iJUG Verbund

Java aktuell – das Magazin für die Java-Community

Das iJUG Magazin

Java aktuell

Oracle und die Zukunft von Java

Erfahrungen, Ideen und Lösungen für Java-Entwickler

www.ijug.eu
Q4 / September 2010

D: 4,90 EUR A: 5,20 EUR CH: 7,10 CHF Benelux: 5,20 EUR ISSN 1234567-8910

Wolfgang Taschner
Chefredakteur Java aktuell

Mehr als 6,5 Millionen Entwickler können nicht irren!

Java ist die am meisten genutzte und populärste Programmiersprache. Sie kommt weltweit in rund 4,5 Milliarden PCs, Mobiltelefonen, Smart Cards, Fahrzeugen oder elektronischen Geräten zum Einsatz und ist damit für viele Branchen unverzichtbar. Und gerade wegen dieser enormen Verbreitung hat der Einfluss auf den Support und die Weiterentwicklung von Java eine essenzielle Bedeutung. Nach der Übernahme von Sun durch Oracle werden hier zumindest einige Karten neu gemischt. Erste Ergebnisse werden zur Java One im Herbst erwartet.

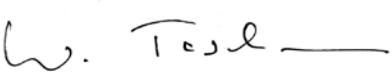
Gemeinsam sind wir stark – unter diesem Motto haben sich acht Java User Groups aus Deutschland gemeinsam mit der DOAG Deutsche ORACLE-Anwendergruppe e.V. im Interessenverbund der Java Usergroups e.V. (iJUG) vereint. Ziel ist die umfassende Vertretung der gemeinsamen Interessen der Java-Anwendergruppen sowie der Java-Anwender im deutschsprachigen Raum, insbesondere gegenüber Entwicklern, Herstellern, Vertriebsunternehmen sowie der Öffentlichkeit. Bei Wahrung der Eigenständigkeit der Mitglieder sollen insbesondere eine gemeinsame Öffentlichkeitsarbeit stattfinden und der Informations- und Erfahrungsaustausch sowie Netzwerkbildung gefördert werden. Eine der ersten Aktivitäten ist die Herausgabe dieser Zeitschrift.

Die Beteiligung der DOAG an diesem Verbund war kein Zufall, schließlich ist Oracle nach der Übernahme von Sun zum Keyplayer für die Programmiersprache Java avanciert. Geprägt von zahlreichen Disputen mit dem amerikanischen Hersteller, bei denen es um die Interessen der Datenbank-Anwender, Entwickler und Benutzer der Business-Applikationen ging, hat sich die DOAG bereits bis hinauf in die Konzernzentrale einen Namen gemacht. Diesen gilt es jetzt beim Einsatz für die Java-Anwender unter Beweis zu stellen.

Andere Mitglieder des iJUG haben seit etlichen Jahren bekannte Veranstaltungen organisiert – etwa die JUG Stuttgart mit dem Java Forum oder die Sun Usergroup Deutschland, die gemeinsam mit der Java User Group Deutschland die Source Talk Tage durchführt. Hier reiht sich die DOAG 2010 Konferenz von 16. bis 18. November in Nürnberg mit einem üppigen Java-Programm ein, das von keinem geringeren als Java-Guru James Gosling mit einer Keynote begleitet wird. Auch die weiteren Keynotes zu Java werden von den Machern bestritten: Ed Burns (Spec Lead) berichtet über Java Server Faces, und Brian Oliver, Senior Principal Solutions Architect aus dem Coherence-Team, referiert über Oracles Lösung für Datenreplikation und verteiltes Datenmanagement. Im Gegensatz zu anderen Java-Veranstaltungen liegt der Mehrwert der DOAG 2010 Konferenz im konkreten Oracle-Bezug.

Ich wünsche Ihnen viel Spaß beim Lesen dieser Zeitschrift und freuen mich auf Ihr Feedback an redaktion @ijug.eu. Unter der selben Adresse können Sie sich auch mit einem Artikel an der nächsten Ausgabe beteiligen, die künftig viermal im Jahr erscheinen soll.

Ihr



Inhalt

3 Editorial
Wolfgang Taschner

5 Die Oracle + Sun Java-Strategie
Oracle Corporation

6 Interview mit Richard Seibt:
„Oracle könnte durch richtige Strategien und entsprechende Investments die Innovationsgeschwindigkeit von Java noch erhöhen …"

9 Interview mit Günther Stürner und Bernd Rintelmann:
„Sun ist als Marke wertvoll, sie stand und steht für Innovation, Geschwindigkeit und IT-Kultur erster Güte …"

13 Oracle und die Zukunft von Java
Markus Eisele

16 Die Geschichte der Bohne
Oliver Szymanski

19 JSXP – das etwas andere Web Framework
David Tanzer

23 Enterprise Java im Griff des Spring Frameworks
Robert Szilinski

27 Ausblick auf Java 7
Oliver Szymanski

30 Aus Alt mach Neu
Markus Eisele

33 Entwicklungsumgebung mit NetBeans, GlassFish v3 und PostgreSQL
Gunther Petzsch

37 Source Talk Tage in Göttingen
Stefan Koospal

39 Clojure
Stefan Kamphausen

41 Pleiten, Pech und PatternTesting – ein Drama in mehreren Akten
Oliver Böhm

46 Der Werkzeugkasten des Admins für den Java-Server
Tobias Frech

50 JUnit – Tests lesbar gestalten
Dirk Dittert

55 Java everywhere
Till Hahndorf

54 Inserentenverzeichnis

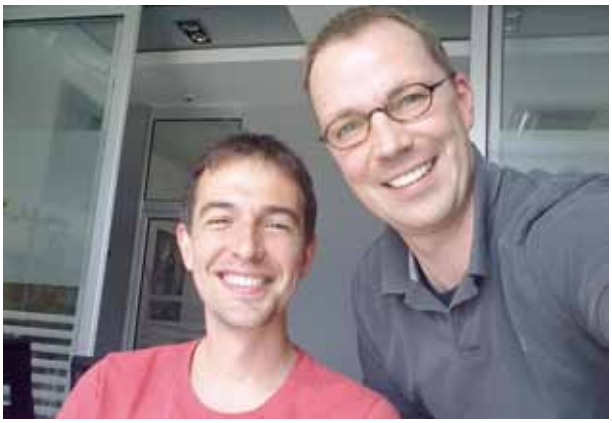
56 Impressum

62 Apex und PL/SQL meet Java: unbegrenzte Möglichkeiten mit der Datenbank-JVM
Carsten Czarski

66 Termine der Java-Community

Oracle-Mitarbeiter stellen sich den Fragen des iJUG
Seite 9

Netbeans-Training im Rahmen der Source Talk Tages
Seite 60



Die globale Welt von Java am Beispiel der JUG Belgrad
Seite 55

Dieser Artikel beschreibt die Java-Strategie von Oracle und Sun, entnommen aus dem gleichnamigen WebCast [1] von Hasan Rizvi, Senior Vice President Oracle Fusion Middleware Development, und Jeet Kaul, Vice President Java-Entwicklung. Im Einzelnen wird auf das Verhältnis von Oracle zu Java, die in der Java-Produktstrategie geplanten Innovationen und die zukunftsorientierte Zusammenarbeit zwischen Oracle und Sun eingegangen.

Die Oracle + Sun Java-Strategie

Oracle Corporation

Java spielt für die Software-Branche eine bedeutende Rolle. Mit weltweit über neun Millionen Entwicklern ist es die am weitesten verbreitete Programmiersprache und damit auch für Oracle von immenser Bedeutung. Die Komponenten der Oracle Fusion Middleware, sowie eine Vielzahl darauf aufbauender Oracle-Anwendungen, sind in Java geschrieben. Oracle arbeitet schon seit Mitte der Neunziger mit Java. Damals wurde die Java-Entwicklungsumgebung „JDeveloper" implementiert und Java- Komponenten in die Datenbank eingebaut, so dass neben PLSQL auch Java Code im Datenbank-Kernel ausgeführt werden konnte. Der Ausbau der Oracle Fusion Middleware wurde konsequent mit Java umgesetzt und Oracle lieferte dank seines TopLink Know-hows mit EclipseLink die Referenzimplementierung des JPA 2.0 Standards. Darüber hinaus ist der Oracle WebLogic Server der am häufigsten eingesetzte Java-Anwendungsserver. Oracle war an über 80 Java Specification Requests beteiligt, häufig als federführender Experte, und gehört seit vielen Jahren dem Vorstand des Java Community Process an, der die offene Weiterentwicklung von Java steuert.

Java ist nicht nur eine sehr vielfältige und weit verbreitete Programmiersprache für Desktops und Web-basierte Anwendungen, sondern findet über die Java Virtual Machine Verwendung auf Geräten aller Art: von Millionen von Servern über Desktop-Rechner, Mobilgeräte und Fernseher bis hin zu Sim-Karten. Die verschiedene APIs für die Anwendungsentwicklung werden neuerdings um JavaFX, einer Entwicklungsumgebung, auf die später noch näher eingegangen wird, erweitert.

Sehen wir uns die zukünftige Java-Strategie nach der Übernahme von Sun durch Oracle etwas näher an. Wie schon erwähnt, hat Oracle viel in Java investiert. In mehreren Gesprächen hat Oracle's CEO Larry Ellison die große Bedeutung von Java für Oracle, die Entwickler und die SW-Industrie hervorgehoben. Deshalb freuen wir uns darüber, dass wir jetzt noch enger in die Java-Welt eingebunden sind.

Wir möchten die Verbreitung von Java auch weiterhin vorantreiben. Java soll die am weitesten verbreitete Programmiersprache bleiben, die es den Entwickler ermöglicht, zuverlässige und leistungsfähige Anwendungen zu implementieren. Und wir möchten uns auf seine Weiterentwicklung konzentrieren, damit irgendwann alle Software dieser Welt auf Java aufbaut.

Es ist beeindruckend zu sehen, wie sehr Java sich in den letzten 12 bis 13 Jahren entwickelt hat. Der Motor dieses Wachstums sind die neun Millionen Java-Entwickler mit der Java Platform Standard Edition als Grundlage. Das Java Developer Kit wird jährlich sieben Millionen Mal heruntergeladen und dabei auf mehr als hundert Hardware-Plattformen eingesetzt, die wir alle auch in Zukunft unterstützen werden. Dabei werden HotSpot als auch JRockit weiterhin die strategische JVMs bleiben. Oracle setzt die Arbeit am JDK7 fort und wird es noch 2010 veröffentlichen. Es wird eine Vielzahl von Neuigkeiten enthalten, um die Modularisierung und Entwicklerproduktivität zu steigern. Besonderer Wert wird hier zusätzlich auf die Unterstützung dynamischer Sprachen und auf verbesserte Performance gelegt.

Die meisten Entwickler nutzen Java auf einem normalen Computer. Weltweit ist die Java Runtime Environment auf über 840 Millionen PCs installiert. Jeden Monat werden die JRE und ihre Updates gut 60 Millionen Mal heruntergeladen. Das macht über 500 Millionen Java-Downloads im Jahr. Deshalb haben wir JavaFX, auf das wir noch näher eingehen werden, in die JRE integriert. Oracle wird Java in dieser Richtung weiterentwickeln, damit es überall auf dem Desktop eingesetzt wird.

Einer der aktivsten Bereiche von Java ist die Java Enterprise Edition. An dieser Server-Software beteiligen sich unterschiedlichste Gruppierungen: Einzelne Mitglieder, Open-Source-Gemeinschaften und Großunternehmen. Innerhalb dieser vielfältigen Entwicklergemeinde ergeben sich ständig Neuerungen. Erst im Dezember wurde die Java EE 6 freigegeben, nachdem sie vom JCP am 14. Dezember genehmigt worden war. Für die JEE wird GlassFish auch weiterhin die Referenz- Implementierung bleiben und als Open-Source-Projekt als auch in einer kommerziellen Version von Oracle verfügbar sein.

Doch Java ist nicht nur auf herkömmlicher Computer-Hardware sondern auch auf Mobiltelefonen allgegenwärtig. Es existieren Installationen auf rund 2,6 Millionen Mobiltelefonen, das sind 85 Prozent aller Geräte. Wir arbeiten in diesem Bereich mit Hunderten Netzbetreibern und zahlreichen Geräteherstellern zusammen. Über hunderttausend mobile Anwendungen, die auf Java beruhen sind auf diese Art entstanden. Welche Pläne haben wir also für diesen Bereich? Wir werden Java für Mobilgeräte weiterentwickeln, und zwar sowohl die Java Mobile Edition als auch die Mobile Service Architecture 2. Die Leistung ist uns dabei sehr wichtig. An ihr werden wir weiterhin arbeiten. Weil wir viel in JavaFX investieren, werden wir sicherstellen, dass es auch in der Java ME optimal ausgeführt werden kann.

Bitte lesen Sie weiter auf Seite 38.

Bei der Übernahme von Sun durch Oracle geht es auch um die Zukunft von Java. Der iJUG sprach darüber mit Richard Seibt, Gründer und CEO der Open Source Business Foundation (OSBF).

„Oracle könnte durch richtige Strategien und entsprechende Investments die Innovationsgeschwindigkeit von Java noch erhöhen …“

Wie haben Sie die Nachricht über die Übernahme von Sun durch Oracle damals aufgenommen?

Seibt: Ich war zunächst wie viele andere sehr erstaunt, dass der Deal nicht mit IBM zustande gekommen ist. Nachdem die Überraschung verflogen war, habe ich schnell die Vorteile erkannt, die eine Übernahme durch Oracle mit sich bringt.

Wie beurteilt die Open-Source-Gemeinde die Sun-Übernahme durch Oracle? Welche Chancen und welche Risiken werden gesehen?

Seibt: Zunächst möchte ich den Begriff „Open-Source-Gemeinde“ etwas näher definieren. Wir haben es auf der einen Seite mit Entwicklern zu tun, die sich auf Projekte wie Java, MySQL, PHP oder Ähnliches fokussieren. Daneben gibt es auch eine Open-Source-Business-Gemeinde, die als Unternehmen auf Basis von Open-Source-Komponenten professionelle Lösungen anbieten.

Bei den Entwicklern sind nach Bekanntgabe der Übernahme durch Oracle zunächst Sorgen entstanden, und sie hätten es lieber gesehen, wenn Sun von IBM übernommen worden wäre, weil man mit IBM mehr Erfahrung im Open-Source-Bereich hat.

Bei der Sun-Übernahme geht es jedoch nicht nur um Software, sondern auch um Hardware. Letztere betrachte ich als größtes Risiko der Übernahme, wobei es im Falle eines Schieflaufens lediglich zu einer erwünschten Bereinigung des Hardware-Marktes kommt. Wenn Oracle Open Source als Go-to-Market-Modell versteht und richtig mit der Software umgeht, ergeben sich durchaus große Chancen. Im Endeffekt muss sich Oracle daran messen lassen, ob der Anwender künftig zufriedener ist als in den Jahren zuvor.

Was erwartet die Open-Source-Gemeinde von der Firma Oracle?

Seibt: Die Entwickler erwarten mit Sicherheit, dass Oracle die übernommenen Open-Source-Projekte genauso oder noch mehr unterstützt, als das zuvor der Fall war. Die Open-Source-Business-Gemeinde wird sicherlich ein professionelles Open-Source-Angebot schätzen.

Viele Java-Technologien im Umfeld von Sun stehen bisher unter der „Common Development and Distribution License“ (CDDL). Können Sie sich vorstellen, dass sich das unter dem Dach von Oracle ändern wird?

Seibt: Ich bin jetzt keine Lizenz-Spezialist, aber Sun hat sich sicher viele Gedanken über die Wahl der CDDL gemacht, um im Sinne des Kunden das Beste zu ermöglichen.

Der Java-Community-Process steht stark in der Kritik. Wie würde für Sie eine dem Open-Source-Gedanken angemessene Steuerung der Java-Technologien aussehen?

Seibt: Das hängt sehr eng mit der Eclipse-Foundation zusammen. Deren Modell gibt

ein gutes Beispiel für Oracle ab. Ich würde mich freuen, wenn Oracle diesen Weg einschlägt und eng mit der Eclipse Foundation zusammenarbeitet.

Sun war mit Open Source nur mäßig erfolgreich. Was macht Oracle hier nach Ihrer Einschätzung anders? Was waren Suns größte Fehler?

Seibt: Sun ist immer eher ein Unternehmen für Hardware als für Software gewesen. Ich bin der festen Überzeugung, dass ein Unternehmen nur eine „DNA" haben kann. Ein hardware-orientiertes Unternehmen neigt dazu, die Software als Vertriebsunterstützung für seine „Ur-DNA" einzusetzen und das konnte bei Sun nicht gut gehen. Mit Jonathan Schwartz kam bei Sun zwar ein Software-Spezialist an die Spitze; er konnte aber in diesem Punkt wenig ändern, da die Firma viel zu sehr von den Hardware-Umsätzen abhängig war. Oracle sollte sich deshalb gut überlegen, ob für die Sun-Hardware überhaupt noch ein Platz im Markt ist. Von daher sollte Oracle sich an der eigenen „Software-DNA" orientieren und nicht den Fehler machen, alles anbieten zu wollen. Das würde auch nicht dem Zeitgeist entsprechen.

Erwarten Sie, dass Oracle die Innovationsgeschwindigkeit von Java erhöht?

Seibt: Zunächst muss ich sagen, dass die Innovationsgeschwindigkeit von Java in den letzten Jahren schon sehr hoch war. Java hat sich zu einem vorherrschenden Standard entwickelt. Oracle könnte allerdings durch richtige Strategien und entsprechende Investments die Innovationsgeschwindigkeit von Java noch erhöhen, beispielsweise durch eine gute Zusammenarbeit und Integration in die Eclipse-Foundation.

Wo sehen Sie die Zukunft von Java?

Seibt: Java wird weiterhin der Standard in der Software-Industrie bleiben.

Welche Rolle spielen hier Interessenvertretungen wie die DOAG beziehungsweise die Java-Usergroups?

Seibt: Die Usergroups sind gegründet worden, um die Interessen der Anwender gegenüber dem Hersteller zu vertreten. Eine gute Interessenvertretung ist aber auch für den Hersteller wichtig, denn nur so kann er gezielt erfahren, was die Anwender bewegt. Ich kann deshalb jedem Hersteller nur empfehlen, auf die Anwendergruppe zu hören.

Oracle fährt jetzt im Bereich der Application Server (WebLogic, GlassFish) und bei den IDEs (JDeveloper, Netbeans, sowie Open-Source-Contributions zu Eclipse) eine zweigleisige Strategie. Welche Geschäftsstrategie erwarten Sie und welche positiven und negativen Effekte könnten sich daraus für die Java-Gemeinde ergeben?

Seibt: Ich kann Oracle nur raten, sich durch eine klare Kommunikation und entsprechendes Handeln auf eine eindeutige Strategie festzulegen. Welche das ist, müssen Fachleute entscheiden, beispielsweise die in der Eclipse-Community.

Soll Oracle OpenOffice an die Open-Source-Gemeinde abgeben, wie in einigen Beiträgen im Internet gefordert?

Seibt: OpenOffice spielt eine wichtige Rolle im Markt. Ich finde, dass Sun mit dem Produkt richtig umgegangen ist. Ich glaube allerdings, dass man OpenOffice innerhalb der Eclipse-Foundation als gesondertes Projekt ansiedeln sollte.

Welche Bedeutung hat Open Source in der Zukunft?

Seibt: Open Source ist für mich ein Go-to-Market-Modell, das es Software-Unternehmen erlaubt, in kürzerer Zeit im Markt erfolgreich zu sein. Das kann man daran sehen, für welche hohen Summen kommerzielle Open-Source-Unternehmen wie XenSource, Zimbra oder SpringSource gekauft worden sind. Open Source hat daher für mich die Bedeutung, Unternehmenswerte schnell und ohne großes Investment zu entwickeln. Beispiele, wie zusätzlich zu Open-Source-Projekten erfolgreiche kommerzielle Open-Source-Unternehmen entstehen, sind Zend mit PHP, EnterpriseDB mit PostgreSQL oder Suse/Novell mit Linux; hier haben diese Unternehmen die Verantwortung für Business Critical Readiness, Vertrieb, Marketing und Support übernommen. Gleichzeitig sehe ich den Trend, dass Unternehmen das Open-Source-Modell anwenden, um ihre Close-Source-Software noch erfolgreicher im Markt zu platzieren, wie zum

Beispiel Oxid-eSales. Einen weiteren Trend sehe ich in der konsortialen Software-Entwicklung, bei der Konsortien – teilweise sogar aus konkurrierenden Unternehmen – gemeinsam die Entwicklung einer Anwendung übernehmen. Die entwickelte Lösung wird unter eine Open-Source-Lizenz gestellt und steht somit allen Mitgliedern des Konsortiums und auch der Allgemeinheit frei zur Verfügung. Ein Beispiel dafür ist die Eclipse-Foundation selbst, bei der die Mitglieder etwa achtzig Prozent gemeinsam entwickeln und die restlichen zwanzig Prozent individuell hinzufügen, oder die Prometheus Foundation, ein Zusammenschluss aus Versicherungsunternehmen und freien Maklerverbünden.

Oracle legt Wert darauf, Open Source nicht mit „gratis" gleichzusetzen. Rechnen Sie damit, dass Open Source kostenintensiver wird?

Seibt: Open-Source-Lösungen verursachen heute schon Kosten. Ich glaube, dass der Einsatz von Open Source kostengünstig bleiben wird, da im Go-to-Market-Modell weniger für Marketing und Vertrieb aufgewendet werden muss. Sobald beim Kunden eine Open-Source-Lösung installiert ist, hat es der Hersteller leichter, eine „Versicherung" zu verkaufen, wenn der Kunde diese Lösung geschäftskritisch einsetzen möchte.

Welchen Weg wird Oracle Ihrer Meinung nach bei Open Source einschlagen?

Seibt: Ich vermute, dass Oracle diesen Weg beibehält, da das Unternehmen damit bessere Ergebnisse erzielen wird.

Wie soll die Zukunft von MySQL aussehen?

Seibt: MySQL zeichnet sich dadurch aus, dass ein Independent Software Vendor sehr einfach auf dessen Basis Anwendungen entwickeln kann. Oracle kann sich auf diesem Weg neue, lukrative Geschäftsfelder erschließen.

Wie sehen Sie die Rolle der Oracle-Wettbewerber wie IBM im Open-Source-Umfeld?

Seibt: IBM hat sich schon lange im Open-Source-Umfeld engagiert. Daran wird sich auch durch die Übernahme von Sun durch Oracle in Zukunft nicht viel ändern.

Welches Unternehmen sollte Oracle Ihrer Meinung nach als Nächstes kaufen?

Seibt: Oracle sollte nach Lösungen suchen, die bereits heute auf Cloud Computing basieren. Ich sehe Lücken im stärksten Wachstumsfeld „Software as a Service". Bereiche wie beispielsweise „Corporate Groupware" oder „Corporate IP-Telefonie" eignen sich besonders, da dort ein sehr starkes Wachstum zu erwarten ist. Deshalb vermute ich, dass sich Oracle hier als nächstes engagieren wird. Allerdings gibt es da nicht mehr sehr viele frei agierende Unternehmen auf dem Markt.

Was wünschen Sie sich in der Zukunft für das Open-Source-Business …?

Seibt: Wir sollten hier in Europa die Bedeutung von Software besser einschätzen. Es gibt sowohl im privaten als auch im geschäftlichen Bereich kaum noch Dinge, die nicht von der Software dominiert sind, beispielsweise das Auto oder ein mobiles Kommunikationsgerät. Damit hängt unsere Wettbewerbsfähigkeit stark vom richtigen Umgang mit der Software ab. Deshalb wünsche ich mir, dass im politischen Bereich die Förderung gestärkt wird und dass mehr Kapital im Gründungsbereich zur Verfügung steht, um gute Software-Unternehmen aufzubauen.

… von Oracle …?

Seibt: Ich hoffe, dass Oracle die Kunden in den Vordergrund stellt und dass die Business-Lösungen noch leistungsfähiger werden.

… und vom iJUG?

Seibt: Hier wünsche ich mir, dass der iJUG immer das richtige Händchen hat, um die Forderungen der Kunden zu bündeln und an Oracle zu kommunizieren, und dass Oracle diese auch versteht und entsprechend realisiert.

Zur Person:

Richard Seibt

Richard Seibt (Jahrgang 1952) hat einen Abschluss als Diplom-Kaufmann in Betriebswirtschaft an der Universität Hamburg und verfügt über 30 Jahre internationale Geschäftserfahrung in der IT-Industrie. Er begann seine Berufskarriere im Vertrieb der IBM in Stuttgart, gefolgt von verschiedenen Management-Positionen auf dem Gebiet der Business Development, im Vertrieb und Marketing, zunächst in Deutschland, später auch in anderen europäischen Ländern. Als Geschäftsführer von IBM Deutschland war er verantwortlich für den Vertrieb und das Marketing der IBM Software-Produkte in Deutschland, Österreich, der Schweiz und Osteuropa; später als Vice-President Software Sales and Marketing auch für Nordamerika. Danach war er als General Manager bei IBM für das Betriebssystem OS/2 verantwortlich.
1998 wurde Richard Seibt Mitglied des Management Boards der United Internet AG, im Januar 2003 CEO von SuSE Linux AG. Nach der Übernahme von SuSE durch Novell wurde er im Februar 2004 President von Novell EMEA.
Seit Mai 2005 ist Richard Seibt Privatinvestor, er gründet und führt als CEO die Open Source Business Foundation (OSBF). Darüber hinaus ist er im Aufsichtsrat verschiedener Software-Unternehmen tätig, darunter OpenXchange AG, Nürnberg; Likewise Software Inc., Seattle/USA; eZ Systems AS, Skien/Norwegen; IISY AG, Würzburg; Sopera GmbH, Bonn; Zimory GmbH, Berlin; Exasol AG, Nürnberg, Knowis AG, Regensburg; visionapp AG, Eschborn.

Vor dem Hintergrund der Sun-Übernahme durch Oracle stellten sich Günther Stürner, Vice President Server Technologies und Sales Consulting bei der ORACLE Deutschland B.V. & Co. KG, und Bernd Rintelmann, Leiter der Business Unit Middleware bei der ORACLE Deutschland B.V. & Co. KG, den Fragen der iJUG-Mitglieder.

„Sun ist als Marke wertvoll, sie stand und steht für Innovation, Geschwindigkeit und IT-Kultur erster Güte …"

Sieht sich Oracle zukünftig als Komplettanbieter für Server, Storage, Middleware, Datenbank und Applications?

Stürner: Eindeutig ja. Oracle bietet zusammen mit dem Sun-Portfolio die gesamte Palette an Produkten, Lösungen und Services aus einer Hand. Ein Kunde kann alles von uns beziehen. Aber wir werden auch weiterhin alle wichtigen Plattformen, Betriebssysteme und Virtualisierungstechnologien unterstützen. Die Verfügbarkeit unserer Software auf unterschiedlichen Plattformen ist unsere Stärke, die wir beibehalten und sogar noch ausbauen wollen.

Wird es zukünftig auch Komplettsysteme wie Exadata für Applicationsprodukte geben?

Stürner: Die Datenbankmaschine mit der Exadata Storage-Technologie ist ein erster wichtiger Schritt, alle „State of the Art"-Technologien im Oracle-Umfeld zu bündeln und den Kunden zur Verfügung zu stellen. Das Ergebnis ist die schnellste und kosteneffizienteste Datenbank-Plattform, die es heute gibt. Dies wurde durch eine enge Kooperation unterschiedlicher Engineering-Gruppen erreicht. Überall dort, wo sich diese enge Kooperation positiv auswirken kann, werden wir zukünftig viel stärker – weil viel einfacher – unser Augenmerk bei der Entwicklung von Software und Systemen darauf legen können. Dies klingt etwas allgemein und beantwortet nicht ganz die Frage, soll aber zeigen, dass wir in diesem Bereich noch viel Potenzial sehen.

Wird die Marke „Sun" erhalten bleiben?

Stürner: Sun ist als Marke wertvoll, sie stand und steht für Innovation, Geschwindigkeit und IT-Kultur erster Güte. Es wäre fahrlässig, diese Marke nicht weiter zu nutzen.

Wie wird Sun hinsichtlich Vertrieb und Entwicklung bei Oracle integriert?

Stürner: Sun wird natürlich in eine Gesamtunternehmung integriert; es soll jedoch zu keiner sinnlosen Vermischung kommen. Wir haben nun ein vollständiges Produkt-Portfolio, aber es gibt auch Einzelkomponenten, die außerhalb des Portfolios ihren Platz im Markt haben. Unsere Datenbank sowie unsere Middleware-, BI-, IDM- und Applikations-Produkte laufen auch weiterhin auf HP-, IBM- oder FTS-Systemen. Unser Anspruch ist, die Kunden bestmöglich zu beraten und die optimale Lösung für ihre IT-Aufgaben zu finden.

Wird der Content der Sun-Website erhalten bleiben?

Stürner: Die Sun-Website wird über die Zeit in eine gemeinsame Oracle/Sun-Website überführt. In den Ländern, in denen die beiden Firmen bereits verschmolzen sind, ist die lokale Sun-Website auf die Oracle-Website verlinkt. Für Deutschland heißt das, dass die Seite „www.sun.de" bis zur Verschmelzung so, wie sie heute ist, existiert.

Günther Stürner, ORACLE Deutschland B.V. & Co. KG

Bernd Rintelmann, ORACLE Deutschland B.V. & Co. KG

Wie ist die Produktstrategie mit Sun hinsichtlich Middleware beziehungsweise beim Application-Server?

Rintelmann: Die primäre Applikations-Server-Infrastruktur innerhalb der Fusion Middleware bleibt weiterhin die Oracle WebLogic Suite. Diese Lösung beinhaltet die WebLogic Server Enterprise Edition, JRockit Real Time, Enterprise Manager Diagnostic Pack und Enterprise Manager for Coherence. Der GlassFish-Server ist jetzt Bestandteil dieser Suite.

Wird der GlassFish-Server weiterhin die Referenz-Implementierung für Java EE bleiben?

Rintelmann: Oracle hat ein klares Commitment zur Weiterführung von Java EE und der Bereitstellung der Referenz- Implementierung für Java EE gegeben. GlassFish ist – und wird es auch in Zukunft sein – die Referenz-Implementierung für Java EE.

Wird GlassFish weiterhin Open Source bleiben?

Rintelmann: Hier gibt es ebenfalls ein klares Commitment von Oracle, dass GlassFish weiterhin als Open-Source-Produkt entwickelt wird.

Wird GlassFish auch als kommerzielles Produkt für Neu-Kunden verfügbar sein?

Rintelmann: GlassFish wird ebenfalls als kommerzielles Produkt „GlassFish Server Standalone" den Kunden zur Verfügung stehen. Darüber hinaus wird der GlassFish-Server Bestandteil der Middleware-Produkte „WebLogic Server Standard Edition", „WebLogic Server Enterprise Edition" und „WebLogic Suite" sein.

Erhalten Kunden, die bereits heute den GlassFish-Server in Produktion einsetzen, weiterhin Support?

Rintelmann: Wie bei den vorausgegangenen Akquisitionen hat Oracle klar zugesagt, die Produktlinien der übernommenen Firmen weiter zu unterstützen. Genauere Informationen dazu sind in der „Oracle Lifetime Support Policy" auf der Oracle-Website dokumentiert.

Wird Oracle Lizenz-Änderungen vornehmen, die den Download, die Entwicklung und das Deployment von GlassFish beeinflussen?

Rintelmann: GlassFish kann weiterhin nach den bestehenden Lizenz-Bestimmungen von „java.net" benutzt werden. Die kommerziellen GlassFish-Lizenzen und die damit verbundenen Downloads werden an die Oracle Lizenz-Richtlinien angepasst.

Gibt es eine gemeinsame technische Infrastruktur zwischen dem WebLogic-Server und GlassFish?

Rintelmann: Oracle liefert bereits heute die Open Source Persistenz-Infrastruktur „EclipseLink" sowohl für den WebLogic-Server als auch als Referenz-Implementierung innerhalb von GlassFish. Außerdem nutzt Oracle die JAX-WS Web-Service-Referenz-Implementierung als Web-Service-Lösung innerhalb des WebLogic-Servers. Weitere gemeinsam zu nutzende Infrastruktur-Bereiche werden zurzeit evaluiert.

Welchen Applikations-Server sollten die Kunden für das Deployment von Fusion Middleware nutzen?

Rintelmann: Oracle zertifiziert den WebLogic-Server für die Deployments der Fusion Middleware.

Welchen Applikations-Server sollten Kunden als Produktions- Umgebung von Java-EE-Anwendungen nutzen?

Rintelmann: Oracle empfiehlt WebLogic Server für das Deployment von Java-EE-Anwendungen innerhalb der Produktion. Kunden können allerdings auch den GlassFish-Server für das Deployment bestimmter Anwendungen in eine produktive Umgebung nutzen. In jedem Fall steht ihnen Oracle Support unter Berücksichtigung der „Oracle Lifetime Support Policies" zur Verfügung.

Können Kunden auf GlassFish entwickeln und die Anwendungen auf den WebLogic-Server einsetzen?

Rintelmann: Oracle ist dabei, Werkzeuge zu entwickeln, die den Kunden eine einfache Verschiebung der Anwendungen von GlassFish auf WebLogic ermöglichen.

Wie ist die Weiterentwicklung des GlassFish-Servers nach dem Release 3.0?

Rintelmann: Oracle wird GlassFish als Referenz-Implementierung für Java EE weiter-

entwickeln und in zukünftigen Releases sowohl Änderungen des Standards als auch Anforderungen der GlassFish-Server-Community berücksichtigen.

Gibt es einen Unterschied zwischen der kommerziellen und der Open-Source-Version von GlassFish?

Rintelmann: Die kommerzielle Version basiert auf der Open-Source-Code-Linie, bietet aber im Vergleich zur Basis zahlreiche Erweiterungen wie Hochverfügbarkeit – aktuell durch ein Produkt namens „HADB", in den zukünftigen 3.x-Releases wird Oracle Coherence integriert –, bessere Load-Balancing-Fähigkeiten, mehr Management- und Monitoring-Fähigkeiten – bestimmte Erweiterungen der Management-Konsole stehen nur in der kommerziellen Version zur Verfügung – sowie Support und Patches. Oracle plant für die Zukunft, bei neuen Releases diese Unterscheidungsmerkmale fortzuführen.

Welche Strategie verfolgt Oracle für Java SE?

Rintelmann: Oracle steht zur Zukunft von Java unter der Mitwirkung einer dynamischen und mitbestimmenden Community bezogen auf Java SE und investiert deshalb mit großem Engagement in die Java-SE- und Java-VM-Technologie.

Welche Pläne hat Oracle hinsichtlich des Java Community Process, der Community selbst und der Plattform?

Rintelmann: Oracle bekennt sich eindeutig zu offenen Standards und dem aktiven Engagement mit der Community von Entwicklern und Herstellern. Der Java Community Process, die Gemeinschaft der Entwickler und der offene Charakter der Plattform sind die wichtigsten Stärken von Java. Hier wird Oracle in Zusammenarbeit mit der Community kurzfristig die Pläne für das weitere Vorgehen erstellen.

Wird Java weiterhin frei zur Verfügung stehen?

Rintelmann: Oracle wird weiterhin Java SE (JDK und JRE) als freien Download zur Verfügung stellen.

Wird Oracle auch weiterhin Java-SE- und JVM-Referenz-Implementierungen bereitstellen?

Rintelmann: Ja, die Sun JDK-Implementierung und die HotSpot-JVM werden weiterhin Referenz-Implementierungen für Java SE und die Java Virtual Machine bleiben.

Welche Pläne hat Oracle mit JDK7 und hinsichtlich der Zukunft von Java SE?

Rintelmann: Oracle wird die aktuelle Entwicklung des JDK7 fortsetzen und innerhalb des Java Community Process an der Java-SE7-Spezifikation arbeiten.

Was ist die Oracle-Strategie bezogen auf die JVM?

Rintelmann: Sowohl die HotSpot- als auch die Jrockit-VM bleiben als strategisches Produkt erhalten und werden unterstützt, weiterentwickelt sowie für den Einsatz mit WebLogic Server und Fusion Middleware zertifiziert.

Wird NetBeans den JDeveloper als strategische IDE ersetzen?

Rintelmann: Nein, der JDeveloper wird weiterhin die „End to End"-Entwicklungsumgebung für unternehmensweite Anwendungen bleiben, einschließlich der Unterstützung für Java, Java EE, XML, Portal- und SOA-Entwicklung, Prozess-Orchestrierung und Datenbank-Entwicklung. Net-Beans wird dagegen die Wahl sein für Entwickler, die eine Open-Source-Umgebung mit Unterstützung für Java SE/EE/Mobile/FX/TV suchen. Der Kunde hat die Wahl, bezogen auf seine Anforderungen. NetBeans stellt für Oracle einen wichtigen Teil des Java-

Zur Person:

Günther Stürner

Günther Stürner ist Vice President Server Technologies und Sales Consulting bei der ORACLE Deutschland B.V. & Co. KG. In dieser Funktion ist er verantwortlich für die Bereiche Datenbank und Middleware.
Günther Stürner arbeitet bereits seit September 1985 für Oracle; seinen Werdegang begann er als Sales-Consultant Deutschland. Von 1987 bis 1993 widmete er sich dem Aufbau der Abteilung STU (Strategisch Technische Unterstützung) und bekleidete anschließend für sechs Jahre die Position des Leiters des Oracle SAP Competence Centers sowie des Oracle SAP Entwicklungszentrums. 1980 begann Günther Stürner seine berufliche Karriere als Software-Entwickler bei IKOSS. Nach drei Jahren wechselte er in gleicher Position zu Alcatel und blieb dort bis zu seinem Wechsel zu Oracle im Jahre 1985.
Günther Stürner studierte Physikalische Technik und hat bereits mehrere Fachbücher zu Oracle Datenbank und SQL sowie diverse Fachartikel veröffentlicht.

Ecosystems dar und wird auch weiterhin als Open-Source-Produkt unter der Verantwortung von „netbeans.org" bleiben.

Beeinflusst die Sun-Aquisition das Oracle-Engagement für Eclipse?

Rintelmann: In keiner Weise. Eclipse ist eine wichtige IDE-Plattform innerhalb der Java-Community und es ist absolut notwendig, dass Oracle hier weiterhin unterstützt. So liefert Oracle beispielsweise Support in Form des „Oracle Enterprise Packs for Eclipse", das Add-ins für die Basis Eclipse-IDE bereitstellt, damit Entwickler mit Oracle-Technologien wie dem WebLogic-Server und der Oracle Datenbank produktiver arbeiten können. Zusätzlich investiert Oracle in Eclipse-Projekte wie EclipseLink, der JPA 2.0 Referenz-Implementierung.

Wie ist die künftige Strategie von Oracle hinsichtlich Open Source?

