Nr. 03 | Herbst 2012 | www.ijug.eu

Java aktuell

Das Magazin der Java-Community

Java aktuell

Java in Höchstform

Java EE 6:
GlassFish, JBoss und Geronimo, Seite 11

Android:
Wissenschaftliche Applikationen der nächsten Generation entwickeln, Seite 38

Jnect:
Bewegungen in Java erkennen, Seite 59

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

4 191978 304903 03

iJUG Verbund

Wolfgang Taschner
Chefredakteur Java aktuell

Java ist in Höchstform!

Die Artikel in dieser Ausgabe spiegeln zahlreiche Segmente der umfangreichen Java-Landschaft wider. Der Reigen beginnt mit einem Vergleich der drei bekanntesten Applikationsserver GlassFish, JBoss und Geronimo. Über den Tellerrand hinaus blicken wir beim WebLogic Server im Zusammenspiel mit Oracle Real Application Clusters. Das Eclipse Modeling Framework hingegen hilft dem Entwickler beim Modellieren. VisualVM wiederum bietet ein reiches Toolset für die Performance-Analyse von Java-Anwendungen.

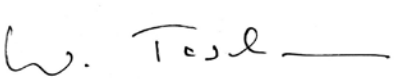
Sehr interessant für Java-Entwickler ist auch Apache Camel, ein mächtiges Open-Source-Integrations-Framework. Nicht zu vergessen das Thema „Android", mit dem sich gleich zwei Artikel beschäftigen. Jonas Feldt und Johannes M. Dieterich von der Georg-August-Universität Göttingen demonstrieren die Entwicklung einer modernen App, während Andreas Flügge die Reihe mit den Android-Grundlagen abschließt. Auch das Testen kommt in dieser Ausgabe nicht zu kurz: Oliver Böhm fasst zehn Jahre Pattern-Testing zusammen. Cloud Computing findet diesmal in einem Artikel über die Plattform „Spring Cloud" statt. Der Windows Azure Service Bus hingegen zeigt, dass auch Microsoft etwas für Java-Entwickler zu bieten hat, und mit dem Jnect-Framework lassen sich Bewegungen und Sprache erkennen und weiterverarbeiten.

Das Interview auf Seite 9 stellt die Java-Vision des Software-Riesen SAP vor. Mit Oracle, IBM und Red Hat haben sich damit in den letzten Ausgaben große Keyplayer unter den Java-Aktivisten bei uns geäußert. In einem weiteren Interview kommt unser iJUG-Mitglied, die Swiss Oracle User Group (SOUG), zu Wort.

Neu ab dieser Ausgabe ist auch ein Redaktionsbeirat, der die Redaktion dabei unterstützt, die Qualität der Artikel zu sichern, neue Autoren zu gewinnen und interessante Themen zu finden. Herzlich willkommen Ronny Kröhne, Daniel van Ross und Dr. Jens Trapp. Ein Porträt der drei finden Sie auf Seite 42.

Bleibt nur noch, Sie auf drei Veranstaltungen hinzuweisen, die demnächst stattfinden: das Java Forum Stuttgart, die Source Talk Tage in Göttingen und die DOAG 2012 Development in Bonn.

Ihr



Java Forum Stuttgart

JUGS
java user group stuttgart

Die Java User Group Stuttgart e.V. veranstaltet am 5. Juli 2012 im Kultur- & Kongresszentrum Liederhalle (KKL) in Stuttgart wieder das Java Forum Stuttgart. Wie im Vorjahr werden rund 1.200 Teilnehmer erwartet. Geplant sind 42 Vorträge in sechs parallelen Tracks. Zudem werden bis zu 35 Aussteller vor Ort sein, darunter auch der Interessenverbund der Java User Groups e.V. (iJUG). An der Community-Wand stehen sowohl offene White-Boards als auch BoF-Boards (Bird-of-a-Feather). Abends gibt es die Gelegenheit, sich bei verschiedenen BoF-Sessions mit Gleichgesinnten zu treffen, um über ein bestimmtes Thema zu diskutieren und sich auszutauschen. Darüber hinaus wird es wieder eine Jobbörse/Karriereecke für die Besucher geben.

Workshop „Java für Entscheider“
Die eintägige Überblicksveranstaltung am Vortag (4. Juli 2012) zeigt Begrifflichkeiten und wichtige Technologien aus der seit Jahren in der Industrie etablierten Plattform Java. Ausgehend von strategischen Gesichtspunkten wie Bedeutung und Verbreitung reicht der Blick über das Client-seitige Java (Java SE) und die wesentlichen Entwicklungswerkzeuge bis zum Server-seitigen Java (Java EE). Dort stehen dann die Bedeutung von Java als Integrationsplattform und die verschiedenen Technologien im Mittelpunkt. Weiterhin wird noch der Einsatz von Java in den immer wichtiger werdenden mobilen Lösungen (Android, iOS) gestreift. Abschließend kommen noch das Ausrollen von Java-Lösungen und das sehr interessante Eclipse als Rich-Client zur Sprache, um dann den Bogen von der Software-Entwicklung hin zum Betrieb zu schlagen.

Experten-Forum Stuttgart
Am 6. Juli 2012 findet wieder im Anschluss an das Java Forum Stuttgart ein Experten-Forum Stuttgart statt. Auf dem Programm stehen zwölf halbtägige Workshops in sechs parallelen Tracks. Die Workshops in kleinen Gruppen mit maximal 25 Teilnehmern ermöglichen einen intensiven Austausch zwischen Trainer und Zuhörern.
Anmeldung und weitere Informationen zum Java Forum Stuttgart unter **www.java-forum-stuttgart.de**

Das Java-Tagebuch

Andreas Badelt, Leiter der DOAG-SIG Java

Das Java-Tagebuch wurde in Ausgabe 2/2010 gestartet, um einen Überblick über die wichtigsten Geschehnisse rund um Java zu geben – in komprimierter Form und chronologisch geordnet. Der vorliegende Teil widmet sich den Ereignissen im ersten Quartal 2012.

13. Januar 2012

Java.net: JUG-Seiten werden aktualisiert

Michael Hüttermann, Organisator der JUG Cologne, hat eine Reihe von „open issues" auf den JUG-Seiten von java.net behoben (http://www.java.net/node/882694). Seit den durch Oracle initiierten Infrastruktur-Änderungen gab und gibt es eine ganze Reihe von fehlerhaften Links etc. Alle bekannten Probleme wurden in einer Liste zusammengefasst, und inzwischen ist ein großer Teil durch JUG-Aktive abgearbeitet. Eine gute Gelegenheit, mal (wieder) einen Blick auf die Seiten zu werfen und Tipps sowie Präsentationen von JUGs auf der ganzen Welt anzuschauen oder gleich bei der Aktualisierung mit anzupacken: http://java.net/projects/jugs/pages/Home#Open_issues

25. Januar 2012

Die Überarbeitung des JCP wird fortgesetzt

Die Überarbeitung des Java Community Process (JCP) geht in die zweite Runde. Nach JSR-348 mit dem Fokus auf Transparenz und Beteiligung wurde heute der JSR-355 eingereicht, mit dem Ziel der Zusammenlegung der beiden Executive Committees (ME und SE/EE). Auch dieser JSR soll innerhalb von sechs Monaten abgeschlossen sein, sodass die Änderungen bereits bei den Wahlen im Herbst 2012 in Kraft treten können. Es könnte allerdings sein, dass die nötigen Anpassungen zeitlich gestreckt wirksam werden. Durch die vermutliche Halbierung der Sitze (zumindest annähernd – Oracle und IBM sitzen in beiden ECs und würden auch nur einen Sitz behalten) kommt es natürlich zu einem erheblichen Verdrängungsprozess. Das soll möglichst fair und wohl in mehreren Phasen geschehen, sodass die Mitglieder, deren Wahlperiode in diesem Jahr abläuft, nicht benachteiligt werden. Trotzdem dürfte eine erhebliche Unruhe auftreten. Mit der neu gewonnenen Transparenz im JPC lässt sich für „einfache" Mitglieder des JCP aber wohl besser entscheiden, wem sie ihre Stimme geben sollen. Das Verhältnis von ratifizierten (Vorschlagsrecht ausschließlich bei Oracle) zu offen gewählten Sitzen soll im Übrigen bei zwei zu eins bleiben: https://blogs.oracle.com/jcp/entry/another_jcp_next_jsr_submitted

Java User Group Leaders live vom Meeting der International Oracle Users Group Community

Das Interview mit den JUG-Leadern aus fünf Kontinenten auf https://blogs.oracle.com/javaspotlight/entry/java_spotlight_episode_66_java

26. Januar 2012

JCP.next in Aktion:
AT&T, Samsung und SK Telecom verlieren ihre Stimmrechte im Executive Committee

Im JSR-348 wurde definiert, dass Mitglieder eines Executive Committees (EC) ihre Stimmrechte verlieren, wenn sie an zwei oder mehreren aufeinanderfolgenden Treffen des EC nicht teilnehmen. Das hatte sich scheinbar noch nicht überall herumgesprochen: Genau zwei Treffen nach Inkrafttreten der neuen Regeln hat es schon drei Mitglieder erwischt. Um ihre Stimmrechte wiederzuerlangen, müssen sie wieder an zwei aufeinanderfolgenden Treffen teilnehmen. Andererseits könnten sie bei weiterem Nichterscheinen (fünf Mal hintereinander oder zwei Drittel aller Treffen innerhalb von zwölf Monaten) auch direkt aus dem EC ausgeschlossen werden. Zumindest haben sie sich schon mal für ein Ausscheiden „beworben" – im Herbst geht es ja erstmals darum, die ECs zu verkleinern. https://blogs.oracle.com/jcp/entry/jcp_ec_updates

30. Januar 2012

Ein Lebenszeichen vom JSR-308

Nach mehreren Jahren ist der JSR-308 (Annotations on Java Types) erneut zum „Early Draft Review" vorgelegt worden. Das erste Mal war dies bereits 2007 der Fall, die Review-Phase wurde jedoch nie abgeschlossen. Hinter den Kulissen wurde anscheinend trotzdem weitergearbeitet. Nachdem die geplante Aufnahme ins OpenJDK 7 nicht stattfand, soll es mit dem OpenJDK 8 auf jeden Fall klappen. Die „Type Annotations" ergänzen sich jedenfalls gut mit anderen neuen Sprach-Features wie „Lambda Expressions" und nach so vielen Jahren sollte die Technologie doch ausgereift sein …: http://types.cs.washington.edu/jsr308

2. Februar 2012

EJB 3.2 geht auch ins erste Review

Ebenfalls in die Early-Draft-Review-Phase – allerdings zum ersten Mal – geht der JSR-345 (EJB 3.2). Da dieses Release einer der Hauptbausteine für Java EE 7 sein soll, ist die Cloud-Unterstützung als wichtiges Thema genannt worden. Daneben soll die Herauslösung einzelner Services erwogen werden, um diese auch anderen Java-EE-Technologien zugänglich zu machen, sowie die bessere Integration mit anderen

JSRs (CDI, Bean Validation etc.) und die Erweiterung der Anwendung von Annotations. In den Draft-Dokumenten ist bislang aber noch nicht allzu viel davon zu sehen: http://java.net/projects/ejb-spec und http://jcp.org/en/jsr/detail?id=345

6. Februar 2012

java.net-Umfrage: Beruflicher Optimismus unter Java/JVM-Entwicklern

Eine gerade beendete Umfrage auf java.net zur Entwicklung von Stellenangeboten im Jahr 2012 hat einen deutlichen Optimismus unter den Teilnehmern gezeigt: 19 Prozent waren der Meinung, dass die Zahl der Angebote schnell steigen wird, 25 Prozent glauben an einen moderaten Anstieg und 23 Prozent an ein stabiles Angebot. Nur 9 Prozent rechnen mit einem schrumpfenden Markt, der Rest entfiel auf „I don't know" beziehungsweise „Other". Man kann natürlich darüber streiten, ob eine Umfrage, in der eine Skala mit zwei positiven, einer neutralen und nur einer negativen Antwort benutzt wird, eventuell ein bisschen suggestiv ist. Auch lässt die Tatsache, dass immerhin 7 Prozent beziehungsweise 25 Teilnehmer mit „Other" gestimmt haben, Fragen bezüglich der Qualifikation einiger Teilnehmer offen. Aber man kann trotzdem festhalten, dass Java/JVM-Entwickler scheinbar optimistisch ins Jahr 2012 gehen: http://blogs.java.net/blog/editor/archive/2012/02/06/poll-result-most-javajvm-developers-expect-profitable-2012-others-agree

8. Februar 2012

Java-Spotlight-Interview mit Patrick Curran

Patrick Curran, der Mann hinter dem Java Community Process, hat in einem aktuellen Java-Spotlight-Podcast einen Einblick in die aktuellen Pläne zum Umbau des JCP gegeben. Einige Details zum neuen JSR-355 wurden ja bereits im Januar genannt. Die anvisierte Größe des gemeinsamen Executive Committees liegt aber offensichtlich bei 25, sodass nach dem Streichen des zweiten Sitzes für IBM und Oracle nur noch Reduzierung um fünf Sitze nötig ist. Um dies etwas verträglicher zu gestalten, wird über eine Verkürzung der Wahlperiode von drei auf zwei Jahre nachgedacht. Ebenso wird diskutiert, die Wahlperiode flexibel zwischen ein bis drei Jahren zu halten, in Abhängigkeit von der Anzahl der Stimmen; das hieße, ein neu gewähltes EC-Mitglied, dem viel Skepsis in Form eines schlechten Wahlergebnisses entgegen schlug, müsste sich umso eher wieder einer Wahl stellen. Der dritte und nach aktueller Planung letzte JSR in dieser Reihe soll alle komplexen und kontroversen Änderungen angehen. Neben weiteren Modifikationen am JCP-Prozess soll dann auch das „Java Specification Agreement" (JSPA) überarbeitet werden, das alle JCP-Mitglieder mit Oracle schließen müssen und das in der Vergangenheit öfter zu hitzigen Diskussionen geführt hat – beispielsweise bei der Frage, ob Apache für ein „Test Compatibility Kit" für Harmony zahlen muss:

- https://blogs.oracle.com/javaspotlight/entry/java_spotlight_episode_68_patrick
- http://jcp.org/aboutJava/communityprocess/ec-public/materials/2012-03-06/JCP.next.3-March-2012.html

15. Februar 2012

Umfrage: Welche EE-Version und welche IDEs werden genutzt?

In einer Kolumne der „Software Development Times" werden Ergebnisse einer „Java & SOA Study" des Magazins veröffentlicht. 18 Prozent der Entwickler dürfen demnach weiterhin „J2EE" sagen, ohne korrigiert zu werden. So hieß die Enterprise Edition bis einschließlich Version 1.4. Große Projekte bewegen sich nun einmal langsam vorwärts, da spielt es mitunter auch keine Rolle, ob eine Java-Version noch unterstützt wird – auch wenn der Betrieb sicher nervös wird. Aber immerhin 45 Prozent setzen Java EE 5 ein, 54 Prozent sogar Java EE 6. Interessant ist auch die Verteilung bei den IDEs: Eclipse liegt mit 65 Prozent deutlich vorn, gefolgt von den Oracle-Produkten NetBeans mit immerhin 26 Prozent und JDeveloper mit 17 Prozent. Dahinter liegen IBMs auf Eclipse basierende IDEs WebSphere Studio Application Developer (13 Prozent) sowie der Nachfolger Rational Application Developer (12 Prozent). Apple XCode (11 Prozent), JetBrains IntelliJ (10 Prozent) und Genuitec MyEclipse (9 Prozent) werden ebenfalls noch namentlich gelistet. Insgesamt gibt es eine deutliche Dominanz der Eclipse(-basierten) IDEs: http://sdtimes.com/link/36362

20. Februar 2012

JSRs zu „Money and Currency" und „Executive Committee Merge" dürfen starten, „Constraint Programming API" ist final

Die JSRs 354 (Money and Currency API) und 355 (JCP Executive Committee Merge) sind jeweils ohne Gegenstimmen von den Executive Committees des JSP angenommen worden. Die Expert Groups dürfen nun die Arbeit aufnehmen. Ebenso einstimmig angenommen ist – nach gut zweieinhalb Jahren Arbeit – der finale Entwurf des „Constraint Programming API" (Referenz-Implementierung durch Open Rules). Die „Final Specification" dürfte damit in Kürze erscheinen.

23. Februar 2012

iJUG: Markus Eisele neues Mitglied in der Expert Group für Java EE 7

Ursprünglich motiviert durch die „adopt a JSR"-Initiative, der sich auch der iJUG verschrieben hat, ist Markus Eisele von der DOAG gleich Mitglied in der Expert Group für Java EE 7 geworden. Gratulation! Wir hoffen natürlich auf spannende Einblicke in die Arbeit der Expert Group und einen noch besseren Draht für den iJUG ins Zentrum des Java-Ökosystems: http://java.net/projects/javaee-spec/lists/jsr342-experts/archive/2012-02/message/54

2. März 2012

Java-Tutorials stehen jetzt auch als eBooks zur Verfügung

Eine hervorragende Idee, insbesondere für Entwickler, die viel unterwegs sind: Die Tutorials für Java SE 7 stehen jetzt auch als eBooks zur Verfügung (im epub- und mobi-Format). Sie können genau wie die HTML-Versionen vom Oracle Technology Network heruntergeladen werden. Hoffentlich wird diese Idee bald auch auf andere Dokumente und Spezifikationen ausgeweitet: http://www.oracle.com/tech-

network/java/javase/downloads/java-se-7-tutorial-2012-02-28-1536013.html

5. März 2012

„End of Life" für Java SE 6, JavaFX 1.2, and JavaFX 1.3

Oracle hat das Ende des kostenlosen Supports für Java SE 6 angekündigt. Mit betroffen sind auch die JavaFX-Versionen 1.2 und 1.3 und somit das in der Version 2.0 durch ein reines Java-API abgelöste JavaFX-Script. Letzteres hatte zwar im Projekt „Visage" von Stephen Chin ein neues Zuhause gefunden, das Projekt weist aber keine größeren sichtbaren Aktivitäten mehr auf. Damit dürfte dann auch JavaFX-Script bald ein Fall fürs Museum sein. Der Zeitpunkt folgt dem normalen Release-Plan und war sogar noch einmal von Juli auf November 2011 verschoben worden. Trotzdem sind nicht alle Entwickler und IT-Verantwortlichen glücklich darüber, da der Java SE 7 auch nach bald einem Jahr noch nicht so verbreitet genutzt wird. Für große Projekte ist auch ein Zeitfenster von etwas mehr als einem Jahr nicht üppig bemessen. Daher hat der iJUG in seinem Newsletter eine Umfrage zu diesem Thema gestartet: https://blogs.oracle.com/java/entry/eoling_java_se_6_javafx

7. März 2012

Adopt a JSR: Neue Präsentation

Auf java.net ist neuer Content zur „Adopt a JSR"-Initiative zu finden. Wie bereits in der letzten Ausgabe berichtet: Die Initiative hat zum Ziel, den Java Community Process und seine Specification Requests, also die Entwicklung der neuen Standards, enger mit der Community zu verzahnen. Sie wurde von der London Java Community ins Leben gerufen und richtet sich auch an JUGs und ihre Mitglieder. Dazu gibt es ein Wiki, in dem inzwischen recht detailliert steht, wie Interessenten sich bei „Adopt a JSR" einbringen können und welche JUGs sich bereits wo engagieren: http://java.net/projects/jugs/pages/AdoptAJSR. Neu dazugekommen ist jetzt eine PowerPoint-Präsentation, die sich besser nutzen lässt, um bei JUG-Treffen die Initiative vorzustellen: http://java.net/projects/jugs/downloads/directory/Adopt%20a%20JSR

8. März 2012

WebSocket-JSR angenommen – unter Vorbehalt!

Der JSR 356 (Java API for WebSocket) ist vom Executive Committee für SE/EE angenommen worden, die Expert Group darf also starten. Allerdings gab es im Vorfeld heftige Diskussionen darüber, dass die Lizenzen für das Test Compatibility Kit (TCK) Gebühren in Höhe von 35.000 Dollar jährlich vorsehen – wenn auch nur für die kommerzielle Nutzung. Die Befürchtung ist, dass dies insbesondere Open-Source-Implementierungen benachteiligt, da diese nicht unbedingt unter die „not-for-profit"-Klausel fallen. In den Kommentaren zur Abstimmung ist aber zu lesen, dass der Specification Lead (Danny Coward für Oracle) Kompromiss-Bereitschaft signalisiert hat, und außer Google haben daher alle anderen Mitglieder für den JSR gestimmt oder sich zumindest enthalten; viele jedoch unter dem Vorbehalt, dass sie einer Finalisierung des JSR nur zustimmen werden, wenn dieser strittige Punkt gelöst ist. Bislang gibt es noch keine Unterstützer im Rahmen der „Adopt a JSR"-Initiative. Freiwillige vor …: http://java.net/projects/jugs/pages/AdoptAJSR

12. März 2012

Java EE 6 Galleria

Markus Eisele berichtet in seinem Blog von einem sehr interessanten Projekt: die Java EE 6 Galleria von Vineet Reynolds. Ziel ist es, ein umfassendes und praxistaugliches Beispiel für den Einsatz von Java EE 6 mit JSF 2.0 und JPA 2.0 zu geben – inklusive Unit- und Integration-Tests (mit JUnit 4 und Arquillian). Ein Blick lohnt sich: https://bitbucket.org/VineetReynolds/java-ee-6-galleria

20. März 2012

„Social Media API" abgelehnt

Der JSR 357 (Social Media API) ist vom Executive Committee für SE/EE abgelehnt worden. Die Gründe, die von den EC-Mitgliedern genannt werden, sind unter anderem: Es sei generell noch zu früh für eine Standardisierung in diesem Bereich; der Umfang des JSR sei zu groß und man solle sich lieber erst mal auf Teilaspekte konzentrieren, bei denen eine Standardisierung bereits sinnvoll ist; der JSR müsse von beiden Executive Committees, also auch dem für Java ME, koordiniert beziehungsweise erst nach dem Merge der beide ECs gestartet werden. Markus Eisele argumentiert in seinem Blog heftig gegen diese Entscheidung und führt ins Feld, dass Standardisierung Innovation verstärken kann und der JCP kein reines Abnickgremium für Quasi-Standards sein soll: http://blog.eisele.net/2012/03/why-i-think-ec-is-wrong-about-jsr-357.html. Das scheint sich nicht mit den Meinungen vieler anderer JCP-Aktiver wie Ed Burns (Specification Lead für JSF) zu decken, die sagen: „Innovation is done elsewhere". Wir dürfen gespannt sein, ob sich hier eine breite Grundsatz-Diskussion entwickelt: http://jcp.org/en/jsr/results?id=5317

Projekt Ceylon: Zweiter Meilenstein freigegeben

Der zweite Meilenstein für Ceylon ist freigegeben worden. Schwerpunkt war die Interoperabilität mit Java. Fast alle Sprachelemente von Java sind jetzt im Umfang von Ceylon enthalten und darüber hinaus einiges, das Java nicht „kennt" (etwa Methoden höherer Ordnung oder statisch typisierte NULL-Werte). Das Thema „File-I/O" wurde offensichtlich auf den dritten Meilenstein verschoben. Insgesamt sollen es fünf sein bis zum Release 1.0: http://www.ceylon-lang.org/blog/2012/03/20/ceylon-m2-minitel/?utm_source=download&utm_medium=web&utm_content=blog&utm_campaign=1_0_M2release

27. März 2012

Rave wird Apache Top-Level-Projekt

À propos „Social Media": Apache Rave ist aus dem Inkubator entlassen und jetzt ein Top-Level-Projekt der Software Foundation. Rave ist in eigenen Worten eine „Open Source Mashup Platform", die das Erstellen von und Integrieren mit „Social Network"-Technologien erleichtern soll. Vielleicht wächst hier ja etwas heran, das einem neuen Anlauf für ein standardisiertes „Social Media API" nützt: https://blogs.

apache.org/foundation/entry/the_apache_software_foundation_announces24

2. April 2012

JavaFX „Community Rock Stars" wechseln zu Oracle
Jim Weaver und Stephen Chin, zwei prominente Namen im Java- FX-Umfeld, wechseln zu Oracle. Nur Amerikaner können es so schön ausdrücken: „We're proud to have both Jim Weaver and Stephen Chin joining Oracle's Java Evangelist Team." Der Konzern demonstriert damit einmal mehr, dass es ihm wirklich ernst ist mit Java FX: https://blogs.oracle.com/java/entry/two_javafx_community_rock_stars

3. April 2012

Trojaner gefährden insbesondere Mac-OS-X-Systeme
Innerhalb der letzten Wochen hat der Heise News-Ticker mehrere Meldungen zu einem Trojaner namens „Flashback" und anderer, ähnlich gearteter Schad-Software gebracht, die bekannte Lücken in der JVM ausnutzt. Die Software bricht dadurch aus der Sandbox aus, um anschließend weitere Programmteile nachzuladen und zum Beispiel Passwort-Eingaben auszuspähen und zu übertragen. Oracle hat die ersten dieser Lücken bereits im Februar für Java für die Windows-Version gepatcht. Mac-User müssen allerdings seit November auf ein Update von Apple warten, sodass sich die Schädlinge hier besonders ungehindert verbreiten können. Daher wird Apple-Nutzern geraten, Java zu deaktivieren, bis ein Patch verfügbar ist. Auch Mozilla hat reagiert – die Firefox-Browser blockieren veraltete Java-Plug-ins, um das Risiko zu minimieren: http://www.heise.de/mac-and-i/meldung/Flashback-Trojaner-nutzt-ungepatchte-Java-Luecke-aus-1499277.html https://addons.mozilla.org/en-US/firefox/blocked/p80

5. April 2012

JCP: Weitere Details zum Executive-Committee-Merge
Der JSR 355 (Zusammenlegung der Executive Committees für SE/EE und ME) hat vor einigen Tagen ein „Early Draft Review" veröffentlicht. JCP-Lead Patrick Curran erklärt Details in seinem Blog: Bei den EC-Wahlen im Oktober 2012 werden neue Mitglieder nur für ein Jahr gewählt. Unmittelbar danach werden die beiden ECs zusammengelegt, sodass nach Wegfall des jeweils zweiten doppelten Sitzes von Oracle und IBM zunächst ein Executive Committee mit 30 Mitgliedern entsteht. Bei den Wahlen im Oktober 2013 werden drei ratifizierte und zwei frei wählbare Sitze gestrichen, sodass das EC nur noch 25 Mitglieder hat (16 ratifizierte Sitze, 8 frei wählbare, plus Oracle). Bei diesen Wahlen werden alle Sitze neu besetzt (der permanente Sitz von Oracle ist hier sicher ausgenommen). Anschließend soll dann jedes Jahr die Hälfte der Sitze neu vergeben werden: https://blogs.oracle.com/pcurran/entry/merging_the_executive_committees

10. April 2012

Tiobe-Index: C ist zurück an der Spitze
Eine gute Nachricht für alle, die sich seit Jahrzehnten mit der Programmiersprache C beschäftigen: Sie sind wieder ganz oben auf der Hype-Welle: Laut Tiobe-Index im April hat C (17,6 Prozent) erstmals seit zwei Jahren wieder Java (17 Prozent) an der Spitze abgelöst. Die beiden Sprachen liefern sich seit 2009 ein Kopf-an-Kopf-Rennen in diesem Index, nachdem über mehrere Jahre der Anteil von Java deutlich zurückgegangen war. Deutlich dahinter liegen C++, Objective-C und C#, keine dieser Sprachen kommt über neun Prozent. Wie aber schon in der letzten Ausgabe berichtet, muss man das Zahlenwerk des Tiobe-Index nicht unbedingt zur Grundlage seines Handelns machen. Letztlich wird hier nur die Treffermenge zu Suchbegriffen im Internet ausgewertet. Kein Java-Entwickler sollte also das Handtuch werfen und im Buchversand seiner Wahl nach Kernighan und Ritchie suchen – es sei denn, er oder sie will es zur Abwechslung mal mit Linux-Kernel-Entwicklung versuchen. Ach ja: Scala ist diesmal doch unter den Top 50, aber weiterhin nur mit gut 0,2 Prozent Anteil: http://www.heise.de/open/meldung/TIOBE-Sprachindex-C-loest-Java-an-Spitze-ab-1517708.html

Auch Twitter verliert Stimmrechte im EC
Nach AT&T, SK Telecom und Samsung hat es jetzt auch Twitter erwischt. Der Executive-Committee-Neuling verliert wegen mehrfacher Nichtteilnahme an Treffen des EC vorübergehend die Stimmrechte: https://blogs.oracle.com/jcp/entry/transparency_call_for_spec_leads

Test-Plattform Arquillian: Erstes stabiles Release herausgegeben!
Die von JBoss initiierte Test-Plattform Arquillian hat das erste stabile Release (1.0.0.Final) erreicht. Arquillian hat zum Ziel, funktionale und Integrations-Tests deutlich zu vereinfachen. Tests sollen im Container ablaufen (nicht in einem speziellen „Embedded Container", sondern wie beispielsweise schon bei Cactus in einem „richtigen" Container). Die Steuerungsaufgaben soll das Framework übernehmen, dazu gibt es Adapter für die verschiedensten Container, die allerdings, natürlich abgesehen von JBoss AS 7, noch nicht final sind. Da es aber ein Open-Source-Projekt ist, können von allen Interessierten Korrekturen vorgenommen oder neue Adapter für den Container der Wahl hinzugefügt werden. Mit Arquillian wird nicht das Rad neu erfunden, sondern es benutzt bestehende Frameworks wie JUnit, TestNG, Selenium und WebDriver oder integriert sich in diese: http://arquillian.org/blog/2012/04/10/arquillian-first-stable-release

11. April 2012

JCP: Spec Leads verzweifelt gesucht
Der JCP sucht nach neuen Specification Leads für eine Reihe inaktiver JSRs, zum Beispiel die JSRs 241 (Groovy) und 304 (Mobile Telephony API). Die Bewerbungsfrist lief allerdings schon am 23. April 2012 aus: https://blogs.oracle.com/jcp/entry/new_jsr_stage_inactive_jsrs

Andreas Badelt ist Technology Architect bei Infosys Limited. Daneben ist er seit 2001 ehrenamtlich als Leiter der SIG Java der DOAG Deutsche ORACLE-Anwendergruppe e.V. aktiv.

„Die Java-Community ist riesig …“

Für Java aktuell ist Chefredakteur Wolfgang Taschner im Gespräch mit Harald Müller, Chief Product Owner für die Java Platform bei SAP.

Welchen Stellenwert hat Java bei SAP?
Aus Sicht von SAP ist Java eine sehr wichtige Technologie, die eine zentrale Stelle in unserem Technologie-Stack einnimmt. Große Teile des SAP-Technologie-Portfolios – insbesondere große Teile der SAP-NetWeaver-Plattform – basieren auf Java. Auch als Entwicklungssprache für Anwendungen werden wir weiterhin ABAP und Java, jeweils entsprechend ihren spezifischen Stärken, verwenden und unterstützen.

Welche Ziele hat SAP mit Java?
Wie gesagt, bei SAP setzen wir jede Technologie entsprechend ihren Stärken ein. Im Fall von Java sind dies zum Beispiel Leichtgewichtigkeit, Performance und vor allem die weite Verbreitung von Java und damit verbunden das globale Vorhandensein von Java-Know-how. Lassen Sie mich das am Beispiel von NetWeaver Neo, SAPs neuer Cloud-Plattform für Java-basierte Anwendungen, illustrieren. Die Gründe für Java liegen hier auf der Hand. Das Java-Modulsystem OSGi mit Eclipse Equinox als Referenz-Implementierung ermöglicht Leichtgewichtigkeit, Modularität und Performance – Grundvoraussetzungen für eine dynamische und skalierbare Cloud-Umgebung. Das auf offenen Standards basierende, breite Spektrum von Open-Source-Java-Implementierungen für praktisch alle benötigten Technologie-Bereiche macht es möglich, die Entwicklung auf das Wesentliche zu konzentrieren, statt das Rad immer wieder neu erfinden zu müssen. Die große Verbreitung von Java und standardbasierten APIs, frei zugängliche Informationen und das weit verbreitete Java-Know-how machen es Partnern und Kunden leicht, auf einer Plattform wie Neo eigene Anwendungen zu entwickeln. Das große Interesse für NetWeaver Neo bestärkt uns in dieser Einschätzung.

Wie beurteilen Sie die Lage im Enterprise-Java-Markt?
Die Java-Community ist riesig und Java als Technologie im Enterprise-Markt sehr stark verbreitet. Auch wenn immer wieder neue Sprachen und Technologien auftauchen und sich verbreiten, wird Java als Basis-Technologie auf lange Zeit nicht aus dem Markt wegzudenken sein. Wir erwarten also, dass der Enterprise-Java-Markt weiter wachsen wird – insbesondere in der Cloud sehen wir hier noch viel Potenzial.

Harald Müller, Chief Product Owner für die Java Platform bei SAP

Wie ist das Verhältnis zwischen SAP und Oracle im Java-Bereich?
SAP und Oracle sind in Bezug auf Java Geschäfts- und Entwicklungspartner und haben in dieser Hinsicht und in der Community ein stabiles und gutes Verhältnis. Das manifestiert sich unter anderem in der gemeinsamen Arbeit innerhalb von OpenJDK. Aber auch in anderen Bereichen wie im Eclipse-Java-Projekt Gemini arbeiten Software-Ingenieure von SAP und Oracle im Java-Umfeld zusammen.

Was erwarten Sie von Oracle in Bezug auf Java?
Wir erwarten und wünschen uns, dass Oracle die Offenheit von Java weiter bewahrt und fördert und damit ermöglicht, dass Java eine attraktive Technologie sowohl für die lebhafte und große Java-Community als auch für Unternehmen bleibt.

Welche Chancen sehen Sie für Android in den kommenden Jahren?
Android ist neben iOS und Windows Mobile eine der wichtigsten Plattformen, auf die wir in unserer Strategie für mobile Anwendungen setzen.

Was erwartet SAP in Zukunft von OpenJDK?
Innovation in Java passiert im OpenJDK-Umfeld. Deshalb fiel im letzten Jahr bei SAP die Entscheidung, sich aktiv am OpenJDK-Projekt zu beteiligen, um so Einfluss auf die Entwicklung der Java-Technologie zu gewinnen. SAP selbst implementiert seit einigen Jahren bereits eine eigene Java Virtual Machine, die sogenannte „SAP JVM“, auf Basis einer kommerziell lizenzierten Version von OpenJDK. Das heißt SAP, unsere Partner und unsere Kunden profitieren direkt von Innovationen aus dem OpenJDK-Umfeld. Darüber hinaus bietet OpenJDK eine Plattform, um Kooperationen und Zusammenarbeit zwischen Firmen und Entwicklern im Java-Umfeld zu ermöglichen. Aktive Unterstützer des OpenJDK-Projekts sind unter anderem IBM, Apple und Red Hat. Ein Beispiel ist die erst kürzlich veröffentlichte Ankündigung der Zusammenarbeit zwischen SAP und IBM. Wir werden mit IBM an einer Portierung von OpenJDK auf AIX und „Linux on PowerPC“ arbeiten.

Wie interpretieren Sie die Entscheidung von Oracle, OpenJDK als Referenz-Implementierung für Java festzulegen?

Die Entscheidung ist aus SAP-Sicht positiv zu bewerten. Bisher wurde die Oracle/Sun-Hotspot-JVM als Referenz-Implementierung der Java-Plattform herangezogen. Diese JVM war nicht im Quelltext verfügbar, sodass gerade die Fehleranalyse erheblich erschwert wurde. Darüber hinaus wird die Erstellung einer Java-kompatiblen JVM deutlich vereinfacht.

Was erwarten Sie von der Java-Community?
Durch das OpenJDK wird die Java-Community vermehrt in die Weiterentwicklung von Java eingebunden. Wir denken, dass dadurch eine Stärkung des Java-Ökosystems erreicht wird und die Java-Plattform weiter an Bedeutung gewinnen wird. Insbesondere hoffen wir, dass durch das Zusammenwirken der Community und großer Firmen wie SAP, IBM, Oracle, Red Hat und Apple auch in Zukunft eine breite Technologie-Abdeckung durch Java gewährleistet ist.

In welche Richtung sollte Java sich entwickeln?
Neue Technologie-Trends generieren neue Anforderungen. Wir wünschen uns, dass die Java-Plattform auch zukünftig Lösungen für diese Herausforderungen liefert. Man erkennt den Einfluss der Cloud bereits an den Plänen für Java SE 8 bezüglich Modularität und Multi-Tenancy. Wir glauben, dass eine lebendige Community, in der Individuen ebenso vertreten sind wie Technologie-Unternehmen, die beste Voraussetzung ist, damit Java auch weiter eine attraktive Plattform für Technologie- und Anwendungs-Entwicklung bleibt.

Java 7 – Mehr als eine Insel

Gelesen von Jürgen Thierack

Beim Begriff „Java-Bibel" denkt man in erster Linie an Christian Ullenbooms „Java ist auch eine Insel" aus dem Galileo-Verlag (ISBN 978-3-8362-1802-3), das zur Java-Version 7 nun schon in der 10. Auflage vorliegt. Zu diesen 1.308 Seiten gesellen sich seit diesem Jahr weitere 1.433 Seiten in dem Fortsetzungsband „Java 7 – Mehr als eine Insel". Wer die Metapher mit der Bibel fortspinnen möchte, kann vom Alten und Neuen Testament sprechen.

Schon der erste Band hat eine enorme Stofffülle, zu der der Autor in seinem Vorwort selber sagt: „Der Detailgrad der Insel wird von keinem anderen (dem Autor bekannten) deutsch- oder englischsprachigen Grundlagenbuch erreicht." Da stellt sich die Frage, wozu ein weiterer Band dienen soll. Die Antwort lautet: Es werden die Bibliotheken der Java Standard Edition 7 durchgesprochen und da ist eben viel Funktionalität kodiert. Das Buch beginnt mit den Neuerungen von Java 7. Über das gesamte Buch hinweg werden vom Autor Themen auch jenseits der Java-Bibliotheken abgehandelt, etwa im Kapitel 8 die Dateiformate. Man kann sagen, dass der Autor das Umfeld, in der die Java-Bibliotheken zum Einsatz kommen, stets sehr ausführlich erläutert. So wird in Kapitel 7, das „XML" zum Thema hat, erst erklärt, was Auszeichnungssprachen sind, bevor die Java-Unterstützung zu XML an die Reihe kommt.

Fazit: Man bekommt für knapp 50 Euro viel Buch. Für den Leserkreis gilt das Gleiche wie für den ersten Band: ab fortgeschrittenem Anfänger aufwärts für jeden Java-Entwickler geeignet. Beide „Bibeln" sind auch als „Galileo Open Book" im Internet einsehbar und können als HTML-Dokumente komplett heruntergeladen werden.

Titel:	**Java 7 – Mehr als eine Insel**
Autor:	Christian Ullenboom
Verlag:	Galileo Computing
Umfang:	1433 Seiten
Preis:	49,90 €
ISBN:	978-3-8362-1507-7

eBook (downloadbar) im Preis enthalten

Jürgen Thierack
thierack@igfm-muenchen.de

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

Java-EE-Dreikampf: GlassFish, JBoss und Geronimo

Frank Pientka, MATERNA GmbH

Ende letzten Jahres wurden einige Applikationsserver für Java EE 6 zertifiziert. Den neuen Flaggschiffen steht ihre Bewährungsprobe jedoch noch bevor. Ein guter Zeitpunkt, sich GlassFish, JBoss und Geronimo näher anzuschauen.

Für Java EE 6 gilt der Satz: „Was lange währt, wird endlich gut." Die lange Wartezeit zwischen Veröffentlichung der Spezifikation und der Referenzimplementierung vor mehr als zwei Jahren haben die meisten Applikationsserver-Anbieter genutzt, um ihre Produkte auf die Anforderungen der Zukunft auszurichten und dazu komplett umzubauen.

Dieser Wechsel stellt nicht nur für die Hersteller, sondern auch für die Nutzer einige Herausforderungen bereit. Doch der Umstieg auf Java EE 6 lohnt sich. Alle drei hier vorgestellten Applikationsserver verwenden intern OSGi; damit wird nicht nur der Server schlanker, sondern es lassen sich auch flexiblere Anwendungen entwickeln. Ein weiterer gemeinsamer Punkt ist, dass die Administrierbarkeit und die Integration in bestehende System- und Entwicklungs-Umgebungen verbessert wurden. Gerade Betriebsthemen wie Überwachung, Skalierbarkeit und Automatisierbarkeit spielen hier eine große Rolle.

GlassFish: die Referenz-Implementierung

Bei GlassFish handelt es sich um die Java-EE-Referenz-Implementierung. Es ist kein Wunder, dass er immer die aktuellsten Standards unterstützt. Die Zukunft von GlassFish war nach der Übernahme von Sun durch Oracle etwas ungewiss. Doch spätestens nach der Vorstellung von GlassFish 3.1.1 im Februar 2011 legte sich die Aufregung, da dieser endlich die lang ersehnte Cluster-Fähigkeit enthält. Genau ein Jahr später erschien die aktualisierte und fehlerbereinigte Version GlassFish 3.1.2. Neben der Open-Source-Variante bietet Oracle eine gleichnamige kommerzielle Variante mit Support an. Die Hoffnung, dass in GlassFish neben Java auch andere Java-Skriptsprachen ausgeführt werden, hat sich nicht erfüllt und wurde in den aktuellen Versionen nicht mehr unterstützt.

