsweise im Projektplan vor einem Meilenstein eine Integrationsphase vorgesehen. Ein weiteres Risiko ist die manuelle und nur gelegentliche Überprüfung der eigenen Software-Entwicklung. Tests werden von Hand auf der eigenen Entwicklungsumgebung ausgeführt und oft genug nur die letzten Änderungen an der Software getestet.

In der Regel gibt es in der professionellen Anwendungsentwicklung ein Test-Team, das die Anwendung gründlich auf Herz und Nieren prüft. Allerdings muss diese schon einen Qualitätsstand erreicht haben, damit die Tests ausgeführt werden können, ohne dass die Anwendung frühzeitig mit einem Fehler abbricht.

Continuous Integration (CI) vermindert die Auswirkungen dieser Probleme in der Anwendungsentwicklung. Die zentrale Idee besteht darin, die Anwendung automatisiert auf einem zentralen Server zusammenzubauen und auf Korrektheit zu prüfen. Dieser Vorgang findet jeden Tag statt und nicht nur in einer Integrationsphase vor einem Meilenstein.

Voraussetzung ist, dass jeder Entwickler mindestens einmal pro Tag seine Arbeitsergebnisse in ein Software-Verwaltungssystem (SCM) einspielt. Damit stellt er seine Ergebnisse dem Team und dem CI-Prozess zur Verfügung. Wenn jeder Entwickler im Team jeden Tag einen Check-in ins SCM durchführt, kann die Anwendung täglich mehrfach zusammengebaut und geprüft werden. Für die notwendige Prüfung auf Korrektheit nach dem Zusammenbau der Anwendung sind Tests notwendig. Die Entwicklung dieser Tests liegt in den Händen der Entwickler und sollte parallel zur Anwendung passieren.

Der typische Softwareentwicklungs-Prozess wird um den Continuous-Integration-Kreislauf ergänzt (siehe Abbildung 1). Nach dem Bau und zur Prüfung der Software schließt er sich an die Entwicklung an, er besteht aus vier Hauptschritten, im Folgenden „Build-Lauf" genannt:

- *Compile*
  Der aktuelle Stand der Software wird komplett aus dem SCM geladen. Je nach Technologie übersetzt ein Compiler die Software in eine Zielsprache oder die Software kommt in eine Laufzeit-Umgebung. In beiden Fällen wird auf syntaktische und grammatikalische Korrektheit geprüft.
- *Test*
  Tests dienen der Überprüfung der Korrektheit des Laufzeit-Verhaltens der Software. Diese Tests müssen schnell ausgeführt werden, damit der Build-Lauf nicht zu lange dauert und der Entwickler auf das Feedback der Tests warten kann.

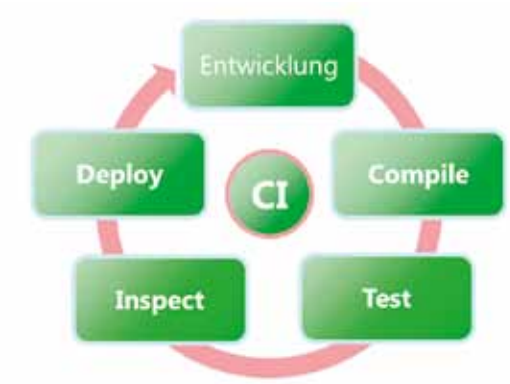


Abbildung 1: Der Continuous-Integration-Kreislauf

- *Inspect*
  Mit einer statischen Code-Analyse wird die Software einer Qualitätsprüfung unterzogen. Sie kann auf typische Fehlermuster in der Programmierung oder auf die Einhaltung von Coding-Guidelines untersucht werden.
- *Deploy*
  Wurden die drei ersten Schritte erfolgreich abgeschlossen, so wird die gebaute und qualitätsgesicherte Software in einer Laufzeit-Umgebung installiert. Dort steht sie für weitergehende Tests oder als Demo-System zur Verfügung.

Der Build-Lauf endet mit einem erfolgreichen oder erfolglosen Ergebnis. Im Erfolgsfall wurde die Applikation ohne Fehler zusammengebaut sowie überprüft und der Entwickler kann an der Applikation weiterentwickeln. Im Fall eines erfolglosen Ergebnisses weiß der Entwickler, dass seine letzten Änderungen fehlerhaft sind. Jetzt muss der Fehler schnellstens behoben werden, bevor der nächste Build-Lauf stattfinden kann. Sonst läuft er ebenfalls in diesen Fehler und die Software ist nicht mehr in einem lauffähigen und qualitätsgesicherten Zustand.

Für den reibungslosen Ablauf der vier Build-Schritte und die Wiederholbarkeit des Build ist die Automatisierung aller Build-Schritte eine wesentliche Voraussetzung. Manuelles Eingreifen in den CI-Prozess birgt die Gefahr, dass der Build nicht nachvollziehbar ist und der CI-Prozess an eine Person gebunden ist, die „weiß, wie man richtig baut". Der Grad der Automatisierung muss zu 100 Prozent bei den Builds liegen, die nach jedem Check-in ins SCM stattfinden. Lassen sich Schritte nicht automatisieren, ist zu überlegen, ob diese Schritte in einen Build-Lauf integriert werden müssen oder ob es nicht besser ist, diese Schritte manuell auf Anforderung durchzuführen.