Stürner: Open Source ist für Oracle nichts Neues und die Übernahme von Sun wird diese Strategie nicht ändern. Seit vielen Jahren sind wir hier aktiv, haben einige Open-Source-Produkte im Portfolio und sind in der Open-Source-Community sehr aktiv. Man hatte bei der Diskussion in den letzten Monaten den Eindruck, als wäre das ein Bereich, den wir meiden wie der Teufel das Weihwasser. Dabei haben wir schon vor Jahren die Berkeley DB übernommen, mit Inno DB haben wir die wichtigste Storage-Engine von MySQL seit Jahren weiterentwickelt und mit ACFS eine wichtige Technologie im Bereich Cluster-Filesysteme als Open Source zur Verfügung gestellt. Das Gleiche gilt für TopLink, der führenden Persistenz-Technologie, oder für Eclipse und NetBeans im Bereich Entwicklungswerkzeuge sowie Apache oder EcpliseLink – nicht zu vergessen GlassFish im Bereich Middleware.

Wie ordnet Oracle MySQL in sein Produkt-Portfolio ein?

Stürner: MySQL hat sich in den letzten Jahren einen wichtigen Platz innerhalb der IT-Landschaft erworben. Wir sehen darin jedoch keine interne Konkurrenz, sondern vielmehr eine gute und interessante Ergänzung zu unserem bisherigen Datenbank-Angebot.

Rintelmann: Oracle Fusion Middleware unterstützt eine Vielzahl von Datenbanken – neben dem Oracle RDBMS unter anderem auch Microsoft SQL Server, IBM DB/2 und MySQL. Wir haben bereits heute zahlreiche Kunden, die Fusion Middleware in Verbindung mit MySQL einsetzen. Diese Open-Source-Datenbank kann für OLTP-Anwendungen verwendet werden, unterstützt allerdings keine Data Marts oder Data-Warehousing-Features, so dass momentan die BI-Komponenten der Fusion Middleware keinen MySQL-Support liefern.

Werden die regelmäßigen Patches für die Oracle Datenbank und für Sun in Zukunft synchron sein?

Stürner: Ob wir zu einer Synchronisation von Patches kommen werden, kann ich zum heutigen Zeitpunkt nicht sagen, es ist aber sehr wahrscheinlich. Unstrittig ist es jedoch, Patch-Strategien zu entwickeln, die es den Kunden vereinfachen, ihre Systeme zu managen und zu betreiben. Dies ist sicherlich auch eine Art Test dafür, ob die Integration auf Engineering-Ebene erfolgreich war oder nicht.

Wie ist die Zukunft von Open Office und Solaris?

Stürner: Solaris wird von vielen als das beste Unix-System am Markt bezeichnet. Sowohl Solaris/Sparc als auch Solaris/x86 sind für uns wichtige Software-Komponenten für die Zukunft. Ähnlich verhält es sich mit Open Office. Dieses ausgezeichnete Produktset wird eine große Zukunft haben.

Zur Person:

Bernd Rintelmann

Bernd Rintelmann leitet als Direktor Middleware Technology seit August 2007 die Business Unit Oracle Middleware bei der ORACLE Deutschland B.V. & Co. KG. Er ist für alle technologischen Belange des Oracle Middleware Stacks verantwortlich – von der Produkteinführung über „Proof-of-Concepts" bis hin zu „Best-Practices" und Benchmarks.
Bernd Rintelmann arbeitet seit mehr als 15 Jahren bei Oracle und hatte in dieser Zeit verschiedene Managementpositionen inne. Unter anderem war er in der Systemberatung tätig, arbeitete zwei Jahre lang in der Produktentwicklung bei der Oracle Corporation in den USA und betreute zuletzt als Chief Architect im Bereich Middleware den Themenkomplex „Service Orientierte Architekturen" (SOA) mit Fokus auf Web Services, Integration und Geschäftsprozesse bei der ORACLE Deutschland GmbH. Vor seinem Wechsel zu Oracle war Bernd Rintelmann als System- und Software-Entwickler in der Automobilzulieferindustrie tätig, unter anderem bei der Continental AG.

„The Sun we knew is gone." Das einprägsame Zitat des Bloggers John Gruber fasst auf brillante Art die Stimmung zusammen, die nach der offiziellen Verlautbarung der Übernahme von Sun durch Oracle in der Java-Welt herrschte. Mehrere Monate danach mag sich noch immer kein rechtes Wohlgefühl einstellen. Auch wenn die rechtliche Integration einiger Landesgesellschaften noch immer nicht abgeschlossen ist, so bewegt sich das neu hinzugekaufte Sun-Universum merklich in Richtung seines neuen Eigners.

Oracle und die Zukunft von Java

Markus Eisele, msg systems ag

Am sichtbarsten wurde die Übernahme sehr zeitnah durch das Rebranding fast aller bestehenden „sun.com"-Seiten. Aus Blau wurde vielfach Rot und ein neues, zumeist sehr kleines Oracle-Logo prangte auf den Seiten. Ähnlich weitreichende Aktionen hätte man nach der fulminanten Serie von Webcasts im Januar an vielen Stellen erwartet – aber weit gefehlt: Bis auf die prominenten Abgänger Schwarz und Gosling gab es kaum wahrnehmbare Veränderungen. Oracle arbeitet anders, Oracle agiert still. Das ist auch der Grund, aus dem noch nicht viel mehr bekannt wurde über die Strategien oder die neue Ausrichtung aller mit Java befassten Produkte, Standards oder Themen aus dem Sun-Portfolio. Dieser Artikel liefert eine aktuelle Positionsbestimmung und gibt einen kleinen Ausblick auf die kommenden Monate und Jahre, in denen Java seine Wurzeln im neuen Unternehmen schlagen soll. Die Informationen entstammen einer Vielzahl von Quellen, im Zentrum steht die offizielle Stellungnahme aus den Oracle/Sun-Merger-Webcasts [1]. Die Vision liefert der Autor, sie erhebt keinerlei Anspruch auf Belastbarkeit.

Die Sprache Java

Es ist schon verrückt, Java SE 6 ist noch immer aktuell. Schon seit mehr als drei Jahren hat sich an der Sprache selbst nichts Nennenswertes getan. Das Warten auf das JDK 7 bleibt spannend. Erst Mitte April wurde JDK 7 M7 vorgestellt. Mit dem Milestone Build 8 sollte die neue Version „Feature Complete" sein. Seit dem 3. Juni 2010 ist er auch vorhanden, allerdings nicht vollständig, wie angekündigt. Aber eigentlich hat die Version 7 auch schon eine Menge an Schwung eingebüßt. Die Diskussionen um das Für und Wider von Closures in der Sprache haben viel Energie gekostet und zu einer gehörigen Verschiebung im Zeitplan geführt. Ein JDK 7 ohne Closures könnte vermutlich recht schnell fertig sein. Aber dazu wird es wohl nicht kommen. Daher bleibt es spannend. Sowohl in der Diskussion, die noch immer andauert, als auch bei der Zeitplanung: Bis Ende dieses Jahres soll alles fertig sein.

Die kommerzielle Seite der Sprache scheint klar. Oracle steht zur Zukunft von Java unter Mitwirkung einer dynamischen und mitbestimmenden Community, bezogen auf Java SE, und investiert mit großem Engagement. Für Oracle-Produkte bleibt sowohl HotSpot als auch JRockit JVM die strategische Wahl. Beide sollen weiterentwickelt und für den Einsatz im Fusion-Middleware-Stack zertifiziert sein und bleiben. Die Basis für die aktuell vorgestellten Virtualisierungslösungen ist allerdings einzig JRockit. Vermutlich wird es also mittelfristig zu einer gemeinsamen Codebase der beiden Produkte kommen und diese werden sich nur noch über einzelne Erweiterungen beziehungsweise Module voneinander unterscheiden.

Scripting in der JVM

Neben vielen anderen Themen [2] ist die wichtigste Erweiterung im JDK 7 die Integration von mehr als 200 Sprachen auf der JVM. Nach dem „Ruby / jRuby"-Hype 2007 und dem „Python / Groovy"-Hype 2008 ist hier vor allem Scala das „Must Have"-Thema der Entwicklergemeinde. Inwieweit insbesondere Scala ein Erweiterung oder ein Ersatz für Java selber ist, bleibt hier unbewertet. Aktuell ist nur festzustellen, dass vor allem die Leichtgewichtigkeit von Scala den Reiz ausmacht. Auch wenn man für viele Aufgaben noch Java-Bibliotheken benötigt, ist klar, dass es selber nicht die Last von mehr als 14 Jahren Entwicklung und den Zwang der Rückwärtskompatibilität tragen muss und daher auch leichtgewichtiger sein kann. Bei all der aktuellen Begeisterung darf man dabei nicht vergessen, dass die Mehrsprachenfähigkeit für Oracle kein zentrales Element darstellt. Neben kleineren Python- und Groovy-Umfängen sind bisher keine Scala-Umfänge in Oracle-Produkten bekannt geworden. Streng genommen kann es auch nicht im Sinne eines Produktherstellers sein, eine möglichst heterogene Landschaft aufzubauen. Daher wird Oracle mittelfristig vermutlich vergleichsweise wenig tun, um die Vielsprachigkeit zu unterstützen. Hier wird über kurz oder lang die Community einspringen müssen, sofern Oracle das in ausreichendem Maße ermöglicht.

Java Community Process (JCP)

Der JCP [3] ist das Herzstück von Java. Dahinter steht eine Gemeinschaft von Entwicklern und Herstellern mit dem Ziel der konstruktiven Weiterentwicklung von Java und der Standards im gesamten Ökosystem. Irgendwie hat man dem „Low-Profit"-Unternehmen Sun auf diesem Weg immer mehr oder minder vertraut und sich der Führung durch Sun ergeben. Gegen geringste Versuche von Sun, Profit zu machen, wurde aber lauthals protestiert. So hat die Apache Software Foundation (ASF) lange Zeit viele Java Specification Requests (JSR) mit „Nein" kommentiert, weil ihr keine kostenlosen Testsuiten (TCK) zur Verfügung standen. Sun hat diskutiert, geblogged und Mails geschrieben. Geändert hat sich an der Situation bis zuletzt wenig. Oracle ist nicht bekannt für die öffentliche Diskussion von Problemen.

Es wird also ruhiger werden um den JCP. Aktuell hat Oracle die von Sun freigegebenen Plätze in den jeweiligen Komitees übernommen. Für die damit von Oracle freigemachten Positionen wurden wiederum neue Abstimmungen über Nachrücker durchgeführt. Auch wenn viele Stimmen der vergangenen Monate (Blogger, Presse, Insider, Unternehmen) immer wieder eine grundlegende Überarbeitung des JCPs gefordert haben, ist damit wohl nicht zu rechnen. Wenn hier ein Umbau stattfindet, dann auf alle Fälle einer, der an den aktuell bestehenden Entscheidungsprozessen nichts verändert, was einer Schlechterstellung von Oracle gleichkäme.

Java Communities

Blogs, Twitter Accounts, Foren, Developer-Seiten, User Groups, Kenai und vieles mehr – Sun hat eine schier endlos lange Liste von Interaktionsmöglichkeiten für Java-Anwender geboten. Im ersten Schritt tut Oracle sicherlich gut daran, hier großflächig nichts zu verändern. Auch wenn schnell und konsequent neue Social-Media-Guidelines griffen und eine Menge Unsicherheit unter den Sun-Bloggern herrschte, es hat sich aktuell doch alles wieder eingeschwungen. Alle gewohnten Anlaufstellen im Netz sind noch vorhanden und so sollte es auch sein. Die Anwender-Community wird jetzt vom Oracle Technology Network (OTN) betreut, Infrastrukturen wurden zusammengelegt (blogs.sun.com und blogs.oracle.com) sowie auch anderweitig konsolidiert.

Brandaktuell wurden auch weitere Teile der Sun-Dokumentation und Downloads auf download.oracle.com umgezogen. Nachdem das Konsolidieren bis dato vergleichsweise wenig von der Community kommentiert wurde, regt sich hier erstmals mehr Wiederstand. Neben dem Fehlen von einprägsamen URLs ist auch die mangelhafte Performance der neuen Seiten ein Problem. Oracle hat angekündigt, hier Abhilfe zu schaffen. Auch die bisher gewohnten URLs der Sun Java APIs kommen gegebenfalls wieder.

Die Sun User Groups sollen nach dem Willen von Oracle mittelfristig in die International Oracle User Group Community (IOUC) aufgenommen werden, in der bereits mehr als 400 Organisationen weltweit zusammengefasst sind. Es gibt also etablierte Kanäle, auf denen Oracle sich eine Kommunikation wünscht und auch ermöglicht. Rein subjektiv betrachtet geht es in den Bereichen der früheren Sun-Community formal lockerer und inhaltlich konkreter an die Themen. Bei Oracle mag einem dies umgekehrt vorkommen. Hier wird es auf beiden Seiten noch Lernprozesse geben müssen, wobei klar ist, dass Java sowohl innerhalb als auch außerhalb des Oracle-Universums gesprochen wird und sich auch große Teile der Communitys genau hier bereits überlappen.

Oracle GlassFish Open Source Server

Ein ursprünglich geplantes „100 Day"-Release des GlassFish Open Source Servers 2.x („Branding Release") wurde abgesagt. Es ist nicht wirklich nötig und die Ressourcen können besser für das 3.1-Release verwendet werden. Hier funktioniert die Flexibilität bei der Zusammenarbeit zwischen der Community und Oracle gut. Es wird also mit Hochdruck am aktuellen GlassFish Minor Release 3.1 gearbeitet. Dieses ist noch für 2010 geplant. Klar ist, dass die primäre Applikations-Server-Infrastruktur innerhalb der Oracle Produktpalette (Fusion Middleware) der Oracle WebLogic Server bleibt. Der Oracle GlassFish Server ist die kommerziell lizenzierbare und supportete Version von Oracle. Weiterhin frei unter gleichen Lizenzbedingungen wie bisher ist der GlassFish Server Open Source Edition.

Beide Versionen unterscheiden sich laut Oracle nicht. Das aktuelle Entwicklungsrelease wird zahlreiche Erweiterungen wie Clustering, Hochverfügbarkeit sowie mehr Management- und Monitoring-Fähigkeiten bieten. Über das Prinzip von Add-Ons werden bestimmte Erweiterungen nur in der kommerziellen Version zur Verfügung stehen. Auch wenn WebLogic und GlassFish mittelfristig auf der Ebene von Modulen deutlich näher zusammenwachsen werden, bleiben signifikante Unterschiede zwischen den Produkten erhalten. Wie die Zukunft konkret aussieht, weiß aktuell wohl noch niemand. Einen ersten Einblick gibt die Roadmap [4].

Java FX

Abseits der Client/Server-Umgebungen mit Java existiert noch eine andere Technologie, die aktuell sehr viel Aufmerksamkeit erregt. Java FX soll die neue Oberflächentechnologie für Rich Clients und mobile Geräte werden. Obwohl es zum Java-Kern gehört, ist Java FX noch sehr weit von einer nennenswerten Verbreitung entfernt. Die aktuelle Version 1.3 macht zwar einen robusten Eindruck, größtes Manko bleibt allerdings die fehlende Java-Unterstützung auf den Apple-Geräten (iPhone / iPod / iPad). Auch wenn Oracle nicht müde wird zu betonen, dass es sich bei Java FX um eine Zukunftstechnologie handelt, in die man investieren wolle, so ist nicht klar, wo und wie der Bedarf auf Kundenseite entstehen soll. Schlimmer noch: Die aktuell aufkommenden neuen Web-Standards (HTML 5 und CSS3) bieten bereits viele, bisher nur von Rich Clients bekannte Möglichkeiten. Damit könnten nicht nur Java FX sondern auch Flash, Silverlight & Co. schon jetzt der Vergangenheit angehören.

Netbeans vs. Eclipse vs. JDeveloper

Oracle hat nun drei Entwicklungsumgebungen (IDE) für Java – und das wird auf absehbare Zeit auch so bleiben. JDeveloper ist und bleibt weiterhin die „End to End"-Entwicklungsumgebung für unternehmensweite Anwendungen, einschließlich der Unterstützung für Java SE, Java EE, XML, Portal- und SOA-Entwicklung, Prozess-Orchestrierung und Datenbank-Entwicklung. JDeveloper bietet damit die komplette Entwicklungsunterstützung für den Fusion-Middleware-Stack. Eclipse stellt für Oracle die zweite, starke Säule mit Fokus auf Java-EE-Entwicklung dar. Neben dem Oracle Enterprise Pack for Eclipse (OEPE) sind diverse Erweiterungen beziehungsweise Plugins verfügbar, um die vereinfachte Entwicklung für Oracle-Produkte (WebLogic Server und Datenbanken) zu ermöglichen. Nicht zu vergessen steht auch die JPA 2.0 Referenzimplementierung EclipseLink auf den Säulen der Eclipse Foundation.

Ebenfalls große Aufmerksamkeit erhält die OSGi-Implementierung Equinox, auch Bestandteil von Eclipse und bereits beim GlassFish eingesetzt. Vermutlich stützt sich auch die als WebLogic DM Server angekündigte Version auf Equinox als OSGi-Implementierung. Im Großen und Ganzen besteht hier keine Gefahr, dass Oracle sein Engagement zurückfährt.

Schwieriger wird die Einschätzung der Zukunft von NetBeans. Diese Technologie soll erste Wahl sein für Entwickler, die eine Open-Source-Umgebung mit Unterstützung für Java SE / EE / Mobile /FX / TV suchen. Die vielfach zu lesende Äußerung „NetBeans wird auch weiterhin als Open-Source-Produkt unter der Verantwortung von netbeans.org bleiben“ räumt allerdings viel Raum für Spekulationen ein. Konsequent wäre es, NetBeans komplett an die Community zu übergeben oder ihr alternativ eine komplett neue Ausrichtung (beispielsweise Mobile Development) zu geben. Hier wird nur die Zeit zeigen, wie die vielfältigen, aber weichen Commitments von Oracle zu NetBeans in der Praxis tatsächlich aussehen. Im Bereich der kommerziellen Produkte ist für Oracle-Kunden eine Ausrichtung auf Eclipse beziehungsweise JDeveloper der richtige Weg.

Virtualisierung

Die Themen „Virtualisierung“ und „Cloud Computing“ sind nicht mehr brandaktuell. Aber auch im Kontext von Oracle haben sie Einzug gehalten. Mit Ausnahme der JRockit sowie der zugehörigen Version des WebLogic Servers profitieren bisher keine weiteren Produkte von diesem Ansatz. Ob es entsprechende Lösungen für die HotSpot JVM beziehungsweise den GlassFish geben wird, ist unklar. Geht man davon aus, dass beide JVMs und beide aktuellen Applikationsserver mittelfristig von jeweils einer gemeinsamen Codebase profitieren sollen, ist es nicht mehr weit, auch für diese beiden Produkte eine weitergehende Virtualisierungs-Unterstützung vorherzusagen.

Fazit

So richtig transparent ist aktuell nicht, was genau mit den Einzelteilen des Java-Ökosystems, wie man es von Sun kannte, geschehen soll. Einzig stabil ist, dass Oracle wie bei früheren Akquisitionen klar zugesagt hat, die bei Kunden im Einsatz befindlichen Produktlinien weiter zu unterstützen – egal, ob diese zukünftig noch eine Rolle für Oracle spielen oder nicht. Ebenfalls klar ist, dass Java ein integraler Bestandteil von Oracle-Produkten ist und man viel dafür tun wird, sowohl die Sprache als auch die darum angeordneten Standards weiter zu entwickeln.

Von einer umfangreichen Konsolidierung und Einordnung der Sun-Software in den Produkt-Stack von Oracle ist man aber vermutlich noch ein gutes Stück entfernt. Es hat den Anschein, als wolle man vorerst mit möglichst wenig Aufsehen und ohne größere Veränderungen Stabilität in die Java-Welt bekommen. Dabei wird es aber nicht bleiben. Auch eine Firma von der Größe Oracles kann sich nicht allen verschiedenen Initiativen mit gleicher Intensität widmen. Das ging in der Vergangenheit schon einmal daneben (BEA-Übernahme) und wird vermutlich diesmal nicht passieren. Eine weitere Detaillierung der Pläne kommt voraussichtlich im Herbst auf der JavaOne 2010.

Weitere Informationen

[1] http://www.oracle.com/events/product-strategy/index.html
[2] http://openjdk.java.net/projects/jdk7/milestones/
[3] http://www.jcp.org
[4] https://glassfish.dev.java.net/roadmap/

Kontakt:
Markus Eisele
markus.eisele@msg-systems.com

Markus Eisele arbeitet im Bereich Software-Technologie im Center of Competence IT-Architecture der msg systems ag. Bei seiner täglichen Arbeit begleitet er Kunden und Projekte auf dem Weg durch die Tücken neuer Technologien und Produkte im Java-EE-Umfeld (siehe auch Seite 30).

iJUG Verbund

Mitglieder stellen sich vor

JUG Saxony

Die Idee zur Gründung der Java User Group Saxony wurde im Dezember 2007 geboren und im Laufe des Frühjahrs 2008 von Torsten Rentsch und Falk Hartmann in die Tat umgesetzt. Schon kurz nach Einrichtung unserer Internetpräsenz stieß Kristian Rink als Nutzer der ersten Stunde zum Organisationsteam. Nach der Gründungsveranstaltung im April 2008 mit 43 Teilnehmern wächst die JUG Saxony stetig weiter.

Es sollen qualitativ hochwertige Vorträge mindestens sechs Mal jährlich stattfinden. Diese Vorträge sind kostenlos. Eine formale Mitgliedschaft gibt es in der JUG nicht, allerdings wird eine Rückmeldung benötigt, um die Raumkapazitäten zu den Veranstaltungen entsprechend planen zu können. Neben den Vorträgen steht der (Wissens-) Austausch zwischen den Teilnehmern und das Knüpfen von Kontakten im Vordergrund. Auch die Zusammenarbeit mit wissenschaftlichen Einrichtungen liegt den Organisatoren am Herzen. Die JUG Saxony steht allen Java-Interessenten offen – egal ob Anfänger oder Profi-Entwickler, Architekt,Projektleiter oder Geschäftsführer.

Weitere Informationen: http://groups.google.de/group/jug-saxony

Java ist eine Insel in Indonesien. Weniger weit gereiste Urlauber kennen die kleinere, künstlichere Ausgabe als Insel im Hafen von Amsterdam. Außerdem sind Java-Bohnen aus Äthiopien und Brasilien die Basis des Arabica-Kaffees. Doch wer schenkt den harten weltlichen Fakten zu Java Beachtung, wenn unser Interesse doch der Software gilt. Und in diesem Bereich ist die Geschichte von Java nicht minder spannend.

Die Geschichte der Bohne

Oliver Szymanski, Source-Knights.com, iJUG

Unser Java war ursprünglich eine Programmiersprache. Sun Microsystems rief sie ins Leben und trug das Wort „Java" als eingetragenes Markenzeichen ein. Im Jahr 1991 begannen 13 Entwickler im Auftrag von Sun im sogenannten „The Green Project" mit der Definition der Sprache Java. 18 Monate lang widmeten sich die Entwickler dem Object Application Kernel, kurz „Oak". Die Eiche (englisch Oak) vor dem Fenster des Hauptentwicklers James Gosling bot sich als Namenspate an. Aus rechtlichen Problemen gab man den Namen „Oak" auf und verwendete stattdessen nach der Lieblingskaffeebohnen-Marke der Entwickler das heute bekannte „Java".

Teil der damaligen Entwicklung war *7 (Star Seven), ein portabler Mikrocomputer mit Java-Software zur Steuerung von Haushaltsgeräten. Interaktion erfolgte über einen Touchscreen und Assistenten-Dialoge. Der bekannte Java-Duke war einer dieser digitalen Assistenten. Obwohl für den *7 extra die Firma First Person, Inc. gegründet wurde und das Gerät sowohl Sun als auch Vertreter der Medienbranche überzeugt hatte, erschien es nie auf dem Verbrauchermarkt. Es war seiner damaligen Zeit zu weit voraus und später gab es modernere Geräte.

Eine der wertvollsten Komponenten war Oak, der Java-Interpreter. Statt der Medienbranche und der Steuerung von Haushaltsgeräten verschaffte das Internet Java die entscheidende Gelegenheit für den Sprung nach vorn. Der Web-Runner, später „HotJava" genannt, wurde auf der Basis des Mosaic Browsers geschaffen sowie der Java-Interpreter integriert. HotJava bot damit als erster Browser die Möglichkeit, Java-Applets auszuführen, und war damit die Referenz-Plattform zum Test für Java-Entwickler. Der nächste Schritt war die Integration in den Netscape Navigator.

Die Geschichte Javas ist die Historie einer Programmiersprache, die durch die Verbindung mit Compiler, Frameworks, anderen Skriptsprachen und APIs immer mehr zu einer Technologie-Plattform aus virtueller Maschine, Java-Sprache (und weiteren auf der Maschine lauffähigen Sprachen), APIs und Programmiermodellen wurde.

1996: Java Version 1

Die erste Version von Java erschien 1996 und war hauptsächlich für die Entwicklung von Applets gedacht. Sie bot im Framework die Basispakete java.lang, java.io, java.net, java.util, java.applet sowie java.awt. Java war von Beginn an als robuste und portable, objektorientierte Programmiersprache gedacht. Die Konstrukte und APIs von Java waren Programmierern, die Erfahrung mit C++, Smalltalk und Entwurfsmustern hatten, sehr vertraut. Dies erleichterte den Einstieg in die neue Sprache.

1997: Java Version 1.1

Bereits das erste Update sah zahlreiche Neuerungen vor. Als Spracherweiterung beinhaltete es innere Klassen. Das Framework wurde erweitert um das Observer-Entwurfsmuster mit Listenern für Callbacks im Rahmen der angepassten Ereignisbehandlung, Ressourcenbündel für Internationalisierung, die heute kaum mehr wegzudenkenden Java Beans, Java-Archive, den Zugriff auf relationale Datenbanken mit JDBC, die Serialisierung von Objekten, verteilte Programmierung über Remote Methode Invocation, Sicherheitsrichtlinien und numerisches Rechnen mit BigInteger und BigDecimal.

1998: Java Version 1.2 => Java 2

Die Version 1.2 (Codename Playground) zog einige Namensänderungen mit sich. Java Version 1.2 hieß jetzt Java 2, das Java Development Kit (JDK) war fortan ein Software Development Kit (SDK). Java 2 enthielt die grafische Oberflächentechnologie Swing. Diese gab es bereits ab der Version 1.1.5, allerdings wurde die Paketstruktur für Java 2 dermaßen überarbeitet, dass einige Nacharbeit vonnöten war, wenn man ein Programm migrieren wollte. Auch Drag & Drop für Oberflächen wurde Teil des Frameworks. Eine der größten Neuerungen war der Just-In-Time-Compiler, der zwar für unterstützte Betriebssysteme eigene Laufzeitbibliotheken notwendig machte, jedoch einen Performance-Gewinn bedeutete. Selbst heute muss Java manchmal noch gegen seinen aus früheren Zeiten stammenden Ruf ankämpfen, extrem langsam zu sein.

Mit der 2D-API und neuer Audio-Funktionalität wurde Java weiter Multimediafähig. Neu waren auch die Collections, schwache Referenzen und CORBA-Schnittstellen (Common Object Request Broker Architecture, eine von der Object Management Group entwickelte Middleware in heterogenen Systemen für verteilte Anwendungen).

Im gleichen Jahr wurde auch der Java Community Process (JCP) ins Leben gerufen, über den per Java Specification Requests (JSR) Änderungen an Java vorgeschlagen und integriert werden. Über den JCP sind auch Unternehmen wie Apple, Siemens, IBM und Hewlett-Packard in die Weiterentwicklung von Java involviert.

Auch erscheint Java jetzt in Editionen, je nachdem, für welche Kategorie Software entwickelt werden soll: Java Standard Edition (Grundlage der anderen Editionen und für Desktop PC), Java Enterprise Edition (mehrschichtige/verteilte Anwendungen seit 1999), Java Card (für den Einsatz von Applets auf Chipkarten seit 2003) oder Java Micro Edition (für eingebettete Systeme wie PDAs seit 2005).

1999: Java 2 Enterprise Edition 1.0

Die Veröffentlichung der Version 1.0 der Enterprise Edition (J2EE) mit seinen APIs für Server-basierte Anwendungen auf Basis von Servlets und Enterprise Java Beans erweiterte die Java-Welt um reichhaltige Technologien beispielsweise für Web, Transaktionssteuerung, Persistenz (wenngleich die Diskussionen um CMP und BMP wohl nie endeten), Komponenten-Technologie und Sicherheit. Wobei der Autor immer noch glaubt, dass die Art der zur Verfügung gestellten Container-basierten Sicherheit in kaum einem Projekt ausreicht.

2000: Java Version 1.3 und J2EE 1.x

Mit Beginn von Java 1.3 (Codename „Kestrel“) haben die Major-Releases Säugetiere und Vögel als Codenamen, während bereits seit Release 1.2.2 Insekten bei Minor Releases für die Namensgebung herhalten und anzeigen, dass es sich um Bugfixes handelt. Neben Erweiterungen an den APIs wurde die Ausführung von Java-Programmen durch die Einführung von HotSpot drastisch gesteigert. Häufig benutzte Programmteile konnten jetzt von der Ausführungsumgebung in nativen Code kompiliert werden. Ebenfalls in diesem Jahr gab es die zwei neuen J2EE-Versionen 1.2 und 1.2.1 und im folgenden Jahr 1.3. Hinzugekommen sind unter anderem Java Messaging Service, Java Connector Architecture sowie Java API for XML Parsing. Im Rahmen des enthaltenen EJBs gab es neu die Message Driven Beans und Container Managed Persistence 2.0, nachdem die vorherige Persistenztechnologie sehr umstritten war.

2002: Java Version 1.4 und J2EE 1.4

Nachdem die Sprache selbst zuletzt in der Version 1.1 erweitert wurde, gab es in Java 1.4 (Codename „Merlin“) mit Assertions einen weiteren Eingriff in die Java-Syntax, dazu diverse Neuerungen in den Bibliotheken.

2003: J2EE 1.4 und Java Card

2003 folgte die nächste J2EE-Version, unter anderem mit vollständiger Webservice-Unterstützung, der J2EE Management API. Auch die erste Java Card Edition wurde veröffentlicht.

2004: Java Version 5 und JEE 5

Diese Version (Codename „Tiger“) änderte endgültig die Art der Versionierung. Das Präfix „1.“ verschwand und verblieb nur in der internen Sun-Nummerierung. Die Version 5 enthielt eine ganze Reihe von Neuerungen. Schwerpunkte waren Generics, Metadaten über Annotations sowie Enumerations. Angereichert wurden diese Innovationen mit Autoboxing, Syntaxänderungen bei Iteration über Collection-Typen, variable Anzahl von Methoden-Parametern, statischem Klassenimport, mehr Look&Feels, kovariante Methoden-Ergebnistypen und Hilfsklassen für Concurrency.

2005: Java Micro Edition

Die erste Version von Java für Mobilgeräte und Embedded Devices erschien. Auf Grundlage von Konfigurationen und Profilen wurde hier Software geräteneutral in Java erstellt. Sie ist aber nicht zu verwechseln mit der Java-Entwicklung für Google-Android-basierte Geräte. Aufgrund der immer leistungsfähiger werdenden Clients gab es bereits 2007 Äußerungen von Sun, eventuell die Micro Edition zugunsten der Standard Edition in den nächsten Jahren einzustellen.

2006: Java Version 6 und JEE 5

Vor vier Jahren erschien die letzte Version von Java (Codename „Mustang“). Sie enthielt mit Java DB eine auf Apache Derby basierende Datenbank, mehr Möglichkeiten hinsichtlich Management und Monitoring sowie die Integration von Skriptsprachen. Das Update 1.6.0_10 enthielt 2008 einige Erweiterungen, die nicht bis Java 7 warten sollten: minimierter Java-Kernel für Web-Anwendungen, Java Quick Starter für verringerte Startzeiten von Java-Anwendungen, bessere Applet-Möglichkeiten mit den Next Generation Java Plugins.

Die „2“ in J2EE ist ebenfalls Geschichte: Die im Jahr 2006 erschienene Version der Enterprise Edition heißt einfach JEE 5. Sie enthält als Highlights Vereinfachungen bei der Entwicklung von Komponenten über sinnvolle Defaults, die Nutzung von Annotations und POJO sowie Resource Injektion. Insgesamt betrachtet sollte die Entwicklung mit JEE einfacher werden. Die Diskussion über Persistenz beziehungsweise Bean- und Container-Managed-Persistence hat mit dem Einzug der Java Persistence API ein Ende. Jetzt können endlich POJOs, wie aus anderen ORM Frameworks bekannt, gesichert werden.

2007: Weg zu freieren Lizenzbestimmungen

Bereits ein Jahr vorher hatte Sun angekündigt, dass Java in Zukunft unter der GPL-Version 2 als Lizenz zur Verfügung gestellt werden soll, was jetzt für einen großen Teil der Technologie umgesetzt wurde. Dieser Weg wird später mit der ersten von Sun geförderten, quelloffenen Java-Implementierung OpenJDK, in deren Rahmen auch Java 7 entwickelt wird, weiter geführt.

2009: JEE 6

Mit JEE 6 erschien die bislang letzte veröffentlichte Version der Enterprise Edition. Zusätzlich zu Modifikationen an den bereits enthaltenen APIs wie Singleton-Beans und asynchronem Methodenaufruf wurden die Java API for RESTful Web Services (JAX-RS) und Java Contexts and Dependency Injection API (JCDI, früher unter dem Namen „Web Beans“ bekannt) neu integriert. Außerdem räumt JEE 6 auf: Mit Pruning können nicht benötigte Teile aus der Plattform entfernt werden, Profiles helfen, den JEE-Stack einzuschränken, und über Extensibility-Mechanismen hat man Extension Points und Service Provider In-

terfaces um Erweiterungen zum Standard einzubinden.

2010: Die Zukunft

Die Zukunft von Java 7 ist der Dolphin, denn dies ist der ursprüngliche Codename, unter dem die kommende Java-Version auch bekannt ist. Der Weg führt weiter weg von einer Programmiersprache, hin zu Java als einer Technologie-Plattform. Ferner wird die Zukunft von Java ist nicht mehr unter Sun stattfinden. Leider hat auch der „Vater" von Java, James Gosling, Oracle/Sun verlassen. Java wird sicher langfristig gepflegt und weiterentwickelt. Zu groß ist das Interesse aller mit Java eng verdrahteten Firmen, die sich sowohl im JCP als auch in der Open Source Community bewegen. Jetzt muss sich nur zeigen, zu welchem Charakter sich Java manifestieren wird. Das hängt aber auch stark vom Feedback der Java-Community – also uns allen – ab.

Kontakt:
Oliver Szymanski
oliver.szymanski@source-knights.com

Oliver Szymanski (Dipl. Inform., Univ.) ist als Software-Architekt / Entwickler, Berater und Trainer in den Bereichen Java, .NET und Mobile-Development tätig. Parallel dazu arbeitet er als Schriftsteller (siehe auch Seite 19 und 27)

iJUG Verbund

Mitglieder stellen sich vor

JUGS java user group stuttgart

Java User Group Stuttgart e.V. (JUGS)

Kurz nach Erscheinen des JDK 1.1 Release wurde vor nunmehr dreizehn Jahren im März 1997 die Java User Group Stuttgart (JUGS) als unabhängiger, gemeinnütziger Verein gegründet und hat heute über 400 Mitglieder. Ziel ist es, Java-Anwendern und Interessierten ein Forum zum Austausch von Informationen und Erfahrungen zu bieten und den Aufbau direkter Kontakte zu innovativen Firmen der Region zu ermöglichen. Dazu werden regelmäßig Vortrags- und Diskussionsabende, größere Events zu aktuellen Themen (z.B. mit Erich Gamma, Kent Beck und Peter Coad) sowie seit 1998 das Java Forum Stuttgart veranstalten.

Der Schwerpunkt der Aktivitäten liegt bis heute im persönlichen Kontakt der Mitglieder untereinander und mit anderen Java-Interessierten aus der Region. Dies stellt ein wichtiges Unterscheidungsmerkmal zu anderen, hauptsächlich auf das Internet und damit meist überregional orientierten Communities dar. Zu diesem Zweck organisieren der Verein im Jahr zwischen 12 und 16 kleinere und größere Veranstaltungen, die nach wie vor auf großes Interesse stoßen. Dies zeigt auch, dass das Interesse an Java allgemein bisher ungebrochen ist. Auf der Web-Seite kann man in der Rubrik „Veranstaltungen" noch Kurzprotokolle und meistens auch die Vortragsfolien aller jemals durchgeführten Veranstaltungen einsehen. Aus heutiger Sicht ist es manchmal ganz amüsant zu sehen, mit welchen Dingen man sich vor so vielen Jahren einmal beschäftigt hat.

Die größte Veranstaltung im Jahr ist das Java Forum Stuttgart (JFS, www.java-forum-stuttgart.de), das 2010 zum 13. Mal stattfand. Aus den bescheidenen Anfängen im Jahr 1998 hat sich daraus im Laufe der Zeit eine respektable Konferenz mit zuletzt rund 1.100 Besuchern entwickelt. Das JFS ist eine eintägige Konferenz, die dem Besucher derzeit mit 42 Vorträgen in 6 parallelen Tracks und einem Ausstellungsteil mit rund 35 Ausstellern ein breites Spektrum der aktuellen Java-Technologie vermittelt. Die Mischung aus Technologie-Vorträgen, Projekt-Erfahrungsberichten und Produkt-Präsentationen fand von Beginn an großen Zuspruch. Hinzu kommt ein einzigartiger Rahmen mit exklusiver Bewirtung und einer attraktiven Abendveranstaltung mit Musikuntermalung die zum Verweilen und angeregten Diskussionen einladen soll. Am Tag nach dem JFS findet dann noch das Experten-Forum-Stuttgart statt, das von unserem Partner, dem Software Experts Network Stuttgart (SENS – www.softwareexperts.de), veranstaltet wird. Hier bietet sich die Möglichkeit aktuelle Software-Themen in halbtägigen Workshops weiter zu vertiefen.

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

JSXP steht für „Just Simple eXtensible Pages" und ist ein Open Source Web-Anwendungs-Framework, für das seit Kurzem ein Release Candidate existiert. Wie der Name schon sagt, wurde bei der Entwicklung des Frameworks großen Wert auf Einfachheit, aber flexible Erweiterbarkeit gelegt. Außerdem bietet das Framework eine enge Zusammenarbeit zwischen Designer und Entwickler.