Für die kommende Java-EE-7-Version existiert bereits eine erste Implementierung mit GlassFish 4.0. Doch bevor Java EE 7 erscheint, wird es noch eine Version 3.2 geben. GlassFish 3.1 wird sicher für die nächsten Jahre die am meisten eingesetzte GlassFish-Version bleiben, da im Bundle mit NetBeans oder dem Java EE 6 SDK regelmäßige Aktualisierungen geliefert werden. Ebenso gibt es Erweiterungen wie das Oracle Enterprise Pack für Eclipse, Integrationen in die Fusion-Produktreihe oder Migrationsassistenten für WebLogic. Gerade die Open-Source-Variante ist bei vielen Oracle-Kunden beliebt.

JBoss: der Platzhirsch

JBoss ist der älteste Open-Source-Applikationsserver. Dadurch ist er am Markt weit verbreitet. Technologisch ist er in den letzten Jahren etwas ins Hintertreffen geraten. Das hat sich mit der Zwischenversion 6 und mit der gerade frisch zertifizierten Version 7.1 geändert. Neben einer modernisierten Administrationsoberfläche wurde vor allem unter der Haube mit OSGi einiges geändert. Neben Felix, Tomcat und Aries werden hier einige Apache-Komponenten verwendet. Dadurch, dass man die CDI-Referenz-Implementierung Weld an Apache übertrug und das eigene Web-Service-Framework zugunsten von Apache CXF aufgab, scheint es bei JBoss ein Umdenken hinsichtlich der Zusammenarbeit mit Apache zu geben.

Gerade mit den weiteren JBoss-Produkten wie Hibernate, SEAM, RichFaces, Jgroups, Infinispan, DROOLS, jBPM, HornetQ oder GateIn integriert sich der JBoss-Applikationsserver gut, wenn man die Versionsabhängigkeiten beachtet. Hier bietet JBoss mit seiner Mutter Red Hat einen bunten Strauß von Produkten an.

Die Administrierbarkeit und Überwachung von großen JBoss-Installationen ist mit RHQ möglich. Die Entwicklung wird sowohl durch Werkzeuge wie die Eclipse-JBoss-Tools oder Test-Frameworks als auch durch JSFUnit oder Arquillian gut unterstützt.

Die am meisten eingesetzte Version ist 5.1. Ein Versionswechsel auf 7.1 ist nicht ohne Probleme. Gerade da die EJB-Abwärtskompatibilität recht spät eingebaut wurde, ist hier mit Überraschungen zu rechnen. Eine Einschränkung der Open-Source-Varianten von JBoss bleibt, da die Entwicklung nicht wirklich öffentlich passiert und Anwender keine Fehlerkorrekturen für alte Produkte erhalten.

Etwas kurios mutet die große Lizenzsammlung mit 22 Lizenzen von GPL bis MIT an. Trotzdem zählt Red Hat zu den großen Open-Source-Namen, die sich im Java-Community-Prozess an der Weiterentwicklung von Java EE beteiligen. Fehlerkorrekturen werden immer nur über die aktuellste Community-Version angeboten oder der Kunde muss auf die erst später erscheinende stabile, aber kostenpflichtige Enterprise-Version wechseln. Wer jedoch mit dieser Unsicherheit leben kann, für den ist JBoss immer noch ein gutes Rund-um-Angebot. Wobei hier der Schritt auf die nächste Applikationsserver-Version, durch die Architektur und Versionsaktualisierung

der verwendeten Komponenten, nicht ohne Stolperfallen ist. Spätestens mit der für Mitte des Jahres geplanten Enterprise-Applikationsserver-Version 6, die wiederum auf der Community-Version 7.1 beruht, sollten die meisten aktuellen Probleme beseitigt sein.

Geronimo: der Nachzügler

Apache Geronimo ist vor allem als freie Herausforderung zu JBoss angetreten. Deswegen setzen viele Entwickler bewährte andere Open-Source-Komponenten ein. So hat es Geronimo seit der Version 1.0 und der Zertifizierung für J2EE 1.4 immerhin als zweiter Open-Source-Server nach GlassFish geschafft, jede JEE-Zertifizierung zu erfüllen. Die am meisten eingesetzte Version ist 2.1.8, wobei bei dieser Version seit der Vorstellung Anfang 2008 immer wieder Fehlerkorrekturen erschienen sind.

Für Geronimo besteht fünf Jahre offizieller Support und weitere drei Support-Jahre sind bei IBM buchbar. Damit bietet Geronimo – typisch Apache – von allen drei Produkten die längsten Support-Zeiträume bei regelmäßigen Weiterentwicklungen, was eine gute Planungssicherheit ergibt. Jedoch darf nicht verschwiegen werden, dass die Entwicklung und die Unterstützung der Community in den letzten Jahren etwas nachgelassen hat. Trotzdem bleibt Geronimo, gerade als Integrationsplattform für die vielen Apache-Java-EE-Projekte, wichtig.

Geronimo bietet gerade für Eclipse und Maven eine gute Integration an. Es gibt viele Möglichkeiten, das Deployment zu automatisieren oder den Server über eine komfortable Oberfläche zu konfigurieren oder zu überwachen.

Wer sich zutraut, selbst Hand anzulegen und sich trotz unvollständiger Dokumentation auf Open Source einzulassen, erhält

	GlassFish	JBoss	Geronimo
Lizenz	CDDL1.1, GPL 2	LGPL 2.1	ASF
Beurteilung	+ sehr gute Dokumentation, viele Beispiele + Cluster Support + Migrations-, Upgrade-Unterstützung + gute IDE-, BUILD-Integration + gute Administrier-, Automatisierbarkeit (GUI, CLI) + Supportmöglichkeit + aktuelle Referenzimplementierung + regelmäßige Releases auch ältere mit klarer Roadmap + mittlere Produktgröße 83 MB	+ gute Dokumentation + verbesserte Administrierbarkeit + gute Integration in JBoss, Red-Hat-Produkte + gute Cloud-Integration + gute Testbarkeit Arquillian + gute IDE-, BUILD-Integration - Webcontainer: Tomcat - seltenere und unregelmäßige Releases nur für aktuellste Version - große Community, jedoch mit wenig Mitwirkungsmöglichkeiten bei der Produktweiterentwicklung - relative neue Architektur - eingeschränkte Abwärts-kompatibilität - Langzeitsupport nur für kommerzielle Variante - größtes Produkt 103 MB	+ stabiles, schlankes und gut anpassbares Produkt mit ausgereiften Best-of-Breed-Komponenten + gute IDE-, BUILD-Integration + Langzeitsupport + gute Mitwirkungsmöglichkeit, jedoch kleine Community + regelmäßige Releases auch ältere + hybride Version für OSGi, Jetty/ Tomcat, CXF, Axis2 - wenig Dokumentation, Beispiele + kleinstes Produkt mit 74-91MB + sehr freie ASF-Lizenz
Plattformen	LINUX, Windows, SOLARIS, AIX	LINUX und Windows	LINUX, Windows, SOLARIS, AIX
Webcontainer	Grizzly	Tomcat	Jetty, Tomcat
OSGi	Felix	Felix, Karaf	Equinox, Felix, Karaf, Aries
JSF	Mojarra	Mojarra, Richfaces, SEAM	myFaces, OpenWebBeans
CDI	Weld	Weld	OpenWebBeans
JAX-WS, JAX-RS	Metro, Jersey	CXF, RESTEasy	CXF, Axis2, WINK
JPA	EclipseLink, Hibernate Validator	Hibernate, Hibernate Validator	OpenJPA, BVal

Tabelle 1: Übersicht und Bewertung

mit Geronimo einen verlässlichen und innovativen Begleiter. Etwas Konkurrenz im eigenen Haus hat Geronimo durch den ebenfalls zertifizierten Tom-EE-Server und den demnächst geplanten Karaf-EE-Server bekommen. Für die hybride Entwicklung von OSGi und Java-EE-Anwendungen scheint Geronimo eine gute Wahl zu sein, da die Beispiele von Karaf und Aries vom gleichen Hersteller ohne zusätzliche Anpassungen in Geronimo lauffähig sind. Durch die stagnierenden Committer-Zahlen und den Ausstieg von Apache aus dem Java-Community-Prozess sind die Apache-Projekte zwar nicht gefährdet, doch die Weiterentwicklung etwas gebremst. Umso erfreulicher ist, dass große Unterstützer des JCP wie IBM und JBoss sich immer mehr bei Apache-Projekten engagieren und so die Open-Source-Fahne weiter wehen lassen.

Zusammenfassung

Die Unterschiede zwischen den Produkten werden immer geringer. Trotzdem kann die folgende Übersicht bei der ersten Orientierung helfen (siehe Abbildung 1 und Tabelle 1). Interessant ist der Vergleich der Entwicklung der Codegröße der jeweiligen Produkte. Hier zeigt sich die Tendenz, dass die Produkte, ähnlich wie der Java-Standard, immer größer werden. Durch die neue Modularität sind die neuen Applikationsserver immer anpassbarer geworden, was sich nicht zuletzt auf das schnelle Startverhalten und den geringen Ressourcen-Verbrauch auswirkt.

Die Installation erfolgt bei JBoss und Geronimo durch das Installationsformat zip/tar einfach durch Auspacken. Dannach muss nur noch ein aktuelles JDK und die „JAVE_HOME"-Variable gesetzt sein. Bei GlassFish oder bei der auf Geronimo basierenden WebSphere Community Edition geht es etwas komfortabler. Es gibt einen grafischen Wizard für die unterstützten Plattformen, der einem hilft, die Java-Umgebungsvariablen zu setzen und die ersten Einstellungen vorzunehmen.

Nach dem Start des Servers muss man nur noch die entsprechende Admin-Konsole mit dem in der Dokumentation beschriebenen Admin-Anmeldung im Browser aufrufen. Für die initiale Verwendung ist erst mal nichts umzukonfigurieren. Viele, aber nicht alle Konfigurationsmöglichkeiten sind über diese Oberfläche möglich.

Der erste Kontakt mit den neuen Java-EE-Servern klappt für Neueinsteiger problemlos, wer jedoch tiefer einsteigen möchte, ist auf eine gute Dokumentation und Erfahrung angewiesen. Gerade was die Bereiche „Hochskalierbarkeit" oder „Migration" betrifft, muss sich beim Umstieg etwas einarbeiten.

Fazit

Durch die Übernahme von Sun durch Oracle hat Java wieder eine Zukunft. Der aktuelle und der kommende Java-EE-Standard haben zu einer Renaissance der Applikationsserver geführt. Trotzdem weisen die Applikationsserver mit OSGi den Weg zu mehr Modularität. Dadurch, dass der Java-Markt gesättigt ist, wird es hier wenig Bewegung geben. Für viele Projekte steht der Schritt in die Java-EE-6-Zukunft noch bevor.

Ungeachtet vieler Unterschiede bei Lizenz, Support, Architektur und den verwendeten Komponenten, haben die hier vorgestellten Produkte durch den Java-EE-Standard viele Gemeinsamkeiten. Kurz zusammengefasst kann man sagen, dass GlassFish sich etabliert hat und für die Java-EE-Funktionen am weitesten ausgereift ist. JBoss und Geronimo bieten erst seit Kurzem eine volle Java-EE-6-Unterstützung, sodass hier noch einige Fehler schlummern, die im Laufe dieses Jahres behoben sein werden. Als Open-Source-Applikationsserver bieten alle drei Kandidaten interessante und innovative Alternativen zu kommerziellen Anbietern oder reinen Webservern an, die einen Wechsel attraktiv machen. Erste Ansätze, wie der Weg der Applikationsserver in die Cloud aussehen könnte, sind zwar sichtbar, doch noch nicht, wer hier das Rennen machen könnte.

Links

- http://www.ohloh.net/p/compare?project_0=GlassFish&project_1=Apache+Geronimo&project_2=JBoss+Application+Server
- http://geronimo.apache.org
- http://www.jboss.org/jbossas
- http://glassfish.java.net/

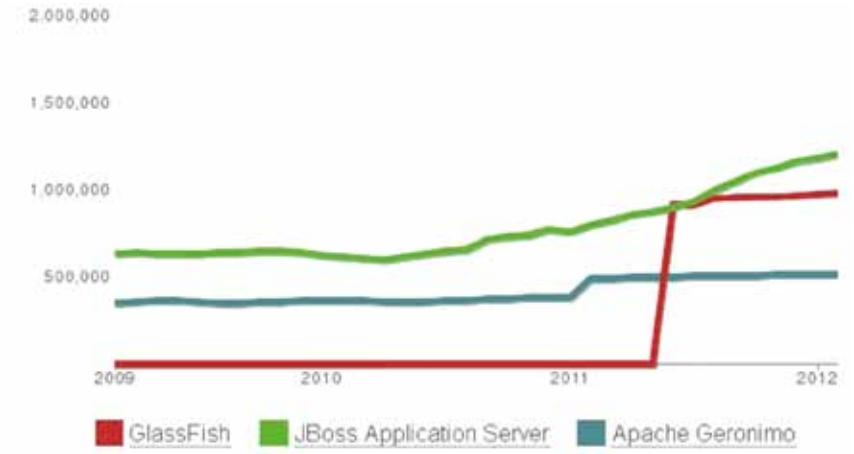


Abbildung 1: Vergleich der Codegrößen (Ohloh.net)

Frank Pientka
frank.pientka@materna.de

Frank Pientka ist Senior Software Architect bei der MATERNA GmbH in Dortmund. Er ist zertifizierter SCJP und Gründungsmitglied des iSAQB. Als Autor eines Buches zu Apache Geronimo beschäftigt er sich intensiv mit dem Einsatz von Java-Open-Source-Software, insbesondere mit Applikationsservern.

WebLogic Server im Zusammenspiel mit Oracle Real Application Cluster

Michael Bräuer und Sylvie Lübeck, ORACLE Deutschland B.V. & Co. KG

„Active GridLink for Real Application Cluster (RAC)" ist eine Funktionalität des Oracle WebLogic Servers, die erstmalig mit dem WebLogic Server 11g (10.3.4) ausgeliefert wurde.

Die damalige Restriktion, „Active GridLink for RAC" nur in Verbindung mit Oracle-Exalogic-Systemen nutzen zu dürfen, wurde inzwischen aufgehoben, sodass die Funktionalität nun auch auf herkömmlichen – auch Nicht-Exalogic-Umgebungen – eingesetzt werden kann. Dadurch wird diese wertvolle Funktion für eine Vielzahl von Kunden interessant, die bereits Anwendungen auf Basis von Oracle RAC und Oracle WebLogic Server betreiben. Der Artikel zeigt die Versionen des Oracle WebLogic Servers 11g (10.3.6) beziehungsweise 12c (12.1.1). Ziel ist es, die Möglichkeiten des „Active GridLink for RAC" vorzustellen und mit den weiterhin verfügbaren „Multi Datasource"-Lösungen zu vergleichen.

Mehrschichtige Architektur

Mit der Verbreitung von Kommunikationsstandards (zum Beispiel Http für die Kommunikation im World Wide Web) finden seit den neunziger Jahren des letzten Jahrhunderts mehrschichtige Systeme auch im betrieblichen Anwendungsumfeld stetige Verwendung. Innerhalb solcher Systeme wird Programmlogik auf verschiedenen Schichten verteilt. Klassisch werden drei Schichten unterschieden (siehe Abbildung 1):

Eine solche Architektur hat gegenüber herkömmlichen Client-Server-Systemen (etwa klassischen Zwei-Schichten-Datenbank-basierten Anwendungen) verschiedenste Vorteile. So kann man bei Bedarf in jeder Schicht skalieren, sowohl horizontal durch Bereitstellung neuer Hardware als auch vertikal durch optimale Nutzung der auf einem physikalischen Rechner vorhandenen Ressourcen. Durch Redundanzen auf jeder Ebene kann die gewünschte Fehlertoleranz erreicht werden. Sowohl Skalierbarkeit als auch Fehlertoleranz werden durch Oracle WebLogic Server auf Middleware-Ebene und durch RAC auf Datenbank-Ebene ermöglicht.

RAC ist eine Option der Oracle-Datenbank. Die grundlegende Idee besteht darin, mehrere Instanzen gleichzeitig für eine physikalische Datenbank zu nutzen. Auf diese Instanzen kann die Last durch die Definition von Server Pools und Services verteilt werden [1].

WebLogic Server im Zusammenspiel mit RAC

Java-EE-Anwendungen, die in der Mittelschicht laufen, müssen sich oft mit einer relationalen Datenbank verbinden, um Daten abzufragen oder zu manipulieren.

Um einen ständigen Wiederaufbau von Verbindungen mit der Datenbank zu verhindern, bieten fast alle Java-EE-Server die Möglichkeit, Verbindungen in einem Pool vorzuhalten und bei Bedarf wiederzuverwenden. So muss nicht für jede Anfrage eine neue Verbindung auf- und wieder abgebaut werden. Eine Datasource abstrahiert die Definition und Verwendung solcher Verbindungen für den Anwendungsentwickler und den Administrator.

Java-EE-Server unterscheiden sich jedoch in der Art und Weise der Funktionalität, welche die Datasources liefern. Eine typische Fragestellung ist, ob eine Datasource und deren Connection Pool die Verfügbarkeit von Ressourcen der zugrunde liegenden Datenbank mitbekommen, sei es bei geplanten als auch bei ungeplanten Ausfällen, und wie auf solche Veränderungen reagiert wird. Speziell beim Einsatz von RAC ergeben sich vielfältige Fragen:

- Wie schnell bekommt eine Anwendung mit, dass zusätzliche RAC-Knoten zur Verfügung stehen, wenn diese gestartet werden?

Abbildung 1: Mehrschichtige Anwendungen

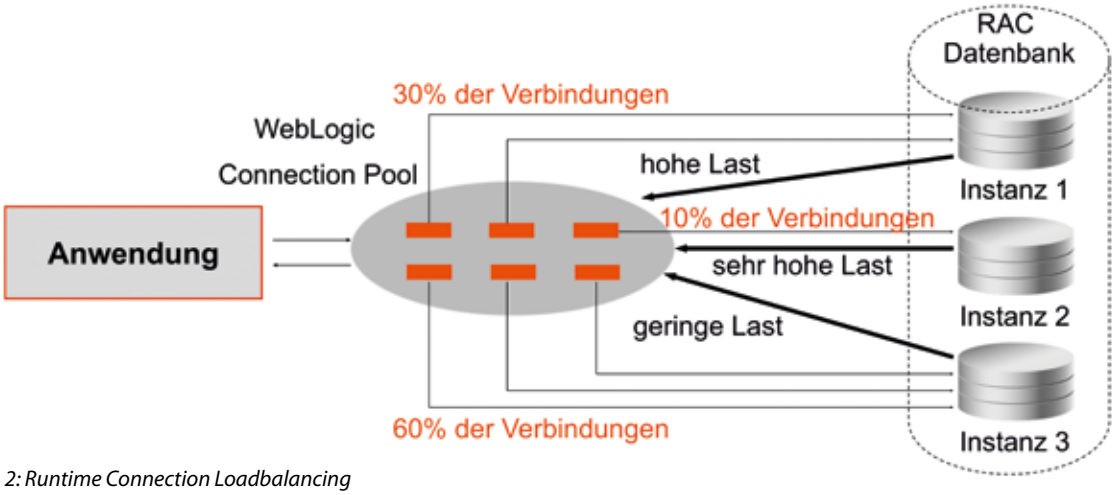


Abbildung 2: Runtime Connection Loadbalancing

- Werden die Verbindungen des Pools bei Veränderungen innerhalb des RAC-Clusters automatisch angepasst, erfolgt also ein automatisches Rebalancieren von Verbindungen beim Hinzunehmen oder Verlassen eines Knotens des RAC-Clusters zur Laufzeit?
- Wie kann die Datasource mit neuen Features von RAC umgehen, beispielsweise mit der Vereinfachung der Konfiguration durch Single-Client-Access-Name-Adressen (SCAN-Adressen)?
- Inwiefern werden Datenbank-Operationen, die an verteilten Transaktionen partizipieren, auf dem gleichen RAC-Knoten ausgeführt (Transaktions-Affinität)?
- Können Datenbank-Operationen, die ein Benutzer innerhalb einer Http-Session durchführt, optimalerweise auf demselben RAC-Knoten ausgeführt werden (Session-Affinität)?

Oracle WebLogic Server kann für die Zusammenarbeit mit Oracle RAC in zwei verschiedenen Varianten konfiguriert werden. Die erste Variante besteht in der Konfiguration von Multi-Datasources, die zweite in der Definition von GridLink-Datasources (Active GridLink for RAC).

Active GridLink for RAC

„Active GridLink for RAC" stellt ein Alleinstellungsmerkmal des Oracle WebLogic Servers dar, um den Anforderungen an Skalierbarkeit, Lastverteilung und Fehlertoleranz im Zusammenspiel von Mittelschicht und Datenbankschicht gerecht zu werden.

Basis für diese Funktion ist der Oracle Notification Service (ONS). ONS-Daemons auf Datenbankseite übermitteln Status-Events – sogenannte „Fast Application Notification (FAN)"-Events – an ONS-Clients, beispielsweise beim Hochfahren einer RAC-Instanz. Diese Information, die Veränderungen des aktuellen Zustands des RAC-Systems widerspiegelt, ist die Basis für die Funktionen „Fast Connection Failover" und „Runtime Connection Load Balancing", die nachfolgend erläutert werden.

Fast Connection Failover

Fast Connection Failover (FCF) ist eine auf oben beschriebenen FAN-Events basierende Funktionalität des Oracle Universal Connection Pools (UCP), die genutzt wird, um Fehlertoleranz für JDBC-Verbindungen zu unterstützen. Aus diesem Grunde ist die Verwendung des Oracle-Thin-JDBC-Treibers eine Voraussetzung für die Verwendung von „Active GridLink for RAC".

Für die Verfügbarkeit einer Anwendung bedeutet dies, dass bei Ausfall einer RAC-Instanz nach Eingang eines entsprechenden FAN-Events der Connection Pool, der auf diese RAC-Instanz verweist, gesperrt wird und Verbindungen anderer Pools, die auf laufende RAC-Instanz verweisen, benutzt werden. Die Anwendung bleibt weiterhin verfügbar. Nicht abgeschlossene SQL-Transaktionen der gekappten Verbindungen müssen durch die ohnehin zu implementierende Fehlerbehandlung seitens der Anwendung abgefangen werden.

Ist zu einem späteren Zeitpunkt die RAC-Instanz wieder verfügbar, wird dies automatisch erkannt und der Pool wieder aktiviert. Auch neue Instanzen, die etwa aus Gründen der Skalierbarkeit hinzukommen, werden automatisch eingebunden. So ist eine optimale Ressourcen-Ausnutzung des Gesamtsystems sichergestellt. Ein Administrator muss dabei nicht aktiv in den Prozess eingreifen.

Skalierbarkeit durch Runtime Connection Loadbalancing

Zur Erreichung der bestmöglichen Performance und Skalierbarkeit ist die optimale Verteilung der Anfragen an die verfügbaren RAC-Instanzen zur Laufzeit erforderlich. Dies wird durch das Zusammenspiel von Active-GridLink-Datasources und deren „Runtime Connection Load Balancing

RAC-Service Konfiguration

GOAL = SERVICE_TIME:

```
srvctl modify service -d db_unique_name -s demosvc -B SERVICE_TIME -j SHORT
```

GOAL = THROUGHPUT:

```
srvctl modify service -d db_unique_name -s demosvc -B THROUGHPUT -j LONG
```

Weitere Informationen zur Konfiguration des RAC Services für die Nutzung von Load Balancing Advisory (siehe http://docs.oracle.com/cd/E11882_01/rac.112/e16795/hafeats.htm#BABICAJC)

Abbildung 3: RAC-Service-Konfiguration für RCLB-Unterstützung

```
[oracle@vm03-rac1-102178 ~]$ srvctl config scan
SCAN name: vm-rac-102178-scan, Network: 1/138.3.20.0/255.255.252.0/cth0
SCAN VIP name: scan1, IP: /vm-rac-102178-scan.osc.uk.oracle.com/138.3.21.220
```

Listing 1: srvctl config scan

```
[oracle@vm03-rac1-102178 ~]$ srvctl status database –d RACWLS
Instance RACWLS1 is running on node vm03-rac1-102178
Instance RACWLS2 is running on node vm04-rac2-102178
```

Listing 2: Wurde der RAC erfolgreich gestartet?

```
[oracle@vm03-rac1-102178 ~]$ srvctl status device -s demosvc -d RACWLS
Service demosvc is running on instance(s) RACWLS1,RACWLS2
```

Listing 3: Wurde der Service erfolgreich gestartet?

```
[oracle@vm03-rac1-102178 ~]$ srvctl config scan listener
SCAN Listener LISTENER_SCAN1 exists. Port:TCP:1522
[oracle@vm03-rac1-102178 ~]$ srvctl status scan_listener
SCAN Listener LISTENER_SCAN1 is enabled
SCAN listener LISTENER_SCAN1 is running on node vm04-rac2-102178
```

Listing 4: Wurde der SCAN-Listener erfolgreich gestartet?

```
[oracle@vm03-rac1-102178 ~]$ sqlplus /nolog
SQL*Plus: Release 11.2.0.2.0 Production on Tue Apr 10 06:09:31 2012
Copyright (c) 1982 2010, Oracle. All rights reserved.
SQL> connect SYS@RACWLS as sysdba
Enter password:
Connected.
SQL> show parameter listener
NAME                                 TYPE        VALUE
------------------------------------------------------------------
Listener networks    string
Local_listener                       string      (DESCRIPTION(ADDRESS_LIST=(AD
DRESS-(PROTOCOL=TCP)(HOST=138.
3.21.222)(PORT=1523))))
remote listener                      string      //vm-rac-102178-scan:1522
SQL>
```

Listing 5: Wurde der Remote-Listener richtig konfiguriert?

```
SQL> connect sys@RACWLS as sysdba
...
SQL> alter system set remote_listener=`//rac-scan:port` scope=both sid=`*`;
System altered.
SQL> alter system register;
System altered.
...
```

Listing 6: Remote-Listener-Konfiguration

(RCLB)"-Funktion gewährleistet. Auch in diesem Fall werden FAN-Events benutzt, um Connection Pools umzuorganisieren. Jedoch beinhalten die Events hier Informationen über die Veränderung der Auslastung des angeschlossenen RAC-Systems.

Die Besonderheit von RCLB besteht darin, dass je nach Auslastung und Verfügbarkeit der RAC-Instanzen die Verteilung der Anfragen optimal ist und dynamisch angepasst wird (siehe Abbildung 2). Bei der Ermittlung der optimalen Verteilung werden CPU-Auslastung, Verfügbarkeit und Antwortzeiten berücksichtigt.

Ohne FAN-Konfiguration auf RAC-Seite erfolgt die Verteilung der Anfragen aus dem Connection Pool zu der jeweiligen RAC-Instanz nach dem Round-Robin-Algorithmus.

RCLB unterstützt sowohl non-XA- als auch XA-Umgebungen mit globalen, verteilten Transaktionen. RCLB wird für den jeweiligen Service auf RAC-Seite konfiguriert. Dazu muss der Service „GOAL" entweder auf „SERVICE_TIME" oder „THROUGHPUT" gesetzt sein (siehe Abbildung 3).

Performanceverbesserung durch „GridLink Affinity"

„GridLink Affinity" umfasst zwei Funktionen, die durch die Vergabe eines Affinitäts-Kontext die Performance der Anwendung verbessern. Die Funktion „Session Affinity" stellt sicher, dass alle Anfragen innerhalb ei-

Oracle WebLogic Server: Letzte Neuerungen im Überblick

Die Übersicht auf der folgenden Seite zeigt die wichtigsten Neuerungen des WebLogic Servers (seit der Version 10.3.1). Nähere Informationen zu neuen Funktionen sind jederzeit in der WebLogic-Server-Dokumentation der jeweiligen Version unter „What's New in Oracle WebLogic Server" zu finden, für WebLogic Server 12*c* (12.1.1) unter http://docs.oracle.com/cd/E24329_01/web.1211/e24494/toc.htm und für WebLogic Server 11*g* (10.3.6) unter http://docs.oracle.com/cd/E23943_01/web.1111/e13852/toc.htm.
Dort ist auch aufgeführt, wo die Übersicht der „Deprecated Features" zu finden ist.

Die wichtigsten Neuerungen aus dem Bereich „Architektur“

Funktion	Beschreibung
Engineered Systems mit WebLogic Server	Oracle WebLogic Server wurde für Exalogic seit WebLogic Server 10.3.4 weiter optimiert. Dazu gehört seit 12.1.1 u.a. das Traffic Management über den Oracle Traffic Director als Teil der Exalogic Elastic Cloud Software.
Cloud Computing	Oracle WebLogic Server ist komplett integriert mit Oracle Enterprise Manager Cloud Control 12c und unterstützt den gesamten Cloud-Lebenszyklus. Eine für die Bereitstellung wichtige Komponente ist der Virtual Assembly Builder (derzeit mit WebLogic Server 10.3.6+ zertifiziert), der das Bundling der gewünschten Software, von Anwendungen etc. für eine Oracle VM unterstützt.
In-Memory Datagrid	Die Integration des WebLogic Servers mit Oracle Coherence (Active Cache) gibt es seit WebLogic Server 10.3.3. Im Zusammenspiel mit dem WebLogic Server löst Oracle Coherence als In-Memory Datagrid-Technologie typische Herausforderungen an Skalierbarkeit und Hochverfügbarkeit von jeglichem Zustand in verteilten, serverseitigen Java-Anwendungen. Es können dabei entweder HTTP-Session-Objekte (Coherence*Web) oder aus der Anwendung über die Coherence APIs Java-Objektstrukturen im Cache vorgehalten werden. Seit WebLogic Server 10.3.4 kann die Überwachung, das Starten und Stoppen von Oracle-Coherence-Knoten, auch über den Node Manager erfolgen.
JPA 2.0 mit Oracle TopLink	Ab WebLogic Server 10.3.4 ist TopLink der Default Persistence Provider und ersetzt Kodo/OpenJPA.
Active GridLink Datasource	Active GridLink for RAC (verfügbar seit WebLogic Server 10.3.4) ist inzwischen nicht nur für Exalogic, sondern auch für allgemeine WebLogic-Server-Installationen im Zusammenspiel mit einem Real Application Cluster einsetzbar. Seit 10.3.6 sind weitere Neuerungen wie Connection Labeling und Session Affinity Policy unterstützt.
JMS	JMS-Nachrichten können jetzt auch an Oracle Advanced Queuing angebunden werden. Dies erfolgt seit WebLogic Server 10.3.2 über eine Foreign JMS und eine JDBC Datasource. Für das Thema Hochverfügbarkeit und Skalierbarkeit werden seit WebLogic Server 10.3.4 zusätzlich Partitioned Distributed Topics unterstützt.

Die wichtigsten Neuerungen aus dem Bereich „Entwicklung“

Funktion	Beschreibung
Java EE 6	Vollständige Unterstützung von Java EE 6 Full Profile ab WebLogic 12c (12.1.1).
Java SE 7	Oracle WebLogic Server 10.3.6 und 12.1.1 sind für das Java SE Development Kit (JDK) 7 zertifiziert. JDK 6 kann mit WebLogic Server weiterhin verwendet werden.
Entwicklungsumgebung (IDE) nach Wahl	Als Java-Entwicklungsumgebung für Oracle WebLogic Server 12c sind derzeit NetBeans (7.1 oder höher) oder Oracle Enterprise Pack for Eclipse (12.1.1 oder höher) verfügbar. Oracle JDeveloper bleibt weiterhin die strategische Entwicklungsumgebung für Oracle Fusion Middleware. Die Unterstützung für WebLogic Server 12c ist für das Kalenderjahr 2012 geplant.
Maven-Plug-in	Das neue Maven-Plug-in für WebLogic Server (wls-maven-Plug-in) beinhaltet erweiterte Funktionalität für die Installation, das Starten und Stoppen des WebLogic Servers, die Erzeugung von Domains, die Ausführung von WLST-Skripten (WebLogic Scripting Tool) und das Kompilieren und Bereitstellen von Anwendungen aus der Maven-Umgebung heraus. Die letzten Neuerungen sind seit dem WebLogic Server 12.1.1 verfügbar. Erstmalig war das Maven-Plug-in in der Version 10.3.4 enthalten.

Die wichtigsten Neuerungen aus dem Bereich „Betrieb“

Funktion	Beschreibung
Custom MBeans per WLST manipulierbar	Es ist seit WebLogic Server 10.3.2 möglich, eigene MBeans, die im Domain-Runtime-MBean-Server registriert wurden, mittels WebLogic Scripting Tool (WLST) zu manipuliert.
WebLogic Thin T3 Client	Der WebLogic Thin T3 Client ist seit WebLogic Server 10.3.3 eine leichtgewichtige, hoch performante Alternative zum WebLogic Full oder IIOP Client.
ClassLoader Analysis Tool (CAT)	Mit dem CAT kann seit WebLogic Server 10.3.4 untersucht werden, welche Java-Klassen wie in die Java VM geladen werden. Dies vereinfacht die Fehlerbehebung bei ClassLoader-Konflikten enorm.
Erweiterte Diagnose-Möglichkeiten	Neben der zusätzlich verfügbaren Enterprise-Manager-Umgebung (WLS Management Pack Enterprise Edition), welche die domänenübergreifende Überwachung, Fehlersuche und Administration unterstützt, gibt es eine Reihe von Neuerungen, die innerhalb der WebLogic Server Console zu finden sind. Dazu gehören beispielsweise das Monitoring Dashboard und die Diagnostics Request Performance Page, die seit WebLogic Server 10.3.3 enthalten sind.
GlassFish-Anwendungen in WebLogic Server deployen	Es ist seit WebLogic Server 10.3.6 möglich, einfache Java-Anwendungen, die bisher auf GlassFish bereitgestellt wurden, unter Verwendung des GlassFish Deployment Descriptors (glassfish-web.xml) auf dem WebLogic Server bereitzustellen. Es wird derzeit eine Teilmenge der in glassfish-web.xml enthaltenen Tags berücksichtigt.
TLogs in der Datenbank	Es ist ab WebLogic Server 10.3.6 möglich, die Transaktions-Logs (TLOG) in der Datenbank persistent vorzuhalten. Dies ist für den Recovery-Fall und den damit erforderlichen konsistenten Zustand eminent wichtig.
Restful Monitoring API	WebLogic Server unterstützt das Monitoring mittels RESTful Web Services (JAX-RS).

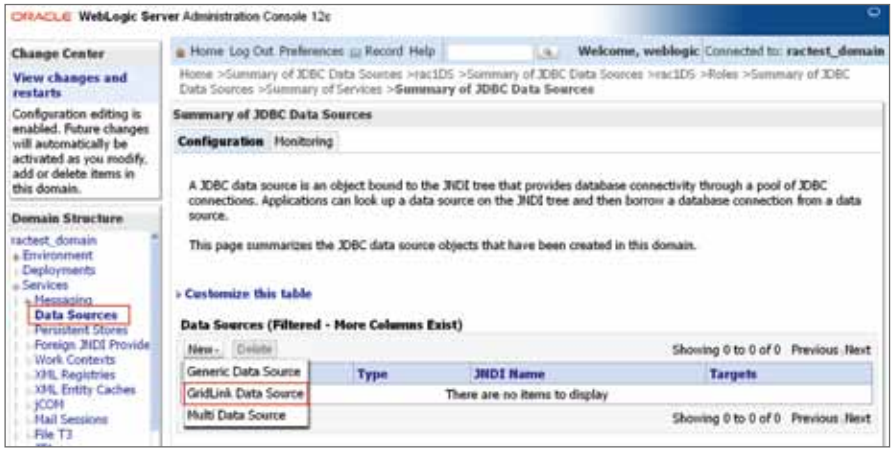


Abbildung 4: Active GridLink for RAC, Schritt 1 der Konfiguration

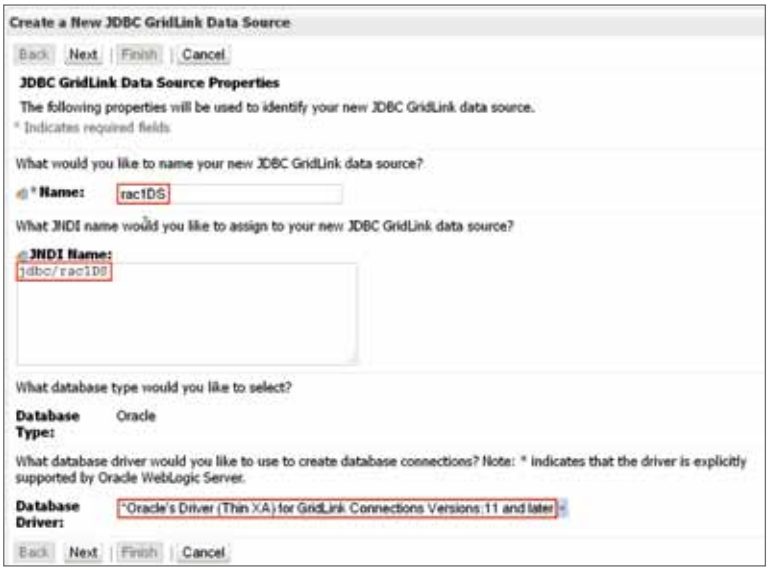


Abbildung 5: Active GridLink for RAC, Schritt 2 der Konfiguration

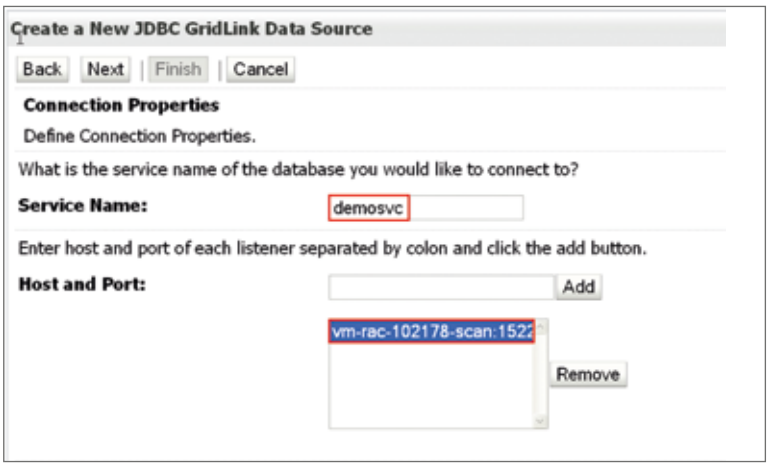


Abbildung 6: Active GridLink for RAC, Schritt 3 der Konfiguration

ner Http-Session an die gleiche RAC-Instanz weitergeleitet werden. Dies wird erreicht, indem beim ersten Aufruf nach der Zuweisung der Verbindung über RCLB ein Affinitäts-Kontext vergeben wird. Session-Affinität ist seit WebLogic Server 10.3.6 verfügbar.

Für Datenbank-Operationen, die an globalen, verteilten Transaktionen partizipieren, wird über einen Affinitäts-Kontext sichergestellt, dass alle Anfragen innerhalb der Transaktion an dieselbe RAC-Instanz weitergeleitet werden. Diese Funktion heißt „Transaction Affinity".

Active-GridLink-Konfiguration

Für die Konfiguration einer Active-GridLink-Datasource werden folgende Informationen von der RAC-Umgebung benötigt: Service-Name, SCAN-Adresse + Port, Benutzername + Kennwort. Der Datenbank-Administrator kann durch den Befehl „onsctl debug" alle notwendigen Informationen liefern. Die SCAN-Adresse lässt sich auch über den Befehl „srvctl config scan" (siehe Listing 1) ermitteln.

Zur Definition der Datasource sollte sichergestellt sein, dass sowohl die RAC-Umgebung (siehe Listing 2) als auch der Service (siehe Listing 3) sowie der SCAN-Listener (siehe Listing 4) erfolgreich gestartet wurden.

Für die erfolgreiche Anbindung via GridLink-Datasource ist die Konfiguration des Remote-Listeners erforderlich (siehe Listing 5). Falls die Konfiguration angepasst werden muss, kann dies wie in Listing 6 dargestellt erfolgen.

Nachdem auf RAC-Seite alles vorbereitet wurde, bleibt noch die Konfiguration seitens Oracle WebLogic Server. In der Oberfläche und teilweise auch in der Dokumentation wird die Active-GridLink-Datasource als „GridLink-Datasource" bezeichnet. Diese kann zum Beispiel über die WebLogic-Server-Admin-Konsole angelegt werden. Unter der „Domain Structure" befindet sich für die Domäne der Menüpunkt „Services –> Datasources". Um eine neue Datasource zu konfigurieren, steht der Menüpunkt „New –> GridLink Datasource" zur Verfügung (siehe Abbildung 4).

Die GridLink-Datasource lässt sich jetzt Schritt für Schritt konfigurieren. Als nächstes werden der Name und der JNDI-Name vergeben. Am Ende erfolgt die Auswahl des gewünschten JDBC-Treibers (siehe Abbildung 5).

Je nach Auswahl des JDBC-Treibers erscheinen im nächsten Schritt unterschiedliche Eingabemöglichkeiten. Zum nächsten Schritt geht es mit „NEXT" weiter. Hier gibt es zwei Alternativen. In unserem Fall

nehmen wir die vorgegebene Auswahl „Enter individual listener information“ und gehen mit „NEXT“ weiter.

Jetzt werden der RAC-Service bekanntgegeben sowie der RAC-Host und der Port eingetragen. Dazu wird die zuvor ermittelte SCAN-Adresse verwendet (siehe Abbildung 6). Darunter ist noch die Konfiguration des Datenbank-Benutzers erforderlich.

Nach Prüfung der Angaben kann jetzt die Verbindung zur Datenbank durch Drücken der Auswahl „Test All Listeners“ getestet werden. Im nächsten Schritt erfolgt die Konfiguration für die FAN/ONS-Events. FAN-Events müssen aktiviert sein und Host und Port für ONS eingetragen werden. In unserem Fall ist das „vm-rac-102178-scan: 6200“.

