1-2013 | Frühjahr | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Java aktuell

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

Java – Programmiersprache mit Tiefgang

Neu auf dem Markt

Mainframe

Wissen für Entwickler

Java und Oracle

4 191978 304903 01

iJUG Verbund

Wolfgang Taschner
Chefredakteur Java aktuell

Java ist erwachsen

Die Java User Group Stuttgart stellt den entsprechenden Rahmen und unser Autor Oliver Böhm findet die richtigen Worte: Es sind mittlerweile 20 Jahre vergangen, seit Java aus dem „Green"-Projekt hervorgegangen ist (siehe Seite 60). 20 Jahre sind ein Alter, bei dem unsereins gerade erwachsen wird. In der IT ist das anders; nur wenige Produkte werden überhaupt so alt. Java erfreut sich bester Gesundheit und dank einer aktiven Community sind Fallstricke und Sackgassen weitgehend ausgeblieben. Von daher gilt es nach wie vor, Java weiterhin gemeinsam auf dem richtigen Kurs zu halten.

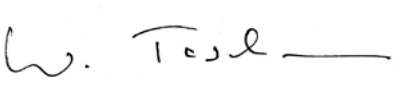
Ob es mit Java wohl eines Tages so geht wie mit dem IBM-Mainframe? Marc Bauer und Tobias Leicher zeigen in dieser Ausgabe auf, wie wichtig manche Applikationen, die weit zurück im letzten Jahrtausend in Cobol oder PL/I geschrieben wurden, insbesondere für große Unternehmen sind. Gleichzeitig erklären sie, wie sich heute Java-Anwendungen mit diesen Uralt-Applikationen verbinden lassen. Wie wohl Java in dreißig Jahren ausschauen mag?

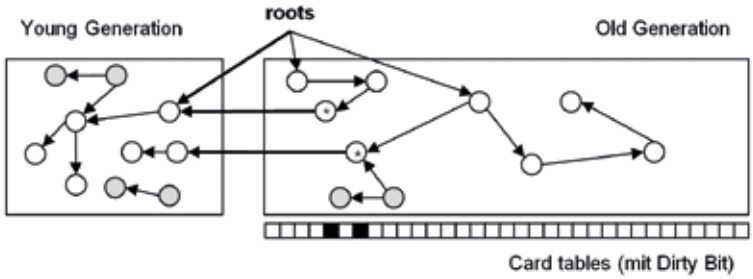
In eine ganz andere Richtung blickt Angelo Veltens in seinem Artikel über Linked Data. Ein Thema, mit dem er sich seit dem Studium beschäftigt und das ihn seitdem nicht mehr loslässt. Im Prinzip geht es darum, die Daten, die sich hinter einer Webseite verbergen, zu standardisieren und allgemein verfügbar zu machen.

Ebenfalls in die Jahre gekommen ist Oracle Forms. Unter anderem dank einer Initiative der DOAG Deutsche ORACLE-Anwendergruppe e.V., die Gründungsmitglied im iJUG ist, konnte das bei Entwicklern beliebte Tool vor einigen Jahren vor dem Aus durch Oracle bewahrt werden. Unser Autor Björn Christoph Fischer zeigt in seinem Artikel, wie man solche Forms-Anwendungen nach Java migriert. Frank Christian Hoffmann hingegen gibt hilfreiche Tipps, wie man Java Beans in Oracle Forms einbindet. Darüber hinaus wird es viele Oracle-Entwickler freuen, dass der Hersteller eine lizenzfreie Version seines Application Development Framework (ADF) auf dem Markt gebracht hat (siehe Seite 45).

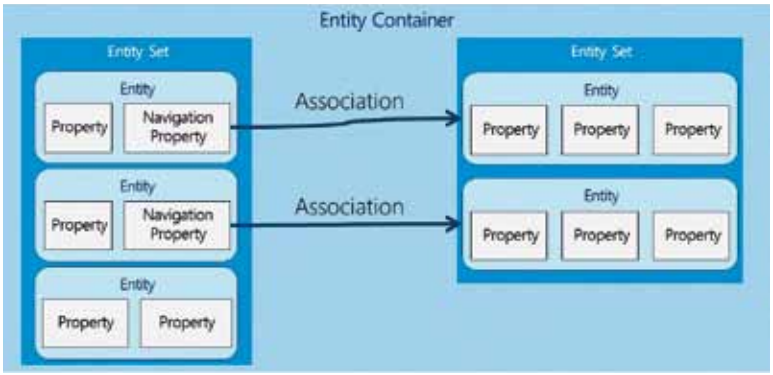
In diesem Sinne wünsche ich Ihnen viel Erfolg mit Ihren Entwicklungsprojekten und hoffe, dass Ihnen auch der eine oder andere Artikel in dieser Java aktuell dabei weiterhilft.

Ihr



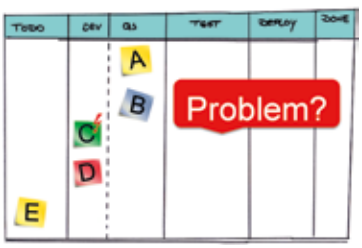


Intergenerational References bei der Garbage Collection, Seite 36



Das Entity Data Model im Open Data Protocol, Seite 30

Das neue Framework, Seite 18



Agiles Entwickeln, Seite 61

Das Java-Tagebuch

Andreas Badelt, Leiter der DOAG SIG Java

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

25. Juli 2012

Codename One

In den Java-Blogs auf oracle.com findet sich ein Hinweis auf ein vielversprechendes Projekt namens „Codename One". Dabei geht es um Werkzeuge zur Erstellung mobiler und nativer Apps auf allen gängigen Plattformen aus Java heraus. Der Java-Code wird dann beispielsweise zunächst in Objective-C übersetzt, bevor er in eine iPhone-App kompiliert wird. Plug-ins sind für Eclipse und NetBeans erhältlich, es ist auch ein GUI-Builder verfügbar. Auf den ersten Blick sieht das recht ausgereift aus und könnte eine Lösung für die fragmentierte „Mobile-Apps"-Landschaft sein. Allerdings ist die Nutzung nur in der Basis-Version kostenfrei, wobei der Support und die Anzahl der Builds pro Monat eingeschränkt sind.
https://blogs.oracle.com/kevinr/entry/codename_one

25. Juli 2012

Umfrage: Die Zukunft von JavaFX

Eine java.net-Umfrage zur Zukunft von JavaFX hat ein sehr uneinheitliches Meinungsbild ergeben. Befragt, ob JavaFX jemals führend für die Entwicklung von Java-Client und Desktop-Apps sein wird, antworteten 23 Prozent der immerhin 660 Teilnehmer mit „Niemals". Fast ein Drittel allerdings glaubt, dass das innerhalb der nächsten zwei beziehungsweise fünf Jahre (14 Prozent und 17 Prozent) eintreten wird. Weitere sechs Prozent sind der Meinung, dass es bis zu zehn Jahre dauert, drei Prozent rechnen damit, dass JavaFX mehr als zehn Jahre brauchen wird, sich aber dann doch durchsetzt. Die Zahl der „Ich weiß nicht"- und „Andere"-Stimmen ist recht hoch (12 beziehungsweise 26 Prozent), was zusätzlich unterstreicht, wie schwer es ist, JavaFX einzuordnen. Wie heißt es doch gleich: „Prognosen sind schwierig, besonders wenn sie die Zukunft betreffen."
Nachtrag: Oracle-Blogger Geertjan Wielenga hat am 10. Oktober 2012 einen interessanten Kommentar zum Verhältnis Swing und JavaFX anhand der Duke's Choice Awards 2012 veröffentlicht. Als Reaktion auf die Aussage „Swing took a bit of a beating this week" auf der JavaOne konstatiert er, dass mehr Preisgewinner Java Swing nutzen als irgendeine andere UI-Technologie (wenn man seiner Argumentation folgt): die NATO mit „MICE", Liquid Robotics, AgroSense sowie das Flüchtlingshilfswerk der UN mit einem NetBeans-basierten Registrierungstool.
http://weblogs.java.net/blog/editor/archive/2012/07/28/will-javafx-ever-become-predominant-developing-java-clientdesktop-apps-poll-result

4. August 2012

Wer kennt die JEPs?

java.net-Chefredakteur Kevin Farnham rührt die Werbetrommel für den „JDK Enhancement-Proposal & Roadmap Process" (JEP) und ich greife das gerne auf, da dieser Prozess ein geeignetes Mittel ist, um alle Interessierten in die Weiterentwicklung des JDK einzubeziehen. JEPs sollen dazu dienen, Verbesserungsvorschläge jenseits von trivialen Änderungen strukturiert zu erfassen und daraus Roadmaps für die zukünftigen JDK-Releases zusammenzustellen. Sie stehen nicht in Konkurrenz zum JCP, sondern sind eine Ergänzung dazu. Jeder kann so Verbesserungsvorschläge für das JDK einreichen (http://openjdk.java.net/jeps/1) und durch diese lässt sich auch leicht verfolgen, was die nächsten Releases bringen werden (http://openjdk.java.net/jeps/0). http://weblogs.java.net/blog/editor/archive/2012/08/04/want-know-where-jdks-going-follow-jeps

8. August 2012

Java SE 6: Kostenloser Support verlängert

Ursprünglich sollte der kostenlose Support für das JDK 6 im November eingestellt werden. Nach vielen Diskussionen wird dieser jetzt doch bis Februar 2013 verlängert; unter anderem hatte der iJUG eine Umfrage durchgeführt (siehe Tagebuch in der letzten Ausgabe der Java aktuell beziehungsweise http://www.ijug.eu/home-ijug/aktuelle-news/article/oracle-reagiert-auf-forderung-der-anwender-und-verlaengert-jdk-6-support.html). Etwas mehr Zeit also, die Migration auf Java SE 7 zu vollziehen und alle gegebenenfalls noch hinderlichen Bugs auszumerzen. Wie erwartet, ist das Echo geteilt: Die einen sind froh über diese Entscheidung, für die anderen lässt damit der Druck zu sehr nach, damit alle auf das neueste Release umsteigen und gemeinsam Java nach vorn bringen.
https://blogs.oracle.com/henrik/entry/java_6_eol_h_h

10. August 2012

Kein Jigsaw im JDK 8 – heiße Diskussionen oder heiße Luft?

Nach Mark Reinholds Ankündigung, das Java-Modulsystem „Project Jigsaw" auf das JDK 9 zu verschieben, gab es teilweise heftige Reaktionen aus der Community (Blogger Markus Eisele sprach von: „Plan B? That is Plan N … Nothing"). Auf java.net wurde daher eine Umfrage durchgeführt mit dem Ziel, ein etwas umfassenderes Stimmungsbild aus der Community zu

bekommen. Sie endete gestern. Die Beteiligung (445 Teilnehmer) war recht niedrig für ein so heiß diskutiertes Thema. Liegt das an der Sommerpause? Oder denkt eine große Zahl der Entwickler, dass für ein eingebautes Modulsystem nicht nur der JDK 8 „Release Train", sondern generell der Zug abgefahren ist? Haben sie sich bereits mit OSGi und anderen Optionen arrangiert und die Diskussionen sind nur noch für den harten Kern der Community relevant? Von denjenigen, die abgestimmt haben, will die größte Gruppe (29 Prozent) lieber das gesamte JDK 8 verschieben, während 23 Prozent die Einhaltung des JDK-8-Zeitplans als wichtiger ansehen. Immerhin 20 Prozent sind der Meinung, dass Jigsaw sowieso zu spät kommt und es keine Rolle mehr spielt. Der Rest entfällt auf „Ich weiß nicht" und „Andere".
http://weblogs.java.net/blog/editor/archive/2012/08/11/poll-result-mixed-views-project-jigsaw-removal-java-8

14. August 2012

Java SE 7 Update 6:
Mac OS X und ARM-Port im Fokus

JDK 7u6 ist veröffentlicht worden. Die wesentlichen Neuerungen: Der Mac-OS-X-Port ist jetzt vollständig. Außerdem gibt es einen Port für Linux auf ARM-Prozessoren, der allerdings nicht alle JDK-Features umfasst und kein „Desktop JRE" enthält (was für die ARM-Zielgruppe sicher kein Hindernis ist). Henrik Ståhl erklärt in seinem Blog Details dazu, insbesondere zum Linux/ARM-Port.
https://blogs.oracle.com/henrik/entry/oracle_releases_jdk_for_linux

16. August 2012

Ceylon: Milestone 3 erreicht

Die Programmiersprache Ceylon hat vor kurzem Milestone 3 (von 5 auf dem Weg zum Release 1.0) erreicht, die Ceylon IDE (ein Eclipse-Plug-in) ist heute bereits in der Version M3.2 freigegeben worden. Einige der neuen Features: „Nested Interfaces", anonyme Funktionen und Zugriff auf hierarchische Filesysteme. Außerdem kann der Compiler jetzt auch JavaScript statt Bytecode erzeugen. Was mir aber am besten gefallen hat, ist der Link auf das „unofficial manifesto" im Blog (http://ceylon-lang.org/blog/2012/08/13/latest-news/ beziehungsweise http://progmofo.com), das ich zugegebenermaßen noch nicht kannte: eine nicht ganz ernst gemeinte Replik auf das Agile Manifest für den Agile-gestressten Entwickler.
http://ceylon-lang.org/documentation/1.0/roadmap/#milestone_3_done