JSXP – das etwas andere Web Framework

David Tanzer, Freiberufler
und Oliver Szymanski, Source-Knights.com

JSXP wurde mit der Zielsetzung entwickelt, Rapid Application Development und eine reibungslose Zusammenarbeit von Designern und Entwicklern zu ermöglichen. Dazu baut das Framework auf folgenden Kern-Prinzipien auf:

- Compiler-Prüfung der Verbindung zwischen Design und Code: Aus Designvorlagen generierter Code stellt sicher, dass im Code nur Elemente referenziert werden können, die im Design auch tatsächlich existieren. Deshalb werden viele Fehler zur Übersetzungszeit erkannt und nicht erst durch Unit-Tests gefunden.
- Agile Entwicklung: JSXP erlaubt es, das Design während des Projekts anzupassen und bietet dabei Zugriff von Designern und Entwicklern auf dieselben Dateien. Insbesondere durch die bei JSXP weiterhin gewährleistete Originalstruktur und Ansicht der Design-Dateien wird dieser iterative und agile Prozess möglich. Zudem kann man nach und nach rein statische Designvorlagen durch die Implementierung mit dynamischen Ansichten ersetzen. JSXP erlaubt von Beginn an eine Navigation über die gesamte Web-Anwendung in Form der Design-Vorlagen, so dass stets eine testbare und im Laufe des Projekts eine um Funktionalität erweiterbare Anwendung vorliegt.
- Trennung von Code und Design: Es ist nicht möglich, Code in die Design-Dateien zu mischen, wie das zum Beispiel bei JSF (durch die Expression Language) der Fall ist. Implementierung erfolgt ausschließlich in reinem Java.
- Einfaches (X)HTML-Design: Der Web-Designer arbeitet mit normalen, gültigen HTML-Dateien und kann somit jedes beliebige Werkzeug zum Bearbeiten von HTML verwenden. Die einzige Einschränkung, die der Standard-Parser von JSXP bietet, ist, dass diese Dateien gültiges XML enthalten müssen (zum Beispiel müssen alle Tags geschlossen werden).
- IDE-Unterstützung: Da JSXP auf reinem HTML und Java basiert, sind keine besonderen Tools notwendig, sondern JSXP ist in jede IDE einfach integrierbar. Durch den generierten Code gibt es trotzdem eine Syntax-Prüfung sowie eine Konsistenzprüfung der Design-Vorlagen mit der Implementierung. Somit ist stets sichergestellt, dass sich mit den bereits eingesetzten Tools ohne Weiteres JSXP-Projekte durchführen lassen. Man ist nicht von speziellen Plugins abhängig, der Designer kann die Vorlagen und der Entwickler die Implementierung auf die Weise erstellen, die er gewohnt ist.
- Lesbare URLs: In vielen Fällen wird durch das Framework selbst sichergestellt, dass die URLs von Seiten „bookmarkable" und lesbar sind. Für die Fälle, in denen das nicht automatisch passieren kann, wird dem Programmierer eine Möglichkeit gegeben, selbst solche URLs auf einfache Weise zu realisieren.
- Das Framework unterstützt außerdem Komponentenorientierung, Element Templating, View Templating, Internationalisierung, einfache Verwaltung von Server-seitigem Status, View Flows und AJAX. Über das Resource Handling System kann nicht nur eine der Arten von Internationalisierung realisiert werden, es lassen sich auch Ressourcen von eingebetteten Komponenten (wie CSS oder Java-Script-Dateien) verwalten.

Entwicklung einer ersten JSXP-Seite

In JSXP geht man davon aus, dass ein Design vorliegt, um eine Seite zu entwickeln. JSXP ist somit Design-Driven. Dabei lässt sich das Design beliebig oft innerhalb des Projekts ändern. Man kann das Projekt also mit einem Prototyp bis hin zur späteren richtigen Anwendung entwickeln. Das Design kann eine beliebige XML-Datei sein. Wir starten hier mit einer HTML-Seite, die XML-konform ist:

```
index.html:
<html>
       <body/>
</html>
```

Aus dieser Design View erzeugt der JSXP-Generator die Basisklasse für den View Controller: „IndexHtmlControllerGenerated.java". Der Generator wird über ein Shell-Skript in den Build-Vorgang eingebaut. Beispielsweise kann man über die „Builder"-Einstellungen eines Eclipse-Projekts konfigurieren, dass diese Basisklassen immer dann generiert werden, wenn sich eine X(HTML)-Datei ändert. Somit hat man immer direkt die Syntax- und Konsistenzprüfungen.

Nachdem die Basisklasse generiert ist, kann die Webseite schon in einem Web-Container „deployed" werden. Wenn man nun „index.html" aufruft, sieht man, dass nicht nur eine leere Seite angezeigt wird, sondern eine ganze Menge Debug-Meldungen des JSXP-Frameworks angehängt sind. Das JSXP-Framework löst zum Namen „index.html" die dazugehörige View-

Controller-Klasse auf und führt dort den Lebenszyklus der Seite aus, was unter anderem zu den Debug-Meldungen führt.

Streng genommen ist man also schon fertig mit der Entwicklung der ersten JSXP-View. Die Seite soll aber noch etwas Funktionalität erhalten. Dazu muss man einerseits auf Komponenten des Design-Views aus dem Code zugreifen können und andererseits auf Ereignisse im Lebenszyklus reagieren.

Für unser Beispiel nehmen wir an, dass wir in einem iterativen Prozess eng mit einem Designer zusammenarbeiten. Dieser hat uns inzwischen eine neue Version der Seite geliefert, die jetzt einen Logout-Link und einen Administrations-Link enthält. Um diese beiden Links aus dem Java-Code ansprechen zu können, vergibt man erst einmal IDs (siehe Listing 1).

```
index.html:
<html>
      <body>
            <a href="admin.html"
jsxp:id="adminLink">Admin</a>
            <a href="index.html
jsxp:id="logoutLink">Logout</a>
      </body>
</html>
```

Listing 1

```
IndexHtmlController.java:

[...]
public class IndexHtmlController extends
            IndexHtmlControllerGenerated<Object> {
      @Override
      protected void init(Importer importer) throws Exception {
            if(!((UserContext)Context.getContext().
getUserContext()).isLoggedIn()) {
                  getElementLogoutLink().remove();
            }
      }
}
```

Listing 2

```
public class UserContext extends HttpUserContext {
@Override
public boolean isAllowedToViewPage (
ViewController<?> view, String uri) {
      if((view instanceof AdminHtmlController.class ||
uri.indexOf(„admin") > 0) && !isLoggedIn()) {
            return false;
      }
      return true;
}
}
```

Listing 3

```
public class Application extends WebApplication{
@Override
public ViewAlias getAliasFor(
String sourcePath, String targetPath,
Map<String, Object[]> parameter) {
      if(!Context.getContext().getUserContext()
.isAllowedToViewPage(null, targetPath)) {
            return new ViewAlias(
new LoginHtmlController(), parameter);
      }
      return null;
}
}
```

Listing 4

Die IDs stören das Design nicht weiter, so dass die Vorlage immer noch mit jedem HTML-Tool angezeigt werden kann. Wir wollen jetzt den Logout-Link verschwinden lassen, wenn der Benutzer nicht eingeloggt ist. Dazu leitet man von der generierten Controller-Klasse ab (siehe Listing 2).

„Init" ist eine der Methoden, die im Lebenszyklus der Komponente aufgerufen werden. Hier kann man eine allgemeine Initialisierung der Komponente durchführen, während man Geschäftslogik in „execute" packt. Diese wird nur aufgerufen, wenn „validate" (ein vorheriger Schritt im Lebenszyklus zur Validierung von Eingaben) keine Fehler geliefert hat.

Um die Methode „isLoggedIn" am UserContext aufrufen zu können, muss noch ein eigener UserContext implementiert und dieser an der Web-Applikation registriert werden.

Das eigentliche Entfernen des Logout-Links ist nur noch ein Methoden-Aufruf. Die Get-Methode, die das benötigte Element liefert, wurde vom Generator anhand der jsxp:id generiert und steht in der Basisklasse zur Verfügung. Durch die Verwendung einer speziellen Methode „getElementLogoutLink" passend zur ID in der Vorlage ist gewährleistet, dass man Fehler, wie versehentliche Umbenennung oder Löschung des Links aus der Vorlage, bereits in der Kompilierungsphase und beim Build bemerkt. Bei den meisten anderen Frameworks treffen einen solche schnell ausgelösten Fehler zur Laufzeit – und was ist schlimmer, als in einem laufenden System einen Fehler zu haben und diesen langwierig analysieren zu müssen?

Auf die Seite „admin.html" soll man aber natürlich nur Zugriff haben, wenn man eingeloggt ist. Also müssen als nächstes der Admin-Bereich abgesichert und der Benutzer gegebenenfalls auf eine Login-Seite weitergeleitet werden.

Absichern des Admin-Bereichs

Um festzulegen, welche Seiten ein Benutzer sehen darf, gibt es zwei Methoden im UserContext, die man als Entwickler überschreiben kann: „public boolean isAllowedToAccessUri (String uri)" und „public boolean isAllowedToViewPage (ViewController<?> view, String uri)". Die erste der beiden Methoden wird aufgerufen, bevor das

```
LoginCommandController.java

public class LoginCommandController extends
AbstractCommandController
<LoginCommandController.InputParameters> {

public class InputParameters {
       private String loginName;
       private String password;
       // […] getter und setter […]
}

private InputParameters bean = new InputParameters();

@Override
public InputParameters getInputParameterBean() {
       return bean ;
}

@Override
public void execute() throws Exception {
String userName = bean.getLoginName();
       if(/* userName gültig */) {
              /* Benutzer einloggen durchführen */
       } else {
              Context
                     .getContext().getFlashMessages()
                     .addErrorMessage(
                            „Username or password wrong“);
       }
       redirect(new AdminHtmlController ());
}
}
```

Listing 5

```
<ul jsxp:id=“list“>
<li jsxp:id=“item“>
       <a href=“$(LinkTarget)“>
              <span jsxp:id=“linkText“>Beispiel
1</span>
       </a>
</li>
<li>Beispiel 2</li>
<li>Beispiel 3</li>
</ul>
```

Listing 6

```
/* Vorab die Beispieltabelle aus der Vorlage leeren */
getElementList().removeElements();
/* zur JSXP-ID „item“ ein Template erzeugen */
ElementTemplate<Item> itemTemplate =
getElementItem().createTemplate();

/* Über Daten iterieren und die Tabelle füllen*/
for(SomeBean bean : allBeans) {
Item i = itemTemplate.createElement();
i.setVariableLinkTarget(bean.getLinkTarget());
i.getElementLinkText().setValue(bean.getLinkText());
getElementList().addElement(i);
}
```

Listing 7

Framework versucht, den View-Controller zu der URI zu finden, und kann verwendet werden, um URIs generell zu sperren. Die zweite Methode wird aufgerufen, nachdem der View-Controller gefunden wurde und wird benutzt, um den Zugriff auf bestimmte View-Controller einzuschränken. Deshalb verwenden wir jetzt diese Methode (siehe Listing 3).

David Tanzer ist seit 2006 als selbstständiger Software-Entwickler und Berater tätig. Im Rahmen dieser Tätigkeit beschäftigt er sich mit Java-Enterprise-Anwendungen, Web-Anwendungen, mobilen Anwendungen und agiler Software-Entwicklung. Er ist Mit-Initiator des Java-Web-Frameworks JSXP.

Versucht man nun, die URL „admin.html“ aufzurufen, so bekommt man einen HTTP-Fehler 403 (Zugriff verboten). Als Nächstes soll der Benutzer in diesem Fall auf eine Login-Seite weitergeleitet werden. Dazu legen wir eine leere HTML-Seite an (nur HTML und Body tags), damit eine View-Controller-Basisklasse generiert wird. Der Designer wird die Login-Seite befüllen, während wir die Weiterleitung implementieren. Um den Benutzer weiterzuleiten, verwenden wir einen sogenannten „View Alias“ (siehe Listing 4).

Hierbei handelt es sich um eine einfache Möglichkeit, für eine bestimmte URI eine andere View auszuliefern, als standardmäßig vorgesehen wäre. Damit kann man nicht nur Weiterleitungen realisieren, sondern auch URLs lesbarer machen: Über die URL „some-page-that-includes-a-query.html“ kann man etwa die Seite „page.html?id=17“ abrufbar machen, wie man dies von gängigen Blogs und Content Management Systemen kennt. Nun wird der Benutzer zum Login-Formular weitergeleitet, das als Nächstes ausgewertet werden muss. Hierzu müssen keine jsxp:ids vergeben werden, aber als „action“ eines Formulars setzen wir hier „login.command“, um die Eingaben später komfortabel auswerten zu können.

Formulareingaben auswerten

Um Formulareingaben auswerten zu können, muss zuerst ein Controller für die Action des Formulars erzeugt werden. Die JSXP-Namenskonventionen geben vor, dass die Controller-Klasse für „login.command“ LoginCommandController heißt. Dafür gibt es keine generierte Basisklasse, darum leitet man solche Controller von der Framework-Klasse „AbstractCommandController“ ab. Diese Basisklasse ist für Controller gedacht, die selber keine View (keine Designvorlage) zugeordnet haben, sondern in der Regel abhängig von der in ihnen implementierten Logik auf andere Seiten weiterleiten (redirect).

Als Generics-Template-Parameter kann man bei Controllern eine Java-Bean-Klasse angeben, in der das Framework die Werte von GET- und POST-Parametern übergibt. Diese Bean-Klasse definieren wir hier der Einfachheit halber direkt in der Controller-Klasse (siehe Listing 5).

Die Namen der Java Properties im Parameter „Bean“ müssen mit den Namen der Eingabefelder übereinstimmen. Das Text-Eingabefeld für den Benutzernamen im HTML-Code muss also das Attribut name ="loginName“ gesetzt haben. Man kann Views dafür noch Präfixe zuweisen, damit man Eingabeparameter unterschiedlichen Komponenten zuweisen kann, was insbesondere bei Templating und der Einbettung fertiger Komponenten auftreten kann. Die Methode „getInputParameterBean“ liefert dem Framework ein Objekt, das mit den aktuellen Eingabeparametern befüllt wird. In den Methoden „validate“ und „execute“ können diese dann ausgewertet werden.

Element-Templating

Im Bereich „Entwicklung einer ersten JSXP-Seite“ wurde bereits gezeigt, wie man aus dem Programmcode auf Elemente der Webseite über generierte Get-Methoden zugreifen kann. Um Funktionen wie Listen oder Tabellen in einer Webseite realisieren zu können, muss man diese Elemente auch vervielfältigen können. Das geschieht mittels Element-Templating. Betrachten wir als Beispiel den folgenden HTML-Code (siehe Listing 6).

Dieses Beispiel definiert eine Liste mit mehreren Beispiel-Einträgen. Einen dieser Einträge haben wir bereits mit JSXP-IDs versehen. Außerdem wurde für den Link die Variable „LinkTarget“ definiert, die auch im Code gesetzt werden kann. Solche Variablen können auch automatisch mit Java Bean Properties befüllt werden, dabei würde man allerdings auf die Compiler-Prüfung verzichten, darum wird es hier nicht gezeigt. Wir wollen nun für eine Liste von Einträgen diese HTML-Liste befüllen (siehe Listing 7).

Zuerst werden aus der Liste alle Beispiel-Elemente entfernt, sie soll ja später nur noch dynamische Elemente enthalten. Die Beispiele müssen nicht aus der Vorlage gelöscht werden. Dies ermöglicht dem Designer, weiterhin seine Beispiel-Tabelle zu sehen. Danach wird aus dem Element „item“ ein Template erzeugt. Hier sieht man auch, dass für alle Elemente ein Interface generiert wird (in diesem Fall Item), das dann zur Laufzeit vom Framework implementiert wird.

In der Schleife entsteht für jeden Listeneintrag aus dem Template wieder ein Element, das dann mit Werten befüllt wird.

Fazit

Dies waren einige der Features, die JSXP bietet. Weitere Beispiele und Erläuterungen der anderen Funktionalitäten findet man unter www.jsxp.org sowie ein Tutorial-Video, in dem eine vollständige Web-Anwendung in einer Stunde live erzeugt wird, auf www.source-knights.com. Gern stehen die Autoren natürlich auch für Antworten auf Fragen und Hilfen/Unterstützung zur Verfügung.

Kontakt:
David Tanzer
business@davidtanzer.net
Oliver Szymanski
oliver.szymanski@source-knights.com

Oliver Szymanski (Dipl. Inform., Univ.) ist als Software-Architekt / Entwickler, Berater und Trainer in den Bereichen Java, .NET und Mobile-Development tätig. Parallel dazu arbeitet er als Schriftsteller (siehe auch Seite 16 und 27)

Die Komplexität der Java Platform Enterprise Edition (JEE) und der damit verbundene Implementierungsaufwand sind auch nach mehr als zehn Jahren in vielen Fällen zu hoch. Um diese Komplexität beherrschbar zu machen, bedienen sich Entwickler und Architekten oft diverser Frameworks und der damit verbundenen Best-Practice-Ansätze. Oracle bietet mit ADF „End-to-End"-JEE Entwicklung mit hoher Produktivität, während Spring die Flexibilität und Framework-Unabhängigkeit in den Vordergrund stellt. Der Artikel stellt das bekannteste Open-Source-Framework vor und zeigt auf, wie sich beide Konzepte sinnvoll ergänzen können.

Mehr Flexibilität auch für Oracle ADF? Enterprise Java im Griff des Spring Frameworks

Mario Herb, esentri consulting GmbH

Zu Beginn ein Rückblick auf die Entstehungszeit von Java-Enterprise-Anwendungen: Im Dezember 1999 wurde mit J2EE 1.0 die erste Version der Java-Middleware-Plattform veröffentlicht. Seither hat man die technischen Erweiterungen der Plattform nicht immer nur positiv gesehen (zum Beispiel EJB 2.X) und die Komplexität der JEE-Anwendungen wurde eher kritisch betrachtet. Inzwischen hat sich JEE aber etabliert und seine Leistungsfähigkeit in zahllosen Projekten bewiesen. An diesem Erfolg haben vor allem die vielen Java-Frameworks einen großen Anteil, da sie die Komplexität reduzierten und die Produktivität erhöhten. Auch Oracle bietet mit Application Development Framework (ADF) ein solches Framework an, und Oracle war auch eines der ersten Unternehmen, die das Potenzial effektiver „End-to-End"-Entwicklung – von der Datenbank bis zum Frontend – von JEE-Anwendungen erkannt haben. So ist es heute mit dem JDeveloper möglich, innerhalb kürzester Zeit komplette JEE-Anwendungen auf Basis von ADF zu realisieren.

Paint your Code mit ADF

Über die Vorteile von ADF und die unglaubliche Produktivität, die sich damit erzielen lässt, wurden schon viele Artikel geschrieben. In der Tat hat Oracle mit ADF und der perfekten Integration in die IDE JDeveloper Maßstäbe gesetzt. Dies gilt insbesondere für die Möglichkeit, die komplette Persistenzschicht für bestehende Datenbank-Strukturen zu generieren und mittels Data Bindings entsprechende Oberflächen per „Drag & Drop" zu designen. Die generelle Herangehensweise lehnt sich an die Entwicklung von Forms-Anwendungen an und bietet zunächst auch unerfahrenen Java Entwicklern die Möglichkeit, aufwendige JEE-Anwendungen zu implementieren.

Insbesondere die Implementierung der Geschäftslogik (Business Services) ist bei JEE-Anwendungen extrem wichtig und oft ein Auswahlkriterium für ein Framework. ADF bietet von Haus aus die Möglichkeit, viele Standard-Technologien einzubinden (siehe Abbildung 1, Business Service Layer). Neben den ADF Business Components können Enterprise Java Beans, Web-Services oder einfache Java-Klassen verwendet werden, was eine saubere Trennung zwischen der Geschäftslogik und dem Frontend ermöglicht. Oracle setzt dabei auf den Standard JSR-227 und das Konzept der Data Bindings, um von der konkreten Implementierung der Services zu abstrahieren. Trotzdem wird ADF außerhalb der Oracle-Community kaum verwendet. Die Gründe dafür sind vielfältig:

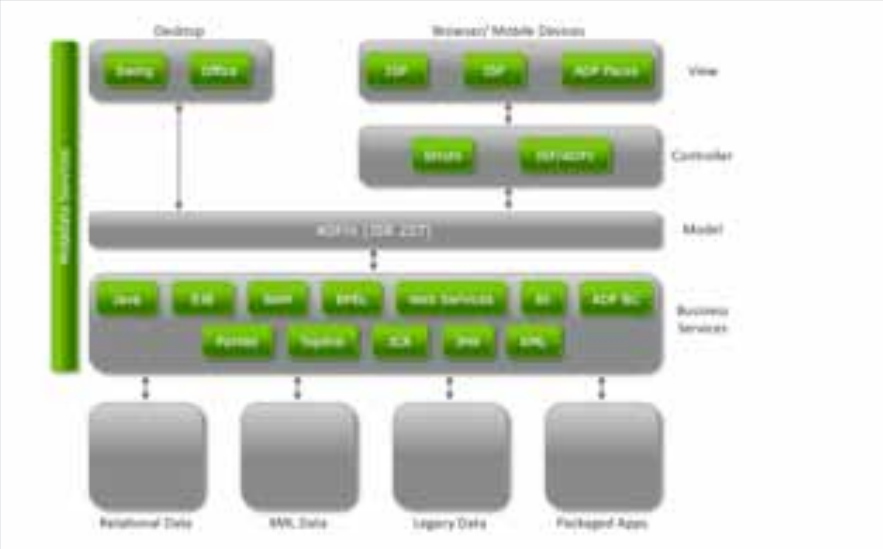


Abbildung 1: Die Architektur von Oracle ADF

- ADF ist nicht kostenlos verfügbar
- Die Abhängigkeit von den Oracle-Bibliotheken wird teilweise als zu groß eingeschätzt

- Erfahrenen Java-Entwicklern wird tendenziell zu viel generiert
- Standard-Anforderungen sind zwar einfach umzusetzen, spezifische Anforderungen benötigen jedoch tiefgreifende Kenntnisse über den internen Aufbau von ADF
- Unzureichende Integration in Konfigurationsmanagement-Tools wie Maven
- ADF-Anwendungen sind zum Teil schwierig zu testen

Folglich sind in den letzten Jahren vor allem im Open-Source-Umfeld weitere Frameworks entstanden, um die Komplexität von großen Java-Anwendungen beherrschbar zu machen und Best-Practice-Programmierkonzepte zu unterstützen.

Leichtgewichtiges JEE mit Spring

Den wohl radikalsten Ansatz verfolgte dabei die Firma Springsource mit dem Spring-Framework. Sie entwickelte ein verständliches, konsistentes Programmier- und Konfigurationsmodell, das als Hauptmerkmal besonders leichtgewichtig ist und ohne große Abhängigkeiten auskommt. Dabei stand vor allem die Entkopplung einzelner Komponenten im Vordergrund, um guten Programmierstil zu fördern.

Inzwischen wird der Marktführer im Open-Source-Umfeld weltweit von Millionen Entwicklern eingesetzt, und als weiteres Indiz für den großen Erfolg wurde Springsource 2009 für 420 Millionen Dollar von der Firma VMWare gekauft. Das Geheimnis dieses Erfolgs ist einfach zu erklären.

Abbildung 2: Die Hauptkomponenten des Spring Frameworks

Weniger ist oft mehr

Ähnlich wie ADF bietet Spring (im weiteren Sinne) inzwischen einfache Möglichkeiten, Web-Anwendungen „End-to-End“ mit einem Framework zu entwickeln (siehe Abbildung 2, Spring Web MVC). Das Grundkonzept von Spring unterscheidet sich dabei jedoch von ADF. Spring ist in allen Punkten non-invasiv, das heißt, es wurde bei der Konzeption des Frameworks darauf geachtet, dass keine festen Abhängigkeiten entstehen. Um die Einfachheit zu gewährleisten, kann Geschäftslogik in Spring mit einfachen „Plain Old Java Objects“ (POJO) implementiert werden, die lose miteinander gekoppelt sind. Möglich macht dies das sogenannte „IoC-Prinzip“ (Inversion of Control), das gemäß der Hollywood-Devise „Don't call me, we call you“ funktioniert.

Im Hintergrund sorgt dann der Spring Context (analog zur ADF Data Control Factory) zur Laufzeit für die Instanziierung der Objekte (siehe Abbildung 4). Dieses Prinzip nennt sich „Dependency Injection“ und ist innerhalb des gesamten Spring Frameworks anwendbar. Ähnlich wie bei ADF erfolgt die Konfiguration der Abhängigkeiten dabei deklarativ und kann wahlweise über eine XML-Konfiguration oder über Annotations vorgenommen werden, jedoch sind die Möglichkeiten nicht nur auf die Verbindung zwischen Backend und Frontend beschränkt (siehe Data-Bindings-Konzept bei ADF).

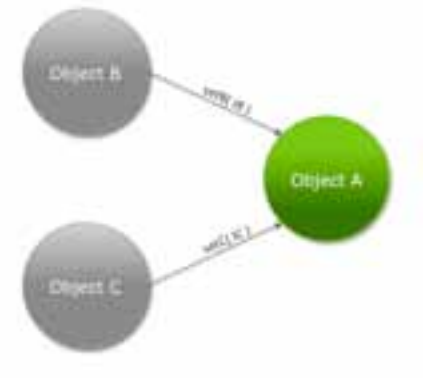


Abbildung 3: Klassische Instanziierung von Objekten

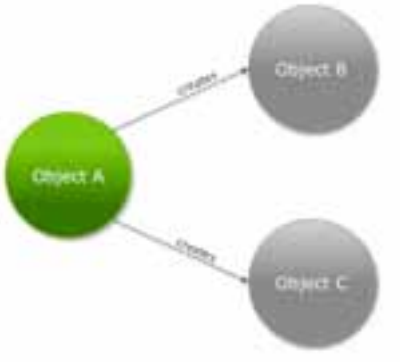


Abbildung 4: „Inversion of Control“ mit „Dependency Injection“, Objekt-Instanzen werden in das Objekt A von außen „injiziert“

```
<bean id="objectA"
class="example.ObjectA">
  <property name="objectB"
ref="objectB"/>
  <property name="objectC"
ref="objectC"/>
</bean>

<bean id="objectB"
class="example.ObjectB"/>
<bean id="objectC"
class="example.ObjectC"/>
```

Listing 1: Konfiguration des Beispiels aus Abbildung 4 mit Spring und XML

```
...
class ObjectA implements IOb-
jectA{
       private IObjectB ob-
jectB;
       private IObjectC ob-
jectC;
...
}
```

```
...
class ObjectB implements IOb-
jectB{
...
}
```

Listing 2: Definition der verwendeten Klassen aus Listing 1

Was im ersten Moment nach technischer Spielerei aussieht, hat jedoch weitreichende Folgen für die Programmierung und bringt weitere Vorteile mit sich, die exemplarisch im Folgenden aufgeführt werden:

- Lose Kopplung und gutes Software-Design wird gefördert
- Bessere Testbarkeit durch Mock-Objekte und JUnit
- Hohe Flexibilität (zum Beispiel einfacher Austausch Hibernate/iBatis als Persistenz-Framework)
- Aspekt-orientierte Konzepte als Ergänzung einfach möglich

Lose Kopplung

Wie bereits erwähnt, fördert Spring guten Java-Programmierstil durch „Dependency Injection" und „Inversion of Control". Eine zentrale Rolle spielen dabei die Java-Interfaces, die eine lose Kopplung erst ermöglichen. Wo bei ADF zumeist erst der Code beziehungsweise dessen Generierung im Vordergrund stehen, sind bei Spring vor allem die Schnittstellen relevant. Im Beispiel aus Listing 2 sind die Beans „objectB" und „objectC" nur über Anpassung der Spring-Konfigurationsdatei durch andere Implementierungen austauschbar, solange diese auch die entsprechenden Interfaces implementieren.

Mocks und JUnit testen

Testen wird bei Spring als integraler Bestandteil der Entwicklung betrachtet. Durch die lose Kopplung der einzelnen Spring-Komponenten wird die Testbarkeit grundsätzlich vereinfacht. Da Spring Beans im Grunde immer einfache POJOs sind, können sie beispielsweise in Unit Tests auch ohne Laden des Kontexts durch einfache Instanziierung per „new" erstellt werden. Aber auch über diese grundlegenden Möglichkeiten hinweg bietet Spring weitreichende Möglichkeiten, das Testen zu unterstützen. Unit Testing wird durch die Integration von Test-Frameworks wie JUnit oder auch TestNG erleichtert. Spezifische, von Spring angebotene Testklassen sowie die deklarative Steuerung der Tests über vielfältige Annotationen vervollständigen hier das Bild.

Darüber hinaus bietet Spring noch eine ganze Menge an Equipment für Integrationstests an. Das Laden des Spring-Kontexts über Annotationen in Testklassen, inklusive der Dependency Injection von zu testenden Spring Beans, und das Angebot diverser Mock-Frameworks (beispielsweise zur Simulation eines JNDI-Kontexts oder auch zur Simulation von Servlet-und Portlet-Aufrufen) gehören hier zum gängigen Repertoire. Vor allem ist es auch auf einfache Weise möglich, das Transaktionsmanagement in das Testing mit einzubeziehen und so das transaktionelle Verhalten der Anwendung automatisiert zu überprüfen.

Flexibilität

Von Anfang an verfolgten die Erfinder von Spring das Ziel, dem Entwickler möglichst viel Flexibilität einzuräumen. Getreu dem Motto „alles kann, nichts muss" kann man alle Module des Spring Frameworks verwenden, oder nur Teile davon. Der Grundgedanke dabei geht allerdings weiter als bei ADF, das eher auf die „End-to-End"-Programmierung ausgelegt ist. Spring ist vielmehr eine echte Integrationsplattform, die hohen Wert auf die Austauschbarkeit einzelner Frameworks legt. So kann man in Spring innerhalb kürzester Zeit die Persistenzschicht austauschen und beispielsweise iBatis anstatt Hibernate einsetzen. Gleiches gilt für die Anbindung des Frontends und der dort verwendeten Technologien. Egal ob man sich entscheidet, mit JSP, JSF oder Adobe Flex Benutzeroberflächen zu realisieren – Spring hat die passenden Konzepte dafür bereits vorgesehen. Diese Flexibilität wird von vielen Entwicklern weltweit geschätzt und garantiert Unternehmen eine hohe Investitionssicherheit. So ist es nicht verwunderlich, dass immer mehr Unternehmen entscheiden, große Teile ihrer Java-Projekte mit Spring zu realisieren.

Aspektorientierte Programmierung (AOP)

Ein wesentliches Ziel von Spring ist, wie bereits beschrieben, die Non-Invasivität des Frameworks. Das bedeutet, dass die Anwendungslogik aus reinen POJOs ohne Abhängigkeiten vom Framework selbst aufgebaut werden kann. Über aspektorientierte Programmierung können bei Spring Querschnittsbelange (wie Logging), die an vielen Stellen einer Anwendung auftauchen, abgedeckt werden. Dies geschieht jedoch ohne Anpassen des Codes und Einfügen von sich ständig wiederholenden Code-Fragmenten an entsprechenden Stellen, sondern über die deklarative Beschreibung der „Einstiegspunkte" (Pointcuts), verbunden mit der zentralen Erstellung des einzufügenden Codes. Hierzu bietet Spring ein eigenes AOP-Framework an. Dieses ermöglicht es, bestimmte Codestücke in die Applikationslogik einzuweben (Weaving), und zwar vor, nach und um in Pointcuts definierte Methoden herum. Diese Technik wird von Spring beispielsweise für das Spring-eigene, deklarative Transaktionsmanagement genutzt. Das Einweben von Code-Fragmenten geschieht bei Spring zur Laufzeit und benötigt daher keinen gesonderten Kompilierungsvorgang. Für tiefergehende AOP-Belange (wie Compile-time Weaving oder Field Interception) wird von Spring vor allem das AOP-Framework AspectJ unterstützt, bei dem aus aspektorientierter Sicht fast keine Grenzen gesetzt sind.

Spring und ADF in Kombination

Die zentrale Frage ist, wie man die Vorteile von Spring mit der Produktivität von ADF kombinieren kann, um so das Beste aus beiden Frameworks für eigene Projekte zu nutzen. Trotz einiger Überschneidungen in den Konzepten ist eine Kombination in der Tat sinnvoll, beispielsweise um die Testbarkeit der Geschäftslogik zu vereinfachen.

Für ADF ist Spring zunächst einmal nichts anderes als ein zusätzlicher Service Provider, den man im ADF-Projekt einbinden kann. Zweckmäßig ist dies natürlich vor allem im Bereich der Geschäftslogik, also im Business Service Layer (siehe Abbildung 1). Auf der Frontend-Seite lassen sich dann die Vorteile von ADF Faces in gewohnter Weise nutzen und man kann hochproduktiv in JDeveloper Oberflächen erstellen.

Zusätzlich gibt es auch eine Spring Extension (siehe Abbildung 5) für JDeveloper, die seit einigen Tagen auch wieder mit der neuesten JDeveloper-Version funktioniert und die Arbeit mit Spring erleichtert. So benötigt man keine zweite IDE wie Eclipse. Wie aus Oracle-Kreisen brandaktuell zu hören ist, soll die Spring-Extension zukünftig sogar direkt in JDeveloper integriert werden.

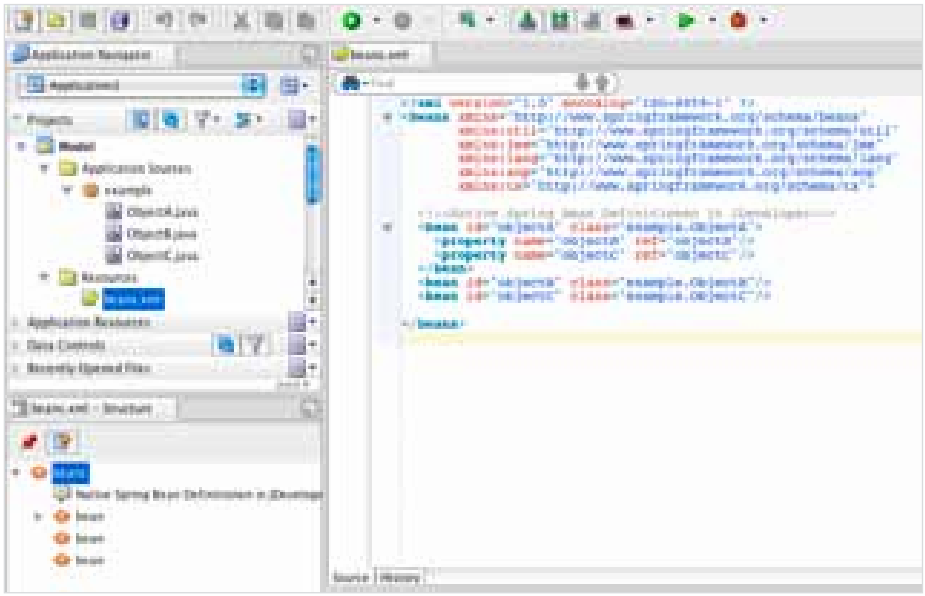


Abbildung 5: Die JDeveloper Spring Extension

Fazit

Spring ist ein äußerst flexibles und leichtgewichtiges Java Framework, das sich hervorragend für JEE-Projekte eignet. Während Oracle mit ADF vor allem auf Rapid Application Development setzt, steht bei Spring die Wart- und Erweiterbarkeit im Vordergrund. Hierfür ist initial ein höherer manueller Aufwand notwendig, der sich aber unserer Meinung nach durchaus rechnen kann.

Mit Inversion of Control, AOP und der Integration weiterer Frameworks über Spring lässt sich der Business Service Layer in Oracle ADF-Projekten aus Sicht des Autors sinnvoll ergänzen. Dabei unterstützt der JDeveloper die Arbeit mit dem Open Source Framework zukünftig auch ohne separate Extension.

Dass die Zusammenarbeit von Oracle und Springsource auch ansonsten sinnvoll ist, zeigen die beiden Unternehmen außerdem in dem kürzlich gemeinsam initiierten Projekt „Gemini", das Standards für die Integration existierender Java-Enterprise-Technologien in Modul-basierte Plattformen definieren soll.

Am Beispiel „Spring" zeigt sich, dass die Kombination aus Oracle und Open-Source-Technologien einen hohen Mehrwert bei der Realisierung komplexer JEE-Anwendungen bietet und ein Blick über den Tellerrand durchaus lohnenswert ist.

Mario Herb ist Geschäftsführer der esentri software GmbH in Karlsruhe. Nach langjähriger Beratungstätigkeit in den Bereichen Oracle, Middleware und Java hat er sich auf die Realisierung komplexer Web-Anwendungen spezialisiert.

Kontakt:
Mario Herb
mario.herb@esentri.com

Mitglieder stellen sich vor

iJUG Verbund

Java User Group JUG Erlangen-Nürnberg

JUG Erlangen/Nürnberg

Die Java User Group Erlangen / Nürnberg wurde 2007 mit der Idee gegründet, die Java-Gemeinschaft der Region zu fördern. Man trifft sich regelmäßig zu interessanten Vorträgen und Themen des Java-Umfelds. Mitgliedschaft bedeutet lediglich, den E-Mail-Newsletter zu abonnieren, um über die aktuellen Events informiert zu sein. Dank der Unterstützung zahlreicher in der Metropolregion Nürnberg ansässiger Unternehmen variieren die Orte der Treffen und fördern so den Kontakt und das Netzwerken in der Region.

Oliver Szymanski, iJUG-Vorstand meint: „Ich bin stolz darauf, gemeinsam mit den anderen JUGs und der DOAG die iJUG gegründet zu haben und damit die Java User Group Erlangen/Nürnberg in die neue starke Gemeinschaft führen zu dürfen. Jetzt treffen wir uns nicht nur, um über Themen informiert zu werden, sondern können auch die Interessen erfolgversprechend vertreten."
Weitere Informationen: http://www.source-knights.com

Kaum ein Java-Release wurde so oft verschoben wie die kommende Version. Ursprünglich unter dem Codenamen „Dolphin" bekannt, wird der Delfin frühestens im September 2010 in freie Gewässer gelassen.

Ausblick auf Java 7

Oliver Szymanski, Source-Knights.com, iJUG

Normalerweise hatte sich Sun als Ziel gesetzt, spätestens alle drei Jahre ein Release zu veröffentlichen. Nachdem Java 6 im Jahr 2006 herauskam, geht dieser Plan nicht mehr auf. Sogar eine weitere Verschiebung für Java 7 über den avisierten Termin im September 2010 hinaus ist möglich. Dies könnte allerdings die Integration weiterer neuer Features bedeuten.

Dafür ist Java auch nicht mehr nur eine Programmiersprache, sondern mittlerweile der Oberbegriff für eine Technologie-Sammlung aus virtueller Maschine, Java-Sprache (und weiteren auf der Maschine lauffähigen Sprachen), APIs und Programmier-Modellen. Mehr als nur eine Sprache laufen auf der Java-Technologie, und dass immer mehr Skriptsprachen auf der Java VM genutzt werden, ist ein Trend für die Zukunft. Je nach Aufgabe wählt man die passende Sprache. Gerade im Bereich Nebenläufigkeit spielen beispielsweise Clojure und Scala eine wichtige Rolle. Daher enthält ein neues Release nicht einfach nur eine angepasste Java-Sprache – es ist ein Bündel aus Anpassungen an die virtuelle Maschine, die Frameworks und letztlich auch an die Syntax der Sprache.

OpenJDK und Java 7

Die OpenJDK-Community ist eine Gemeinschaft von Entwicklern, die an quelloffenen Realisierungen von Java (dem OpenJDK) und den damit verbundenen Projekten arbeiten. Die Lizenz für OpenJDK-Software ist dabei die GPLv2 mit Classpath-Ausnahmen. Damit fließen Optimierungen zurück ins JDK, aber auch Software, die den Code nutzt, kann unter beliebiger Lizenz stehen.