Jetzt kann die ONS-Client-Kommunikation zum RAC durch Drücken der Auswahl „Test All ONS Nodes“ getestet werden. Zuletzt wird noch definiert, auf welchen Servern oder Clustern die Datasource zur Verfügung stehen soll, und mit „Finish“ wird die Konfiguration abgeschlossen. Nach Definition der GridLink-Datasource kann unter „$WLS_DOMAIN_HOME/yourdomain/config/jdbc/“ die Konfiguration eingesehen werden.

Multi-Datasources: Abgrenzung zu Active GridLink for RAC

Zwar kann Oracle RAC mit WebLogic Server auch ohne „Active GridLink for RAC“-Datasources mittels Multi-Datasource betrieben werden, jedoch haben diese im Vergleich gewisse Einschränkungen:

- Multi-Datasources abstrahieren physikalische Datasources, die jeweils einen Verbindungspool zu einem RAC-Knoten ermöglichen. Es müssen dabei bei n RAC-Knoten n+1 Datasources im WebLogic Server konfiguriert werden. Bei Topologie-Änderungen des RAC-Clusters muss die Konfiguration durch den Administrator angepasst werden.
- Multi-Datasources „pollen“: In wiederkehrenden Intervallen wird der Zustand der RAC-Knoten abgefragt, um Zustandsveränderungen zu ermitteln. Durch die Konfiguration des Polling-Intervalls wird bestimmt, wie „zeitnah“ eine Zustandsänderung erkannt wird.
- Multi-Datasources können entweder für Failover oder für Loadbalancing (natives Round-Robin, aber kein Runtime Connection Loadbalancing) konfiguriert werden.
- Multi-Datasources ermöglichen rudimentäre RAC-Instanzaffinität (für verteilte Transaktionen).

Fazit

Oracle WebLogic Server und RAC wurden entwickelt, um hochskalierbare und fehlertolerante Anwendungen zu betreiben. Oracle WebLogic Server „Active GridLink for RAC“ stellt das wichtige Bindeglied dar, um auch für Java-EE-Anwendungen Eigenschaften von Oracle RAC zur Lastverteilung, Skalierbarkeit und Hochverfügbarkeit bestmöglich auszureizen.

Weiterführende Informationen

- Übersicht zu Oracle RAC [1]: http://www.oracle.com/technetwork/database/clustering/overview/index.html
- Konfiguration von Active GridLink for RAC Datasources in WebLogic Server 12c: http://docs.oracle.com/cd/E24329_01/web.1211/e24367/gridlink_datasources.htm#CHDDJGBH
- Active GridLink for RAC – ein technisches Oracle Whitepaper: http://www.oracle.com/technetwork/middleware/weblogic/gridlink-rac-wp-494900.pdf
- Use Oracle WebLogic Server with a JDBC GridLink Data Source: http://www.oracle.com/technetwork/middleware/weblogic/wls-jdbc-gridlink-howto-333331.html

Michael Bräuer
michael.braeuer@oracle.com
Sylvie Lübeck
sylvie.luebeck@oracle.com

Michael Bräuer ist leitender Systemberater der ORACLE Deutschland B.V. & Co. KG. Er nutzt seine langjährige Berufserfahrung in den Bereichen Anwendungsintegration, Middleware und Java EE, um kritische Geschäftsanwendungen auf das passende technologische Fundament zu stellen.

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

Das Eclipse Modeling Framework: EMFStore, ein Modell-Repository

Jonas Helming und Maximilian Kögel, EclipseSource München GmbH

Mit dem Eclipse Modeling Framework (EMF) können Entitäten einer Anwendung modelliert und als Java-Klassen generiert werden. Dies ermöglicht einen sehr kurzen Entwicklungszyklus vom Datenmodell zu einer lauffähigen ersten Anwendung, in der Entitäten, Attribute und Referenzen bereits in einer Benutzeroberfläche sichtbar sind.

Im ersten Teil dieser Reihe demonstrierten wir, wie Modelle in EMF erzeugt und daraus Code generiert werden kann. Der zweite Teil der Reihe beschrieb den Aufbau des generierten Codes, die API der generierten Klassen sowie nützliche Hilfsklassen. Die Inhalte der ersten beiden Teile sind unter [1] zum Download verfügbar. Nach der Einführung der Grundlagen von EMF widmen wir uns nun den zahlreichen weiteren Frameworks, die auf EMF basieren. Wir beginnen mit dem EMFStore, der es ermöglicht, zügig eine Client-/Server-Anwendung rund um ein EMF-Modell umzusetzen.

Einführung

In den ersten beiden Teilen dieser Reihe wurden ein Datenmodell in EMF erstellt, daraus Code generiert sowie die generierten Klassen im Detail vorgestellt. Abbildung 1 zeigt das bisher verwendete Beispielmodell, das das Spiel Bowling mit einer League und Tournaments abbildet. Im Beispiel wäre nun der nächste logische Schritt, eine Anwendung rund um das Modell zu entwickeln, mit der Entitäten des Modells erstellt, bearbeitet sowie lokal gespeichert werden können. In den meisten echten Szenarios müsste zusätzlich eine Serverlösung bereitgestellt werden, damit Entitäten kollaborativ auf mehreren Clients bearbeitet werden können.

EMF stellt insbesondere für die Entwicklung von Benutzeroberflächen bereits zahlreiche zusätzlich generierte Hilfsklassen bereit; Details dazu stehen beispielsweise in der englischen Version des EMF-Tutorials unter [1]. EMF bietet aber neben generierter Funktionalität noch einen ganz anderen Vorteil. Alle generierten Klassen folgen ähnlich wie beispielsweise Java Beans einer definierten und stabilen API. Außerdem bieten sie Zusatzfunktionen wie beispielsweise einen Notifizierungs-Mechanismus an. Diese Tatsache ermöglicht es, wiederkehrende Features durch Frameworks generisch zu lösen und wiederzuverwenden. Der EMFStore ist ein solches Framework und ermöglicht das kollaborative und verteilte Bearbeiten von Modell-Instanzen (Modell-Repository). In einer fiktiven Anwendung, die auf dem Bowling-Beispielmodell basiert, könnten also die Daten über Bowlingspieler und Turniere in verschiedenen Bowlingcentern parallel angezeigt und bearbeitet werden.

Bei der verteilten Bearbeitung von Modell-Instanzen sind prinzipiell zwei Szenarien zu unterscheiden. Im ersten, dem klassischen Datenbankszenario, arbeiten alle Benutzer live auf den gleichen Daten, Änderungen sind sofort für alle Clients sichtbar. Konflikte sind in diesem Szenario unwahrscheinlich. Im zweiten Szenario arbeiten die Benutzer zunächst offline auf den Entitäten. Dies entspricht dem Anwendungsfall vieler Tools, beispielsweise einem Modellierungstool für Ingenieure. In diesen Domänen spielt es meist weniger eine Rolle, dass ein Client wirklich offline, also der Server nicht erreichbar ist. Vielmehr geht es darum, dass ein Benutzer des Tools nur konsistente Zustände mit dem Rest des Teams teilen will. Erst wenn eine Änderung abgeschlossen ist, wird ein Commit angestoßen, bei dem die Änderungen in die zentrale Version der Entitäten eingespielt werden.

Das zweite Szenario ist also ähnlich einem Source-Code-Management-System (SCM). Für dieses Szenario wurde der EMFStore entwickelt. Im Gegensatz zu SCM-Systemen ist er jedoch nicht auf die Versionierung von Textdateien ausgelegt, sondern versioniert die Entitäten auf der

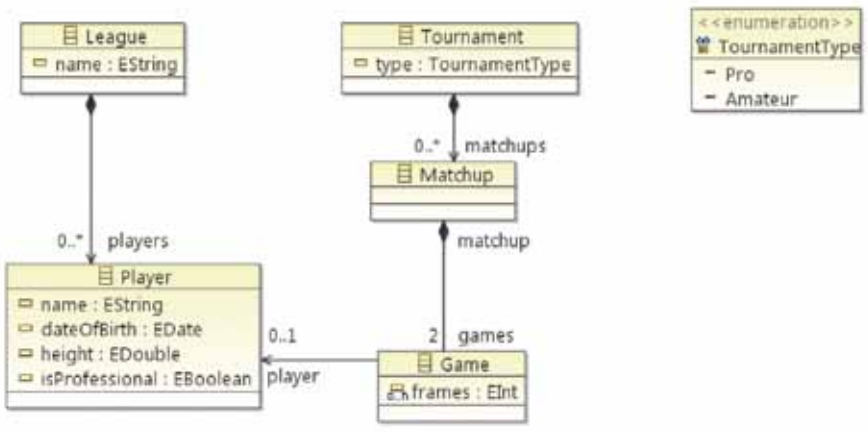


Abbildung 1: Das Beispielmodell

Modellebene. SCM-Systeme interpretieren den Inhalt eines Textfiles meist lediglich auf Zeilenebene, das bedeutet, sie „verstehen" nicht wirklich, was darin steht. Der EMFStore hingegen nutzt die Vorteile von EMF aus, um Änderungen im Modell aufzuzeichnen, zu interpretieren und potenzielle Konflikte aufzulösen. Wie das genau funktioniert, wird im Abschnitt „Was steckt dahinter?" beschrieben. EMFStore ist ein Eclipse-Open-Source-Projekt und kann direkt in die eigene Anwendung integriert werden.

Abbildung 2: Im Navigator und Editor können Entitäten erstellt und bearbeitet werden

Aller Anfang ist leicht

EMFStore ist ein Server-Framework, das üblicherweise „headless" in Anwendungen integriert wird, mehr dazu im Abschnitt „API". Um die Funktionalität mit einem eigenen Modell möglichst einfach testen zu können, nutzt EMFStore die EMF Client Platform [4], um eine generische Benutzeroberfläche (UI) zu erzeugen. Damit kann man ohne weiteres Zutun Entitäten eines Modells erzeugen und modifizieren. Diese lassen sich dann auf den EMFStore übertragen und verteilt bearbeiten. Damit erhält man einen guten Überblick über die Funktionalität, ohne bereits eine eigene Anwendung rund um ein Modell entwickelt zu haben. Initial wird also zunächst nichts anderes als das eigene Modell, beispielsweise das Bowling-Modell, benötigt. Nach dem Setup (siehe Kasten auf Seite 23) und dem Starten von Client und Server kann zunächst lokal ein Projekt erstellt werden. Dazu sollte in die EMFStore-Perspektive gewechselt werden. Alternativ kann man auch manuell die entsprechenden Views, Navigator und EMFStore Browser aus dem View-Menü in Eclipse öffnen.

Modell-Entitäten sind im EMFStore in Projekten organisiert. Ein lokales Projekt kann durch einen Rechtsklick in den leeren Navigator erzeugt werden (Other –> New Project). Innerhalb eines Projekts können dann über „New Model Element" Entitäten eines Modells erzeugt werden, beispielsweise eine League. Ein Rechtsklick auf eine bereits existierende Entität erlaubt es, Kind-Entitäten (Containment) zu erstellen, beispielsweise einen Player innerhalb einer League (siehe Abbildung 2). Aus dem Navigator kann per Doppelklick für eine gewählte Entität ein Editor geöffnet werden. Dieser erlaubt es, beliebige EMF-Entitäten anzuzeigen und zu editieren. Der Navigator und der Editor basieren auf der EMF Client Platform [4].

Ein lokal erstelltes Modell ist zunächst nicht für andere Clients zugreifbar. Dazu muss es auf den EMFStore hochgeladen werden („Share Project"). Beim „Share" kann ein Server ausgewählt werden, auf den das Projekt übertragen wird, im Test-Setup ist dieser Server „localhost". Bei der ersten Anfrage verlangt der Server einen Login, im Test-Setup kann man den Standard-Benutzer „super" mit dem Passwort „super" verwenden. Im EMFStore-Browser (siehe Abbildung 3) wird das Projekt nach dem erfolgreichen Share unter dem gewählten Server angezeigt. Projekte können alternativ auch direkt auf dem Server erstellt werden, sie sind also zunächst nicht lokal verfügbar.

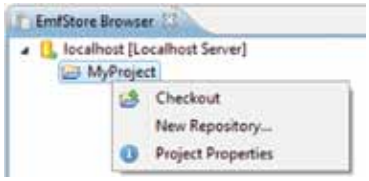


Abbildung 3: Der EMFStore-Browser zeigt auf Servern verfügbare Projekte an und erlaubt den Check-out

Um die Zusammenarbeit über den EMFStore zu testen, ist ein zweiter Client erforderlich. Im Test-Setup kann dies auch der gleiche Client sein, auf dem die Entitäten anfänglich erstellt wurden, das heißt es werden auf dem gleichen Client zwei Versionen des gleichen Projekts erzeugt. Dazu wird das eben geteilte Projekt ein zweites Mal ausgecheckt. Dies wird durch einen Rechtsklick auf das Projekt im EMFStore-Browser über „Check-Out" ausgelöst. Der Navigator zeigt nun zwei Versionen des gleichen Projekts an.

Nun kann eines der beiden Projekte modifiziert werden, beispielsweise wird das „Name"-Attribut eines Players im Editor geändert oder man fügt neue Entitäten hinzu. Sind die Änderungen abgeschlossen, werden sie an den Server übertragen. Dazu wird im Rechtsklick-Menü des Projekts ein „Commit" ausgelöst. Im angezeigten Dialog können die vorgenommenen Änderungen detailliert betrachtet und mit einem Kommentar versehen werden (siehe Abbildung 4). Dieser Commit-Kommentar findet auch Eingang in die Historie des Projekts.

Abbildung 4: Im Commit-Dialog werden Änderungen im Detail betrachtet und mit einem Kommentar versehen

Ein „Update" bringt die zweite Projekt-Instanz auf den gleichen Stand. Erneut zeigt ein Dialog Details der Änderungen an, in diesem Fall derjenigen, die vom Server auf den lokalen Client übertragen werden.

Mit diesem Workflow (Commit/Update) können nun beliebig viele Clients auf den gleichen Entitäten zusammenarbeiten. Hierbei kann es natürlich zu überlappen-

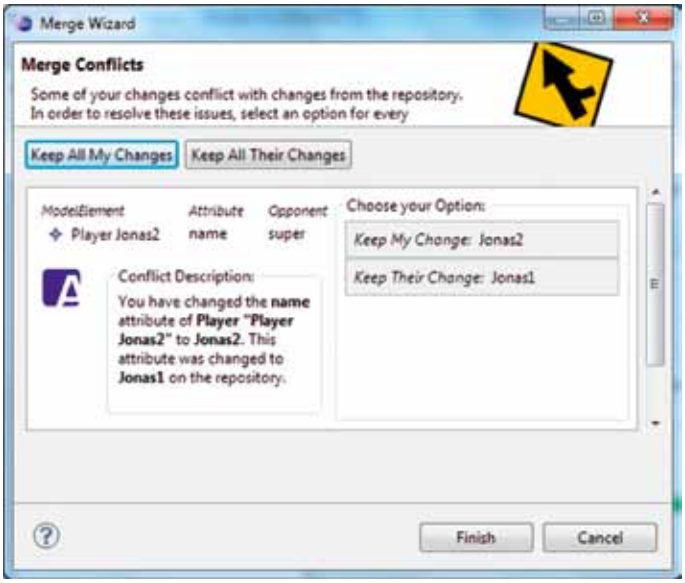


Abbildung 5: Im Merge-Dialog wird im Konfliktfall entschieden, welche Änderungen angewendet werden sollen

den Änderungen kommen. Im Beispiel könnten zwei Clients gleichzeitig den Namen einer Entität auf unterschiedliche Werte setzen. In diesem Fall unterstützt der EMFStore interaktives Merging. Der User kann im Fall von überlappenden Änderungen auswählen, welche Änderungen er tatsächlich übernehmen will (siehe Abbildung 5).

Das generische UI bietet noch einige weiterführende Funktionen. Beispielsweise lässt sich sowohl für Projekte als auch für einzelne Elemente eine Historie anzeigen. In dieser History-View können auch vergangene Versionen wiederhergestellt werden. Alle beschriebenen Funktionen kann man natürlich auch ohne das generische UI verwenden.

Was steckt dahinter?

Die Anwendungsfälle und das Einsatzszenario des EMFStore ähneln stark einem SCM-System wie SVN oder Git. Technisch gesehen arbeitet der EMFStore jedoch fundamental anders. Wollte man EMF-Entitäten in einem SCM versionieren, so müssten diese dazu beispielsweise in XML serialisiert werden. Beim Commit werden dann die Dateien vor und nach der Änderung verglichen, um die Unterschiede zu berechnen – das sogenannte „Diffing". Dieses Vorgehen funktioniert im Wesentlichen zeilenweise.

Während sich SCMs für ihren Einsatzzweck – das Verwalten von Source Code – bewährt haben, hat die Vorgehensweise signifikante Schwächen für Modelle beziehungsweise Entitäten (siehe auch [7]). Dies liegt im Wesentlichen an der Tatsache, dass das SCM auf der Datei- und Textebene arbeitet und das enthaltene Modell nicht kennt. Wird nun auf zwei Clients parallel eine Referenz erzeugt, werden also im Bowling-Modell beispielsweise zwei Player in eine League eingefügt, wird ein SCM dies möglicherweise als Konflikt erkennen, nur weil zufällig die gleiche Zeile in einer Datei verändert wird. Aus Sicht des Modells spricht aber möglicherweise nichts gegen das parallele Einfügen von zwei Referenzen.

Das zweite Problem von SCM-Systemen ist das Auflösen von Konflikten. Gerade XMI-Serialisierungen können mit traditionellen SCMs nur äußerst mühsam und fehlerträchtig zusammengeführt werden. In vielen Tools ist das zeilenweise Mergen für Benutzer ohnehin keine Option.

Um dieses Problem zu lösen, nutzt der EMFStore eine Eigenschaft von EMF-Entitäten aus. EMF bietet die Möglichkeit, über sämtliche Änderungen an den Entitäten informiert zu werden. Der EMFStore zeichnet die vorgenommenen Änderungen auf dem Client zunächst lokal auf. Wird nun ein Commit ausgelöst, muss kein Vergleich mit der Server-Version der Entitäten vorgenommen werden. Durch das Aufzeichnen der Änderungen ist bereits bekannt, was lokal geändert wird. Beim Commit müssen nun lediglich diese Änderungen übertragen werden, gleiches gilt parallel für ein Update. Die Aufzeichnung der Änderungen bietet einen weiteren entscheidenden Vorteil, sie ist deutlich präziser als der Vergleich von Dateien.

Bei einem potenziellen Konflikt kann der EMFStore Informationen auf der Modellebene berücksichtigen, eine Änderung ist also nicht nur eine Modifikation in einer Datei, sondern eine Operation am Modell, beispielsweise „Referenz hinzugefügt". Diese zusätzlichen Informationen erlauben eine deutlich präzisere Konflikterkennungs-Strategie, es kommt also zu weniger falsch erkannten Konflikten (siehe auch [7]). Kommt es zum Konflikt, kann der User mit einer komfortablen UI beim Merge-Prozess unterstützt werden. Die genaue Konflikterkennungs-Strategie kann bei Bedarf sogar modellspezifisch angepasst werden. Nicht zuletzt bietet der EMFStore auch eine deutlich genauere Historie der vorgenommenen Änderungen, beispielsweise wird durch die Aufzeichnung die Reihenfolge der Modifikationen am Modell konserviert.

Die API

Im Abschnitt „Der erste Schritt" wurden die wichtigsten Anwendungsfälle in der generischen UI der EMF-Client-Plattform beschrieben. Diese lässt sich zwar auch anpassen, der EMFStore kann aber auch völlig unabhängig von dieser UI in existierende Tools integriert werden. Dazu sind in diesem Abschnitt ein paar wichtige Beispiele der EMFStore-API beschrieben. EMFStore verwaltet die Projekte auf dem Client in einem sogenannten „Workspace". Den aktuellen Workspace kann man durch folgenden Code abrufen:

```
Workspace workspace = WorkspaceManager.
getInstance().getCurrentWorkspace();
```

Der Workspace bietet den Einstiegspunkt in eine API, um beispielsweise Projekte lokal zu erzeugen:

```
ProjectSpace projectSpace = workspace.cr
eateLocalProject("ProjectName","Project
Description");
```

„ProjectSpace" ist dabei ein Verwaltungscontainer für Projekte. Zusätzlich zu den Projektdaten, also den Entitäten/EObjects, enthält er Meta-Informationen wie

den Projektnamen, die Version, die lokalen Änderungen etc.

Die Zugehörigkeit von Entitäten zu einem Projekt ist über den Containment-Baum in EMF definiert, ein Projekt enthält also alle direkten Teile, Kinder des Projekts und deren Nachfahren. Um Entitäten der Anwendung direkt als Kind in ein Projekt hinzuzufügen, genügt es folgenden Code aufzurufen:

```
projectSpace.getProject().getModelEle-
ments().add(someEObject);
```

Fügt man später Kinder zu einem Element hinzu, das bereits Teil des Projekts ist, so werden diese Kinder automatisch auch Teil des Projekts. Das oben erstellte Projekt ist anfänglich nur lokal verfügbar, es kann durch folgenden Aufruf an den Server übertragen und damit anderen Clients zugänglich gemacht werden (siehe Listing 1).

Die User-Session bestimmt dabei, mit welchem Benutzer und an welchen Server das Projekt übertragen wird. Um das Projekt an einem anderen Client anzufordern (Checkout), genügt folgender Code. Zunächst werden alle Projekte vom Server angefordert und dann (im Beispiel willkürlich) das erste Projekt für den Checkout ausgewählt (siehe Listing 2).

Wenn nun Änderungen am Projekt, also an einem der enthaltenen Elemente ausgelöst werden, beispielsweise „someElement.setName(„newName“)“, dann werden diese aufgezeichnet und können an den Server synchronisiert werden mit:

```
projectSpace.commit(null, null, new NullPro-
gressMonitor());
```

Umgekehrt können Änderungen vom Server, das heißt von anderen Clients, mit folgendem Aufruf abgerufen werden:

```
projectSpace.update();
```

Mit den vier einfachen Funktionen „Share“, „Checkout“, „Commit“ und „Update“ kann eine Anwendung ihre Daten bereits über einen Server synchronisieren. „ProjectSpace“ bietet noch eine Reihe anderer Funktionen, um etwa Änderungen eines Projekts aus der Historie abzurufen oder das Projekt zu einem bestimmten Zeitpunkt in der Historie zurückzusetzen.

Fazit

EMFStore ist ein Framework, um EMF-Entitäten kollaborativ zu bearbeiten. Dabei nutzt er die Vorteile von EMF aus, beispielsweise die generische API und den Notifizierungs-Mechanismus. EMFStore funktioniert damit prinzipiell mit allen EMF-Modellen. Der EMFStore arbeitet nicht wie existierende Source-Code-Management-Systeme auf der Datei-Ebene, sondern versioniert die Entitäten auf Modell-Ebene. Dies erlaubt eine präzisere Konflikterkennung sowie ein komfortableres Mergen. Alle Funktionen des EMFStore sind direkt per API ansprechbar, damit können eigene Clients mit (oder auch ohne) Benutzeroberfläche leicht realisiert werden.

Mithilfe der generischen UI der EMF-Client-Plattform lassen sich die wichtigsten Anwendungsfälle ohne zusätzlichen Aufwand schnell testen. EMFStore und EMF Client Platform stehen unter der Eclipse Public License. Quellcode, Releases und Dokumentation sind unter [5] verfügbar. Im nächsten Teil dieser Reihe wenden wir uns der UI genauer zu und beschreiben die EMF Client Platform im Detail. Dabei ist insbesondere interessant, wie diese an die eigenen Bedürfnisse angepasst werden kann.

Weitere Informationen

[1] http://eclipsesource.com/emftutorial
[2] http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/indigosr1
[3] http://eclipse.org/emfstore/download.php
[4] http://eclipse.org/emfclient
[5] http://eclipse.org/emfstore
[6] http://eclipsesource.com/en/services/eclipse-training/eclipse-modeling/
[7] Operation-based Model Evolution, Maximilian Kögel, Dr. Hut Verlag, ISBN: 978-3843900812
[8] http://eclipsesource.com/blogs/wp-content/uploads/2011/03/exampleSolution.zip

Jonas Helming
jhelming@eclipsesource.com
Maximilian Kögel
mkoegel@ eclipsesource.com

Jonas Helming und Maximilian Kögel sind Eclipse-EMF/RCP-Trainer und Consultants sowie Geschäftsführer der EclipseSource München GmbH. Sie leiten die Eclipse-Projekte „EMFStore“ und „EMF Client Platform“.

```
Usersession usersession = EMFStoreClientUtil.
createUsersession(“super”,“super”,
 “localhost”, 8080);
projectSpace.shareProject(usersession);
```

Listing 1

```
List<ProjectInfo> projectList =
workspace.getRemoteProjectList(usersession);
workspace.checkout(usersession, projectList.get(0));
```

Listing 2

EMFStore-Setup

Für ein einfaches Test-Setup wird der EMFStore in eine Eclipse-Modeling-Edition-Instanz installiert. Im produktiven Einsatz kann der EMFStore natürlich auch „headless“ laufen. Nach dem Download der aktuellen Eclipse-Modeling-Edition [2] kann der EMFStore von der Update-Site [3] installiert werden. Für ein einfaches Setup installiert man zunächst alle Features. Die Installation enthält damit auch gleich die generische Oberfläche der EMF Client Platform [4]. Nach der Installation wird Eclipse neu gestartet. Um den EMFStore zu testen, wird nun noch ein Modell benötigt. Dazu kann entweder ein eigenes Modell verwendet werden oder kann man alternativ auch auf das Beispiel-Modell zurückgreifen (siehe [1]), das unter [8] heruntergeladen werden kann.

Um ein EMF-Modell mit dem EMFStore zu nutzen, sollte zunächst eine Änderung am Generator-Modell vorgenommen werden, die eine effiziente Speicherung der Entitäten erlaubt. Dazu öffnet man im Modell-Plug-in die Datei „*.genmodel“ aus dem Ordner „model“ und klickt den Wurzel-Knoten doppelt. Es öffnet sich die Properties-View. In dieser ändert man den Wert von „Containment Proxies“ von „false“ nach „true“. Nun kann der Modell-Code neu generiert werden mit Rechtsklick auf den Wurzelknoten im Generatormodel und durch Auswahl von „Generate Model Code“ und „Generate Edit Code“.

Als letzten Schritt müssen der EMFStore-Server und ein entsprechender Client mit dem erstellten Modell gestartet werden. Dazu legt man zunächst zwei neue Launch-Konfigurationen unter „Eclipse-Application“ an. Man belässt alle Einstellungen außer dem Namen auf den Standardwerten und ändert für den Server den Wert „Run as a product“ auf dem Reiter „Main“ nach „org.unicase.emfstore.server“. Durch Ausführen der Launch-Konfigurationen können nun eine Client-und die Server-Applikation gestartet werden. Die gestartete Client-Instanz führt alle anfangs genannten Anwendungsfälle bereits aus. Ein Video-Tutorial dazu sowie zu den ersten Schritten mit dem EMFStore ist online [5] zu finden.

Plug-ins für die VisualVM entwickeln: die MemoryPoolView

Kirk Pepperdine, Java Champion und Performance-Expert

Die VisualVM ist eine Desktop-Anwendung, die viele JDK-Tools unter einer grafischen Oberfläche vereinigt und ein reiches Toolset für die Performance-Analyse von Java-Anwendungen bietet. Eine der herausragenden Eigenschaften der VisualVM ist ihre Erweiterbarkeit. Um zu demonstrieren, wie leicht die VisualVM um zusätzliche Fähigkeiten erweitert werden kann, wird in diesem Artikel schrittweise ein Plug-in entwickelt. Dieses wird Informationen über den Code Cache der JVM aus einer MemoryMXBean gewinnen und diese grafisch in einer Zeitleiste darstellen.

Zum Hintergrund: Die „Just In Time (JIT)"-Kompilierung übersetzt Java-Byte-Code zur Laufzeit. Dadurch bekommt der Compiler die einmalige Gelegenheit, Informationen über die Laufzeitumgebung und die Dynamik der Anwendung in die Optimierung einfließen zu lassen. Dies führt zu einer massiven Beschleunigung der Anwendung, die zum Teil die Performance von entsprechendem C/C++ Code übertreffen kann. Der durch den JIT-Compiler erzeugte, native Code wird in einem Memory-Pool gespeichert, der als „Code-Cache" bekannt ist. Ist dieser voll, stellt der JIT-Compiler seine Arbeit ein. Wenn dies passiert, wird unsere Applikation langsamer und verbraucht mehr und mehr CPU-Zeit.

Bedenkt man die Wichtigkeit des Code-Cache für die Performance, ist es überraschend, dass es kaum Werkzeuge für dessen Überwachung gibt. Auf der anderen Seite ist die Nachfrage nach solchen Werkzeugen klein, weil vielen die wichtige Rolle des Code-Cache für die Performance der Anwendung nicht voll bewusst ist. Es ist ebenfalls überraschend, dass die JVM selbst nicht viel Einblick in diese geheimnisvollen internen Abläufe bietet. Immerhin wird der Code-Cache aber durch eine Instanz einer „java.lang.management.MemoryMXBean" überwacht. Diese gibt uns einen Einblick, wie viel von diesem Speicherpool durch die JIT-Kompilierung genutzt wird. Abbildung 1 zeigt die Daten, wie sie durch das „MBeans Browser VisualVM"-Plug-in angezeigt werden.

Das MBeans-Browser-Fenster bietet eine einfache, rein textbasierte Darstellung der Attribute des Memory-Pools. Für eine sinnvolle Auswertung der Daten wäre jedoch eine grafische Darstellung in Form einer Zeitleiste sinnvoll. Um dies zu verwirklichen, hat der Autor ein eigenes VisualVM-Plug-in entwickelt, das in Abbildung 2 dargestellt ist. Nachfolgend ist seine Erstellung beschrieben.

Ein VisualVM-Plug-in ist nichts anderes als ein NetBeans-Plug-in und damit ein spezielles JAR-Archiv. Zur Erstellung des

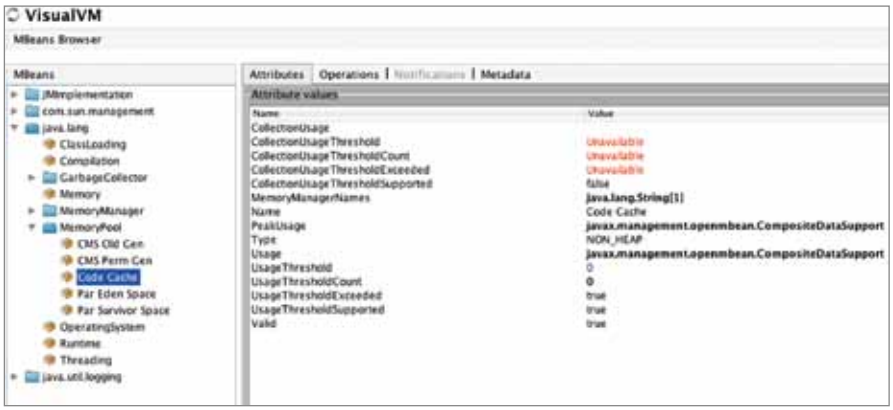


Abbildung 1: VisualVM MBean

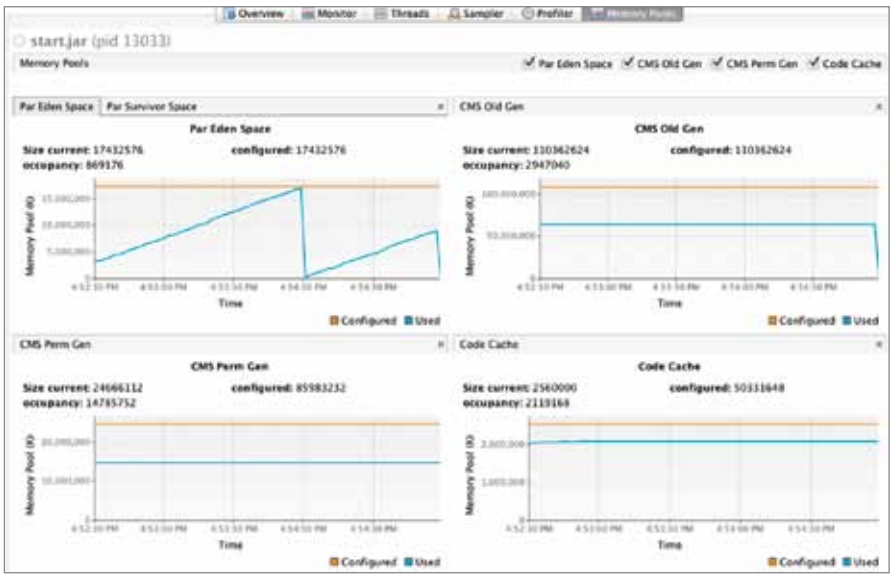


Abbildung 2: Memory-Pool-Visualisierung

```
public class Installer extends ModuleInstall {

    @Override
    public void restored() {
        MemoryPoolViewProvider.initialize();
    }

    @Override
    public void uninstalled() {
        MemoryPoolViewProvider.unregister();
    }

}
```

Listing 1: Erweiterter Quellcode

lus einer Anwendung reagieren. Auch unser MemoryPoolView benötigt einen eigenen Installer, um den eigenen Lebenszyklus zu verwalten. Der Installer wird am einfachsten mit dem Menüpunkt „New –> Module Installer" erstellt. Der Installer wird dabei auch automatisch im Manifest des Moduls registriert, sodass er von der Laufzeitumgebung gefunden wird. Listing 1 zeigt, wie wir den generierten Quellcode erweitern.

Der Installer überschreibt die zwei wichtigsten Methoden „restored()" und „uninstalled()". Diese Methoden werden

```
class MemoryPoolViewProvider extends DataSourceViewProvider<Application> {

    private static DataSourceViewProvider<Application> instance=new MemoryPoolViewProvider();

    @Override
    public boolean supportsViewFor(final Application application) {
        return true;
    }

    @Override
    public synchronized DataSourceView createView(final Application application) {
        return new MemoryPoolView(application);
    }

    static void initialize() {
        DataSourceViewsManager.sharedInstance().addViewProvider(instance,Application.class);
    }

    static void unregister() {
        DataSourceViewsManager.sharedInstance().removeViewProvider(instance);
    }
}
```

Listing 2: DataSourceViewProvider

Plug-ins kommt die NetBeans-IDE zum Einsatz. Man könnte theoretisch auch andere Entwicklungstools verwenden, die NetBeans-IDE bietet jedoch Vorlagen für das Erstellen und Testen der Erweiterung und erleichtert so die Entwicklung. Zunächst muss dazu die VisualVM (hier Version 1.3.3) beim Plattform-Manager als Anwendungsplattform registriert werden. Als Nächstes erzeugt man das neue NetBeans-Modul-Projekt namens „MemoryPoolView". Im Projektassistenten muss dabei die neu registrierte VisualVM-Plattform ausgewählt werden. Sobald das erledigt ist, kann die Implementierung beginnen.

Der Lebenszyklus eines NetBeans-Moduls

NetBeans-Module können mit einem Module-Installer auf Ereignisse im Lebenszyklus beim Start und beim Beenden der Anwendung aufgerufen. In der Implementierung von „restored()"wird der MemoryPoolViewProvider beim DataSourceViewsManager registriert. Der DataSourceViewsManager verwaltet DataSourceView-Instanzen. Daher muss der MemoryPoolViewProvider diese Klasse erweitern.

Für ein besseres Verständnis dafür, wie das funktioniert, ist es wichtig, das Konzept der DataSources in der VisualVM zu verstehen. Wie der Name vermuten lässt, füttert eine DataSource die VisualVM mit (Performance-)Daten. VisualVM wird bereits mit einer Reihe von fertigen Datenquellen ausgeliefert, darunter „SnapShot", „Host" und, für uns wichtig, „Application". Wenn der Anwender eine DataSource öffnet, fragt die VisualVM den DataSourceViewsManager nach Views, die für die Darstellung von Daten aus dieser Quelle registriert sind. Daraufhin fragt der DataSourceViewsManager jede der Views, ob sie die Daten der in Frage kommenden DataSource darstellen kann. Alle ViewProvider, die das bejahen, bekommen die Datenquelle übergeben und liefern dafür eine View zurück. Die VisualVM erstellt für jede der Views einen neuen Reiter. Für die Darstellung der Daten ist von diesem Punkt an der ViewProvider zuständig.

Listing 2 zeigt den Quellcode, den der ViewProvider in Abhängigkeit vom Lebenszyklus registriert und deregistriert. Die Methoden „supportsViewFor()" und „createView()" werden bei Bedarf vom DataSourceViewsManager aufgerufen. Bei der „supportsViewFor()"-Methode wird ein Application-Objekt übergeben. In unserem Fall geben wir als Antwort einfach „true" zurück, da wir alle Arten von Anwendungen unterstützen. Für andere Zwecke können wir hier gegebenenfalls weitere Tests durchführen, um zu entscheiden, ob wir eine View anbieten möchten.

Nachdem der Provider implementiert ist, wird die View gebaut und mit der Applikation verbunden. Dabei sind zwei Themen wichtig:

- Wie sind Views aufgebaut, die in der VisualVM dargestellt werden können?
- Wie kann man die benötigten Daten vom Application-Objekt erhalten?

Wir beginnen mit der DataSourceView, die vom Aufruf „MemoryPoolViewProvider.createView()" zurückgeliefert werden muss.

Die Anatomie einer VisualVM-View

Abbildung 2 zeigt, dass jede einzelne View in einem eigenen Reiter enthalten ist. Jeder dieser Reiter hat einen Titel und ein Icon. Darüber hinaus sieht das Layout jeder View weitestgehend frei aus. Obwohl alle Views daraufhin optimiert werden können, die dargestellte Information optimal darzustellen, gibt es einige Layout-Vorgaben.

Der Inhalt des Panels ist in einen Hauptbereich oben und vier Slots für die Darstellung von DetailViews unterteilt. Der Hauptbereich wird üblicherweise dafür verwendet, Einstellungen vorzunehmen, während die Daten selbst in den vier übrigen Darstellungsbereichen präsentiert

```
class MemoryPoolView extends DataSourceView {

  public MemoryPoolView(Application application) {
    super(application,"Memory Pools",
                      new ImageIcon(ImageUtilities.loadImage(IMAGE_PATH, true)).getImage(),
                      60, false);
  }
  @Override
  protected DataViewComponent createComponent() {
    //Data area for master view:
    JEditorPane generalDataArea = new JEditorPane();
    generalDataArea.setBorder(BorderFactory.createEmptyBorder(7, 8, 7, 8));

    //Master view:
    DataViewComponent.MasterView masterView =
            new DataViewComponent.MasterView(„Memory Pools“, „View of Memory Pools“,
            generalDataArea);

    //Configuration of master view:
    DataViewComponent.MasterViewConfiguration masterConfiguration =
            new DataViewComponent.MasterViewConfiguration(false);

    //Add the master view and configuration view to the component:
    dvc = new DataViewComponent(masterView, masterConfiguration);

    // the magic that gets a handle on all instances of MemoryPoolMXBean
    findMemoryPools();

    MemoryPoolPanel panel;
    for ( MemoryPoolModel model : models) {
      panel = new MemoryPoolPanel(model.getName());
      model.registerView(panel);
      Point position = calculatePosition(model.getName());
      dvc.addDetailsView(new DataViewComponent.DetailsView(
              model.getName(), „memory pool metrics“, position.y, panel,
                                                          null), position.x);
    }
    return dvc;
  }
  private Point calculatePosition(String name) {
    return positions.get(name);
  }

  static {
    positions = new HashMap<String,Point>();
    positions.put( „Par Eden Space“, new Point(DataViewComponent.TOP_LEFT,10));
    positions.put( „PS Eden Space“, new Point(DataViewComponent.TOP_LEFT,10));
    positions.put( „Eden Space“, new Point(DataViewComponent.TOP_LEFT,10));
    positions.put( „G1 Eden“, new Point(DataViewComponent.TOP_LEFT,10));
    positions.put( „Par Survivor Space“, new Point(DataViewComponent.TOP_LEFT,20));
    positions.put( „PS Survivor Space“, new Point(DataViewComponent.TOP_LEFT,20));
    positions.put( „Survivor Space“, new Point(DataViewComponent.TOP_LEFT,20));
    positions.put( „G1 Survivor“, new Point(DataViewComponent.TOP_LEFT,20));
    positions.put( „CMS Old Gen“, new Point(DataViewComponent.TOP_RIGHT,10));
    positions.put( „PS Old Gen“, new Point(DataViewComponent.TOP_RIGHT,10));
    positions.put( „Tenured Gen“, new Point(DataViewComponent.TOP_RIGHT,10));
    positions.put( „G1 Old Gen“, new Point(DataViewComponent.TOP_RIGHT,10));
    positions.put( „CMS Perm Gen“, new Point(DataViewComponent.BOTTOM_LEFT,10));
    positions.put( „Perm Gen“, new Point(DataViewComponent.BOTTOM_LEFT,10));
    positions.put( „PS Perm Gen“, new Point(DataViewComponent.BOTTOM_LEFT,10));
    positions.put( „G1 Perm Gen“, new Point(DataViewComponent.BOTTOM_LEFT,10));
    positions.put( „Code Cache“, new Point(DataViewComponent.BOTTOM_RIGHT,10));
  }
```

Listing 3: MemoryPoolView

werden. In der MemoryPoolView enthält jeder Quadrant die Darstellung eines Memory-Pools. Unten rechts wird eine Zeitleiste der Code-Cache-Zähler angezeigt. Auch in der MonitorView sind alle vier Felder des Rasters belegt.

In unserem Hauptdarstellungsbereich sind vier Checkboxen angeordnet. Vielleicht hat mancher diese (optional angezeigten) Checkboxen bereits in anderen Views bemerkt. Das Threads-Plug-in zum Beispiel enthält zwei Checkboxen, eine für jeden der zwei enthaltenen Views. Diese Checkboxen kontrollieren, ob die zugeordnete View dargestellt wird oder nicht.