Apex im CI-Prozess

Oracle Application Express (Apex) ist ein „Rapid Application Development"-Tool, das das Erstellen von Web-Applikationen auf der Basis der Oracle-Datenbank ermöglicht. Eine Apex-Applikation besteht aus einem Web-Frontend und einem Datenbank-Backend. Das Web-Frontend wird durch Konfiguration des Apex-Frameworks und optionalen JavaScript-Routinen implementiert.

Ein Aspekt der Apex-Entwicklung hat im Rahmen des CI-Prozesses eine besondere Bedeutung. Alle Entwickler einer Applikation arbeiten gemeinsam mithilfe eines Web-Browsers auf einem zentralen Entwicklungssystem. Im Unterschied zu einer Entwicklung mit Java oder .NET findet die Apex-Entwicklung nicht lokal und damit nicht separiert von anderen Entwicklern statt. Stattdessen arbeiten die Entwickler parallel auf einem Server an derselben Applikation, aber an verschiedenen Anwendungsteilen. Dies bedeutet, dass die Integration der Arbeitsergebnisse jedes einzelnen Entwicklers durch Apex schon gegeben ist.

Wird Continuous Integration dann überhaupt noch gebraucht? Ja, denn die folgenden Punkte stellen einen Mehrwert des CI-Prozesses dar:

Abbildung 2: Build-Kreislauf in zwei Phasen

- Bau der Applikation auf Basis der Quelldateien, die im SCM vorhanden sind
- Überprüfung der Applikation auf einem neutralen Server (nicht der Entwicklerserver), der nicht direkt durch die Entwickler modifiziert wird, damit der technische Stand für alle nachvollziehbar ist
- Integration weiterer Entwicklungen zu einer Apex-Lösung, wie beispielsweise externe Schnittstellen oder weitere PL/SQL-Packages
- Automatisches Testen und Inspektion der Applikation

Ebenso, wie eine Apex-Applikation aus einer Datenbank- und einer Web-Komponente besteht, ist der Build-Lauf ebenfalls in zwei Teile aufgeteilt (siehe Abbildung 2). Zunächst werden in einer ersten Phase die Datenbank-Objekte aufgebaut und überprüft. War der Durchlauf erfolgreich, schließt sich die zweite Phase an. Dazu werden die Apex-Dateien in die Datenbank geladen und anschließend ebenfalls getestet. Nach einem erfolgreichen Build-Lauf steht die aktuelle Apex-Applikation auf der Apex-Datenbank zur Verfügung.

Check-in und Check-out SCM

Bevor der CI-Build beginnt, übergibt der Entwickler seine Weiterentwicklung an der Apex-Applikation als Sourcecode an ein SCM wie beispielsweise Subversion. Da der Sourcecode die gesamte Applikation und somit auch den Code der anderen Teammitglieder enthält, muss der Zeitpunkt des Check-ins im Entwicklungsteam abgesprochen sein. Ansonsten wird gegebenenfalls nicht-fertiger Sourcecode im SCM gespeichert. Der CI-Build lädt zunächst alle Sourcen der Apex-Applikation aus dem SCM, danach können die DB- und die Web-Phase ausgeführt werden. Ziel dieser Phase ist eine funktionierende Datenbank-Komponente, die als qualitätsgesicherte Basis für die Web-Komponente dient. Dazu wird die Datenbank automatisiert aus dem SCM aufgebaut.

Der Schritt lädt alle benötigten Sourcen zum Aufbau der Datenbank-Objekte aus dem SCM. Dies bedeutet, dass alle DDL- und DML-Befehle als Skript im SCM hinterlegt werden. Dabei ist eine grundsätzliche Überlegung wichtig: Die Skripte sollen so aufgebaut und organisiert sein, dass jede beliebige Version der Datenbank-Kompo-

nente aufgebaut werden kann. Dies hat in einem CI-Prozess den Hintergrund, dass zu älteren Ständen zurückgegangen werden kann, um beispielsweise einen erfolgreich gebauten, älteren Stand als Testsystem zu installieren. Es unterstützt den versionierten Fortschritt der Web-Komponente und fördert die Aufteilung der Datenbank-Arbeitspakete in kleine Entwicklungsschritte. Beim Aufbau der Datenbank-Komponente dürfen die Test- und die Stammdaten der Applikation nicht vergessen werden. Auch sie müssen versioniert abgelegt und durch Skripte auf den aktuellen Stand migriert werden.

Die Datenbank

Im Compile-Schritt dieser Phase werden die Skripte zum Aufbau der Datenbank-Objekte und zum Laden der notwendigen Daten ausgeführt. Der Erfolg dieses Schritts lässt sich durch einfache SQL-Skripte überprüfen, beispielsweise durch eine Suche nach invaliden Objekten im Schema. Diese Prüfskripte kann jeder Datenbank-Entwickler oder -Administrator implementieren.

Der Test-Schritt führt die Unit-Tests zum Prüfen des PL/SQL-Codes durch. Damit sie isoliert von anderen Tests und in beliebiger Reihenfolge laufen können, muss die Datenbank zu jedem Testlauf in einen definierten Zustand gebracht werden. Der Aufwand zum Entwickeln der Tests lohnt sich, denn damit erfährt der Entwickler zuverlässig, ob seine Änderungen an der Datenbank korrekt sind.