24. August 2012

Mark Reinhold veröffentlicht
Q&A-Blog-Eintrag zu Project Jigsaw

Oracle reagiert auf die Diskussionen um Jigsaw und das JDK 8. Mark Reinhold hat in seinem Blog einen Q&A-Eintrag zu diesem Thema veröffentlicht. Ein guter und gut vorbereiteter Schritt, um die Community einzubinden: Unter anderem verlinkt er auf eine Mail aus dem OpenJDK-Verteiler mit einer Liste von (externen) Blogs und Artikeln, die für die Entscheidungsfindung der Expert Group zusammengestellt wurde. Weiteres Feedback möge über die „Java SE 8 Comments List" direkt an die Expert Group gesendet werden. Die Sache ist bei Erscheinen des Tagebuchs zwar schon abgeschlossen, aber es sei hier als generell gute Möglichkeit erwähnt, Feedback zu allen möglichen anderen Themen direkt an die „Entscheider" hinter dem JDK-Projekt zu schicken.
http://mreinhold.org/blog/late-for-the-train-qa

29. August 2012

Umfrage zu regelmäßigen „Release Trains" für Java

Eine weitere java.net-Umfrage endete gestern, diesmal zur Frage „Soll Java einen festen Release-Zeitplan haben"? Während bei der vorherigen Umfrage konkret zu Project Jigsaw nur 23 Prozent der Teilnehmer den Zeitplan über den Inhalt gestellt haben, sieht es bei dieser allgemeineren Frage schon anders aus: 48 Prozent sprechen sich dafür aus, weitere zwölf Prozent wünschen sich einen festen Zeitplan „mit etwas Flexibilität". Nur 18 Prozent würden die Release-Pläne vollständig der Verfügbarkeit von sinnvollen Neuerungen unterwerfen. Der Rest hatte keine feste Meinung (5 Prozent) oder etwas ganz anderes im Sinn (17 Prozent). Die Teilnehmerzahl war diesmal wieder etwas höher (683).
http://weblogs.java.net/blog/editor/archive/2012/08/29/poll-result-community-prefers-major-java-releases-regular-time-intervals-also-lambda-jigsaw-and-m

30. August 2012

Java EE 7 Roadmap „realigned"

Nach den Diskussionen um Java SE 8 wird nun auch die Release-Planung für Java EE 7 neu aufgerollt. Specification-Lead Linda DeMichiel berichtet von zu langsamem Fortschritt für wichtige Aspekte der PaaS- und Multi-Mandanten-Features. Teilweise liegt das an dem generell langsamen Reifeprozess dieser Themen allerorts – schließlich soll im Rahmen des JCP nur eine Standardisierung vorgenommen werden und nicht die eigentliche Innovation stattfinden. Um das Release nicht um mehr als ein Jahr zu verschieben, sollen die entsprechenden Features erst mit Java EE 8 kommen, so der Vorschlag an die Expert Group. Auch hier gibt es enttäuschte Reaktionen aus der Community, auch weil die Ankündigung relativ spät kommt (EE 7 sollte ursprünglich schon im Q4 2012 veröffentlicht werden und wurde vor einiger Zeit auf das Frühjahr 2013 verschoben). Im Gegensatz zu Project Jigsaw scheint aber eine deutliche Mehrheit DeMichiels Meinung zu teilen, dass die Cloud noch ein wenig Zeit braucht.
http://blog.eisele.net/2012/08/realigning-java-ee-7-promise-is-cloud.html

5. September 2012

Apache Wicket 6.0 veröffentlicht

Gute Neuigkeiten für Web-Entwickler: Wicket 6.0 ist veröffentlicht worden – ein deutlicher Zahlensprung von 1.5, die Nummerierung wurde umgestellt und dieses Release ist das sechste „Major Release". Für die neue Version wurde viel im Bereich „JavaScript/Aja" getan. So wurde die eigene JavaScript-Bibliothek komplett durch JQuery ersetzt; gleichzeitig hat man darauf geachtet, den Austausch der Ajax-Implementierung möglichst einfach zu gestalten, sodass auch andere Frameworks

eingesetzt werden können. Ein weiteres Feature ist die (experimentelle) Unterstützung für WebSockets.
https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces31

13. September 2012

Java 7 bei 79 Prozent Anteil?
Henrik Ståhl hat einen Blog-Eintrag mit dem reißerischen Titel „Java 7 Adoption at 79 Prozent" veröffentlicht. Er bezieht sich dabei auf eine aktuelle Statistik des Cloud-Providers Jelastic und schwächt die Aussage direkt etwas ab („this is a single data point"), um dann allerdings hinzuzufügen, dass sich dies mit anderen von Oracle gesammelten Statistiken deckt (Downloads, Online-Aktivitäten etc.). Aber diese Zahlen wirken trotzdem überoptimistisch. Software, die bei Jelastic in der Cloud gehostet wird, ist sicher mehrheitlich für die Cloud neu entwickelt worden. Wer würde da noch mit einem älteren Release loslegen, das bald keinen freien Support mehr hat? Ähnliches gilt auch für andere Statistiken – es gibt einfach mehr Anlässe, ein neues Release herunterzuladen oder Fragen dazu zu stellen, als ein Release, das schon viele Jahre alt ist und bei dem sich die Downloads in der Regel auf Security Updates beschränken. Es wird auch in einem Jahr noch einen hohen Prozentsatz von Anwendungen auf Java 6 (und früher) geben – mit kostenpflichtigem Support oder Mut zum Risiko.
https://blogs.oracle.com/henrik/entry/java_7_adoption_at_79

13. September 2012

Java.net-Umfrage: Welche Sprache wird in zehn Jahren bei Neuentwicklungen für die JVM führen?
Noch eine interessante java.net-Umfrage. Diesmal wurden erstaunliche 1038 Stimmen abgegeben, das Thema scheint also die Leute zu bewegen. Auch etwas überraschend ist das Resultat, nämlich dass die überwältigende Mehrheit (68 Prozent) auch in zehn Jahren noch mit der Sprache Java als „führend" rechnet. Zehn Jahre sind schließlich ein langer Zeitraum. Nur wenige trauen Scala (sieben Prozent), Groovy (vier Prozent), Clojure, JRuby oder irgendeiner anderen bereits existierenden Sprache (je ein Prozent) eine Ablösung von Java in diesem Zeitraum zu. Sechs Prozent glauben, dass es eine noch zu erfindende Sprache schafft.
http://weblogs.java.net/blog/editor/archive/2012/09/13/poll-result-community-believes-java-will-still-be-dominant-jvm-language-10-years-now

21. September 2012

Project Jigsaw verschoben
Nun ist es offiziell: Die Expert Group zu Java SE 8 hat entschieden, Jigsaw auf das folgende Release zu verschieben. Der IBM-Vertreter macht aber deutlich, dass er trotzdem Grundlagen bereits in Java SE 8 sehen möchte (das Metadaten-Format, erste Definitionen von Modulgrenzen), da die Änderungen insgesamt zu groß für ein einzelnes Release seien. „Runtime Enforcement" sei dann ein Thema für Java SE 9. Das wäre zumindest ein Hoffnungsschimmer, damit sich das Schicksal von Jigsaw nicht bei jedem Release wiederholt.
http://mreinhold.org/blog/on-the-next-train

25. September 2012

Oracle bringt neue Java-Embedded-Produkte heraus
Oracle marschiert im Embedded-Segment weiter voran. Heute hat der Konzern zwei neue Produkte veröffentlicht: Java ME Embedded 3.2 (basierend auf dem Java Wireless Client) und die Java Embedded Suite 7.0, die auf Basis von Java SE Embedded einen integrierten Middleware-Stack liefert. Java ist ja ursprünglich für den Einsatz in TV-Steuerungen und Ähnlichem entwickelt worden, aber wer hätte gedacht, dass darauf mal ein GlassFish-Application-Server läuft? Oracle-Blogger wie Terrence Barr und Henrik Ståhl präsentieren direkt Details zum technischen Umfang und zur Zielsetzung. Dabei wird immer wieder der Begriff „Java im Internet der Dinge" verwendet. Dieser Satz geistert schon seit mehr als zehn Jahren herum, aber offensichtlich ist der Markt aus Oracles Sicht jetzt langsam reif. Um das Thema weiter voranzubringen, hat Oracle auch eine eigene „Java-Embedded"-Konferenz am Rande der JavaOne im Oktober in San Francisco organisiert.
https://blogs.oracle.com/henrik/entry/oracle_releases_two_new_embedded

3. Oktober 2012

Java-Deployment über Apples App Store
Die Zusammenarbeit zwischen Oracle und Apple funktioniert. Noch vor nicht allzu langer Zeit musste man befürchten, dass Java auf Mac OS zum Aussterben verurteilt ist. Jetzt gab es auf der JavaOne eine Session, in der ausführlich erklärt wurde, wie Java-Applikationen auf dem Mac eingesetzt werden, insbesondere über den App Store. Details unter:
https://blogs.oracle.com/thejavatutorials/entry/javaone_2012_java_deployment_on

6. Oktober 2012

JavaOne-Nachlese
Die JavaOne ist vorgestern zu Ende gegangen, Zeit für eine Mini-Nachlese. Welche Neuigkeiten und Strategien wurden verkündet, welche Eindrücke sind bei den Teilnehmern hängen geblieben? Die geänderten Roadmaps für Java SE 8 und Java EE 7 waren natürlich ein Thema, aber zur Konferenz waren die Weichen ja bereits gestellt. Die neuen Produkte im Bereich „ME und Embedded" wurden auch schon einige Tage vorher ins Rampenlicht gestellt. Generell fehlten die großen oder überraschenden Neuheiten, aber nach Meinung vieler Teilnehmer stand die Konvergenz der verschiedenen Produkte und Technologien ganz vorne. Ein paar Splitter aus den Keynotes und anderen Sessions, die das auch unterstreichen: Das Zusammenwachsen der HotSpot- und JRockit-VMs soll mit JDK 8 abgeschlossen sein. JavaFX 8 (neue Nummerierung) wird das Default-UI-Toolkit für Java SE 8 Embedded werden. Mit Java SE 9 sollen relevante Teile von JavaFX im JCP standardisiert werden (was offensichtlich bedeutet, dass JavaFX den Standard vorgibt). Project Nashorn, die neue JavaScript-Engine für die JVM, soll im Rahmen des OpenJDK weiterführt

werden. Bei den Tools tauschen NetBeans (dessen neue Version 7.3 zur JavaOne als Beta freigegeben wurde) und JDeveloper zumindest einzelne Features aus (bislang eher von NetBeans in Richtung JDeveloper). Und last but not least: Die JavaOne und James Gosling sind wieder vereint! Der Vater von Java war Teil der Java-Community-Keynote und wurde begeistert empfangen. Generell scheint die JavaOne wieder ein großes Fest für die Community zu sein, das Oracle-Bashing vergangener Jahre war nicht zu hören. Auch wenn nicht alles rund läuft: Beide Seiten können inzwischen konstruktiv miteinander umgehen.

8. Oktober 2012

JavaOne: Hunderte Stunden Material für die „ruhige Jahreszeit"!
Sämtliche JavaOne-Sessions einschließlich der BoF-Treffen und Tutorials sind frei über das Oracle-Media-Network verfügbar – Hunderte von Stunden Material! Das ist ein sehr sinnvolles Geschenk an alle, die nicht auf der JavaOne waren, oder sich dort kaum entscheiden konnten zwischen den vielen parallelen Streams.
https://blogs.oracle.com/java/entry/javaone_content_available_for_free

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

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

Neues von der JavaOne 2012

Mylène Diacquenod, DOAG Online

Oracle betonte auf der Veranstaltung die Kontinuität der Arbeit mit der Community. Die Java-Keynote war auch Anlass, eine Bilanz zu ziehen und die weitere Roadmap vorzustellen.

Java SE erfreut sich insgesamt stärkerer Beteiligung von Unternehmen wie Suse oder CBOE. Auch Open JDK findet immer mehr Zuspruch und verbucht 68 neue Einzel-Kontributoren. Im Java Community Process (JCP) wurde die Zusammenführung der beiden Executive Committees von Java SE und Java ME eingeleitet.
Das Projekt „Penrose", das die Interoperabilität zwischen Project Jigsaw und OSGi sicherstellen soll, wurde vorgestellt. Gearbeitet wurde auch am Open-Sourcing von Java FX sowie dem Projekt „Sumatra". Dieses soll eine transparente Unterstützung von Graphics Processing Units (GPU) und Accelerated Processing Units (APU) für die Java Virtual Machine (JVM) ermöglichen, was wiederum zu einer besseren Performance führen soll. Inzwischen ist Java 7 für zwei weitere Plattformen verfügbar: Mac OSX und Linux/ARM.

Für JDK 8 geht das Projekt Lambda, das 2011 angekündigt wurde, in die finale Phase, während die Modularität Bestandteil von JDK 9 ist. Im Projekt „Nashorn" wird an einer neuen und performanteren JavaScript-Implementierung auf der JVM gearbeitet, die auch besser mit Java zusammenspielt. Ferner existiert seit JDK 7 mit „invokedynamic" eine Unterstützung dynamischer Sprachen auf der JVM. Auch die aktuellen Fortschritte in puncto Modularität basierend auf Jigsaw wurden vorgestellt. Diese ist für Java 9 vorgesehen.
2011 kündigte Oracle auch die Integration von Java FX 3.0 in JDK 8 an. Die Umsetzung soll tatsächlich kommen – allerdings unter dem Namen „Java FX 8". Java FX ist noch nicht Teil der Java-SE-Spezifikationen.
Mit Java Embedded sieht Oracle einen Markt, der kommerzielles Potenzial hat und noch nicht erschlossen wurde. Aus diesem Grund will Oracle die Entscheider im Rahmen der Konferenz „Java Embedded @JavaOne" adressieren. Das sogenannte „Internet of Things" – so das Schlagwort der Keynote – soll von Java Embedded profitieren.