Der Titel und das Icon von MemoryPoolView werden im Konstruktor übergeben. Der ViewProvider gibt in seiner „createView()“-Methode eine DataSourceView zurück. Daher muss MemoryPoolView diese Klasse erweitern. MasterView und DetailsView werden in einem Aufruf der createComponent() erzeugt, wie in Listing 3 gezeigt.

Das letzte visuelle Detail ist die Positionierung. Eines der Argumente im Konstruktor ist die magische Zahl „60“. VisualVM sortiert die Reiter nach den Benutzereinstellungen. Die magische Nummer schlägt die Position „60“ vor. Das heißt, dass jeder Tab mit einer kleineren magischen Rückfallposition links davon positioniert wird; Views mit einer größeren Nummer landen rechts von unserem Tab. Um eine DetailsView zu positionieren, sind ein Quadrant und ein Slot anzugeben. In der DataViewComponent werden zu diesem Zweck die Konstanten „TOP_LEFT“, „BOTTOM_RIGHT“ etc. angeboten. Die Position wird verwendet, wenn zwei DetailsViews im selben Quadranten liegen. Abbildung 2 zeigt, dass die Young Generational Spaces beide in „TOP_LEFT“ platziert wurden. Eden Space hat die Position „10“ und der Survivor Space die Position „20“. All das wird in MemoryPoolView konfiguriert (siehe Listing 3).

Die Anbindung der Daten

Nachdem das Basislayout eingerichtet ist, werden die darzustellenden Daten bereitgestellt. Genau wie für den Aufbau der Views unterstützt die VisualVM bei der Daten-Akquise und -Behandlung. In unserem Fall brauchen wir einen Handle für alle Instanzen von MemoryPoolMXBean.

```
protected void findMemoryPools() {
   try {
      MBeanServerConnection conn = getMBeanServerConnection();
      ObjectName pattern = new ObjectName("java.lang:type=MemoryPool,name=*");
      for (ObjectName name : conn.queryNames(pattern, null)) {
         initializeModel(name, conn);
      }
   } catch (Exception e) {
         LOGGER.throwing(MemoryPoolView.class.getName(),"Exception:", e);
   }
}

private MBeanServerConnection getMBeanServerConnection() {
   JmxModel jmx = JmxModelFactory.getJmxModelFor((Application)super.getDataSource());
   return jmx == null ? null : jmx.getMBeanServerConnection();
}
```

Listing 4: Ein Handle für alle MemoryMXBean-Instanzen

```
public class MemoryPoolPanel extends JPanel implements MemoryPoolModelListener {

   private SimpleXYChartSupport chart;

   public MemoryPoolPanel(String name) {
      setLayout(new BorderLayout());
      SimpleXYChartDescriptor description = SimpleXYChartDescriptor.bytes(0,
      20, 100, false, 1000);
      description.setChartTitle(name);
      description.setDetailsItems(new String[3]);

      description.addLineItems("Configured");
      description.addLineItems("Used");

      description.setDetailsItems(new String[]{"Size current","configured",
                                    "occupancy"});
      description.setXAxisDescription("<html>Time</html>");
      description.setYAxisDescription("<html>Memory Pool (K)</html>");

      chart = ChartFactory.createSimpleXYChart(description);

      add(chart.getChart(),BorderLayout.CENTER);

   }

   @Override
   public void memoryPoolUpdated(MemoryPoolModel model) {
      long[] dataPoints = new long[2];
      dataPoints[0] = model.getCommitted();
      dataPoints[1] = model.getUsed();
      chart.addValues(System.currentTimeMillis(), dataPoints);

      String[] details = new String[3];
      details[0] = Long.toString(model.getCommitted());
      details[1] = Long.toString(model.getMax());
      details[2] = Long.toString(model.getUsed());
      chart.updateDetails(details);
   }
}
```

Listing 5: MemoryPoolPanel

VisualVM ermöglicht den Zugriff ohne die Notwendigkeit, komplexen JMX Clientcode zu erstellen. Listing 4 zeigt, wie wir die „findMemoryPools()"-Methode unserer MemoryPoolView implementieren.

Die Handles für MXBeans liefert eine MBeanServerConnection. Die Serververbindung selbst stammt von einem JmxModel, das wiederum von der JmxModelFactory kommt. Diese gesamte Infrastruktur wird von der VisualVM bereitgestellt. Es sind lediglich die Kriterien zu definieren, nach denen die Server-Verbindung durchsucht werden soll. Das erfolgt über ein Objektnamens-Pattern. Abbildung 1 zeigt, dass der Objektname, der verwendet wird, um Instanzen von MemoryPoolMXBean zu binden, in etwa so aussieht: „java.lang:type=MemoryPool,name=»CMS Old Gen»". Das Namens-Attribut können wir dabei mit Wildcards anpassen, um alle Speicherpools zu finden. Jetzt müssen wir nur noch die Datenpunkte aus den erhaltenen Beans abfragen, um diese anzuzeigen.

Charting mit VisualVM

Abbildung 2 zeigt die Daten der MXBean als XY-Plot. Wenn das Plug-in läuft, wird der Graph alle zwei Sekunden aktualisiert. So komplex die Graphen auch aussehen, die mitgelieferten VisualVM Charts sind erstaunlich einfach zu verwenden.

MemoryPoolView verwendet von den drei verschiedenen Graph-Typen „Dezimal", „Prozent" und „Byte" den Byte-Typ. Wir bauen unser Chart unter Verwendung des vorgegebenen Builder-Patterns (siehe Listing 5).

Der Prozess startet durch die Erstellung einer Beschreibung unseres Charts. Die initialen Parameter sind „Minimalwert", „Maximalwert", „Anfangs-Y-Begrenzung", ein Bool-Wert, um festzulegen, ob das Chart versteckt werden kann (zeigt eine Checkbox in der MasterView an), sowie die Größe des Puffers, der die Datenpunkte enthält. Dieser Beschreibung fügen wir einen Titel und die Labels für die derzeitige Größe, Voreinstellung und Belegung sowie Labels für die X- und Y-Achse hinzu. Dann ergänzen wir noch eine Linie für die Belegung und derzeit konfigurierte Größe. Danach müssen wir lediglich noch die ChartFactory anweisen, unser Chart zu erstellen.

Im abschließenden Schritt wird das Model mit der View verbunden. Wir brauchen einen Timer, um das Chart in regelmäßigen

Abständen zu aktualisieren. Auch hier hilft uns VisualVM durch die Bereitstellung einer CachedMBeanServerConnection. Eine CachedMBeanServerConnection speichert die Werte aus einer MBeanServerConnection zwischen. Ebenso verfügt sie über einen Timer, der in festgelegten Intervallen das „Flushen" des Cache auslöst. Dies aktualisiert den Cache und benachrichtigt anschließend alle MBeanCacheListener über die Aktualisierung. Wir müssen lediglich die im Interface vorgegebene „flushed()"-Methode implementieren. Wenn flushed() aufgerufen wird, graben wir uns durch die CompositeData, die alle Datenwerte bündeln, die wir benötigen (siehe Listing 6).

Zum Schluss müssen wir uns noch um die Aktualisierung des Charts kümmern. Sobald der Cache aktualisiert wurde, meldet das Model der View die Verfügbarkeit frischer Daten. Die View kann diese Daten dann aus dem Model abfragen. Um Werte hinzuzufügen, müssen wir sie in ein „long[]" packen und sie dann dem Chart weitergeben. Die Details werden nach demselben Muster aktualisiert (siehe Listing 7).

Fazit

Die VisualVM bietet eine Menge Unterstützung bei der Visualisierung von Performance-Daten. Unser Beispiel deckt dabei nur einen kleinen Teil der verfügbaren Unterstützung ab. So bietet die VisualVM zum Beispiel auch Unterstützung beim Erstellen von Snapshots. Der hier vorgestellte MemoryPoolView ist unter http://java.net/projects/memorypoolview verfügbar.

Aus dem Amerikanischen übersetzt von Anton Epple, http://eppleton.de

Kirk Pepperdine
http://kirk.blog-city.com/

Kirk Pepperdine beschäftigte sich viele Jahre mit Hochleistungssystemen und verteilten Anwendungen. Er ist heute auf Java fokussiert mit dem Ziel, die Performance in jeder Phase des Projekt-Lebenszyklus zu verbessern. Aktuelles über Werkzeuge und Techniken zum Thema Java Performance Tuning vermittelt er in seinem Blog (http://kirk.blog-city.com) und auf der Webseite „Java Performance Tuning" (http://www.javaperformancetuning.com).

```
class MemoryPoolModel implements MBeanCacheListener {
   public MemoryPoolModel(final ObjectName mbeanName, final JmxModel model,
                                    final MBeanServerConnection mbeanServerConnection) throws
Exception {
      this.mbeanName = mbeanName;
      this.mbeanServerConnection = mbeanServerConnection;
      CachedMBeanServerConnectionFactory.getCachedMBeanServerConnection(model,
                  2000).addMBeanCacheListener(this);
      name = mbeanServerConnection.getAttribute(mbeanName,"Name").toString();
      type = mbeanServerConnection.getAttribute(mbeanName,"Type").toString();
   }

   @Override
   public void flushed() {
      try {
         CompositeData poolStatistics = (CompositeData)mbeanServerConnection.
          getAttribute(mbeanName,"Usage");
         if ( poolStatistics != null) {
            CompositeType compositeType = poolStatistics.getCompositeType();
            if ( compositeType != null) {
               Collection keys = compositeType.keySet();
               for ( String key : compositeType.keySet()) {
                  if ( key.equals("committed"))
                     this.committed = (Long)poolStatistics.get("committed");
                  else if ( key.equals("init"))
                     this.initial = (Long)poolStatistics.get("init");
                  else if ( key.equals("max"))
                     this.max = (Long)poolStatistics.get("max");
                  else if ( key.equals("used"))
                     this.used = (Long)poolStatistics.get("used");
                  else
                     LOGGER.warning("Unknown key:" + key);
               }
               tickleListeners();
            }
         }
      } catch (Throwable t) {
         LOGGER.throwing(MemoryPoolModel.class.getName(),"Exception recovering data from
          MemoryPoolMXBean ", t);
      }
   }
```

Listing 6: MBeanCacheListener

```
public void memoryPoolUpdated(MemoryPoolModel model) {
   long[] dataPoints = new long[2];
   dataPoints[0] = model.getCommitted();
   dataPoints[1] = model.getUsed();
   chart.addValues(System.currentTimeMillis(), dataPoints);

   String[] details = new String[3];
   details[0] = Long.toString(model.getCommitted());
   detailss[1] = Long.toString(model.getMax());
   details[2] = Long.toString(model.getUsed());
   chart.updateDetails(details);
}
```

Listing 7: Aktualisieren des Chart

Apache Camel Security – Payload Security

Dominik Schadow, Trivadis GmbH

Mit Apache Camel steht Java-Entwicklern ein mächtiges Open-Source-Integrations-Framework zur Verfügung. Es können unterschiedlichste Systeme, von Datenbanken über Messaging-Systeme bis hin zu Web-Services, über Routen integriert werden. Bei der Übertragung sensibler Daten steht der Entwickler dann vor der Aufgabe, diese entsprechend zu schützen. Gleichzeitig sollen auch nur berechtigte Benutzer überhaupt Zugriff auf bestimmte Routen erhalten. Der Artikel zeigt die verschiedenen Sicherungsmöglichkeiten von Camel-Routen auf.

Apache Camel [1] stellt zur Sicherung von Routen bereits out-of-the-box verschiedene Komponenten bereit und ist gleichzeitig durch weitere Security-Frameworks wie Apache Shiro und Spring Security erweiterbar. Die Sicherheit von Camel und der Camel-Routen lässt sich dabei grob in die folgenden vier Kategorien [2] einteilen:

- Configuration Security
- Endpoint Security (Component Security)
- Route Security
- Payload Security

Alle genannten Kategorien lassen sich beliebig miteinander kombinieren. Configuration Security und Endpoint Security sind auch außerhalb der Camel-Welt bekannt, der Fokus dieses Artikels liegt deshalb auf der Kategorie „Payload Security". Ein Folgeartikel widmet sich dann der Route Security. Bei diesen beiden Kategorien sind im Apache-Camel-Umfeld einige Besonderheiten vorhanden. Die beiden anderen Kategorien werden der Vollständigkeit halber hier nur kurz beschrieben.

- *Configuration Security*
  Hierunter fällt das Sichern sensibler Bereiche der Camel-Konfiguration, etwa das von Passwörtern in Property-Dateien oder die Konfiguration der Transport Layer Security (TLS) in Camel. Bei Passwörtern funktioniert dies ähnlich wie in Spring mit Jasypt [3], bei TLS hilft die Verwendung der JSSE Utility [4].

- *Endpoint Security*
  Die Endpunkte wie CXF oder auch Jetty können auf verschiedene Arten gesichert werden, beispielsweise durch die Verwendung von Interzeptoren oder durch andere, von der Komponente angebotenen Möglichkeiten. Aus diesem Grund wird Endpoint Security oft auch als „Component Security" bezeichnet.
- *Route Security*
  Hier dreht sich alles um die Authentifizierung und Autorisierung. Verwendet werden können dabei zwei bekannte Frameworks: Apache Shiro und Spring Security. Mehr zur Route Security erfahren Sie in der nächsten Ausgabe.
- *Payload Security*
  Diese Kategorie kümmert sich um die Nachrichtensicherheit, also die Verschlüsselung oder Signierung der Nutzdaten. Hierfür stehen zwei Camel-Datenformate zur Verfügung: das XML-Security-Data-Format [5] und das allgemeinere Crypto-Data-Format [6].

Das Beispielprojekt

Camel-Projekte, ob gesichert oder nicht, sind dank der Maven Camel Archetypes [7] sehr schnell und einfach aufgesetzt. Als Ausgangsbasis kommt hier der Archetyp „camel-archetype-spring" zum Einsatz. Ziel des Beispielprojekts ist es, eine vorhandene und völlig ungesicherte Camel-Route (findUserData) auf ausgewählten Abschnitten der Route zu sichern (siehe Abbildung 1).

Gesichert werden die Daten immer nach der „UserDataBean", vor dem Aufruf des Web-Service „CategoryEndpoint". Das bedeutet, dass der Request an diesen Web-Service auf unterschiedliche Arten gesichert ist, die Response der Übersichtlichkeit halber hingegen nicht. Diese könnte genauso wie der Request gesichert werden, mit entsprechend vertauschten Ver- und Entschlüsselungs-Rollen.

Im Beispiel sollen möglichst viele unterschiedliche Sicherungsvarianten von Camel zum Einsatz kommen, was in der Praxis so sicherlich nicht gemacht wird. Das Beispiel verfügt dazu über einen CXF-Endpoint !cxf:bean:userDataEndpoint!, der zur Unterscheidung der unterschiedlich gesicherten Routen am Ende über sechs verschiedene Operationen verfügt. Ausgangsbasis ist die ungesicherte Web-Ser-

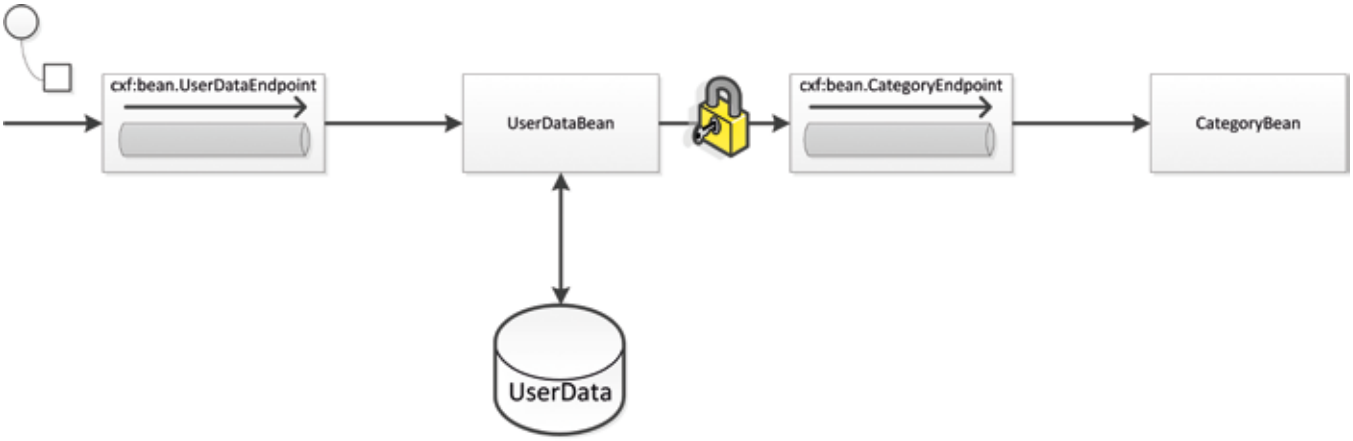


Abbildung 1: Die Camel-Route des Beispielprojekts im Überblick

vice-Operation „findUserData(int ssn)", die zu einer gegebenen Sozialversicherungsnummer die Benutzerdaten zurückliefert.

In einem normalen Projekt würde man diese Route entsprechend direkt absichern. Damit der Unterschied zwischen den verschiedenen Sicherungsmethoden deutlicher wird, ist das Beispiel um weitere Methoden ergänzt. Ein an die Operation angehängtes „Enc", zum Beispiel bei „findUserDataAsymEncXML()", liefert die Daten asymmetrisch verschlüsselt zurück. Im Fall von XML im XMLSecurity-Data-Format, andernfalls im Crypto-Data-Format.

Die Endung „Sign", also bei „findUserDataSign()", steht entsprechend für eine Route mit digital signierter Nachricht. Der Web-Service umfasst so am Ende sechs Operationen: die ungesicherte, vier verschlüsselte (zwei herkömmlich und zwei mit XML, jeweils symmetrisch und asymmetrisch verschlüsselt) sowie eine digital signierte Operation.

Die folgenden Listings zeigen nur Ausschnitte des Source-Codes. Das vollständige Beispielprojekt steht auf GitHub unter https://github.com/dschadow/CamelSecurity im SpringSource-Tool-Suite-Projekt „CamelPayloadSecurity".

Das XMLSecurity-Data-Format

Der große Vorteil des XMLSecurity-Data-Formats (und der XML-Encryption allgemein) besteht darin, dass neben der vollständigen Nachricht auch nur einzelne Elemente oder Elementinhalte gezielt verschlüsselt werden können. Hierbei bestimmen XPath-Ausdrücke, welches Element, beziehungsweise Elemente oder auch welcher Element-Inhalt mit einbezogen werden sollen. Gleichzeitig lassen sich so auch unterschiedliche Fragmente mit unterschiedlichen Schlüsseln und/oder Algorithmen sichern, sodass ein Empfänger nur einzelne für ihn verschlüsselte Bestandteile der Nachricht wieder zu Klartext entschlüsseln kann. Lediglich das Verschlüsseln von Attributen ist nicht möglich.

Diese Eigenschaften sind keine Erfindung von Apache Camel. Stattdessen wird hier auf die W3C-Empfehlung zur XML-Encryption [8] zurückgegriffen (siehe Kasten „Unsichere XML-Verschlüsselung"). Die Implementierung dieser Empfehlung stammt vom Apache-XML-Security-Projekt „Santuario" [9].

Camel selbst bietet dabei nur eine Untermenge der XML-Security-Möglichkeiten an, XML-Signatures sind (derzeit?) nicht nativ enthalten (siehe Kasten „Asymmetrische und symmetrische XML-Verschlüsselung mit Camel"). Apache Santuario selbst unterstützt aber auch die XML-Signatures, eine selbstentwickelte Integration in eine Route ist also denkbar.

Um die XML-Security-Features in Camel nutzen zu können, muss das Artefakt

Vorsicht geboten: Unsichere XML-Verschlüsselung

Im Oktober 2011 meldete die Ruhr-Universität Bochum [15], dass der XML-Encryption-Standard erfolgreich gebrochen werden konnte. Da hierbei die W3C-Empfehlung selbst, und nicht nur eine Implementierung, für Probleme sorgt, ist kein einfacher Workaround möglich.

Allerdings werden mit der sich zurzeit in der Verabschiedung befindlichen XML-Encryption-1.1-Empfehlung gezielt diese Probleme angegangen. Wer sich jetzt in Projekten für die XML-Sicherheit entscheidet beziehungsweise entscheiden muss, sollte daher, sobald entsprechende aktualisierte Implementierungen vorliegen, den Wechsel angehen.

Asymmetrische und symmetrische XML-Verschlüsselung mit Camel

Bis einschließlich Version 2.8 unterstützt Camel nur die symmetrische Verschlüsselung innerhalb des XMLSecurity-Data-Formats. Zur Verfügung stehen dabei die Algorithmen „Triple-DES" und „AES" (128, 192 und 256 Bit Länge). Seit Version 2.9 kann auch eine asymmetrische Verschlüsselung verwendet werden. Dabei wird, wie üblich, ein symmetrischer Schlüssel (ein sogenannter „Content Encryption Key") verwendet. Dieser wird anschließend mit dem öffentlichen Schlüssel des Empfängers verschlüsselt (also ein Key Encryption Key) und so zusammen mit der verschlüsselten Nachricht an den Empfänger übertragen. Für die asymmetrischen Verfahren stehen dabei die Algorithmen RSA 1.5 (RSA_v1dot5) sowie RSA OAEP (RSA_OAEP) zur Verfügung.

```
<dependency>
 <groupId>org.apache.camel</groupId>
 <artifactId>camel-xmlsecurity</artifactId>
 <version>2.9.0</version>
</dependency>
```

Listing 1

```
<route id="findUserDataSymEncXML">
 <from uri="direct:findUserDataSymEncXML" />
 <marshal>
  <secureXML secureTagContents="true"
   secureTag="/userData/socialSecurityNumber"
   xmlCipherAlgorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc"
   passPhrase="My own really secret key" />
 </marshal>
</route>
```

Listing 2

Attribut	Default	Beschreibung
secureTag	null	XPath des zu ver-/entschlüsselnden Elements, „null" verschlüsselt die vollständige Nachricht.
secureTagContents	false	„true" verschlüsselt nur den Elementinhalt, „false" das komplette Element. Hat nur Auswirkungen bei Verwendung des secureTag-Attributs.
passPhrase	null	Passphrase für die Ver-/Entschlüsselung. Ohne Angabe wird die Default–Pass-Phrase verwendet. Die Pass-Phrase muss zum Algorithmus und dessen Schlüssellänge passen.
xmlCipherAlgorithm	TRIPLEDES	Einer der folgenden symmetrischen Verschlüsselungsalgorithmen für den Nachrichteninhalt: XMLCipher.TRIPLEDES (http://www.w3.org/2001/04/xmlenc#tripledes-cbc), XMLCipher.AES_128 (http://www.w3.org/2001/04/xmlenc#aes128-cbc), XMLCipher.AES_192 (http://www.w3.org/2001/04/xmlenc#aes192-cbc), XMLCipher.AES_256 (http://www.w3.org/2001/04/xmlenc#aes256-cbc)
namespaces	none	Namespace(s) verwendet im secureTag-Attribut.

Tabelle 1: XML-Encryption-Attribute (symmetrisch/ asymmetrisch)

Attribut	Default	Beschreibung
recipientKeyAlias	none	Schlüssel-Alias zur Identifikation im Schlüsselspeicher.
keyCipherAlgorithm	none	Asymmetrischer Algorithmus zur Ver-/Entschlüsselung des Schlüssels aus folgender Liste: XMLCipher.RSA_v1dot5 (http://www.w3.org/2001/04/xmlenc#rsa–1_5), XMLCipher.RSA_OAEP (http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p)
keyOrTrustStoreParameters	none	Verweist auf das KeyStoreParameters-Element mit den notwendigen Key-/TrustStore-Informationen.

Tabelle 2: XML-Encryption-Attribute (asymmetrisch)

„camel-xmlsecurity" zur Liste der Abhängigkeiten in der Maven-POM-Datei hinzugefügt werden (siehe Listing 1).

Symmetrische Verschlüsselung

Die Verschlüsselung mit dem XMLSecurity-Data-Format ist Camel-typisch sehr einfach, vor allem bei der Verwendung von symmetrischer Kryptografie. Durch die Verwendung der von Camel gesetzten Standardwerte (Algorithmus „Triple-DES", Pass-Phrase „Just another 24 Byte key", Verschlüsselung der kompletten Nachricht) sieht die einfachste symmetrische Form an einer beliebigen Stelle der Route wie folgt aus:

```
<marshal>
 <secureXML />
</marshal>
```

In der Realität sollte zumindest ein eigenes Passwort gesetzt sein. Noch komplexer wird die Route, wenn weitere Attribute vom Standard abweichen und beispielsweise nur ein bestimmtes Element verschlüsselt werden soll (siehe Tabelle 1 und Listing 2).

Natürlich kann man die Nutzdaten an jedem beliebigen Punkt der Route ver- und entschlüsseln. So lassen sich auch unterschiedliche Routenabschnitte oder Nachrichtenteile mit verschiedenen Schlüsseln oder Algorithmen sichern. Nur der jeweilige Empfänger kann dann sein Fragment entschlüsseln und die Daten verarbeiten. Egal an welchem Punkt der Route, Listing 3 zeigt, wie die Daten entsprechend entschlüsselt werden.

Wie man sieht, muss das Attribut „xmlCipherAlgorithm" auch bei der Entschlüsselung angegeben werden. Andernfalls wird der Camel-Standard-Algorithmus (TripleDES) verwendet und nicht etwa das Attribut „algorithm" aus dem „EncryptedData"-Block geladen. Der Entschlüsselungsversuch des Beispiels (Route „encryptSymmetric") würde dann mit einer Fehlermeldung abbrechen.

Asymmetrische Verschlüsselung

In der Praxis wird man häufiger auf die asymmetrische Kryptografie zurückgreifen, allein schon wegen des deutlich einfacheren Schlüsselaustausches mit den beteiligten Partnern. Hier müssen dann zwei Algorithmen angegeben werden:

„xmlCipherAlgorithm“ zur symmetrischen Verschlüsselung der Nutzdaten (Content Encryption Key) und „keyCipherAlgorithm“ zur asymmetrischen Verschlüsselung des symmetrischen Schlüssels (Key Encryption Key) aus dem Schritt zuvor. Die asymmetrische Verschlüsselung ist damit in Wahrheit eine hybride Verschlüsselungsform. Wie Listing 4 zeigt, sind dabei auch einige Konfigurationsparameter mehr notwendig.

Wichtig ist die Verknüpfung der sogenannten „keyOrTrustStoreParametersId“ mit dem „keyStoreParameters“-Element innerhalb des Camel-Kontexts. Dieses Element konfiguriert den Trust-/KeyStore, der den öffentlichen Schlüssel des Empfängers enthält (bei der Entschlüsselung entsprechend den privaten Schlüssel). Im Beispiel ist dies ein gewöhnlicher Java-KeyStore, selbstgeneriert mit dem Java-KeyTool. Dabei sind sowohl der private als auch der öffentliche Schlüssel im selben KeyStore gespeichert. In der Praxis würde man diese beiden Schlüssel in unterschiedlichen KeyStores speichern und mit verschiedenen Passwörtern sichern, beziehungsweise würde man bei fremden Empfängern ohnehin nur deren öffentlichen Schlüssel besitzen.

Auf Empfänger-Seite muss dann nur der zugehörige private Schlüssel vorhanden sein, damit die Entschlüsselung mit „unmarshall“ durchgeführt werden kann (siehe Listing 5). Auch hier ist das (selbe) „keyStoreParameters“-Element zwingend erforderlich.

Typische Fehler und Probleme, die auftreten können: Wird für die Verschlüsselung ein nicht vorhandener Algorithmus angegeben, lautet die Fehlermeldung: „java.lang.NullPointerException: Transformation unexpectedly null…“. Vor allem bei der XML-basierten Konfiguration (Spring) gilt es zu beachten, dass als Algorithmus nicht etwa XMLCipher.AES_128, sondern http://www.w3.org/2001/04/xmlenc#aes128-cbc angegeben werden muss.

```
<unmarshal>
 <secureXML secureTag="/userData/socialSecurityNumber"
  xmlCipherAlgorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc"
  passPhrase="My own really secret key" />
</unmarshal>
```

Listing 3

```
<keyStoreParameters id="keyStore" resource="./camel.jks" password="camelsec" />

<route id="findUserDataAsymEncXML">
 <from uri="direct:findUserDataAsymEncXML" />
 <marshal>
  <secureXML secureTagContents="true"
   secureTag="/userData/socialSecurityNumber"
   xmlCipherAlgorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc"
   keyCipherAlgorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"
   recipientKeyAlias="rsaCamelSec"
   keyOrTrustStoreParametersId="keyStore" />
  </marshal>
 </route>
```

Listing 4

```
<unmarshal>
 <secureXML secureTag="/userData/socialSecurityNumber"
  xmlCipherAlgorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc"
  keyCipherAlgorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"
  recipientKeyAlias="rsaCamelSec"
  keyOrTrustStoreParametersId="keyStore" />
</unmarshal>
```

Listing 5

```
<dependency>
 <groupId>org.apache.camel</groupId>
 <artifactId>camel-crypto</artifactId>
 <version>2.9.0</version>
</dependency>
```

Listing 6

Das Crypto-Data-Format

Auch das Crypto-Data-Format kümmert sich um die Verschlüsselung der Nutzdaten. Unterstützt wird dabei jedes beliebige Datenformat, natürlich auch XML. Verschlüsselt wird daher auch immer die vollständige Nachricht mit einem Schlüssel, eine Teilverschlüsselung ist nicht möglich. Dafür stehen aber auch deutlich mehr Algorithmen und Crypto-Parameter zur Verfügung, schließlich integriert das Crypto-Data-Format die „Java Cryptography Extension“ in Camel. Auf Systemen mit „JCE Unlimited Strength Jurisdiction Policy Files“ [10] oder gar „BouncyCastle“ [11] Crypto-Provider stehen weitere und nochmals deutlich stärkere Algorithmen zur Verfügung. Um die Crypto-Features verwenden zu können, müssen diese ebenfalls über ein weiteres Artefakt zur Liste der Maven-Abhängigkeiten hinzugefügt werden (siehe Listing 6).

Wie beim XMLSecurity-Data-Format findet auch beim Crypto-Data-Format die Verschlüsselung mit „marshal“, die Entschlüsselung mit „unmarshal“ statt. Neu mit Version 2.9 ist die Unterstützung für das weit verbreitete PGP und damit für asymmetrische Kryptografie.

Symmetrische Verschlüsselung

Grundsätzlich kann der symmetrische Schlüssel per Routen-Konfiguration in XML oder Java angegeben oder aber im

```
<dataFormats>
 <crypto id="des" algorithm="DES" keyRef="desKey" />
 <crypto id="aes" algorithm="AES" keyRef="aesKey" />
</dataFormats>

<route id="findUserDataSymEnc">
 <from uri="direct:findUserDataSymEnc" />
 <marshal ref="aes" />
</route>
```

Listing 7

```
<bean id="desKey"
 class="com.trivadis.bds.camel.security.crypto.KeyGeneration"
 factory-method="getDesKey" />

<bean id="aesKey"
 class="com.trivadis.bds.camel.security.crypto.KeyGeneration"
 factory-method="getAesKey" />
```

Listing 8

```
import java.security.Key;
import java.security.NoSuchAlgorithmException;
import javax.crypto.KeyGenerator;

public class KeyGeneration {
 private static Key desKey = null;
 private static Key aesKey = null;

 static {
  KeyGenerator generator;
  try {
   generator = KeyGenerator.getInstance(„DES");
   desKey = generator.generateKey();
   generator = KeyGenerator.getInstance(„AES");
   aesKey = generator.generateKey();
  } catch (NoSuchAlgorithmException e) {
   e.printStackTrace();
  }
 }

 public static Key getDesKey() {
  return desKey;
 }

 public static Key getAesKey() {
  return aesKey;
 }
}
```

Listing 9

```
<marshal ref="asymEncrypt" />
<unmarshal ref="asymDecrypt" />
```

Listing 10

Header der Nachricht unter dem Namen „CamelCryptoKey" abgelegt werden. Letzteres entspricht dabei in etwa einem verschlossenen Tresor, dessen Schlüssel man auf der Rückseite angebracht hat. Listing 7 zeigt daher die sicherere Variante per Konfiguration.

Wie man sieht, ist hier die Konfiguration etwas aufwändiger: Neben der Route muss noch ein Crypto-Data-Format innerhalb des Camel-Kontexts konfiguriert werden. Dieses verweist per „keyRef"-Attribut auf eine Bean, aus der der entsprechende Schlüssel geladen (oder generiert) wird. Die Konfiguration dieser Bean(s) zeigt Listing 8.

Listing 9 zeigt die Bean selbst, wobei die Schlüssel der Einfachheit halber nicht gespeichert, sondern beim Start einmalig generiert werden.

Es fällt auf, dass pro verwendetem Algorithmus eine solche Bean angelegt beziehungsweise eine allgemeine KeyGenerator-Bean um weitere Methoden zur Schlüsselerzeugung erweitert werden muss. Auch die Konfiguration muss dann um entsprechende „crypto"-Elemente ergänzt werden. Einmal angelegt, lassen sich Crypto-Data-Format-Elemente aber nach Belieben wiederverwenden. Die Entschlüsselung funktioniert erwartungsgemäß einfach mit dem Aufruf von „<unmarshal ref="aes" />". Das Attribut „ref" verweist dabei ebenfalls auf das jeweilige Crypto-Data-Format-Element.

Asymmetrische Verschlüsselung

Asymmetrisch verschlüsselt wird, wie bereits kurz erwähnt, mit PGP. Dazu generiert man sich beispielsweise mit „GnuPG" ein neues Schlüsselpaar und stellt Camel den öffentlichen zum Ver- und den privaten Schlüssel zum Entschlüsseln zur Verfügung. In Camel muss dabei kaum noch etwas konfiguriert werden. Nach Angabe der Dateien, der Schlüssel-Benutzer-ID und des Passworts (nur beim privaten Schlüssel) extrahiert Camel die weiteren Informationen wie Algorithmus und Schlüssellänge automatisch aus dem gewählten Schlüssel. Die einzige Änderung in obiger symmetrischer Route ist die Anpassung der „ref"-Attribute in den „marshal"- und „unmarshal"-Elementen (siehe Listing 10).

Außerdem muss das „dataFormats"-Element um die Einträge zur Ver- und Entschlüsselung mit PGP erweitert werden

(sofern nur ver- oder entschlüsselt wird genügt der jeweilige Eintrag allein, siehe Listing 11).

Typische Fehler und Probleme sind: Standard bei fehlendem „algorithm"-Attribut ist der DES/CBC/PKCS5Padding-Algorithmus. „DES" gilt aufgrund seiner kurzen Schlüssellänge bereits seit geraumer Zeit als unsicher und sollte nicht mehr verwendet werden. Hier wäre „Triple-DES" (DESede) oder auch „AES" die deutlich bessere Wahl für den Standardalgorithmus gewesen. Im Gegensatz zum XMLSecurity-Data-Format sollte beim Crypto-Data-Format daher unbedingt ein sicherer Algorithmus angegeben werden.

```
<dataFormats>
 <pgp id="asymEncrypt" keyFileName="./pubring.gpg"
  keyUserid="camel@trivadis.com" />
 <pgp id="asymDecrypt" keyFileName="./secring.gpg"
  keyUserid="camel@trivadis.com" password="camelsec" />
</dataFormats>
```

Listing 11

```
<route id="findUserDataSign">
 <from uri="direct:findUserDataSign"/>
 <to uri="crypto:sign://keystore?  keystore=#signatureStore
  &alias=camelsign&password=camelsec" />
 <to uri="crypto:verify://keystore?
  keystore=#signatureStore&alias=camelsign" />
</route>
```

Listing 12

```
<bean id="signatureStore" class="org.springframework.ws.soap.security.support.
KeyStoreFactoryBean">
 <property name="password" value="camelsec" />
 <property name="location" value="classpath:signatureStore.jks" />
</bean>
```

Listing 13

Digitale Signaturen

Von der asymmetrischen Verschlüsselung ist es dann nur noch ein kleiner Schritt hin zu den digitalen Signaturen. Da das XMLSecurity-Data-Format (noch?) keine XML-Signaturen in Camel unterstützt, bietet das Crypto-Data-Format die einzige Möglichkeit, Nachrichten mit Camel-Mitteln digital zu signieren [12]. Wer jetzt erwartet, einfach „marshal" durch „sign" und „unmarshal" durch „verify" zu ersetzen sowie einige Attribute für den Schlüssel anzupassen, wird leider enttäuscht. Digitale Signaturen funktionieren in Camel ein bisschen anders. Um eine Nachricht zu signieren, wird diese an einen Endpunkt ‚crypto:sign´ geschickt. Die notwendigen Konfigurationsangaben werden nicht über Attribute hinzugefügt, sondern als Parameter an die Endpunkt-URI angehängt. Das Verifizieren einer Nachricht funktioniert entsprechend mit einem Endpunkt „crypto:verify". In Listing 12 wird die Nachricht sofort nach der Signierung verifiziert, normalerweise sind diese Schritte natürlich getrennt.

Die Route allein genügt in diesem Fall nicht. Der Parameter „keystore=#signatureStore" in der URI deutet es an: Ein „#"-Zeichen in URI-Attributen verweist auf eine Bean mit entsprechender ID. Im Beispiel ist daher zusätzlich eine Bean mit der ID „signatureStore" notwendig. Diese muss den KeyStore zur Verfügung stellen. Hierfür gibt es verschiedene Möglichkeiten, von fertigen Beans bis hin zur Eigenentwicklung. Das Beispiel verwendet der Einfachheit halber die „KeyStoreFactoryBean" aus dem „Spring-WS-Security"-Paket. Entsprechend muss dazu die POM-Datei um die Abhängigkeit „spring-ws-security" erweitert werden (siehe Listing 13).

Camel stellt hier noch weitere Möglichkeiten zum Laden des Schlüssels und auch zur Angabe weiterer Parameter wie beispielsweise des Algorithmus zur Verfügung. Details dazu stehen auf der Camel-Homepage [12]. Eine Einschränkung aber gibt es: Leider lässt sich das Element „keyStoreParameters" aus dem XMLSecurity-Data-Format nicht als KeyStore für „crypto:sign" verwenden.

Wer jetzt die entsprechende Route aufruft, wird sich vielleicht über die scheinbar unveränderte Nachricht wundern. Nach dem Aufruf des „crypto:sign"-Endpunkts enthält der Header der Camel-Message nur ein zusätzliches Attribut: „CamelDigitalSignature". Dieses besitzt den verkürzten Hashwert der Nachricht, zum Beispiel „CamelDigitalSignature=[B@19ac5d2". Der Body der Nachricht ist damit signiert.

Nach der erfolgreichen Verifizierung durch „crypto:verify" sollte dieses Attribut wieder automatisch von Camel gelöscht werden. Bis einschließlich Camel 2.9.0 verhindert das ein Bug (CAMEL-4996 [13]), sodass der Header auch nach der Verifizierung exakt dem Header nach der Signierung entspricht. Im Falle einer erfolgreichen Verifizierung gibt es übrigens keinerlei Rückmeldung von Camel, die Verarbeitung der Route wird stattdessen einfach fortgesetzt. Lediglich bei einer gescheiterten Verifizierung bricht die Verarbeitung im „crypto:verify"-Endpunkt mit der nicht so ganz eindeutigen Meldung „Cannot verify signature of exchange" ab. Auf diese Exception muss der Entwickler dann entsprechend in der Route reagieren und beispielsweise die Nachricht auf eine andere Weise verarbeiten oder komplett verwerfen. Um diese Exception zu simulieren, genügt es, nach dem Aufruf von „crypto:sign" den Body beispielsweise mit „<setBody><constant>Another body</constant></setBody>" zu verändern und diesen dann zu verifizieren.

Typische Fehler und Probleme sind: Eine häufig auftretende Exception (leider auch bei stark unterschiedlichen Ursachen) lautet „Cannot sign message as no Private Key

has been supplied. Either supply one in the route definition sign(keystore, alias) or sign(privateKey) or via the message header 'CamelSignaturePrivateKey'". Sie tritt zunächst einmal auf, falls der KeyStore nicht geladen werden konnte. Das heißt, dass entweder die ID in der URI nicht mit der Bean übereinstimmen, der KeyStore-Pfad/-Dateiname in der Bean nicht korrekt oder das Passwort des KeyStores falsch ist. Auch ein nicht über den Alias gefundener Schlüssel führt zu dieser Exception. Lediglich ein falsches Schlüssel-Passwort liefert eine andere Exception zurück: „Cannot recover key".

Wie im Beispiel umgesetzt, ist es wichtig, dass für das XMLSecurity- und das Crypto-Data-Format getrennte KeyStores verwendet werden. In der Realität wird dieser Fall wohl ohnehin nicht häufig anzutreffen sein, meist wird man sich für das Crypto- oder das XMLSecurity-Data-Format entscheiden. Verwendet jedoch die „signatureStore"-Bean denselben KeyStore wie beim XMLSecurity-Data-Format (camel.jks), kann der Schlüssel nicht geladen werden und es kommt zu entsprechenden Fehlern.

Fazit und Ausblick

Bereits die Absicherung der Nutzdaten einer Camel-Nachricht kann, wie gezeigt, sehr komplex werden. Viele verschiedene Möglichkeiten stehen zur Verfügung. Gleich zu Beginn der Implementierung steht dabei die Entscheidung an, ob das herkömmliche Crypto- oder das XMLSecurity-Data-Format verwendet werden soll. Welches der beiden Formate besser geeignet ist, wird nicht unbedingt vom zu sichernden Datentyp bestimmt. Ein besseres Entscheidungsmerkmal ist sicherlich, ob die vollständige Nachricht verschlüsselt werden soll oder nur Teile davon, eventuell auch mit unterschiedlichen Schlüsseln.