Der Inspect-Schritt führt eine Analyse des PL/SQL-Codes durch. Der Code wird auf Coding-Guideline-Verletzungen hin untersucht, wie man es von Java oder C# kennt. Der Code Checker lädt die PL/SQL-Skripte und führt eine Reihe von Regelprüfungen durch. Er analysiert auf Ebene der PL/SQL-Grammatik den Code und kann beispielsweise ungenutzte Variablen erkennen. Nach erfolgreichem Ablauf der genannten Schritte steht das Datenbank-Fundament für die Web-Komponente bereit.

Die Web-Phase

Jetzt gilt es, das Web-Frontend der Apex-Applikation zu bauen und zu prüfen. Dazu werden im ersten Schritt aus dem SCM die Apex-Export-Datei und gegebenenfalls weitere Dateien (CSS, Images, JavaScript-Dateien) in die Apex-Umgebung des Build-Servers geladen. Der Vorgang entspricht im CI-Kreislauf dem Compile-Schritt, denn die Apex-Applikation wird beim Import-Vorgang einer einfachen Prüfung durch die Apex-Laufzeitumgebung unterzogen. Ein Import-Tool prüft die Meldungen während des Imports und signalisiert dem Build-Lauf den Erfolg oder Misserfolg des Imports.

Wie schon erwähnt, wird eine Apex-Applikation integrativ entwickelt. Somit ist ein Mindestmaß an Integration schon während der Entwicklung gewährleistet. Dennoch ist dieser Schritt aus verschiedenen Gründen wichtig. Die Apex-Applikation muss in einem SCM versioniert sein, damit der Build gestartet werden kann. Dies fördert die Aufteilung der Arbeit in kleine Teilschritte. Die Applikation wird auf einem zweiten Apex-Server getestet und zeigt damit, dass sie im aktuellen Stand nicht nur auf dem Entwicklungsserver läuft. Letztendlich wird die Applikation auch auf dem Build-Server getestet. Würde man die Tests auf dem Entwicklungsserver ausführen, würden diese mit hoher Wahrscheinlichkeit mit der Weiterentwicklung der Applikation und den benötigten Daten kollidieren.

Beim Import der Apex-Dateien wird die Applikation einer einfachen, grundlegenden Überprüfung unterzogen. Mit dem „Oracle Apex Advisor" steht ein Tool zur Verfügung, das weitere Überprüfungen der Applikation durchführt. Es prüft auf Fehler, Sicherheitseinstellungen, Warnungen, Performance-Probleme, Usability und Regelverletzungen. Einerseits führt der Advisor zum Teil Prüfungen durch, die beispielsweise auch ein Java-Compiler erledigt, andererseits werden unter anderem Regeln überprüft, die typischerweise im „Inspect"-Schritt erfolgen. Im Build-Lauf wird der Advisor-Schritt vor dem Web-Testing aufgerufen, da der Advisor grundlegende Fehler wie fehlende Referenzen findet, die die Ausführung von Unit-Tests stark behindern.

Im zweiten Schritt wird das Web-Frontend überprüft. Die Prüfungen umfassen den eigenen Apex-Code und nicht das Apex-Framework, das von Oracle getestet wird. Die Geschäftslogik, die das Web-Frontend nutzt und in PL/SQL entwickelt ist, wird in der Datenbank-Phase geprüft. Der Entwickler kann in Apex PL/SQL-Code im Rahmen eines anonymen PL/SQL-Code-blocks einbinden, was der Autor nicht empfiehlt. Diese Art der Programmierung führt dazu, dass im Apex-Code Geschäftslogik enthalten ist, die sowieso in der Datenbank ausgeführt wird. Dieser PL/SQL-Code ist isoliert schlecht testbar; er wird erst zu Laufzeit in der Datenbank geparst und ausgeführt. Besser ist es, den PL/SQL-Code als Package in der Datenbank zu implementieren und in der Apex-Applikation aufzurufen.

In diesem Schritt wird die Oberfläche der Applikation durch automatisierte Browser-Tests untersucht. Ziel der Untersuchung sind beispielsweise Ablauflogik, Eingabe- und Ausgabe-Validierung oder Einhaltung der GUI-Spezifikation. An dieser Stelle empfiehlt sich das Tool Selenium (siehe http://seleniumhq.org). Es erlaubt das Entwickeln und Ausführen von GUI-Tests einer Web-Applikation in einem Browser. Falls eigener JavaScript-Code eingesetzt wird, kann er beispielsweise durch eigene Unit-Tests mit dem Unit-Test-Framework „Jasmine" (siehe http://pivotal.github.com/jasmine) getestet werden.

Bei genauer Betrachtung sind die meisten Oberflächen-Tests keine Unit-Tests, denn sie laufen von der Oberfläche bis tief in die Datenbank hinein. Es werden also nebenbei auch die PL/SQL-Routinen in diesem Schritt geprüft. Eine stärkere Trennung von Oberfläche und Datenbank ist technologisch nicht (sinnvoll) durchführbar. Allerdings ist im Rahmen des Build-Laufs wichtig, dass diese Tests schnell durchgeführt werden können, damit der Build nicht zu lange dauert und die Entwickler nicht zu lange auf das Ergebnis warten müssen.