Die kommende Java-Version wird im Rahmen des OpenJDK entwickelt und erste Versionen lassen sich von der OpenJDK.java.net-Website herunterladen. Das OpenJDK-Projekt entscheidet auch über die Neuerungen, die in das Release aufgenommen werden.

Neuerungen

Viele Kleinigkeiten und einige große Änderungen werden uns in Java 7 begegnen. Da der Prozess flexibel ist, können sich bis zum endgültigen Release weiter Änderungen ergeben. Nachfolgend ein Überblick über einen Großteil der momentan geplanten Neuerungen: Zuerst einmal werden die XML-APIs JAXP, JAXB und JAX-WS API aktualisiert. XRender Pipeline für Java 2D für die Nutzung moderner GPU-Funktionen, Unicode 5.1 sowie Elliptic Curve Cryptography sind neu. Statt des Swing-Application-Umfeld-Frameworks mit seiner Infrastruktur für Swing-Anwendungen nach JSR 296, der leider auf inaktiv gesetzt wurde, wird zu Swing der JXLayer-Komponenten-Dekorator eingeführt. Der vormals geplante JXDatePicker ist momentan für Java 7 verworfen. Alle bereits aus dem letzten Java 6 Update 10 bekannten Features wie Quickstarter und das Nimbus Look & Feel sind integriert. Für Solaris sind Neuerungen im Stream Control Transmission Protocol und im Sockets Direct Protocol enthalten.

Neuerungen an den APIs

Ein Problem in der bisherigen ClassLoader-Architektur wurde behoben. Es konnte zu einem Deadlock führen, wenn ClassLoader ohne fixe Ordnung untereinander delegieren. Ein weiteres Problem liegt im URLClassLoader, mit dem man Ressourcen über URLs nachladen kann. Ein Update der Ressourcen war kaum möglich, da man keinen Einfluss darauf hatte, wann die Ressource wieder vom ClassLoader freigegeben wurde. Es wird daher eine „close()"-Methode am URLClassLoader eingeführt, mit der die Ressource explizit wieder freigegeben werden kann.

Der JSR 203 und 51 wird umgesetzt und damit Java NIO angepasst. Eingeführt werden damit Multicast-Datagramme, Binding, asynchrone Zugriffe und Service Provider Interfaces für die Implementierung der Anbindung beliebiger Dateisysteme.

Mit JSR166y gibt es neue Klassen und Funktionalitäten mit dem leichtgewichtigen ForkJoin-Task-Framework für Java im Bereich „Concurrency". Verantwortlich dafür ist Doug Lea, ein bekannter Namen gerade bei Concurrency.

Spracherweiterungen

Es gibt Annotation (fast) überall. Bei jedem Auftreten eines Java-Typen kann man Java-7-Annotation dank des JSR 308 angeben. Dies kann negative Auswirkungen auf die Lesbarkeit des Java-Codes haben und es bleibt die Befürchtung, dass unerfahrene Entwickler die Konstrukte falsch oder zu exzessiv nutzen. Dennoch ist dies ein Schritt weiter dahin, Code mit Metadaten anzureichern (siehe Listing 1).

```
@VolatileDataContainer
@NeverNull
class ConstantList implements @Rea-
donly List <@Readonly T> {

       ConstantList(String name)
throws @VeryCritical ExplosiveExcep-
tion
{
              if (! name instanceof @
NonNull String)
                     throw @VeryCri-
tical new ExplosiveException();
}
...
}
```

Listing 1

In Java 7 wird über den JSR 294 der Weg zu einem Modulsystem eingeleitet. Der JSR sieht Super-Packages vor, die ihrerseits

aus Paketen und Klassen bestehen und definieren, welche ihrer Inhalte „public“ sind. Passend dazu gibt es den JSR 277, der aber leider nicht in Java 7 Einzug hält. Dieser JSR sieht Java-Module mit Lösungen für die bislang fehlende Versionierung der JARs, Exportlisten und Abhängigkeiten zwischen Modulen vor. Da das System aber zu komplex für die momentane Integration in die VM ist, ist stattdessen das leichtgewichtigere Modulsystem des Projekts Jigsaw Teil von Java 7. Bleibt die Frage, ob nicht das OSGi-Framework die beste Wahl gewesen wäre, gerade weil ohnehin geplant war, an der Kompatibilität des Frameworks und dem JSR 294 zu arbeiten.

Bereits früher für Java 7 geplant, dann wieder verworfen und momentan wieder als Bestandteil angedacht sind die Ziele des „Project Lambda“. Es sieht First-class-Functions, Function Types und Lambda-Ausdrücke (auch als „Closures“ bekannt) für Java vor. Die genaue Schreibweise steht noch nicht fest, die hier angegebenen Beispiele sind in der von Mark Reinhold aus dem Project Lambda benutzten Schreibweise verfasst. Die Features, die mit Project Lambda in Java zur Verfügung gestellt werden, erleichtern die Angabe anonymer Methoden und ermöglichen die Definition von Funktionstypen. Über diese können Funktionen per Parameter übergeben werden. Dies hilft, Callback-Interfaces zu vermeiden und stattdessen einen Funktionsparameter auf die Zielfunktion des Callbacks zu übergeben (siehe Listing 2).

```
/* Lambda Ausdruck für Funktion mit
Parameter x,y die x*y berechnet */
#(int x, int y) (x * y);

/* Lambda Ausdruck für Funktion die
das Minimum der Parameter x,y berech-
net */
#(int x, int y){
if (x < y)
return x;
else
return y;
 }

/* Funktionstyp auf die Multiplizier-
Funktion */
int(int,int) multiplier = #(int x, int
y)(x * y);
int result = multiplier(7, 2);
```

Listing 2

Dies lässt sich beliebig vertiefen. Eine Methode kann einen Funktionstyp zurückgeben, mit dem man dann weiterarbeitet etc. Es ist auch vorgesehen, Funktionstypen automatisch in anonyme Instanzen eines Interfaces oder einer abstrakten Klasse zu konvertieren, so dass die neuen Funktionstypen auch beim Aufruf alter APIs genutzt werden können (statt eines Listener-Interfaces einfach einen Funktionstyp mit der entsprechenden Signatur der Methode des eigentlichen Listeners). Closures werden, ähnlich wie früher die Generics, die Java-Syntax und den Umgang mit APIs erheblich verändern. Und wie auch bei Generics wird es sicher einen großen Unterschied in der notwendigen Erfahrenheit geben zwischen einem Entwickler, der sie nur benutzt und einem, der selbst APIs auf den neuen Konzepten aufbaut.

Im Zuge des Projekts „Coin“ fließen weitere Spracherweiterungen in Java, die insbesondere das tägliche Leben der Entwickler erleichtern sollen. Eine Gefahr bietet dabei die neue Möglichkeit, Switch-Konstrukte auch mit Strings auswerten zu können – verführt dies Entwickler doch dazu, inperformant über String-Vergleiche den Programmfluss zu steuern.

Eine verkürzte Schreibweise wird bei der Instanziierung von Objekten mit generischen Konstruktoren eingeführt, was ein wenig die Lesbarkeit hinsichtlich Generics bessert. Tat man sich früher noch schwer mit den unzähligen Compiler-generierten Warnings beim Aufruf von Methoden mit variabler Parameter-Anzahl, so wird diese Warnung stattdessen fortan bei der Methodendeklaration hervorgerufen. Dies hat zur Folge, dass wir nur bei der Methodendeklaration einmalig ein @SuppressWarnings angeben müssen, statt wie vorher bei jedem Aufruf. Des Weiteren sind Literale mit Binärmaske vorgesehen, und der Unterstrich soll zwischen Ziffern bei integralen Literalen zur besseren Lesbarkeit erlaubt werden.

```
int bin = 0b101;
long cardNumber =
1234_5678_9012_3456L;
```

Auch bei der Sprachunterstützung für Collections tut sich einiges; wir werden vereinfachte Schreibweisen für die Erzeugung von Sets, Maps, Lists und den Zugriff auf den Index bei Maps und Lists bekommen.

```
final List<Integer> digits =
[3, 1, 5, 9];
Map<Integer, String> m = new
HashMap<Integer, String>(2);
m[0] = „First“;
```

Ressourcen-Management wird erleichtert, indem Ressourcen (wie Stream, Writer, Reader) am „try“-Ausdruck angegeben werden können. Diese Ressourcen werden dann automatisch freigegeben. Wer sie früher nicht manuell geschlossen hat, mag nichts im Code vermissen, hat aber einiges falsch gemacht.

```
try (BufferedReader reader =
 new BufferedReader(new
FileReader(path))) {
       return reader.read-
Line();
}
```

Momentan sieht es auch so aus, als wenn in Java 7 eine Erweiterung beim Exception-Handling vorgesehen ist. Es soll möglich werden, dass man mit einem Catch-Clause mehrere Exception-Typen behandeln kann. Wenn dieses Feature in Java 7 tatsächlich Einzug erhält, ist es sicher angenehm, mehrere Catch-Blöcke, die eine Exception nur in die Logausgabe schreiben, zusammenzufassen. Allerdings kann es innerhalb eines komplizierteren Catch-Blocks schwierig werden, welchen Typ die behandelte Exception eigentlich hat. Hier ist der Entwickler gezwungen, stark auf die Details zu achten. Darüber hinaus kann man den Exception-Parameter des Catch-Clauses als „final“ kennzeichnen. Dies ermöglicht bei „rethrow“ einer Exception eine bessere Prüfung (zur Kompilierungszeit), welcher Exceptiontyp geworfen wird (der Typ ist dann Subtyp des Catch-Typen, ein im Try-Block auftretbarer Typ und kein Typ, der in vorherigen Catch-Clauses verarbeitet wurde).

```
try { ... }
catch ( final IOException |
DBException ex) {
  logger.log(ex);
}
```

Weitere Spracherweiterungen finden sich auch im Zuge der Anpassungen an der virtuellen Maschine.

Anpassungen bei der virtuellen Maschine

Für verbesserte Speicherverwaltung wird die Komprimierung für 64-Bit-Pointer in 32 Bit eingeführt. Der „Garbage First (G1)"-Algorithmus nimmt seinen Platz neben den anderen Garbage-Kollektoren ein, um den bekannten Concurrent Mark Sweep (CMS) zu ersetzen. Er war bereits in Java 6 Update 14 als experimenteller GC enthalten. Der G1 ist auf niedrige Wartezeiten optimiert und teilt den nicht permanenten Speicherbereich in gleich große Stücke. Er merkt sich die noch verwendeten Objekte in den Bereichen und die ein- und ausgehenden Referenzen. Bevorzugt behandelt der G1 die Bereiche mit den wenigsten Live-Objekten. Zeigen keine Referenzen in einen Bereich hinein, wird er direkt geleert, ansonsten werden betroffene Objekte in andere Bereiche verschoben. Zusätzlich erstellt der G1 Metriken, um abschätzen zu können, wie lange Garbage Collection für die Bereiche dauert. Dies erlaubt, mit einer maximalen Zeitdauer zu starten, so dass der G1 möglichst viele Bereiche in der vorgegebenen Zeit bearbeitet.

Die wichtigsten Änderungen an der VM betreffen jedoch die Unterstützung von Skriptsprachen (JSR 292, Supporting Dynamically Typed Languages). Der Java-Bytecode erhält einen „invoke dynamic", um dynamische Methodenaufrufe zu bestimmen, und die Struktur von Klassen kann zu Laufzeit modifiziert werden. Damit man auch in der Programmiersprache Java dynamische Aufrufe ohne komplizierte Verwendung der Reflection-API (mit der auch nicht alles möglich ist) ausführen kann, wird mit Dynamic ein neuer Typ eingeführt, der quasi alle denkbaren Kombinationen aus Methoden-Namen und -Signatur definiert. Von allen Typen kann auf Dynamic konvertiert werden sowie auch von Dynamic nach Object. Andere Konvertierungen müssen per „cast" erfolgen.

```
Dynamic x = new MyObject();
Object y = x.anyExpression(„Any
Parameters are allowed")
```

Eine weitere Änderung hinsichtlich des Austauschs mit Skriptsprachen ist die mögliche Verwendung beliebiger String-Literale als Identifier. Ein Hash-Zeichen, gefolgt von dem String-Literal, bildet dann den Namen.

```
Object #"source knights" = new
SourceKnights();
#"source knights".tournament();
```

Diese Erweiterung sollte man nicht unbedingt permanent verwenden. Sie dient sicher nicht der besseren Lesbarkeit des Programmcodes, sondern sollte lediglich in Ausnahmefällen genutzt werden.

Fazit

Bei Erweiterungen an einer Sprache – und im Falle von Java an einer Technologie-Sammlung – muss man beachten, dass man zwar mit den Werkzeugen kreativ sein will, dass Kreativität jedoch innerhalb der Definition der Werkzeuge begrenzt sein muss. Mit in sich abgeschlossenen Basis-Funktionen will die Welt der Entwickler reichhaltige Applikationen bauen. Auch die Abwärtskompabilität (teilweise vielleicht überschätzt) ist generell zu beachten. Aus diesen Gründen ist es wichtig, dass ein Prozess zur Gestaltung eines neuen Java-Releases nicht jede Idee direkt integriert, sondern saubere Gesamtlösungen vorsieht. Dies bremst sicherlich die Evolution einer Technologie, hält sie aber stabil.

Auf Java 7 haben wir lange gewartet und warten noch immer, daher können wir uns freuen, viele neue Features darin enthalten zu sehen. Java wird sich weiterhin als stabile Plattform entwickeln, die mit Java 7 für alle künftigen Einsatzzwecke bereit ist. Jetzt hängt es von uns als Community ab, Feedback zu geben und die Zukunft von Java dadurch mitzugestalten.

Kontakt:
Oliver Szymanski
oliver.szymanski@source-knights.com

Oliver Szymanski (Dipl. Inform., Univ.) ist als Software-Architekt / Entwickler, Berater und Trainer in den Bereichen Java, .NET und Mobile-Development tätig. Parallel dazu arbeitet er als Schriftsteller (siehe auch Seite 16 und 19)

iJUG Verbund – Mitglieder stellen sich vor

JUG Ostfalen

Die Region um Braunschweig und Wolfsburg hat in den letzten Jahren fantastische Zuwachsraten im Bereich der IT. Was lag also näher, als eine Java User Group zu gründen, die dem Rechnung trägt? Gesagt, getan! Im Januar 2010 war es soweit und Uwe Sauerbrei hat die JUG Ostfalen aus der Taufe gehoben. Sie ist damit noch relativ „frisch", aber kräftig am wachsen. Erste Ziele wurden bei einem lockeren Stammtisch formuliert, dem inzwischen zwei Vorträge folgten. Neben reinem Networking ist das Ziel, Industrie und Forschung (hier gibt es überall Hochschulen!) einander näher zu bringen, Know-how weiterzugeben und einen fachlichen Austausch zu ermöglichen. Das ist aber noch lange nicht alles! Eine ganze Menge Ideen warten noch auf Umsetzung, aber davon mehr zu gegebener Zeit.
Weitere Informationen: www.jug-ostfalen.de

Unter gewissen Umständen ergibt eine Migration von früheren Java-Versionen durchaus Sinn. Der Artikel zeigt, wann es sich lohnt und was dabei zu beachten ist.

Aus Alt mach Neu

Markus Eisele, msg systems ag

Die Enterprise Java Edition 5 ist schon seit einigen Jahren auf dem Markt, und auch für den Nachfolger mit der Versionsnummer 6 ist die Referenz-Implementierung bereits verfügbar. Die Zeit der Applikations-Server, die die Java 2 Enterprise Edition unterstützen, ist abgelaufen. Dazu gehören bei Oracle der OC4J genauso wie die Bea WebLogic Plattform (7.x – 10.2). Spätestens mit dem Erscheinen der zur Java Enterprise Edition 6 kompatiblen Version von WebLogic gehen die alten Produktlinien End-of-Life (EOL) im Extended Support. Auch wenn dies konkret für WebLogic 8.1 noch bis zum März 2011, für 9.x bis November 2013 und 10.2 sogar bis März 2015 dauert [1], sollte man sich aktuell schon die Frage stellen, was mit den etablierten Anwendungen älterer Bauart dann passieren soll. Sie alle müssen auf den Prüfstand und es ist zu entscheiden, ob beziehungsweise wie man diese migriert. Neben rein technischen Aspekten spielen hier noch weitere Dinge eine entscheidende Rolle. Den einzig richtigen Weg gibt es nicht, allerdings lässt sich anhand von Beispielen relativ einfach zeigen, welche Optionen es gibt und welche Wege sich ergeben. Die Ansätze hierzu sind vielfältig: Unter welchen Umständen ist es möglich und sinnvoll, alte Konzepte zu übernehmen? Welche neuen Technologien sind verpflichtend? Wie sieht die richtige Entscheidung zur Migration aus?

Strategische Überlegungen

Allem voran steht die Frage nach dem Sinn: Warum sollte man alte Anwendungen überhaupt anpacken? Vielfach ergibt sich der Bedarf aus Umständen von Produkt-Kompatibilitäten und Support-Verträgen. So wird WebLogic 10.0 beispielsweise nicht mehr für das aktuelle SUSE Enterprise Linux 11 unterstützt. Ist man daher gezwungen, ein Update der Hardware durchzuführen und kommt damit nicht um das aktuelle Betriebssystem herum, steht man schon vor der Entscheidung. Somit ist der erste Schritt die Betrachtung der aktuell in Betrieb befindlichen Komponenten mit den jeweiligen Versionen unter Berücksichtigung der vereinbarten Support-Umfänge. Sind nicht alle Anwendungen betroffen oder ist keine generelle, unternehmensweite Entscheidung für eine zukünftige Ziel-Plattform im Hinblick auf Produkt oder Technologie möglich, kann nur zu einer separaten Betrachtung der einzelnen Migrationskandidaten geraten werden.

Grundsätzlich sind bei einer Migration auf eine neue Enterprise Java Version drei verschiedene Szenarien denkbar:

- Beibehalten des bisherigen Standards
- Mix zwischen Alt und Neu
- Komplette Migration auf neuen Standard

Behält man den bisherigen Standard bei, vertraut man auf die sogenannten „Kompatibilitätsmodi" der jeweiligen Produkte. Ein aktueller WebLogic 11*g* (10.3.3.0) ist beispielsweise durchaus in der Lage, Container Managed Persistance (CMP) Entity Beans auszuführen, auch wenn diese Technologie in der Java EE 5 durch die Java Persistance API (JPA) abgelöst wurde. Im einfachsten Fall sind dann nur ein paar Deployment-Deskriptoren auszutauschen und die Anwendung auf dem neuen Server zu deployen. Klar muss allerdings sein, dass man sich damit lediglich ein paar weitere Jahre Laufzeit sichert. Dieses Szenario kommt also maximal für Anwendungen in Betracht, die eine fest definierte Laufzeit haben und für die keinerlei Weiterentwicklung geplant ist.

Ein Mix von Standards ist vielfach notwendig, wenn es sich um große Wartungsprojekte handelt oder ein Produktwechsel im Spiel ist. Soll eine Anwendung nicht nur technologisch auf den neuesten Stand gebracht, sondern auch beispielsweise vom GlassFish auf den WebLogic umziehen, ist auch der einfache Kompatibilitätsmodus vielfach keine Option mehr, da in der J2EE noch vergleichsweise viel Raum für herstellerspezifische Erweiterungen vorhanden war. Ein einfaches Ändern der Deployment-Deskriptoren bleibt zwar auch hier theoretisch möglich, scheitert in der Praxis allerdings an zu vielen Stellen. Bei großen Projekten hat der Mix von Alt und Neu vor allem vor dem Hintergrund des Wartungs-Budgets Vorteile. Hier sollte man daher problemorientiert vorgehen, was konkret bedeutet, dass zumindest die Persistenz-Schicht aktualisiert wird und der Rest mehrheitlich unangetastet bleibt.

Eine komplette Umstellung auf alle beteiligten Java-EE-Standards kann schnell einen Aufwand von mehrstelligen Personentagen verursachen. Neben Anpassungen in der Business-Logik (CMP auf JPA oder EJB 2.x auf EJB 3.0) sind auch die Anwendungs-Oberflächen komplett umzubauen (JSP auf JSF). Vielfach ist hier auch die Unternehmens-IT Impulsgeber und die Projekte müssen den Technologiesprung mitmachen. Dennoch bleibt auch hier der Rat zum Pragmatismus. Nur in wenigen Ausnahmefällen lohnt sich der Kraftakt wirklich.

Analog eines Entscheidungsdiagramms (siehe Abbildung 1) lässt sich schnell die grundsätzliche Richtung der Migrationsbemühungen festlegen. In der überwiegenden Mehrheit der Fälle dürfte es auf einen Mix von Alt und Neu herauslaufen. Lediglich in besonders performancekritischen Projekten oder bei jenen, die sich aktiv in Weiterentwicklung befinden, lohnt sich eine komplette Modernisierung.

Grundsätzliches Vorgehen

Steht das Szenario fest, macht man sich über das Vorgehen Gedanken. Der vorgestellte Fünf-Phasen-Plan (siehe Abbildung

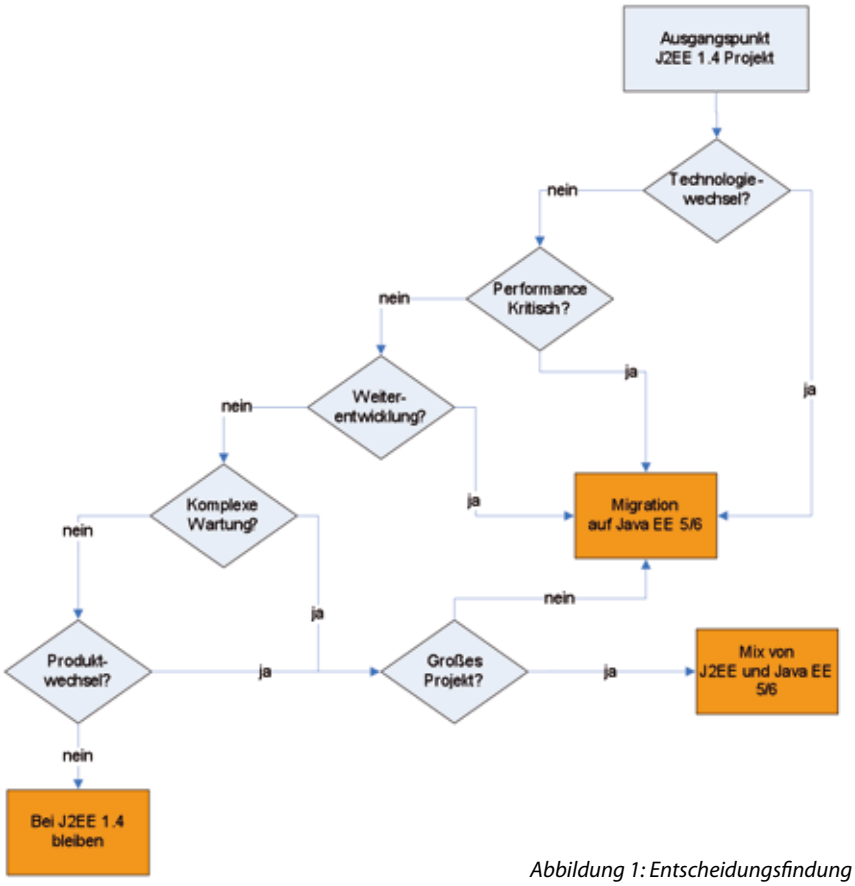


Abbildung 1: Entscheidungsfindung

2) umfasst die wichtigsten Aktivitäten für die jeweiligen Szenarien. In Abhängigkeit vom Szenario sind allerdings pro Phase entsprechend mehr Dinge zu tun.

Begonnen wird mit dem konkreten Überprüfen und Inventarisieren der Anwendung. Dabei sollten alle relevanten Themen im Fokus stehen. Eine J2EE-Anwendung besteht aus unterschiedlichen Anteilen – neben Konfigurationsdateien und Frameworks auch aus den Java-Sourcen der Anwendung. Zumeist findet auch ein umgebungsspezifischer Build statt. Manchmal werden Funktionen der jeweiligen Applikations-Server-Plattform verwendet, die in einer neuen Version ausgetauscht werden müssen, um die Ablauffähigkeit zu garantieren. Alle genannten Anteile sind für die Migration in geeigneter Weise anzupassen. In dieser Migrationsphase sollte daher eine komplette Liste aller eingesetzten Frameworks und Komponenten erstellt werden. Dazu gehört auch die Untersuchung des Sourcecodes auf eventuelle Inkompatibilitäten auf Ebene der JDK-Version. Die Praxis zeigt, dass man hier an nur wenigen Stellen eingreifen muss. So sind mit dem JDK 1.5 beispielsweise neue Klassen hinzugekommen (java.net.Proxy). Auch die Einführung von typsicheren Enumerationen sorgt für Probleme. Der Variablenname „enum“ darf nun nicht mehr auftauchen. Oracle stellt hierzu eine detaillierte Liste bereit [2]. Ebenfalls problematisch sind sogenannte „deprecated features“. Dies sind Funktionalitäten, die beim Wechsel von einer Produktversion auf die nächste weggefallen sind. Für den Oracle WebLogic Server finden sich diese Informationen klassischerweise leider nur zwischen den Zeilen in den jeweiligen „What's New“-Dokumenten. Der GlassFish v3 bringt ein wenig Ordnung mit

Abbildung 2: Migrationsschritte

einer eigenen API-Doc-Seite [3]. Am Ende dieser Phase steht eine konkrete Liste aller Dinge, welche für das konkrete Migrationsvorhaben zu ändern sind.

Dieses Verzeichnis stellt auch automatisch die To-do-Liste für die konkreten Anpassungen dar. Im gleichen Zuge ist natürlich auch der Build der Anwendung anzupassen. Neben eventuellen neuen Server-Bibliotheken sind auch die Framework-Abhängigkeiten anzupassen. Egal um welche Art von Migration es sich handelt, eine Adaption der aktuellen Build-Werkzeuge lohnt sich immer. Der Autor ist bekennender Maven-Anhänger und rät daher zur konsequenten Umstellung aller alten ANT-Builds. Berücksichtigt werden müssen beim Anpassen des Builds darüber hinaus auch entsprechende Staging-Konzepte oder Vorgaben für die Auslieferung.

Tests stellen einen elementaren Bestandteil jeder Migration dar. Je nach Szenario kann hier von vereinfachten, technischen Tests bis hin zum kompletten, fachlichen Regressionstest alles vorkommen. Ein absolutes Minimum stellt dabei die korrekte Lauffähigkeit der JUnit-Tests dar. Diese sind daher in dieser Phase auch konkret anzupassen. Empfehlenswert ist auch die Durchführung von Last- und Performance-Tests. Hierbei werden nicht selten produktionskritische Probleme aufgedeckt, die noch im Rahmen der Migration zu beheben sind. Alle weiteren Test-Abstufungen bis hin zum vollumfänglichen Regressionstest sind vom Sicherheitsbedürfnis der Verantwortlichen abhängig und lassen sich daher auch nicht generisch in diesem Rahmen definieren.

Gerne vergessen wird auch das Update der Dokumentation. Auch ist zwischen einem simplen Update der Release-Notes bei der Verwendung des J2EE Kompatibilitätsmodus' bis hin zur kompletten Überarbeitung der Projekt-Dokumentation (Design, Architektur, Betrieb etc.) alles möglich. Diese Phase liegt bewusst vor dem tatsächlichen Deployment, um ihr entsprechende Bedeutung zuzumessen.

Deployment im Sinne dieses Phasenmodells stellt die erfolgreiche Übergabe der Anwendung in den Linienbetrieb dar. Wie dies konkret und auf welche(r) Umgebung(en) geschieht, ist stark geprägt von den jeweiligen Unternehmensvorgaben.

Probleme und Risiken

Bei jeder Migration gibt es Gefahren und Stolpersteine. Auch wenn man diese nie vollständig im Voraus benennen kann, sollte man sich auf die wichtigsten einstellen. Ihr Vorhandensein und die Auswirkungen auf die konkrete Migration sind direkt abhängig von der Komplexität des Projekts und sie treten daher auch in unterschiedlicher Schwere zu Tage.

Das größte Problemfeld stellen die herstellerspezifischen Erweiterungen dar. Sind diese im Zuge einer Migration ohne Produktwechsel vielfach noch beherrschbar, stellen sie anderenfalls ein großes Risiko für die Migration dar. Gerade im Umfeld der J2EE ließ der Standard den Herstellern noch vergleichsweise viele Freiheiten bei der Implementierung unterschiedlicher Spezifikationen. Somit ist es nur wahrscheinlich, dass die besonderen Funktionen auch genutzt wurden. Was auf dem WebLogic neueren Datums dann auch vielfach funktioniert, existiert beispielsweise auf dem brandaktuellen GlassFish überhaupt nicht. Verdächtige Funktionen sind hier nur mit Spezialwissen zu identifizieren. Für eine Anwendung, die im WebLogic Server läuft, könnten das beispielsweise folgende sein:

- Weblogic-eigene JSP Tag Libraries
- Verwendete Packages com.bea.* oder weblogic.*
- Verwendung der Split development Structure

Bereits mit einfachen Open-Source-Frameworks können Probleme auftreten. So sind beispielsweise Apache-Struts-Versionen vor 1.0 nicht mit dem JDK 1.5 lauffähig. Konkret steht hier eine Abhängigkeit zu einer Funktionalität aus dem JDK 1.3 im Weg, die erst mit neueren Framework-Versionen behoben wurde. Daher ist es unumgänglich, alle im Projekt vorhandenen Frameworks auf ihre Kompatibilität mit der Zielplattform genauer zu untersuchen.

Die größten Risiken bergen allerdings hausgemachte Themen. Allen voran steht eine schlechte Code-Qualität. Was unübersichtlich und wenig nach OO-Prinzipien designed wurde, birgt dabei die größten Probleme. Diese Situation kann nur noch durch das Fehlen jeglicher automatisierter Tests verschlimmert werden. Trifft man auf einen solchen Problemfall, bleibt ungeachtet aller vorgenannten Überlegungen zumeist nur die Entscheidung zwischen „so weitermachen wie bisher" oder „komplett neu machen". Ähnlich dramatisch können sich eine schlechte Architektur, das Fehlen von Designpattern oder schlechte beziehungsweise gar keine Dokumentation auswirken.

Fazit

Der Aufwand einer Migration hängt grundlegend von der Architektur der Anwendung ab. Dabei spielen viele unterschiedliche Kriterien eine Rolle. Schwerpunkt von gemischten oder kompletten Java-EE-5/6-Migrationen bildet sicherlich die Persistenzschicht. Ist diese bereits in der Vergangenheit durch die Verwendung von Java Design Patterns (Session Facade, Data Transfer Object etc.) entsprechend gekapselt worden, ist jede Migration beherrschbar. Dennoch bleiben Risiken, die man durch stringente Überwachung minimieren kann.

Weitere Informationen

[1] Lifetime Support Policy: Oracle Fusion Middleware Products (PDF): http://www.oracle.com/support/library/brochure/lifetime-support-middleware.pdf

[2] Incompatibilities in Java SE 6.0 (since 1.4.2): http://java.sun.com/javase/6/webnotes/compatibility.html

[3] GlassFish v3 Deprecated API: http://javadoc.glassfish.org/v3/apidoc/deprecated-list.html

Kontakt:
Markus Eisele
markus.eisele@msg-systems.com

Markus Eisele arbeitet im Bereich Software-Technologie im Center of Competence IT-Architecture der msg systems ag. Bei seiner täglichen Arbeit begleitet er Kunden und Projekte auf dem Weg durch die Tücken neuer Technologien und Produkte im Java-EE-Umfeld (siehe auch Seite 13).

Dieser Artikel gibt einen kurzen Überblick über den Aufbau einer Entwicklungsumgebung mit NetBeans 6.9, GlassFish v3 und PostgreSQL 8.4. Die Grundlage für die Entwicklungsumgebung bildet eine moderne Drei-Schichten- Architektur. Es werden die wichtigsten Konfigurationspunkte dargestellt und Lösungsmöglichkeiten dafür gezeigt.

Entwicklungsumgebung mit NetBeans, GlassFish v3 und PostgreSQL

Gunther Petzsch, Saxonia Systems AG

Als Datenhaltungsschicht wird eine virtuelle Maschine mit einer PostgreSQL-Datenbank in Version 8.4 eingesetzt. Die Logikschicht ist der Applikationsserver GlassFish v3. Die Präsentationsschicht soll mittels NetBeans 6.9 entwickelt werden (siehe Abbildung 1). Darüber hinaus soll NetBeans die Entwicklung durch die Verwaltung der Datenbank und des Applikationsservers erleichtern. Als Hostbetriebssystem wird Windows eingesetzt.

PostgreSQL-Installation unter JeOS und VirtualBox

Das Betriebssystem JeOS ist eine minimale Server-Distribution von Ubuntu und frei benutzbar. Es muss lediglich das entsprechende Server-Image von der Ubuntu-Webseite heruntergeladen und installiert sein. Als virtuelle Umgebung wird aufgrund guter Benutzbarkeit und Stabilität bei der Benutzung Virtualbox verwendet. Die Virtualmaschine läuft mit 192 MByte RAM und einer Festplattenkapazität von 1,2 GByte. Dies sollte aktuelle Computersysteme nicht überfordern oder den laufenden Betrieb merklich stören. Ferner muss die virtuelle Maschine (JeOS) über eine Netzwerkverbindung zu dem Hostsystem verfügen bzw. ins Netzwerk eingebunden sein.

Abbildung :1 Schichtenmodell

PostgreSQL stellt an sich selbst den Anspruch, die beste Open-Source-Datenbank zu sein. Sie ist in vielen Anwendungsgebieten einsetzbar. Einen speziellen Ruf hat sich die Datenbank im Anwendungsgebiet von Geo-Informationssystemen gemacht.

Die Datenbank PostgreSQL 8.3 liegt für JeOS standardmäßig als deb-Paket im Ubuntu-Software-Repository bereit. Die aktuelle PostgreSQL-Version 8.4 ist nur im Backports-Repository für JeOS vorhanden und kann somit auch genutzt werden. Um diese Backports nutzen zu können, müssen folgende Einträge in der Datei /etc/apt/source.list eingefügt werden:

- deb http://de.archive.ubuntu.com/ubuntu hardy-backports main restricted universe multiverse
- deb-src http://de.archive.ubuntu.com/ubuntu hardy-backports main restricted universe multiverse

Die Datenbank wird dann durch folgenden Befehl installiert:

```
gpetzsch@jeos:~$ sudo apt-get
install postgresql
```

Dabei ist zu beachten, dass der Betriebssystem-Anwender den „sudo“-Befehl ausführen darf und die Paketverwaltungssoftware „apt“ entsprechend konfiguriert ist. Dies sollte aber nach der Ubuntu-Installation ohne Zutun funktionieren. Weiterhin wird durch den oben genannten Befehl immer die aktuell für das Betriebssystem verfügbare Software-Version aus dem Repository geholt und installiert.

PostgeSQL wird unter Ubuntu von einem speziellen User (postgres) verwaltet. Dieser besitzt alle Rechte, um die Datenbank zu verwalten. Ein neuer Benutzer kann mit folgendem Befehl angelegt werden:

```
gpetzsch@jeos:~$ sudo sudo -u
postgres createuser -P Nut-
zername
```

Dabei ist zu beachten, dass die Befehlszeile mit „sudo sudo“ beginnt, da man sonst nicht an die Rechte des Postgres-Users kommt. Mit dem Parameter „P“ wird auch die Eingabe eines Passwortes für den neuen Benutzer verlangt.

Nun wird eine neue Datenbank für den im vorherigen Schritt erstellten User angelegt:

```
gpetzsch@jeos:~$ sudo sudo -u
postgres createdb -O Nutzername
Datenbank
```

Nach der Installation von PostgreSQL akzeptiert die Datenbank keine Verbindung aus dem Netzwerk. Um dies zu ermöglichen, müssen noch Anpassungen in der Datei /etc/postgresql/8.4/main/postgresql.conf durchgeführt werden.

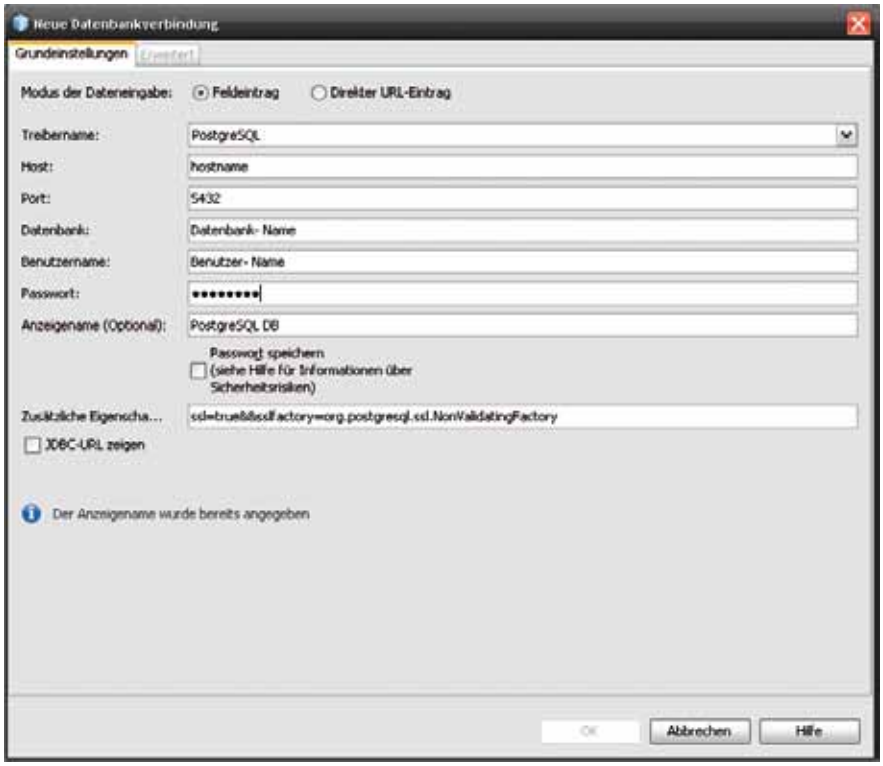


Abbildung 2: Datenquellen anbinden

```
listen_adressen =
`192.168.56.1`
```

Mit dieser Änderung wird dem Rechner mit der IP 192.168.56.1 erlaubt, eine Verbindung zur Datenbank aufzunehmen. Weiterhin sollte man eine sichere Authentifizierung für die Verbindung zur Datenbank konfigurieren. Diese Einstellungen kann man in der Datei /etc/postgresql/8.4/main/pg_hba.conf vornehmen. Dabei sind folgende Optionen möglich:

- Muss die Verbindung verschlüsselt sein?
- Welche Datenbank darf benutzt werden?
- Welcher Benutzer kann die Verbindung herstellen?
- Von wo aus darf die Verbindung hergestellt werden?
- Welche Methode soll zur Authentifizierung benutzt werden?