Anschließend müssen der Algorithmus sowie die dafür notwendigen Parameter festgelegt und die passenden Schlüssel generiert und verwaltet werden. Hier ist es bei beiden Formaten besser, nicht auf die Standard-Algorithmen bei fehlender Angabe zu vertrauen, sondern explizit einen (sicheren) Algorithmus anzugeben. Allein schon, um bei einer Änderung des Standard-Algorithmus nicht mit unerwarteten Exceptions konfrontiert zu werden.

Ärgerlich ist, dass die drei hier gezeigten kryptografischen Anwendungen leicht oder gar völlig unterschiedliche Konfigurationsmöglichkeiten verwenden. Zumindest bei der KeyStore-Konfiguration wäre es wünschenswert, wenn diese nur einmal neutral und unabhängig vom Data-Format konfiguriert werden müsste und dann mit jeglicher Variante – Crypto, XMLSecurity oder digitalen Signaturen – verwendet werden könnte.

Gleichzeitig gilt es bei der Routen-Konfiguration auch, die vorhandenen Fallstricke zu beachten. Gerade die sonst so hervorragende Camel-Dokumentation ist in den Bereichen „Crypto", „XMLSecurity-Data-Format" sowie „digitale Signaturen" noch sehr rudimentär und enthält nur wenige Beispiele. Auch das ansonsten herausragende Buch „Camel in Action" [14] schweigt sich zum Thema Sicherheit leider aus.

Subjektiv betrachtet, lässt sich das XMLSecurity-Data-Format besser verwenden und macht einen moderneren Eindruck. Gleichzeitig sind hier die Sicherungsmöglichkeiten (vollständiges Dokument, Dokument-Fragmente, mit unterschiedlichen Schlüsseln/ Algorithmen) deutlich umfangreicher. Aber auch mit dem Crypto-Data-Format lässt sich ein immenser Sicherheitsgewinn für die Camel-Routen erzielen und man hat mehr Algorithmen zur Verfügung.

In der nächsten Ausgabe dreht sich alles um den zweiten großen Bereich der Camel Security: die Absicherung einer Camel-Route (Route Security) mit Apache Shiro und Spring Security.

Weitere Informationen

[1] http://camel.apache.org
[2] http://camel.apache.org/security.html
[3] http://www.jasypt.org
[4] http://camel.apache.org/camel-configuration-utilities.html
[5] http://camel.apache.org/xmlsecurity-dataformat.html
[6] http://camel.apache.org/crypto.html
[7] http://camel.apache.org/camel-maven-archetypes.html
[8] http://www.w3.org/TR/xmlenc-core
[9] http://santuario.apache.org
[10] http://www.oracle.com/technetwork/java/javase/downloads/index.html
[11] http://www.bouncycastle.org
[12] http://camel.apache.org/crypto-digital-signatures.html
[13] https://issues.apache.org/jira/browse/CAMEL-4996
[14] Ibsen, Claus and Anstey, Jonathan: "Camel in Action", 2011, Manning, ISBN 978–1–935182–36–8
[15] http://aktuell.ruhr-uni-bochum.de/pm2011/pm00330.html.de

Dominik Schadow
dominik.schadow@trivadis.com

Dominik Schadow arbeitet als Senior Consultant im Bereich „Application Development" bei der Trivadis GmbH in Stuttgart. Neben seinem Fokus auf Java-Enterprise-Applikationen und -Architekturen sowie Integrationsthemen rund um Apache Camel beschäftigt er sich als Discipline Manager Application Development Security mit dem Thema „Sichere Software-Entwicklung". In seiner Freizeit leitet er das Open-Source-Projekt „JCrypTool" , mit dem Anwendern die Kryptografie näher gebracht werden soll.

Projektmanagement-Zertifizierung Level D nach GPM

Gunther Petzsch, Saxonia Systems AG

Das Arbeiten mit und in Projekten ist für viele IT-Fachkräfte zum Alltagsgeschäft geworden. Doch mit der Frage, warum Projekte überhaupt so wichtig sind und wie diese erfolgreich durchgeführt werden können, haben sich wahrscheinlich nur wenige beschäftigt. Warum also nicht eine Projektmanagement-Zertifizierung besuchen und sich richtig mit der Materie befassen?

Gerade bei den täglich relevanten Themen wie „Kommunikation", „Konfliktmanagement" und „Planung" kann nicht genug Wissen vorhanden sein. Auch das Verständnis für die Arbeitsweise eines Projektleiters und dafür, wie Projektziele durch das Team beeinflusst werden, kann zum Erfolg eines Projekts entscheidend beitragen.

Projektmanagement-Zertifizierung Level-D

Die GPM ist die Deutsche Gesellschaft für Projektmanagement. Gemessen an der Anzahl von Mitgliedern ist die Organisation das größte Kompetenz-Zentrum im Bereich Projektmanagement in Europa. Über die International Project Management Association ist die GPM weltweit vernetzt. Durch die Mitarbeit an internationalen Normen und Richtlinien trägt sie wesentlich zur Professionalisierung und Weiterentwicklung des Projektmanagements bei. Sie nimmt weitreichenden Einfluss auf Projektmanager und fordert von diesen zusätzlich auch die Einhaltung eines Ethik-Kodex, der sich an den wesentlichen Grundwerten ausrichtet. Die Projektmanagement-Zertifizierung Level-D ist die erste von vier Stufen in den Zertifizierungen zum Projektmanagement, wobei die Bereiche von „A" bis „D" unterteilt werden. Diese Zertifizierungen sind unter dem Namen „IPM 4-L-C"- Zertifikate für Projektmanager zusammengefasst. Level-D steht für „Zertifizierter Projektmanagement-Fachmann". Zahlen der GPM besagen, dass Ende 2011 etwa 21.200 Projektmitarbeiter das Zertifikat der Stufe D besaßen.

Schritte zur Zertifizierung

Den Aussagen der Trainer und Prüfer zufolge gelingt es den wenigsten Teilnehmern, die Prüfung ohne eine Teilnahme am jeweiligen Vorbereitungskurs zu bestehen. Denn dieser baut exakt auf den prüfungsrelevanten Themen auf und vermittelt den Kursbesuchern das gewünschte und benötigte Wissen. Darüber hinaus geben die Dozenten spezielle Anmerkungen bei den wichtigen beziehungsweise interessanten Themen für die Prüfung. Aber auch die umfassende Praxiserfahrung und die vielen Beispiele im Kurs helfen, die Lerninhalte aufzunehmen und das Gelernte umzusetzen.

Der bei der Saxonia Systems AG durchgeführte Vorbereitungskurs umfasste etwa zwölf Tage. In dieser Zeit wurde der Lehrstoff auf knapp 280 Folien vermittelt, was sich natürlich von Kurs zu Kurs unterscheiden kann. Womöglich schwankt die Dauer des Kurses auch je nach Zeit oder Informationsgehalt der Veranstaltung. Ausschlaggebend für die Wahl eines Kurses werden letztendlich oftmals die Kosten oder eine Weiterempfehlung sein.

Ein solcher Kurs bedeutet einen erheblichen Aufwand für die Teilnehmer, wobei eine gewisse Portion Eigenmotivation von Vorteil ist. Die Schulung, über die hier berichtet wird, war eine Inhouse-Schulung in den Firmenräumen des Autors. Dies hatte den Vorteil, dass Teilnehmer, die oft in Projekten innerhalb Deutschlands unterwegs sind, nicht noch eine Reise zusätzlich einplanen mussten und sich somit alle am Firmenstandort treffen und gemeinsam arbeiten konnten. Die Schulung war in fünf Blöcken von jeweils drei beziehungsweise zwei aufeinanderfolgenden Tagen koordiniert. Einen Großteil der Zeit nahm die Vermittlung von Wissen anhand der entsprechenden Kapitel und Folien aus den Schulungsunterlagen ein. Dieses Arbeiten wurde in der Schulung aber gut anhand von praxisnahen Beispielen, Lern- und Teamspielen, freiem Arbeiten an entsprechenden Lernaufgaben und deren anschließenden Ergebnispräsentation aufgelockert. Auch wenn dies das Lernen erleichterte, so war das Arbeiten doch für alle Teilnehmer anstrengend.

Aufwand und Voraussetzungen für die Prüfung

Im Prinzip gibt es keine fachlichen Voraussetzungen für die Teilnahme zur Prüfung. Aber natürlich ist es von Vorteil, wenn man selbst schon in dem einen oder anderen Projekt tätig gewesen ist und somit auch Erfahrungen in den entsprechenden Kompetenzelementen gesammelt hat.

Um zur Prüfung zugelassen zu werden, muss eine Projektarbeit ausgearbeitet und abgegeben werden. Im Rahmen der Level-D-Zertifizierungen spricht man von einem Transfernachweis. Das Thema für diesen Nachweis ist ein selbst gewähltes Projekt, das entsprechend den Kompetenzen des Projektmanagements aufgearbeitet werden muss. Dieses Projekt kann fiktiv, aber auch real stattgefunden haben. Folgende

Themen müssen Inhalt des Transfernachweises sein:

- Projekt und Projektziele
- Stakeholder-Analyse und Projektumfeld-Betrachtung
- Risiko-Analyse
- Projektorganisation
- Projektphasen-Planung
- Projektstruktur-Planung
- Ablauf- und Terminplanung
- Einsatzmittel- und Kostenplanung
- Ein Wahlthema zu den Verhaltenskompetenzen
- Ein weiteres Wahlthema aus den Projektmanagement-Kompetenzen

Der Transfernachweis wird als PDF-Datei im Portal der GPM nach Abschluss der Arbeit hochgeladen. Diese muss spätestens vierzehn Tage vor dem eigentlichen Prüfungstermin erfolgen und darf die maximale Seitenanzahl von 60 sowie eine Dateigröße von 7 MB nicht überschreiten. Dennoch ist der Transfernachweis schon mit gewissem Aufwand verbunden. Der Dozent sprach von etwa 80 Stunden Arbeitszeit. Erwähnenswert ist noch, dass bis zu drei Prüfungsteilnehmer diese Projektarbeit gemeinsam erstellen können. Der Aufwand wird also zunächst für die einzelne Person geringer. Nachteilig dabei ist, dass diese Arbeit bei einer höheren Zertifizierung (Level-C) nicht komplett anerkannt wird und an dieser Stelle Mehraufwand entsteht. Weiterhin ist natürlich die Abarbeitung aller Themengebiete des Transfernachweises gleichzeitig Vorbereitung auf die Prüfung zur Zertifizierung.

Prüfung

Die Prüfung zum Abschluss der Zertifizierung besteht aus zwei Teilen – einer schriftlichen und einer mündlichen Prüfung. Die schriftliche Prüfung hat eine Dauer von zwei Stunden. Dabei dürfen keine Unterlagen und keine sonstigen technischen Geräte, außer einem nicht programmierbaren Taschenrechner, benutzt werden. Die mündliche Prüfung hat eine Dauer von etwa 30 Minuten und wird von zwei Assessoren abgenommen. Der Prüfling muss dabei drei Karten aus einem Pool von etwa 150 Prüfungsfragen ziehen und nach dem Prüfen und Vorlesen der Fragen eine Karte davon zurück in den Pool legen. Im Anschluss sollte er die verbleibenden zwei Fragen mündlich beantworten. Hierbei dürfen Hilfsmittel wie Whiteboard, Zettel und Stift benutzt werden. Für die Beantwortung der Fragen stehen dem Prüfling 20 Minuten der Prüfungszeit zur Verfügung. Die restliche Zeit ist für weitere Fragen auch zu anderen Themen oder zur Besprechung des Transfernachweises vorgesehen.

Beide Prüfungen finden am selben Tag statt. Nach Bestehen der Prüfung erhält man das entsprechende Zertifikat und einen Antrag auf eine kostenlose Mitgliedschaft in der GPM über ein Jahr. Das Zertifikat ist fünf Jahre gültig und kann danach gegen Zahlung eines bestimmten Betrags um weitere fünf Jahre verlängert werden.

Das intensive Lernen für die Prüfung ist aus Sicht des Autors unabdingbar. Die Gründe hierfür sind der Umfang des Lernstoffs, die unzähligen Möglichkeiten der Fragestellung und Inhalte sowie das Zusammenlegen der Prüfungen (mündliche und schriftliche) an einen Tag.

Die Teilnehmer des Kurses hatten zwei Tage vor der Prüfung einige Beispielfragen bekommen, um diese probeweise zu beantworten. Das Ergebnis war niederschmetternd. Nur etwa die Hälfte der Gruppe hatte gerade so bestanden. Dabei hatten sich alle schon auf die Prüfung vorbereitet. Das bedeutete für einige noch zwei zusätzliche Nachtschichten zum Lernen. Unterschätzen sollte man die Prüfungsvorbereitung auf keinen Fall.

Der Prüfungstag begann mit der schriftlichen Prüfung. Die maximal erreichbare Punktezahl lag bei 120 Punkten, zum Bestehen wurden also 60 Punkte benötigt. Die Reihenfolge für die mündliche Prüfung war am Vortag in der Gruppe bestimmt worden. Somit wusste jeder Einzelne, wann er in etwa an der Reihe war. Alle Teilnehmer des Kurses bestanden die mündliche Prüfung.

Fazit

Zum Erlangen dieses Zertifikats muss richtig Aufwand betrieben werden. Zitat eines Kursleiters am ersten Kurstag: „Melden Sie sich schon mal für die nächsten drei Monate bei Ihrem Lebenspartner ab." Im Gegenzug werden Wissen und Werte vermittelt, die man tagtäglich im Beruf oder auch in der Freizeit anwenden kann. Ferner verbessert sich die Mitarbeit und das Verstehen von Projekten. Strukturen und Probleme können besser erfasst und somit frühzeitig positiv beeinflusst werden. Auch für Menschen, die nicht unbedingt den Wunsch hegen, eine Projektleitung zu übernehmen, ist dieser Kurs beziehungsweise das Zertifikat aufgrund der oben genannten Punkte von großem Nutzen. Aus persönlicher Erfahrung ist die Teilnahme an einem Vorbereitungskurs sehr zu empfehlen. Im Vergleich mit einer SCJP-Zertifizierung, bei der man mittels eines entsprechenden Buches sehr konkret vorbereitet wird, ist die Literatur für das Projektmanagement doch sehr umfassend und durchaus in manchen Punkten konträr. Ohne die hilfreichen Tipps vom Kursleiter kann man hier schnell stolpern.

Man könnte sich die Frage stellen, ob für die Zertifizierung nicht zu viel Aufwand verlangt wird. Aber im Nachhinein betrachtet war aus Sicht des Autors alles stimmig. Ohne den Transfernachweis, bei dem das erworbene Wissen gleich verstanden und umgesetzt werden musste, wäre die Prüfung durchaus schwieriger geworden. Und natürlich hat dieses Zertifikat auch eine Auswirkung auf das Berufsleben, empfiehlt man sich doch für höhere Aufgaben und vielleicht auch für eine höhere Lohngruppe.

Quellen

www.gpm-ipma.de

Gunther Petzsch
gunther.petzsch@saxsys.de

Gunther Petzsch ist Sun-zertifizierter Java-Entwickler. Er lebt und arbeitet in Dresden. Dort studierte er auch erfolgreich Wirtschaftsinformatik an der Hochschule für Technik und Wirtschaft Dresden. Danach arbeitete er unter anderem für das Bundeskriminalamt in Wiesbaden. Aktuell ist Gunther Petzsch als Senior Consultant für die Saxonia Systems AG tätig.

Android in Lehre und Forschung: Entwicklung wissenschaftlicher Applikationen der nächsten Generation

Jonas Feldt und Johannes M. Dieterich, Institut für Physikalische Chemie, Georg-August-Universität Göttingen

Das rapide Wachstum des Marktes für intelligente Mobilgeräte und die schnelle Entwicklung von Hard- und Software schaffen neue Möglichkeiten für Lehre und Forschung. Höchste Mobilität und intuitive Interaktionsmöglichkeiten zeichnen diese neue Geräteklasse aus und stehen der Nutzung durch wissenschaftliche Apps offen. Der Artikel beleuchtet am Beispiel der „Atomdroid"-App, einer Android-Applikation für Computer-Chemie und –Biochemie, technische Aspekte von Entwicklung und Nutzung wissenschaftlicher Applikationen im Android-Kontext.

Klassische wissenschaftliche Applikationen sind immobil. Serverseitige Applikationen laufen auf Rechenzentren, clientseitig sind sie an Workstations oder – im mobilsten Fall – an Notebooks gebunden (siehe Abbildung 1). Begründet wird dies mit den hohen Rechenanforderungen, die für Visualisierung und Modellierung der wissenschaftlichen Systeme benötigt werden. Mit dem Aufkommen der neuen Geräteklasse intelligenter Mobilgeräte stellt sich die Frage, ob sich die klassentypischen Eigenschaften (Mobilität, veränderte Haptik, intuitivere Kontrolle) im wissenschaftlichen Kontext nutzen lassen. Ein vollständiger Ersatz althergebrachter Programmcodes ist weder gewünscht noch realistisch. Ziel ist lediglich, die neugeschaffenen Möglichkeiten mit neuen, modernen Apps zu nutzen. Naheliegende Ziele sind hierbei, Forschern ultramobile Werkzeuge an die Hand zu geben, die auch in der Wissensvermittlung intuitiv eingesetzt werden können. Der Artikel zeigt Schwierigkeiten und Lösungsmöglichkeiten auf diesem Weg am Beispiel der Erfahrungen mit der von den Autoren entwickelten „Atomdroid"-App für Android-Systeme [1], wobei diese weder einen Anspruch auf Vollständigkeit noch auf optimale Lösungen erheben.

Android: Allgemeine Probleme und Lösungsansätze

Neben der offensichtlichen Frage, inwieweit die Leistung der heutigen, mobilen Geräte-Generation und der virtuellen Maschine von Android (DalvikVM) für die oben genannten Anforderungen ausreicht, ist ein häufig genannter Kritikpunkt am Android-Ökosystem die hohe Markt-Fragmentierung hinsichtlich Hardware-Spezifikationen und Android-Versionen. Die Erfahrung zeigt, dass die Nutzerbasis tatsächlich stark fragmentiert ist (siehe Abbildungen 2 und 3). Damit assoziierte Probleme lassen sich allerdings durch zwei relativ einfache Regeln minimieren: Erstens sollte eine strikte Einhaltung der Android-Programmierstandards erfolgen [2] und zweitens sind die bekannten manuellen und/oder automatischen Stellschrauben geboten, die unter anderem Grafik-Qualität und -Aufwand je nach Gerät anpassen. Eine sinnvolle Standard-Konfiguration und harte Limits können hier helfen, Fehlabstimmungen und daraus resultierende Beschwerden seitens der Nutzer zu minimieren. Zusätzlich dazu ermöglicht Google durch die Fragments-Kompatibilitätsbibliothek eine vereinfachte Anpassung der Benutzeroberfläche auf verschiedene Bildschirmgrößen auch für Android-Versionen ab 1.6 [3].

Unabhängig von diesen Regeln sollte zu Beginn eines Android-Projekts die minimal vorausgesetzte API-Version gut bedacht werden. Ältere API-Level korrespondieren mit einer größeren Zahl potenzieller Nutzer, aber auch mit einem erhöhten Ent-

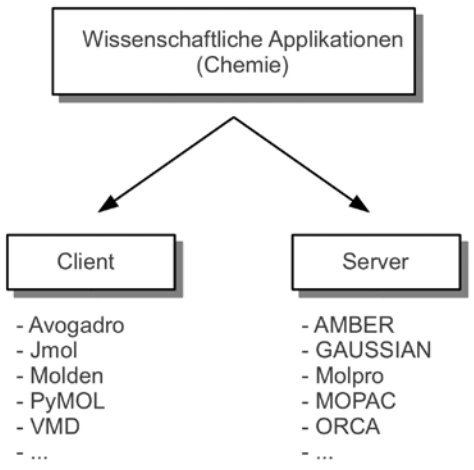


Abbildung 1: Klassische Aufteilung von Computerchemie-Programmen

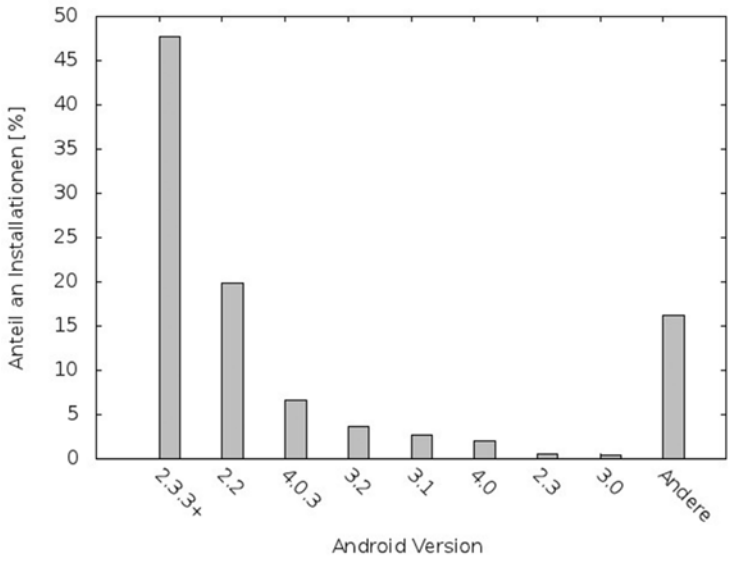


Abbildung 2: Fragmentierung bezüglich der Android-Version

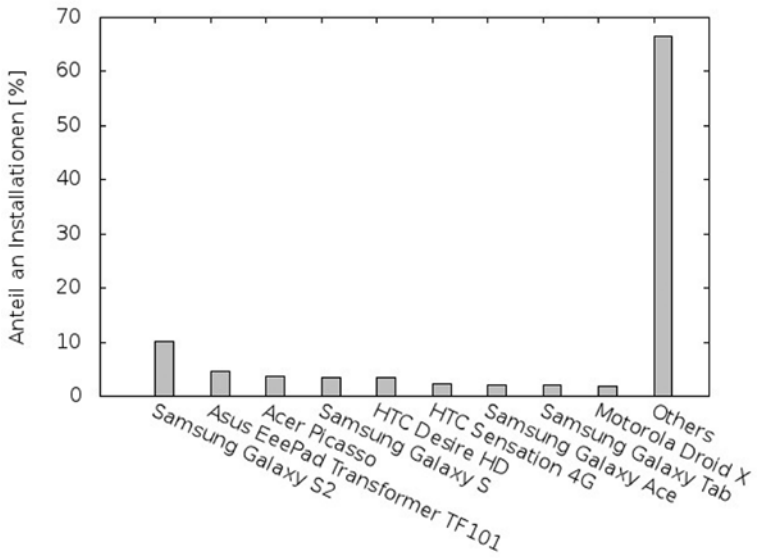


Abbildung 3: Hardware-Fragmentierung

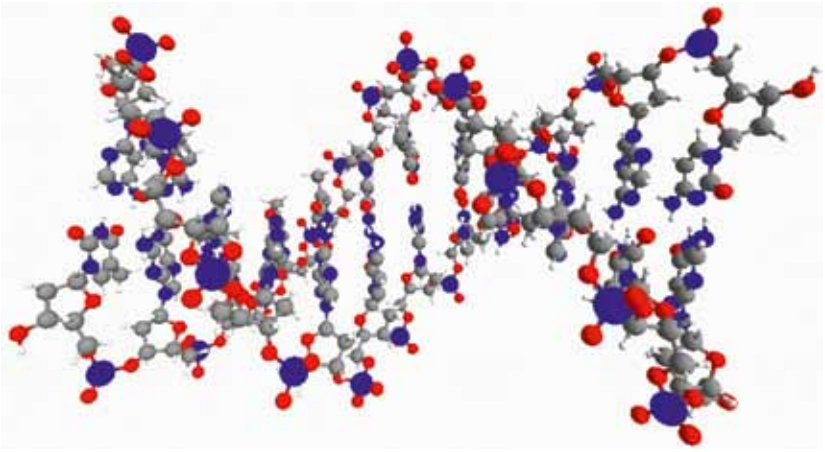


Abbildung 4: Grafische Darstellung eines Moleküls (PDB-ID 2LFA)

wicklungsaufwand, da sich alte Versionen häufig auf nicht performanten Geräten finden. Zum jetzigen Zeitpunkt erscheint den Autoren API-Level 8 (Android 2.2) ein guter Kompromiss zu sein. Schlüssel-Technologie ist hier hinsichtlich Performance der in der DalvikVM enthaltene JIT [4].

Das Erreichen potenzieller Nutzer ist durch die Market-Umgebungen prinzipiell sehr einfach. Die Statistikmöglichkeiten des Google-Markts sind exzellent und bieten einen genauen, anonymisierten Überblick über die Nutzerbasis. Es stellt sich allerdings heraus, dass Nutzer zwar sehr einfach Apps installieren und kommentieren können, eine beidseitige Kommunikation von Entwicklern mit Nutzern aber unnötig erschwert wird. Durch das Fehlen einer Antwortmöglichkeit ist es nicht direkt möglich, auf Kritik oder Vorschläge zu reagieren. Die Option, dem Entwickler eine Mail zu schicken, nutzen trotz expliziter Aufforderung in der App-Beschreibung nur vergleichsweise wenige Nutzer (unter 0,1 Prozent bei Atomdroid). Hier stehen mangelhafte Rückkopplungsmechanismen einer agilen, nutzerorientierten Entwicklung im Weg.

Performance für den täglichen Einsatz

Zunächst muss man Visualisierungs- und Simulations-Performance unterscheiden. Während wissenschaftliche Simulationen Prozessor- und Speicher-intensiv sind (unter Vernachlässigung einer Beschleunigung durch GPU-Einsatz), spielt die Grafik-Performance offensichtlich eine zentrale Rolle bei Visualisierungsaufgaben. Einschränkend gehen die Autoren im Folgenden davon aus, dass immer die DalvikVM verwendet wird und nicht durch Einsatz des Android Native Development Kits (NDK) Sprachen wie C/C++ zum Einsatz kommen.

Die Visualisierung wird auf Android durch eine Mischung von Android-internen APIs für die Darstellung von Menüs oder Listen sowie OpenGL ES für die Darstellung von 3D-Objekten realisiert. Diese Kombination ist leicht verwendbar und im Fall von OpenGL ES standardisiert und portabel. Die intuitive Haptik mit Multitouch-Events erlaubt eine sehr natürliche Interaktion (siehe Abbildung 4). Die heutige Generation von Geräten ermöglicht die Darstellung großer Datensätze.

Interessant ist, ob rechenintensivere Simulationen auf mobilen Geräten möglich sind. Eine mobile wissenschaftliche App sollte auch eine gewisse Autonomie hinsichtlich Standardaufgaben ermöglichen. In der Computerchemie ist dies gleichzusetzen mit einfachen, kraftfeldbasierten Simulationen, um beispielsweise die räumliche Struktur eines Moleküls zu berechnen. Kraftfelder sind durch ihren geringen Speicherplatzbedarf gut für mobile Geräte geeignet.

Um eine gute Performance auf Android-basierten Geräten sicherzustellen, sind nach Erfahrung der Autoren mehrere Faktoren wichtig. Neben der Wahl einer passenden Methode sind die üblichen Optimierungen zur Minimierung von Fließkommaoperationen von Bedeutung. Speziell ist bei Android allerdings die Notwendigkeit der Minimierung von Speicheroperationen (Allokation und Freigabe).

Zum einen muss vermieden werden, für typische Vektoroperationen kleine „double []" in Schleifenkontexten zu allozieren. Die Optimierung ist aus Standard-Java bekannt und im Listing 1 exemplarisch für die Berechnung eines Diederwinkels dargestellt ist.

```
static double dihedralAngle(double[] first, double[] second, double[] third,
        double[] fourth, double[] t1, double[] t2, double[] t3, double[] t4) {
    for(int i = 0; i < 3; i++){
      t1[i] = first[i] - second[i];
      t2[i] = third[i] - second[i];
    }
    crossproduct(t1, t2, t3);
    for(int i = 0; i < 3; i++){
      t1[i] = fourth[i] - third[i];
      t2[i] = second[i] - third[i];
    }
    crossproduct(t1, t2, t4);
    crossproduct(t3, t4, t1);

    final double length = value(t1);
    if(length < 0.001) return 0.0;

    return Math.PI - angle(t3, t4);
}
```

Listing 1

```
final Gradient grad = uff.gradient(Geometry g, float[][] bonds, double[][] distances,
                                   double[][] freeScratch,UFFData[] data);
final Gradient grad = uff.gradient(Geometry g);
```

Listing 2

Hier sind die Arrays „t1" bis „t4" temporär benötigte Vektoren. Der so optimierte Code ist deutlich (etwa Faktor fünf) schneller als eine naive Implementation. Darüber hinaus ist es, anders als bei Standard-Java, sehr wichtig, auch weniger häufig auftretende Speicher-Operationen zu vermeiden. Der Unterschied wird verständlich, wenn man verschiedene Interfaces zu unserer eingebauten Kraftfeld-Routine betrachtet. Aufgabe ist, einen kartesischen Gradienten des Moleküls zu berechnen, eine Operation, die für die implementierte Methode „$O(N^2)$" mit der Anzahl von Atomen skaliert und mehrfach für eine lokale Optimierung aufgerufen wird (siehe Listing 2).

Im ersten Fall werden benötigte Arrays und Felder wiederverwendet, im zweiten Fall in der Gradient-Routine jeweils einmal lokal alloziert. Wie aus Abbildung 5 zu ersehen, beschleunigen solche Optimierungen auf Android-Plattformen den Code beträchtlich und sind auf älteren Versionen wie Froyo zwingend notwendig. Anhand der Honeycomb-Timings (Android 3.0) ist ersichtlich, dass der Gingerbread Garbage Collector ein guter Schritt in die richtige Richtung war. Dennoch reicht die Performance der DalvikVM hier nicht an eine aktuelle JVM heran, die solche Optimierungen schlicht nicht benötigt.

Optimierte Apps sind durchaus für die erwähnten Standardaufgaben geeignet, ein kartesischer Gradient (inkl. Energie) lässt sich selbst für ein Molekül mit 155 Atomen auf Tegra2-Hardware in 106,8 ± 0,2 Millisekunden berechnen. Es ist somit möglich, die Minimumsstruktur durch eine lokale Optimierung in circa einer Minute zu finden. So vorbereitete Strukturen sind Startpunkt für weitere Analysen und Simulationen.

Die Unterteilung in zwei verschiedene Anwendungsklassen ermöglicht darüber hinaus eine einfache Parallelisierung mit Android-Bordmitteln. Hierzu werden rechenintensive Simulationsaufgaben aus dem User Interface Thread mittels AsyncTasks ausgelagert. Diese Abstraktion erhöht zum einen die grafische Interaktivität der App, zum anderen vereinfacht sie nebenläufiges Programmieren durch definierte Interfaces, ohne den Aufwand eines direkten Thread-Managements.

Ein Problem stellt sicherlich noch die Verwendung externer Bibliotheken dar. Obwohl einer prinzipiellen Verwendung der allermeisten Java-5-kompatiblen Bibliotheken nichts im Wege steht, ist die Performance von JVM-orientiertem Code in der DalvikVM häufig suboptimal. Abhängig davon, ob die Bibliothek in einem Performance-kritischen Teil des Programms eingesetzt wird und als Quellcode vorliegt, lohnen sich Optimierungsanstrengungen.

Mobile Moleküle

Ein zentraler Aspekt wissenschaftlicher Apps ist die Verfügbarkeit relevanter Daten. Für chemische Zwecke sind dies meistens Molekül-Strukturen. Daher ist für die Entwicklung wichtig, dass es Nutzern einfach ermöglicht wird, neue Moleküle zu erhalten und selbst erstellte zu verteilen. Man unterscheidet zwischen direkter Nutzer-Interaktion ohne zusätzliche Hard- oder Software und indirekter mittels Server-Funktionalitäten und verfügbaren Netzwerken.

Für den direkten Datenaustausch zwischen Mobilgeräten kommen Bluetooth und NFC in Betracht. Bluetooth ist weiter verbreitet

und besitzt eine höhere Übertragungsrate bei einer größeren Reichweite. Atomdroid erlaubt es, Molekül-Strukturen zwischen zwei Geräten per Bluetooth verschlüsselt zu versenden. Obwohl für den direkten, autonomen Austausch sehr praktisch, skaliert diese Lösung nicht. Hier ist serverbasierter Datenaustausch notwendig. Die Autoren haben sich entschieden, keine echte Server-Software zu entwickeln, sondern lediglich Dateibäume auf einem vorhandenen Webserver zu nutzen. Dieser Ansatz hat neben einem verringerten Entwicklungsaufwand die Vorteile, dass Webserver in den meisten Fällen leicht verfügbar sind und zusätzlich auch andere potenzielle Applikationen einfachen Zugriff erhalten. Ein offener Standard-Server ist verfügbar, zusätzlich gibt es die Option, einen eigenen Server einzurichten. Forschungs- oder Lerngruppen wird so ermöglicht, eigene Spezialserver zu unterhalten.

Die HTTP-basierte Kommunikation von Atomdroid mit dem Server ist schematisch in Abbildung 6 dargestellt. Dabei sind mehrfach verwendete Daten immer zwischengespeichert, um die hohe Latenz und die geringe Bandbreite mobiler Netzwerke zu umgehen. Nutzerseitig findet eine Navigation mithilfe von Molekül-Kategorien statt. Nach der Auswahl eines Moleküls wird der Inhalt der Info-Datei angezeigt und der Nutzer kann falls gewünscht die Strukturdaten herunterladen.

Zur einfachen Suche (Namen- und Summenformel-basiert) werden auf dem Server vorbereitete Index-Dateien vorgehalten, die clientseitig abgerufen, gecacht und durchsucht werden können. Bei der namensbasierten Suche erfolgt durch einen Damerau-Levenshtein-Algorithmus eine einfache Korrektur von Tippfehlern. Die Summenformel-basierte Suche ermöglicht es zum Beispiel, nach allen Molekülen auf dem Server zu suchen, die sechs Kohlenstoff- und zwei Stickstoff-Atome enthalten.

Nutzer können aus Atomdroid heraus Geometrien mit dem Standard- oder Custom-Server teilen. Die Geometrie und weitere, per Eingabemaske abgefragte Daten werden hierzu per Mail an eine Kontaktadresse geschickt. Obwohl diese Vorgehensweise umständlicher als ein direkter Upload ist, ermöglicht sie eine direkte Kontrolle der Korrektheit des Moleküls und zusätzlicher Daten.

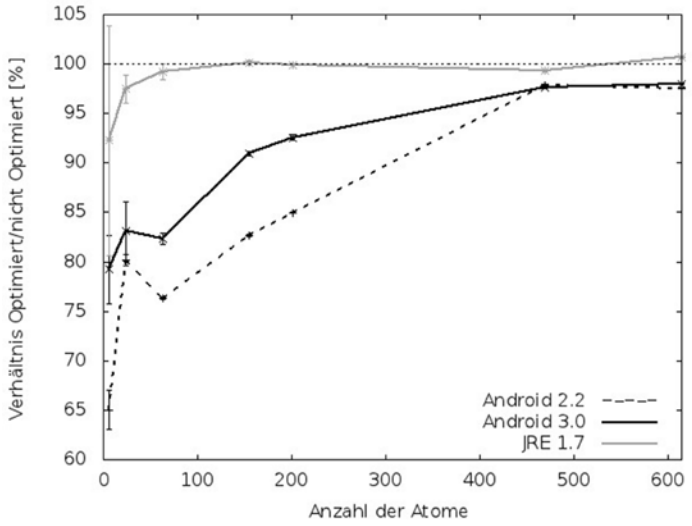


Abbildung 5: Einfluss von speicheroptimiertem Code in Android und JavaSE

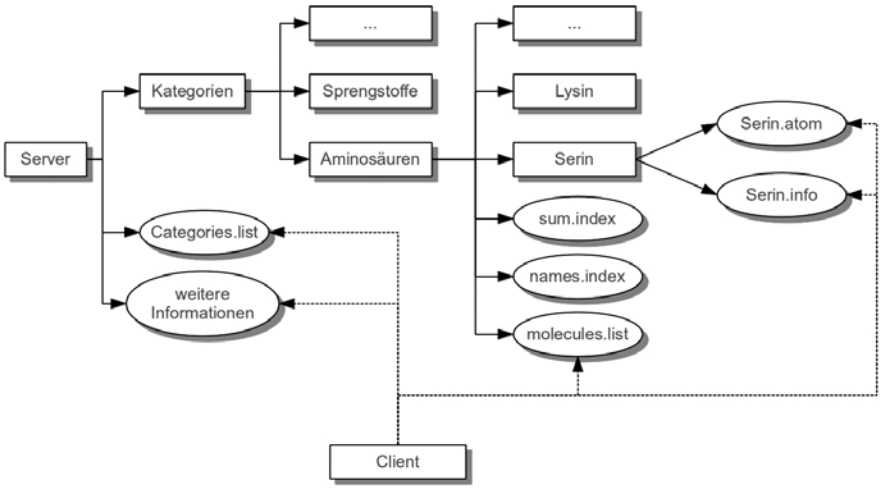


Abbildung 6: Schematische Darstellung der Client-Server-Interaktion

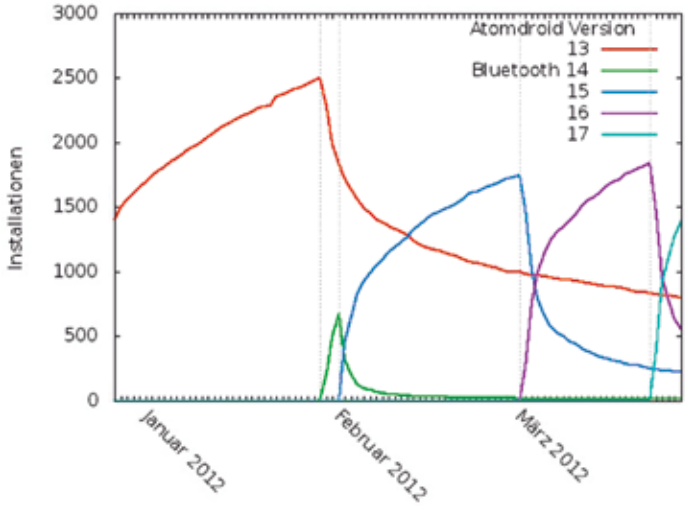


Abbildung 7: Nutzungszahlen verschiedener Atomdroid-Versionen

Fazit

Es scheint schon jetzt – trotz Nischendaseins – eine signifikante Nachfrage nach wissenschaftlichen Anwendungen auf Android zu geben. Die Anzahl an aktiven Installationen der App (Stand März 2012: 3700) übersteigt die anfänglichen Vorstellungen um ein Vielfaches (siehe Abbildung 7). Auch andere wissenschaftliche Apps finden eine recht breite Verwendung, wobei die Anzahl der Nutzer proportional zum Lehranspruch der Apps zu sein scheint. Ein interessanter Nebenaspekt ergibt sich aus der fehlenden Akzeptanz von Updates, die weitere Permissions erfordern.

Viele Nutzer scheinen eine kritische Einstellung zu neuen Berechtigungen bei Updates zu haben. Dies ist eine positive Entwicklung, die allerdings bei der Entwicklung bedacht werden sollte. So ist zum Beispiel bei Atomdroid eindeutig zu sehen, dass das Bluetooth-Update mit den neuen assoziierten Berechtigungen viele Nutzer von einem Update abgehalten hat. Dies erhöht nochmals die Fragmentierung, diesmal auf App-Seite. Google scheint hier durch das neue Feature der Developer Privacy Policies das Ziel zu haben, eine Vertrauensbasis zwischen Nutzer und Entwickler zu schaffen.

Für die Zukunft kann sicherlich ein weiteres Wachstum der Leistungskapazität mobiler Endgeräte erwartet werden. Ihre Nutzung als lokale Rechen-Ressource wird daher von wachsender Bedeutung sein. Dennoch werden auf absehbare Zeit alle wissenschaftlichen Rechen-Anforderungen, die heutzutage typischerweise durch Rechenzentren befriedigt werden, nicht erfüllbar sein. Wichtig wird hier eine nahtlose Integration mit zentralisierten Rechenkapazitäten in Form von Cloud Computing sein. Entscheidend könnte hier Flexibilität hinsichtlich der serverseitigen Umgebung sein, die typischerweise verschiedene Programmcodes und serverabhängig Queuing-Systeme umfasst. Eine offene und einfache Plug-in-Architektur sollte hier ausreichende Anpassungsmöglichkeiten zur Verfügung stellen.

Im Lehrkontext muss zuerst eine breite Adaption der existierenden Technologien und Medien erfolgen. Ausdrücklich sollte diese nicht nach dem Prinzip „neuer ist besser“ erfolgen, sondern evolutionär neue Technologien zur effektiven Wissensvermittlung einbinden. Zentrale Herausforderung für Entwickler wird hier sein, Nutzer-Feedback aktiv zu suchen und zu verarbeiten. Wünschenswert wären hier allerdings einfachere Kommunikationsmöglichkeiten zwischen Entwicklern und Nutzern auf den Vertriebsplattformen.

Die Autoren gehen davon aus, dass mobile Systeme und mit ihnen Android als Plattform für wissenschaftliche Applikationen eine wichtige Rolle spielen werden. Google wird hierbei sicherlich prägend dafür sein, in welche Richtung sich das Android-Ökosystem entwickelt. Schon jetzt bietet Android allerdings mehr Möglichkeiten als Telefonie und Zeitvertreib und es obliegt Nutzern und Entwicklern, diese zu nutzen und dadurch einen neuen Markt zu erschließen.