Markus Heinisch
info@trivadis.com

Markus Heinisch ist studierter Informatiker und hat rund 20 Jahre Erfahrung im Bereich Softwareentwicklung und Consulting. Seine Themenschwerpunkte sind Architektur und Software Engineering. Er ist bei Trivadis als Principal Consultant tätig und treibt Themen wie Continuous Integration und Qualitätssicherung innerhalb und außerhalb Trivadis voran.

Datenschutz-konformes Social Sharing mit Liferay

Michael Jerger, Informatikbüro Jerger

Soziale Netze sind aus dem heutigen Geschäftsalltag nicht mehr wegzudenken. Der Artikel gibt einen Überblick über die wichtigen sozialen Plattformen und zeigt das typische, damit verbundene Datenschutz-Problem auf. Speziell für Liferay wird eine erste Datenschutz-konforme Integration vorgestellt. Ein aktuell laufendes Crowdfunding-Experiment eröffnet die Aussicht auf eine komfortablere Version.

Wer heute in die Welt der sozialen Netze einsteigen möchte, der wird rasch von einer Flut von Möglichkeiten, sich sozial zu betätigen, überschüttet. Neben vielen Gemeinsamkeiten der sozialen Plattformen lassen sich aber auch unterschiedliche Schwerpunkte erkennen, die eine Einteilung in die folgenden Kategorien erlaubt:

- *Soziale Netze*
  Die sozialen Netze zeichnen sich durch „Freundschaft" und die Beziehung zwischen Personen aus. Zu den sozialen Netzen gehört Facebook, Xing oder LinkedIn.
- *Sozialer Content*
  Hier stehen Themen und Inhalt im Vordergrund. Google+ legt den Schwerpunkt auf das Teilen von Inhalten – so ist es hier zum Beispiel möglich, zu Themen eigene Seiten zu pflegen. Noch klarer wird die Ausrichtung bei Twitter (Kurznachrichten), Spotify (Audio), YouTube (Video), Flickr/Picasa (Bilder) und Delicious (Bookmarks).
- *Crowdfunding*
  Diese jüngste Kategorie ermöglicht es den Konsumenten, gezielten und direkten Einfluss auf die Schöpfung von neuen Produkten zu nehmen. Die bekanntesten Vertreter hier sind Kickstarter, Flattr oder in Deutschland auch startnext.

Neben dieser Kategorisierung nach Schwerpunkten bietet jede der Plattformen immer auch eine bunte Mischung aus all den anderen Bereichen an. So ist beispielsweise in Flattr auch ein Überblick über alle selbst gesponserten Blog-Artikel möglich – was eine recht deutliche Empfehlung für diese „Bookmarks" darstellt. Alle Plattformen erlauben das Teilen der Inhalte – erwartungsgemäß jedoch nur mit Benutzern derselben Plattform. Ein weiteres wesentliches Element auf allen Plattformen ist die Möglichkeit, dass jeder Benutzer kommentieren kann. Somit ist ein einheitlicher Kommunikationskanal für Benutzer vorhanden. Weitere Informationen über die Relevanz der einzelnen sozialen Plattformen stehen unter „http://de.statista.com/statistik/daten/studie/173771/umfrage/besucherzahlen-sozialer-netzwerke-in-deutschland" sowie unter „http://blog.kennstdueinen.de/2010/03/social-media-statistiken-daten-zu-nutzerzahlen-und-mehr".

Mit dem ersten Überblick stellt sich nun die Frage, was Menschen dazu motiviert, sich in sozialen Netzen aufzuhalten. Wird diese Motivation greifbar, ermöglicht sie zum Beispiel Unternehmen, von der Situation zu profitieren. Hier wird unterstellt, dass der Betrieb eines Liferay-Servers eher kommerzielle Absichten impliziert.

In sozialen Netzen unterhalten sich die Nutzer mit ihren Bekannten und Freunden, entweder um ein Thema von gemeinsamem Interesse versammelt oder eher beziehungsorientiert. Ein wesentlicher Aspekt ist, Zugehörigkeit zu einer Gruppe sowie Anerkennung zu erhalten. Eine häufig genutzte Möglichkeit dazu ist, im Netz entdeckte Perlen den Bekannten zu präsentieren. Solche Perlen sind vielleicht ein neues Musik-Video, eine Nachricht aus der entfernten Heimat, ein cooles Werbe-Video oder ein Beitrag mit wertvollem Inhalt. Das Teilen von Inhalten ist also ein Grundpfeiler des sozialen Netzwerkens – und folgerichtig bieten alle sozialen Plattformen auch eine einfache Möglichkeit zum Teilen von Inhalten an. Allerdings hat dieses komfortable Angebot der sozialen Plattformen für die Benutzer auch einen gravierenden Nachteil: Die Betreiber der sozialen Plattformen haben die Möglichkeit, ein durchgängiges Surfprofil aller Internet-Nutzer zu erstellen.