In der jüngsten Vergangenheit war das Java-Plug-in mehrfach aufgrund von Security-Problemen negativ aufgefallen. Oracle beabsichtigt, sich nun davon zu verabschieden. Allerdings wird das Projekt nur langfristig realisierbar sein, damit eine Aufwärtskompatibilität weiterhin gewährleistet werden kann.
Während der Schwerpunkt bei Java EE 7 klar auf HTML 5 und WebSocket lag, steht in Java EE 8 klar die Unterstützung von Cloud-Architekturen im Vordergrund. Deswegen werden Erfahrungen in entstehenden PaaS-Implementierungen gesammelt und an der Mandantenfähigkeit zur Unterstützung einer SaaS weitergearbeitet.

Weitere Informationen

- https://blogs.oracle.com/javaone/entry/the_javaone_2012_sunday_technical
- http://medianetwork.oracle.com/video/player/1871687106001

Mylène Diacquenod ist seit Juli 2011 als Online-Redakteurin bei der DOAG tätig. Sie hat es sich zum Ziel gesetzt, die Online-Medien der DOAG als wichtige Plattform für den Wissensaustausch zu etablieren.

Java als Treiber für den Erfolg der Kunden

Wolfgang Taschner, Chefredakteur Java aktuell

Als führender Web-Dienstleister in der Schweiz und in Deutschland profitiert Namics von den Erfahrungen und der Kompetenz seiner Java-Entwickler. Diese setzen sich intensiv mit der Programmiersprache auseinander und sind auch gewillt, neue Wege zu gehen.

Java-Entwicklung findet bei Namics im Umfeld von kommerziellen Produkten statt. Der Fokus liegt auf der System-Integration von Content-Management- und E-Commerce-Lösungen auf der Basis von Adobe WCMS und Hybris Multichannel Commerce. Daraus entstehen individuelle Web-Lösungen für den Kunden. Deren Liste ist lang, darunter weltbekannte Unternehmen aus der Automobilbranche, aus dem öffentliche Bereich sowie Versicherungen und Banken.

„Unser Anspruch ist es, die Online-Präsenz unserer Auftraggeber nachhaltig zu prägen, um somit die Firmen online messbar erfolgreicher zu machen", erklärt Patrick Habel, verantwortlich für Head of Marketing. „Dazu stellen wir uns den Online-Herausforderungen der Unternehmen und bieten entsprechende Lösungen – bis hin zum kompletten Portal. Alle Aufgaben haben eines gemeinsam: Sie sind businesskritisch für das Unternehmen."

Typischerweise wird ein Projekt in enger Zusammenarbeit mit dem Auftraggeber ausgeführt. Das beginnt mit der Auswahl der optimalen Technologie. Sobald konzeptionell die Weichen gestellt sind, fängt die eigentliche Entwicklung an.

Der Erfolg von Namics ist eng mit dem Content-Management-System von Adobe verbunden, das in Java programmiert ist und über Java an alle Arten von Applikationen angepasst werden kann. „Java ist für uns der Schlüssel zur Flexibilität", erklärt Sven Niedner, Head of Technology. „Die Funktionalität der Standard-Version des Content-Management-Systems ist in vielen Dingen sehr eingeschränkt. Deshalb erweitern wir es überall dort, wo der Kunde Bedarf hat, und implementieren neue Geschäftsfunktionen." Dafür sind bei Namics mehr als hundert Java-Entwickler aktiv. Sie arbeiten projektbezogen in einzelnen Teams, je nach Anforderung zwischen drei und zwölf Entwicklern. Sven Niedner fasst zusammen: „Java hat bei Namics strategische Bedeutung und Tradition. Seit mehr als zehn Jahren implementieren wir Java-Technologie in unseren Commerce-, Content-Management- und Produktinformations-Projekten."

Alle Aktivitäten rund um die Programmierung in Java sind in einer Java-Practice gebündelt, die sowohl die Arbeit der Entwickler als auch die der Architekten und CTOs umfasst. Diese Practice stellt sicher, dass im gesamten Unternehmen eine standardisierte Methodik im Entwicklungsprozess und einheitliche Entwicklungswerkzeuge eingesetzt sind. Sie umfasst auch die Weiterbildung sowie regelmäßige Technologie- und Innovations-Updates. So sind eine konstant hohe Qualität, breit abgestütztes Know-how und eine hohe Innovationsfähigkeit sichergestellt.

Die Technologie ist allerdings nur ein Teil des Erfolgs von Namics. Der Schlüssel sind die Menschen, die Kompetenz besitzen und bereit sind, diese weiterzugeben. Die Mitarbeiterinnen und Mitarbeiter üben eine kreative Arbeit in einem teamorientierten, leistungsfördernden Umfeld aus. Sie können dort kontinuierlich dazulernen und sich weiterentwickeln. Die Beschäftigten kommen je zur Hälfte aus dem gestalterischen und aus dem technologischen Bereich.

Teamarbeit: Im Rahmen eines Kreativ-Tages werden Floße gebaut und anschließend zu einem lustigen Rennen eingesetzt

Stand der Dinge: Das Scrum-Board dokumentiert den Projekt-Status

Namics ist erst vor fünfzehn Jahren in Form eines Spin-offs der Hochschule St. Gallen als Beratungsunternehmen für Lotus Notes entstanden. Später kam das Content-Management hinzu. Das Portfolio ist mit dem Aufkommen neuer Technologien und dem steigenden Bedarf der Kunden laufend gewachsen. Aus dem ursprünglich achtköpfigen Team ist heute ein Unternehmen mit 380 Mitarbeitern in der Schweiz und in Deutschland geworden. „Die Schweizer haben sehr viel Basis-Demokratie und viel Transparenz eingebracht, aber auch viel Pragmatismus und die Ergebnis-Orientierung", so Patrick Habel.

Auf dem Weg zur agilen Entwicklung

„Wir haben vor einem Jahr festgestellt, dass wir besonders leistungsfähig sind, wenn wir agil arbeiten", erläutert Sven Niedner. „Seitdem sind wir dabei, den gesamten Entwicklungsprozess darauf umzustellen. Sobald das Grobkonzept fertig ist, beginnen wir mit dem Programmieren." Da eine solche Umstellung auf agile Entwicklung nicht von heute auf morgen stattfinden kann, sind die Teams noch am Experimentieren und Ausprobieren, welcher Weg am besten realisierbar ist. Gerade hier setzt die Basis-Demokratie ein – das Thema wird auf dem firmeneigenen Blog http://blog.namics.com viel diskutiert.

Das agile Entwickeln erfolgt in mehreren kurzen Iterationen, sogenannten „Sprints". Dies ermöglicht einen schnellen und flexiblen Projektfortschritt in einer sich ständig verändernden Umgebung. Jedes Scrum-Team besteht dabei aus einem Product Owner, einem Scum Master und den Teammitgliedern aus den Disziplinen „Konzeption", „Frontend Engineering" und „Software Engineering". Die vorgegebenen Projektziele erreicht das interdisziplinär aufgestellte Team dann über die einzelnen Sprints und gewährleistet so eine optimale Lösung für die Kunden.

Im Großraum-Büro der Entwickler hängt ein Scrum-Board an der Wand, davor steht ein Tisch mit Poker-Chips. „Diese Poker-Chips repräsentieren den Aufwand für einen bestimmten Teil eines Projekts", erläutert Software Engineer Pascal Erb. „Jeder Entwickler liefert beim Stand-up-Meeting die Anzahl der Chips ab, von der er der Meinung ist, dass er diese im Projekt abgearbeitet hat. Auf diese Weise ist der Stand des Projekts jederzeit auch optisch ersichtlich." Gleichzeitig wird der Projektverlauf auf dem Scrum-Board an der Wand gegenüber dem Team und dem Kunden dokumentiert. Die Entwickler bewegen ihre Zettel und diskutieren untereinander darüber.

Ein weiteres auffallendes Symbol in diesem Raum ist eine originale Verkehrsampel. Sobald Pascal Erb einen Build startet, nehmen alle Entwickler gespannt die Ampel ins Visier. „Wenn der Build Fehler meldet, geht die Ampel auf Rot, ansonsten gibt sie grünes Licht."

In die Jahre gekommen

Zurück zu Java: Benedikt Eger, Senior Software Engineer, sieht große Vorteile durch die weite Verbreitung der Programmiersprache: „Es gibt für jedes Problem nicht nur eine Lösung, sondern gleich mehrere Frameworks, aus denen der Entwickler wählen kann. Das Gleiche gilt für die Entwicklungstools." Er äußert sich auch positiv über die Community: „Selbst auf schwierige Fragen findet man schnell eine Antwort in irgendwelchen Foren. Zudem ist es einfacher, einen guten Java-Entwickler zu finden, als dies bei weniger verbreiteten Technologien der Fall ist."

Sein Kritikpunkt an Java ist, dass neue Sprachfeatures wie Generics recht kompliziert sind, damit sie abwärtskompatibel bleiben. Auch die lange Historie von Java ist ein Komplexitätstreiber. Das findet auch Pascal Erb, der Software Engineer: „Java ist schon sehr in die Jahre gekommen. In dieser Zeit haben sich viele Dinge angesammelt, die jetzt extrem unflexibel sind, beispielsweise, dass wir Funktionen nicht als Objekte behandeln können. Im Vergleich dazu lässt sich mit modernen Sprachen wie Scala oder Ruby vieles einfacher realisieren." Ob diese modernen Sprachen mit Java kombiniert werden dürfen, hängt von der Infrastruktur des Kunden ab. „Leider geht das dann häufig zugunsten von Pure Java aus."

Stefan Bechtold, ebenfalls Software Engineer, sieht im reifen Alter von Java aber auch Vorteile: „Die Sprache ist über die Zeit gewachsen und bringt deshalb eine entsprechende Robustheit mit. Java könnte aber in manchen Bereichen sicherlich eine elegantere und schlankere Programmierung bieten."

Nachwuchs gesucht

Die interessanten Projekte machen Namics auch zu einem interessanten Arbeitgeber. Deshalb ist man auch laufend auf der Suche nach Java-Entwicklern. Nico Gerhold, Human Resources Manager, skizziert die Anforderungen: „Das Wichtigste bei einem Software-Entwickler ist die Leidenschaft für seine Arbeit." Es geht also primär weniger um Noten als um die Begeisterung. „Da die Release-Zyklen der Projekte sehr kurz sind, müssen die Bewerber das hohe Tempo mögen", bringt sie es auf den Punkt. Namics bietet den Bewerbern die Möglichkeit, an einem Schnuppertag die Firma und das Team kennenzulernen. Danach besitzen sie einen relativ guten Eindruck, worauf sie sich einlassen. Attraktive Projekte gibt es genügend.

Über Namics

Das inhabergeführte Unternehmen wurde 1995 gegründet. An den Standorten in Frankfurt, Hamburg, München, St. Gallen und Zürich betreuen 380 Mitarbeiter unter anderem folgende Kunden: ABB, BASF, Bauhaus, Bausparkasse Schwäbisch Hall, Daimler, EnBW, European Patent Office (EPO), Fraunhofer Gesellschaft, Migros, Panalpina, SBB, Swiss Life, UBS, Viessmann und Webasto. Weitere Informationen unter http://www.namics.com/

Datum in Java

Foto: Fotolia

Jürgen Lampe, A:gon Solutions GmbH

Java fehlt es bis heute an einem durchdachten Konzept für die Abbildung des Datums. Der ohne gründlichere Überlegungen eingeführte Typ „java.util.Date" ist nichts anderes als ein unreflektiert aus der Unix-Welt übernommener Zeitstempel. In der Praxis führt das immer wieder zu unerwarteten Fehlern oder unnötig aufwändigen Lösungen.

Dass diese Klasse „java.util.Date" nicht besonders gut gelungen war, ist wohl auch den Entwicklern bei Sun recht schnell bewusst geworden. Inzwischen sind 22 der 33 öffentlichen („public") Methoden beziehungsweise Konstruktoren als überholt („deprecated") gekennzeichnet. Allerdings ist eine echte Aufarbeitung bisher leider unterblieben, was bedeutet, dass innerhalb der Java-Bibliotheken diese problematischen Funktionen weiterhin zum Einsatz kommen. So verwendet z.B. die statische Methode „java.sql.Date.valueOf(String)" immer noch den überholten Konstruktor „Date(int, int, int)". Das kann zu unerwarteten Ergebnissen führen, wenn sich die Zeitzonen-Einstellungen der Datenbank von denen der Java-Anwendung unterscheiden.

Bereits mit Version 1.1 wurden die Klassen „java.util.Calendar"/„GregorianCalendar" kreiert, um die offensichtlichsten Defizite beim Umgang mit „Date" zu beheben. Grundsätzlich leisten diese Klassen bei konsequenter Anwendung das Erwartete. Allerdings gibt es zwei problematische Punkte:

- Kalender benötigen immer eine Zeitzone. Für die bequemere Verwendung muss diese jedoch nicht unbedingt explizit angegeben werden. Falls sie fehlt, wird die Default-Zone der jeweiligen Installation genommen. Da das gängige Programmierpraxis ist, erhält man implizite Abhängigkeiten von der Installation.
- Die Berechnungen des Kalenders sind relativ kompliziert und die Erzeugung eines Kalender-Objekts ist aufwändig. Wenn Datums-Konvertierungen oder -Berechnungen sehr häufig nötig sind, braucht es spezielle Maßnahmen, um zu vermeiden, dass daraus ein Performance-Engpass entsteht.

Die diffuse Interpretation des Datum-Begriffs findet sich auch in der JDBC-Spezifikation wieder. Da „java.sql.Date" eine Subklasse von „java.util.Date" ist, hat man es de facto immer mit einem Zeitpunkt zu tun, der erst in ein Datum konvertiert werden muss. Das kann zu subtilen Fehlern führen, wenn man sich auf die Standard-Einstellungen verlässt, weil SQL für „DATE" keine Zeitzone kennt (nur für „TIME" und „TIMESTAMP"). Die erforderliche explizite Behandlung etwa durch Aufruf der „getDate"-Methode mit „Calendar"-Parameter (seit JDBC 2.0) ist eine völlig unangemessene Verkomplizierung, die überdies Leistung kostet. Außerdem kann die im Allgemeinen unterschiedliche und nicht immer standardkonforme Implementierung des „DATE"-Typs von SQL in den verbreiteten Datenbanken zu weiteren Problemen führen.