Literatur

[1] Atomdroid: https://play.google.com/store/apps/details?id=org.atomdroid

[2] Android Coding Standards: http://developer.android.com/guide/practices/compatibility.html

[3] Fragments: http://developer.android.com/guide/topics/fundamentals/fragments.html

[4] Android API-Versionen: http://developer.android.com/guide/appendix/api-levels.html

Jonas Feldt und Johannes Dieterich
jdieter@gwdg.de

Jonas Feldt ist Student im Master-Studiengang für Chemie an der Georg-August-Universität Göttingen. Neben seinem Studium ist er als wissenschaftliche Hilfskraft in der Arbeitsgruppe „Computational Chemistry and Biochemistry“ tätig.

Johannes Dieterich, Jahrgang 1985, ist seit September letzten Jahres als Postdoktorand am Institut für Physikalische Chemie der Georg-August-Universität Göttingen beschäftigt. Er befasst sich dort unter anderem mit der Weiterentwicklung genetischer Algorithmen zur globalen Optimierung chemischer Proleme.

Der Redaktionsbeirat der Java aktuell stellt sich vor

Dr. Jens Trapp (links) arbeitet seit 2007 als Software-Entwickler bei Google in Hamburg. Davor war er 8 Jahre für Sun Microsystems unter anderem als Software-Architect und Java Ambassador tätig. Der Schwerpunkt seiner Arbeit liegt auf Client-Technologien (neben Java auch Javascript, Html5/SVG, etc.) und Datenmodellierung und -verarbeitung (Google BigQuery, etc.).

Ronny Kröhne (Mitte) ist ein IBM Architekt mit 12 Jahren Projekterfahrung. Schwerpunkt seiner Arbeit ist die Erstellung von Java-EE-Anwendungen mit lokalen und global verteilten Teams. Branchenerfahrungen sammelte er im Automobil- und Versicherungsbereich sowie im öffentlichen Sektor. Neben technischen Themen gilt sein Interesse den Vorgehensmodellen in der Softwareentwicklung und der Organisation von verteilten Projektteams. Er ist Mitglied im Redaktionsbeirat, weil für ihn der Erfahrungsaustausch innerhalb der Java-Comunity eine große Bedeutung hat.

Daniel van Ross (rechts) ist Software-Entwickler bei der Neptune-Labs GmbH in Lemgo. Sein Arbeitsschwerpunkt ist die Entwicklung serverseitiger Java-Anwendungen. Daneben versorgt er die Webseite der Java User Group Deutschland e.V. mit aktuellen Neuigkeiten und ist an der Planung der jährlich in Göttingen stattfindenden Source Talk Tage beteiligt. Er ist Java-User seit Erscheinen der Version 1.1 und möchte mit der Arbeit an der Java aktuell einen fachmännischen Beitrag für die Community leisten.

10 Years PatternTesting – ein Rückblick

Oliver Böhm, T-Systems GmbH

Wir schreiben das Jahr 2002. Das Y2K-Problem ist bereits Geschichte, der Euro eingeführt und der Zusammenbruch des Neuen Marktes noch deutlich zu spüren. Man ist vorsichtiger geworden mit neuen Technologien, die die Welt verändern werden. In dieser Zeit machte sich eine kleine Gruppe um Vincent Massol (heute XWiki) und Matt Smith auf, eine der ersten AOP-Bibliotheken in die Welt zu setzen. Was aber verbirgt sich hinter Aspekt-orientierter Programmierung (AOP)?

Dieser Artikel gibt einen Einblick in die wunderbare Welt der Aspekt-Orientierung und die Arbeit an und mit PatternTesting. Was hat sich in den letzten zehn Jahren verändert, außer dass der AspectJ-Compiler inzwischen Eingang in das Eclipse- Öko-System gefunden hat? Sind Aspekte inzwischen in der Java-Welt angekommen? Oder ist AOP nur die Antwort auf die Frage, die keinen interessiert?

Es war einmal …

… eine Programmiersprache, die erfreute sich unter Kaffeetrinkern (und nicht nur dort) großer Beliebtheit. Aber es fehlte etwas. Man konnte zwar mit Objekten alle Dinge dieser Welt nachbilden, aber die Realisierung von Transaktionen, Autorisierung oder Logging ging durch alle Klassen und entzog sich meist erfolgreich der Kapselung. Wie schafft man es, diese „Querschnittsbelange" (auch als „crosscutting concerns" bezeichnet) herauszuziehen und in einer eigenen Klasse zu kapseln? Dies waren die Überlegungen, die bereits im letzten Jahrhundert zu einem neuen Paradigma führten – der Aspekt-orientierten Programmierung (AOP). AOP ist nichts komplett Neues, sondern baut auf der Objekt-Orientierung (oder auch der prozeduralen Programmierung) auf, bietet jedoch zusätzliche Sprachmittel an, um die bereits angesprochenen Querschnittsbelange zwar nicht in Klassen, aber in Aspekten zu kapseln.

Im Land der Aspekte

Um uns einen ersten Eindruck von der Aspekt-orientierten Vorgehensweise zu verschaffen, betrachten wir Listing 1 aus dem Bank-Bereich. Für eine reale Konto-Verwaltung sollte man jetzt zwar nicht unbedingt ein „double" verwenden (siehe http://haupz.blogspot.de/2009/01/ich-bin-reich.html), aber ansonsten kann man hier die eigentliche Business-Logik einer Konto-Verwaltung noch sehr gut nachvollziehen. Im Gegensatz zu diesen fachlichen Anforderungen (im AOP-Jargon auch als „Concern" bezeichnet) gibt es jede Menge nicht-fachlicher „Concerns" wie folgende Punkte, die sich mit OO-Mitteln schwer kapseln lassen:

- Protokollierung (Logging)
- Autorisierung
- Sicherheit
- Transaktionen

Betrachten wir dazu die Anforderung „Alle Kontobewegungen müssen protokolliert werden" (siehe Listing1).

```
public class Konto {

private double kontostand = 0.0;

public double abfragen() {
return kontostand;

}

public void einzahlen(double betrag) {
kontostand = kontostand + betrag;

}

public void abheben(double betrag) {
kontostand = kontostand - betrag;

}

public void ueberweisen(double betrag, Konto anderesKonto) {
this.abheben(betrag);
anderesKonto.einzahlen(betrag);
}

}
```

Listing 1

```
public class Konto {

private static Logger log = Logger.getLogger(Konto.class)
private double kontostand = 0.0;
...

public void einzahlen(double betrag) { kontostand =
kontostand + betrag;
log.info(„neuer Kontostand: „ + kontostand);
}

public void abheben(double betrag) { kontostand =
kontostand - betrag; log.info(„neuer Kontostand: „ +
kontostand);
}
...

}
```

Listing 2

```
public aspect LogAspect {

private static Logger log = Logger.getLogger(LogAspect.
class);

after(double neu) : set(double Konto.kontostand) &&
args(neu) {
log.info("neuer Kontostand:" + neu);
}

}
```

Listing 3

```
public void ueberweisen(double betrag, Konto anderes-
Konto) {
assert anderesKonto != null; this.abheben(betrag); ande-
resKonto.einzahlen(betrag);
}
```

Listing 4

```
public static User login(String name) throws LoginEx-
ception {
assert name != null;
…
assert user != null;
return new user;
}
```

Listing 5

```
public aspect NullPointerTrap {
after() returning(Object ret) : execution(public Ob-
ject+ *..*(..)) {
assert ret != null;
}
}
```

Listing 6

```
@MayReturnNull
public static User login(String name) {
…

}
```

Listing 7

```
after() returning(Object returned) :
execution(public Object+ *..*(..))
&& !execution(@MayReturnNull public Object+
*..*(..)) {
...
}
```

Listing 8

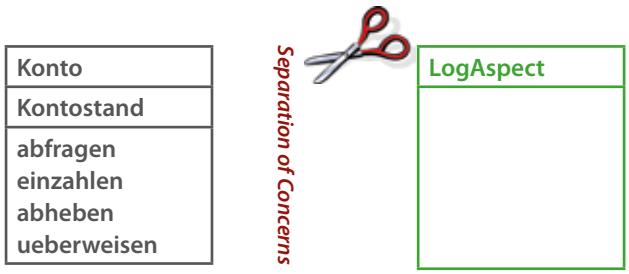


Abbildung 1: Separation of Concerns

Achtung: Code-Verschmutzung

In Java würde man dieses Problem vielleicht wie in Listing 2 lösen (die neu hinzugekommenen Anweisungen sind kursiv gekennzeichnet).

Die Business-Logik ist schon etwas schwieriger zu erkennen. Wenn weitere Anforderungen wie Transaktionssicherheit oder Autorisierung hinzukommen, wird es sehr schnell unübersichtlich.

Rettung naht

Wie würde man die obigen Anforderungen in der Aspekt-Orientierung ausdrücken? In etwa so: Füge an allen Stellen, an denen das Attribut „kontostand" geändert wird, eine Log-Anweisung hinzu. Mit AspectJ (einer Aspekt-orientierten Erweiterung von Java) sähe die Realisierung dieser Anforderung ungefähr so aus (siehe Listing 3).

„aspect" ist der Rahmen um das Ganze, in dem die eigentliche Anforderung implementiert wird. Die tatsächliche Anweisung ist kursiv hervorgehoben und entspricht in etwa dem, was im obigen Kasten als Pseudo-Anweisung geschrieben steht. Damit hat man es geschafft, diese neue Anforderung der Protokollierung in einem eigenen Aspekt zu kapseln (siehe Abbildung 1).

Außer beim Zugriff auf Klassen-Attribute kann man noch bei folgenden Punkten im Programm eingreifen:

- Aufruf einer Methode
- Ausführen einer Methode
- Behandeln einer Exception
- Initialisierung einer Klasse
- Initialisierung eines Objekts

Diese Punkte heißen im AOP-Jargon auch „Joinpoints". Über „Advices" können diese Stellen dann um weitere Funktionalitäten ergänzt werden. Advices sind so etwas wie die Methoden der Aspekt-Orientierung – im obigen Beispiel war es die (einzelne) Log-Anweisung.

Damit haben wir alle Zutaten beisammen, um die Anforderungen auf Klassen (wie bisher) oder Aspekte aufzuteilen und dem Ziel, der „Separation of Concerns", ein

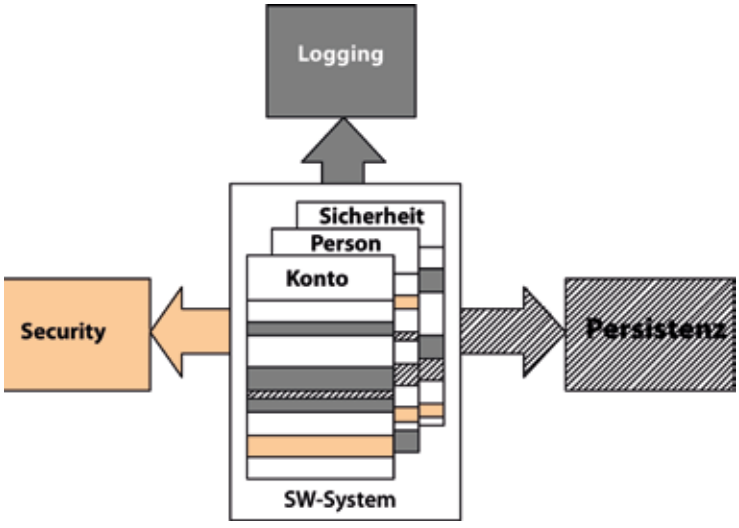


Abbildung 2: Das System als Menge von „Concerns"

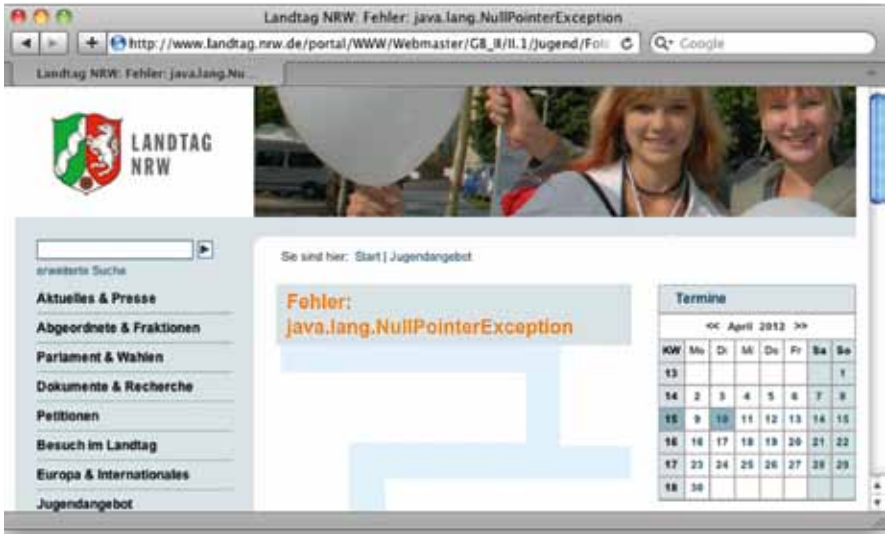


Abbildung 3: Das Jugendangebot des Landtags von NRW (Stand: 10. April 2012)

gutes Stück näher zu kommen. Und das Schönste daran ist, dass wir bei der Entwicklung nicht immer an alles auf einmal denken müssen, sondern uns jetzt immer nur auf einen „Concern" konzentrieren können (siehe Abbildung 2).

Neues Altes aus der Java-Welt

Verlassen wir kurz die Welt der Aspekte und wenden wir uns wieder der Java-Welt zu. 2002 erschien Java 1.4, das als Spracherweiterung „asserts" mitbrachte. Damit kann man Sicherungen im Code einbauen, um die Fehlersuche zu beschleunigen (siehe Listing 4).

Aktiviert werden „asserts" über den Schalter „-ea" (enable assertions) beim Start der Java-VM. Übergibt man dann während der Test-Phase (in der man üblicherweise diesen Schalter aktiviert) einen Null-Parameter an diese Methode, gibt die assert-Anweisung eine Warnmeldung aus, dass die zugesicherte Bedingung verletzt wurde, und das Programm wird beendet. Dadurch kommt man der Fehlerursache sehr viel früher und schneller auf die Spur. Da sich dieses Test-Hilfsmittel noch nicht bei allen Entwicklern herumgesprochen hat, nachfolgend ein Vorschlag, wie man damit NullPointerExceptions fangen kann.

Wie man NullPointerExceptions fängt

NullPointerExceptions sind mit die ärgerlichsten Fehler, die anzutreffen sind (siehe Abbildung 3). Bei der Eingabe von „NullPointerException filetype:jsp" als Suchbegriff in Google erhält man alle JSP-Seiten, in denen „NullPointerException" auftaucht (darunter auch einige Foren – die man getrost übergehen kann). Dabei ist es eigentlich relativ einfach, NullPointerExceptions zu vermeiden – man lässt einfach keine Null-Werte zu, weder als Argument, noch als Rückgabewert (siehe Listing 5).

Wenn jetzt während der Testphase (und umgelegtem „-ea"-Schalter) doch ein null-Argument übergeben wird, ist der Schuldige schnell gefunden. Auch wenn man damit NullPointerExceptions drastisch reduzieren kann, ist es ziemlich öde, diese assert-Anweisungen manuell am Anfang und Ende jeder Methode einzufügen. Hier kommt wieder die Aspekt-Orientierung ins Spiel: Sie erlaubt uns, diese „Querschnitts"-Anforderung zu formulieren (siehe Listing 6). Damit wird der zweite Teil der obigen Anforderung realisiert: Am Ende jeder Methode mit Rückgabewert eine assert-Anweisung einfügen.

Eine der Stärken von AOP ist, dass man über Wildcards mehrere Stellen (im AOP-Jargon auch als Joinpoint bezeichnet) im Programm ansprechen kann, um wie hier zusätzliche Überprüfungen einzufügen. Um auf die Eingangsfrage zurückzukommen („Wie fängt man NullPointerExceptions?"): Gar nicht! Man vermeidet einfach „null" als gültigen Wert, indem man sich an folgende Dinge hält:

- Mit Exceptions wie mit einer FinderException arbeiten, um „kein Ergebnis" oder andere Ausnahme-Situationen zu signalisieren
- Objekte mit einer Null-Semantik (wie „Collections.EMPTY_LIST") einführen, die als Ersatz für „null" als Parameter oder Rückgabewert dienen können

Die Anfänge von PatternTesting

Die gedankenlose Verwendung von null-Werten ist nur eine von vielen Ursachen für Fehler im ausgelieferten Programm. Weitere, immer wieder gern gemachte Fehler sind fehlende Freigaben von Ressourcen, Deadlock-Situationen, vergessene Exception-Handler (mit generierter „e.printStackTrace()"-Anweisung) und viele andere mehr. Einige dieser Kandidaten lassen sich durch statische Code-Analyse und Tools wie FindBugs oder PMD aufspüren; für andere Fehlerquellen wie den sorglosen Umgang von null als Parameter oder Rückgabewert müsste man in den Code eingreifen, um sie aufzudecken.

Wie wir aber bereits gesehen haben, ist dies mit der Aspekt-Orientierung möglich und führte dazu, dass Vincent Massol im März 2002 mit PatternTesting eine der ersten AOP-Bibliotheken aus der Taufe hob. Anfangs war PatternTesting mehr ein „Proof of Concept", um Architektur- und Implementierungs-Entscheidungen automatisch überprüfen zu können. Es war als Sammlung von Aspekten angelegt, die man erweitern und spezifizieren konnte, wie weit diese Prüfungen reichen sollte (sofern man sich mit AspectJ auskannte). Es gab einen AbstractNullTest, der auf „null" als Argument oder Rückgabewert hinwies, einen AbstractSop-Aspekt, der bei „System.out.println(..)"-Anweisungen Compiler-Warnungen ausgab oder einen AbstractDatabaseTest, mit dem man beim Einsatz von Hibernate oder anderer Persistenz-Frameworks Aufrufe aus java.sql unterbinden konnte.

AspectJ 1.0 und 1.1

In der Anfangsphase arbeitete der AspectJ-Compiler noch als eine Art Präprozessor, der die Aspekte in Java-Code umwandelte, bevor er sie an den Java-Compiler weiterreichte. Entsprechend war die erste Version (v0.2) von PatternTesting auch ein Proof of Concept für den Einsatz des AspectJ-Compilers, zumal auch die Tool- Un-

GLOSSAR

Concern
Spezifische Anforderung oder Gesichtspunkt, welche(r) in einem Software-System behandelt werden muss, um die übergreifenden Systemziele zu erreichen (nach Ramnivas Laddad, 2003)

Querschnittsbelang (Crosscutting Concern)
Dies sind Anforderungen und Dinge wie Transaktionen, Autorisierung oder Security, die sich durch alle Klassen ziehen und sich mit Mitteln der Objekt-Orientierung nur schwer kapseln lassen.

Joinpoint
Ein Punkt im Programm, an dem man einen –> Advice zur Ausführung bringen kann. Joinpoints sind zum Beispiel der Aufruf oder die Ausführung von Methoden, der Zugriff auf Attribute oder die Initialisierung von Objekten.

Pointcut
Pointcuts sind ein AOP-Sprachmittel und dienen dazu, einzelne oder mehrere Joinpoints zusammenzufassen.

Advice
Der Code, der an den –> Pointcuts ausgeführt wird.

Compile-Time-Weaving (CTW)
Beim Compile-Time-Weaving wird vom Compiler aus den Java-Klassen und Aspekten direkt während des Compile-Vorgangs Code erzeugt.

Load-Time-Weaving (LTW)
Beim Load-Time-Weaving wird über einen Weaving-Agent während des Lade-Vorgangs der Byte-Code der geladenen Klassen erweitert.

Avalon-Framework
Ein ehemaliges Komponenten-Framework für serverseitige Container, das inzwischen eingestellt wurde.

terstützung für die Entwicklung noch in den Kinderschuhen steckte.

Dies änderte sich mit AspectJ 1.1, das 2003 auf dem Markt erschien. Damit wandelte sich die Architektur grundlegend. Der AspectJ-Compiler setzte jetzt auf dem Java-Compiler aus Eclipse auf und konnte so direkt Bytecode erzeugen. Auch die inkrementelle Kompilierung hielt damit Einzug in den AspectJ-Compiler und sorgte zusammen mit AJDT, dem AspectJ-Development-Tool-Plug-in für Eclipse, dafür, dass die IDE- Unterstützung immer besser wurde. Mit PatternTesting 0.3 wurde 2004 auf AspectJ 1.1 umgestellt und Unterstützung für das Avalon-Framework hinzugefügt. Gleichzeitig übernahm Matt Smith die Projekt- Leitung.

Java 5 und AspectJ 5

Mit JDK 1.5 (das von Sun als Java 5 verkauft wurde) hielten 2005 einige Neuerungen Einzug in die Sprache: Generics, Autoboxing oder Annotations. Auch bei der AOP- Unterstützung für Java tat sich einiges: AspectJ und AspectWerkz fusionierten. AspectWerkz war neben AspectJ ebenfalls eine Aspekt-orientierte Erweiterung für Java, webte aber die Aspekte beim Laden einer Klasse ein (LTW: Load Time Weaving). Dies ist der Grund dafür, dass AspectJ heute neben dem Compile Time Weaving (CTW) auch diesen Weg unterstützt, um die Aspekte in den Code zu bekommen. Beide Verfahren haben Vor- und Nachteile, sodass es durchaus sinnvoll ist, sie zu unterstützen:

- Beim Compile Time Weaving (CTW) werden Laufzeit-Fehler vermieden, da die Aspekte bereits durch den Compiler überprüft und übersetzt wurden.
- Mit Load Time Weaving (LTW) können auch Bibliotheken von Dritt-Anbietern instrumentiert („kompiliert“) werden. Prinzipiell ist dies auch mit CTW möglich, allerdings verbieten manche Lizenz-Bestimmungen die Veränderung von Jar- Dateien.

Gleichzeitig passte man sich mit der Nummerierung der Java-Versionierung an. Nach AspectJ 1.2 (das sich von AspectJ 1.1 nur durch eine Verbesserung der Tool-Kette unterschied) kam AspectJ 5, das sich (analog zu Java) intern als Version 1.5 zu erkennen gab. Wichtigste Neuerung war neben der Fusion mit AspectWerkz und der Unterstützung von Load Time Weaving die Verwendung und Unterstützung von Annotations – sie können jetzt auch zur Auswahl von Joinpoints herangezogen werden.

Nehmen wir an, wir wollen für die Login-Methode „null“ als gültigen Rückgabewert zulassen, dann könnten wir dies durch eine @MayReturnNull-Annotation anzeigen (siehe Listing 7). Listing 8 zeigt eine Annotation, die von PatternTesting verwendet wird, um diese Methoden von der Überprüfung auszuschließen.

Während man in PatternTesting 0.3 noch AspectJ-Kenntnisse benötigte, um die bereitgestellten (abstrakten) Aspekte in ein Projekt einzubinden, kann ab PatternTesting 0.5 (das 2008 nach vierjähriger Ruhezeit erschien) diese Bibliothek auch ohne solche Kenntnisse eingesetzt werden, da alle Aspekte über Annotationen gesteuert werden können. Nach wie vor ist aber auch die Erweiterung bestehender Aspekte vorgesehen und möglich. Als weiterer Vorteil haben die verwendeten Annotationen auch Dokumentations-Charakter. So zeigt die „@MayReturnNull“-Annotation dem Entwickler an, dass die Login-Methode auch „null“ als Rückgabe-Wert liefern kann.

Von Maven 1 nach Maven 2

2008 war ein sehr bewegtes Jahr in der Geschichte von PatternTesting. Der Autor hatte ein Jahr zuvor Kontakt zu Vincent Massol und Matt Smith und ihnen unter anderem seine Hilfe bei PatternTesting angeboten, da seit ein paar Jahren keine Weiterentwicklung mehr zu erkennen war. Ehe er sich versah, hatte er die Projekt-Leitung inne, da beide mit anderen Projekten beschäftigt waren.

Anfangs befasste der Autor sich noch hauptsächlich damit, den Build und die Projekt-Struktur von Maven 1 auf Maven 2 umzustellen. Dabei leistete ihm das „aspectj-maven“-Plug-in wertvolle Dienste bei der Kompilierung. Als Infrastruktur für den Build wurde anfangs Continuum von Apache eingesetzt, inzwischen läuft der Build mit Jenkins auf einem ausgedienten Mac-Mini (PowerPC, Ubuntu). Während für die sonstige Infrastruktur (CVS, Wiki) Sourceforge.net sehr nützlich ist, sieht es im Bereich der öffentlichen Build-Server noch mau aus. Einzig CloudBees.com scheint

hier Angebote für Open-Source-Projekte zu unterbreiten, deren kostenlose Ressourcen sich aber als nicht ausreichend für PatternTesting erwiesen haben. Die nächsten Schritte waren dann die Umstellung auf AspectJ 5, die Verwendung von Annotationen sowie die Weiterentwicklung der vorhandenen Aspekte.

Performance-Probleme bis AJDT 1.6.1

Mit dem Anwachsen der Code-Basis und der Entwicklung neuer Aspekte litten vor allem größere Projekte unter der fehlende Performance des AJDT-Plug-ins in Eclipse 3.3. Dies änderte sich erst mit Eclipse 3.4 und AJDT 1.6.2. Vor allem die Möglichkeit, eigene Warnungen und Fehler schon während der Compile-Phase ausgeben zu lassen, erhöhten die Wartezeiten nach dem Speichern der Sourcen, wenn die automatische Kompilierung angestoßen wurde. Dies führte schließlich dazu, dass PatternTesting aufgeteilt wurde in:

- PatternTesting Runtime (die Basis für alle anderen Unter-Projekte)
- PatternTesting Check-CT (Compile-Time-Checks)
- PatternTesting Check-RT (Runtime-Checks)

Dadurch war man jetzt in der Lage, während der Arbeit mit Eclipse die Compile-Time- Checks wegzulassen und sie nur während des Daily Builds zu aktivieren. Weiterer Vorteil dieser Trennung war, dass man auch die Runtime-Checks weglassen konnte (z.B. nach der Test-Phase, wenn die Anwendung ausgeliefert werden soll), ohne dass der Source-Code geändert werden muss. Lediglich PatternTesting Runtime muss mit ausgeliefert werden, da dort die ganzen Annotationen enthalten sind.

Ein Jahr später kam dann mit PatternTesting Exception noch ein weiteres Projekt für die Unterstützung aussagekräftigerer Exceptions hinzu. Wenn etwa ein Socket mit „Socket socket = new Socket(„10.11.12.13“, 80);“ geöffnet wird und der angegebene Rechner (10.11.12.13) nicht erreichbar ist, bekommt man eine ConnectException mit der lapidaren Meldung „Connection refused“. Wäre es nicht sinnvoller, wenn stattdessen „Connection to 10.11.12.13 refused“ ausgegeben werden würde? Dies ist genau das Einsatz-

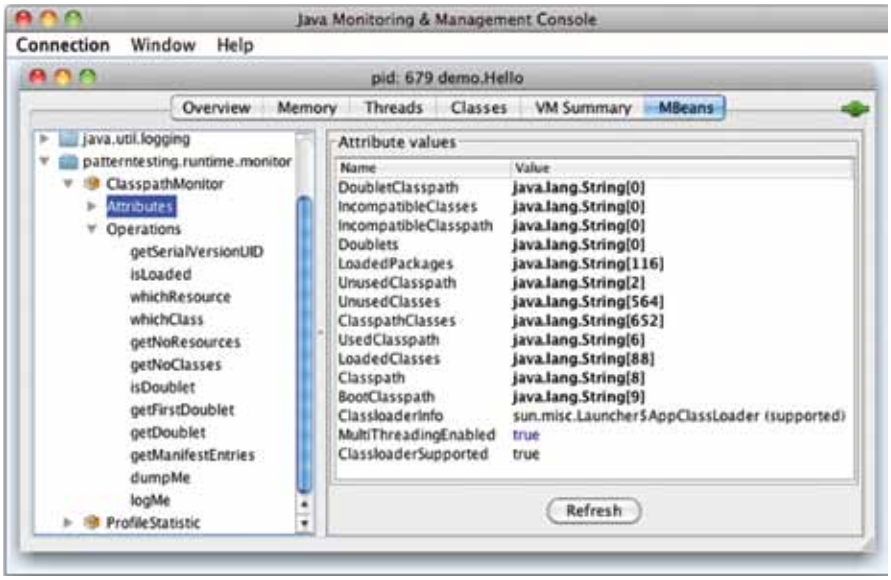


Abbildung 4: Der ClasspathMonitor innerhalb der JConsole

Szenario von PatternTesting Exception. Einige Exceptions werden um zusätzliche Informationen angereichert, um die Fehlersuche zu erleichtern. Und man erhält darüber hinaus Unterstützung für das Testen der Exception-Handler.

PatternTesting 2010

2010 war es dann endlich soweit: Die Version 1.0 wurde am 20. Juni 2010 ausgeliefert. Gleichzeitig gab es mit http://patterntesting.org eine neue Heimat, die von der Agentes GmbH gesponsert wird. Damit änderten sich auch die Maven- Koordinaten für die aktuelle Version von PatternTesting Runtime (siehe Listing 9). Seit Version 1.0.3 ist PatternTesting auch wieder im zentralen Maven-Repository zu finden, nachdem es zwischenzeitlich durch diverse Umstellungen dort längere Zeit nie angekommen war.

JUnit-Unterstützung

Mit 1.0 gab es auch einen eigenen JUnit-Runner, um bestehende Tests bei Bedarf ausblenden oder als „kaputt“ kennzeichnen zu können (siehe Listing 10).

Mit dem SmokeRunner können Tests als „Broken“ (mit Fix-Datum) oder als Integrations-Tests (die während dem Entwickler-Test ausgeblendet werden) gekennzeichnet werden. Auch können Tests an bestimmte Plattformen oder Bedingungen verknüpft werden, um ausgeführt zu werden.

Der kleine Bruder des SmokeRunners ist der ParallelRunner (aus PatternTesting Concurrent), der die gleiche Funktionalität besitzt, aber zur Beschleunigung der Tests alle Test-Methoden parallel ausführt. Eine weitere Beschleunigung ermöglicht die ParallelSuite (ab 1.2), die alle in einer Suite zusammengefassten Klassen parallel startet.

Durch die Abhängigkeit von PatternTesting Runtime zu JUnit 4.8 laufen diese Runner leider nicht in älteren Eclipse-Versionen. Dies wird sich aber mit der nächsten Version ändern – hier werden auch ältere Versionen bis Eclipse 3.4 (das z.B. Basis für IBMs RAD 7.5 ist) unterstützt werden.

```
<groupId>org.patterntesting</groupId>
<artifactId>patterntesting-rt</artifactId>
<version>1.2.10-YEARS</version>
```

Listing 9

```
@RunWith(SmokeRunner.class)
public class Rot13Test {
@Broken(till = "01-Jun-2012")
@Test
public final void testCryptBobsFile() throws
IOException { ...
}
@IntegrationTest("online access needed")
@Test
public final void testCryptURI() throws IOException {
...
}
}
```

Listing 10

Rückblick und Ausblick

Was haben uns die letzten zehn Jahre im Java-Umfeld gebracht? Der größte Sprung war sicherlich der von Java 1.4 auf Java 5, der uns neue Sprach-Features bescherte. Bei Java 6 lag der Fokus auf dem Desktop, Java 7 ließ fast 5 Jahre auf sich warten, nachdem viele geplante Features auf Java 8 verschoben wurden.

Bei AspectJ sind vor allem AspectJ 5 und die Vereinigung mit AspectWerkz hervorzuheben. Danach hat sich am Sprach-Kern nichts geändert, lediglich die Performance und die IDE-Unterstützung wurden stetig verbessert. Die Versionsnummer wurde dabei an Java angepasst – aktuell ist AspectJ 1.7 (oder kurz: AspectJ 7).

Der große Durchbruch für AspectJ lässt noch auf sich warten. Aber AOP-Techniken haben sich in der Zwischenzeit ihren Platz in der Java-Welt erobert, um zusätzliche Anforderungen im Code unterzubringen, ohne bestehende Klassen zu verändern. Manche Frameworks agieren dazu als Java-Agent oder als Proxy (zum Beispiel einige Mock-Frameworks), andere setzen dazu einen eigenen Classloader ein (etwa JBoss-AOP).

PatternTesting hat sich von einer reinen AspectJ-Bibliothek zu einem Hybrid entwickelt. Die Runtime-Komponente kann auch als reine Java-Bibliothek eingesetzt werden und man erhält neben dem SmokeRunner noch einige Tester, um beispielsweise „equals(..)"- und „hashCode()"-Implementierungen zu überprüfen, die Serializable-Eigenschaft zu testen oder Dateien auf Gleichheit zu überprüfen. Ein ClasspathMonitor gibt Aufschluss über den Classpath und findet unter anderem doppelte und inkompatible Klassen (siehe Abbildung 4).

Leider funktioniert der ClasspathMonitor nur mit der Sun-VM korrekt. Für die Java-VM von IBM ist jedoch Besserung in Sicht: Hier wird es einen PatternTesting Agent geben, der die notwendigen Informationen für den ClasspathMonitor sammelt. Was bleiben wird, ist die Unterstützung für Java 5, da es noch einige Projekte gibt, die nicht auf ein aktuelleres JDK umschwenken können oder dürfen. Dazu passt auch die Unterstützung älterer JUnit-Versionen, um PatternTesting auch mit älteren Eclipse-Versionen einsetzen zu können. Ferner ist die Unterstützung weiterer Exceptions (wie der SQLException) geplant, um im Falle eines Falles aussagekräftigere Fehlermeldungen zu bekommen. Wer jetzt Lust auf PatternTesting bekommen hat, findet im PatternTesting-Wiki unter „Getting Started" (siehe http://sourceforge.net/apps/mediawiki/patterntesting/index.php? title=Getting_Started) ein einfaches Hello-World-Beispiel sowie mit „Testing with PatternTesting" eine Anleitung, wie man den SmokeRunner und die verschiedenen Tester einsetzen kann.

Links

1. PatternTesting: http://patterntesting.org/
2. PatternTesting Wiki: http://sourceforge.net/apps/mediawiki/patterntesting/ AspectJ-Homepage: http://eclipse.org/aspectj/
3. Performance Boost with Eclipse 3.4: http://oli.blogger.de/20090323/

Oliver Böhm

oliver.boehm@t-systems.com

Oliver Böhm beschäftigt sich mit Java-Entwicklung unter Linux und Aspekt-Orientierter SW-Entwicklung. Neben seiner hauptberuflichen Tätigkeit als JEE-Architekt bei T-Systems ist er Buchautor, Projektleiter bei PatternTesting und Board-Mitglied der Java User Group Stuttgart.

„Von den Erfahrungen der anderen zu profitieren, ist essentiell …"

Usergroups bieten vielfältige Möglichkeiten zum Erfahrungsaustausch und zur Wissensvermittlung unter den Java-Entwicklern. Sie sind aber auch ein wichtiges Sprachrohr in der Community und gegenüber Oracle. Wolfgang Taschner, Chefredakteur der Java aktuell, sprach darüber mit Tony Fräfel, dem Vorsitzenden der Swiss Oracle User Group (SOUG).

Wie bist du zur SOUG gekommen?

Fräfel: 1987 hat sich mein damaliger Arbeitgeber für das rDBMS Oracle entschieden. Die Oracle-Kundschaft in der Schweiz war damals noch überschaubar und ein Erfahrungsaustausch sehr willkommen. In der SOUG fanden wir erfahrene Oracle-Spezialisten und ich durfte meine Firma dort vertreten.

Wie ist die SOUG organisiert?

Fräfel: Die SOUG ist ein unabhängiger Verein. Ein siebenköpfiger Vorstand ist verantwortlich für die Aktivitäten. Er wird unterstützt von einem Komitee, das für die Aktivitäten in der französischsprechenden Schweiz verantwortlich ist, sowie von unserem Sekretariat.

Was zeichnet die SOUG aus?

Fräfel: Im Fokus steht klar der Erfahrungsaustausch. Unsere Mitglieder profitieren von den praktischen Erfahrungen, die andere Mitglieder gemacht haben. Wir ermöglichen es, Kontakte auf nationaler und internationaler Ebene zu knüpfen und zu pflegen. Wichtig ist uns auch unsere Unabhängigkeit – wir vertreten die Interessen unserer Mitglieder, nicht diejenigen von Herstellern.

Wie viele Veranstaltungen gibt es pro Jahr?

Fräfel: Zurzeit sind es circa vier Special-Interest-Group-Meetings pro Jahr in der

deutschsprachigen Schweiz und etwa zwei in der Westschweiz. Dazu starten wir unter dem Namen „SOUG bi dä Lüt" (SOUG bei den Leuten) mit lokalen Meetings am späteren Nachmittag. Nicht zu vergessen ist unsere Teilnahme an der DOAG Konferenz + Ausstellung in Nürnberg, die aufgrund unserer intensiven Zusammenarbeit mit der DOAG zur SOUG-Jahreskonferenz avancierte. Durch unsere Kooperationen mit anderen Usergroups haben unsere Mitglieder auch die Möglichkeit, an vielen Veranstaltungen zu günstigen Mitgliederkonditionen teilzunehmen.

Was motiviert dich besonders, als Vorstand die SOUG zu führen?

Fräfel: Eine Usergroup macht für mich Sinn. Als Mitglied habe ich viel profitiert und mit meinem Engagement im Vorstand möchte ich dafür sorgen, dass möglichst viele Leute auch davon profitieren können.

Was bedeutet Java für dich?

Fräfel: Nach 4GL und CASE war Java für mich zuerst ein Rückschritt und ich fühlte mich wie zu Beginn meiner Informatik-Karriere als Assembler-Programmierer. Die Vorteile der Objektorientierung und Plattformunabhängigkeit wurden mir jedoch schnell bewusst. Auch wenn ich heute nicht mehr selber programmiere, ist Java für mich ein wichtiges Instrument, um die immer komplexer werdende IT-Infrastruktur im Griff zu haben. Unterschiedliche Plattformen wird es auch in Zukunft geben – es ist aber nicht nötig, dass jede Plattform ihre eigene, proprietäre Programmiersprache besitzt.

Welchen Stellenwert besitzt die Java-Community für dich?

Fräfel: Der Erfahrungsaustausch ist für mich im vielfältigen Java-Umfeld noch wichtiger als bei den Oracle-Produkten. Niemand kann sich all die Frameworks und Weiterentwicklungen anschauen. Von den Erfahrungen der anderen zu profitieren, ist essentiell. Mit der Mitgliedschaft der SOUG im iJUG ermöglichen wir unseren Mitgliedern die Vernetzung mit noch mehr Java-Spezialisten und können noch attraktivere Java-Events in der Schweiz anbieten.

Tony Fräfel, Präsident der Swiss Oracle User Group (SOUG)

Was hast du bei der Übernahme von Sun durch Oracle empfunden?

Fräfel: Mit der SOUG waren wir ja sozusagen auf der Seite von Oracle. Mein erster Gedanke war, dass Oracle den schon lange versuchten Einstieg ins Hardware-Business mit Sun nun wohl schaffen werde. Interessant war für mich aber die Frage, was mit den Software-Assets von Sun geschehen würde. Die meisten Produkte würden wohl, wie in solchen Situationen üblich, in die Oracle-Produkte integriert oder stillgelegt werden. Was aber mit Java und MySQL? Beide sind für mich Symbole von Open Source und Oracle ist, wie wir alle wissen, sehr profitorientiert. Java kommerziell auszuschlachten würde wohl auch Oracle nicht gelingen, und wenn sie Java stilllegen wollten, würden es andere unabhängig von Oracle weiterentwickeln. Auf jeden Fall war ich gespannt, wie es mit Java weitergehen würde.

Wie sollte sich Java weiterentwickeln?

Fräfel: Ganz klar als Open Source. Nur so bleibt für mich die Plattformunabhängigkeit erhalten. Das ist in meinen Augen eine große Chance für die Hersteller – sowohl hardware- als auch softwareseitig.

Wie sollte Oracle deiner Meinung nach mit Java umgehen?

Fräfel: Oracle soll dafür sorgen, dass Java Open Source bleibt und die namhaften Hersteller an der Weiterentwicklung beteiligt sind. Indirekt profitiert dann auch Oracle, obwohl, wie der Rechtsstreit mit Google zeigt, Oracle natürlich auch direkt profitieren will. Ein Sieg von Oracle wäre in meinen Augen schlecht für Java und könnte sich auch für Oracle als Bumerang erweisen. Ich bin gespannt, wie das Gericht entscheidet.

Wie sollte sich die Community gegenüber Oracle verhalten?

Fräfel: Als SOUG arbeiten wir eng und gut mit Oracle zusammen. Wir sind aber nicht mit allem einverstanden, was Oracle macht. Konstruktive Kritik wird von unseren Oracle-Kontakten offen aufgenommen und nach Möglichkeit auch berücksichtigt. Die Unabhängigkeit von Oracle ist in der Java Community noch stärker verankert als in der SOUG und das soll auch so bleiben. Unabhängig heißt für mich aber miteinander und nicht gegeneinander. Ein konstruktiver Dialog soll von beiden Seiten angestrebt werden. Mein Eindruck ist, dass verschiedenste Vertreter der Java Community von Oracle mit offenen Armen empfangen werden und damit einen starken Einfluss auf die Weiterentwicklung von Java haben.

Tony Fräfel
tony.fraefel@soug.ch

Zur Person: Tony Fräfel

Anfang der 1980er Jahre begann Tony Fräfel seine Informatik-Laufbahn als Programmierer/Analytiker. Von 1987 bis 2000 war er im Umfeld von Oracle als Consultant und Projektleiter tätig, danach in verschiedenen Management-Positionen im Bereich Anwendungsentwicklung und Integration. Heute ist er als Senior Manager bei der Trivadis AG verantwortlich für das Ressourcen-Management. Tony Fräfel engagiert sich seit 1991 im Vorstand der SOUG: zuerst als Redakteur des SOUG Newsletters, seit 2009 als Präsident.