Folgende Einstellungen werden hier verwendet.

```
hostssl Datenbank Nutzername
191.168.56.1/24 md5
```

Die Verbindung soll mittels SSL verschlüsselt sein und nur der erstellte Benutzer darf eine Verbindung zur Datenbank aufbauen. Damit alle Änderungen aktiviert werden, muss man die Datenbank neu starten.

```
gpetzsch@jeos:~$ sudo /etc/
init.d/postgresql-8.4 restart
```

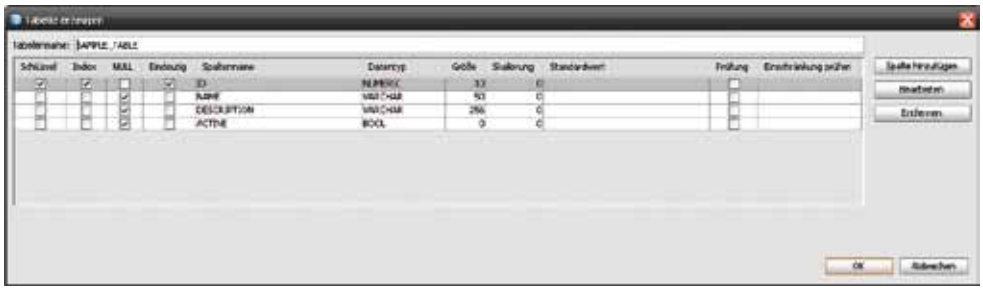


Abbildung 3: Tabelle erstellen mittels NetBeans

Mit diesem letzten Schritt ist die Datenbank fertig konfiguriert und kann nun an eine IDE angebunden werden.

NetBeans 6.9

NetBeans IDE ist ein mächtiges Werkzeug für Entwickler. Sie unterstützt viele Programmiersprachen und ist ohne Nutzungsbeschränkungen kostenlos zu benutzen. Durch eine weltweite NetBeans- Gemeinschaft wird die Plattform kontinuierlich verbessert und ausgebaut. Der Autor empfiehlt die Version für Java-Entwickler (NetBeans-6.9-ml-javase-windows.exe), Die deutsche Version ist etwa 66 MB groß und lässt sich auf Wunsch beliebig erweitern. Zur Installation startet man einfach die Setup-Datei und folgt dem Installationsdialog.

Neben dem guten Quellcode-Editor bietet NetBeans auch die Möglichkeit, bestimmte Datenquellen direkt anzubinden und zu verwalten. Über den Reiter „Dienste“ kann man diverse Datenquellen in NetBeans integrieren. Folgende Datenbanken kann man generell ohne zusätzliche Treiber einbinden:

- JAVA DB
- MySQL
- PostgreSQL
- JDBC-ODBC Bridge

Andere Datenquellen können mit den entsprechenden Treibern ebenfalls angebunden werden. Um die PostgreSQL-Anbindung zu realisieren, wählt man im Reiter „Dienste“ unter dem Menüpunkt „Datenbanken“ über die rechte Maustaste eine „neue Verbindung“ aus. Es erscheint der Verbindungsdialog (siehe Abbildung 2). Dort gibt man folgende Daten ein:

- Treibername: PostgreSQL
- Host: entsprechender Hostname von Ihrem System bzw. IP- Adresse
- Port: Standardport von PostgreSQL ist 5432.
- Datenbankname: Name der gewählten Datenbank
- Benutzername: Name des Datenbanknutzers
- Anzeigename: beliebiger Name Ihrer Wahl

- Zusätzliche Eigenschaften: „ssl=true&sslfactory=org.postgresql.ssl.NonValidatingFactory“

Wichtig ist der Punkt „Zusätzliche Eigenschaften“. Wenn die Verbindung zur Datenbank mittels SSL verschlüsselt werden soll, muss dies in NetBeans konfiguriert sein.

Nachdem alle Daten eingegeben wurden, kann man noch ein bevorzugtes Datenbank-Schema auswählen. Hier wird am besten das Schema für den angelegten Nutzer verwendet. Wenn die Verbindung erfolgreich aufgebaut ist, kann man die Schema- und Tabellenstruktur der PostgreSQL-Datenbank in NetBeans sehen.

Über NetBeans 6.9 ist es auf einfache Weise möglich, in der PostgreSQL-Datenbank Datenstrukturen anzulegen. Besteht eine Verbindung zur entsprechenden Datenbank, werden alle vorhandenen Tabellen, Views und Prozeduren pro Datenbankschema sortiert angezeigt. Um eine neue Tabelle zu erzeugen, klickt man mit der rechten Maustaste auf den Tabellen-Ordner und wählt „Tabelle erstellen“. Es erscheint der Dialog zum Anlegen einer neuen Tabelle (siehe Abbildung 3).

Zudem kann man über ein SQL-Formular Datenbank-Befehle aus NetBeans heraus ausführen. Dies ist beispielsweise beim Füllen oder Anzeigen von Tabellenstrukturen hilfreich. Abbildung 4 zeigt ein Beispiel für ein Insert- und Select-Statement.

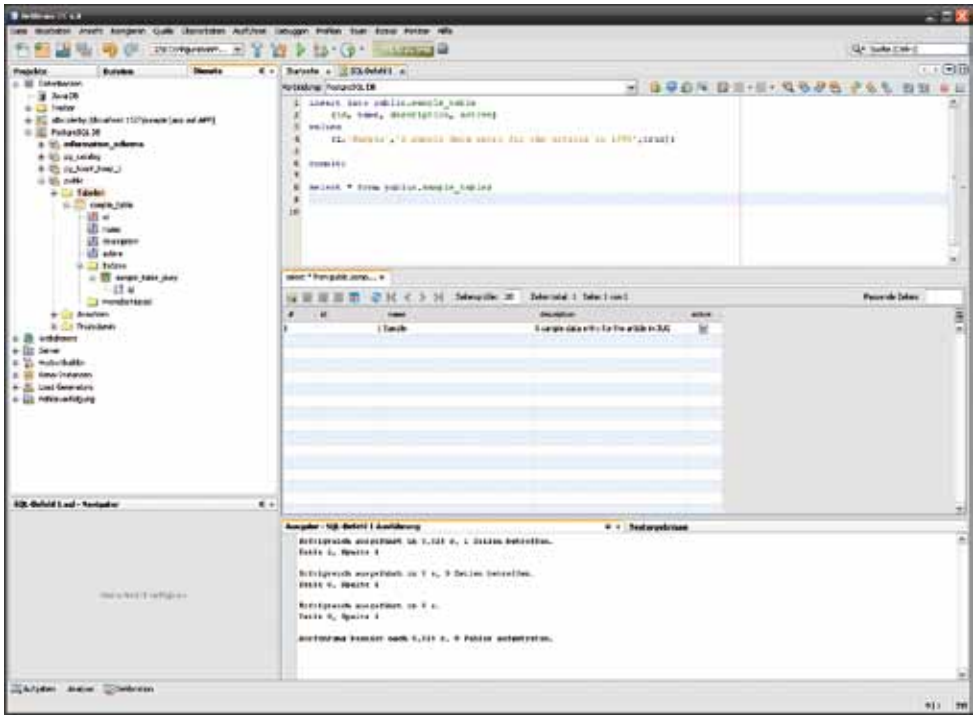


Abbildung 4: Insert- und Select-Statement mittels NetBeans

GlassFish v3

GlassFish v3 ist der neue Applikationsserver von Oracle. Er ist in der Open Source Edition und der Oracle GlassFish Server Edition verfügbar. Nachfolgend ist die Verwendung der Open Source Edition beschrieben. Der GlassFish-Server besticht durch seinen modularen und erweiterbaren Aufbau (OSGi). Zudem wird er von einer freien Community unterstützt und weiterentwickelt.

GlassFish lässt sich unkompliziert in NetBeans integrieren und verwalten. Im Reiter „Dienste“ befindet sich der Menüknoten „Server“. Über diesen können beliebige Applikationsserver in NetBeans integriert werden. Ist dieser Menüknoten nicht vorhanden, muss man noch das entsprechende Plugin installieren. Dies kann man über Extras->Plugin->Verfügbare Plugins durchführen. Dabei wählt man das „Java Web Applikation“-Plugin aus und installiert es. Nach einem Neustart der IDE steht der Menüknoten dann zur Verfügung.

Nun wählt man über die rechte Maustaste „Server hinzufügen“ und der entsprechende Einrichtungsdialog wird angezeigt. Im ersten Schritt wählt man „GlassFish v3 Server“ aus. Befindet sich schon ein installierter GlassFish auf dem System, kann man diesen über die Auswahl des Verzeichnisses einbinden. Ist noch keine Installation vorhanden, kann dieser direkt über NetBeans heruntergeladen und in ein gewähltes Verzeichnis installiert werden (siehe Abbildung 5). Dazu müssen im Dialog der Installationspfad ausgewählt, die Lizenzvereinbarungen gelesen, akzeptiert und der Button „Herunterladen“ be-

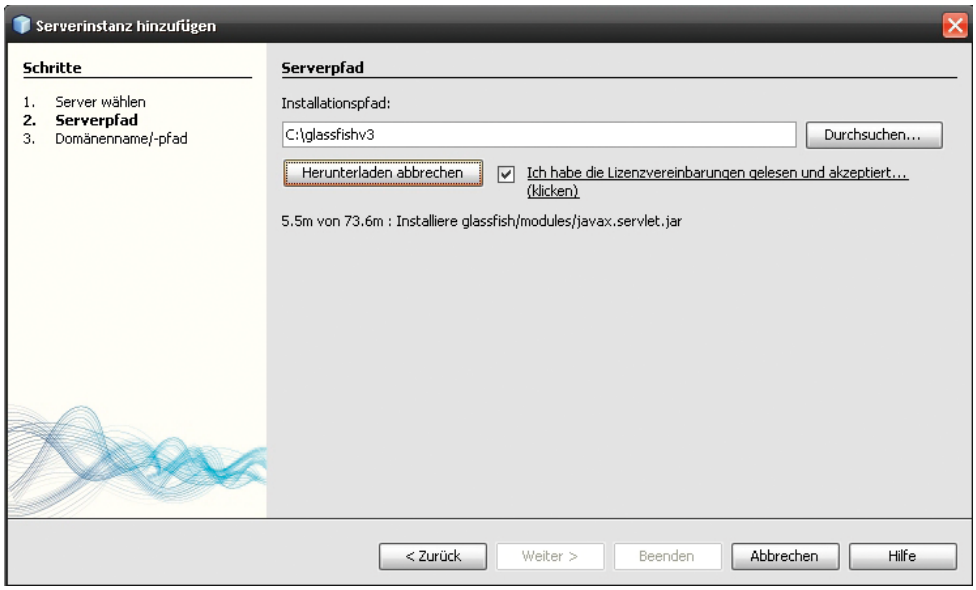


Abbildung 5: GlassFish-Installation aus NetBeans

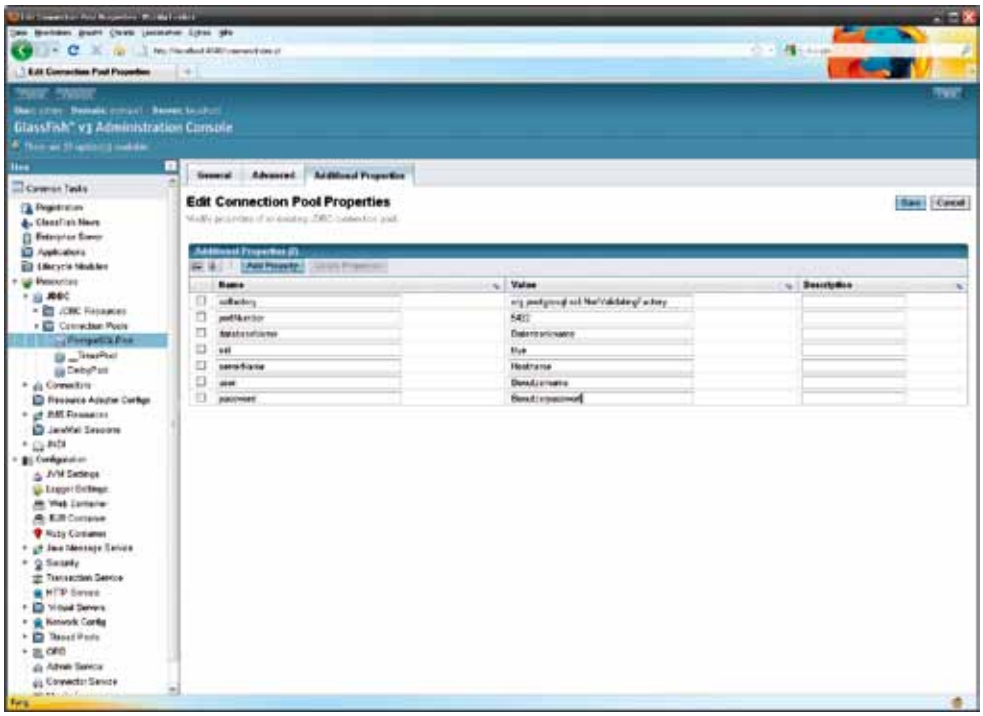


Abbildung 6: Connection Pool Properties

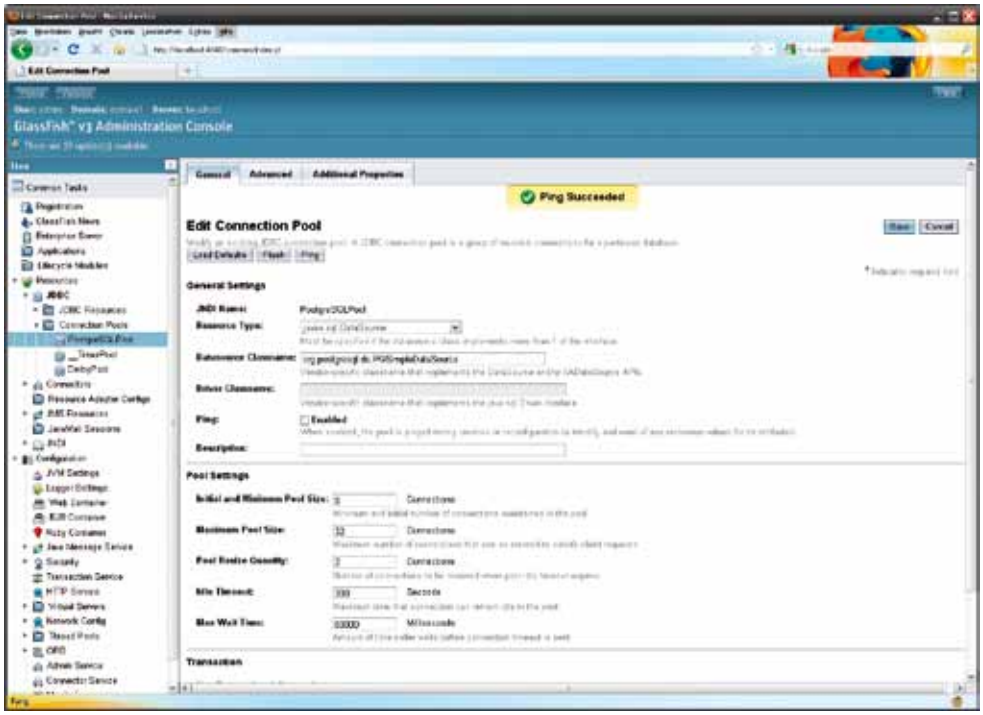


Abbildung 7: Ping zur Datenquelle war erfolgreich

tätigt werden (Internetverbindung muss vorhanden sein). GlassFish wird dann aus dem Internet heruntergeladen und in das ausgewählte Verzeichnis installiert. Ist dies geschehen, muss im nächsten Schritt ein Name für die Domäne vergeben werden. Im Standardfall ist dies „Domain1". Der Applikationsserver ist danach ohne weitere Einstellungen funktionsbereit. Das Starten, Stoppen und Administrieren des Servers kann bequem über das Kontextmenü in NetBeans vorgenommen werden.

Um eine Datenquelle mit GlassFish nutzen zu können, müssen generell folgende Schritte durchgeführt werden:

1. Einbinden des JDBC- Treibers für die jeweilige Datenbank
2. Connection Pool anlegen
3. JDBC- Ressource anlegen

Die Treiber für die PostgreSQL-Datenbank sind im Internet unter www.jdbc.postgresql.org zu finden. Es gibt zwei Versionen, den JDBC3- und den JDBC4-Treiber. Ab Java-Version 6 sollte man den JDBC4-Treiber (postgresql-8.4-701.jdbc4.jar) verwenden. Dieser muss in das Verzeichnis „lib/ext" der entsprechenden Domain kopiert sein, in der die Datenbank angebunden werden soll. In der Regel ist das die Domain1. Im vorliegenden Beispiel lautet der Pfad „C:\GlassFishv3\GlassFish\domains\domain1\lib\ext".

Das Anlegen eines Connection Pools geschieht in der Administrationskonsole von GlassFish v3 in zwei Schritten. Im ersten Schritt werden folgende Einstellungen getroffen:

- Name: z.B. PostgreSQLPool
- Resource Type: „javax.sql.DataSource" auswählen
- Database Vendor: „Postgresql" auswählen

Im zweiten Schritt muss man die Optionen für die Datenbank-Anbindung ähnlich wie in NetBeans angeben. Dabei werden die meisten Attribute vorgegeben, es müssen aber noch die entsprechenden Werte dazu gesetzt sein (Siehe Abbildung 6).

Zwei Werte müssen in diesem Beispiel noch hinzugefügt werden:

- ssl
- sslfactory

Dabei steht „ssl" für die Verschlüsselung der Verbindung zur Datenbank und die „sslfactory" für die dazu verwendete Java-Klasse. Das Attribut „ssl" muss auf dem Wert „true" und „sslfactory" auf dem Wert „org.postgresql.ssl.NonValidatingFactory" stehen. Sind diese Werte nicht gesetzt, kann keine Verbindung zur Datenbank aufgebaut werden, und es wird eine entsprechende Fehlermeldung angezeigt. Hat man diese Einstellungen durchgeführt und gespeichert, kann

man die Verbindung zur Datenbank testen. Dies kann mittels eines Pings aus GlassFish heraus erfolgen (Siehe Abbildung 7).

War die Erstellung der Connection Pools erfolgreich, müssen diese noch mit einer JDBC-Ressource verknüpft werden. Dazu erstellt man eine neue Ressource, beispielsweise mit dem Namen „jdbc/PostgreSQL", und hinterlegt den zuvor angelegten Connection Pool. Damit ist die Einrichtung der Datenbank-Verbindung in GlassFish v3 erfolgreich abgeschlossen und man kann mit der Entwicklung beginnen.

Weitere Informationen

- www.netbeans.org
- www.glassfish.dev.java.net
- www.postgresql.org
- www.ubuntu.com
- http://wiki.ubuntuuser.de/postgresql
- www.de.wikipedia.org/Schichtenarchitektur
- www.virtualbox.org

Kontakt:
Gunther Petzsch
gunther.petzsch@saxsys.de

Gunther Petzsch ist Sun zertifizierter Java-Entwickler. Er ist als Senior Consultant für die Saxonia Systems AG tätig.

Mitglieder stellen sich vor

Java User Group München (JUGM)

Die Java User Group München (JUGM) besteht seit dem Jahr 2000 und hat zur Zeit über 700 Mitglieder. Initiiert wurde die JUGM von der Regionalgruppe München der Gesellschaft für Informatik e.V. Die monatlichen Treffen widmen sich meist einem Thema, das in der Regel durch einen Vortrag präsentiert wird. In angenehmer Gesellschaft kommen so schnell angeregte Diskussionen zustande und die Abende enden oft erst gegen 22 Uhr. Die Firma mgm technology partners stellt kostenlos Räumlichkeiten und Getränke zur Verfügung.

Die Themen werden auf der Internet Seite angekündigt, und es ist jede/jeder herzlich eingeladen, an unseren Treffen teilzunehmen. Auf der Webseite finden sich die meisten Vorträge nachträglich im PDF-Format wieder. Zusätzlich kann das Wiki als Nachschlagewerk oder Diskussionsplattform genutzt werden.

Im Umfeld der JUGM sind Gruppen mit spezialisierter Ausrichtung entstanden:

- JBoss User Group München, http://www.jbug-munich.org
  Aus der Selbstbeschreibung: „Sie hat das Ziel, den regionalen Wissensaustausch zwischen Benutzern von JBoss-Technologien zu ermöglichen bzw. zu unterstützen und ist damit bis jetzt einzigartig in Deutschland."
  Zum 1. Oktober 2010 hat die JBUG München eine Konferenz für Java Frameworks & Enterprise Edition Technologien, http://www.onedaytalk.org, organisiert.
- Groovy, Grails & Griffon User Group München, http://ggg.mixxt.de
  Aus der Selbstbeschreibung: „Die User Group soll ein lockerer Zusammenschluss von Interessierten im Umfeld von Groovy, Grails und Griffon im Großraum werden. Neben Vorträgen zu bestimmten Themen sollen auch der Erfahrungsaustausch und die Diskussion wichtig sein."
- Scala München, http://scala-southerngermany.mixxt.de/networks/groups/ScalaMuenchen/index in der Scala Users Southern Germany Community.
  Aus der Selbstbeschreibung: „Eine Gruppe für Scala-Enthusiasten und -Interessierte im Großraum München."

Weitere Informationen: http://www.jugm.de

Fortsetzung von Seite 5

Zu den Dingen, die man sich von Java erhoffte, gehörte die Möglichkeit, einmal geschriebenen Code überall auszuführen. Aber zwischen der Standard Edition und der Mobile Edition tat sich eine Kluft auf. Jetzt wird Oracle die API beider Editionen vereinheitlichen und dafür sorgen, dass Java, JavaFX und JavaScript reibungslos zusammenarbeiten. Damit steht Ihnen als Entwickler eine Fülle von Möglichkeiten zur Verfügung und Sie können Ihre Anwendungen schneller erstellen. Unseren Gerätepartnern und den Netzbetreibern werden wir auch in Zukunft wie gewohnt optimierten Binärcode zur Verfügung stellen.

Mit JavaFX haben wir zahlreiche Neuerungen in das Java-System eingeführt. Ein Bereich, den wir bisher vernachlässigt haben, ist die Steigerung der Entwicklungsproduktivität und die Erstellung von sogenannten „Rich Internet Applications". Dafür gibt es jetzt JavaFX. Wenn Sie eine designorientierte Website oder Anwendung visuell entwickeln möchten, sollten Sie sich an JavaFX halten. Wenn Sie Ihre Produktivität steigern und in kürzester Zeit eine beeindruckende Benutzeroberfläche erstellen möchten, ist JavaFX das Richtige. Wir haben jedoch noch einen weiteren Schwerpunkt gesetzt. Die Funktionsgrenzen zwischen den Geräten verschwimmen zunehmend: Es gibt Fernseher mit Laufzeitumgebung. Immer mehr Menschen möchten sich Fernsehsendungen nicht mehr nur auf dem Fernseher ansehen, sondern auch auf einem Bildschirm im Auto oder auf ihrem Mobiltelefon. Deshalb wollten wir die Möglichkeit schaffen, eine Anwendung zu erstellen, die auf allen diesen Geräten ausgeführt werden kann. JavaFX macht das für Java möglich und wird die Arbeit, die Oracle mit dem Application Development Framework und ADF Mobile geleistet hat, ergänzen.

Wir haben schon angesprochen, dass Java allgegenwärtig ist. Nicht viele Menschen wissen, wo Java überall eingesetzt wird: in Satelliten, militärischen Systemen, kleinen Geräten, die die Temperatur messen und an Server senden, E-Book-Readern und Kassensystemen. Java ist unglaublich vielseitig. Deshalb wird Oracle seine Partnerschaften ausbauen und neue Gelegenheiten und Einsatzbereiche für Java suchen.

Es gibt auf dem Markt Milliarden von Java-Karten. Die meisten Menschen wissen jedoch nicht, dass Java Card auf über 90 Prozent des SIM-Marktes in Nordamerika, Südamerika und Europa installiert ist. Trotz des minimalen Platzes fungiert die Version Java Card 3.0 als echter Webserver, so dass Sie sogar Dienste einrichten können. Aber Java Card kann noch viel mehr.

In den letzten Jahren hören wir von vielen großen Firmenkunden immer wieder: „Wir haben Millionen von Dollar in Java investiert. Wir brauchen unbedingt Support für den Kernbereich von Java und wir müssen uns darauf verlassen können, dass wir für unsere Investitionen einen Support bekommen, der geschäftskritischen Anwendungen angemessen ist." Vor ungefähr einem Jahr haben wir deshalb ein Support-Programm ins Leben gerufen, das wir bei Oracle fortsetzen und ausbauen werden.

Sprechen wir über Ihre Zusammenarbeit mit Oracle, darüber, was sich in diesem Bereich tun wird. Die Entwickler sind das Herz von Java. Sie lesen die Websites Java.sun.com, java.com und java.net. Bitte besuchen Sie diese Websites auch weiterhin. Wir möchten Sie unterstützen. Wir möchten, dass unser Entwicklerstamm weiter wächst. Auch auf der Website von Oracle haben wir ein Portal für Sie eingerichtet. Oracle kann seine Kunden und Partner auf vielerlei Weise unterstützen. Unter oracle.com/java finden Sie eine Fülle weiterer Informationen. Sie können sich auf ein Paket von Support- und Beratungsservices von Oracle freuen.

Wie Sie wissen, haben wir zahlreiche Kunden, die Java in geschäftskritischen Anwendungen und in ihrer Entwicklungsumgebung einsetzen. In diesen Bereich werden wir in Zukunft noch mehr investieren, damit Sie Ihren Kunden für ihre geschäftskritischen Systeme hochwertigere Dienste und Produkte anbieten können. Viele unserer Partner haben einen Beitrag zu Java geleistet. Auch dafür werden wir weiterhin Support anbieten. Sehen Sie sich auch partner.oracle.com an, wo Sie eine Menge zusätzlicher Informationen und Ressourcen finden. Wir freuen uns auf die Zusammenarbeit mit Ihnen allen und möchten unsere Partnerschaft in Zukunft intensivieren.

Auch die JavaOne wird weiterhin stattfinden. Die JavaOne wird die wichtigste offene Java-Konferenz bleiben, die Entwicklergemeinde zusammenbringen und Sie mit den neuesten Informationen vertraut machen. In diesem Jahr halten wir sie zusammen mit der Oracle OpenWorld in der Woche ab dem 19. September 2010 ab. Wir haben vor, die Reichweite der JavaOne zu vergrößern, sie sozusagen auf Tournee zu schicken und näher zu Ihnen zu bringen. Für dieses Jahr haben wir bereits Brasilien, Indien, China und Russland eingeplant. Wir rufen die Entwickler auf, Beiträge einzureichen, und laden sie zur Teilnahme und zum Wissensaustausch mit ihren Kollegen ein. Auch für die nächste JavaOne bitten wir um Beiträge. Deshalb freue ich mich besonders, dass wir dieses Konzept ausbauen und die JavaOne zu den Entwicklergemeinden in aller Welt bringen.

Fazit

Oracle wird die Investitionen in all die Standards, Editionen und Produkten rund um Java weiter vorantreiben. Beispielhaft sei hier nur das Thema „JavaFX" genannt, das angesichts der vielen Einsatzbereiche und Möglichkeiten aus unserer Sicht ein immens wichtiges Projekt ist, so dass wir es möglichst schnell praktisch nutzbar machen möchten. Wir werden die Entwicklergemeinschaft, Open-Source sowie den Java Community Process weiterhin unterstützen – mit dem Ziel eine noch größere Verbreitung von Java zu erreichen. Vor allem werden wir unseren Unterstützung für die Entwickler, als Säule der Open-Source- und Java-Welt, weiter aubauen.

Wir freuen uns sehr über unsere noch engeren Beziehung zur Java- Welt, auf die Zusammenarbeit mit Ihnen und die Innovationen, mit denen wir Java in Zukunft unterstützen und weiterentwicklen werden.

Weitere Informationen

Java Strategie WebCast: http://www.oracle.com/events/productstrategy/index.html [1]
Oracle und Sun: http://www.oracle.com/us/sun/index.htm
Java One: http://www.oracle.com/us/javaonedevelop/062264.html
Entwickler: java.sun.com, java.net, java.com, javafx.com, otn.oracle.com/java
Kunden: www.oracle.com/java
Partner: partner.oracle.com

Kontakt:
Kersten Mebus
kersten.mebus@oracle.com

Mit Clojure hat vor kurzer Zeit ein weiterer Spieler das Feld der alternativen Programmiersprachen für die JVM betreten, der bereits häufig mit Scala verglichen wurde. Dieser Artikel beschreibt anhand eines Ausschnitts aus dem für Herbst 2010 geplanten Buchs die REPL, die eine wichtige Grundlage dieses Entwicklungsmodells darstellt.

Clojure

Stefan Kamphausen

Fokus von Clojure ist die Unterstützung bei der Entwicklung nebenläufiger Programme. Dies erreicht Clojure durch Mischen verschiedener Technologien wie unveränderliche Daten, persistente Datenstrukturen und Software Transactional Memory zu einem stimmigen Gesamtbild. Clojure ist eine funktionale Programmiersprache. Sie erbt von Lisp neben der Syntax auch das dynamische Entwicklungsmodell.

Read-eval-print loop (REPL)

Entwickler kennen den gängigen Entwicklungszyklus als die immer wiederkehrende Wiederholung der folgenden Schritte:

- Editieren
- Kompilieren (entfällt bei Interpreter-Sprachen)
- Programm laufen lassen
- Fehler analysieren

In Clojure, wie auch in anderen Sprachen der Lisp-Familie, ist das anders. Hier startet der Anwender die Laufzeitumgebung der Sprache einmal und interagiert dann direkt mit dieser. Das lässt sich am ehesten mit der Arbeit in einer Shell wie der Bash vergleichen. An einem Prompt werden Befehle eingegeben, die dann ausgeführt werden und Dinge bewirken.

Etwas genauer betrachtet, nimmt der Reader, das „R" in REPL, den eingegebenen Befehl entgegen. Der Reader ist aber ebenso für das Lesen von Dateien mit Clojure-Quellcode verantwortlich und steht auch als Befehl „read" zur Verwendung in eigenen Programmen bereit. Dabei kennt der Reader Clojures komplette Syntax sowie alle darüber hinausgehenden Konstrukte. Das Ergebnis des Readers sind Daten, die evaluiert werden können, das „E" in REPL. Clojure ist *homoikonisch*, das heißt, der Code ist eine Datenstruktur der Sprache selbst. Quellcode in Clojure besteht (im Wesentlichen) aus Listen und Vektoren in einer Form, die der Reader versteht, und Listen sind ebenso Datenstrukturen in Clojure wie Vektoren. Im Schritt der Evaluation werden eventuelle Nebeneffekte realisiert und am Ende ein Resultat zurückgegeben. Dieses Ergebnis wird ausgegeben (Print, das „P" in REPL) und der ganze Vorgang wiederholt (Loop, „L").

```
shell> java -cp clojure.jar clojure.main
Clojure 1.1.0
user=> (println „Hello World")
Hello World
nil
user=> „Hello World"
„Hello World"
user=> (+ 2 3)
5
user=> (defn addier-2-zahlen [x y] (+ x y))
#'user/addier-2-zahlen
user=> (addier-2-zahlen 2 3)
5
user=> (quit)
java.lang.Exception:
  Unable to resolve symbol: quit in this context
user=> (System/exit 0) ;; oder drücke Strg-d
shell>
```

Listing 1

Für den Entwickler heißt das, dass er eine Funktion entwickelt und interaktiv testet, bis sie seinen Vorstellungen entspricht. Dazu wird er gegebenenfalls den Aufruf anderer Funktionen, die er in der eigenen verwenden möchte, ausprobieren und so den benötigten Aufruf finden. Diese Art zu arbeiten führt zu Funktionen, die besser getestet sind, weil sie schon während der Entwicklung immer wieder geprüft werden, gegebenenfalls auch mit verschiedenen Argumenten. Daneben haben solchermaßen entwickelte Funktionen die Tendenz, kürzer zu sein, was zu besser granuliertem Code führt.

Listing 1 zeigt, wie zunächst von der Shell aus die REPL von Clojure 1.1 gestartet wird, an der danach einige Befehle eingegeben werden, bis die Sitzung beendet wird.

Diese einfache REPL, die direkt auf der Shell gestartet werden kann, ist allerdings nicht sehr komfortabel. Sie dient dazu, beispielsweise die einleitenden Beispiele aus dem Grundkurs in Lisp (Abschnitt im Buch, Anmerkung des Autors) nachzuvollziehen. Aber sie hat weder eine Historie noch nennenswerte Editiermöglichkeiten auf einer Zeile. Es ist möglich, durch den Einsatz von „rlwrap" [1], „JLine" [2] oder ähnlicher Technologien einige fehlende Fähigkeiten nachzurüsten, allerdings lässt sich auch damit kein wirklich effizientes Arbeitsmittel einrichten. Zum Ausprobieren, Testen und zum Start produktiver Programme ist diese REPL ausreichend, für ernsthafte Entwicklung ist die Integration in eine IDE angebracht.

Da Clojure eine enge Beziehung zu Java hat, existieren Plugins für die Integration in NetBeans, Eclipse und IntelliJ. Die Verwandtschaft zu Lisp sorgt dafür, dass die Kombination von Emacs, SLIME

und Clojure-spezifischen Anpassungen derzeit die am häufigsten eingesetzte Umgebung ist. Interessanterweise hat sich Clojure auch im Umfeld von Vim etablieren können. Laut einer online durchgeführten Rundfrage [3] findet sich die Kombination von Vim mit Vimclojure sogar auf dem zweiten Platz, noch vor NetBeans mit Enclojure.

Durch die IDE-Integration gewinnt der Programmierer viele Funktionen hinzu:

- Syntax-Highlighting
- Automatisches korrektes Einrücken (sehr wichtig bei Lisp)
- Completion auf Clojure-Funktionen sowie die Namen von Java-Klassen und Methoden
- Zugriff auf die in Clojure eingebauten Hilfsfunktionen
- Direktes Kompilieren einzelner Funktionen aus dem Quelltext heraus
- Navigation im gesamten Quellcode auch größerer Projekte
- Templates für Quellcode
- Integration eines Debuggers

Da aber die jeweiligen Communities ihre Projekte zurzeit zügig weiterentwickeln, ist die Einrichtung einer solchen Umgebung in aktualisierbaren Dokumenten im Internet deutlich besser aufgehoben als in Buchform. Im weiteren Verlauf gehen wir daher davon aus, dass eine solche Umgebung vorhanden ist.

Für einen Clojure-Programmierer gestaltet sich der Entwicklungszyklus durch das Vorhandensein der REPL deutlich anders. Zunächst wird er eine Clojure-Sitzung starten. Diese kann durchaus tage- oder wochenlang ununterbrochen laufen, während permanent in ihr, am offenen Herzen, gearbeitet wird. Danach wird er eine Funktion schreiben, in der Regel in einer separaten Datei, deren Inhalt dann durch die IDE an Clojure weitergereicht wird. In der Kombination Emacs, SLIME und Clojure reicht dafür das Drücken der Tastenkombination „C-c C-k“ in Emacs-Notation, also „Strg“ und „c“ zusammen gedrückt, gefolgt von „Strg“ und „k“. Das geht schnell. Wenn bei der Entwicklung einer Funktion Aufrufe von Bibliotheksfunktionen notwendig sind, die sich dem Programmierer nicht sofort erschließen, wird er diese an der REPL testen. Sobald eine erste Version der zu entwickelnden Funktion vorliegt, kann diese ebenfalls interaktiv an der REPL getestet werden, wobei auch gleich verschiedene Kombinationen von Argumenten getestet werden; das schließt das explizite Testen von Fehlerszenarien mit ein. In den meisten Fällen wird eine Veränderung der Funktion notwendig sein. Dazu wird der Entwickler den Quelltext der Funktion umbauen und nur diese eine Funktion separat neu kompilieren („C-c C-c“; das geht noch schneller). Es ist kein Neubau des gesamten Projektes notwendig. Nach einigen Zyklen kann diese Funktion dann als fertig und gut getestet gelten. Wenn zur Unterstützung der Entwicklung ein Framework für automatisiertes Testen zum Einsatz kommt, können die interaktiv eingegebenen Befehle leicht als Basis für das Testskript dienen.

Diese Art zu arbeiten kann sich deutlich flüssiger anfühlen. Der angestrebte Flow – der Zustand, in dem Programme nahezu direkt aus den Gedanken entstehen, indem all die Werkzeuge, die tatsächlich zwischen der CPU des Menschen und der des Computers vermitteln, zu verschwinden scheinen – kann leichter erreicht werden. Es muss nicht auf langwierige Kompiliervorgänge gewartet werden, es muss kein künstliches Szenario geschaffen werden, um nur eine Funktion eines größeren Programms zu testen. Es gibt auch keine Probleme im Deployment, bei denen im Fehlerfall vorsichtshalber doch ein kompletter Neustart eines Servers durchgeführt wird, um Fehler im Hot-Deploy auszuschließen. Die direkte Arbeit mit dem Code umfasst jetzt nicht mehr nur das Bearbeiten sondern auch das Ausführen.

Links

[1] rlwrap: http://utopia.knoware.nl/~hlub/uck/rlwrap/

[2] Jline: http://jline.sourceforge.net/

[3] Umfrage zur Entwicklungsumgebung: http://acidrayne.net/?p=81

Kontakt:
Stefan Kamphausen
ska@clojure-buch.de

Clojure – Einführung und Praxis

Das Buch von Stefan Kamphausen und Tim Oliver Kaiser gibt eine Einführung in die Philosophie und Technik der jungen und auf Concurrency fokussierten Programmiersprache Clojure. Nach einem ausführlichen Einstieg in die Grundlagen der Sprache werden die Integration mit Java in beide Richtungen und die Besonderheiten in Bezug auf Concurrent Programming erläutert. Die Beschreibung wichtiger Pakete der Contrib-Bibliothek sowie technische Hintergründe und Einblicke in die Details der Implementation runden das Werk ab.

Nach dem Aufruf von Rich Hickey, die Entwicklung von Clojure durch Spenden zu unterstützen, kamen der Verlag und die Autoren überein, mit dem Buchverkauf dazu beizutragen. Aktueller Plan ist es, von jedem verkauften Buch 50 Cent an Rich Hickey und das Clojure-Projekt zu spenden.

Clojure – Einführung und Praxis
erscheint voraussichtlich im September 2010 im dpunkt.verlag,
ISBN 978-3-89864-684-0

Stefan Kamphausen ist gelernter Physiker und wie viele dieser Zunft in der Computerei in vielfältiger Form tätig. Die Vorliebe des seit mehr als 15 Jahren begeisterten Linux-Anwenders gilt den Sprachen der Lisp-Familie.

Das Entwickeln in vollen Zügen stellt besondere Anforderungen an die Fehlersuche. Dieses kleine Theaterstück stellt typische Situationen und Exceptions nach, die dabei auftreten können. Ähnlichkeiten mit realen Projekten sind rein zufällig, aber durchaus beabsichtigt.