Bei Oracle, wo der Typ „DATE" immer mit Zeit, aber standardmäßig ohne Zeitzone abgelegt wird („If no time was specified when the date was created, the time defaults to midnight"), findet sich folgender Hinweis: „There is little need to use the Oracle external DATE datatype in ordinary database operations. It is much more convenient to convert DATE into character format, because the program usually deals with data in a character format, such as DD-MON-YY" [1].

Ein ganz spezielles Problem kann beim Remote Procedure Call (RPC) oder bei anderen Protokollen, die Objekt-Serialisierung verwenden, auftreten, wenn die beteiligten Seiten mit unterschiedlichen Zeitzonen-Einstellungen arbeiten. Weil Datumswerte in der Regel als „Zeitstempel" Mitternacht der entsprechenden Zeitzone realisieren, führt das bei der Übertragung ohne spezielle Vorkehrungen in einer Richtung immer zu einer Verschiebung um einen Tag – in der anderen Richtung nur zu einer dem Zeitzonen-Offset entsprechenden Stundenverschiebung. Dafür kann es keine generelle Lösung geben, weil dieses Verhalten für „Zeitstempel" absolut korrekt ist und die Serialisierung nicht unterscheiden kann, ob ein „Date"-Objekt nun einen Zeitpunkt oder ein reines Datum darstellt.

Aus heutiger Sicht war es ebenfalls falsch, „Date"-Objekte nicht unveränderbar zu machen, ähnlich „Integer" oder „Long" [2]. Das ist zwar keine prinzipielle Frage, aber die Veränderbarkeit hat an vielen Stellen für Verwirrung gesorgt.

Ein Versuch, Unzulänglichkeiten bei der Zeitbehandlung in Java zu beheben, ist das Joda-Time-Projekt [3]. Ebenso wie beim daraus hervorgegangenen JSR 310 besteht das Ziel darin, eine funktionale Alternative für die Standard-Funktionen rund um „java.util.Date" zu bieten. Schwerpunkte sind eine verständlichere API und

bessere Performance. Der Grundgedanke, Datum und Uhrzeit gemeinsam zu verwalten, also das Datum letztlich aus einem Zeitpunkt abzuleiten, bleibt aber unverändert.

Eine echte Datumsklasse in Java

Wichtig ist es, bereits beim Design klar zu unterscheiden, wo ein reines Datum ausreicht und wo doch auf einen „Zeitstempel (Date)" zurückgegriffen werden muss. Es ist nicht immer ganz offensichtlich, ob ein Datum plus Uhrzeit oder ein „Date" die passende Wahl ist. Beispielsweise muss man unterscheiden, ob etwas täglich zur gleichen Uhrzeit oder alle 24 Stunden ausgeführt werden soll. „java.util.Timer. schedule(TimerTask task, Date firstTime, long period)" erledigt das Letztere. Was im allgemeinen Verständnis zunächst nach einer spitzfindigen Unterscheidung aussieht, hat bei jeder Zeitumstellung Konsequenzen.

Auf jeden Fall vermeiden sollte man die wiederholte Konvertierung von „Date" nach „Datum" und zurück. Das verbraucht nicht nur unnötig Leistung, sondern ist ein Indiz dafür, dass der Entwurf nicht konsistent ist. Es kann natürlich sein, dass eine projektexterne Schnittstelle, der man sinnvollerweise ein Datum übergeben sollte, ein „Date" erfordert. Die beste Lösung, die Schnittstelle zu modifizieren, wird nicht immer möglich sein. Dann ist es eine Frage der Abwägung, ob die Vorteile in anderen Teilen der Anwendung den zusätzlichen Aufwand an der Schnittstelle rechtfertigen.

Wenn Datumswerte in eine Datenbank geschrieben oder aus dieser gelesen werden sollen, ist unbedingt zu prüfen, wie das so erfolgen kann, dass die gewonnenen Vorteile dabei nicht wieder verloren gehen. Insbesondere sollte man Konvertierungen in und aus „Date" unbedingt vermeiden. Die einfachste Lösung ist dabei das Ablegen als Ganzzahl. Sie hat aber den Nachteil, dass die Interpretation bei direkten SQL-Abfragen umständlicher ist. Möglicherweise bietet die Datenbank dafür geeignete Hilfsfunktionen oder man kann entsprechende Stored Procedures selbst

```
public class Day {
   final int year, month, day;
   String asString;
   public Day(int year, int month, int day) {
      this.year= year;
      this.month= month;
      this.day= day;
   }
   public int getYear() {
      return year;
   }
   public int getMonth() {
      return month;
   }
   public int getDay() {
      return day;
   }
   @Override
   public int hashCode() {
      return year * 400 + month * 31 + day;
   }
   @Override
   public boolean equals(Object obj) {
      boolean result= false;
      if (obj instanceof Day) {
         Day dayobj= (Day) obj;
         result= dayobj.year==year && dayobj.month==month
               && dayobj.day==day;
      }
      return result;
   }
   @Override
   public String toString() {
      if (asString==null){
         asString= year + "-" + (month<10?"0":"") + month +
               "-" + (day<10?"0":"") + day;
      }
      return asString;
   }
}
```

Listing 1

```
public class JulianDayConverter {
  public static int calculateJD(int year,int month,int day) {
      int y= 240 * year + 20 * month - 57;
      int a= ((367*y /240) * 4 - 7*(y /240)  + 4*day) /4;
      int b =  (4 * a - 3 * (y / 24000)) / 4;
      int jd = b + 1721115;
     return jd;
  }
  public static int calculateJD(Day gregorianDay) {
     return calculateJD(gregorianDay.getYear(),
                  gregorianDay.getMonth(),
                  gregorianDay.getDay());
  }
  public static Day createDayFromJD(int julianDayNumber){
     int g= ((julianDayNumber << 2) - 7468865) / 146097;
     int a= julianDayNumber + 1 + g - (g >> 2);
     int b= a + 1524;
     int c= (20 * b - 2442) / 7305;
     int d= (1461 * c) >> 2;
     int e= 10000 * (b - d) / 306001;
     int day= b - d - 306001 * e / 10000;
     int month= e < 14 ? e - 1 : e - 13;
     int year= month > 2 ? c - 4716 : c - 4715;
     return new Day(year, month, day);
  }
}
```

Listing 2

```
  Date date= …;
  Calendar calendar= new GregorianCalendar(timezone);
  calendar.setTime(date);
  int jdToday= JulianDayConverter.calculateJD(
                  calendar.get(Calendar.YEAR),
                  calendar.get(Calendar.MONTH)+1,
                                 calendar.get(Calendar.DAY_
OF_MONTH));
```

Listing 3

schreiben. Diese Form hat Vorteile, wenn auch auf der Datenbank Berechnungen auf Tages-Basis ausgeführt werden sollen. Als Alternative bietet sich ein Abspeichern als „String“, etwa in der Form „YYYY-MM-DD“ an. Das erfordert zwar vor Berechnungen jeweils eine Konvertierung, ist aber für den Menschen leichter lesbar.

Eine weitere Frage, die möglichst früh geklärt werden muss, ist die nach dem benötigten Bereich gültiger Datumswerte. Ein erheblicher Teil des Aufwands bei den Kalender-Berechnungen in Java wird durch die Berücksichtigung von Datums-Umstellungen (Julianisch nach Gregorianisch) und Zeitzonen-Veränderungen verursacht. Die Zeitzonen sind für das reine Datum außer bei Konvertierungen von und nach „Date“ irrelevant. Für den weit überwiegenden Teil aller Projekte ist die Begrenzung auf die Gregorianische Zeitrechnung keine Einschränkung. Anwendungsgebiete, in denen das nicht ausreichend ist, werden mit den Standardmitteln ohnehin nicht zurechtkommen, da es beispielsweise in Deutschland keinen einheitlichen Kalenderwechsel gab und in einigen Gebieten beide Kalender über Jahrzehnte parallel verwendet wurden. Solche Anwendungen werden hier nicht betrachtet. Wir beschränken uns auf den Gregorianischen Kalender und Daten vom 1. Januar 1583 bis zum 31. Dezember 9999.

Die Julianische Tageszahl

Prinzipiell kann man natürlich jedes beliebige Datum auswählen, um davon beginnend die Tage zu zählen. Es erleichtert aber die Kommunikation und das Verständnis, wenn man auf eine Konvention setzt. In diesem Artikel wird das chronologische Julianische Datum als Berechnungsbasis verwendet. Der Name ist etwas irreführend, da es keinen unmittelbaren Bezug zum Julianischen Kalender gibt [4]. Vorteil dieser Wahl ist, dass die benötigten Algorithmen erprobt und aufbereitet verfügbar sind. Überdies gibt es Online-Tools zur Berechnung [wie 5], die den Test erleichtern.

Die in [4] angegebenen Umrechnungs-Algorithmen sind stabil gegenüber Wertüberschreitungen, sodass man beispielsweise für das Zahlentripel (0, 8, 2012) die gleiche Julianische Tageszahl erhält wie für (31, 7, 2012). Das gilt in analoger Weise für den Monatswert und ist nicht auf die „0“ beschränkt; es sind beliebige positive und negative Werte zulässig, solange man den definierten Datumsbereich dadurch nicht verlässt. Das Datum des Ablaufs einer Monats- oder Tagesfrist kann damit sehr einfach berechnet werden, indem der entsprechende Ausgangswert einfach um die Frist vergrößert und anschließend eine Hin- und Rückkonvertierung ausgeführt wird.

Implementierung

Es folgt nun die exemplarische Implementierung der ausgewählten Strategie. Wir beginnen mit einer Containerklasse für das Datum (siehe Listing 1). Kommentare wurden der Übersichtlichkeit halber weggelassen. Diese Klasse enthält als reiner Container keine Logik, die die oben erwähnte Stabilität gegenüber scheinbar unsinnigen Werten berücksichtigt, das heißt „new Day(2012, 8, 0).equals(new Day(2012, 7, 31))“ ist immer falsch.

Die Umrechnung zwischen „Day“ beziehungsweise den Integer-Werten für „Jahr“, „Monat“ und „Tag“ und der Julianischen Tageszahl (JD) erfolgt durch statische Methoden der Klasse „JulianDayConverter“ (siehe Listing 2). Die Algorithmen wurden aus [4] übernommen und lediglich so umgeformt, dass sie mit Integer-Arithmetik berechnet werden können. Für ihre Herleitung wird auf die dort angegebene Literatur verwiesen. Der Code enthält noch keine Überprüfungen darauf, ob die Beschränkung auf das Zeitintervall „1.1.1583“ bis „31.12.9999“ eingehalten ist. Es hängt vom konkreten Projekt ab, ob ein solcher Test wirklich erforderlich ist.

Bei Bedarf kann die Umrechnung auf Julianische Daten (also Datumsangaben des Julianischen Kalenders) oder andere Kalender erweitert werden. Außerdem kann es sinnvoll sein, Versionen der Methode „createDayFromJD“ zu haben, die jeweils nur einen der Werte „Tag“, „Monat“ oder „Jahr“ zurückliefern, wenn diese häufig gebraucht werden, um so unnötige Instanziierungen von „Day“-Objekten zu vermeiden (siehe Listing 1 und 2).

Vorteile

Im Kasten auf Seite 15 finden sich einige Beispiele für die Verwendung der Methoden. Die Vorteile der Julianischen Tageszahl sind klar ersichtlich:

```
int readableDay=
day.getYear() * 10000 + day.getMonth() * 100 + day.getDay();
```

Listing 4

```
new Day(readableDay / 10000, (readableDay % 10000) / 100, readableDay % 100);
```

Listing 5

- Fortlaufende Folge, deren Teilfolgen leicht zur Indexierung von Feldern und Listen benutzt werden können
- Einfache Berechnung von Fristen oder Tageszahlen
- Effiziente Umwandlung von und nach (Tag, Monat, Jahr)
- Einfache String-Konvertierung

Die Umrechnung erfolgt durch statische Methoden, die auch intern keinerlei Objekte anlegen. Das ist ein wichtiger Unterschied zur Verwendung der „Calendar"-Klasse in Java, die aufwändig initialisiert werden muss und die überdies nicht „thread-safe" ist.

Dadurch, dass hier der Umweg über einen Zeitpunkt – wie er implizit bei allen Datumsrechnungen in Java gegangen wird – vermieden wird, vereinfacht sich die Konvertierung aus dem und in das String-Format ganz entscheidend. Es bleiben nur drei Integer-Werte umzuwandeln, ohne jede weitere Berechnung.

Ein einfacher Test auf einem PC mit Intel 7-2600 CPU (3.40 GHz) und 8 GB Speicher ergab im Interpretermodus (-Xint), dass die Konvertierung JD -> Day -> JD aller circa drei Millionen Werte des Gültigkeitsbereichs weniger als 25 Sekunden benötigt, die Konvertierung der „Day"-Werte in ein „Date" und zurück hingegen etwa 100 Sekunden (eine vorallokierte „Calendar"-Instanz für alle) bis 130 Sekunden (ein „Calendar" pro Datum). Im Server-Modus lauten die bei der Ausführung in einer Schleife weniger aussagekräftigen Werte 0,7 s, 3,0 s und 3,8 s.