Cloud Foundry: die Spring Cloud

Eperon Julien, Trivadis AG

Die Java-kompatible Cloud-Plattform „Spring Cloud" wird durch Cloud Foundry bereitgestellt. Der Artikel zeigt den Reifegrad der Plattform und ihre Eignung für unternehmensweite Anwendungen. Zudem erleichtert ein Vergleich von Spring Cloud mit anderen Cloud-Angeboten das Verständnis und die Beurteilung der Technologie.

Zum Einstieg eine kleine Einführung in die drei Kategorien von Cloud-Angeboten:

- *Software-as-a-Service (SaaS)*
  SaaS ist ein Softwareverteilungs-Modell, bei dem Anwendungen entweder durch den Anbieter der Anwendung oder durch einen separaten Provider gehostet werden. Typische Vertreter sind:
  - Salesforce.com (http://www.salesforce.com): Dieses Unternehmen bietet seit rund zehn Jahren ein CRM-System als Service an. Salesforce hat in dem Bereich eine solide Reputation aufgebaut.
  - Business Productivity Online Standard (BPOS, http://www.microsoft.com/online/): Eine von Microsoft bereitgestellte Online-Office-Lösung und als Alternative zu den lokal installierten Microsoft-Office-Produkten positioniert.
- *Platform-as-a-Service (PaaS)*
  Das Paradigma PaaS bietet via Internet erreichbare Server mit installiertem und gewartetem Betriebssystem. Typische Vertreter sind:
  - Windows Azure (http://www.microsoft.com/windowsazure/): Dies ist die Cloud-Infrastruktur von Microsoft.
  - Google App Engine (http://code.google.com/appengine/): Google App Engine ist die Cloud-Lösung von Google.
- *Infrastructure-as-a-Service (IaaS)*
  Diese Kategorie bietet die größten Freiheiten bei der Wahl der Software, jedoch gleichzeitig die wenigsten Optionen hinsichtlich Vorkonfiguration und vorhandener Services. IaaS bietet lediglich die Server-Hardware und zugehörige Technologie (Netzwerk, Storage, Data-Center-Space) an. Installation und Konfiguration eines Betriebssystems sowie jeglicher Software sind dem Kunden überlassen. Oft stehen aber vorkonfigurierte Images als Basis zur Verfügung.
  - Amazon EC2 (http://aws.amazon.com/ec2/): Einer der ersten großen Cloud-Anbieter. Cloud Foundry benutzt diesen Service im Hintergrund.

Cloud-Foundry-Grundlagen

Die Cloud-Foundry-Plattform hat in ihrer Geschichte die zweite Evolutionsstufe erreicht. Die erste Version baute auf den Amazon-Services EC2 und S3 auf und war auch von dieser Infrastruktur abhängig. Der Hauptbestandteil der ersten Version war ein Abstraktionslayer in Form einer Management-Console. Mit dieser ließen sich Java-Web-Anwendungen und dazugehörende Dateien in die Cloud hochladen und verwalten. Die aktuelle Version beschränkt sich nicht mehr nur auf Java, sondern unterstützt beispielsweise auch Ruby und Node.js. Zudem müssen als Infrastruktur-Grundlage auch nicht mehr zwingend die Amazon-Angebote verwendet werden. Die erste Version ist als „Classic"-Version immer noch verfügbar (siehe „Links" am Ende des Artikels).

Cloud Foundry befindet sich immer noch im Beta-Stadium und bietet unter anderem auch deshalb noch keine kommerzielle Version an. Die Plattform basiert ausschließlich auf Open-Source-Software. Sie unterscheidet sich von anderen Cloud-Angeboten hauptsächlich in der Art, wie Anwendungen dafür entwickelt werden. Üblicherweise gehen die Cloud-Angebote davon aus, dass man seine bisherige, dedizierte Umgebung in der Cloud abzubilden versucht und lediglich Aspekte wie Persistenz oder Hochverfügbarkeit auf das jeweilige Cloud-Angebot anpasst. Es ist bei solchen Angeboten nicht möglich, seine Anwendung offline zu testen, und auch nicht, eine realitätsnahe Kopie der Cloud-Umgebung lokal aufzusetzen. Typischerweise richtet man eine dedizierte Entwicklungs- und Testumgebung in der Cloud ein.

Cloud Foundry geht hier einen ganz anderen Weg. Das zentrale Angebot besteht aus einer Virtual Machine, genannt „Micro Cloud Instance". Diese VM lässt sich auf der lokalen Entwicklermaschine im VMware-Player starten. So hat der Entwickler die gesamte Cloud lokal bei sich. Die Vorteile liegen auf der Hand: Der Entwickler ist völlig unabhängig von einem Netz-Zugang und hat immer alles, was er benötigt, bei sich. Für den Cloud-Anbieter lohnt es sich ebenfalls, denn es dümpeln bei ihm keine Entwicklungs-Instanzen herum, die dauerhaft Ressourcen belegen, aber nur selten benutzt werden. Es ist deshalb gut möglich, dass dieses Modell Schule macht und dieser Ansatz ein Standard für Cloud-basierte Entwicklung wird.

Mit Cloud Foundry verkürzt sich die Zeit wesentlich, bis eine Anwendung „up and running" ist. Beschaffung und Inbetriebnahme der Hardware fallen komplett weg. Die grundlegende Installation und Konfiguration von Betriebssystem, Datenbank und Application-Server übernimmt Cloud Foundry. In der Micro Cloud VM ist alles Notwendige bereits installiert und konfiguriert. Falls erforderlich, kann der Entwickler über die Kommandozeile selber Hand bei der Konfiguration der VM-Instanz anlegen.

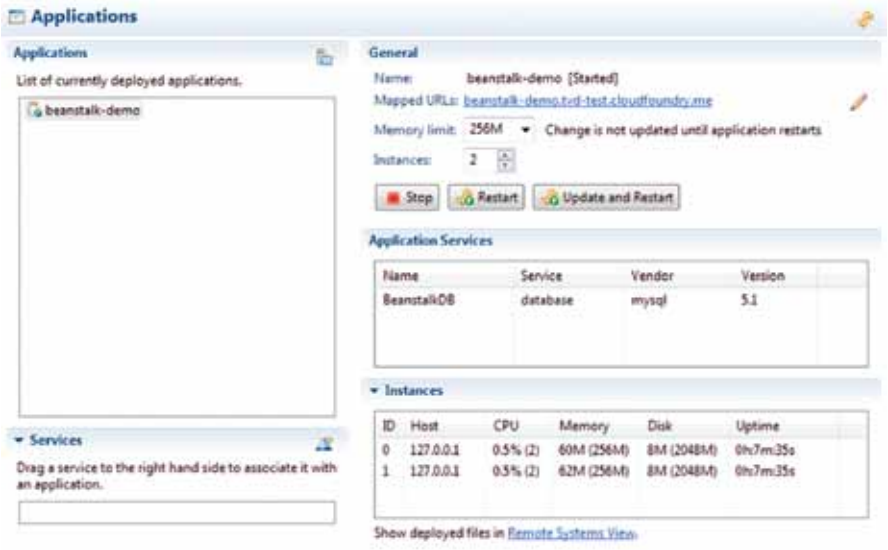


Abbildung 1: Verwaltung einer installierten Anwendung aus der STS heraus

Aus Sicht des Entwicklers ist die Zielplattform durch die Micro Cloud VM vorgegeben. Darin ist Cloud Foundry der Google App Engine ähnlich. Auf der Micro Cloud VM hat der Entwickler aber Zugriff auf Maschinen-Ebene, ähnlich wie er es in einer Amazon EC2-Instanz hat. Zudem lassen sich Cloud-Foundry-Anwendungen potenziell in jeder Cloud publizieren, die das CF-API implementiert hat. Dieses API ist Open-Source und lizenzkostenfrei. Falls sich das Modell durchsetzt, werden andere Cloud-Anbieter davon profitieren wollen. Zurzeit ist allerdings kein weiterer CF-API-Anbieter bekannt.

Im Vergleich zu anderen Cloud-Angeboten wie der Google App Engine, die die Basis-Plattform dem Benutzer nicht zugänglich machen, bietet die Cloud Foundry wesentlich umfangreichere Kontrolle und größere Flexibilität. Die Art der Applikationen auf diesen beiden Plattformen kann jedoch sehr ähnlich sein. Bekannterweise schränkt Google jedoch auch den Web-Entwickler ein. So ist nur ein Subset der Klassen aus dem JDK verwendbar. Zudem schreibt Google statt einer relationalen Datenbank die Verwendung ihres Datastores vor. Dafür muss man sich in einer Cloud-Foundry-Anwendung mehr darum bemühen, diese skalierbar zu entwickeln. Die Vorgaben der App Engine führen direkt zu einer skalierbaren Architektur. Dies erlaubt es Google wiederum, bei Bedarf einfacher zu skalieren (siehe Tabelle 1).

Die Benutzung der Micro Cloud Foundry ist momentan noch gratis. Allerdings werden damit auch hauptsächlich die Entwickler angesprochen. Es wird in Zukunft ein kommerzielles Angebot auf cloudfoundry.com geben. Gerüchteweise sollen die Preise mit denen einer Private Cloud auf vSphere-Basis vergleichbar sein. Von Cloud Foundry gibt es jedoch noch keine offiziellen Preispläne oder Preisvergleiche.

Technische Details

Micro Cloud Foundry empfiehlt die Benutzung ihrer eigenen Virtual Machine, die in einem VMware Player oder einem kompatiblen Produkt gestartet werden kann. Sie enthält die folgenden Software-Pakete, ausnahmslos Open-Source-Produkte (gültig für Version 1.1.0 der Micro Cloud VM):

- Java 1.6 (JRE 1.6.0_24-b07): Die Java Virtual Machine von Oracle (http://www.java.com)
- Nginx 0.8.54: Ein HTTP-Server und Reverse-Proxy (http://www.nginx.org/)
- Tomcat 6.0.32: Der oft benutzte Java Web Application Server von Apache (http://tomcat.apache.org/)
- MySQL 5.1: Relationale Datenbank (http://www.mysql.com/)
- PostgreSQL 9.0: Relationale Datenbank (http://www.postgresql.org/)
- MongoDB 1.8: NoSQL-DB (http://www.mongodb.org/)
- Redis 2.2 Key-Value Store (http://redis.io)
- RabbitMQ 2.4 Messaging (http://www.rabbitmq.com/)
- Unterstützung der folgenden Runtimes: Java 1.6, Node.js 0.4.5, Ruby 1.8 und 1.9
- Unterstützung für die folgenden Frameworks: Grails, Ruby on Rails 3, Sinatra, reine Java-EE WebApps (Servlet 2.5), Node.js, Lift, Spring

	Amazon EC2	Micro Cloud Foundry	Google App Engine
OS	Beliebig	Cent OS	Google-eigen
Programmier-Sprache	Beliebig	Gesamter Java-Stack	Teile des Java-Stacks, Python
Deployment- Komplexität	Große Unterschiede	Ein paar Klicks	Skripte und Shortcuts
Unterhalt/Support	Sehr große Community	Noch Beta	Mittelgroße Community
Plattform-Bindung	Lediglich Bindung an die Infrastruktur	Bindung an Infrastruktur, OS und Application Server, falls die Micro Cloud VM benutzt wird	Bindung an gesamte Plattform (die aber sehr nahe an Java ist)
Preismodelle	Verschiedene Preispläne	Noch kein kommerzielles Angebot	Relativ einfache Preispläne

Tabelle 1: Vergleich von Cloud Foundry und App Engine

Wie aus der Liste ersichtlich ist, unterstützt die VM auch Nicht-Java-Runtimes und -Frameworks wie Ruby und Node.js. Aktuell können Java-Anwendungen auf zwei Arten in die Micro Cloud Foundry installiert werden: Man kann entweder die „vmc" benutzen, die Kommandozeilenbasierte VM-Konsole, oder man geht den direkten Weg und installiert direkt aus Eclipse beziehungsweise aus der Spring Tool Suite (STS, siehe Abbildung 1).

Sowohl auf der Kommandozeile als auch in der STS können Einstellungen der Applikation vorgenommen werden, beispielsweise in Bezug auf die Anzahl der zu verwendenden Instanzen, die Speicher-Zuteilung und die Konfiguration abhängiger Services. Die Kommandozeilen-Version ist allerdings ausgereifter als die STS und bietet häufig mehr Optionen an. Als Ausgleich ermöglicht die STS komplexere Vorgänge auf Knopfdruck. Man erhält zum Beispiel den direkten Zugriff auf Dateien in den Instanzen über eine „Remote Systems View".

Das Konzept der Instanzen erlaubt Hochverfügbarkeit. Als Load Balancer wird Nginx den Tomcat-Servern vorgeschaltet. Alle Aspekte werden durch den Server transparent verwaltet. Eine Basis-Konfiguration ist standardmäßig vorhanden. Muss ein Entwickler tiefer in die Konfiguration eintauchen, kann er sich jederzeit über die Kommandozeile auf den Server einloggen und hat vollen Zugriff auf alle lokalen Ressourcen.

Cloud Foundry bietet natürlich auch einen Monitoring-Service, der die installierten Anwendungen überwacht. Durch die Kommandozeile werden vergangene Abstürze, die durch Cloud Foundry registriert wurden, ans Tageslicht gebracht. Natürlich lassen sich auch die Logs einsehen.

Mit Cloud Foundry lässt sich direkt nur ein limitiertes Set von Datenbanken verwenden. Für E-Mail gibt es keine direkte Unterstützung. Solche Services können aber mit Java-Mitteln einfach selber erstellt werden. Schließlich steht einer auf Cloud Foundry betriebenen Java-Anwendung der gesamte Java-Stack zur Verfügung und es lassen sich somit prinzipiell alle durch Java unterstützten Services verwenden. Zudem bietet die Micro Cloud Foundry von Haus aus Unterstützung für Ruby und Node.js. Diese beiden Frameworks stehen zur Service-Implementation natürlich ebenso zur Verfügung.

Abbildung 2: Die Micro Cloud Foundry Virtual Machine

Benutzung der Cloud Foundry

Das Prinzip der Micro Cloud Foundry erfordert während der Entwicklung von Anwendungen weder privaten noch öffentlichen Zugang zu einer Cloud-Infrastruktur. Stattdessen betreibt jeder Entwickler seine Cloud-Instanz als lokale VM auf seinem Entwicklungsrechner. Wenn die Anwendung für den produktiven Betrieb bereit ist, lässt sie sich prinzipiell in jeder Cloud installieren, die kompatibel zum Micro-Cloud-Foundry-API ist. Da das Cloud-Foundry-Angebot immer noch im Beta-Stadium ist, gibt es noch keinen kostenpflichtigen Support und damit einhergehend natürlich auch keine Betriebsgarantien.

Auch wenn die Cloud-Foundry-VM komplett vorinstalliert geliefert wird, kann es dennoch sein, dass einige weitere Einstellungen vorgenommen werden müssen. Beispiele sind Umgebungsvariablen oder zusätzliche Benutzer für den Zugriff auf externe Systeme (siehe Abbildung 2).

Integration in die Entwicklungsumgebung

SpringSource bietet von Haus aus in der IDE SpringSource Tool Suite (STS) integrierte Unterstützung für Cloud Foundry. Damit lassen sich Deployments in die Cloud oder auf die lokale VM per Drag & Drop ausführen. Mit der Ruby-basierten Kommandozeile ist dieselbe Aufgabe ebenfalls sehr einfach zu erledigen und besteht aus einem Einzeiler. Als weitere nützliche Werkzeuge für die Entwicklung sind die Cloud Tools und ein Maven-Plug-in für einfaches automatisiertes Deployment zu nennen.

Das Cloud Foundry API ist, wie erwähnt, öffentlich. Somit kann prinzipiell jedermann ein Toolset zur Verwaltung der Deployments schreiben. Ein Plug-in, das dieses API benutzt, wurde kürzlich für die IDE IntelliJ IDEA angekündigt.

Nachteile und Alternativen

Wie bei jedem Cloud-Angebot besteht der größte Nachteil der Cloud Foundry darin, dass keine volle Kontrolle über jedes Detail der Maschine und den Deployment-Prozess erlangt werden kann. Die Tatsache, dass das API öffentlich ist und die Cloud-Foundry-VM zu 100 Prozent mit Open-Source-Technologien gebaut wurde, schwächt diesen Nachteil ein wenig ab.

Die Cloud Foundry kann durchaus als produktionsreif bezeichnet werden. Diese Aussage bezieht sich hauptsächlich auf die Infrastruktur. Das Tooling und die API wurden in der Vergangenheit häufigen auch konzeptionellen Anpassungen unterworfen, was für den breiten Einsatz in Unternehmen ein Risiko darstellt. Die Cloud-Foundry-Konsole wird lediglich zur Konfiguration der Instanzen verwendet. Die Hauptanforderungen an ein Produktivsystem wie Ausfallsicherheit, Skalierung, Security und Backup werden bereits durch die Cloud-Infrastruktur abgedeckt. Wirklich vergleichbare Alternativen zur Cloud Foundry existieren zurzeit nicht. Am

nächsten kommt Amazon Elastic Beanstalk. Dieser Service bietet eine Ablaufumgebung für Java-Web-Applikationen, ist aber auch auf diese beschränkt. Ruby- oder Node.js-Anwendungen wie bei der Cloud Foundry lassen sich auf dieser Plattform nicht betreiben. Es gibt bei AWS Beanstalk auch keine Möglichkeit, sich für die Entwicklung ein VM-Image lokal zu installieren.

Zukünftige Entwicklung

Cloud Foundry weist noch ein gewisses Entwicklungspotential auf. Hier sind einige der Features, die in naher Zukunft auf dem Plan stehen:

- Die Möglichkeit, andere Clouds zu nutzen, wie beispielsweise vCloud oder vSphere von VMware
- Die Verfügbarkeit von professionellem Support und Betriebsgarantien

Cloud Foundry ist immer noch im Beta-Stadium und wurde in der Vergangenheit komplett überarbeitet. Damit hat das aktuelle Konzept der lokalen Entwicklungs-VM erst Einzug gehalten. Zuvor war die Cloud Foundry komplett von der Amazon-EC2-Infrastruktur abhängig und war auch nur dort lauffähig. Es ist zu hoffen, dass das aktuelle Konzept mehr Bestand hat und es in naher Zukunft nicht umgekrempelt wird.

Die Benutzung der Plattform ist aus Entwicklersicht immer noch gratis. Es wird aber in Zukunft kostenpflichtige Enterprise-Versionen geben. Es existiert bisher noch sehr wenig Literatur zum Thema „Cloud Foundry". Dies kann als mangelndes Interesse auf Entwicklerseite gedeutet werden.

Es ist jedoch davon auszugehen, dass das Fehlen von professionellem Support und Betriebsgarantien die Hauptursache für die noch dürftige Verbreitung ist.

Fazit

Es empfiehlt sich, Cloud Foundry für einfachere und strategisch nicht wichtige Anwendungen und Show-Cases bereits im produktiven Umfeld einzusetzen. Als Entwicklungs-Plattform bietet sie den speziellen Vorteil, dass sich die Entwickler nicht zu sehr mit dem Setup eines Entwicklungs- und Testservers beschäftigen und mit dem Template-Mechanismus alle denselben Setup verwenden. Cloud Foundry bewegt sich durchaus in eine erfolgversprechende Richtung. Das API bietet via Kommandozeile oder STS die wichtigsten Elemente für den durchschnittlichen Java-Entwickler und die Einarbeitungszeit bewegt sich in einem annehmbaren Rahmen. Da die Cloud Foundry im Gegensatz zu früheren Versionen die Wahl der Frameworks nicht mehr so stark einschränkt, öffnet sich der Anwendungsbereich.

Links

- Cloud Foundry Homepage: http://www.cloudfoundry.com
- Cloud Foundry Open-Source-Plattform: http://www.cloudfoundry.org
- Cloud Foundry Maven-Plug-in: https://github.com/cloudfoundry/vcap-java-client/tree/master/cloudfoundry-maven-plugin
- Classic-Version der Cloud Foundry, die für Instanzen der ersten Generation verwendet wird: https://classic.cloudfoundry.com

Julien Eperon
julien.eperon@trivadis.com

Julien Eperon schloss sein Studium 2006 an der ETH Lausanne mit einem Master in Informatik ab. Danach arbeitete er in einem amerikanischen Unternehmen als Consultant für Software-Entwicklung. Seit April 2010 ist er bei Trivadis in Lausanne in den Bereichen „Private Banking" und „Gesundheitswesen" tätig. Sein Fokus liegt auf Java-Entwicklung im Linux-Umfeld.

Windows Azure Service Bus: Kommunikationsdienst auch für Java

Holger Sirtl, Microsoft Deutschland GmbH

Microsoft stellt mit Windows Azure eine Cloud-Plattform bereit, die nicht nur von .NET-, sondern auch von Java-, PHP- oder Node.js-Entwicklern dazu genutzt werden kann, eigene Anwendungen in der Cloud zu betreiben und gezielt einzelne Cloud Services (etwa zur Datenspeicherung oder Benutzer-Authentifizierung) zu nutzen. Ein besonders interessanter Dienst in Windows Azure ist der Service Bus zur netzwerk-, unternehmens- und technologieübergreifenden Kommunikation zwischen verteilten Web-Services. Dieser Artikel stellt den Service Bus vor und zeigt, wie damit auch Java-basierte Services mit lokalen oder entfernten Software-Komponenten kommunizieren können.

Cloud-Computing-Dienste werden häufig entsprechend ihrer Abstraktionsebene in Infrastruktur-, Platform-, Software-as-a-Service und weitere kategorisiert. Gemäß dieser Einteilung lässt sich Windows Azure als Platform-as-a-Service-Angebot einordnen, das auch einige Elemente aus der Infrastruktur-Schicht enthält. Windows Azure ist zum einen eine Sammlung von Cloud-Services, die für die Entwicklung, Erstellung und den Betrieb eigener Cloud-basierter Anwendungen genutzt werden können und die in Microsofts weltweit verteilten Rechenzentren betrieben werden, wobei der Ausführungsort eines jeden genutzten Azure Service bestimmt werden kann. Zum anderen bietet Windows Azure auch eine Reihe von Ressourcen, die Entwicklern bei Implementierung, Test und Deployment dieser Anwendungen helfen.

Abbildung 1 zeigt eine Referenz-Architektur, in der sowohl die Services der Windows-Azure-Plattform als auch deren Interaktionsmöglichkeiten zu sehen sind. Die Plattformdienste lassen sich grob drei Schichten zuordnen: Datenschicht (Data Layer), Anwendungsschicht (Application Layer) und Integrationsschicht (Integration Layer). Clients, die auf entsprechende Azure-basierte Dienste zugreifen, lassen sich einer Client-Schicht (Client Layer) zuordnen.

Die Windows-Azure-Storage-Services auf der Datenschicht fassen alle Non-SQL-Persistenzdienste zusammen. Diese sind über „RESTful"-Schnittstellen zugänglich. Wer für den Zugriff nicht den Weg über diese Schnittstellen gehen möchte, erhält in den frei verfügbaren Software Development Kits (SDKs) für .Net, Java, PHP und Node.js entsprechende Klassenbibliotheken, in denen die „RESTful"-Aufrufe gekapselt sind. Der Caching-Service ist ein verteilter In-Memory-Cache, den Anwendungskomponenten zur temporären Zwischenspeicherung von Daten nutzen können. Mit SQL Azure besitzt Windows Azure auch ein echtes relationales Datenbanksystem (RDBMS) als

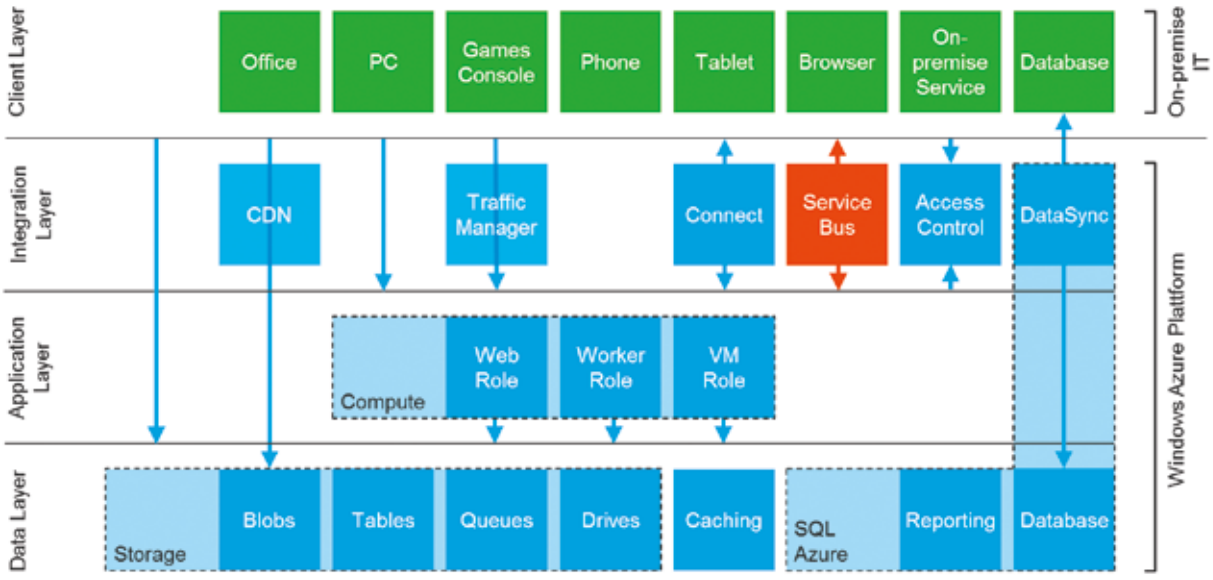


Abbildung 1: Windows-Azure-Referenz-Architektur

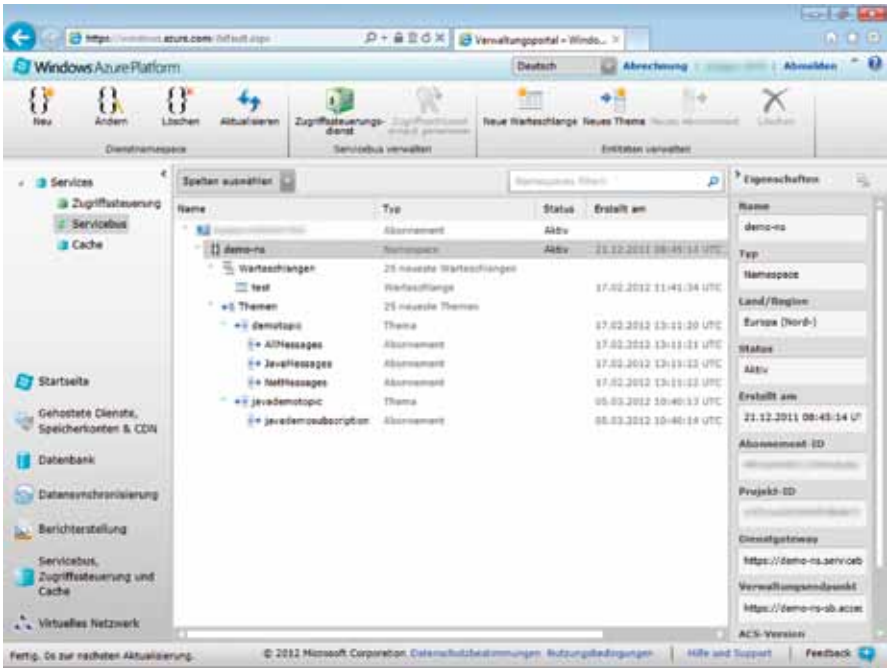


Abbildung 2: Screenshot der Service-Bus-Administrationsseite

```
package com.samples;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import com.microsoft.windowsazure.services.serviceBus.*;
import com.microsoft.windowsazure.services.serviceBus.models.*;
import com.microsoft.windowsazure.services.core.*;

public class ServiceBusQueueSender {
  public static void main(String[] args) {
    String issuer    = „[YOUR_ISSUER]“;
    String key       = „[YOUR_KEY]“;
    String namespace = „demo-ns“;
    String queueName = „testqueue“;

    Configuration config =
      ServiceBusConfiguration.configureWithWrapAuthentication(namespace, issuer, key);
    ServiceBusContract service = ServiceBusService.create(config);

    try
    {
      QueueInfo queueInfo = new QueueInfo(queueName);
      service.createQueue(queueInfo);
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      String messageText = br.readLine();
      while(messageText.length()>0)
      {
        BrokeredMessage message = new BrokeredMessage(messageText);
        service.sendQueueMessage(queueName, message);
        messageText = br.readLine();
      }
    } catch (Exception e) {
      System.out.printf(„Exception encountered: %s“, e.getMessage());
      System.exit(-1);
    }
  }
}
```

Listing 1: Client zum Versenden von Nachrichten an eine Service-Bus-Queue

Cloud Service, das ein Subset von SQL Server abbildet und entsprechend schnittstellenkompatibel ist. Somit kann von Java aus über den JDBC-Treiber für SQL Server auf SQL Azure zugegriffen werden.

Der Windows-Azure-Compute-Service auf der Anwendungsschicht ermöglicht die Ausführung eigener Anwendungen in der Cloud. Die Bereitstellung von virtuellen Maschinen (VMs) in der gewünschten Anzahl und Größe, deren Konfiguration, die Installation des Anwendungspakets, die Einstellung eines vorgeschalteten Load Balancers etc. werden vollautomatisch durch den Compute-Service gesteuert. Der Entwickler muss sich demnach nicht mit dem Setup der virtuellen Maschinen auseinandersetzen. Diese werden aus einem Pool bereits vorkonfigurierter VMs bezogen.

Für Java-Entwickler sind sogenannte „Worker Roles“ interessant. In deren VMs können eigene Webserver eingesetzt werden (beispielsweise Apache Webserver oder Tomcat). Bei VM Roles kann der Entwickler das VM Image selbst erstellen und in Azure betreiben. Auf alle VMs hat der Entwickler vollen Administrator-Zugang. Auch der Remote-Desktop-Zugang auf einzelne VM-Instanzen ist möglich.

Oft werden Cloud-basierte Services aus lokal betriebener Software heraus genutzt. Tatsächlich tragen Hybride-Szenarien, in denen Teile einer Anwendung lokal und andere in der Cloud ausgeführt werden, dem Wunsch Rechnung, die Vorteile der Cloud (hohe Skalierbarkeit, flexible Verfügbarkeit, nutzungsabhängige Kosten etc.) mit den Stärken der lokalen IT (individuelle Konfigurationen, Compliance-Anforderungen etc.) zu kombinieren. Auf der Integrationsschicht (siehe Abbildung 1) bietet Windows Azure deshalb eine Reihe von Cloud Services, die Funktionen für Interaktion und Integration von lokaler mit Cloud-IT bereitstellen. Soll von einem Azure Service heraus eine Ressource in der lokalen IT angesprochen werden, bietet Windows Azure Möglichkeiten auf mehreren Ebenen: Windows Azure Connect ermöglicht auf Netzwerk-Ebene den Aufbau einer „Ipsec“-gesicherten Verbindung zwischen VMs in Windows Azure und Rechnern in der lokalen IT. Auf Daten-Ebene synchronisiert der SQL-Azure-DataSync-Service die Daten zwischen lokalen und Azure-basierten SQL-Datenbanken.

Integration verteilter Anwendungskomponenten über den Service Bus

Auf Anwendungsebene bietet der Windows Azure Service Bus die Möglichkeit, lokale Services untereinander beziehungsweise auch mit auf Azure ausgeführten Services zu vernetzen. Die Service-Bus-Infrastruktur kann über entsprechende SDKs aus verschiedenen Technologien wie .NET, Java, PHP etc. heraus angesprochen werden. Bevor der Service Bus in eigenen Anwendungen genutzt werden kann, muss der Entwickler über das Windows-Azure-Portal einen Service-Namespace anlegen. Dieser dient als Scoping-Container für die Adressierung von Service-Bus-Ressourcen in eigenen Anwendungen. Die Administrationsseite zum Service Bus ist in Abbildung 2 zu sehen. Das Anklicken der Schaltfläche „Neu" öffnet ein Dialogfenster, über das ein neuer Namespace, also dessen Name, Ausführungsort der zugehörigen Ressourcen etc., konfiguriert werden kann.

Abbildung 2 zeigt auch einen Namespace mit der Bezeichnung „demo-ns". Alle Ressourcen, die diesem Namespace zugeordnet sind, werden in Nordeuropa ausgeführt. Jede Software-Komponente, die mit dem Service Bus interagieren möchte, muss sich mit einem Zugriffsschlüssel authentifizieren. Dieser besteht aus einer 44-stelligen Zeichenfolge und kann über das Portal ausgelesen werden.

Service-Bus-Queues

Die unter [1] verfügbaren Windows-Azure-Libraries für Java enthalten APIs zur Nutzung des Service Bus. Service-Bus-Queues können als Messaging Broker zur asynchronen Kommunikation zweier Software-Komponenten verwendet werden. Sender stellen Nachrichten in eine Queue ein, wo sie ein Empfänger für einen definierbaren Zeitraum auslesen kann. Abbildung 3 skizziert den Mechanismus.

Sendende Software-Komponenten können Nachrichten in eine Queue einstellen. Diese sind dann innerhalb ihrer Lebensdauer von einem Empfänger auslesbar. Listing 1 zeigt, wie eine einfache Konsolenanwendung zum Anlegen einer Queue und Versenden von Nachrichten aussehen kann.

Zunächst werden die erforderlichen Bibliotheken aus dem Windows-Azure-SDK importiert. Zu Beginn der „main"-Methode wird eine Konfiguration erzeugt. Hierzu werden der über das Portal zuvor angelegte Namespace sowie die über das Portal verfügbaren Werte für den Issuer und den Zugriffsschlüssel benötigt. Mithilfe dieser Konfiguration entsteht ein ServiceBusContract-Objekt. Über dieses werden dann die meisten weiteren Service-Bus-Operationen ausgeführt. Zunächst wird eine Queue angelegt, anschließend werden in einer Endlosschleife Texteingaben von der Konsole eingelesen und als Nachrichten in die Queue eingestellt.

Listing 2 zeigt die Implementierung eines einfachen Empfängers, der Nachrichten aus einer Service-Bus-Queue ausliest und deren Inhalte auf der Konsole anzeigt.

```
package com.samples;
import [...]
public class ServiceBusQueueReceiver {
 public static void main(String[] args) {
  String issuer    = „[YOUR_ISSUER]";
  String key       = „[YOUR_KEY]";
  String namespace = „demo-ns";
  String queueName = „testqueue";

  Configuration config = ServiceBusConfiguration
     .configureWithWrapAuthentication(namespace, issuer, key);
  ServiceBusContract service = ServiceBusService.create(config);

  try {
   ReceiveMessageOptions opts = ReceiveMessageOptions.DEFAULT;
   opts.setReceiveMode(ReceiveMode.RECEIVE_AND_DELETE);
   while (true) {
    ReceiveQueueMessageResult resultQM = service
       .receiveQueueMessage(queueName, opts);
    BrokeredMessage message = resultQM.getValue();
    if (message != null && message.getMessageId() != null) {
     InputStream is = message.getBody();
     InputStreamReader isr = new InputStreamReader(is);
     BufferedReader br = new BufferedReader(isr);

     System.out.println(„Body: „ + br.readLine());
    } else {
     Thread.sleep(1000);
    }
   }
  } catch (Exception e) {
   System.out.printf(„Exception encountered: %s", e.getMessage());
   System.exit(-1);
  }
 }
}
```

Listing 2: Empfänger zum Auslesen und Anzeigen von Nachrichten aus einer Service-Bus-Queue

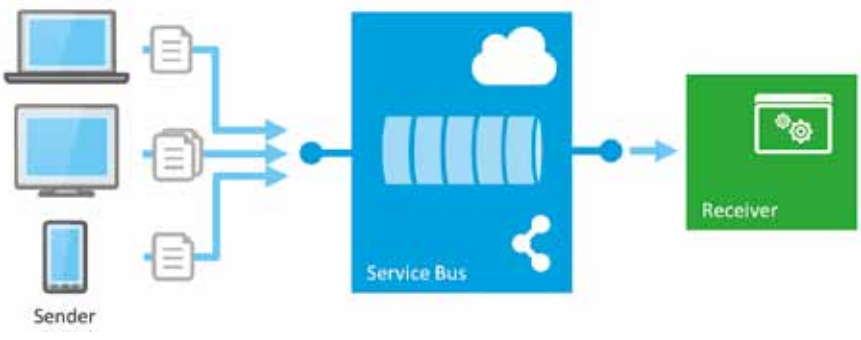


Abbildung 3: Service-Bus-Queue

```
package com.samples;
import [...]
public class ServiceBusTopicSender {
  public static void main(String[] args) {
    String issuer    = "[YOUR_ISSUER]";
    String key       = "[YOUR_KEY]";
    String namespace = "demo-ns";
    String topicName = "testtopic";

    Configuration config =
      ServiceBusConfiguration.configureWithWrapAuthentication(namespace, issuer, key);
    ServiceBusContract service = ServiceBusService.create(config);

    try
    {
      TopicInfo topicInfo = new TopicInfo(topicName);
      service.createTopic(topicInfo);
      SubscriptionInfo subAllInfo = new SubscriptionInfo("AllMessages");
      service.createSubscription(topicName, subAllInfo);
      SubscriptionInfo subErrInfo = new SubscriptionInfo("ErrorMessages");
      service.createSubscription(topicName, subErrInfo);
      RuleInfo ruleInfo = new RuleInfo();
      ruleInfo = ruleInfo.withSqlExpressionFilter("error = '1'");
      ruleInfo.setName("Errors");
      CreateRuleResult ruleResult = service.createRule(topicName, "ErrorMessages", ruleInfo);
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      String messageText = br.readLine();

      while(messageText.length()>0)
      {
        BrokeredMessage message = new BrokeredMessage(messageText);

        if((messageText.length()>4) && (messageText.startsWith("ERR:")))
        {
          message.setProperty("error", "1");
        }
        else
        {
          message.setProperty("error", "0");
        }
        service.sendQueueMessage(topicName, message);
        messageText = br.readLine();
      }

      service.deleteSubscription(topicName, "AllMessages");
      service.deleteSubscription(topicName, "ErrorMessages");
      service.deleteTopic(topicName);
    } catch (Exception e) {
      System.out.printf("Exception encountered: %s", e.getMessage());
      System.exit(-1);
    }
  }
}
```

Listing 3: Service Bus Topic Sender

Auch der Empfänger importiert die benötigten Bibliotheken, erstellt eine Service-Bus-Konfiguration und damit ein ServiceBusContract-Objekt, das dann Methoden zum Auslesen von Nachrichten aus Queues bereitstellt. Für das Auslesen gibt es zwei Möglichkeiten: ReceiveAndDelete und PeekLock. Bei ReceiveAndDelete wird eine ausgelesene Nachricht sofort aus der Queue entfernt. Bei PeekLock erhält der Empfänger eine Kopie der Nachricht. Das Original der Nachricht verbleibt in der Queue, wird allerdings für einen vom Anwender festgelegten Zeitraum für Zugriffe weiterer Empfänger gesperrt. Hat der Empfänger die Nachricht erfolgreich verarbeitet, kann er sie aus der Queue löschen. Erfolgt dies nicht innerhalb des Sperrzeitraums, wird die Nachricht wieder frei gegeben, sodass sie erneut ausgelesen (bei Bedarf von einem anderen Empfänger) und verarbeitet werden kann. Im Beispiel in Listing 2 kommt ReceiveAndDelete zum Einsatz.

Service Bus Topics und Subscriptions

Queues eignen sich in Szenarien, in denen einzelne Nachrichten in der Regel nur von einem Empfänger ausgelesen und verarbeitet werden sollen. Sollen Nachrichten an mehrere Empfänger verteilt werden, können Topics und Subscriptions verwendet werden. Dabei schicken Sender Nachrichten an ein Topic. Für ein solches Topic können für einen oder mehrere Empfänger Subscriptions angelegt werden, hinter denen Queues arbeiten. Empfänger lesen Nachrichten aus den ihnen zugeordneten Subscription-Queues aus. Für jede Subscription können Filterregeln definiert werden, die auf Basis von Meta-Informationen der Nachrichten bestimmen, in welche Subscription-Queues eine Nachricht, die an das zugehörige Topic geschickt wurde, eingestellt wird. Der Mechanismus und die beteiligten Komponenten sind in Abbildung 4 skizziert. In Listing 3 ist die Implementierung eines Publishers zu sehen, der Nachrichten an ein Topic sendet.

Auch hier werden zunächst erforderliche Bibliotheken importiert und mithilfe einer Konfiguration ein ServiceBusContract-Objekt erzeugt. Mit diesem werden zunächst ein Topic und dann darauf zwei Subscriptions angelegt. Die erste erhält alle Nachrichten, die an das Topic geschickt werden. Für die zweite wird über eine Regel definiert, dass nur solche Nachrichten an die Subscription gehen, die über Meta-Informationen vom Absender als Fehler-Nachricht gekennzeichnet wurden. In einer Endlosschleife werden dann Eingabetexte über die Konsole gelesen und als Nachrichten an das Topic geschickt. Beginnt ein Eingabetext mit dem String „ERR:", wird die Nachricht als Fehlernachricht markiert. Listing 4 zeigt einen entsprechenden Empfänger, der Nachrichten aus den im Publisher angelegten Subscriptions auslesen kann.

Auch hier wird wieder mit den entsprechenden Schritten ein ServiceBusCon-

tract-Objekt erzeugt. Grundsätzlich liest dieser Empfänger Nachrichten aus der allgemeinen Subscription, es sei denn, es wurde per Übergabeparameter bestimmt, dass er Nachrichten aus der Fehlernachrichten-Subscription lesen soll. Wie im Queue-Empfänger in Listing 2 wird auch hier ReceiveAndDelete als Mechanismus zum Lesen der Nachrichten verwendet.