Surfprofil von allen Internet-Nutzern

Eine Seite, die die angebotenen „Share Buttons" zum sozialen Teilen mit eingebunden hat, übermittelt bei jedem Seiten-Aufruf einen Request an den Betreiber der sozialen Plattform. Die neue Qualität dabei ist, dass auch über unterschiedliche Websites hinweg so ein globales Tracking aller Internet-Nutzer möglich wird – und das nicht nur, wenn sie einen Inhalt mit ihren Freunden aktiv teilen wollen, sondern immer. Schon das Besuchen einer Seite reicht aus und der Plattform-Betreiber hat ein neues Stückchen Information über den Besucher. Wie das funktioniert? Angenommen der Internet-Nutzer besucht die folgenden drei Websites (siehe Abbildung 1). Dabei geschieht Folgendes:

- Damit „www.example1.de" den „Share-Button" einbinden kann, sendet der Browser des Benutzers beim Laden der Seite einen Request an die soziale Plattform.
- Als Antwort erhält der Benutzer den „Share-Button" und auch ein Cookie mit einer eindeutigen ID.
- Der Benutzer besucht eine Seite „www.example2.de", die keine „Share-Buttons" enthält. Hier fließt keine Information an die soziale Plattform.
- Die nächste aufgerufene Seite „www.example3.de" besitzt wieder einen

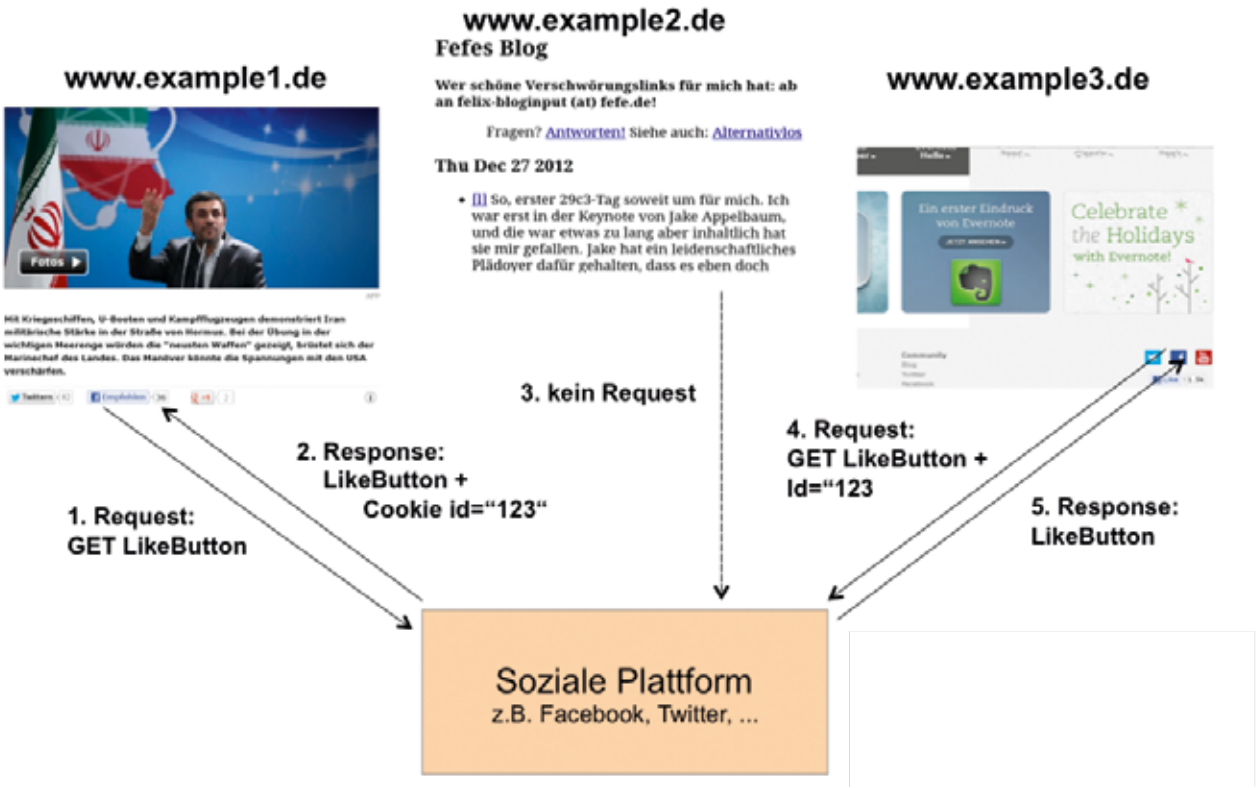


Abbildung 1: Globales Surfprofil für alle Internetnutzer

„Share-Button". Entsprechend schickt der Browser des Benutzers hier eine Anfrage mit der schon bekannten Session-ID an die soziale Plattform.

- Als Antwort erhält der Benutzer den „Share-Button" angezeigt.

Als Ergebnis hat der Betreiber der sozialen Plattform alle Daten, um das angegebene Surfprofil zu erstellen. Solange der Benutzer der Website sich nicht bei der sozialen Plattform anmeldet, bleibt das angelegte Surfprofil anonym. Sobald er sich aber anmeldet, um einen kurzen Blick auf die neuesten Nachrichten zu werfen oder irgendeinen Inhalt zu teilen, ist sein Surfprofil mit seinem Namen personalisiert. Bei einer Anmeldung kann das Surfprofil übrigens auch rückwirkend personalisiert werden.