Pleiten, Pech und PatternTesting – ein Drama in mehreren Akten

Oliver Böhm, agentes AG

Abbildung 1: Login-Maske

Prolog

Ich war mit einem meiner Arbeitskollegen, einem begeisterten Poker-Spieler, neulich im Zug Richtung Mailand unterwegs. Während wir im Abteil saßen, erzählte er von einem Projekt-Angebot, das er unmöglich ablehnen konnte und das ihm jetzt starke Kopfschmerzen bereitete. Es handelte sich dabei um eine hochsicherheitskritische Portal-Lösung, bei dem das Login-Modul noch nicht so richtig zur Zusammenarbeit bereit sei (siehe Abbildung 1).

Bei den meisten Benutzern würde zwar die Anmeldung funktionieren, aber ausgerechnet beim Haupt-Auftraggeber, Guiseppe, sträubte sich die Anwendung noch (siehe Abbildung 2).

Da mein Kollege etwas übernächtigt aussah, bot ich ihm meine Hilfe an. Es sollte ja nicht allzu schwer sein, anhand des Stacktraces die fehlerhafte Stelle zu finden (im Listing 1 in Fettschrift hervorgehoben).

```
protected void doPost(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
String password = request.getParameter(„password“);
String userName = request.getParameter(„user“);
User user = LoginService.login(userName, password);
request.setAttribute(„user_login“, user.getName());
RequestDispatcher dispatcher = request
       .getRequestDispatcher(„/overview.jsp“);
dispatcher.forward(request, response);
}
```

Listing 1

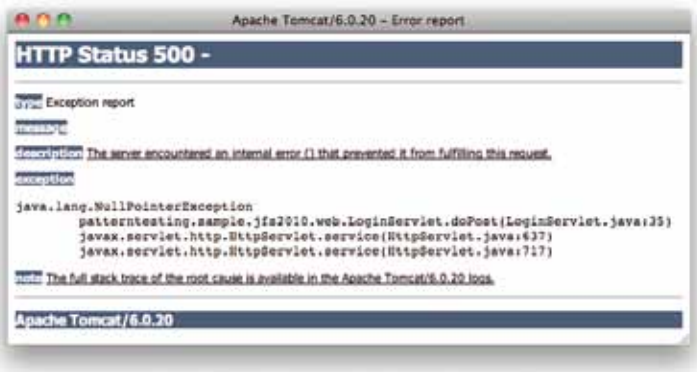


Abbildung 2: NullPointerException nach der Anmeldung

Auf den ersten Blick sah alles normal aus. „Lass doch mal die JUnit-Tests sehen!“, bat ich meinen Nachbarn.

„Welche JUnit-Tests?!?“

Erster Akt: Ein Königreich für einen Test

Um den Fehler einzugrenzen und um später dessen Abwesenheit zu garantieren, sind JUnit-Tests ein probates Mittel, alles andere als Zeitverschwendung. In diesem Fall kam allerdings erschwerend hinzu, dass für Servlets die Anfangshürde etwas höher liegt. Diese lässt sich aber mithilfe von Mock-Objekten (z.B. von Spring-Mock) nehmen (siehe Listing 2).

```
public void testDoPost() throws ServletException,IOException {
    LoginServlet servlet = new LoginServlet();
    MockHttpServletRequest req = new MockHttpServletRequest();
    MockHttpServletResponse resp =
        new MockHttpServletResponse();
    req.setParameter(„user“, „guiseppe“);
    req.setParameter(„password“, „topsecret“);
    servlet.doPost(req, resp);
    log.info(„Yeah, no exception!“);
}
```

Listing 2

Abbildung 3: Einbindung der PatternTesting-Bibliothek

Das ist zwar nur ein sehr, sehr einfacher Test ohne jegliche Prüfung, aber immerhin ließ sich damit der Fehler reproduzieren (siehe Listing 3).

```
java.lang.NullPointerException
at sample.jfs2010.web.LoginServ-
let.doPost(LoginServlet.java:35)
at sample.jfs2010.web.LoginServ-
letTest.testDoPost(LoginServletTest.
java:52)
```

Listing 3

Damit konnte die Fehlerursache der Anwendung ohne Hoch- und Runterfahren des Tomcats eingegrenzt und gedebuggt werden. Aus irgendeinem Grund ist das User-Objekt an dieser Stelle „null“ – aber warum?

Zweiter Akt: PatternTesting betritt die Bühne

Das Problem bei vielen NullPointerExceptions ist, dass die Ursache meistens ganz woanders liegt. Entweder wird unerlaubterweise „null“ als Parameter übergeben oder aber eine Methode liefert „null“ zurück. „Lass uns doch mal sehen, ob irgendeine Methode „null“ zurückgibt!“, riet ich meinem Kollegen.

„Und wie?“

„Na, wir nehmen die PatternTesting-Bibliothek; diese bietet die Möglichkeit, sämtliche Übergabe-Parameter und Rückgabewerte automatisch durch Asserts abzusichern. Hast du zufällig das AspectJ-Plugin eingebunden?“

AspectJ ist eine Aspekt-orientierte Erweiterung zu Java und PatternTesting eine der ersten AspectJ-Bibliotheken. Daher wird für Eclipse das AJDT – das AspectJ-Plugin – für den Einsatz dieser Bibliothek benötigt. Falls es noch nicht eingebunden ist, kann man es über http://www.eclipse.org/ajdt/ herunterladen beziehungsweise als Update-Site hinzufügen. Ich hatte aber Glück, mein Kollege arbeitete mit der Springsource Toolsuite (STS), die das AspectJ-Plugin bereits beinhaltet.

Nachdem das Projekt in ein AspectJ-Projekt umgewandelt wurde (Kontext-Menü, „Configure -> Convert to AspectJ Project“), konnte PatternTesting in den Aspect-Path aufgenommen werden (siehe Abbildung 3).

Damit die Asserts auch aktiv sind, musste in der Run Configuration der LoginServletTest-Klasse unter „Arguments“ der Schalter „-ea“ als VM-Argument noch angegeben werden. Ansonsten bleiben die eingestreuten Asserts still. Als wir damit den Testfall starteten, bekamen wir Folgendes zu sehen (siehe Listing 4).

Jetzt flogen wir bereits früher in der Login-Methode mit dem Hinweis heraus, dass hier ein „null“-Wert zurückgegeben wurde. Wir betrachteten diese Methode nun genauer (So würde man zwar keine Login-Methode schreiben, aber für eine erste Demo ist dies ausreichend, siehe Listing 5).

```
java.lang.AssertionError: User LoginService.login(String, String) re-
turns null!
...
at sample.jfs2010.LoginService.login(LoginService.java:62)
at sample.jfs2010.web.LoginServlet.doPost(LoginServlet.java:26)
...
```

Listing 4

```
public static User login(String name, String passwd)
        throws IOException {
    User user = null;
    URL url = getLoginURL(name);
    HttpURLConnection connection =
        (HttpURLConnection) url.openConnection();
    int rc = connection.getResponseCode();
    if (rc != 404) {
        user = new User();
        user.setName(name);
    }
    return user;
}
```

Listing 5

```
    ...
    if (rc == 404) {
        throw new LoginException(name +
„: login failed“);
    }
    return new User(name);
}
```

Listing 6

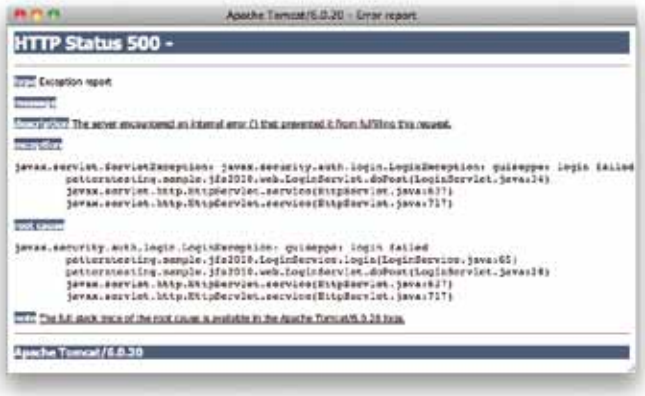


Abbildung 4: LoginException

```
pointcut nonVoidMethods() :
     execution(public Object+ *..*(..));
```

Listing 7

```
java.net.NoRouteToHostException: No route to host
       at java.net.PlainSocketImpl.socketConnect(Native
Method)
       ...
       at java.net.HttpURLConnection.getResponseCode(...)
       ...
```

Listing 8

```
java.net.NoRouteToHostException: No route to 216.34.181.96
at ...
```

Listing 9

Und da hatten wir schon das Problem: Wenn die Prüfung fehlgeschlagen ist (hier durch einen Response-Code von 404 realisiert), wird ein „null"-User zurückgegeben. Ganz schlechter Stil – da sollten wir doch lieber eine Exception verwenden (siehe Listing 6)

Damit funktionierte der Login zwar noch immer nicht, aber wir erhielten jetzt wenigstens eine aussagekräftigere Exception (siehe Abbildung 4).

Pause: Ein Blick hinter die Bühne

„Warum kam jetzt eigentlich der AssertionError am Ende der login-Methode?", fragte mich mein Kollege. „Wir hatten doch noch gar nicht die PatternTesting-Bibliothek aufgerufen?"

„Doch, haben wir!", entgegnete ich ihm. „Indem wir das Projekt in ein AspectJ-Projekt umgewandelt haben, wurde auch der Java-Compiler gegen den AspectJ-Compiler ausgetauscht. Dieser hat dann die Aspekte aus PatternTesting eingewebt."

Mein Gegenüber schaute mich etwas ungläubig an. Ich musste also etwas weiter ausholen.

„Auf der einen Seite hast du hier den LoginService mit der Login-Methode. Auf der anderen Seite stellt PatternTesting den Aspekt NullPointerTrap bereit, der am Ende jeder Methode ein „assert(Rückgabewert != null)" einfügt." Ich zeichnete eine kleine Skizze mit der Klasse und dem Aspekt auf den ausliegenden Reiseplan.

„Und wie kommt der Assert in die Login-Methode?", wollte mein Kollege wissen.

„Der AspectJ-Compiler nimmt den Code aus dem Aspekt und fügt ihn an den Code an der entsprechenden Stelle ein. Dies wird auch als „Weaving-Prozess" bezeichnet (siehe Abbildung 5). Anschließend greift er auf den Java-Compiler zurück, um daraus 100%igen Byte-Code zu generieren."

„Und woher weiß AspectJ, an welchen Stellen er den zusätzlichen Code einfügen muss?"

„Der zusätzliche Code wird übrigens „Advice" genannt", lehrte ich ihn. „Über sogenannte „Pointcuts" sage ich dem Compiler, wo er den zusätzlichen Code einfügen soll." (siehe Listing 7).

„Dies ist zum Beispiel ein Pointcut, der auf alle Klassen-Methoden zutrifft, die irgendetwas vom Objekt Abgeleitetes zurückliefern. Mithilfe dieses Pointcuts kann ich dann dem AspectJ-Compiler sagen, dass er am Ende dieser Methoden einen Assert einfügen soll."

Fragende Blicke. „Muss ich das alles wissen, um PatternTesting verwenden zu können?"

„Nein, aber es schadet auch nicht. Manchmal siehst du im Stacktrace komische Methodennamen wie „$advice...." – dann weißt du, dass das interne Methoden sind, die der AspectJ-Compiler hinzugefügt hat."

Dritter Akt: Kein Anschluss unter dieser Nummer

Nachdem wir die JUnit-Tests etwas ausgebaut hatten, brach uns auf der Fahrt über die Schwäbische Alb plötzlich die Netzverbindung ab (siehe Listing 8).

„Gegen welchen Server will er denn hier gehen?", fragte ich meinen Kollegen. „Das weiß ich jetzt auch nicht mehr so genau.

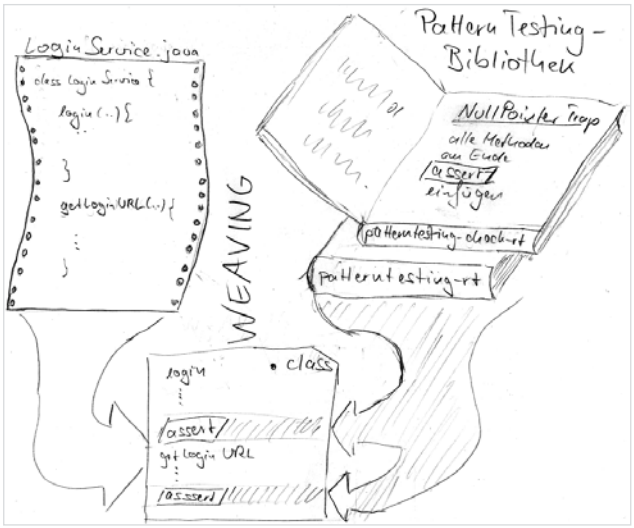


Abbildung 5: Der Weaving-Prozess

```
@RunWith(SmokeRunner.class)
public final class LoginServletTest {

    @IntegrationTest(„needs LoginService")
    public void testDoPost() throws IOException {
        ...
}
```

Listing 10

Warum, verdammt noch mal, kann mir das die Exception eigentlich nicht selber sagen?"

Ja, warum eigentlich nicht? Muss ich mich denn als Entwickler immer um alles kümmern? Die hier verwendete HttpURL-Connection hat doch alle Informationen, um hier wenigstens noch den betroffenen Rechner anzugeben. Aber Moment mal, die PatternTesting-Bibliothek hat doch noch eine Komponente für Exceptions...

„Lass uns noch patterntesting-exception.jar einbinden und sehen, was passiert", bat ich. Gesagt, getan. Wir fügten die Bibliothek ebenfalls über den Aspect Path ein und erhielten jetzt die Exception angezeigt (siehe Listing 9). Ich konnte damit zwar immer noch nichts anfangen, aber meinem Gegenüber dämmerte es, wo hier diese IP-Adresse herkam.

Vierter Akt: Testen in vollen Zügen

In Ulm standen wir etwas länger als geplant auf dem Bahnsteig, so dass wir wieder eine relativ stabile Netzverbindung über UMTS hinbekamen und unsere Tests fortführen konnten. Dann wurde es plötzlich eng, weil wohl wieder die eine Hälfte des Zuges ausgefallen war und die Mitreisenden auf unsere Zughälfte verteilt wurden. Als sich dann der Zug endlich in Bewegung setzte, war es mit unserer Netzverbindung wieder vorbei. „Schade – wir waren gerade so schön am Testen. Jetzt schlagen leider alle unsere Tests fehl, die ein Netz brauchen."

„Eigentlich sind es doch schon fast Integrationstests, da wir hier das Zusammenspiel der externen Schnittstellen testen, oder?", erwiderte ich.

„Worauf willst du hinaus?", fragte mein Kollege.

„Sag ich gleich – lass uns doch mal alle Integrationstests als solche kennzeichnen."

PatternTesting stellt für diesen Zweck „@IntegrationTest" als Annotation bereit. Diese kann man vor einzelnen Test-Methoden, aber auch vor einer ganzen Test-Klasse schreiben.

„Und jetzt?", fragte mein Freund, als wir alle Schnittstellen-Tests so gekennzeichnet hatten.

„Jetzt nehmen wir den SmokeRunner aus PatternTesting und starten nochmals alle Tests."

JUnit4 bietet die Möglichkeit, einen eigenen JUnit-Runner über @RunWith anzugeben. Unsere Test-Klasse sah jetzt so aus (siehe Listing 10).

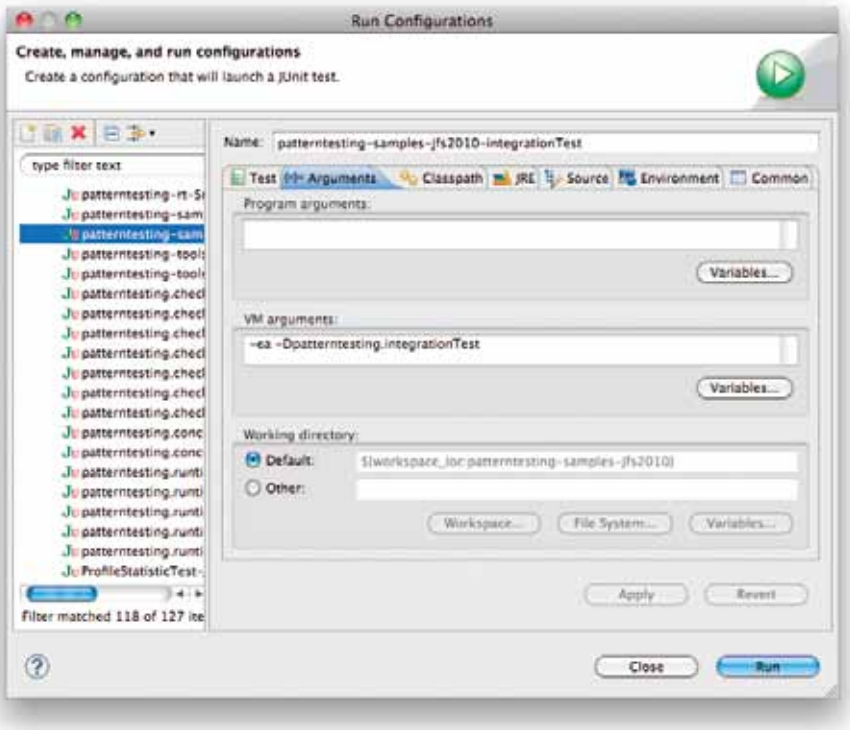


Abbildung 6: Eigene Launch-Konfiguration für Integrations-Tests

„Aber wir haben doch JUnit3-Tests!", warf mein Kollege ein, als ich ihm die Möglichkeiten von JUnit 4 erklärte.

„Macht nichts, das klappt trotzdem", tröstete ich ihn.

Und tatsächlich, als wir die kompletten Tests wieder starteten, war wieder alles im grünen Bereich.

„Toll, aber was ist, wenn ich jetzt sämtliche Tests starten will?".

„Dann setzt du einfach die System-Property „patterntesting.integrationTest", dann werden auch die Integrations-Tests durchlaufen. Am besten, du machst dir dafür eine zusätzliche JUnit-Launchkonfiguration (siehe Abbildung 6)."

„Warum heißt denn der Runner eigentlich SmokeRunner?", wollte noch mein Kollege wissen.

„Eigentlich war der Runner für die @SmokeTest-Annotation vorgesehen. Damit kannst du die JUnit-Tests markieren, die für einen ersten, schnellen Test ausreichend sind. Dann kamen aber noch weitere Annotationen wie eben @IntegrationTest, aber auch @SkipTestOn, @RunTestOn oder @Broken hinzu, mit denen Tests an Bedingungen geknüpft oder als kaputt gekennzeichnet werden können."

Schluss-Akt: Und wenn er nicht gestorben ist ...

Langsam zogen bereits die ersten Vororte von München am Fenster vorbei.

„Das ist ja alles schön und recht", meinte mein Gesprächspartner, „aber weißt du, mein Auftraggeber ist da etwas kritisch, wenn es um neue Technologien wie AOP und so geht. Wenn der mitbekommt, dass ich hier den AspectJ-Compiler verwende, bin ich gleich zwei Köpfe kürzer."

„Ach", beruhigte ich ihn, „das wird er gar nicht merken. Intern greift der AspectJ-Compiler auf den Java-Compiler zurück, das heißt, du kannst mit Fug und Recht behaupten, dass mit dem Java-Compiler übersetzt wurde. Und die zusätzlichen Überprüfungen sind sowieso nur aktiv, wenn der Schalter „-ea" gesetzt ist."

„Na ich weiß nicht – und wenn es doch herauskommt?"

„Dann fahr doch zweigleisig – verwende zum Testen und innerhalb Eclipse den AspectJ-Compiler und die PatternTesting-Bibliothek im Aspect Path, und zum Bauen

verwendest du Java und patterntesting-rt.jar als ganz normale Java-Bibliothek."

„Warum patterntesting-rt.jar?"

„Da sind Annotationen wie @IntegrationTest enthalten, über die du die Aspekte in der PatternTesting-Bibliothek steuern kannst. Die haben damit zwar nur noch Dokumentations-Charakter, aber du kannst sie im Code belassen."

Wir näherten uns langsam dem Münchner Hauptbahnhof.

„Nein, nein, nein, das ist mir alles zu riskant. Du kennst meinen Auftraggeber nicht. Kann ich die PatternTesting-Bibliotheken nicht einfach wie eine ganz normale Java-Bibliothek verwenden?"

„Das sagte ich doch bereits, ja, das geht. Du verlierst zwar vieles an Funktionalität, die über die Aspekte hereinkommen, aber du hast den SmokeRunner, der neben @IntegrationTest auch noch andere Annotationen versteht. Für Classpath-Probleme gibt es den ClasspathMonitor, der sich auch über JMX abfragen lässt. Und falls du mal alle vier Prozessor-Kerne deines Notebooks zum Glühen bringen willst, gibt es noch einen ParallelRunner in patterntesting-concurrent.jar, der eine Erweiterung des SmokeRunners darstellt."

Wir waren bereits am Hauptbahnhof an-gekommen und ich musste hier aussteigen.

„Und wo bekomme ich weitere Informationen zur PatternTesting-Bibliothek?", fragte er mich zum Abschied.

„Findest du alles unter patterntesting.org."

Wir verabschiedeten uns. Ich sah noch im Vorbeilaufen durchs Fenster, wie er sich wieder an sein schickes Notebook setzte. Er sah etwas mitgenommen aus. Ob das am gut gefüllten ICE lag? Ich weiß es nicht. Ich hoffe, ihn spätestens zum Java Forum Stuttgart im Juli wiederzusehen.

Links

- PatternTesting: http://patterntesting.org/
- AspectJ-Homepage: http://eclipse.org/aspectj/
- Beispiel-Code: http://patterntesting.cvs.sourceforge.net/viewvc/patterntesting/PatternTesting10/patterntesting-samples/jfs2010/

Kontakt:
Oliver Böhm
oliver.boehm@agentes.de

Oliver Böhm beschäftigt sich mit Java-Entwicklung unter Linux und Aspekt-Orientierte SW-Entwicklung. Neben seiner hauptberuflichen Tätigkeit als J2EE-Entwickler und -Coach bei agentes ist er Buchautor, gibt AOSD-Vorlesungen und ist Board-Mitglied der JUGS (Java User Group Stuttgart)

iJUG Verbund

Mitglieder stellen sich vor

Java User Group Köln

Die Java User Group Köln (JUGC) ist eine im weltweiten JUG-Verbund intensiv verknüpfte Gruppe von Java-Interessierten der gesamten Rheinschiene (Aachen, Bonn, Ruhrgebiet, insbesondere Köln). Neben dem Wissens- und Erfahrungsaustausch stehen das Kennenlernen und der Spaß im Mittelpunkt.

In der unabhängigen Gruppe tauschen sich seit 2001 sowohl Anfänger als auch gestandene Java-Profis aus. Die JUGC bietet (etwa) monatliche, hochqualifizierte Vorträge mit häufig international bekannten Vortragenden kostenlos an. Es besteht die Möglichkeit, vor jedem Vortrag fünfminütige Kurzvorträge, sogenannte Lightning Talks zu halten. Komplementär zu den Vortragsabenden gibt es einen offenen Stammtisch. Dieser findet immer jeden letzten Freitag im Monat statt.

Bei den Vortragsabenden wird auf ein ausgewogenes Verhältnis von Themen sowie regionalen, nationalen und internationalen Vortragenden geachtet. Ein Highlight der JUGC ist sicherlich, dass es regelmäßig gelingt, sehr einzigartige Events auszurichten. Zu den vergangenen Höhepunkten gehören unter anderem das erste IDE Shootout, mit offiziellen Vertretern der vier führenden IDEs, oder ein interaktiver, fünfstündiger Talk mit Dr. Neal Gafter, seinem ersten Talk in Deutschland überhaupt. Ferner werden sehr aktuelle Themen aufgegriffen, die nicht nur von Nutzern, sondern von den Urhebern selbst nahegebracht werden. Zwei Beispiele von vielen sind Vorträge über JavaFX oder Scala, als diese Themen noch kaum allgemeine Beachtung fanden.

Weitere Informationen: http://www.jugcologne.eu

Entwickelte Java-Anwendungen müssen auch betrieben werden. Jede Java Virtual Machine (JVM) stellt dafür eine Reihe von Werkzeugen zur Verfügung. Der Artikel stellt die Administrationswerkzeuge der Sun JVM vor, die für den Einsatz eines Java-Servers notwendig sind.

Der Werkzeugkasten des Admins für den Java-Server

Tobias Frech, Frech.IT

Wenn die Java-Anwendung auf dem Server immer wieder kleine „Denkpausen" einlegt oder den Betrieb sporadisch mit einem OutOfMemory-Error quittiert, fragt sich der Admin: „Woran liegt das?" Mithilfe der Standardwerkzeuge des Betriebssystems wird er versuchen, den Java-Server-Prozess zu untersuchen und die Ursache des Problems zu ermitteln. Die Sun JVM hält zusätzliche Werkzeuge zur Diagnose bereit, mit deren Hilfe sich in den laufenden Java-Prozess hineinschauen lässt. Diese Werkzeuge werden nachfolgend anhand von typischen Administrationsproblemen vorgestellt.

Die meisten Informationen bieten sicherlich die beiden grafischen Werkzeuge JConsole und VisualVM. Diese sind jedoch nicht auf jedem Server einsetzbar, insbesondere, wenn er „remote" administriert wird. Sie setzen zudem eine gewisse Kenntnis der Materie voraus. Wir wenden uns daher in diesem Artikel den Grundlagen und den einfachen Kommandozeilen-basierten Werkzeugen zu. Dazu muss ein Sun Java Development Kit (JDK) installiert sein; das Java Runtime Environment (JRE) bringt diese Werkzeuge nicht mit. Im Verzeichnis, in dem sich die JVM „java" befindet, sind die Kommandozeilen-Werkzeuge jps, jstat/jstatd, jinfo, jmap und jstack abgelegt (siehe http://java.sun.com/javase/6/docs/technotes/tools/index.html#monitor). Zum JDK sind sie seit der Version 5 hinzugekommen. Insbesondere unter Windows sind diese Werkzeuge aber nur eingeschränkt oder teilweise vorhanden. Abbildung 1 zeigt die genaue Übersicht. Die folgenden Beispiele sollen dazu animieren, selbst auszuprobieren und für die eigene Situation nützliche Werkzeuge zu entdecken.

Werkzeug	JDK 5	JDK 6
jps	ok	ok
jinfo	kein Windows/Linux Itanium	Windows eingeschränkt
jstat	ok	ok
jstack	kein Windows/Linux Itanium	Windows eingeschränkt
jmap	kein Windows/Linux Itanium	Windows eingeschränkt

Abbildung 1: Die JDK-Werkzeuge unterscheiden sich je nach Betriebssystem in der Verfügbarkeit und dem Funktionsumfang

jps: Anwendungen finden

Jedes der Werkzeuge erwartet als einen Parameter die sogenannte „lvmid", die üblicherweise die Prozess-ID des Java-Prozesses darstellt. Diese lässt sich über jps ermitteln (Merkhilfe für Unix-Admins: „java ps"). Es werden nur die Java-Prozesse aufgelistet, die für den aktuellen Benutzer gemäß den Systemrechten erreichbar sind. In der Liste wird auch jps mit ausgegeben, da die Kommandozeilen-Werkzeuge selbst in Java realisiert sind. Mit der Option „-v" lassen sich alle Parameter ausgeben, die der JVM übergeben wurden (etwa für die Speicher- und GC-Einstellungen). Mit der Option „-l" wird der vollständige Name der Main-Klasse (zum Beispiel org.jboss.Main für einen JBoss-Server) ausgegeben und die Option „-m" zeigt alle Parameter, die an die Main-Klasse als Parameter weitergereicht wurden (beispielsweise die Server-Configuration bei einem JBoss-Server). Abbildung 2 zeigt eine solche Ausgabe.

```
[tobi@fedora ~]$ jps
13257 Jps
13199 Main

[tobi@fedora ~]$ jps -l -m
13199 org.jboss.Main -c minimal
13244 sun.tools.jps.Jps -l -m

[tobi@fedora ~]$ jps -l -v
13199 org.jboss.Main -Dprogram.name=run.sh -Xms128m -Xmx150m -Dsun.
rmi.dgc.client.gcInterval=3600000 -Dsun.rmi.dgc.server.gcInter-
val=3600000 -Xloggc:gc.log -Djava.net.preferIPv4Stack=true -Djava.
endorsed.dirs=/home/tobi/JBoss/jboss-4.2.3.GA/lib/endorsed
13232 sun.tools.jps.Jps -Dapplication.home=/usr/java/jdk1.6.0_16
-Xms8m
```

Abbildung 2: jps listet alle laufenden Java-Prozesse auf, hier ein JBoss-Server

jinfo: JVM-Konfiguration auslesen

Ähnlich wie die Environment-Variablen eines Unix-Prozesses hat auch jede JVM ihr eigenes internes „Environment": die

System Properties. Um Klarheit über die eigentliche Konfiguration einer JVM zu bekommen, sind nicht nur die Parameter beim JVM-Start erheblich (etwa „-Xmx5g" für den maximalen Heap-Speicher). Auch die Inhalte eben dieser System Properties sind relevant, die entweder beim JVM-Start mit der „-D"-Option gesetzt werden können (beispielsweise „-Djava.awt.headless=true" für Server mit Grafik-Verarbeitung), programmatisch durch die JVM oder die durch Java-Anwendung selbst. Mit jinfo lassen sich alle Schlüssel-Wert-Paare der System Properties anzeigen. Dies kann dann nützlich werden, wenn beispielsweise unklar ist, mit welcher Timezone oder Locale die jeweilige JVM arbeitet. Abbildung 3 zeigt ein Beispiel dazu.

Mit der Option „-flag" kann sogar ein Kommandozeilen-Parameter nachträglich gesetzt werden, um zum Beispiel automatische Heap Dumps bei Speicherproblemen zu aktivieren:

```
jinfo -flag +HeapDumpOnOutOfMe-
moryError <lvmid>
jinfo -flag HeapDumpPath=/tmp/
<lvmid>
```

Nicht jede Einstellung der JVM lässt sich jedoch nachträglich, das heißt nach dem Start, verändern.

```
[tobi@fedora ~]$ jinfo 14042 | grep user.
Attaching to process ID 14042, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 1.5.0_17-b04
user.name = tobi
user.language = de
user.timezone = Europe/Berlin
user.country = DE
user.home = /home/tobi
user.dir = /home/tobi/JBoss/jboss-4.2.3.GA/bin
```

Abbildung 3: jinfo gibt Auskunft über die Inhalte der System Properties der JVM

```
[tobi@fedora ~]$ jstat -gc 14242
S0C      S1C      S0U      S1U      EC       EU
9856,0   7296,0   0,0      7288,4   90496,0  32828,6

OC       OU       PC       PU
28480,0  20606,8  32896,0 32826,9

YGC      YGCT     FGC      FGCT     GCT
13       0,461    3        0,381    0,842
```

Abbildung 4: Ausgabe eines Aufrufs mit der lvmid

jstat: Garbage Collection und andere Laufzeit-Informationen

Eines der wertvollsten Werkzeuge im Werkzeugkasten des Java-Administrators ist jstat. Mit diesem Werkzeug lassen sich aktuelle Informationen über die Anzahl der geladenen Klassen, die Aktivitäten des Just-in-Time-Compilers, die Garbage-Collection-Aktivitäten und die Speicherverwaltung ausgeben. In der JDK-Dokumentation sind alle zwölf Optionen zu finden, welche die jeweils auszugebenden Informationen steuern. Ebenfalls findet man dort die Bedeutung der verschiedenen Spaltenüberschriften in der Ausgabe. Hier soll nun näher auf die Ausgabe der Option „-gc" eingegangen werden. Abbildung 4 zeigt die Ausgabe eines Aufrufs mit der lvmid zusammen ohne weitere Optionen.

Zur besseren Übersichtlichkeit wurden die Spaltenüberschriften zusammen mit dem zugeordneten Inhalt umbrochen. Die Überschriften der Spalten haben folgende Bedeutungen:

- S0C/S1C: Survior Space 0 bzw. 1 Capacity (also aktuelle Größe)
- S0U/S1U: Survior Space 0 bzw. 1 Utilization (also Füllstand des Bereichs)
- EC und EU: Eden Capacity und Utilization
- OC und OU: Old Generation Capacity und Utilization
- PC und PU: Permanent Generation Capacity und Utilization
- YGC und YGCT: Anzahl und kumulative Zeit der Minor (Young) Garbage Collection
- FGC und FGCT: Anzahl und kumulative Zeit der Full Garbage Collection
- GCT: kumulative Zeit der Garbage Collection gesamt (Minor + Full GC)

Eine genaue Beschreibung der Funktionsweise der Garbage Collection (GC) findet sich in einem White Paper von Sun unter http://java.sun.com/j2se/reference/whitepapers/memorymanagement_whitepaper.pdf. Es lassen sich jedoch auch leicht einige grundlegende Zusammenhänge an den obigen Werten beobachten. Die beiden Survivor Spaces zusammen mit dem Eden-Bereich bilden die sogenannte „Young Generation". Neue Objekte werden im Eden-Bereich abgelegt. Eine Minor GC (Zähler: YGC) räumt diese Bereiche auf und sollte zu einem leeren Eden-Bereich führen. Gleichzeitig können langlebigere Objekte in die Old Generation verschoben werden (OC und OU erhöhen sich). Ist die Old Generation voll (OU erreicht 100), so wird spätestens dann eine Full GC (Zähler: FGC) ausgelöst. Dass sowohl die Young als auch die Old Generation immer wieder bis zu 100 Prozent gefüllt werden, ist in einer JVM vollkommen normal. Wichtig ist, dass nach der jeweilig zugeordneten GC wieder Speicher freigegeben werden konnte, das heißt, der jeweilige Utilization-Wert ist dann Null oder sinkt merklich. Es lassen sich nun die drei folgenden Effekte beobachten:

1. Wenn die Anwendung immer wieder eine merkliche Pause macht, so könnte dies an einer „Full GC" liegen. Während einer GC (Minor und Full) wird in der Grundeinstellung die Anwendung pausiert, bis die GC abgeschlossen ist. Minor GCs sind im Allgemeinen sehr schnell durchgeführt. Full GCs können jedoch je nach Speichergröße von mehreren Gigabyte und abhängig von der Speichergeschwindigkeit und Prozessoranzahl durchaus auch mehrere Sekunden bis zu Minuten in Anspruch nehmen. Solche Pausen wären in deut-

lichen Sprüngen in der kumulativen Zeit der Full GC sichtbar (Zähler: FGCT in Sekunden). Abhilfe kann hier schaffen, eine GC-Strategie zu aktivieren, die während der Full GC mehrere Prozessoren nutzt oder die parallel zur Anwendungsausführung angewendet wird. Beide Wege sind nicht unproblematisch und müssen sorgfältig geplant, konfiguriert und getestet werden.

2. Die Anwendung wird mit der Zeit merklich langsamer, bis dann die aktuelle Verarbeitung mit einem OutOfMemory-Error abgebrochen wird. In dieser Situation wird der Speicher der JVM zu knapp, nach einer Full GC kann kaum noch Speicher freigegeben werden und die nächste Full GC folgt in relativ kurzem Zeitabstand. Bei normalem Betrieb sollte die Minor GC deutlich häufiger auftreten als die Full GC, das heißt, YGC hat einen deutlich höheren Wert als FGC. Tritt die oben beschriebene Situation ein, so erhöht sich fast nur noch der FGC-Zähler und letztendlich nimmt die kumulative Zeit der GC (Zähler: GCT) genau so schnell zu, wie die reale Zeit verstreicht. Die JVM beschäftigt sich dann fast ausschließlich mit der GC. Nach einiger Zeit führt dies zum OutOfMemory-Error. Hier lässt sich ein gutes Monitoring ansetzen, das dieses beschriebene Muster erkennt und frühzeitig warnt, bevor der Speicher endgültig zu voll ist.
3. Sowohl die Young als auch die Old Generation werden durch die GC immer wieder frei geräumt. Die Anwendung steigt jedoch trotzdem mit einem OutOfMemory-Error aus. Hier lohnt ein Blick auf die Utilization der Permanent Generation (Zähler: PU). Steigt dieser Wert selbst nach einer Full GC immer weiter an, so ist die Permanent Generation eventuell zu klein gewählt. Tückischerweise wird dieser Speicherbereich nicht mit dem Parameter „-Xmx1500m“ gesteuert, sondern mit „-XX:MaxPermSize=512m“.

Damit man zur Beobachtung der JVM das jstat-Kommando nicht immer wieder absetzen muss, gibt es die Möglichkeit, die Werte zyklisch wiederholt ausgeben zu lassen. Dazu gibt man als zusätzlichen Parameter die Zeit in Millisekunden oder in Sekunden mit der Endung „s“ an (siehe Abbildung 5).

```
[tobi@fedora ~]$ jstat -gcutil 14343 2s
  S0     S1     E      O      P     YGC     YGCT    FGC    FGCT     GCT
  0,00  39,14  38,86  63,89  72,26     33    0,628     7    1,161    1,789
 60,22   0,00  48,74  63,89  72,27     34    0,648     7    1,161    1,809
 46,10   0,00  39,14  65,21  72,29     36    0,691     7    1,161    1,852
 50,88   0,00  15,55  65,68  72,30     38    0,717     7    1,161    1,878
  0,00  83,74  83,53  65,84  72,32     39    0,727     7    1,161    1,888
  0,00  98,73  85,56  66,17  72,32     41    0,750     7    1,161    1,911
```

Abbildung 5: Zyklische Ausgabe

jstack: Was macht die Anwendung eigentlich?

Wenn unklar ist, mit welchen Aufgaben sich die Anwendung momentan beschäftigt, so kann auf verschiedenen Wegen ein sogenannter „Thread-Dump“ für die gesamte JVM erzeugt werden. Dabei wird für jeden einzelnen Thread der momentane Stand des Call-Stacks ausgegeben. Abbildung 6 zeigt einen kurzen Beispielausschnitt. Jede einzelne Zeile in diesem Stack beginnt mit dem Objektnamen und der konkreten Methode innerhalb des Objekts, die gerade abgearbeitet wird. In Klammern wird dann noch der Name der Quellcodedatei mit Angabe der Zeilennummer in dieser Datei ausgegeben. Der Administrator kann aus den reinen Objekt- und Methodennamen natürlich nicht schließen, was genau an dieser Codestelle passiert. Jedoch kann der Anfang des Objektnamens – das sogenannte „Package“ – sehr wohl deutliche Hinweise darauf geben, in welcher Schicht der Anwendung der Programmteil angesiedelt ist. So ist zum Beispiel leicht erkennbar, wenn der entsprechende Code aus dem JDBC-Datenbanktreiber stammt (für Oracle beginnt der Package-Name beispielsweise mit oracle.jdbc). Beim Lesen des Call-Stacks ist darauf zu achten, dass die unterste Zeile der eigentliche Start ist und jede Zeile darüber einen Methodenaufruf innerhalb der Bearbeitung darstellt, das heißt, dass die oberste Zeile eines Call-Stacks die aktuelle Position der Programm-Abarbeitung für diesen Thread darstellt. Mit dem Thread-Dump für alle Threads lässt sich schnell ein grober Überblick darüber verschaffen, mit welchen Funktionen die verschiedenen Threads im Server beschäftigt sind.