Konvertierung von und zu „java.util.Date"

Trotz aller Nachteile ist das Java-Standard-Date unverzichtbar, Konvertierungen sind also unvermeidbar. Trotzdem sollte an erster Stelle immer der Versuch stehen, direkte Konvertierungen zu vermeiden:

- Wenn ein String bereits das Datum enthält, ist es meist günstiger, den Datumsteil nochmal zu parsen.
- Wenn das Date nur gebraucht wird, um es als String zu formatieren, kann man einen eigenen Formatter bauen, der beispielsweise ein „Day"-Objekt als Argument akzeptiert.
- Manche Datenbanken bieten Funktionen zur Darstellung von Datumswerten als ganze Zahlen, die der hier vorgestellten Version weitgehend entsprechen. Wenn das nicht der Fall ist, kann man Datumswerte als Zeichenketten transferieren.

Wenn die direkte Konvertierung unvermeidlich ist, erfolgt sie mithilfe eines „GregorianCalendar"-Objekts (siehe Listing 3). Nicht vergessen darf man dabei die mit Null beginnende Monatszählung in Java – und selbstverständlich wird auch die Zeitzone explizit gesetzt.

Implizite Verwendung

Eine Variante der vorgestellten fortlaufenden Tagesnummer stellt die Verwendung einer unmittelbar lesbaren ganzen Zahl, etwa „19700101" für „1.1.1970", dar. Sie ist leicht aus einem „Day" zu gewinnen (siehe Listing 4). Listing 5 zeigt, wie man umgekehrt eine „Day"-Instanz erhält. Der Nachteil der impliziten Verwendung ergibt sich daraus, dass viele Berechnungen eine Umwandlung in „JD" und anschließend zurück in das lesbare Format erfordern. In den meisten Fällen dürfte dieser zusätzliche Aufwand vernachlässigbar sein.

Alternativen

Wegen der relativ großen Werte der „JD" wurde zu Zeiten des 16-Bit-Integer-Formats oder vorausgehender ähnlich beschränkter „BCD"-Zahldarstellungen das Modifizierte Julianische Datum (MJD) eingeführt, dessen Startpunkt am 17. November 1858 liegt, wobei gilt: JD(x) = MJD(x) + 2400000. Diese Einschränkung bringt auf heutigen Computern keinerlei Vorteile mehr. Das „MJD" kommt hauptsächlich in den Geo-Wissenschaften und der Raumfahrt zum Einsatz.

Grundsätzlich kann jeder beliebige Tag als Basis einer fortlaufenden Tageszählung genommen werden. Wie bereits erwähnt, verwendet Cobol den 1. Januar 1601 als Tag „eins", einige Tabellenkalkulationsprogramme den 1. Januar 1900 beziehungsweise den 1. Januar 1904. Gerade das Beispiel „Excel", das 1900 fälschlicherweise als Schaltjahr ansieht, zeigt, dass bei solchen willkürlich gewählten Stichtagen leicht Fehler unterlaufen können.

Der auf den ersten Blick naheliegende Gedanke, die Tageszahlen parallel zur „UTC"-Epoche vom 1. Januar 1970 aus zu berechnen, bietet ebenfalls keine wirklichen Vorteile. Stattdessen hat man es dann für Daten vor diesem Stichtag mit negativen Werten zu tun, was in der Handhabung unpraktischer ist. Eine Konvertierung von Zeitpunkten in Tageszahlen durch einfache Division ist zwar unter bestimmten Umständen möglich, birgt aber wegen der zu beachtenden Randbedingungen (Sommer-/Winterzeit, Schaltsekunden) erhebliche Risiken.

Die Julianische Tageszahl wird durch Erweiterung um einen gebrochenen Anteil, der die Tageszeit darstellt, zum Julianischen Datum, das dann wieder ein Zeitpunkt ist. Es wird vor allem in der Astronomie verwendet.

Praktische Erfahrungen

In zahlreichen Projekten hat sich gezeigt, dass „Date" in Java unangemessen ist, wenn wirklich nur ein Datum benötigt wird. Deshalb findet man immer wieder Ad-hoc-Lösungen, bei denen Tage von einem willkürlich gewählten Stichtag aus gezählt werden. Ohne klares Konzept sind solche Lösungen jedoch kritisch für die Stabilität des Codes. Insbesondere Konvertierungen in andere Typen („java.util.Date") oder String-Formate erweisen sich häufig als Schwachstellen.

Im Allgemeinen erlauben Tageszahlen effiziente und übersichtliche Programme. Vor allem der Wegfall von Berechnungen, die Kalender-Klassen benutzen, kann eine erhebliche Leistungssteigerung bringen.

Testfälle

Um die Anwendung der Julianischen Tageszahl zu illustrieren, sind hier einige Ausschnitte aus den Testfällen aufgeführt:

```
Day d1= new Day(1990, 1, 1);
int jd= JulianDayConverter.calculateJD(d1);
assertEquals(2447893, jd);
Day day= JulianDayConverter.
createDayFromJD(jd);
assertEquals(d1, day);
assertEquals(d1.toString(), day.toString());
assertEquals("1990-01-01", day.toString());
```

Hin- und Rückrechnung, Vergleich mit Beispielwert aus [4]

```
Day d1= new Day(2012, 7, -2); // == 28.6.2012
int jd= JulianDayConverter.calculateJD(d1);
assertEquals(2456107, jd);
Day day= JulianDayConverter.
createDayFromJD(jd);
assertEquals(new Day(2012, 6, 28), day);
assertEquals("2012-06-28", day.toString());
```

Fristberechnung Tageswert

```
Date date= new Date(13449255136000L);
      //Tue Aug 14 08:25:13 CEST 2012
Calendar calendar= new GregorianCalendar(
      TimeZone.getTimeZone("Germany/Berlin"));
calendar.setTime(date);
int jdGermany= JulianDayConverter.calculateJD(
      calendar.get(Calendar.YEAR),
      calendar.get(Calendar.MONTH)+1,
      calendar.get(Calendar.DAY_OF_MONTH));
assertEquals("2012-08-14",
      JulianDayConverter.
createDayFromJD(jdGermany).
      toString());
calendar.setTimeZone(TimeZone.getTimeZone(
      "America/Los_Angeles"));
int jdPacific= JulianDayConverter.calculateJD(
      calendar.get(Calendar.YEAR),
      calendar.get(Calendar.MONTH)+1,
      calendar.get(Calendar.DAY_OF_MONTH));
assertEquals("2012-08-13", JulianDayConverter.
      createDayFromJD(jdPacific).toString());
```

Konvertierung bezüglich „java.util.Date"

```
Day d1= new Day(2012, 8, 6);
int jd= JulianDayConverter.calculateJD(d1);
assertEquals(0, jd%7);  // ein Montag == 0
Day day= JulianDayConverter.
createDayFromJD(jd);
assertEquals(d1, day);
assertEquals(d1.toString(), day.toString());
```

Wochentagsermittlung

```
Day d1= new Day(2011, 19, 8); // == 8.7.2012
int jd= JulianDayConverter.calculateJD(d1);
assertEquals(2456117, jd);
Day day= JulianDayConverter.
createDayFromJD(jd);
assertEquals(new Day(2012, 7, 8), day);
assertEquals("2012-07-08", day.toString());
```

Fristberechnung Monatswert

Ein typisches Anwendungsbeispiel für das vorgeschlagene Konzept war die Optimierung von Geldautomaten-Befüllungen. Auf der Basis der täglichen Entnahmen der letzten zwei Jahre war eine Prognose für die nächsten sechzig Tage zu erstellen. Dabei waren die Verschiebungen von Wochentagen, Ultimos und Feiertagen zu berücksichtigen. Allein durch Umstellung auf eine fortlaufende Tageszahl konnten die Berechnungen so beschleunigt werden, dass auf einen speziellen Batch-Prozess mit Ablage der Ergebnisse in der Datenbank verzichtet werden konnte und stattdessen die benötigte Prognose jeweils in Echtzeit zur Verfügung stand. Neben diesen Leistungsvorteilen gewinnt der Code an Klarheit und wird dadurch leichter versteh- und wartbar.

Fazit

Im Gegensatz zu Software, die originär aus dem Geschäftsbereich stammt (COBOL, Excel etc.), bietet Java keine angemessenen Mittel zur Behandlung reiner Datumswerte. Dieser Artikel liefert einen Ansatz, mit dessen Hilfe sich dieses Manko in Projekten überwinden lässt. Als Grundlage dient die aus der Astronomie stammende Julianische Tageszahl. Für die Umrechnung stehen schnelle und stabile Algorithmen zur Verfügung. Eine Implementierung in Java wird vorgestellt, wobei es sich dabei naturgemäß nur um die elementaren Funktionen handelt, die in konkreten Projekten durch weitere (Convenience-)Methoden ergänzt werden können.

Neben der direkten Verwendung einer fortlaufenden Tageszahl wird die Variante einer nicht fortlaufenden, aber unmittelbar lesbaren Tageszahl diskutiert. Sie stellt in allen Fällen, in denen der resultierende Overhead tolerierbar ist, einen guten Kompromiss dar.

Abschließend sei nochmal darauf hingewiesen, dass die Konzepte „Datum", „Zeit" und „Zeitpunkt" trotz ihrer Alltäglichkeit nicht trivial sind und sorgfältig unterschieden werden müssen. Unter dieser Voraussetzung ist die beschriebene Behandlung von Werten des Typs „Datum" nicht nur sehr effizient implementierbar, sondern trägt zu klar strukturiertem Code bei.

Quellen

[1] Oracle Call Interface Programmer's Guide, Part Number A96584-01: http://docs.oracle.com/cd/B10500_01/appdev.920/a96584/oci03typ.htm#421890

[2] Joshua J. Bloch: Java: The Good, the Bad, and the Ugly Parts, devoxx 2011, Antwerpen: http://www.devoxx.com/display/DV11/Java++The+Good%2C+the+Bad%2C+and+the+Ugly+Parts

[3] Joda Time – Java date and time API: http://joda-time.sourceforge.net

[4] Julianisches Datum: http://de.wikipedia.org/wiki/Julianisches_Datum

[5] Kalender-Umrechner: http://www.heinrich-bernd.de/calendar/index_html?mode=jd

Jürgen Lampe
juergen.lampe@agons-solutions.de

Dr. Jürgen Lampe ist IT-Berater bei der A:gon Solutions GmbH in Frankfurt. Vor seiner Tätigkeit als Berater wirkte er als Hochschullehrer an einer Technischen Universität. Seit mehr als 15 Jahren befasst er sich mit Design und Implementierung von Java-Anwendungen im Bankenumfeld. An Fachsprachen (DSL) und Werkzeugen für deren Implementierung ist er seit seiner Studienzeit interessiert.

„Wir möchten nicht, dass Java eine Einzelunternehmens-Plattform wird …“

Gil Tene, CTO von Azul Systems und Mitglied im Exekutiv-Komitee des Java Community Process (JCP) , erläutert in einem Interview mit dem Oracle-Java-Magazine den aktuellen Stand von Java und des JCP und deren Entwicklungsmöglichkeiten. Azul Systems, Hersteller skalierbarer Java Virtual Machines (JVMs), ist seit 2003 Mitglied im JCP und seit November 2011 im Exekutiv-Komitee.

Was sind Ihre größten Bedenken hinsichtlich des JCP?

Tene: Wir sind seit rund neun Jahren Mitglied beim JCP. 2011 wurden wir dann auch Mitglied im Exekutiv-Komitee, um uns stärker aktiv am Prozess zu beteiligen. Wie auch die übrige Community waren wir besorgt über die Veränderungen in der Plattform, die durch die Übernahme von Sun durch Oracle stattfinden könnten. Daher beschlossen wir, uns aktiv zu beteiligen und nicht nur daneben zu sitzen und zuzuschauen. Unser Anliegen besteht darin sicherzustellen, dass der JCP nicht von einem einzigen Unternehmen beherrscht wird und dass der JCP für Oracle kein willenloses Werkzeug ist – etwas, das Oracle meiner Meinung nach auch nicht möchte. Das erfordert die Beteiligung der JCP-Mitglieder auf allen Ebenen – angefangen mit dem Exekutiv-Komitee bei der Erarbeitung der Prozesse, Richtlinien und Governance über die Expertengruppen bei den Standardisierungsbemühungen bis hin zu den Leuten, die Code entwickeln und sich auf unterschiedliche Art und Weise beteiligen. Wir möchten nicht, dass Java eine Einzelunternehmens-Plattform wird, was bedeutet, dass die Community zur Führungsposition beitragen muss.

Wie sollte der JCP den Balanceakt zwischen der Förderung von Innovationen und Schaffung fester Standards meistern?

Tene: Diese beiden Aspekte sind nicht immer zwingend widersprüchlich. Für mich persönlich ist es sehr wichtig, dass der JCP in dem Bereich bleibt, in dem er als Standardisierungs- und Community-Anlaufstelle glaubwürdig und produktiv arbeiten kann. Sie haben vielleicht die jüngsten Diskussionen über das Social-Media-API JSR 357 verfolgt, wobei es um ein standardisiertes Social-Media-API für Java geht, das das Exekutiv-Komitee mit acht zu fünf Stimmen abgelehnt hat. Wir haben dagegen gestimmt. Ich denke, dass der JCP auf Gebieten innovieren sollte, die er kontrolliert und die in seinen Aufgabenbereich fallen, und, wenn er kann, glaubwürdige Expertengruppen zusammenbringen sollte, die die Branche bei dem entsprechenden Thema anführen. Aber ich glaube auch, dass der JCP nicht versuchen sollte, außerhalb dieser Grenzen zu innovieren. Wenn der JCP versucht, Dinge zu standardisieren, die weit über die Java-Plattform hinausreichen und in Bewegung sind, ist das meiner Meinung nach völlig falsch und wird nur zu einem Standard führen, den niemand verwendet.