Die eigenen Besucher schützen

Wie kann man als Website-Betreiber seine Besucher nun schützen? Das Beispiel „http://blog.fefe.de" zeigt eine Möglichkeit: puristisch, kein Design, kein soziales Teilen. Vielleicht ist der puristische Stil aber nicht für jeden geeignet – und auf das bewusste Teilen von Inhalten zu verzichten, ist nur für wenige eine Lösung. Daher rückt die Art und Weise, wie die „Share-Buttons" eingebunden werden können, in den Fokus. Am Beispiel von Google ist hier gezeigt, wie einfach die Integration von Google gedacht ist (siehe Listing 1, Quelle: „https://developers.google.com/+/plugins/+1button").

```
<script type="text/javascript" src="https://
apis.google.com/js/plusone.js"></script>
<g:plusone></g:plusone>
```

Listing 1

Das Tag „<g:plusone/>" wird dabei vom eingebundenen JavaScript zur Ladezeit der Seite direkt im „HTML-DOM" ersetzt. Typischerweise wird ein kleines I-Frame injiziert, in dem dann der „Share-Button" geladen und angezeigt wird. Dabei extrahiert das JavaScript Informationen wie Seitentitel, Abstrakt oder den Autor schon aus der aktuellen Seite. Durch diese automatisch extrahierten Informationen wird das Teilen von Inhalten sehr komfortabel.

Unglücklicherweise lässt sich auf diesem Wege das Nachladen von der sozialen Plattform nicht vermeiden und die Kommunikation mit der sozialen Plattform findet zum Zeitpunkt des Seitenladens statt. Damit gehen die schon beschriebenen Datenschutz-Nachteile zwingend einher. Auf eine Initiative des Heise-Verlags ist es einigen JavaScript-Entwicklern gelungen, die sogenannte „Zwei-Klick-Lösung" umzusetzen. Dabei müssen Leser vor einem Teilen erst den eigentlichen „Share-Button" einschalten, erst dann ist das soziale Teilen möglich. Diese Lösung mag schon von der einen oder anderen Website bekannt sein (siehe Abbildung 2). Das Heise-Script wird dabei ähnlich wie die beschriebene Google-Lösung auf einer Seite eingebunden (siehe Listing 2, Quelle: „http://www.heise.de/extras/socialshareprivacy"). Diese Zwei-Klick-Lösung ist heute für alle Websites Standard, denen der Schutz ihrer Benutzer am Herzen liegt.

```
<script type="text/javascript" src="jquery.
js"></script>
<script type="text/javascript" src="jquery.
socialshareprivacy.js"></script>
…
<div id="socialshareprivacy"></div>
```

Listing 2

Abbildung 2: Leser durch die Zwei-Klick-Lösung schützen

Abbildung 3: Soziale „Share Buttons" in Liferay – ohne Datenschutz

Soziales Teilen in Liferay

Liferay ist als Portal-Server für alle Anwendungsfälle, in denen eine einfache Wordpress-Präsenz nicht ausreicht, sicher eine der relevanten Web-Plattformen. Seit der Version 6.1 bietet Liferay auch in der Community Edition „out of the box" soziale „Share-Buttons" für Blogbeiträge an (siehe Abbildung 3).

Leider wird nicht die datenschutzkonforme Variante verwendet. Um hier nichtinvasiv Datenschutz-konforme „Share-Buttons" verwenden zu können, hat der Autor eine Lösung in zwei Schritten realisiert (siehe Abbildung 4):

- Die Verwendung der „Share-Buttons" an sich und deren Konfiguration wird über einen Liferay-Hook angepasst. Liferay-Hooks überschreiben im Wesentlichen bestehende Properties oder JSPs von Liferay. Dabei kann die Website-spezifische Default-Konfiguration (etwa für den verwendeten Flattr-Benutzer-Account) hier angepasst werden.
- Alle Ressourcen wie Bilder, JavaScripts, Stylesheets und die Einbindung des „jquery.socialshareprivacy.js" auf jeder Seite wird zentral durch das Liferay-Theme geleistet.

Die Details auf Code-Ebene sind im Blog des Autors unter „https://www.jerger.org/blog" beschrieben.

Dadurch, dass die skizzierte Lösung einerseits einen Liferay-Hook und andererseits ein Liferay-Theme benötigt, wird es schwer, alles zusammen sauber in einer Komponente zu verpacken. Darüber hinaus wird die Heise-Lösung nicht mehr aktiv weiterentwickelt. Die Integration eines „Flattr-Buttons" ist daher nur wenig elegant gelungen. Die Website-spezifische Konfiguration direkt im Liferay-Hook zu platzieren, ist sicher ebenfalls nur für einen Prototyp geeignet. Aus diesen Gründen ist der nachfolgende Entwicklungsschritt eigentlich logisch.

Abbildung 4: Soziale „Share Buttons" in Liferay – mit Datenschutz

Datenschutz-Konformität mit Crowdfunding

Mit den vielen kritikwürdigen Stellen im Hinterkopf müsste eine gute Lösung die folgenden Eigenschaften haben:

- Alle notwendigen Bestandteile sind in einer Komponente „Portlet" gebündelt
- Das soziale Teilen findet auf der Granularität einer Seite statt
- Für sämtliche Konfigurationsmöglichkeiten steht eine komfortable Oberfläche zur Verfügung
- Eine Integration in weitere sozialen Plattformen ist einfach möglich
- Die Komponente steht unter einer Open-Source-Lizenz wie der Apache-Lizenz 2.0 für alle Interessierten zur freien Benutzung zur Verfügung

Leider gibt es ein solches Portlet noch nicht – was liegt also näher, als eines durch Crowdfunding finanziert zu erstellen? Genau dieser Versuch wird vom Autor des Artikels gerade unternommen. Die Details dazu stehen unter https://www.jerger.org/socialshareprivacyportlet. Crowdfunding ist dabei sowohl Finanzierung als auch Experiment. Für alle, die das Experiment begleiten wollen, werden der Verlauf, Einsichten und Erkenntnisse unter der oben genannten Adresse fortlaufend dokumentiert. Feedback, Ideen und Unterstützung sind herzlich willkommen.