Um nun einen solchen Thread-Dump zu erhalten, kommt das Werkzeug jstack zum Einsatz. Nach Aufruf mit der entsprechenden lvmid wird der Thread-Dump erzeugt und ausgegeben. Der Server läuft nach dieser Ausgabe üblicherweise normal weiter.

Wird diese Ausgabe wiederholt angefordert, so kann man sich die statistische Eigenschaft zunutze machen, dass die Codestellen, die am meisten Zeit „verbrauchen“, am häufigsten in einem Thread-Dump auftauchen sollten. jstack ist sozusagen der Profiler des Admins, der nicht zu stark in eine Produktiv-Umgebung eingreifen will oder darf und der sich den Gang zur Beschaffungsabteilung ersparen will. Das Verfahren, sich wiederholt Thread-Dumps von der JVM zu holen, wird übrigens vom Sampler-Plugin des VisualVM-Werkzeugs eingesetzt, um eine gewohnte Ausgabe eines Profilers zu erzeugen. Dies funktioniert sogar mit Produktivsystemen unter Last erstaunlich gut.

jmap: Speicherverschwendung aufspüren

Gerade wenn der Speicher knapp geworden ist beziehungsweise schon nicht mehr ausgereicht hat, stellt sich die Frage, wodurch dieser verbraucht wird. Eine Analyse auf einem Entwicklungs- oder Testsystem kann nur in manchen Fällen den Verursacher ermitteln, der auch im produktiven Betrieb den Speicherverbrauch nach oben treibt.

Bei vielen Anwendungen im laufenden Betrieb ist in einer solchen Situation für eine Diagnose des Speichers eine Unterbrechung, die nur wenige Sekunden dauert, noch vertretbar. Dann kann das Werkzeug jmap mit der Option „-histo“ eingesetzt werden, um ein Histogramm zu erzeugen, welches pro geladener Klasse die Anzahl der Objektinstanzen anzeigt (siehe Abbildung 7).

Neben der Anzahl der Instanzen wird auch noch die Speichergröße ausgegeben,

```
„pool-1-thread-2“ prio=10 tid=0x2b670400 nid=0x264c runnable [0x2b3ad000]
   java.lang.Thread.State: RUNNABLE
        at java.lang.String.intern(Native Method)
        at org.apache.lucene.document.Field.<init>(Field.java:285)
        at org.apache.lucene.index.FieldsReader.addField(FieldsReader.java:357)
        at org.apache.lucene.index.FieldsReader.doc(FieldsReader.java:197)
        at org.apache.lucene.index.SegmentReader.document(SegmentReader.java:733)
...
        at org.apache.lucene.search.Searcher.search(Searcher.java:105)
        at info.frech.fema.SearchService.searchInternal(SearchService.java:77)
        at info.frech.fema.SearchService.search(SearchService.java:42)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
...
        at com.sun.net.httpserver.Filter$Chain.doFilter(Filter.java:65)
        at sun.net.httpserver.ServerImpl$Exchange.run(ServerImpl.java:527)
        at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.java:886)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:908)
        at java.lang.Thread.run(Thread.java:619)
```

Abbildung 6: Ausschnitte aus einem Call-Stack für einen Thread

die durch die Instanzen selbst belegt wird. Hierbei ist es nicht ungewöhnlich, dass Arrays (Prefix [) von Bytes ([B), Integern ([I) oder Chars ([C) den meisten Speicher verbrauchen. Allerdings sind eigentlich die Klassen von Interesse, welche diese Arrays im Speicher referenzieren und damit im Speicher halten. Ein solches Histogramm kann also nur mit entsprechender Kenntnis, wie eine Software intern auf Klassenebene funktioniert, richtig interpretiert und dadurch Auffälligkeiten festgestellt werden. Hilfreich kann es hierbei auch sein, wenn über die Zeit immer wieder Histogramme erstellt werden und sich so Trends erkennen lassen. Wenn die Unterbrechungen noch ein wenig länger sein dürfen, so ist die Option „-histo:live“ zu bevorzugen, die vor der Histogramm-Erstellung noch eine Full Garbage Collection ausführt und damit nur Klassen anzeigt, die in diesem Moment noch erreichbar und damit noch aktiv sind.

```
[tobi@fedora ~]$ jmap -histo 3649

 num     #instances         #bytes  class name
----------------------------------------------
   1:        584317      612516808  [B
   2:          3618       23825536  [I
   3:        400239       17117128  [C
   4:         29569       14920848  [Ljava.lang.Object;
   5:        397347        9536328  java.lang.String
   6:        168406        4041744  java.util.LinkedList$Entry
   7:        111696        2680704  info.frech.fema.Feature
   8:         24033        2444688  <constMethodKlass>
   9:         24033        1926520  <methodKlass>
  10:         34899        1596104  <symbolKlass>
  11:          2269        1138096  <constantPoolKlass>
  12:          2269         897264  <instanceKlassKlass>
  13:         27924         893568  info.frech.fema.History$Entry
  14:         26219         839008  org.apache.lucene.index.TermInfo
  15:          2060         731328  <constantPoolCacheKlass>
  16:         28364         680736  java.util.ArrayList
  17:         28313         679512  java.util.LinkedList
  18:         27924         670176  java.util.Date
  ...           ...            ...  ...
```

Abbildung 7: Ergebnis der Option „-histo“

Soll die Analyse des Speicherinhalts ausführlich ausfallen, so kann mit jmap auch ein Speicherabbild (Dump) erzeugt werden:

```
jmap
-dump:live,format=b,file=heap.
hprof 5197
```

Je nach Größe des momentan verwendeten Speichers dauert diese Aktion im Bereich von Sekunden bis zu Minuten. Die Anwendung wird für diese Dauer angehalten. In einer produktiven Umgebung ist also die Erzeugung eines Speicherabbilds aufgrund der Unterbrechung nur eingeschränkt möglich. Wurde jedoch ein solches Abbild erzeugt, so kann mit Kenntnis der Funktionsweise der Software und einem entsprechenden Werkzeug der Speicherverbrauch analysiert werden. Empfehlenswert ist hierfür der Eclipse Memory Analyzer (MAT), der aber nicht mehr Teil der JDK-Tools ist.

Tobias Frech hilft Unternehmen als Trainer und Coach ihre Java-Anwendungen besser zu betreiben. Er ist SENS-Experte und leitet (mit anderen) die SIG JBoss bei der JUGS.

Kontakt:
Tobias Frech
tobias@frech.info

Das Java Framework JUnit ist sicher jedem Entwickler ein Begriff. Den Wenigsten dürfte aber bekannt sein, dass es Erweiterungen gibt, die seine Verwendung wesentlich bequemer machen. Ein Blick hinter die Kulissen von JUnit, Hamcrest und Make-It-Easy zeigt, wie man diese verwendet, um Unit-Tests verständlich und wartbar zu gestalten.

JUnit – Tests lesbar gestalten

Dirk Dittert, Method Park Software AG

Freitag 15 Uhr: Nur noch wenige Stunden bis zur Deadline für das Release in Ihrem aktuellen Projekt. Stellen Sie sich vor, Sie müssen jetzt überraschend noch eine größere Änderung an einer zentralen Komponente vornehmen. Haben Sie dabei ein gutes Gefühl? Können Sie sicher sein, dass die Anwendung auch nach dieser Änderung noch problemlos funktioniert?

JUnit in der Realität

In der Praxis sind bei vielen Projekten Unit-Tests höchstens in rudimentärer Form vorhanden. Oft wurden diese schon vor Jahren in JUnit 3 programmiert und seitdem kaum gepflegt. Die Fehlermeldungen sind entsprechend kryptisch und im Eifer des Gefechts oft keine Hilfe, um zu verstehen, warum eben jener Test fehlgeschlagen ist. Dies muss nicht immer so sein. Durch die hier vorgestellte Vorgehensweise lässt sich die Qualität der Tests deutlich steigern. Dazu verdeutlichen wir uns erst einmal die wesentlichen Unterschiede zwischen Test- und Produktcode.

Generell lässt sich sagen: Der Produktcode beschreibt eine konkrete Vorgehensweise und betrachtet Werte nur durch abstrakte Variablen. Kapselung ist ein wichtiges Thema, so dass Abhängigkeiten zwischen den einzelnen Modulen hier einen zentralen Stellenwert haben. Der Testcode beschreibt hingegen, was der Produktcode tut. Er enthält eine Reihe konkreter Werte, die die abstrakte Funktionsweise des getesteten Codes messbar machen. Modularität und Abhängigkeiten sind nur am Rande von Bedeutung, weil sich der Testcode immer nur auf einzelne, abgegrenzte Aspekte des Produktcodes bezieht.

Wichtig ist es, immer den großen zeitlichen Versatz zwischen der Entwicklung eines Tests und dem Auftreten von Fehlern im Hinterkopf zu behalten: Der Entwickler der ersten Fassung der Unit-Tests hat meist auch die zugehörige Funktionalität entwickelt. Dadurch sind ihm die Feinheiten der Spezifikation und ihrer Implementierung bis ins Detail bewusst. Anderen Entwicklern, die später durch ihre Änderungen Fehler hervorrufen, fehlt dieser Bezug jedoch meist vollständig. Die wichtigste Eigenschaft eines Unit-Tests ist daher, dass er intuitiv verständlich ist und wertvolle Hinweise auf mögliche Fehlerursachen liefert. Hand aufs Herz: Könnten Sie aus der in Abbildung 1 gezeigten Fehlermeldung Rückschlüsse auf die Ursache des Fehlers ziehen?

Struktur eines JUnit-Tests

Ein einheitlicher Bauplan erlaubt es allen Entwicklern, sich schnell auch in fremden Unit-Tests zurechtzufinden und so die eben erwähnten Gegensätze zu überbrücken. Dabei hat sich folgendes Schema in der Praxis bewährt:

- Vorbereiten: Der erste Schritt bereitet das Umfeld des Tests vor und stellt die Ausgangssituation her. Hier werden alle benötigten Objekte in den Ausgangszustand versetzt und die Verbindungen zwischen den beteiligten Objekten hergestellt.
- Durchführen: Aufruf des zu testenden Codes
- Überprüfen: Vergleicht die Resultate des getesteten Codes mit den erwarteten Ergebnissen
- Aufräumen: Verhindern, dass der Unit-Test negative Auswirkungen auf andere Testfälle hat.

Typische Stilprobleme in Unit-Tests

- Die für Tests, Hilfsmethoden und Variablen gewählten Namen sind wenig aussagekräftig oder gar irreführend
- Testfälle versuchen, mehrere unterschiedliche Szenarien abzudecken
- Die Intention eines Testfalls wird durch Code zur Initialisierung der beteiligten Objekte und Code zur Fehlerbehandlung verschleiert
- Testfälle können nicht quer gelesen werden, weil sie zu unterschiedlich strukturiert sind
- Die Tests enthalten magische Werte, ohne ihre Bedeutung zu erläutern

```
@Test
public void shippingAndBillingAddressMustBeFromSameCountry() {
    final Order order = make(an(_Order_));
    final Address billingAddress =
        make(an(_Address_, with(country, GERMANY)));
    final Address shippingAddress =
        make(an(_Address_, with(country, ARGENTINIA)));

    // stellt sicher, dass beide Adressen aus dem gleichen Land
stammen
    order.setAddresses(billingAddress, shippingAddress);

    assertThat("shipping and billing addresses from same country",
                billingAddress, sameCountryAs(shippingAddress));
 }
```

Listing 1

Das Framework JUnit [1] erkennt Methoden als Testfälle, die mit „@Test" annotiert sind. Diese dürfen keine Parameter besitzen und müssen den Rückgabetyp „void" haben. Der Aufruf der Testklassen geschieht durch Reflection. Es kann pro Testklasse jeweils eine Methode mit „@Before" beziehungsweise „@After" annotiert sein, die dann vor beziehungsweise nach jedem Testfall ausgeführt wird. Es sind im Internet eine Reihe von Tutorials verfügbar, die die API von JUnit ausführlicher demonstrieren, als dies an dieser Stelle möglich ist (etwa das JUnit Cookbook [2]). Im Folgenden liegt der Fokus auf Stilfragen und den Möglichkeiten der zusätzlichen Bibliotheken.

Die Beispiele sind aus den fiktiven Tests eines Systems zur Bestellabwicklung eines international agierenden Großhändlers gegriffen. Verschiedene Waren müssen termingerecht an Händler in aller Welt verschickt werden. Listing 1 zeigt einen Unit-Test aus diesem Kontext. Aus Sicherheitsgründen verlangt der Händler, dass Liefer- und Rechnungsadresse aus dem gleichen Land stammen müssen. Dies soll dadurch gewährleistet werden, dass „setAddresses(...)" bei Abweichungen die Lieferadresse automatisch auf die Rechnungsadresse ändert.

Die ersten drei Zeilen stellen die Ausgangssituation her, indem Objekte für eine Bestellung und die zugehörigen Rechnungs- und Lieferadressen erzeugt werden. Der daran anschließende Aufruf von „setAddresses(...)" steht für die Phase „Durchführen". Die abschließende Assertion überprüft die Korrektheit des Ergebnisses. Da die gezeigte Business-Logik keine Seiteneffekte aufweist, kann ein abschließender Aufräumschritt entfallen.

Im Beispiel besteht die Überprüfung nur aus einer einzigen Assertion, da dies ausreicht, um das Feature des Testfalls zu kontrollieren. Man sollte es vermeiden, ein Konzept in verschiedenen Testfällen mehrmals zu überprüfen, da dies häufig zur Vermischung verschiedener Konzepte führt und dadurch in Zukunft die Wartung der Tests deutlich aufwändiger wird.

Gelingt es nicht, einen Test in das vorgestellte Schema einzupassen, oder wird der Testfall dadurch unübersichtlich, hat dies meist mehrere Gründe. In vielen Fällen ist der zu testende Code zu kompliziert und vereint selbst mehrere unterschiedliche Aufgaben unter einer einzigen Schnittstelle. Auch eine enge Verknüpfung des Produktcodes mit anderen Schnittstellen kann problematisch sein. Ähnlich verhält es sich, wenn der Unit-Test nicht fein genug gegliedert wurde und wegen seines Umfangs eigentlich in kleinere Einheiten aufgeteilt werden sollte.

Die Struktur der Tests unterstützt den Entwickler dabei, die Orientierung über die Vielzahl der Tests zu behalten. Aussagekräftige Fehlermeldungen spielen später beim Debuggen eine ebenso wichtige Rolle.

Fehlermeldungen lesbar gestalten

Hamcrest [3] ist ein Framework, mit dem man deklarative Kriterien definieren kann, die Objekte erfüllen müssen. Streng genommen handelt es sich dabei nicht um ein eigenständiges Testframework, sondern um eine Bibliothek, die in sich verschiedene Testframeworks (JUnit, TestNG oder jMock) integriert.

In Hamcrest überprüft ein Matcher, ob ein Kriterium erfüllt ist, und übernimmt im Fehlerfall die Aufgabe, eine leicht verständliche Fehlermeldung zu erzeugen. Die Bedingungen können dadurch häufig eleganter formuliert werden, als dies bei den von JUnit angebotenen Assertions der Fall ist:

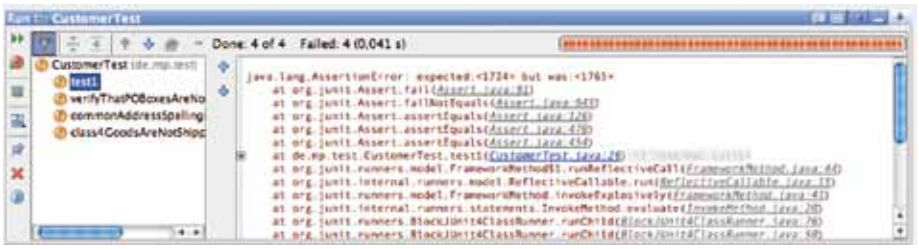


Abbildung 1: JUnit-Testrunner von IntelliJ IDEA mit gescheiterten Testfällen

```
assertFalse(
      actual.contains(poBox));
```

Der Leser muss die negierte Bedingung erst ins Gegenteil verkehren, bevor er ihren Sinn erschließen kann. Die folgende „assertThat()"-Anweisung aus dem Hamcrest-Framework entspricht der natürlichen Leseweise deutlich mehr.

```
assertThat(
   "customer shipping address",
   address,
   not(containsString(
       "Postfach")));
```

Im Fehlerfall erzeugt sie außerdem eine Fehlermeldung, die leichter auf mögliche Problemursachen schließen lässt:

```
java.lang.AssertionError:
customer shipping address
Expected: not a string
      containing "Postfach"
got: "Postfach 12"
```

Hier liegt sofort die Vermutung nahe, dass die Business-Logik, die den Versand an Postfächer unterbinden soll, versagt haben könnte. „ContainsString()" ist einer der von Hamcrest mitgelieferten Matcher. Tabelle 1 bietet einen Überblick über die wichtigsten Matcher der Bibliothek.

Hamcrest bietet die Möglichkeit, eigene Matcher zu entwickeln und diese zu umfangreichen Kriterien zu kombinieren. Dadurch können auch komplexe Sachverhalte später leicht nachvollzogen werden. Die folgende Anweisung verifiziert beispielsweise, dass sich das Versandziel in Afrika oder Südamerika befindet:

```
assertThat(
   "shipping country is valid",
   shippingCountry,
   is(anyOf(southAmerican(),
            african())));
```

Eine mögliche Implementierung für den zu „southAmerican()" gehörenden Matcher ist in Listing 2 skizziert. Die Methode „matchesSafely()" ist dabei für den eigentlichen Vergleich auf Korrektheit verantwortlich, während „describeTo()" die Beschreibung der Aufgabe des Matchers übernimmt. Diese wird im Fehlerfall durch Hamcrest in die Meldung übernommen. Durch die bereitgestellten Hilfsmethoden „appendText()", „appendValue()" und „appendList()" kann die gewünschte Beschreibung schnell zusammengestellt werden.

Findet JUnit durch die letzte Assertion eine Abweichung vom spezifizierten Verhalten, scheitert der Test mit der folgenden aussagekräftigen Fehlermeldung:

```
java.lang.AssertionError:
shipping country is valid
Expected: (a country of South
   America [<ARGENTINIA>,
  <BOLIVIA>, <BRAZIL>, <CHILE>]
  or a country of Africa
  [<BURUNDI>, <COMOROS>,
  <DJIBOUTI>, <ERITREA>])
got: <GERMANY>
```

Assertions sollen möglichst präzise beschreiben, was vom Code erwartet wird. Das Wie tritt dabei in den Hintergrund, da die Implementierungsdetails dem Leser den Blick auf das wirklich Wichtige verstellen. Die ausschweifende Syntax von Java bekommt man am besten durch eigene Matcher und Hilfsmethoden (hier: „south American()") in den Griff. Die Erfahrung zeigt, dass sich der zusätzliche Schreibaufwand beim Erstellen der Tests bei der späteren Fehlersuche schnell bezahlt macht.

Hamcrest ist in einer abgespeckten Fassung seit Version 4.4 in JUnit enthalten. Die auf der Webseite des Projekts bereitgestellte Fassung enthält eine Reihe zusätzlicher Matcher, die im JAR von JUnit nicht enthalten sind. Leider sind die Paketnamen der einzelnen Klassen nicht kompatibel. Daher muss beim Einsatz der umfangreicheren

Grundlegende Matcher	
anything	trifft auf jedes Objekt zu
describedAs	ermöglicht die Angabe einer individuellen Fehlerbeschreibung
is	syntaktischer Zucker zur Verbesserung der Lesbarkeit ohne eigene Funktionalität
Logik	
allOf	trifft zu, falls alle als Parameter übergebenen Matcher zutreffen (Gegenstück zu &&)
anyOf	trifft zu, falls mindestens ein als Parameter übergebener Matcher zutrifft (Gegenstück zu ||)
not	negiert die Aussage des übergebenen Matchers
Object	
equalTo	überprüft auf Objektgleichheit durch equals()
instanceOf	trifft zu, falls der Wert den angegebenen Typ hat
notNullValue, nullValue	Test auf null bzw. ungleich null
sameInstance	trifft zu, falls die Objekte identisch sind (Gegenstück zu ==)
Collections	
hasEntry, hasKey, hasValue	überprüft, ob eine Map einen Entry, Schlüssel oder Wert enthält
hasItem	überprüft, ob eine Collection einen Wert enthält
hasItemInArray	überprüft, ob ein Array einen Wert enthält
Number	
closeTo	überprüft, ob sich zwei Gleitkommazahlen mit einer gewissen Genauigkeit entsprechen
greaterThan greaterThanOrEqualTo lessThan lessThanOrEqualTo	Vergleich anhand der Ordnung von Objekten
Text	
equalToIgnoringCase	überprüft auf Gleichheit unabhängig von Groß-/Kleinschreibung
equalToIgnoringWhiteSpace	überprüft auf Gleichheit unabhängig von Leerzeichen
containsString, endsWith, startsWith	überprüft Zeichenketten

Tabelle 1: Auszug aus den mitgelieferten Hamcrest-Matchern

Version darauf geachtet werden, dass die Hamcrest-Bibliotheken vor den JUnit-Bibliotheken im Klassenpfad stehen.

Make-It-Easy: Erzeugen von Testdaten

Testdaten sind neben der Spezifikation des erwünschten Verhaltens der wichtigste Bestandteil eines Unit-Tests. Features können oft nur sinnvoll getestet werden, wenn eine Reihe von Geschäftsobjekten für den Aufruf der API bereitsteht. Meist müssen die beteiligten Objekte erst durch einen umfangreichen Initialisierungsprozess in den Ausgangszustand versetzt werden. Durch das Framework Make-It-Easy [4] lässt sich der im Unit-Test nötige Initialisierungscode auf ein Minimum reduzieren. Betrachten wir unter diesem Aspekt noch einmal den Testfall in Listing 1.

In den ersten drei Zeilen, dem Schritt „Vorbereitung", werden die benötigten Objekte initialisiert. Durch die statisch importierten Hilfsmethoden des Frameworks ist die Ausgangssituation auf einen Blick erkennbar: Es werden eine Bestellung und Adressen aus zwei verschiedenen Ländern erzeugt:

```
make(an(_Address_,
    with(country, GERMANY)))
make(an(_Address_,
    with(country, ARGENTINIA)))
```

Alle anderen Felder des Objekts sind aus Sicht des Testfalls nicht relevant und müssen daher auch nicht explizit spezifiziert werden. Sie werden durch das Framework automatisch mit vordefinierten Werten belegt. Listing 3 zeigt den Aufbau der Factory.

Für jedes Feld des Geschäftsobjekts wird eine statische Property definiert, die im Unit-Test durch eine „with(...)"-Klausel mit dem gewünschten Wert belegt werden kann. Durch die Verwendung von Generics wird die Typsicherheit gewährleistet.

```
public final static
     Property<Address, String>
     street = newProperty();
```

Die eigentliche Erzeugung des gewünschten Objekts wird durch den Aufruf der Methode „instantiate()" des bereitgestellten Instantiator-Objekts „_Address_" erledigt. Dies geschieht im Kontext des Unit-Tests durch den statischen Aufruf von „make(…)", nachdem alle Properties

```
public class SouthAmericanCountryMatcher extends TypeSafeMatcher<Country> {
        private final static Country[] SOUTH_AMERICA = {
                Country.ARGENTINIA, Country.BOLIVIA,
                Country.BRAZIL, Country.CHILE // usw.
        };

        @Override
        protected boolean matchesSafely(Country c) {
            return Arrays.asList(SOUTH_AMERICA).contains(c);
        }

        public void describeTo(Description description) {
            description.appendText(„a country of South America „);
            description.appendValueList("[", ", ", "]", SOUTH_AMERICA);
        }

        public static Matcher<Country> southAmerican() {
            return new SouthAmericanCountryMatcher();
        }
    }
```

Listing 2

```
public class AddressMaker {
    public final static String DEFAULT_STREET = "Hansestraße 7";
    public final static String DEFAULT_CITY = "20359 Hamburg";
    public final static Country DEFAULT_COUNTRY = Country.GERMANY;

    public final static Property<Address, String> street = newProperty();
    public final static Property<Address, String> city = newProperty();
    public final static Property<Address,  Country> country = newProperty();

    public static Instantiator<Address> _Address_ = new
Instantiator<Address>() {
        public Address instantiate(PropertyLookup<Address> lookup) {
            return new Address(
                    lookup.valueOf(street, DEFAULT_STREET),
                    lookup.valueOf(city, DEFAULT_CITY),
                    lookup.valueOf(country, DEFAULT_COUNTRY));
        }
    };
}
```

Listing 3

mit den Werten gefüllt wurden, die in den „with(…)"-Klauseln spezifiziert wurden. Innerhalb der Methode erfolgt der Zugriff dann bequem über den bereitgestellten Parameter „lookup". Lässt sich dort keine Belegung für einen Parameter finden, wird stattdessen der angegebene Vorgabewert zurückgeliefert:

```
public Address instantiate(
PropertyLookup<Address> lookup)
...
lookup.valueOf(street,
     DEFAULT_STREET)
```

Durch diese Gestaltung der API können auch komplizierte Objekte konstruiert werden, die eine konsistente Belegung umfangreicher Konstruktor-Parameter erfordern.

Fazit

Wir haben gesehen, dass man sich durch den richtigen Einsatz der verfügbaren Frameworks auf das Wesentliche konzentrieren kann: den zu testenden Use Case. Durch sorgfältiges Formulieren wird das Augenmerk des Entwicklers auf die eigentliche Intention des Codes gelegt, statt diesen hinter der manchmal etwas ausschweifenden Syntax von Java zu verstecken. Änderungen, wie die verständliche Benennung von Variablen oder auch

sorgfältig formulierte Fehlermeldungen, können die Lesbarkeit bereits drastisch steigern.

Unit-Tests sind eine Investition in die Zukunft, da sie im Laufe der Zeit ein Auffangnetz bilden, das vor den unbeabsichtigten Auswirkungen von Änderungen schützt. Der Mehraufwand, den man anfangs für die Erstellung einplanen muss, macht sich langfristig durch die bessere Unterstützung bei der Fehlersuche wieder bezahlt. Auch wenn also kurz vor dem Release noch dringend Änderungen nötig werden sollten, können Sie danach wenigstens beruhigt ins Wochenende starten.

Weitere Informationen

[1] Unit: http://www.junit.org/

[2] JUnit Cookbook: http://junit.sourceforge.net/doc/cookbook/cookbook.htm

[3] Hamcrest: http://code.google.com/p/hamcrest/

[4] Make-It-Easy: http://code.google.com/p/make-it-easy/

Dirk Dittert begann 2006 bei der Method Park Software AG als Software Engineer und übernahm Ende 2009 die Aufgaben eines Projektleiters in der Produktentwicklung. Er ist ISQi Certified Professional for Project Management und IREB Certified Professional for Requirements Engineering.

Kontakt:
Dirk Dittert
dirk.dittert@methodpark.de

Mitglieder stellen sich vor

Sun User Group Deutschland e.V.

Die Sun User Group e.V. (SUG) wurde 1987 gegründet. In der Anfangszeit trat Sun Microsystems im großen Umfang als Förderer auf. Ende der 1990er Jahre verlor das Unternehmen das Interesse an User Groups. Trotzdem hat der Verein überlebt und mit den Source Talk Tagen in Göttingen weiterhin Service-Leistungen für seine Mitglieder erbracht und den Kontakt zum Unternehmen gehalten. Die Übernahme von Sun Microsystems durch Oracle stellt den Verein vor die Herausforderung, den Interessen Anwender von sun Technologie beim neuen Eigentümer Gehör zu verschaffen.

In den Anfangsjahren hat der Verein durch sein Wirken sicher dazu beigetragen, die Workstations und Server der Firma in Deutschland bekanntzumachen. Anfang der 1990er Jahre waren die Jahrestagungen der SUG ein beliebter Treffpunkt zum zwanglosen Informationsaustausch. Der CD-Service der SUG mit Open Source Software war damals populär und mangels Breitbandinternet auch dringend notwendig. Nachdem sich Sun Microsystems aus der Vereinsunterstützung und damit einhergehend aus den Universitäten zurückgezogen hatte, musste sich der Verein neu orientieren.

Mit den Source Talk Tagen, die aus den Jahrestagungen der Java User Group Deutschland e.V. (JUG) und SUG hervorgegangen ist, wurde die Tradition der 1990er Jahre wieder aufgenommen. Wichtige Themen sind dabei immer noch OpenSolaris und andere Sun-Produkte. Das Spektrum wurde aber in Richtung Open Source erweitert, um der Nachfrage der Mitglieder nach fundierten Informationen entgegenzukommen.

Der Verein ist dem iJUG beigetreten, um besonders im Interesse seiner Mitglieder die Betonung auf Java als Sun-Technologie zu legen.

Weitere Informationen: http://www.sugd.de

Java gibt es überall auf der Welt. Der Artikel zeigt, was man beachten muss, um die Vorteile der Globalisierung genießen zu können, ohne die Nachteile zu erleiden.

Java everywhere

Till Hahndorf, Sourceconomy GmbH

Wenn ein Projekt ansteht, das eine gewisse Manpower zur Umsetzung benötigt, geht es los: Wo ist auf die Schnelle ein kompetentes Java-Team zu finden, wer hat Zugang zu den entsprechenden Netzwerken, wie unterscheidet man schnell zwischen den guten und den nicht ganz so guten Kollegen? Und wie funktioniert das im Ausland?

In der globalisierten Welt ist ja eigentlich alles klar: Wenn man für sein Projekt vor der Haustür keine Leute findet, sieht man sich eben einmal etwas weiter weg um. Es hat sich ja inzwischen herumgesprochen, dass es Java-Entwickler nicht nur in Deutschland gibt. Ausgereifte technische Spezifikationen und bewährte Prozesse sorgen im Normalfall dafür, dass ein ausländisches Team harmonisch in die Entwicklung integriert werden kann. So werden Kapazitäten schnell verfügbar gemacht und flexibel entlang der Bedarfslage in ein Projekt eingebaut. Günstiger als mit deutschen Kollegen ist das im Normalfall auch, denn die Tagessätze im Ausland liegen bei 25 bis 30 Prozent der hier üblichen Sätze, dazu kommt ein gewisser Zusatzaufwand für das internationale Handling – je erfahrener der Auftraggeber ist und je längerfristig er plant, desto mehr dieser Kosteneinsparungen kann er am Ende tatsächlich realisieren. Soweit der Stand der Dinge.

Bisweilen wird mit einem Entwicklungsprojekt allerdings konzeptionelles Neuland betreten: Dann wird im Entwicklerteam weit mehr gesucht als nur Kapazität für die Abarbeitung von Anforderungen. Es werden erfahrene Gesprächspartner gefordert, die auf Augenhöhe mit dem Auftraggeber eine Architektur entwerfen und eine Gesamtlösung planen können, die ein Gespür für das Businessumfeld und den Markt des Auftraggebers haben und die in vergangenen Projekten reichlich Erfahrung gesammelt haben, um als Sparringpartner in einer Entwurfsphase zu dienen – kurz gesagt: die einem deutschen Entwicklerteam in wenig nachstehen, aber wesentlich flexibler einsetzbar sind und weniger kosten.

In einer solchen Zusammenarbeit ist der Kommunikationsbedarf hoch und mit ihm die Rahmenanforderungen an den Partner im Ausland: möglichst parallele Arbeitszeiten zu denen des Auftraggebers und eine möglichst niedrige Sprachbarriere, was heutzutage heißt: gutes Englisch. Ferner etwas, das man als kulturelle Reibungsarmut bezeichnen kann – denn sonst geht an den kommunikativen Schnittstellen vielleicht mehr Energie verloren, als für das Gesamtprojekt überhaupt zur Verfügung steht. Eine Anforderung aber steht über allen anderen: technische Kompetenz und echtes inhaltliches Verständnis der anstehenden Aufgaben.

Für viele Kollegen verbietet sich die Zusammenarbeit mit Teams aus Indien und China – nicht immer aus tatsächlicher Erfahrung, sondern oftmals auch aus einer Mischung aus Vermutungen und Anekdoten sowie einer gewissen Scheu, sich auf diese unbekannten Kollegen einzulassen und eigene Erfahrungen zu sammeln. Es gibt ja tatsächlich gute Alternativen: In Osteuropa hat sich längst eine professionelle Java-Szene etabliert, sowohl innerhalb als auch außerhalb der EU – wobei der Schritt über die EU-Außengrenze zusätzliche Kosteneinsparungen mit sich bringen kann, gewöhnlich aber keinen Qualitätsverlust bedeutet.

Maßgeblich ist nicht, in welchem Land der Dienstleister der Wahl angesiedelt ist – das einzigWichtige sind seine individuelle Kompetenz, seine Prozessreife, seine Qualitätssicherung, seine Infrastruktur und all die anderen Merkmale, die einen ernst zu nehmenden Dienstleister auszeichnen. In welchem Land sein Entwicklungszentrum steht, ist zwar auch wichtig, aber erst in zweiter Linie – wenn es um die langfristige Dimension der Zusammenarbeit geht und darum, ob der Dienstleister nach heutigem Ermessen auch einige Jahre später noch eine gesunde und buchbare Java-Mannschaft vorweisen kann. Java hat eine professionelle Kultur rund um den Erdball hervorgebracht – es gibt nur wenige Länder, in denen noch keine JUG die Fahne hochhält und sowohl inner- als auch außerhalb von konkreten Projekten die Java-Philosophie diskutiert und voranbringt.

Als aktuelles Beispiel dient die JUG Belgrad (www.javasvet.rs – siehe Kasten), die mit großem Erfolg seit Jahren die Java-Community der serbischen Hauptstadt zusammenhält. Solche Gruppen gibt es eben nicht nur bei uns – es gibt sie auch „dort" – und sie sind zugleich Stütze und Bühne der jeweiligen regionalen Java-Community. Für fast alle osteuropäischen Nachbarländer – die nahen wie die etwas entfernteren – gilt: Die IT-Szene vor Ort ist lebendig und facettenreich, es gibt dort wie hier gute und weniger gute Anbieter – und dort wie hier muss man jemanden haben, der einem die Szene erschließt und die wirklich kompetenten Anbieter zugänglich macht. Eine JUG vor Ort ist sicherlich kein schlechter Ausgangspunkt.

Till Hahndorf hat einige Jahre Technologieberatung bei Accenture absolviert und danach das europäische Standbein einer US/asiatischen Softwarefirma aufgebaut. Mit dem Ziel, kleine und mittelgroße Unternehmen bei ihrem Schritt in die globalisierte IT-Welt zu unterstützen, gründete er im Jahr 2007 das Beratungs- und Systemhaus Sourceconomy in Freiburg, das er bis heute als Geschäftsführer leitet.

Der Autor im Interview mit Igor Spasić, dem Gründer und Evangelisten der JUG Belgrad (www.javasvet.rs)

Wie läuft die JUG Belgrad?

Igor Spasić: Die Java User Group heißt hier „Javasvet", „svet" ist serbisch und bedeutet „Welt". Wir haben einen Verteiler von gut 1000 Adressen und immer zwischen 20 und 50 Leute auf unseren Treffen.

Engagieren sich denn tatsächlich so viele Kollegen für die JUG?

Igor Spasić: Die Hauptarbeit hängt an einem sehr kleinen Kreis von Begeisterten. Wir würden uns sehr wünschen, dass wir von Unternehmensseite mehr Unterstützung bekommen.

Igor Spasić (links, Java User Group Belgrad) und Till Hahndorf (Sourceconomy GmbH, D)

Was tut Sun für Euch?

Igor Spasić: Wir hatten letztes Jahr ein großes Event, da kam sogar jemand von Sun und hat einen Vortrag gehalten. Ansonsten ist das Engagement, sagen wir mal „überschaubar" (lacht).

Und die Unternehmen hier vor Ort?

Igor Spasić: Die meisten Firmen hier sind Outsourcing-Anbieter für den westeuropäischen Markt, viele davon sehr professionell unterwegs. Wir bekommen Räume für unsere Treffen, das ist schon sehr gut. Aber der Druck ist auch hoch – viele von uns arbeiten viele Stunden und in engen Zyklen, da muss für zusätzliche Aktivitäten erst einmal Zeit geschaffen werden.

Wie sieht es mit externen Referenten aus?

Igor Spasić: Letztes Jahr kam Rod Johnson und hat uns seine Vision vom Spring Framework vorgestellt, das war schon sehr interessant. Aber es ist wirklich immer ein Kampf, bis man ein bekanntes Gesicht hierher nach Belgrad holen kann.

Was könnten andere JUGs tun – gibt es Partnermodelle?

Igor Spasić: Nichts Vordefiniertes – aber es wäre großartig! Wir würden uns über Java-Besuch aus Deutschland freuen, über einen fachlichen und persönlichen Austausch, vielleicht über eine Kooperation mit unseren jeweiligen Universitäten.

Kontakt:
Till Hahndorf
till.hahndorf@sourceconomy.com

Impressum

Herausgeber:
Interessenverbund der Java User Groups e.V. (iJUG)
Tempelhofer Weg 64, 12347 Berlin
Tel.: 0700 11 36 24 38
www.ijug.eu

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

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

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

Gestaltung und Satz:
Claudia Wagner,
DOAG Dienstleistungen GmbH

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

Mediadaten und Preise finden Sie unter:
www.ijug.eu/images/vorlagen/2010-ijug-mediadaten_java_aktuell.pdf

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

Die Fachzeitschrift für Datenprofis!

Für Entwickler Administratoren Softwarearchitekten

Bestellen Sie Ihr kostenloses Kennenlern-Exemplar unter:

www.databasepro.de/probelesen

Mit den Eclipse-Projekten EMF, Xtext, GMF und MWE entstehen Werkzeugketten für DSLs im Handumdrehen. Sind die Potenziale der modellbasierten Software-Entwicklung erst erkannt, beginnen die Modelle schnell zu wachsen. Die User Experience leidet dann oft an der schlechten Performance der Werkzeuge. Fortgeschrittene Entwickler können mit etwas Aufwand Modelle schneller laden lassen, unnötige Proxyfizierung der EObjects vermeiden und den MWE Generator beschleunigen. Der Artikel setzt Wissen im Umgang mit dynamischen Generator-Templates und den Laufzeiterweiterungen der GMF-Tools voraus.

Modeling-Performance

Robert Wloch, itemis AG

Das Unternehmen des Autors entwickelt für einen Kunden eine komplexe Werkzeugkette zur Software-Modellierung. Diese basiert auf Open-Source-Komponenten des Eclipse-Projekts, speziell aus dem Modeling Project (http://www.eclipse.org/modeling/). Die Speicherung des Domänen-Modells erfolgt mit Xtext. Zur Laufzeit dient das EMF als Schnittstelle zwischen den Modulen der Werkzeugkette. Ein GMF-basierter Editor ermöglicht die grafische Bearbeitung von Fragmenten des Modells. Die Generierung der Artefakte erfolgt aus dem Domänenmodell mit Hilfe der MWE durch entsprechende Xpand-Templates.