Fazit

Windows Azure stellt eine Vielzahl von Cloud Services zur Verfügung, die in eigenen Anwendungen genutzt und über Test-Accounts kostenlos evaluiert werden können (siehe [1]). Über den Windows Azure Service Bus können verteilte Software-Komponenten Nachrichten austauschen. Der Service Bus stellt hier eine leistungsfähige, hochskalierbare Messaging-Plattform zur Verfügung, die über Java-APIs recht einfach genutzt werden kann. Wenngleich für den Service Bus noch keine Implementierung des JMS-API zur Verfügung steht, ist die Nutzung in eigenen (auch lokal betriebenen) Anwendungen leicht implementierbar.

Weitere Informationen

[1] Windows Azure Libraries für Java: http://go.microsoft.com/fwlink/?LinkID=236226&clcid=0x409

[2] Kostenloser Test-Account für Windows Azure: http://www.windowsazure.com/de-de/pricing/free-trial/

[3] Holger Sirtls Blog zur Windows Azure: http://blogs.msdn.com/hsirtl

Holger Sirtl
holger.sirtl@microsoft.com

Holger Sirtl ist seit 2006 als Architecture Evangelist bei Microsoft in München tätig und berät in dieser Rolle Unternehmen im Aufbau Cloud-basierter Anwendungsarchitekturen. Schwerpunktthemen seiner Arbeit sind Cloud Computing und die Windows Azure Plattform. Vor seinem Einstieg bei Microsoft arbeitete Holger Sirtl sechs Jahre lang als Technologieberater für eine international führende Unternehmensberatung sowie zwei Jahre lang als Senior-IT-Projektmanager für einen großen deutschen Energieversorger.


```
package com.samples;
import [...]
public class ServiceBusSubscriptionReceiver {
  public static void main(String[] args) {
    String issuer     = „[YOUR_ISSUER]“;
    String key        = „[YOUR_KEY]“;
    String namespace  = „demo-ns“;
    String topicName  = „testtopic“;
    String subscrName = „AllMessages“;

    if((args.length>0) && (args[0].equals(„ERR“)))
    {
      subscrName = „ErrorMessages“;
    }

    Configuration config = ServiceBusConfiguration
        .configureWithWrapAuthentication(namespace, issuer, key);
    ServiceBusContract service = ServiceBusService.create(config);

    try {
      ReceiveMessageOptions opts = ReceiveMessageOptions.DEFAULT;
      opts.setReceiveMode(ReceiveMode.RECEIVE_AND_DELETE);

      while (true) {
        ReceiveSubscriptionMessageResult result =
            service.receiveSubscriptionMessage(topicName, subscrName);
        BrokeredMessage message = result.getValue();
        if (message != null && message.getMessageId() != null) {
          InputStream is = message.getBody();
          InputStreamReader isr = new InputStreamReader(is);
          BufferedReader br = new BufferedReader(isr);

          System.out.println(br.readLine());
        } else {
          Thread.sleep(1000);
        }
      }

    } catch (Exception e) {
      System.out.printf(„Exception encountered: %s“, e.getMessage());
      System.exit(-1);
    }
  }
}
```

Listing 4: Service Bus Subscription Receiver

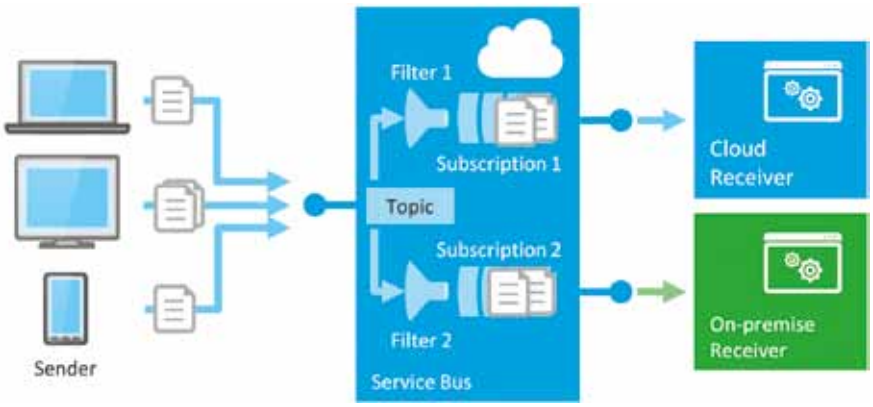


Abbildung 4: Service Bus Topic und Subscriptions

Jnect: Kinect goes Java

Jonas Helming und Maximilian Kögel, EclipseSource München GmbH

Das Jnect-Framework stellt Interoperabilität zwischen dem Microsoft Kinect SDK und Java her. Jnect ist als Eclipse-Plug-in realisiert und erlaubt damit auch die Steuerung von Eclipse-basierten Applikationen wie beispielsweise der Java IDE Eclipse selbst.

Kinect von Microsoft unterstützt die Erkennung von Bewegungen, Gesten und sogar Sprache. Es wurde ursprünglich für die Microsoft Xbox entwickelt, dient zur Steuerung von Spielen oder sogar als Ersatz für den persönlichen Fitnesstrainer. Microsoft veröffentlichte jedoch auch ein SDK für Windows 7, mit dem es möglich ist, unabhängig von der Xbox eigene Anwendungen für Kinect zu programmieren. Dazu wird bisher C++ oder C# verwendet.

Wie häufig bei der Einführung neuer Technologien ist der Spiele-Markt im Falle von Bewegungs- und Gesten-Erkennung ein wesentlicher Innovationstreiber. Die Nintendo Wii wurde bekannt durch das neue Steuerungskonzept, bei dem man einen mit einem Bewegungs- und Infrarot-Sensoren ausgestatteten Controller frei im Raum bewegt, um Spiele zu steuern. Microsoft ging mit Kinect für die Xbox noch einen Schritt weiter und verzichtete gänzlich auf einen Controller. Stattdessen erkennt die 3D-Kamera des Kinect direkt den menschlichen Körper und kann dessen Bewegungen zur Steuerung von Spielen interpretieren. Der Mensch wird damit selbst zum „Controller". Springt man beispielsweise in die Luft, wird auch die Spielfigur einen Sprung ausführen. In diesem Zusammenhang ist vor allem auch die geringe Latenz zwischen realer und virtueller Bewegung von nur wenigen Millisekunden faszinierend.

Geräte aus der Spiele-Industrie haben neben ihrer Innovation meist noch einen weiteren Vorteil: Sie sind für den Massenmarkt konzipiert und damit günstig. Ein Gerät wie Kinect, das in dieser Art als Speziallösung mehrere Tausend Euro kosten würde, ist plötzlich für etwa 100 Euro in jedem Elektronikmarkt zu erwerben. Und als USB-Device ist Kinect prinzipiell mit allem interoperabel, was USB kennt.

Es dauerte nicht lange, bis Kinect von Tüftlern und Technikbegeisterten für ganz andere Zwecke als das Spielen mit der Xbox verwendet wurde. So verleiht Kinect Robotern die Fähigkeit zu sehen oder dient als 3D-Scanner zum Digitalisieren von räumlichen Objekten. Microsoft reagierte schnell auf diesen Trend und erleichterte mit der Veröffentlichung des Kinect-SDK die Interoperabilität mit Kinect. Mit diesem SDK kann über ein API direkt auf die Funktionen des Kinect zugegriffen werden. So lassen sich beispielsweise Desktop-Applikationen über von Kinect erkannte Bewegungen steuern – das Ganze allerdings bisher nicht in Java.

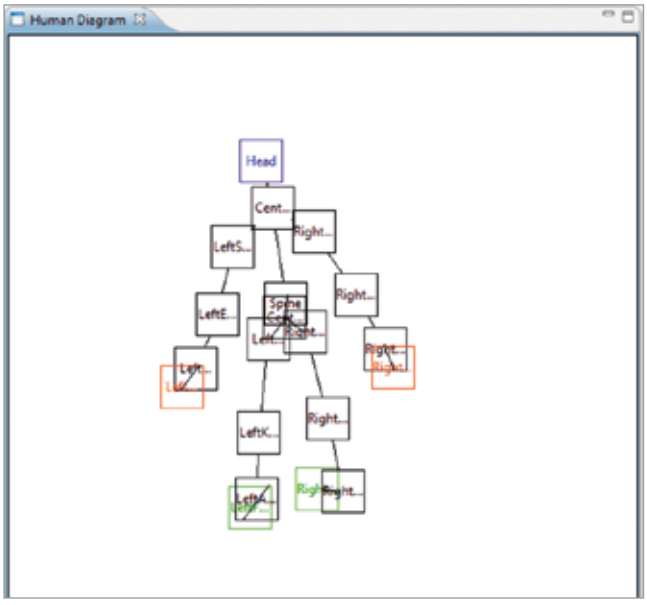


Abbildung 1: Die Positions-Informationen des Körpers können beispielsweise auf einem Diagramm visualisiert werden

Das Jnect-Projekt schließt diese Lücke und stellt einen Java-Wrapper für das Microsoft Kinect SDK in Form eines Eclipse-Plug-ins zur Verfügung. Damit lassen sich die von Kinect erkannten Daten direkt in der eigenen Eclipse-Anwendung verarbeiten und nutzen. Die erste Version von Jnect wurde im Rahmen eines Universitätsprojekts als Open Source entwickelt. Dieser Artikel zeigt, wie man mit Jnect die Körpererkennung (Body Tracking) und die Spracherkennung des Kinect für die Steuerung eigener Anwendungen nutzen kann. Alle beschriebenen Beispiele sowie das Framework selbst finden Sie unter [1].

Mit dem Body Tracking ist es sehr einfach möglich, Teile der Anwendung durch Gesten zu steuern. Unter [1] finden sich Demonstrationen, in denen beispielsweise der Debugger von Eclipse über Gesten kontrolliert wird. Um einen „Step Over" auszuführen, drückt man dabei nicht mehr irgendeine Taste, sondern hebt ganz einfach seine rechte Hand. Derartige Formen der Steuerung sind insbesondere in Umgebungen interessant, in denen der Benutzer nicht oder nur schwierig auf eine Tastatur zugreifen kann. Auch visuelle Darstellungsformen wie beispielsweise Diagramme können gut durch das Body Tracking gesteuert werden. Als Demonstra-

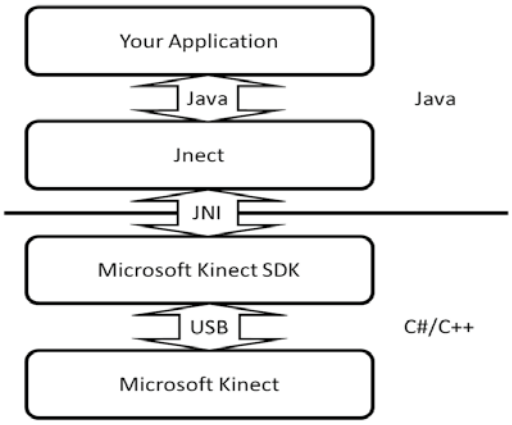


Abbildung 2: Das Jnect-Framework stellt über JNI eine Verbindung zum Microsoft Kinect SDK her

tion zeigt Abbildung 1, wie sämtliche Teile des Körpers in der aktuellen Position auf einem Diagramm visualisiert werden. Auf diese Weise lassen sich per Handbewegung beispielsweise UML-Diagramme modifizieren oder auf Scrum Task Boards Aufgaben verschieben. Es lassen sich jedoch nicht nur existierende Anwendungen steuern, Kinect erlaubt es auch, neue Anwendungsfelder zu erschließen. So könnte beispielsweise überwacht werden, ob sich eine Person im beobachteten Bereich noch in aufrechter Form befindet, oder, falls sie gestürzt sein sollte, ein Alarm ausgelöst werden.

Ähnlich umfangreich sind potenzielle Anwendungsfelder der Sprachsteuerung. In einer Demonstration unter [1] lassen sich der Eclipse Debugger und einige weitere Kommandos der Eclipse IDE per Sprache steuern. Diese IDE ist auch ein gutes Beispiel für eine Anwendung mit vielen für sich genommen einfachen Kommandos, deren Menge jedoch die mögliche Anzahl an Tastenkombinationen teilweise überreizt. So ist es zum einen schwierig, eine noch freie Tastenkombination für einen Befehl zu finden, zum anderen ist es fast unmöglich, sich alle verfügbaren Kommandos zu merken.

Hinter den Kulissen

Das Microsoft SDK spricht Kinect über USB an und verarbeitet die Daten vor. So liefert es fertig berechnete Körperpositionen, beispielsweise wo genau sich der Kopf eines von Kinect beobachteten Menschen befindet. Ziel von Jnect ist es, die Microsoft Kinect in Java, genauer gesagt in einer Eclipse-Anwendung, nutzbar zu machen.

Gleichzeitig soll die umfangreiche Funktionalität des Microsoft Kinect SDK verfügbar bleiben. Zu diesem Zweck greift Jnect per JNI direkt auf das Microsoft SDK zu. Damit kann die bereits existierende Funktionalität des SDK direkt wiederverwendet werden, das Jnect-Framework stellt lediglich die Interoperabilität mit Java her.

Für das Microsoft SDK wird zur Laufzeit Windows als Plattform benötigt. Durch die Verwendung von Jnect kann nun eine Applikation, die auf die Daten des Kinect zugreifen möchte, direkt in Java geschrieben werden. Jnect bildet dabei zwar nicht das komplette Microsoft Kinect SDK ab, stellt aber mit Body Tracking und Speech Recognition wohl die beiden wichtigsten Features bereit, für die in den folgenden Abschnitten Beispiele beschrieben werden.

Body Tracking

Body Tracking liefert genaue Daten darüber, wo sich die Körperteile eines von Kinect beobachteten Menschen befinden. Jnect liefert diese Positionen in Form von Java-Objekten, eines für jeden Teil des Körpers. Diese Java-Objekte stellen damit ein Modell des Körpers dar. Sie sind in Jnect unter Verwendung des Eclipse Modeling Frameworks (EMF) realisiert. Für genauere Informationen zu EMF siehe [2].

Von jedem Teil des Modells kann nun sowohl die X- als auch die Y-Position abgefragt werden. Alle Teile des Körpers sind in einem Container-Objekt (SkeletonModel) enthalten, das wiederum von einem KinectManager abgefragt werden kann. Das Codebeispiel in Listing 1 fragt die X-Position der linken Hand ab. Dazu wird zunächst die Verbindung zu Kinect aufgebaut sowie das Body Tracking gestartet.

In den meisten Anwendungsszenarios ist jedoch mehr als die aktuelle Position die Veränderung einer Position, also eine Bewegung des Körpers, interessant. Um über Änderungen am Modell notifiziert zu werden, kann man in EMF-Listener Teile des Körpers registrieren. Um auf eine Änderung zu reagieren, kann dann innerhalb des Listeners auf das Körperteil, dessen Position sich verändert hat, aber auch auf die Position jedes anderen Teiles zugegriffen werden. Das Codebeispiel in Listing 2 registriert einen Listener auf Änderungen der linken Hand und fragt im Falle einer Änderung die X-Position ab.

```
KinectManager kinectManager = KinectManager.INSTANCE;
kinectManager.startKinect();
kinectManager.startSkeletonTracking();
float x = kinectManager.getSkeletonModel().getLeftHand().getX();
```

Listing 1

```
kinectManager.getSkeletonModel().getLeftHand().eAdapters().add(new AdapterImpl(){
        @Override
        public void notifyChanged(Notification msg) {
                kinectManager.getSkeletonModel().getLeftHand().getX();
}

});
```

Listing 2

```
final KinectManager kinectManager = KinectManager.INSTANCE;
kinectManager.startKinect();
GestureProxy.INSTANCE.addGestureDetector(new Gesture() {
        @Override
        protected boolean isGestureDetected(Notification notification) {
                float head = kinectManager.getSkeletonModel().getHead().getY();
                float rHand = kinectManager.getSkeletonModel().getRightHand().getY();
                return y<y2;
                }
});
GestureProxy.INSTANCE.addGestureListener(new GestureListener() {
        @Override
        public void notifyGestureDetected(Class<? extends Gesture> gesture) {
                //Do something
                GestureProxy.INSTANCE.removeGestureListener(this);
        }
});
kinectManager.startSkeletonTracking();
```

Listing 3

```
final KinectManager kinectManager = KinectManager.INSTANCE;
kinectManager.startKinect();
kinectManager.addSpeechListener(new SpeechListener() {
        @Override
        public void notifySpeech(String speech) {
                if (speech.equalsIgnoreCase(YOURWORD)) {
                        //Do something
                        kinectManager.stopSpeechRecognition();
                }
        }
        @Override
        public Set<String> getWords() {
                return Collections.singleton(YOURWORD);
        }
});
kinectManager.startSpeechRecognition();
```

Listing 4

Um das Erkennen von Gesten mit nur einem Zustand zu erleichtern, bietet Jnect zwei Interfaces an. GestureDetectors entscheiden, ob im aktuellen Zustand eine Geste erkannt ist. Sie werden bei jeder Änderung von Positionen beliebiger Köperteile befragt. GestureListener werden aufgerufen, wenn eine registrierte Geste erkannt wurde, und können daraufhin beliebige Aktionen ausführen.

Das Codebeispiel in Listing 3 registriert einen GestureListener, der überprüft, ob die rechte Hand über den Kopf gehoben wird. Im GestureDetector wird anschließend der GestureListener zunächst entfernt, da sonst die Geste dauerhaft erkannt werden würde. In einem echten Szenario bräuchte man also eine zweite Geste, die der Listener wieder registriert, sobald die Hand wieder unter den Kopf bewegt wird.

Spracherkennung

Das Spracherkennungs-Feature des Microsoft Kinect SDK erlaubt es, gesprochene Wörter aus einem vorher definierten Vokabular aus Phrasen zu erkennen. Dazu werden die Raum-Mikrofone von Kinect genutzt. Die Spracherkennung funktioniert zwar nicht mit beliebigen Wörtern, da diese vorher festgelegt werden müssen, dafür erreicht sie umgekehrt aufgrund des beschränkten Vokabulars sehr hohe Erkennungsraten. Einsatzszenario ist also nicht das Diktieren von Texten, sondern die Auswahl von vorher bekannten Kommandos.

Jnect bildet dieses Feature über das Interface SpeechListener ab. In diesem werden zwei Methoden implementiert. In der Methode „getWords()“ gibt der SpeechListener an, auf welche Phrasen er reagieren möchte, beispielsweise „Do something“. Die Phrasen werden dann von Jnect in das Vokabular der Kinect-Spracherkennung eingefügt. Wird eine Phrase erkannt, werden die Methode „notifySpeech()“ aufgerufen und die erkannten Phrasen als Parameter übergeben. Das Codebeispiel in Listing 4 registriert einen SpeechListener mit einer Phrase und reagiert auf diese.

Spannend ist auch die Kombination von Sprach- und Gestenerkennung. Damit lassen sich Kommandos wie beispielsweise das Vergrößern eines Fensters bequem starten und beenden: Der Benutzer streckt die Hände aus und aktiviert den Resize-Modus mit einem Sprachkommando. Durch das Verkleinern und Vergrößern des Abstands der Hände wird das Fenster entsprechend vergrößert und verkleinert. Durch das Stopp-Sprachkommando wird die Fenstergröße wieder eingefroren.

Fazit

Jnect stellt die Interoperabilität zwischen dem Microsoft Kinect SDK und Java her. Zu diesem Zweck verbindet es sich via JNI zum SDK und bietet für die Features Body Tracking und Speech Recognition ein Java-API in Form eines Eclipse-Plug-ins an. Auf Basis dieses API kann eine Java-Anwendung an Kinect angebunden werden.

Jnect ist ein Open-Source-Projekt, eine erste Beta-Version wurde als Eclipse-Labs-Projekt veröffentlicht. Geplante nächste Schritte sind die Anbindung des neuen Kinect für den PC sowie die Unterstützung weiterer Features des Microsoft Kinect SDK. In diesem jungen Projekt sind die Entwickler von Jnect über jede Art von Feedback dankbar.

Links

[1] jnect.org
[2] eclipsesource.com/emftutorial

Jonas Helming
jhelming@eclipsesource.com
Maximilian Kögel
mkoegel@ eclipsesource.com

Jonas Helming und Maximilian Kögel sind Eclipse-EMF/RCP-Trainer und Consultants sowie Geschäftsführer der EclipseSource München GmbH. Sie leiten die Eclipse-Projekte „EMFStore“ und „EMF Client Platform“.

Unbekannte Kostbarkeiten des SDK

Heute: Double Brace Initialization und Instance Initializer

Bernd Müller, Ostfalia

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

```
String[] ziffern =
    new String[] { „eins“, „zwei“, „drei“, „vier“, ...};
```

Listing 1

```
List<String> ziffern = new ArrayList<>();
ziffern.add(“eins”);
ziffern.add(“zwei”);
ziffern.add(“drei”);
ziffern.add(“vier”);
...
```

Listing 2

```
List<String> ziffern =
    Arrays.asList(new String[] { „eins“, „zwei“,
    „drei“, „vier“, ... });
```

Listing 3

```
List<String> ziffern = new LinkedList<String>() {{
    add(„eins“);
    add(„zwei“);
    add(„drei“);
    add(„vier“);
    ...
}};
```

Listing 4

```
Map<String, String> ziffern = new HashMap<String,
String>() {{
    put(“1”,“eins”);
    put(“2”,“zwei”);
    put(“3”,“drei”);
    put(“4”,“vier”);
    ...
}};
```

Listing 5

Wir haben in dieser Reihe bereits den Service-Loader, das Dynamic Proxy und die VisualVM vorgestellt. Während die beiden erstgenannten Kostbarkeiten über Klassen beziehungsweise Interfaces des SDK realisiert werden, ist die VisualVM eine separate und eigenständige Java-Anwendung des SDK. Es gibt jedoch auch unbekannte Kostbarkeiten, die eine Ebene tiefer angesiedelt sind: in der Sprache Java selbst. Die Double Brace Initialization erlaubt die elegante und einfache Initialisierung von Variablen komplexerer Datenstrukturen, etwa Collections, und basiert auf dem Instance Initializer Block. Wir motivieren zunächst die Verwendung der Double Brace Initialization und gehen dann, wenn wir deren innere Funktionsweise analysieren, auf Instance Initializer ein.

Double Brace Initialization

Die Initialisierung von Arrays kann mit dem sogenannten „Array Initializer“ relativ einfach realisiert werden (siehe Listing 1). Anders sieht es etwa mit Klassen des Collection-Frameworks aus. Bei Listen, Mengen oder Maps müssen die einzelnen Listenelemente durch explizite Methodenaufrufe hinzugefügt werden (siehe Listing 2).

Ein wenig Erleichterung für Listen bringt die „asList()“-Methode der Klasse „Arrays“ (siehe Listing 3). Mit der „Double Brace Initialization“ kann dies wesentlich eleganter realisiert werden (siehe Listing 4). Dies funktioniert auch mit „Maps“ (siehe Listing 5).

Die Hintergründe, Teil 1: Anonyme innere Klassen

Java 1.1 führte innere Klassen ein. Eine lokale innere Klasse kann innerhalb einer Methode definiert werden. Benötigt man nur eine einzige Instanz dieser Klasse, kann auf eine Namensvergabe verzichtet werden, und Definition und Instanziierung können zu einem einzigen Konstrukt, einer sogenannten „anonymen inneren Klasse“, zusammengefasst werden. Anonyme innere Klassen können ein Interface implementieren oder von einer Oberklasse erben. Listing 6 zeigt ein Beispiel, das häufig verwendet wird, um einem „JButton“ einen „ActionListener“ hinzuzufügen.

Hier ist anzumerken, dass die anonyme innere Klasse das Interface „ActionListener“ implementiert und damit die einzige Methode dieses Interface definieren muss. Bei der zweiten Verwendungsart wird von einer Oberklasse abgeleitet, wie im Listing 7 gezeigt.

Hier erbt die anonyme innere Klasse von der Klasse „Thread“ und überschreibt die geerbte Methode „run()“. Wir wiederholen nun den Code unseres einführenden Beispiels mit dem doppelten Klammerpaar, löschen jedoch das innere Klammerpaar samt Inhalt (siehe Listing 8).

Was wir hier sehen, ist eine anonyme innere Klasse, die von „LinkedList“ erbt, aber keine der geerbten Methoden überschreibt und auch keine Methode zusätzlich definiert. Wir sind der Syntax der Double Brace Initialization also auf der Spur.

Die Hintergründe, Teil 2: Instance Initializer

Ebenfalls mit Java 1.1 wurde der Instance Initializer eingeführt, dessen Definition in der Java Language Specification im Abschnitt 8.6 nachgelesen werden kann. Ein Instance Initializer ist ein einfacher Block, der beliebige Anweisungen enthalten kann und bei der Erzeugung eines Objekts nach den Initialisierungen der Variablendefinition und vor dem Konstruktor ausgeführt wird (siehe Listing 9).

Beim Erzeugen einer Instanz über den Konstruktor wird zunächst die Variable „a" mit 5 initialisiert. Die beiden Instance Initializer werden in der Reihenfolge ihrer Definition ausgeführt. Die Variable „a" bekommt also zunächst den Wert 23, dann den Wert 77 zugewiesen. Zuletzt werden die Anweisungen des Konstruktor-Rumpfs ausgeführt, hier also der Variablen „a" der Wert 117 zugewiesen. Da Instance Initializer Blöcke sind, können verschiedene Anweisungen, nicht nur Werte-Zuweisungen verwendet werden. Insbesondere sind auch mehrere Anweisungen erlaubt. Man kann sich mit „System.out.println()"-Aufrufen davon überzeugen, dass die Variable „a" tatsächlich nacheinander die entsprechenden Werte zugewiesen bekommt. Wenn wir nun unser einführendes Beispiel nochmals betrachten, wird die Semantik schnell klar (siehe Listing 10).

Hier wird eine anonyme innere Klasse als Unterklasse von LinkedList erzeugt und ein Instance Initializer definiert, der aus den „add()"-Methodenaufrufen besteht. Da „LinkedList" serialisierbar ist, warnt der Compiler bei unserer anonymen inneren Klasse vor einer fehlenden Versionsnummer der Serialisierung. Wir definieren diese im folgenden Beispiel, um den Instance Initializer noch einmal hervorzuheben (siehe Listing 11). Zusätzlich demonstrieren wir eine Variablendefinition im Instance Initializer, die im Beispiel zwar relativ sinnlos ist, aber die allgemeine Verwendbarkeit des Instance Initializer beispielhaft darstellt.

```
Map<String, String> ziffern =
    new HashMap<String, String>() {
        private static final long serialVersionU-
ID = 1L;
        {
            String fuenf = „5";
            put(„1", „eins");
            put(„2", „zwei");
            put("3","drei");
            put("4","vier");
            put(fuenf,"fünf");
        }};
```

Listing 11

Die Definition neuer Methoden im äußeren Block ist ebenfalls möglich, aber wenig sinnvoll: Da die Klasse anonym ist, kann die Variable „ziffern" nicht auf die Klasse gecastet werden, was die Voraussetzung für die Verwendung der Methode ist, wenn man einmal von Reflection absieht.

Fazit

Wir haben hier gezeigt, wie mit der Double Brace Initialization komplexe Variablen einfach initialisiert werden können. Während die Double Brace Initialization ein Java-Idiom darstellt, das bei einer Google-Suche zu 178.000 Treffern (April 2012) führt, ist die Grundlage des Idioms, der Instance Initializer, ein Bestandteil der Sprache Java und wird in der Sprachbeschreibung explizit im Abschnitt 8.6 definiert. Sowohl die Double Brace Initialization als auch der Instance Initializer gehören zu den unbekannten Kostbarkeiten des SDK, hier sogar zu den unbekannten Kostbarkeiten der Sprache Java.

Bernd Müller
bernd.mueller@ostfalia.de

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

```
JButton button = new JButton(...);
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        ... // some code
    }
});
```

Listing 6

```
Runtime.getRuntime().addShutdownHook(new Thread() {
    public void run() {
        ... // some code
    }
});
```

Listing 7

```
List<String> ziffern = new LinkedList<String>() {
};
```

Listing 8

```
public class InstanceInitializerTest {
    private int a = 5;
    {
        a = 23;
    }
    {
        a = 77;
    }
    public InstanceInitializerTest() {
        a = 117;
    }
    ...
}
```

Listing 9

```
List<String> ziffern = new LinkedList<String>() {{
    add(„eins");
    add(„zwei");
    add(„drei");
    add(„vier");
    ...
}};
```

Listing 10

Oracle gegen Google: Interne E-Mails

Beim Prozessauftakt am 16. April 2012 zwischen Oracle und Google in San Francisco hat der Datenbankriese seine Position bekräftigt und in seiner Eröffnungserklärung Google vorgeworfen, bei der Entwicklung von dem Betriebssystem Android sich bewusst fremder Technologie bedient zu haben, um im Smartphone-Markt nicht hinter der Konkurrenz zurückzufallen. Der Anwalt von Oracle, Michael Jacobs, belegte die Aussage mit Zitaten aus internen E-Mails von Google. Im Mittelpunkt stand eine E-Mail des Entwicklers Tim Lindholm, in der der ehemalige Sun-Angestellte sich dafür ausgesprochen hatte, mit Sun eine Java-Lizenz auszuhandeln. Oracle beteuerte, es sei dem Suchmaschinenbetreiber durchaus bekannt gewesen, dass er eine Lizenz hätte erwerben müssen.

Android: von Maps und Libraries

Andreas Flügge, Object Systems GmbH

Es ist relativ einfach, Geo-Koordinaten aus einem Android-Device auszulesen und in einer View textuell darzustellen. Richtig interessant sind Geo-Koordinaten aber erst, wenn man sie in einer Kartendarstellung verwenden kann. Der Artikel zeigt, wie die aktuelle Position auf einer Google-Maps-Karte verfolgt werden kann.

Für die komfortable Darstellung geografischer Informationen liefert Google für Android eine zusätzliche Library aus, die nicht Bestandteil des SDK ist (siehe Abbildung 1). Sie kann über den Android SDK Manager aber jederzeit nachinstalliert werden. Bereits beim Anlegen eines Projekts wird im Wizard angegeben, dass das Google-Maps-API verwendet werden soll. Dieses ist frei verfügbar, jedoch sollte man sich intensiv mit den Nutzungsbedingungen vertraut machen, sofern man eine Verwendung in kommerziellen Applikationen in Erwägung zieht. Für die folgenden Ausführungen wird als Basis die Applikation so verwendet, wie sie zum Ende des Artikels der vorigen Ausgabe vorlag.

API-Key

Das Google-Maps-API enthält Funktionen, die eine Verbindung mit Google-Diensten über das Internet herstellen und mit diesen Daten austauschen. Damit dies reibungslos funktioniert, benötigt man einen sogenannten „API-Key", den Google kostenlos zur Verfügung stellt [2].

Für die Erstellung eines solchen Schlüssels ist der MD5-Fingerprint des Zertifikats erforderlich, mit dem eine Android-Applikation signiert wird. Jede Anwendung, die mit dem gleichen Zertifikat signiert ist, kann den gleichen API-Key verwenden. Für unsere Zwecke (während der Entwicklung) reicht temporär das Debug-Zertifikat. Es befindet sich im Debug Keystore unter „../android/debug.keystore" im Installationsverzeichnis des SDK. Listing 1 zeigt, wie man den Fingerprint mit dem Keytool von der Kommandozeile ausgeben kann.

Der Fingerprint wird nun zur Erzeugung des API-Keys verwendet. Google stellt dazu eine eigene Internetseite bereit [4]. Der so generierte Key wird für die weitere Nutzung zunächst gespeichert. Zu beachten bleibt, dass bei einer produktiven Applikation ein anderes Zertifikat verwendet werden sollte und damit ein neuer API-Key generiert werden muss.

Map View

Zur Darstellung der Map benötigen wir zunächst eine View. Dazu legen wir unterhalb des Verzeichnisses „res/layout" die Datei „mymapview.xml" an. Der Inhalt ist in Listing 2 dargestellt. Hier wird der oben generierte API-Key eingetragen. Die View wird wie üblich von einer Activity verwendet. Das Google-Maps-API sieht dafür die Klasse „MapActivity" vor, von der wir unsere neu zu erstellende Klasse „MyMap" (siehe Listing 3) ableiten. Anschließend muss das Manifest entsprechend angepasst werden. Zum einen registrieren wir dort die neue Activity und die Benutzung des API, zum anderen benötigen wir eine zusätzliche Berechtigung, da das API Daten aus dem Internet nachlädt. Die neuen Einträge sind im Android-Manifest (siehe Listing 4) kenntlich gemacht.

Klasse MyMap

Die Klasse „MyMap" initialisiert zunächst die View für die Kartendarstellung mit den Zoom-Elementen und in einem mittleren Maßstab. Der „LocationListener" wird durch die Klasse zusätzlich implementiert. Der Einfachheit halber wird dies hier genauso durchgeführt wie in der Activity für die textuelle Darstellung. In einer produktiven Applikation würde man diese Funktionalität auslagern und wiederverwendbar machen.

Im Gegensatz zur Aktualisierung der textuellen Geokoordinaten wird die Kartenansicht über einen sogenannten „MapController" gesteuert. Er wird direkt über die View ermittelt und ist in der Lage, die Karte über eine weiche Animation auf vorgegebene Koordinaten zu bewegen. Dazu wird er bei geänderter Position über die Methode „onLocationChanged" veranlasst.

Overlays

Die Kartendarstellung in Form der Klasse „MapView" kann beliebig viele Overlays

```
keytool -list -alias androiddebugkey
        -keystore <SDK_HOME>\.android\debug.keystore
        -storepass android -keypass android
Zertifikatsfingerabdruck (MD5):
E4:9F:53:35:0D:39:64:C0:DC:17:73:B5:90:E3:63:CF
```

Listing 1: Fingerprint

```
<?xml version="1.0" encoding="utf-8"?>
<com.google.android.maps.MapView
        xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/mymapview"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:apiKey="HIER API-KEY EINTRAGEN"
   android:clickable="true" />
```

Listing 2: mymapview.xml

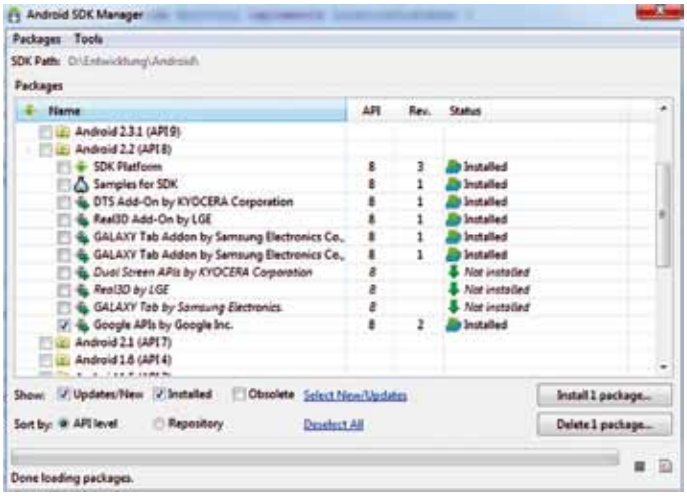


Abbildung 1: Google APIs im SDK Manager

verwalten. Overlays muss man sich als beschreibbare, durchsichtige Folien vorstellen, die über die eigentliche Karte gestapelt werden. Man kann sie nach eigenen Vorstellungen frei gestalten und der View hinzufügen. Für bestimmte Zwecke bringt das Google-Maps-API auch schon vorgefertigte Overlays mit. Die Anzeige der aktuellen Geoposition ist ein solcher Sonderfall, der über die Klasse „MyLocationOverlay" implementiert wird.

In der Methode „onCreate" der Klasse „MyMap" wird ein Overlay-Objekt instanziiert und in die Liste der verwalteten Overlays eingefügt. Jedes Overlay kann einzeln aktiviert beziehungsweise deaktiviert werden. Außerdem benötigt es eine Aktion (als „Runnable"-Instanz), die die Ansicht beim ersten Fix des GPS-Device initialisiert (siehe Listing 3). Der Test im Emulator lässt sich wie gewohnt durchführen. Sofern der Host-Rechner mit dem Internet verbunden ist, werden die Kartendaten entsprechend nachgeladen.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="de.objectsystems.mylocation"
    android:versionCode="1"
    android:versionName="1.0">
  <uses-sdk android:minSdkVersion="8" />
  <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"></uses-permission>
                    <uses-permission android:name="android.permission.INTERNET">
                    </uses-permission>

  <application android:icon="@drawable/icon"
android:label="@string/app_name">
    <uses-library android:name="com.google.android.maps" />
    <activity android:name=".MyLocation"
          android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:name=".MyMap"></activity>
  </application>
</manifest>
```

Listing 4: AndroidManifest.xml

Sonstiges

Wie auch bei der textuellen Darstellung tragen wir in den Methoden „onResume" und „onPause" den Energiespar-Aspekten Rechnung. Ein kleiner Unterschied existiert jedoch bei der Kartendarstellung, die durch einen Aufruf der Methode „invalidate()" bei der Reaktivierung der Activity zu einem Neuzeichnen explizit gezwungen werden muss (siehe Listing 3, Seite 66). Zudem ist zu erwähnen, dass die Klasse „GeoPoint" nicht auf den Fließkommawerten der Koordinaten, sondern auf Integer-Werten beruht. Dies zeigt sich durch die Umrechnung mit dem Faktor „1E6" in der Methode „onLocationChanged".

Fazit

Das Google-Maps-API bietet vielfältige Möglichkeiten der Kartendarstellung, die auch über das hier Gezeigte hinausgehen. So lassen sich mit ihm beispielsweise Geodaten referenzieren (Mapping-Koordinaten zu Adressen oder Points of Interest), Routen darstellen oder Satellitenansichten anzeigen. Das API ist leicht anzuwenden und gut dokumentiert. Für diejenigen, die nicht auf eine Google-Lösung bauen möchten oder (aus rechtlichen Gründen) können, sei darauf hingewiesen, dass mit osmdroid [5] eine nahezu vollständige und freie Implementierung als Ersatz für die Klasse „MapView" basierend auf den Daten des OpenStreetMap-Projekts [6] existiert.

```
public class MyMap extends MapActivity implements LocationListener {
    LocationManager locationManager;
    MapView myMapView;
    MyLocationOverlay myLocationOverlay;

    @Override
    public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.mymapview);
       myMapView = (MapView)findViewById(R.id.mymapview);
       myMapView.setBuiltInZoomControls(true);
       myMapView.getController().setZoom(13);
       myLocationOverlay = new MyLocationOverlay(this, myMapView);
       myMapView.getOverlays().add(myLocationOverlay);
       myLocationOverlay.enableMyLocation();
       myLocationOverlay.runOnFirstFix(new Runnable() {
          @Override
          public void run() {
                                          myMapView.getController().animateTo
                                                (myLocationOverlay.getMyLocation());
          }
       });
       locationManager = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
    }
    @Override
    protected boolean isRouteDisplayed() {
          return false;
    }
    @Override
    protected void onPause() {
          locationManager.removeUpdates(this);
          super.onPause();
    }
    @Override
    protected void onResume() {
         locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, this);
         myMapView.invalidate();
       super.onResume();
    }
    @Override
    public void onLocationChanged(Location location) {
       MapView myMapView = (MapView)findViewById(R.id.mymapview);
       MapController mapController = myMapView.getController();
       int latitude = (int)(location.getLatitude() * 1E6);
       int longitude = (int)(location.getLongitude() * 1E6);
       GeoPoint geoPoint = new GeoPoint(latitude, longitude);
       mapController.animateTo(geoPoint);
    }
    ...
}
```

Listing 3: MyMap.java

Links

[1, 2 und 3] Verweis auf Java-Aktuell-Artikel

[4] Internetseite des Google Maps API-Key: http://code.google.com/intl/de-DE/android/add-ons/google-apis/mapkey.html

[5] Internetseite zur Erzeugung eines API-Key: http://code.google.com/android/maps-api-signup.html

[6] Internetseite für osmdroid: http://code.google.com/p/osmdroid/

[7] Internetseite von Open Street Map: http://www.openstreetmap.de/

Andreas Flügge
info@ordix.de

Andreas Flügge ist seit 2002 Senior Consultant bei der Object Systems GmbH, eine Tochtergesellschaft der ORDIX AG. Seit 1999 ist er im Java-Umfeld tätig. Seine Schwerpunkte sind Java-Enterprise-Umgebungen und Softwarearchitekturen.

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

Redaktionsbeirat:
Ronny Kröhne, IBM-Architekt;
Daniel van Ross, NeptuneLabs;
Dr. Jens Trapp, Google

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

Titel, Gestaltung und Satz:
Claudia Wagner
DOAG Dienstleistungen GmbH

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

Mediadaten und Preise:
http://www.ijug.eu/images/vorlagen/2011-ijug-mediadaten_java_aktuell.pdf

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

Java aktuell
Magazin der Java-Community

www.ijug.eu

JETZT ABO BESTELLEN

Sichern Sie sich 4 Ausgaben für 18 EUR

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

FAXEN SIE DAS AUSGEFÜLLTE FORMULAR AN

0700 11 36 24 39

ODER BESTELLEN SIE ONLINE

go.ijug.eu/go/abo

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

Java aktuell

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

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

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

ANSCHRIFT

Name, Vorname

Firma

Abteilung

Straße, Hausnummer

PLZ, Ort

GGF. RECHNUNGSANSCHRIFT

Straße, Hausnummer

PLZ, Ort

E-Mail

Telefonnummer

X

Die allgemeinen Geschäftsbedingungen* erkenne ich an, Datum, Unterschrift

*Allgemeine Geschäftsbedingungen:

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