Gil Tene, CTO von Azul Systems und Mitglied im Exekutiv-Komitee des Java Community Process (JCP)

Der JCP soll die Java-Plattform vorwärtsbringen, aber auf Gebieten, in denen sich die Branche bereits bemüht, Handlungsweisen zu stabilisieren, zu modellieren und zu standardisieren. Die Rolle des JCP ist es, zu folgen und nicht zu führen. Stellen Sie sich vor, der JCP versucht, XML zu standardisieren, während es sich im Umbruch befindet und bevor das W3C entscheidet, wie die Standards aussehen sollen. Der JCP ist nicht der richtige Ort für die Standardisierung dieser Dinge, aber er ist sicher der richtige Ort für die Integration externer Standards bei APIs für Java.

Wie können wir die Beteiligung der Community am JCP erhöhen?

Tene: Dafür gibt es gegenwärtig zahlreiche Ideen. Ich glaube, dass der JCP sicherlich ein Ort für Innovationen ist – niemand wird für uns in der Java-Plattform innovativ sein. Wir sollten aber unsere Innovationen auf Bereiche konzentrieren, über die wir die Kontrolle haben und die im Zuständigkeitsbereich des JCP liegen. Wenn beispielsweise eine zukünftige Version von Java SE bestimmte Features mit neuer Syntax, neuen Funktionalitäten oder APIs

haben beziehungsweise neue Möglichkeiten definieren soll, damit Container im Bereich von Java EE funktionieren, kann die Standardisierung hierfür nur im JCP erfolgen. Wir können dafür Prototypen erstellen sowie in Java.net und in Open-Source-Communities arbeiten und mit Implementierungen experimentieren, aber die Definition der Java-Plattform und ihrer Features ist Aufgabe des JPC. Wenn also die Entwicklungs- und Innovationsanstrengungen der Community angegangen und sie beim natürlichen Ablauf des JCP-Prozesses unterstützt werden, ist das meiner Meinung nach die beste Möglichkeit, um die Beteiligung der Community beizubehalten und zu verstärken sowie gleichzeitig die dem JCP obliegenden Aufgaben zu erledigen.

Welche Hauptprobleme muss der JCP lösen?

Tene: Es gibt klärungsbedürftige Fragen hinsichtlich Lizenzierung, IP-Erreichbarkeit und Transparenz, die vielen Leuten Sorgen bereiten. Der JCP hat bereits im Exekutiv-Komitee einige gute Maßnahmen ergriffen, und der nächste Schritt des JCP befasst sich mit der Transparenz und anderen Prozessregeln, insbesondere mit JSR 348. Das führt zu größerer Transparenz in den Arbeitsgruppen. Es ist ein guter erster Schritt, es sind jedoch ein zweiter und ein dritter Schritt im nächsten Prozess des JCP notwendig. Einiges ist struktureller und prozesstechnischer Art, während anderes fundamental ist und sich mit klärungsbedürftigen Fragen zur Zugänglichkeit der tatsächlichen Implementierungen der Technologie-Kompatibilitäts-Kits (TCK) beschäftigt sowie damit, welche Regeln gelten und mit welchen Spec Leads der Zugang möglich ist. Das ist ein gefährliches Terrain, da viele sehr überzeugende Meinungen existieren.

Es gibt Leute, die die ganze Sache in Open Source haben wollen. Ich hingegen gehöre zu denjenigen, die darin keinen gangbaren Weg sehen, und ich glaube, dass eine Umgebung, die Lösungen in jeder beliebigen Form ermöglicht, der richtige Weg zu einem guten Standard ist. Unabhängig davon, ob es sich um eine proprietäre, Gewinn-orientierte und stark kontrollierte oder um eine kostenlose Open-Source-Implementierung handelt, sollten wir in der Lage sein, die gleichen Standards anzuwenden und einzuhalten. Daher glaube ich nicht, dass der JCP spezifische Lizenzbedingungen diktieren oder ausdrücklich sagen sollte, ob bestimmte Lizenzen erlaubt oder nicht erlaubt sind. Ich glaube jedoch, dass einige Richtlinien und Grenzen notwendig sind. So ist beispielsweise ein zuverlässiger und anhaltender Zugang unter vorhersagbaren Bedingungen für Unternehmen, Projekte und Privatpersonen eine grundlegende Notwendigkeit, um in die Umsetzung und Befolgung eines Standards gemäß dem JCP zu investieren. Wir sollten die Mindestanforderungen ganz klar definieren, die die JSRs zur Bereitstellung eines derartigen Zugangs erfüllen müssen. In den letzten Jahren hat es hier sehr viel Stagnation gegeben.

Hat Oracle das Versprechen für mehr Transparenz und Offenheit im JCP eingelöst?

Tene: Ich würde sagen, dass Oracle dabei ist, und nicht, dass es schon eingelöst ist. Es wird sehr viel versprochen und es gibt sehr gute Absichten, aber es ist immer noch zu früh, um zu sagen, dass es auch tatsächlich Ergebnisse gibt. Als Teil der Community betrachten wir das als etwas, das wir ständig im Auge behalten müssen. Zu unserer Aufgabe im Exekutiv-Komitee gehört es auch, den externen, nicht zu Oracle gehörenden Beobachter zu spielen, der darauf hinweist, wenn wir eine falsche Richtung einschlagen. Dazu ein konkretes Beispiel: Der JCP hat sich sicherlich beim JSR 348 richtig verhalten und die Regeln und die Transparenz verbessert, sodass die Anstrengungen der Community mit ihren Wünschen in Einklang stehen. Aber der Prozess ist nur zur Hälfte abgeschlossen. Uns bleibt beispielsweise hinsichtlich des Java Specification Participation Agreement (JSPA), mit dem sich ein zukünftiger JSR befassen wird, noch einiges zu tun.

Die speziellen Bereiche, mit denen wir uns beschäftigen, sind Situationen in Bewegung, bei denen die neuen Richtlinien, die wir festgelegt haben, im Widerspruch zu unserer vorherigen Arbeitsweise stehen. Das heißt, dass Dinge, die vor einem Jahr im vorherigen Prozess noch in Ordnung waren, jetzt den neuen Richtlinien widersprechen. Und einiges davon wurde bisher noch nicht geklärt. Neue Richtlinien, mit denen wir kontrollieren, was wir tun können und was nicht, behindern die Arbeit, die wir vorher erledigen konnten. Die Einbeziehung vertraulicher Informationen in die von den JSR-Expertengruppen diskutierten Unterlagen widerspricht meiner Meinung nach den Transparenz-Richtlinien und Anforderungen von JCP 2.8 und JSR 348, aber die JSPA erlaubt sogar in einigen Fällen, dass die Mitglieder der Expertengruppen die Unterlagen, die einer JSR zugrunde liegen, vertraulich behandeln. Diese Art des Widerspruchs kann die JSR-Arbeit ins Stocken bringen, wenn wir ihn nicht durch Änderungen der Richtlinien und in den nächsten Bemühungen des JCP lösen.

Glauben Sie, dass es in der IT-Community den Eindruck gibt, dass Java ins Alter kommt oder dass es bald durch eine andere Sprache oder Plattform ersetzt wird?

Tene: Ja, ich glaube, es gibt die Auffassung, dass Java veraltet. Es wird viel darüber geredet, dass die Innovationen hier nachlassen. Interessanterweise hört man das normalerweise im Zusammenhang mit neuen Technologien, die eine ältere Plattform ablösen und ersetzen. Aber ich glaube, dass das auf die Java-Plattform nicht zutrifft. Es gibt sie jetzt seit etwa 17 Jahren und sie war überaus erfolgreich. Zwei oder drei Jahre, nachdem Java herauskam, war bereits klar, dass es andere Entwicklungs- und Bereitstellungsplattformen verdrängen würde. Ich sehe gegenwärtig keine neue Plattform, die Java bedroht, so wie Java damals andere Plattformen bedroht hat. Es gibt viele interessante neue Entwicklungen in dynamischen und funktionalen Sprachen, schnelle Entwicklungsverfahren und andere Innovationen, die nicht unbedingt in Java sind, aber der Großteil davon ist für die Java-Plattform ausgelegt und bedroht sie nicht.

Hinweis: Das Interview führte Janice J. Heiss, Java Acquisitions Editor bei Oracle und Technologie-Redakteurin für das Java Magazine. Ins Deutsche übersetzt von Azul Systems, www.azulsystems.com

Ein erster Blick auf Eclipse 4

Dr. Jonas Helming, EclipseSource München GmbH, und Marc Teufel, hama GmbH & Co.

Technologien und Frameworks, die sich nicht weiterentwickeln und neu erfinden, werden früher oder später ausgemustert. Das gilt auch für das so erfolgreiche und verbreitete Eclipse und die Eclipse Rich Client Platform.

Da die Eclipse Community schon immer für ihre Innovationskraft bekannt war, steht nach vierjähriger Entwicklung das erste Major Release „4.2" der neuen Eclipse Application Platform in den Startlöchern, um die Anwendungs-Entwicklung auf Eclipse-Basis zu revolutionieren. Mit der modellierten Workbench, der Verwendung von Dependency Injection, Annotationen und Services sowie der Unterstützung von CSS Styling halten zahlreiche moderne Konzepte Einzug in die Eclipse-Welt. Dieser Artikel ist ein Auszug aus dem anlässlich des Eclipse-4.2-Release erschienenen Buchs „Eclipse 4 – Rich Clients mit dem Eclipse SDK 4.2" von Dr. Jonas Helming und Marc Teufel. Er behandelt einen Aspekt der neuen Plattform, die Verwendung von Dependency Injection, im Detail.

Die Eclipse 4 Application Platform baut mit dem Application Model und Dependency Injection im Wesentlichen auf zwei Grundpfeilern auf. Jede Anwendung, die mit „e4" entwickelt wird, enthält ein sogenanntes „Workbench- oder Applikationsmodell". Die einzelnen Anwendungsteile werden in diesem Modell in einer abstrakten, hierarchischen Struktur zerlegt und verwaltet. Alle wesentlichen Elemente, die später als Bestandteil der Anwendung am Bildschirm erscheinen, sind somit im Application Model definiert, gespeichert (persistiert) und jederzeit zugreifbar. Beispiele für solche Elemente einer Anwendung sind Fenster, Views, Toolbars oder Menüs. Spezielle Renderer sorgen zur Laufzeit dafür, dass dieses abstrakte Modell im Speicher immer mit dem, was gerade auf dem Bildschirm dargestellt wird, synchron ist.

Dabei trennt die Eclipse 4 Application Platform konsequent zwischen dem Modell, das den Aufbau der Anwendung definiert, auf der einen Seite und den Implementierungen einzelner Teile auf der anderen Seite (siehe Abbildung 1). Das macht einzelne Teile der Anwendung leicht austausch- und wiederverwendbar. Im Falle einer View wird im Modell festgelegt, wo und in welcher Größe eine View im Workbench platziert wird. Eine genauere Beschreibung des Application Model ist nicht Teil dieses Artikels, findet sich aber ausführlich im erwähnten Buch wieder. Der eigentliche Inhalt einer View, die im Modell definiert ist, wird dann in einer eigenen Java-Klasse implementiert. Das Framework kümmert sich um das Erzeugen, das Befüllen, das Ausführen und auch das Abbauen eigener Objekte. Damit dieser Ansatz funktioniert, muss es jedoch eine definierte Schnittstelle zwischen dem Framework und den selbst implementierten Klassen geben. Das Eclipse Framework muss konkret wissen, welche Methoden eines eigenen Objekts zu welcher Zeit und mit welchen Parametern aufgerufen werden müssen. Dazu werden in Eclipse 4 im Wesentlichen zwei Konzepte angewandt: das Programmieren mit Annotationen sowie Dependency Injection. Die Verwendung von Dependency Injection in Eclipse 4 wird in diesem Artikel näher vorgestellt.

Dependency Injection

Objekte, die eine Schnittstelle für Eclipse bereitstellen, werden typischerweise in der Benutzeroberfläche angezeigt oder können von dort aufgerufen werden. Beispiele für derartige Objekte sind Views, die innerhalb von Parts in der Workbench angezeigt werden, oder Handler, deren Code nach einer Benutzerinteraktion ausgeführt werden soll. In beiden Fällen muss Eclipse bestimmte Methoden des eigenen Objekts aufrufen. Einfache Beispiele für solche Methoden sind die Initialisierung einer View, in der beispielsweise Listener registriert werden, oder umgekehrt das Schließen einer View, in der ebendiese Listener wieder entfernt werden.