In diesem Projekt besteht ein typisches Modell aus etwa 400 Dateien, wobei rund zwei Drittel davon GMF-Diagramme sind. Ein solches Modell benötigt zwischen 4 und 6 MB Speicherplatz. Sollen Modell-Editoren zugleich ein auf mehrere Dateien verteiltes Modell bearbeiten können, müssen sie sich über eine gemeinsame EditingDomain synchronisieren. Dafür stellt das GMF Tools Projekt (http://code.google.com/p/gmftools/) entsprechende Erweiterungen für das GMF bereit.

Plug-in	Version
Eclipse Modeling Tools	1.2.1.20090918-0703
Eclipse Platform	3.5.1.M20090917-0800
EMF SDK	2.5.0.v200906151043
GMF SDK	2.2.1.v20090814-1000
MWE SDK	0.7.2.v200908120417
Xpand SDK	0.7.2.v200908120436
Xtend SDK	0.7.2.v200908120607
Xtext SDK	0.7.2.v200908120607
GMF Tools	1.2.0.201002161105

Tabelle 1: Eclipse Plug-in Versionen des Testsystems

Aufgrund einer Anforderung enthält die Werkzeugkette einen Modell-Browser, der es ermöglicht, Teile des Modells auch ohne geöffneten Editor zu bearbeiten. Dazu wird die Properties View von Eclipse angesteuert, ähnlich wie das herkömmliche EMF- und GMF-Editoren bereits realisieren. Das Bearbeiten von Modellen ohne geöffneten Editor ist allerdings nur möglich, wenn es zuvor bereits unabhängig von Editoren geladen wird. Dazu laden Anwender das Modell einmalig und vollständig in ein ResourceSet über eine Aktion des Modellbrowsers. Die gemeinsame EditingDomain verwaltet dieses ResourceSet, auf das dann alle Editoren zugreifen können.

Das Testsystem

Die Vergleiche der folgenden Performance-Verbesserungen erfolgen unter Windows XP auf einem Pentium 4 (2 x 3 GHz) mit 2 GB RAM. Das Modell belegt auf der Festplatte 4,26 MB Speicherplatz und ist verteilt auf 386 Dateien, 308 davon sind GMF-Diagramme. Die Modellierungsumgebung läuft unter Eclipse Galileo mit den Plug-in-Versionen, die in Tabelle 1 gelistet sind.

Tuning 1: Modell-Ladezeit reduzieren

Das Laden des Testmodells in ein GMF ResourceSet kann sehr lange dauern, da während des Ladevorgangs einige Adapter ihre Arbeit verrichten. Insbesondere der CrossReferenceAdapter von GMF benötigt sehr viel Zeit für das Auflösen von Querverweisen. Insgesamt lädt das Testmodell etwa 110 Sekunden. Diese Ladezeit knapp zwei Minuten ist allerdings zu lang und nach einem Tipp eines Kollegen gelang es, sie auf 13 Sekunden zu reduzieren.

Die Suche mit einem Profiler zeigt, dass Xtext nicht das Problem ist: Es lädt alle Modell-Dateien sehr schnell. Flaschenhals ist der CrossReferenceAdapter von GMF. Er versucht standardmäßig, alle Querverweise sofort aufzulösen. Das ist jedoch das Gegenteil von dem, was Xtext mit dem Lazy-Loading-Mechanismus verfolgt. Das Verhalten des Adapters musste daher geändert werden. Die plugin.xml, die GMF in jedes Diagramm Plug-in generiert, enthält eine Extension-Deklaration ähnlich dieser:

```
<extension
              point="org.eclipse.emf.
transaction.editingDomains">
      <editingDomain
              factory="org.eclipse.gmf.
runtime.diagram.core.DiagramEditingDo-
mainFactory"
              id="xyz.sharedEditingDo-
main">
      </editingDomain>
</extension>
```

Die generierte DiagramEditingDomainFactory erbt von GMFEditingDomainFactory, die schließlich den CrossReferenceAdapter erzeugt (siehe Listing 1).

Da der zu ändernde Bereich nun bekannt ist und er sich nicht im generierten Code befindet, gibt es zwei Möglichkeiten, die gewünschte Modifikation einzubauen:

- Den GMF-Generator dazu bringen, eine veränderte plugin.xml zu generieren.
- Mittels ObjectTeams [http://www.eclipse.org/objectteams/] oder AspectJ (http://www.eclipse.org/aspectj/) das Laufzeitverhalten der generierten DiagramEditingDomainFactory ändern.

Die zweite Variante sorgt zwar für deutlich weniger Code, kann aber nicht bei jedem Projekt umgesetzt werden. Für den ersten Lösungsweg ist es erforderlich, das GMF-Generatortemplate plugin.xpt anzupassen, da es die plugin.xml erzeugt (siehe Listing 2).

Die UneagerDiagramEditingDomainFactory schaltet mit „resolveProxies=false" das Auflösen der Querbeziehungen im CrossReferenceAdapter aus (siehe Listing 3).

Mit dieser Änderung konnte die Ladezeit des Testmodells um fast 90 Prozent reduziert werden. Das ist auf den ersten Blick sehr beeindruckend. Aber es gibt einen Nachteil, der nicht unterschätzt werden sollte: Die gesparte Zeit ist nicht verpufft, sondern wird durch den Lazy-Loading-Mechanismus nur auf später verschoben. Irgendwann später wird über einen Modelleditor oder die Properties View ein Zugriff auf noch nicht aufgelöste Querverweise erfolgen. Dann friert die Anwendung für einen kurzen oder längeren Augenblick ein, bis der CrossReferenceAdapter fertig ist. Diese Phasen sind nicht immer im Sinne des Nutzers. Da die Optimierung des CrossReferenceAdapters über den Rahmen des Kundenprojektes hinausgeht, wird diese Tuning-Maßnahme zu Lasten einer längeren Ladezeit nicht angewendet.

Tuning 2: Proxyfizierungen minimieren

Standardmäßig arbeiten generierte GMF-Editoren jeweils in einer eigenen Editing Domain. Wird ein Editor geschlossen, werden alle darin geladenen Modellelemente für ungültig erklärt. Der Vorgang der Proxyfizierung überführt die URI der EObjects in eine spezielle Form, sodass sie beim nächsten Zugriff erneut aus dem Dateisystem geladen werden.

Sollen jedoch verschiedene Komponenten einer komplexen Werkzeugkette auf ein gemeinsam genutztes Modell zugreifen, teilen sich all diese Komponenten eine Editing Domain. Mehrere Diagramm-Editoren können dann in beliebiger Reihenfolge und Anzahl geöffnet und geschlossen werden. Daher muss man das Standardverhalten der GMF-Editor anpassen, da es sonst zu überflüssigen und unnötigen Proxyfizierungen und Dateizugriffen während der Modellierung führt.

Offensichtlich sollte die Proxyfizierung deaktiviert werden. Eine Modifikation des

```
protected void configure(final TransactionalEditingDomain domain) {
[...]
// ensure that the cross-referencing adapter is installed
if (CrossReferenceAdapter.getExistingCrossReferenceAdapter(rset)
== null) {
      rset.eAdapters().add(new CrossReferenceAdapter());
}
[...]
}
```

Listing 1

```
«IMPORT ‚http://www.eclipse.org/gmf/2009/GenModel'»
«EXTENSION xpt::GenEditorGenerator»

«AROUND additions FOR gmfgen::GenPlugin»
<extension
      point="org.eclipse.emf.transaction.editingDomains">
      <!-- ÄNDERUNG: eigene DiagramEditingDomainFactory registrie-
ren -->
      <editingDomain
                  factory="xyz.gmf.runtime.editingdomain.Unea-
gerDiagramEditingDomainFactory"
                  id="«editorGen.diagram.editingDomainID»">
      </editingDomain>
</extension>
«ENDAROUND»
```

Listing 2

```
package xyz.gmf.runtime.editingdomain;

import org.eclipse.emf.ecore.resource.ResourceSet;
import org.eclipse.emf.transaction.TransactionalEditingDomain;
import org.eclipse.emf.workspace.WorkspaceEditingDomainFactory;
import org.eclipse.gmf.runtime.diagram.core.DiagramEditingDomainFactory;
import org.eclipse.gmf.runtime.emf.core.util.CrossReferenceAdapter;

public class UneagerDiagramEditingDomainFactory extends DiagramEditingDo-
mainFactory {
private static UneagerDiagramEditingDomainFactory instance = new
UneagerDiagramEditingDomainFactory();

public static WorkspaceEditingDomainFactory getInstance() {
      return instance;
}

protected void configure(TransactionalEditingDomain domain) {
      final ResourceSet resourceSet = domain.getResourceSet();
      if (CrossReferenceAdapter.getExistingCrossReferenceAdapter(r
esourceSet) == null) {
            boolean resolveProxies = false;
            resourceSet.eAdapters().add(new CrossReferenceAdapte
r(resolveProxies));
      }
      super.configure(domain);
}
}
```

Listing 3

```
«IMPORT 'http://www.eclipse.org/gmf/2009/GenModel'»

«AROUND dispose FOR gmfgen::GenDiagram-»
«EXPAND xpt::Common::generatedMemberComment»
public void dispose() {
«IF null = editorGen.application-»
//stopResourceListening();
«ENDIF-»
// ÄNDERUNG: De-registrieren des synchronizerDelegate
sharedResourceSetInfoDelegate
                              .removeWorkspaceSynchronizerDelegate(s
ynchronizerDelegate);

getResourceSet().eAdapters().remove(myResourceSetListener);
// ÄNDERUNG: Code für unload() auskommentiert -> keine Proxyfizie-
rung mehr
//for (java.util.Iterator/*<org.eclipse.emf.ecore.resource.Resour-
ce>*/ it = getLoadedResourcesIterator(); it.hasNext();) {
//  org.eclipse.emf.ecore.resource.Resource resource = (org.eclipse.
emf.ecore.resource.Resource) it.next();
//  resource.unload();
//}
}
«ENDAROUND»
```

Listing 4

```
Map<String, Object> externalSlotContents = new HashMap<String, Object>();
List<xyzmodel.Foo> foos = getAllFoosFromResourceSet(sharedResourceSet);
externalSlotContents.put("model", foos);
[...]
runner.run(workflowFileName, new ProgressMonitorAdapter(monitor), proper-
ties, externalSlotContents);
```

Listing 5

ResourceSetInfo.xpt Generatortemplates erzeugt eine veränderte XYZDocument-Provider.java Klasse im generierten Diagramm Plug-in (siehe Listing 4).

Die Auswirkung dieser Änderung ist sofort wahrnehmbar beim Schließen und Öffnen von GMF-Editoren. Je größer und vor allem quer-verzweigter ein Modell ist, desto stärker ist der Unterschied.

Tuning 3: Generator beschleunigen

In der Werkzeugkette des Kunden wählen Anwender Modellelemente in einem Modell-Browser aus und stoßen dann einen Generator an. In seiner ersten Variante erhielt der MWE Workflow einen Pfadverweis zu allen Modelldateien und filterte anschließend das Selektierte heraus. Um es sogar noch schlimmer zu machen, wurde der Generator für jedes ausgewählte Modellelement einzeln gestartet. Der Effekt dieses Ansatzes war, dass der Workflow bei jedem Aufruf das gesamte Modell in ein eigenes ResourceSet laden musste. Da dies für jedes selektierte Element geschah, dauerte ein vollständiger Generatorlauf ungefähr 30 Minuten.

Das Ziel ist offensichtlich: Der Generator muss das bereits geladene Modell aus dem gemeinsamen ResourceSet für die Generierung erhalten. Dazu werden alle ausgewählten Modellelemente in einer Liste gesammelt und dem MWE WorkflowRunner beim Aufruf übergeben (siehe Listing 5).

Zudem ist die MweReader-Komponente in der MWE-Steuerdatei workflow.mwe nun überflüssig, da das Modell nicht mehr eingelesen werden soll, sondern bereits übergeben wird. Der entsprechende Eintrag sieht ungefähr so aus:

```
<component class="org.
eclipse.xtext.MweReader"
uri="${modelDir}/${modelFile}">
      <!-- this class will be
generated by the xtext genera-
tor -->
      <register class="xyz.
XYZModelStandaloneSetup"/>
      <validate
value="false"/>
</component>
```

Da der WorkflowRunner nun eine Liste von Modellelementen übernimmt, ändert sich der Aufruf des Hauptemplates von:

```
<expand
value="templates::Main::main
FOR model"/>

nach:

<expand
value="templates::Main::main
FOREACH model"/>
```

Im Haupt-Template muss natürlich sichergestellt sein, dass die Regel „main" nun eine Liste von Modellelementen verarbeiten muss.

Mit diesen Änderungen braucht der Generator nur noch einmal mit den ausgewählten Modell-Elementen starten. Da diese direkt aus dem gemeinsamen ResourceSet kommen, sind auch alle referenzierten Modellelemente für den Generator erreichbar. Die Zeit für einen vollständigen Generatorlauf über dem Testmodell beträgt nun nur noch vier bis fünf Minuten im Gegensatz zu ursprünglich 30 Minuten.

Glossar

EMF	Eclipse Modeling Framework
GMF	Graphical Modeling Frame work
MWE	Modeling Workflow Engine
DSL	Domänenspezifische Sprche
EObject	Wurzelklasse aller EMF Modellelemente

Kontakt:
Robert Wloch
robert.wloch@o2online.de

Robert Wloch arbeitet als Softwareentwickler, Berater und Trainer bei der itemis AG in Leipzig. Seit fünf Jahren liegt sein Fokus auf der Entwicklung von Plug-ins für Eclipse, insbesondere zur Unterstützung modellbasierter Softwareprozesse.

Abbildung 1: Ganz konzentriert beim Netbeans-Training

Die Göttinger Source Talk Tage sind seit fünf Jahren der Technologiekongress im Bereich" IT und eLearning" im Zentrum Deutschlands. Die dreitägige Mischung aus bundesweit interessanten, einmaligen Spezialtagungen, Tracks und Trainings zu Basistechnologien – wie Java, Webanwendungen, Datenbanken, Betriebssystemen – kommt an. Veranstalter sind die Java User Group Deutschland e.V. (JUG), die Sun User Group e.V. (SUG), der Stud.-IP e.V. sowie Technologietransfer und MatheTransfer der Uni Göttingen.

Source Talk Tage in Göttingen

Stefan Koospal, Uni Göttingen

Entstanden sind die Source Talk Tage aus den Jahrestagungen der JUG und SUG. Deshalb stehen im Technikteil der Tagung Themen wie „Java" und „OpenSolaris" im Vordergrund. In diesem Jahr sind neue Themen wie „HPC with Open Source" hinzugekommen und der Bereich „Trainings" wurde wesentlich erweitert. Damit wird die Tagung immer mehr ihren Zielen gerecht.

Unternehmen und Studierenden sollen günstige Möglichkeiten geboten werden, frisches Know-how zu erwerben und gleichzeitig beim Kaffee und beim Social Event ungezwungen Kontakt untereinander aufzunehmen. Unternehmen können sich über technologisch geprägte Vorträge als interessante Arbeitgeber darstellen und ihre zukünftigen Mitarbeiter kennenlernen. Deshalb werden an Studierende Voucher für die Trainings und die Tracks vergeben.

Daneben wird es noch Spezialtagungen im Bereich „eLearning" geben. Weitere interessante Trainings sind noch in Vorbereitung. Webseite: http://www.sourcetalk.de

Kontakt:
Stefan Koospal
koospal@uni-math.gwdg.de

Das Programm von Dienstag, 30. August bis Donnerstag 2. September

Tracks:

Solutions for Business with Open Source
Lead: Dr. Hans-Georg Pagendarm, DNW Göttingen

HPC with Open Source
Lead: Dr. Thomas Alrutz, T-Systems Solutions for Research Göttingen,
Dr. Gerd Rapin, Volkswagen Wolfsburg

Aus- und Weiterbildungsmanagement mit Open Source
Lead: Marco Bohnsack, data-quest Göttingen

Java
Lead: Frank Schwichtenberg, FIZ Karlsruhe, Dr. Jens Trapp, Google Hamburg

Webtechnologien
Lead: Antonia Schwichtenberg, ontoprise Karlsruhe,
Dr. Robert Zores, mgm technology partners München

Trainings:

Apps für das iPhone
Lead: Oliver Szymanski, Source-Knights Erlangen

Netbeans 2-tägig
Lead: Toni Epple, Eppleton München, Aljoscha Rittner, Sepix Hann. Münden

Clojure
Lead: Stefan Kamphausen, Tim Oliver Kaiser, Crossgate Rosdorf

Weiterbildungs- und Seminarmanagement über Intranet und Internet
Lead: Marco Bohnsack, data-quest Göttingen

Medienbearbeitung mit Linux
Lead: Dr. Walter Stickan, Jürgen Kaeding, iwf Göttingen

Oracle Application Express (Apex) basiert vollständig auf PL/SQL – und auch der Apex-Entwickler nutzt PL/SQL für seine Anwendungen; die Programmiersprache Java kommt normalerweise nicht zum Einsatz. Allerdings ist die Oracle Datenbank bereits seit der Version 8i mit einer eingebetteten Java-Umgebung (JVM) ausgestattet, deren Funktionsumfang dem einer normalen JVM entspricht. Man kann also Stored Procedures auch in Java entwickeln. Dieser Artikel stellt die datenbankinterne JVM vor, geht auf deren Besonderheiten ein und zeigt, wie man sie in Apex-Anwendungen nutzt.

Apex und PL/SQL meet Java: unbegrenzte Möglichkeiten mit der Datenbank-JVM

Carsten Czarski, ORACLE Deutschland B.V. & Co. KG

Zunächst drängt sich natürlich die Frage auf, warum man Java in Stored Procedures oder in Apex-Applikationen überhaupt benötigt. PL/SQL ist schließlich eine mächtige Sprache mit einer umfangreichen Bibliothek (Packages) – näher kann man eigentlich gar nicht an der Datenbank sein. Es gibt jedoch Aufgaben, die PL/SQL nicht lösen kann, weil es keine Packages dafür gibt, wie zum Beispiel das Auslesen von Verzeichnissen auf dem Datenbank-Server.

Das Paket „UTL_FILE“ erlaubt zwar das Lesen, Schreiben oder Löschen einer Datei, man muss deren Namen jedoch vorher kennen. Für Java ist das hingegen überhaupt kein Problem – Verzeichnis-Listings gehören zum normalen Sprachumfang. Ruft man den Java-Code dann als „Table Function“ auf, so werden Abfragen wie in Listing 1 möglich. Das entsprechende Java-Package steht

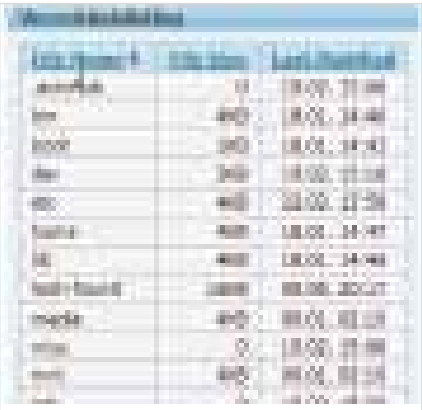


Abbildung 1: Das Verzeichnis-Listing als Apex-Bericht in einer 11g-Datenbank (siehe SQL-Abfrage in Listing 1)

```
SQL> select file_name, file_size, last_modified
  2  from table(file_pkg.get_file_list_p(file_pkg.get_file(,/‘)));

FILE_NAME                      FILE_SIZE LAST_MODIFIED
------------------------------ ---------- ----------------
bin                                  4096 18.01.2010 18:20
boot                                 1024 18.01.2010 18:17
:
usr                                  4096 18.01.2010 18:17
var                                  4096 18.01.2010 18:18

25 Zeilen ausgewählt.
```

Listing 1: Java Stored Procedures in Aktion: Verzeichnis-Inhalte als Tabelle zurückgeben

als Download bereit [3]. Mit der SQL-Abfrage könnte man als Nächstes einen Apex-Bericht erzeugen (siehe Abbildung 1).

Die Datenbank-JVM hat grundsätzlich den gleichen Funktionsumfang wie eine JVM außerhalb der Datenbank. Damit erweitern sich die Möglichkeiten von Stored Procedures in der Datenbank erheblich:

- ZIP-Archive ein- und auspacken
- Die Datenbank als FTP-Client nutzen
- E-Mail von einem Mailserver abrufen

Die erste Java Stored Procedure

Java-Code kann wie PL/SQL mit einem „CREATE OR REPLACE“-Kommando in die Datenbank geladen werden (siehe Listing 2).

Erfahrenen Java-Programmierern fällt sicher sofort auf, dass die Java-Methode „sayHello“ als „static“ definiert ist. Java ist eine objektorientierte Programmiersprache, PL/SQL dagegen rein prozedural. Das Schlüsselwort „static“ auf Java-Seite bewirkt, dass die Methode eine „Klassenmethode“ ist und nicht an ein Objekt gebunden – nur solche Java-Methoden sind als „Stored Procedures“ einsetzbar. Innerhalb der Methode darf man dagegen sehr wohl Objekte und damit den gesamten Java-Sprachumfang verwenden.

Zum Aufrufen der Java-Methode muss noch eine Call Specification, auch „PL/SQL-Wrapper“ genannt, eingerichtet werden (siehe Listing 3). Neben der Zuordnung der PL/SQL-Funktion zur Java-Methode findet darin auch die Abbildung der Java-Datentypen auf ihre SQL-Pendants statt (Datentyp-Mapping). Die Java-Methode liefert einen String zurück, dieser wird auf „VARCHAR2“ abgebildet. Analog dazu gibt es Mappings für die anderen Datentypen wie „NUMBER“, „DATE“ oder „TIMESTAMP“. Selbst komplexe Objekte oder Arrays können auf Datenbank-Objekttypen abgebildet

werden; allerdings sind dazu auf Java-Seite die Klassen „oracle.sql.ARRAY“ beziehungsweise „oracle.sql.STRUCT“ erforderlich.

Der Aufruf der Java Stored Procedure wird, auch wenn er aus einem PL/SQL-Package heraus erfolgt, in jedem Fall durch die „SQL-Ebene geleitet. Das bedeutet, dass Java-Typen nur auf SQL-Datentypen abgebildet werden können – die direkte Abbildung auf reine PL/SQL-Datentypen wie „boolean“ oder „record“ ist dagegen nicht möglich.

Fertige Java-Klassen oder Bibliotheken laden

Java Stored Procedures sind nicht nur wegen des Standard-Sprachumfangs der Java-Umgebung interessant, sondern auch, weil es viele – teilweise quelloffene – Java-Bibliotheken für verschiedene Anwendungsgebiete gibt. Da diese mitunter aus sehr vielen Java-Klassen bestehen, wäre das Laden per SQL-Skript eher unpraktisch. Die Datenbank bietet daher das Kommandozeilen-Werkzeug „loadjava“ an, mit dem kompilierte Java-Klassen oder ganze Java-Archive (jar-Dateien) auf einmal in die Datenbank geladen werden können. Dazu ein Beispiel:

```
$ loadjava -user scott/tiger -o
-r -v javalib.jar
```

```
create or replace and compile java source named meine_erste_java_pro-
zedur as
public class HelloWorld {
  public static String sayHello() {
    return „Hallo Welt! Ich bin eine Java Stored Procedure.\n“ +
           „Die Datenbank-JVM hat die Version „ +
           System.getProperty(„java.version“);
  }
}
/

Java wurde erstellt.
```

Listing 2: Eine einfache Java Stored Procedure: „Hallo Welt“

```
create or replace function say_hello return varchar2 is
language java name ‚HelloWorld.sayHello() return java.
lang.String‘;
/

Funktion wurde erstellt.
```

Listing 3: PL/SQL Call Specification für Java Stored Procedure einrichten

```
SQL> select say_hello from dual;

SAY_HELLO
------------------------------
Hallo Welt! Ich bin eine Java Stored Procedure.
Die Datenbank-JVM hat die Version 1.4.2_04
```

Listing 4: Test der Java Stored Procedure in einer 10g-Datenbank

```
FEHLER in Zeile 2:
ORA-29532: Java-Aufruf durch nicht abgefangene Java-Exception beendet:
java.security.AccessControlException: the Permission
(java.io.FilePermission / read) has not been granted to SCOTT.
The PL/SQL to grant this is
dbms_java.grant_permission( ‚SCOTT‘,
‚SYS:java.io.FilePermission‘, ‚/‘, ‚read‘ )
ORA-06512: in „SYS.FILE_PKG“, Zeile 27
ORA-06512: in „SYS.FILE_PKG“, Zeile 59
```

Listing 5: Der Zugriff auf Ressourcen (hier: Dateien) ist geschützt

```
begin
  dbms_java.grant_permission(
    grantee           => ‚SCOTT‘,
    permission_type   => ‚SYS:java.io.FilePermission‘,
    permission_name   => ‚/‘,
    permission_action => ‚read‘
  );
end;
```

Listing 6: Das Lese-Privileg auf Verzeichnis „/“ an SCOTT vergeben

```
:
Connection con = null;
con = DriverManager.getConnection(„jdbc:default:connection:“);

stmt = con.createStatement(“select * from emp“);
:
```

Listing 7: JDBC-Code in einer Java Stored Procedure

Analog dazu entfernt „dropjava“ die Java-Klassen oder -Bibliotheken aus dem Datenbank-Schema.

Abhängigkeiten auflösen: Der Resolver

Verwendet ein Java-Programmierer außerhalb der Datenbank eine Bibliothek, die nicht zum Standard-Sprachumfang gehört, stellt er sie im Dateisystem bereit und bindet sie in die Umgebungsvariable „CLASSPATH“ ein. Diese funktioniert wie die bekannte Umgebungsvariable „PATH“: Die JVM sucht Java-Bibliotheken in allen darin enthaltenen Dateien.

Die Datenbank-JVM bietet dagegen kein CLASSPATH, da sich die Java-Klassen und -Bibliotheken nicht im Dateisystem, sondern in der Datenbank befinden. Zum Auflösen von Abhängigkeiten wird daher der sogenannte „Resolver“ verwendet. Standardmäßig sucht dieser zunächst im eigenen Datenbank-Schema und danach nach öffentlichen Java-Klassen (PUBLIC). Es kann pro Java-Klasse eine eigene Resolver-Spezifikation angegeben und die Datenbank damit zur Suche auch in anderen Datenbank-Schemata angewiesen werden. Grundsätzlich versucht die Datenbank, alle Abhängigkeiten schon beim Laden – in jedem Fall aber vor Ausführung des Java-Codes – aufzulösen. Erst bei Erfolg wird die Java-Klasse als „VALID“ markiert und somit nutzbar. Lässt sich irgendeine Abhängigkeit nicht auflösen, bleibt der Java-Code „INVALID“ – es sei denn, man verwendet die loadjava-Parameter „force“ und „genmissing“:

```
$ loadjava -user scott/tiger
-o -r -v -force -genmissing
javalib.jar
```

In letzterem Fall ist der Code trotz nicht aufgelöster Abhängigkeiten „VALID“ und kann aufgerufen werden. Fehlermeldungen erscheinen erst dann, wenn die Ausführung des Java-Programms zur Laufzeit auf eine nicht aufgelöste Abhängigkeit stößt. Ab der Datenbank 11*g* bietet das Kommandozeilen-Werkzeug „ojvmtc“ Hilfestellung beim Auflösen von Abhängigkeiten.

Wie die Datenbank-JVM Ressourcen schützt

Der Zugriff auf Betriebssystem-Ressourcen wie Dateisystem oder Netzwerk erfordert in der Datenbank natürlich ein Berechtigungskonzept. Oracle hat dazu eine auf die Datenbank abgestimmte Variante der „Java2 Security“ implementiert. Ein Security-Manager ist standardmäßig aktiv und lässt sich auch nicht abschalten. Mit dem Privileg zum Ausführen einer Java Stored Procedure hat man nicht automatisch das Privileg zum Zugriff auf Ressourcen – diese sind separat geschützt. Listing 5 zeigt den Versuch, ein Verzeichnis im Dateisystem ohne entsprechende Privilegien auszulesen. Praktisch ist, dass der PL/SQL-Aufruf zum Einräumen der Privilegien gleich mitgeliefert wird. Sobald der DBA das Privileg gewährt hat (siehe Listing 6), kann SCOTT das Verzeichnis (und nur dieses) lesen.

Die JVM in der Datenbank regelt den Zugriff auf das Dateisystem im Gegensatz zu PL/SQL feingranular. Privilegien können für einzelne Dateien vergeben werden – mit „DBMS_JAVA.RESTRICT_PERMISSION“ sind darüber hinaus auch „negative“ Privilegien möglich: So kann man den Zugriff auf ein Verzeichnis freigeben, eine bestimmte Datei jedoch davon ausnehmen. Das gilt analog auch für Netzwerk-Privilegien.

Tabellenzugriffe: JDBC

Selbstverständlich kann man mit Java in der Datenbank auch auf Tabellen zugreifen; auch dies funktioniert genauso wie außerhalb der Datenbank mit JDBC. Da eine Java Stored Procedure sich bereits in der Datenbank befindet, muss man die JDBC-Verbindung nicht mehr eigens aufbauen. Das macht sich im Code dadurch bemerkbar, dass ein besonderer JDBC-Treiber (jdbc:default:connection:) zum Einsatz kommt. Beim Aufbau der Datenbank-Verbindung (siehe Listing 7) werden weder Nutzername noch Passwort mitgegeben. Ein Login findet nicht statt, da sich die Java Stored Procedure bereits in einer Datenbank-Sitzung befindet. Beim Aufruf von „getConnection“ kann man Username und Passwort zwar angeben, sie werden aber ignoriert. Der JDBC-Code in einer Java Stored Procedure ist ansonsten gleich wie

in einem Java-Programm außerhalb der Datenbank. Für datenzentrische Aufgaben ist PL/SQL allerdings nach wie vor die geeignetere Programmiersprache. Die Java-Version und damit der Funktionsumfang sind durch das Datenbank-Release festgelegt: So wird in der Datenbank 9*i* Java 1.3 verwendet, 10*g* enthält Java 1.4 und 11*g* ist in Release 2 mit Java 1.5 ausgestattet. Die JVM ist nicht austauschbar. Ein wichtiges Feature in 11*g* ist der neue Just-In-Time-Compiler, der die Ausführung der Java Stored Procedures massiv beschleunigt. Die in früheren Versionen vorhandene Option zum nativen Kompilieren der Java-Klassen mit dem Werkzeug „ncomp" wird damit überflüssig. Multithreading wird nicht unterstützt: Java-Programme, die dies verwenden, laufen ohne Fehler, es werden jedoch alle Threads serialisiert. Zur Parallelisierung von Java Stored Procedures steht wie für PL/SQL das Job-Management mit dem Datenbank-Scheduler (DBMS_SCHEDULER) zur Verfügung. Grafische Benutzeroberflächen (AWT oder Swing) stehen in Java Stored Procedures nicht zur Verfügung. Ab der Datenbank-Version 10*g* sind allerdings „Headless AWT" und damit unter anderem Funktionen zur einfachen Bildbearbeitung nutzbar.

In Aktion: Java in der Datenbank

Die deutschsprachige Apex-Community [1] bietet zahlreiche Beispiele zur Nutzung von Java in der Datenbank. Da Java Stored Procedures mit dem beschriebenen PL/SQL-Wrapper nach außen wie PL/SQL-Prozeduren oder -Pakete aussehen, ist die Nutzung in Application Express völlig transparent möglich:

1. *FTP-Client in Apex beziehungsweise PL/SQL: http://www.oracle.com/global/de/community/tipps/ftpclient/index.html*
   Der Tipp beschreibt, wie man BLOBs mithilfe der Java-Klasse „java.net.URLConnection" als Dateien auf einen FTP-Server hoch- beziehungsweise von diesem herunterladen kann. Da das Netzwerk vom Java Security Manager geschützt wird, muss der DBA hier entsprechende Privilegien vergeben. Welche das sind, ist im Tipp beschrieben.

2. *ZIP-Archive in Apex beziehungsweise PL/SQL ein- oder auspacken: http://www.oracle.com/global/de/community/tipps/zip/index.html*
   Java ist von Haus aus in der Lage, ZIP-Archive ein- und auszupacken. Das Package „java.util.zip" enthält die nötige Funktionsbibliothek. Der Community-Tipp beschreibt, wie die Funktion in einem PL/SQL-Package bereitgestellt wird, so dass ein ZIP-Archiv (als BLOB) in mehrere BLOBs ausgepackt werden kann. Auf der anderen Seite kann man mehrere BLOBs (beispielsweise in einer Tabelle) in ein ZIP-Archiv verpacken.

3. *Betriebssystem-Zugriffe: http://www.oracle.com/global/de/community/tipps/filesystem/index.html*

iJUG Verbund — Mitglieder stellen sich vor

Java User Group Deutschland e.V.

Die Java User Group Deutschland e.V. (JUG) wurde 1998 von Java-Aktiven aus der Sun User Group (SUG) gegründet. In den Jahren von 1994 bis 1998 war die Gruppe schon innerhalb der SUG aktiv. Nachdem sich Java immer mehr auch ausserhalb von sun microsystems entwickelte, erschien ein eigener Verein sinnvoll. Der Verein versteht sich als Forum für den Austausch von Erfahrungen und als Kontaktadresse für Java-interessierte im deutschsprachigen Raum. Er betreibt den Webserver java.de und veranstaltet zusammen mit der SUG die Source Talk Tage in Göttingen.

In den Anfängen des Vereins standen die Java-Stammtische verteilt auf die Republik noch in Kontakt mit der JUG. Nach etlichen Auflösungen und Neugründungen gibt es jetzt eine Vielzahl von eigenständigen lokalen Gruppen, die sich vor kurzem zur iJUG zusammengeschlossen haben. Die JUG ist dabei mit ihrem Stammtisch und Ihrer Jahrestagung in Göttingen mit anderen regionalen Gruppen vergleichbar. Lediglich der Webserver java.de hat eine überregionale Bedeutung und wird auch von Mitgliedern aus ganz Deutschland betreut und genutzt.

Vielen Java-Interessierten ist der Server java.de bekannt. Diente er am Anfang noch der Softwareverteilung, stehen heute die Diskussion im Forum und die Verteilung der Termine der regionalen Java-Gruppen im Vordergrund. Gerade Neulinge verirren sich gerne – bedingt durch den Namen – auf diesen Webserver. Hier werden die ersten Fragen zum Thema „Java" gestellt und Kontakt zu Gleichgesinnten in der eigenen Region gesucht. Wer nicht das Glück hat, eine aktive User Gruppe in der Nachbarschaft zu haben, bleibt über das Forum im Kontakt mit der Community. Nicht zu unterschätzen ist auch die Rubrik „Jobangebote" im Forum.

Weitere Informationen: http://www.java.de

27.08.2010
JUG Deutschland
XML-Parsermodelle
kontakt@java.de

31.08.2010 – 02.09.2010
Source Talk Tage 2010

09/2010
JUG Saxony
Semantic Web und Java
Kristian Rink
kristian@jugsaxony.de

02.09.2010
JUG-Ostfalen
Open Source Reporting mit Eclipse BIRT
Uwe Sauerbrei
info@jug-ostfalen.de

08.09.2010
JUG-Ostfalen
Winds Of Change –
Datenbankänderungen verwalten
und testen
Uwe Sauerbrei
info@jug-ostfalen.de

09.09.2010
DOAG SIG Java
Java Enterprise Edition 6 (JEE 6)
und die darin enthaltenen Technologien
Andreas Badelt
sig-java@doag.org

13.09.2010
JUG München
20 Gründe, warum man keine
Architekten braucht
Andreas Haug
ah@jugm.de

14.09.2010 – 15.09.2010
JUG Stuttgart
JazzConf – die erste Konferenz zu Jazz
Dr. Michael Paus
mp@jugs.org

16.9.2010
JUG Erlangen-Nürnberg
Anwendung von Vorgehensmodellen
in MDA und MDG
Oliver Szymanski
oliver.szymanski@source-knights.com

22.09.2010
JUG-Ostfalen
(Lineas) – Theorien, Regeln und
Kategorien. JUnit – experimentelle Features
Uwe Sauerbrei
info@jug-ostfalen.de

24.09.2010
Java User Group Köln (JUGC)
Stammtisch

10/2010
JUG München
Business Applikationen schnell
entwickeln – JVx Framework Live!
Andreas Haug
ah@jugm.de

11/2010
JUG München
Android
Andreas Haug
ah@jugm.de

16.11.2010 - 18.11.2010
DOAG 2010
Konferenz + Ausstellung
office@doag.org

Das Eingangsbeispiel dieses Artikels, der Zugriff auf Dateien oder das Ausführen von Betriebssystem-Kommandos, ist ebenfalls Standardfunktionalität der Java-Engine. Der Tipp zeigt, wie man mithilfe des Java- und PL/SQL-Packages von plsqlexecoscomm.sourceforge.net [3] die PL/SQL-Pakete „FILE_PKG“ und „OS_COMMAND“ sowie den Typ „FILE_TYPE“ installiert und zum einfachen Zugriff auf das Dateisystem nutzt.

4. *E-Mail aus einem Postfach abrufen: http://plsqlmailclient.sourceforge.net/*
   Mit diesem ebenfalls frei verfügbaren Java- und PL/SQL-Paket können eine Verbindung zu einem Mailserver (POP3 oder IMAP) geöffnet sowie E-Mails abgerufen, gelöscht oder verschoben werden. Die Funktionen stehen im PL/SQL-Paket MAIL_CLIENT bereit.

Fazit

Nutzt man die Java-Umgebung in der Datenbank, so sind die Möglichkeiten, die man in Stored Procedures hat, nahezu unbegrenzt. Java gleicht in PL/SQL fehlende Funktionalitäten perfekt aus, die unzählbaren, frei verfügbaren Java-Pakete der Open-Source-Community runden diese Möglichkeiten ab.

Weitere Informationen

[1] Deutschsprachige Apex Community: http://www.oracle.com/global/de/community/index.html
[2] Blog des Autors zu SQL und PL/SQL: http://sql-plsql-de.blogspot.com
[3] Betriebssystem-Zugriffe in PL/SQL und SQL: http://plsqlexecoscomm.sourceforge.net
[4] PL/SQL E-Mail-Client: http://plsqlmailclient.sourceforge.net
[5] Oracle Database Java Developers' Guide: http://download.oracle.com/docs/cd/E11882_01/java.112/e10588/toc.htm

Kontakt:
Carsten Czarski
carsten.czarski@oracle.com

Java aktuell – Q1 / Dezember 2010

Die nächste Ausgabe der Java aktuell erscheint am 1. Dezember 2010.
Bitte schreiben Sie uns an redaktion@ijug.eu, was Ihnen an diesem Heft gefallen hat und welche Inhalte Sie sich für die künftigen Ausgaben wünschen.
Falls Sie selbst einen Artikel in der nächsten Java aktuell veröffentlichen möchten, schicken Sie bitte vorab Ihren Themenvorschlag an redaktion@ijug.eu, Redaktionsschluss für den fertigen Text ist am 30. September 2010.

Carsten Czarski ist seit 2001 bei der ORACLE Deutschland B.V. & Co. KG in der Business Unit Database. Schwerpunkt ist die Unterstützung und Beratung von Kunden und Partnern bei der Entwicklung datenbankgestützter Anwendungen.