Michael Jerger
buero@jerger.org

Michael Jerger ist IT-Architekt, Projektmanager und immer noch neugierig. Sein Motto lautet „Demotivation eliminieren bedeutet mit Vergnügen arbeiten" – was stets schnell zu den wirklich interessanten Punkten im Projekt führt und meist auch zu erfolgreichen Teams.

Unbekannte Kostbarkeiten des SDK
Heute: Der ZIP-File-System-Provider

Bernd Müller, Ostfalia

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

Mit Java SE 7 wurde die zweite Version des neuen Ein-/Ausgabesystems (NIO.2) eingeführt. Es abstrahiert von realen Dateisystemen und stellt Klassen und Methoden für das Erzeugen, Kopieren, Verschieben, Umbenennen und Löschen von Dateien bereit. Als Standard-Dateisysteme werden die typischen Java-Plattformen, also Windows und Unixoide, unterstützt. Zusätzlich werden jedoch auch ZIP-Archive als Dateisysteme behandelt und deren Dateien sind unter demselben API wie gewöhnliche Dateien verfügbar, was die Handhabung von ZIP-Archiven im Vergleich zu den bisherigen Möglichkeiten stark vereinfacht.

ZIP-Dateien in Java

ZIP-Dateien sind durch die Klasse „ZipFile" im Package „java.util.zip" repräsentiert. Die Klasse „ZipEntry" und verschiedene Klassen für In- und Output-Streams vervollständigen das Package. JAR-Files werden durch die Klasse „JarFile", eine direkte Unterklasse von „ZipFile", im Package „java.util.jar" repräsentiert. Auch hier vervollständigen weitere Klassen in diesem Package die Handhabung von JAR-Dateien.

Als kleines Beispiel realisieren wir eine Methode zum Lesen einer Datei innerhalb einer ZIP-Datei. Die Methode liefert für den gegebenen ZIP-Dateinamen und den Eintrag (ZIP-Entry) innerhalb der ZIP-Datei einen „BufferedReader", mit dem der Inhalt des ZIP-Entry gelesen werden kann (siehe Listing 1). Typisch für diese Art der Verwendung ist die Suche nach dem ZIP-Entry durch eine Iteration über alle ZIP-Entries der Datei.

Der ZIP-File-System-Provider

Mit Java 7 wurde der sogenannte „ZIP-File-System-Provider" eingeführt. Auf seine Entstehungsgeschichte und Implementierung gehen wir später kurz ein. Zunächst soll das obige Beispiel mit dessen Hilfe reimplementiert werden. Die folgende Methode zeigt diese Implementierung, wobei die Verwendung allgemeiner Klassen und Methoden von NIO.2 und der völlige Verzicht auf ZIP- beziehungsweise JAR-spezifische Klassen und Methoden hervorzuheben ist (siehe Listing 2).

Die Methode bekommt im ersten Parameter „zipFile" den Namen der ZIP-Datei und im zweiten Parameter „entryPath" den Pfad des ZIP-Eintrags übergeben und ist damit Signatur-identisch zur ursprünglichen Methode. Die Methode „URI.create()" erwartet im Parameter zunächst ein Schema mit anschließendem schemaspezifischem Anteil. Im Falle des ZIP-Filesystem-Providers muss dieser hierarchisch sein, sodass der Name der ZIP-Datei mithilfe der Klasse „Path" zunächst in diese Form umgewandelt wird.

Die Fabrikmethode „FileSystems.getFileSystem()" verwendet das Schema des URI, um ein passendes File-System zurückzuliefern. Im Beispiel erhält man das Dateisystem des ZIP-Archivs. Der zweite Parameter der Methode sind dateisystemspezifische Properties, die, wie in unserem Beispiel, auch leer sein dürfen. Zuletzt wird mit „Files.newBufferedReader()" das Methoden-Ergebnis erzeugt. Im Vergleich zur ursprünglichen Implementierung entfällt die Suche nach dem ZIP-Eintrag und die Realisierung erscheint insgesamt einfacher und konsistenter.

Das Erzeugen eines ZIP-Archivs

Das Erzeugen eines ZIP-Archivs mithilfe des ZIP-File-System-Providers ist ebenfalls deutlich einfacher als bisher und verwendet ausschließlich Standard-APIs (siehe Listing 3).