Im Beispiel eines Handlers muss Eclipse wissen, welche Methode das auszuführende Verhalten implementiert. Ein einfacher Weg für ein Framework, entsprechende Methoden zu definieren, ist die Verwendung von Interfaces. In Eclipse 3.x wurde beispielsweise für die meisten Elemente über Interfaces eine Methode „dispose()" vorgeschrieben, die beim Abbau eines Objekts, etwa beim Schließen einer View, aufgerufen

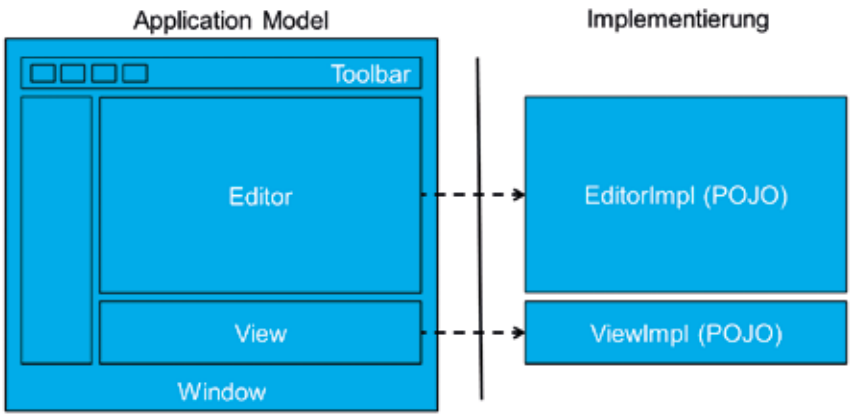


Abbildung 1: Eclipse 4 trennt konsequent zwischen Application Model und Implementierung einzelner Komponenten

```
@Override
public Object execute(ExecutionEvent event)
throws ExecutionException {    //Hier steht
der eigentliche Handler-Code
}
```

Listing 1

```
ISelection selection = HandlerUtil.
getCurrentSelection(event);
if(selection instanceof IStructuredSelection) {
  Object first = ((IStructuredSelection) selec-
tion).getFirstElement();
  MyObject myObject = (MyObject) firstE-
lement;
}
```

Listing 2

```
@Execute
public void execute(MyObject myObject){
   // Hier steht der eigentliche Handler-Code
}
```

Listing 3

wird. Das Interface für die Implementierung von Handlern, also von Klassen, die über Menüs oder Toolbar-Einträge ausgeführt werden, definierte beispielsweise eine Methode „execute()", in der der auszuführende Code untergebracht werden musste. Dieser Ansatz hatte seine Vorteile, denn er gibt dem Entwickler sehr genau vor, welche Implementierung das Framework von ihm erwartet. Dieser Vorteil ist gleichzeitig auch ein Nachteil. Zum einen müssen teilweise Methoden implementiert werden, obwohl sie gar nicht benötigt werden. Zum anderen sind Namen und Signaturen der Methoden fest vorgegeben. Dies macht Implementierungen komplizierter und vor allem deutlich schwieriger zu testen. Ein gutes Beispiel hierfür ist die Implementierung eines Handlers in Eclipse 3.x. Listing 1 zeigt, welche Methode das Interface für die Implementierung vorsieht.

Die Übergabe eines eigenen Event-Typs macht das Interface zwar für Erweiterungen flexibel genug, erfordert aber ein sehr umständliches Auspacken bei der Implementierung. Soll beispielsweise die aktuelle Selektion abgefragt werden, sind nicht wenige Codezeilen notwendig (siehe Listing 2).

Neben der umständlichen Verarbeitung des Events beeinträchtigen vorgegebene Interfaces auch immer die Testbarkeit negativ. Soll nun beispielsweise ein Handler mit JUnit getestet werden, muss zunächst ein „ExecutionEvent" erzeugt werden, in das benötigte Felder eingefüllt werden müssen. Es entsteht somit doppelter Aufwand, auf der Seite der Implementierung und auf der Seite des Tests. Nicht zuletzt legt ein Interface auch die Verwendung des enthaltenen Sourcecodes fest, da es Abhängigkeiten vom zugrunde liegende Framework erzeugt. Gerade im Falle von Handlern könnte der enthaltene Quellcode aber häufig auch in anderem Kontext, beispielsweise für einen programmatischen Aufruf einer Funktion, wiederverwendet werden. Da die Wiederverwendung potenziell auch außerhalb einer Eclipse-Anwendung geschehen kann, ist eine Abhängigkeit von Teilen der Eclipse Workbench schon immer ein Problem gewesen.

Eclipse 4 geht hier einen völlig neuen Weg und verzichtet fast vollständig auf die Vorgabe festgelegter Interfaces. Damit sind die Parameter einer eigenen Methode nicht mehr fest vorgeschrieben, vielmehr kann der Programmierer einer Methode selbst entscheiden, welche Parameter er zur Ausführung benötigt. Eclipse 4 füllt die gewünschten Objekte dann automatisch ein, vorausgesetzt natürlich, sie sind im jeweiligen Kontext vorhanden. Diese sogenannte „Dependency Injection" wird im nächsten Abschnitt detailliert beschrieben. Durch den Wegfall festgelegter Interfaces muss es aber nun einen anderen Weg geben, dem Framework mitzuteilen, wann eine bestimmte Methode von außen aufgerufen werden soll. Dazu werden Methoden, die vom Framework aktiviert werden sollen, über Annotationen markiert. Diese Annotationen geben im Wesentlichen an, wann das Framework eine bestimmte Methode aufrufen soll. Im vorherigen Beispiel eines Handlers muss die Methode, die den auszuführenden Code enthält, mit der Annotation „@Execute" markiert werden. Listing 3 zeigt die Implementierung eines Handlers in Eclipse 4, der zur Ausführung ein bestimmtes Objekt benötigt.

Der Handler muss nicht mehr ein generisches Event verarbeiten, sondern gibt direkt den Parameter-Typ an, den er zur Verarbeitung benötigt. Die Verwendung von Annotationen und Dependency Injection schafft also minimale Signaturen. Das bedeutet, Objekte und Methoden erwarten als Parameter idealerweise genau das, was sie zur Ausführung benötigen, nicht mehr. Dies verbessert die Wiederverwendbarkeit und Testbarkeit beträchtlich, da deutlich weniger Parameter erzeugt werden müssen. Im Beispiel genügt es nun, ein Objekt zu erzeugen, anstatt dieses aufwändig in ein Event zu verpacken.

Objekte können entweder in Felder einer Klasse, als Parameter des Konstruktors oder als Parameter einer Methode, die vom Framework aufgerufen wird, injiziert werden. Im einfachsten Fall werden benötigte Objekte dabei über die Annotation „@Inject" markiert. Zusätzlich existiert eine Reihe weiterer Annotationen, die das Verhalten und den Zeitpunkt einer Injection genauer steuern. Im vorherigen Beispiel spezifiziert die Annotation „@Execute" im Falle des Handlers, dass die markierte Methode bei der Ausführung des Handlers aufgerufen wird und gleichzeitig die dazu benötigten Parameter injiziert werden.

Der Eclipse Context

Die Verwendung von Dependency Injection scheint intuitiv und praktisch, lässt aber die Frage völlig offen, wo die injizierten Objekte eigentlich herkommen. Woher weiß also das Framework, welche Objekte es an welcher Stelle einfügen soll? Dazu gibt es in Eclipse 4 den sogenannten „Eclipse Context". Dieser ist eine Art „Verzeichnis von Objekten", die zum Injizieren bereitstehen. Technisch gesehen ist dieser Context eine Map von Strings und Objekten. Ohne weitere Angaben wird ein Objekt unter seinem vollständigen Klassennamen gespeichert, beispielsweise „org.eclipse.swt.Composite". Wird nun ein Objekt eines bestimmten Typs angefordert, wird der jeweils gültige Context daraufhin durchsucht, ob er ein Objekt des geforderten Typs enthält (siehe Abbildung 2). Dieses Objekt wird dann für den Aufruf von Konstruktoren und Methoden oder zum Befüllen eines Feldes verwendet.

In Eclipse 4 existiert aber nicht nur ein globaler Context; auf diese Weise wäre es sehr schwierig, das korrekte angeforderte Objekt auszumachen. Im klassischen

Beispiel der View, die ein Composite als Parent benötigt, wäre so völlig unklar, welches Composite der Anwendung dies sein soll. Einige Elemente des Application Model haben daher zusätzlich ihren eigenen Context, beispielsweise ein Window, eine Perspective oder einen Part. Diese sind hierarchisch verknüpft; wird beispielsweise im Context eines Parts kein passendes Objekt gefunden, wird an die Perspektive, an das Fenster, an den Workbench Context oder final an den OSGi -Context verwiesen. Dieser enthält dann Objekte, die für die ganze Anwendung gültig sind, beispielsweise Services. Eine detaillierte Beschreibung der verfügbaren Services in Eclipse 4 findet sich im oben genannten Buch.

Generell enthält der Context alle Elemente des Application Model, zugreifen kann man entlang der aufsteigenden Hierarchie (siehe Abbildung 3). So kann man sich beispielsweise aus dem Context eines Parts das Fenster injizieren lassen, in dem der Part enthalten ist (siehe Listing 4).

Weiterhin enthält der Context einige dem Application-Modell assoziierte SWT-Elemente der jeweiligen „Elements", beispielsweise das Composite eines Parts oder die Shell der laufenden Anwendung. Die Eclipse-4-Services sorgen auch dafür, dass bestimmte allgemein zugreifbare Objekte, wie beispielsweise die aktuelle Selection sowie die Eclipse Preferences, im Workbench Context verfügbar sind. Der oberste Context enthält schließlich alle OSGi-Services. Nicht zuletzt können auch eigene Objekte in den Context eingefügt werden, eine Beschreibung dazu findet sich ebenfalls im erwähnten Buch.

```
@Inject
public void myMethod(MWindow window)
{
}
```

Listing 4

```
@Inject
@Named(„org.eclipse.swt.widgets.Composite")
Composite parent;
```

Listing 5

```
@Inject
@Named(IServiceConstants.ACTIVE_SHELL)
Shell shell;
```

Listing 6

@Named

Will man nun nicht nur auf einen bestimmten Typ, sondern auf eine bestimmte Instanz eines Typs zugreifen, kann man bei der Injektion auch einen Namen angeben. Dies geschieht über die zusätzliche Annotation „@Named", die in Kombination mit „@Inject" verwendet wird. „@Named" erlaubt die zusätzliche Angabe eines Strings, der den Namen des zu injizierenden Objekts angibt. In diesem Falle wird nicht mehr nach dem Typ des zu injizierenden Objekts gesucht, sondern nach dem entsprechenden String. Genauer gesagt ist eine Injektion ohne „@Named" nur eine Kurzform, bei der der Typ eines Parameters als Name der zu injizierenden Variablen angenommen wird. Umgekehrt werden Objekte im Context ohne weitere Angabe eines Namen unter ihrem vollständigen Klassenpfad abgelegt. In unserem Beispiel hat „@Named" also keinen Effekt und kann daher weggelassen werden (siehe Listing 5).

Die in Eclipse 4 bereits enthaltenen Services legen bereits einige Objekte unter speziellen Namen in den Context ab, die gültigen Namen finden sich im Interface „IserviceConstants". Auf diese Weise kann beispielsweise die aktive Shell injiziert werden (siehe Listing 6).

Weiterhin ist es möglich, eigene Objekte unter einem bestimmten Namen im Context abzulegen, eine Beschreibung dazu findet sich in genanntem Buch.

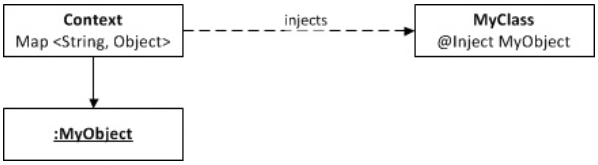


Abbildung 2: Dependency Injection mithilfe des Eclipse Context

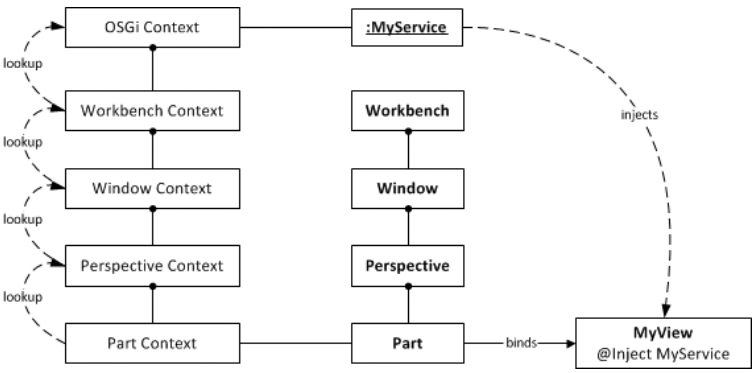


Abbildung 3: Der Eclipse Context ist hierarchisch aufgebaut

@Optional

Nun ist es natürlich immer möglich, dass kein Context ein passendes Objekt enthält, das injiziert werden kann. Der Dependency-Injection-Mechanismus von Eclipse 4 wird in diesem Fall einen Fehler anzeigen, konkret eine Exception werfen. Eine Klasse, die fehlende Parameter im Konstruktor einer Methode oder als Felder benötigt, die nicht injiziert werden können, kann in diesem Fall nicht korrekt initialisiert werden. Einige Parameter werden aber auch nicht in jedem Fall benötigt, beispielsweise die aktive Selection oder bestimmte Services. Für diese Parameter kann die Annotation „@Optional" verwendet werden. Ist ein so markiertes Objekt nicht im Context verfügbar, wird stattdessen „null" injiziert. Wichtig ist natürlich in diesem Falle vor einem Zugriff auf das injizierte Objekt, dieses auf „null" zu überprüfen.

@Active

In bestimmten Anwendungsfällen ist es notwendig, nicht nur auf einen bestimm-

```
@Execute
public void save(@Active MPart part) {

   partService.save(part);
}
```

Listing 7

```
@Inject
public void MyView(Composite parent){
            //Die View auf dem Parent imple
            mentieren
}
```

Listing 8

```
@Inject
ESelectionService service;
...
service.setSelection(mySelection);
```

Listing 9

```
@Inject
public void setSelection(@
Named(IServiceConstants.ACTIVE_SELEC-
TION)@Optional MyObject myObject) {
            //Selection verarbeiten
}
```

Listing 10

ten Typ eines Elements aus dem Application Model zuzugreifen, sondern auf das gerade aktive Element. Mit der Annotation „@Active" wird der gerade aktive Context für die Injection genutzt. So kann beispielsweise ein Handler auf den aktiven Part zugreifen (siehe Listing 7).

Objekte injizieren

Im einfachsten Fall wird die Injection über die Annotation „@Inject" ausgelöst, diese kann vor einzelnen Parametern, vor Methoden, vor dem Konstruktor oder vor Feldern einer Klasse platziert sein. Markiert man eine Methode oder einen Konstruktor mit „@Inject", werden alle ihre Parameter injiziert. Die Suche nach dem passenden Objekt richtet sich dabei, ohne weitere Angaben über „@Named", nach dem entsprechenden Typ des Parameters oder Felds. Alle weiteren, später vorgestellten Annotations steuern den genauen Zeitpunkt der Injektion, verhalten sich aber prinzipiell wie „@Inject".