Die Methode bekommt die zu kopierende Datei (toCopy), den Namen des ZIP-Archivs (zipFile), das Verzeichnis im ZIP-Archiv (entryDir) sowie den im ZIP-Archiv zu verwendenden Dateinamen (entryFile) übergeben. Im Gegensatz zum ersten Beispiel ist der zweite Parameter für „FileSystems.newFileSystem()" nicht leer, sondern enthält für den Schlüssel „create" den Wert „true". Damit wird das ZIP-Archiv angelegt, falls es noch nicht existiert. Der Default-Wert ist „false" und wurde somit implizit im ersten Beispiel verwendet. Die Syntax entspricht dem Double-Brace-Initialization-Pattern, das wir in den verborgenen Kostbarkeiten in der Java aktuell 3/2012 [1] vorgestellt haben. Die weiteren Methoden-Aufrufe sind selbsterklärend.

Entwicklungsgeschichte und Implementierung

Laut [2] wurde der ZIP-File-System-Provider als Demonstrator-Projekt der Möglichkeiten von NIO.2 begonnen. Nach der Fertigstellung wurde beschlossen, ihn in das JDK 7 zu übernehmen. Der Quell-Code ist im JDK 7 im Unterverzeichnis „demo/nio/zipfs" einzusehen. Der Provider wird dem Laufzeitsystem über die Service-Provider-Schnittstelle bekannt gegeben, die wir in den verborgenen Kostbarkeiten in Java aktuell 4/2011 [3] beschrieben haben. Das zu implementierende Interface ist FileSystemProvider im Package „java.nio.file.spi". Eine Einführung in den ZIP-File-System-Provider findet man in [4]. Falls ein eigener File-System-Provider realisiert werden soll, hilft [5] weiter.

Fazit

Mit dem SDK 7 wurde der ZIP-File-System-Provider eingeführt. Er realisiert über das Standard-Dateisystem-API des NIO.2 den Zugriff auf ZIP-Archive und erleichtert damit den Umgang mit solchen Archiven erheblich.

Weitere Informationen

[1] Bernd Müller. Unbekannte Kostbarkeiten des SDK – Heute: Double Brace Initialization und Instance Initializier. Java aktuell 3/2012.

[2] Xueming Shen. The ZIP filesystem provider in JDK 7: https://blogs.oracle.com/xuemingshen/entry/the_zip_filesystem_provider_in1

[3] Bernd Müller. Unbekannte Kostbarkeiten des SDK – Heute: Der Service-Loader. Java aktuell 4/2011

[4] Oracle, Zip File System Provider: http://docs.oracle.com/javase/7/docs/technotes/guides/io/fsp/zipfilesystemprovider.html

[5] Oracle, Developing a Custom File System Provider: http://docs.oracle.com/javase/7/docs/technotes/guides/io/fsp/filesystemprovider.html

Bernd Müller
bernd.mueller@ostfalia.de

Bernd Müller ist Professor für Software-Technik an der Ostfalia. Er ist Autor des Buches „JavaServer Faces 2.0" und Mitglied in der Expertengruppe des JSR 344 (JSF 2.2).


```
public BufferedReader newBufferedReaderForEntry(String zipFile,
                                    String entryPath) throws IOException {
    ZipFile zf = new ZipFile(zipFile);
    Enumeration<? extends ZipEntry> entries = zf.entries();
    while (entries.hasMoreElements()) {
        ZipEntry entry = entries.nextElement();
        if (entry.getName().equals(entryPath)) {
            return new BufferedReader(new InputStreamReader(zf.getInputStream(entry)));
        }
    }
    throw new IOException("ZIP-Entry not found");
}
```

Listing 1

```
public BufferedReader newBufferedReaderForEntry(String zipFile,
                                    String entryPath) throws IOException {
    Path path = Paths.get(zipFile);
    URI uri = URI.create("jar:file:" + path.toUri().getPath());
    Map<String, String> env = new HashMap<>();
    FileSystem zipFileSystem = FileSystems.newFileSystem(uri, env);
    return Files.newBufferedReader(zipFileSystem.getPath(entryPath), StandardCharsets.UTF_8);
}
```

Listing 2

```
public void copyIntoZip(String toCopy, String zipFile,
                String entryDir, String entryFile) throws IOException {
    Path path = Paths.get(zipFile);
    URI uri = URI.create("jar:file:" + path.toUri().getPath());
    Map<String, String> env = new HashMap<String, String>() {{
        put("create","true"); // DBI, siehe [1]
    }};
    FileSystem zipfs = FileSystems.newFileSystem(uri, env);
    Path external = Paths.get(toCopy);
    Files.createDirectory(zipfs.getPath(entryDir));
    Path pathInZipfile = zipfs.getPath(entryDir + "/" + entryFile);
    Files.copy(external, pathInZipfile);
    zipfs.close();
}
```

Listing 3

DevFest Vienna 2012 – es war ein voller Erfolg!

Am 10. und 11. November 2012 hat die Java Student User Group Wien (JSUG) gemeinsam mit der Google Developer Group Vienna ihre erste eigene Konferenz abgehalten. Am ersten Tag gab es zwölf Vorträge für die knapp 150 Teilnehmer, Unmengen an Pizza, Drinks, Süßigkeiten, T-Shirts, Gimmicks und sozialer Interaktion. Es folgte eine tolle Afterparty inklusive Gewinnspiel und Verlosung von Raspberry Pis, Arduino Duos und Büchern. Ein Android Birthday Hackaton am Folgetag schloss die erfolgreiche Veranstaltung ab. Das DevFest Vienna 2013 ist für Anfang Oktober 2013 geplant. *Kontakt: Dominik Dorn, dominik.dorn@jsug.at*