Eine entscheidende Rolle spielt die Reihenfolge der Injektion. Wird eine Klasse instanziiert, beispielsweise eine View, wird zunächst der Konstruktor aufgerufen und dessen Parameter injiziert. Direkt im Anschluss werden Felder befüllt. Es kann also im Konstruktor noch nicht auf injizierte Felder zugegriffen werden. Parameter von Methoden werden injiziert, wenn diese Methoden vom Framework aufgerufen werden. Mit „@Inject" markierte Methoden werden zusätzlich beim Initialisieren des Objekts aufgerufen, also nach dem Konstruktor und den Feldern. Ändert sich das injizierte Objekt im Context, wird es erneut injiziert. Dabei werden Methoden erneut ausgeführt, wenn sich die injizierten Werte ändern.

Konstruktoren

Konstruktoren sollten Parameter enthalten, die für die Existenz eines Objekts essenziell sind. Jeder unnötige Parameter schränkt die Wiederverwendungsmöglichkeiten und vor allem die Testbarkeit eines Objekts ein. Insbesondere Initialisierungen eines Objekts sollten in eigene Methoden ausgelagert werden, die erst nach dem Konstruktor aufgerufen werden. Ein typisches Beispiel für Dependency Injection im Konstruktor ist das Injizieren des Parents für einen Part. Da die View im Context eines Parts aus dem Application Model initialisiert wird, ist die Angabe des Typs „Composite" in diesem Fall eindeutig (siehe Listing 8).

Felder

Nach dem Konstruktor einer Klasse werden deren Felder injiziert. Ein typischer Anwendungsfall ist das Injizieren von Services, die in der Klasse global verfügbar sein sollen. Ein Beispiel dafür ist der Selection Service, mit dem eine View die aktuelle Selektion setzen kann. Da Services üblicherweise nur einmal pro Anwendung existieren, ist auch in diesem Fall die Angabe des Typs ausreichend (siehe Listing 9). Bei der Injection von Feldern ist zu beachten, dass diese nicht als final markiert sein dürfen. Finale Felder müssen explizit durch den Konstruktor gesetzt und dort injiziert werden.

Lust auf mehr Eclipse 4? Dieser Artikel ist ein Auszug aus dem neuen Eclipse-4-Buch von Marc Teufel und Dr. Jonas Helming, das bei entwickler-press erscheint. An professionelle Software-Entwickler gerichtet, geben die Autoren mit diesem Werk einen Überblick über das Eclipse SDK 4.2 mit e4. Das Buch richtet sich auch an Einsteiger sowie Umsteiger von Eclipse 3.x und zeigt die Möglichkeiten der Rich-Client-Plattform an anschaulichen Beispielen. Es enthält eine Einführung in die Technologie und bietet im weiteren Verlauf einen detaillierten Überblick über alle relevanten Themen, um eine Eclipse-RCP-Anwendung auf Basis von e4 zu programmieren und auszuliefern.

„Eclipse 4 – Rich Clients mit dem Eclipse 4.2 SDK"
Umfang: 275 Seiten
Preis: 34,90 Euro
ISBN: 978-3-86802-063-2
Weitere Informationen:
www.entwickler-press.de

Methoden

Nach Konstruktor und Feldern werden beim Initialisieren einer Klasse alle mit „@Inject" annotierten Methoden nacheinander aufgerufen. Das gilt auch für Methoden, die keine Parameter besitzen. Ändert sich einer der injizierten Parameter einer

Methode im Context, wird die Methode mit den neuen Parametern erneut aufgerufen. Ein gutes Beispiel für eine Injection in einer Methode ist die aktuelle Selection, auf die man häufig in einer View oder in einem Handler reagieren möchte. In diesem Fall reicht aber die Angabe des Typs des zu injizierenden Parameters nicht aus, der Parameter muss zusätzlich mit der Annotation „@Named" versehen werden. Außerdem wird im Beispiel „@Optional" verwendet, da beispielsweise zum Start der Anwendung noch keine Selection im Context vorhanden ist. Im folgenden Beispiel wird die Injection jedes Mal wiederholt, die Methode also erneut aufgerufen, wenn sich die Selektion ändert (siehe Listing 10).

Annotationen

Für Konstruktoren und für Felder ist die Annotation „@Inject" in Kombination mit „@Named" und „@Optional" im Prinzip ausreichend, da der Zeitpunkt, an dem die Injection stattfinden soll, eindeutig ist. Werden Methoden mit „@Inject" markiert, werden diese nach der Initialisierung aufgerufen sowie dann, wenn sich einer der Parameter im Context ändert. In vielen Anwendungsfällen, insbesondere bei der Entwicklung von UI-Komponenten, gibt es aber noch andere interessante Events, auf die man als Programmierer reagieren möchte, beispielsweise wenn eine View fokussiert wird. In Eclipse 3.x wurde zu diesem Zweck per Interface eine Methode „setFocus()" definiert, die vom Framework entsprechend aufgerufen wurde. Da UI-Komponenten in Eclipse 4 jedoch reine POJOS sind, deren Methoden frei benannt werden dürfen, müssen die aufzurufenden Methoden per Annotationen markiert werden, beispielsweise „@Focus".

Alle diese Annotationen sind Spezialisierungen von „@Inject". Das bedeutet, alle Parameter von markierten Methoden werden injiziert. Einzig der Zeitpunkt des Aufrufs einer Methode lässt sich genauer steuern. Die Annotationen „@Inject" und „@Named" sind im Standard JSR-330 (Dependency Injection for Java) definiert. „@PostConstruct" und „@PreDestory" dagegen sind im JSR-250 (Common Annotations for the Java Platform) beschrieben. Eclipse 4 implementiert hier also Standards. Alle weiteren Annotationen entsprechen dagegen keinem Standard, sondern sind Eclipse-spezifisch. Hierzu gehört beispielsweise die bereits beschriebene Annotation „@Optional". Genauere Informationen zu den verfügbaren Annotationen und deren Verwendung sowie zu verwandten Themen wie dem Application Model oder den Eclipse 4 Services finden sich im Buch „Eclipse 4 – Rich Clients mit dem Eclipse SDK 4.2" (siehe Kasten S 21).

Fazit

Dependency Injection erlaubt es, Abhängigkeiten zwischen eigenen Implementierungen und Framework Interfaces aufzulösen. Objekte definieren selbst genau, welche Parameter oder Services sie nutzen. Dies macht entsprechende Klassen besser wiederverwendbar und leichter testbar. Die Angabe von zusätzlichen Tags erlaubt es, genauer zu spezifizieren, welche Objekte injiziert werden, beispielsweise können Parameter als optional markiert werden. Mit der Einführung dieser Technologie wird die Eclipse Application Platform ein gutes Stück flexibler und modularer und ist auch für zukünftige Einsatzgebiete gut gerüstet.

Dr. Jonas Helming
jhelming
@eclipsesource.com

Jonas Helming ist Trainer, Software Engineer Consultant sowie Geschäftsführer der EclipseSource München GmbH. Er ist aktives Mitglied der Eclipse-Community und Projekt Lead der Eclipse-Projekte EMFStore und EMF Client Platform. Der Autor führt unter anderem Trainings zu Eclipse 4 und EMF durch und veröffentlicht Online-Tutorials unter developers.eclipsesource.com.

Marc Teufel
teufel.marc
@googlemail.com

Marc Teufel arbeitet als Projektleiter und Software-Architekt bei der hama GmbH & Co. und ist dort für die Durchführung von Software-Projekten im Bereich internationale Logistik zuständig. Er ist Autor zahlreicher Fachartikel, insbesondere im Web-Services- und Eclipse-Umfeld. Er hat drei Bücher zu Web Services publiziert und spricht regelmäßig auf Fachkonferenzen.

Impressum

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

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

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

Redaktionsbeirat:
Ronny Kröhne, IBM-Architekt;
Daniel van Ross, NeptuneLabs;
Dr. Jens Trapp, Google; Robert Szilinski,
esentri consulting GmbH

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

Titel, Gestaltung und Satz:
Fana-Lamielle Samatin, Claudia Wagner
DOAG Dienstleistungen GmbH

Foto Titelseite: Fotolia

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

Mediadaten und Preise:
http://www.ijug.eu

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

Java aktuell
Magazin der Java-Community

Foto: Fotolia

Immer hübsch der Reihe nach …

Uwe Sauerbrei, emmert CONSULTING + partner

Die Umbrella AG hat eine Expansion in den internationalen Markt beschlossen. Das brandneue Produkt, der europäische Rettungsschirm, soll auf Wunsch des Geschäftsführers mit einer modernen Workflow-Software entwickelt werden. Die Wahl fiel auf Atlassians Jira.

Der Geschäftsführer ist voller Ideen von einem einwöchigen Manager-Lehrgang zum Thema „Workflows und Projektorganisation" zurückgekehrt und hat den Prokuristen, den Entwicklungsleiter sowie Erwin Müller, einen Projektleiter der ersten Stunde, zu einem Meeting einberufen.

Geschäftsführer: Meine Herren, ich will nicht viele Worte machen, aber wir benötigen ein innovatives Produkt, um in den Weltmarkt einsteigen zu können. Wir sind führend in der Produktion von Schirmen und unser neues Ziel wird die Entwicklung eines europäischen Rettungsschirms sein. Müller, was schlagen Sie vor?

Müller: Wir schreiben alle Anforderungen auf, machen eine Spezifikation, modellieren alles mit UML und geben es der Entwicklung.

Entwicklungsleiter: Da sehe ich jetzt schon viele Probleme. Ich garantiere für nichts!

Prokurist: Das wird teuer!

Geschäftsführer: Deswegen machen wir es dieses Mal anders.

Müller: Ach?!

Geschäftsführer: Ja, wir arbeiten agil.

Entwicklungsleiter: Cool!

Geschäftsführer: Für die Verwaltung werden wir Jira von der Firma Atlassian einsetzen. Das wird Ihre Aufgabe sein, Müller.

Müller schluckt und findet so schnell keine Worte.

Prokurist: Das wird teuer!

Geschäftsführer: Mitnichten! Eine Lizenz für zehn Anwender kostet uns zehn Euro im Monat. Wäre unser Rettungsschirm Open Source, würde es uns gar nichts kosten.

Prokurist: Eine gute Wahl.

Geschäftsführer: Müller, in zwei Tagen treffen wir uns wieder und Sie zeigen uns, was mit Jira möglich ist. Koordinieren Sie sich mit dem Entwicklungsleiter!

Müller: Ich werde mein Bestes geben (seufzt).

Zwei Tage später

Geschäftsführer: Meine Herren?

Entwicklungsleiter: Ich wollte ein Testprojekt anlegen, aber das ging alles nicht, weil ich kein Admin oder Sysadmin bin. Wo ist der Unterschied? Ich überflog die Dokumentation und habe mir die Issues und den Workflow angeschaut. Erstere sehen wie folgt aus:

- Bug
- Improvement
- New Feature
- Task
- Custom Issue

Das passt eher zur Software-Entwicklung, aber nicht in die Konstruktion. Ähnlich der Workflow. Hier gibt es nur:

- Open
- In Progress
- Resolved
- Reopened
- Closed

Damit könnten wir schon mehr anfangen, aber mir stellt sich die Frage, ob das Tool wirklich für unsere Zwecke geeignet ist.

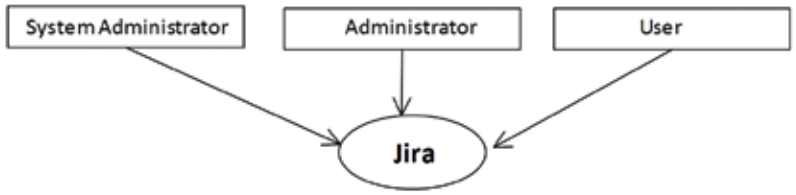


Abbildung 1: Hauptrollen in Jira

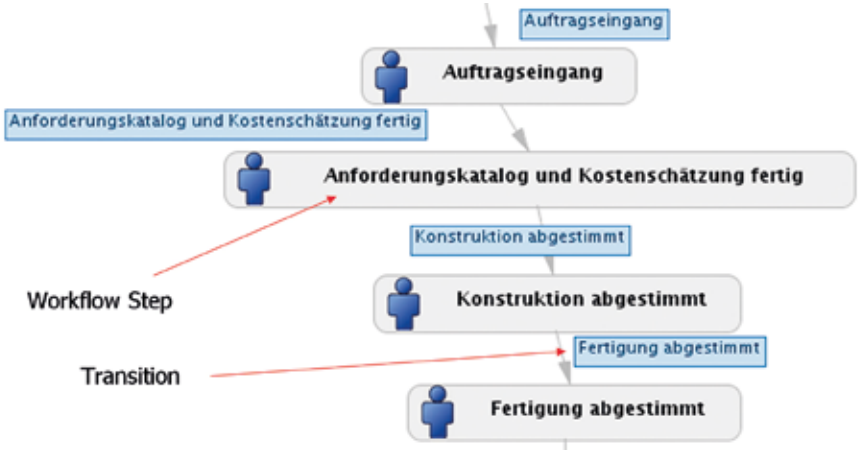


Abbildung 2: Ausschnitt aus dem Workflow

Geschäftsführer: Nun mal langsam. Was erzählen Sie mir hier von Issues und Workflows und Rollen? Lassen wir doch erst mal Herrn Müller zur Wort kommen!
Müller: Nun, ich habe die Dokumentation gelesen. Fangen wir mit den Berechtigungen an. Hier gibt es drei Hauptrollen, die ich nachfolgend skizziert habe (siehe Abbildung 1):

1. Der User ist der User.
2. Ein Administrator kann für sein Projekt bereits existierende User mit bereits existierenden Berechtigungen verknüpfen, Komponenten oder Versionen hinzufügen, kurzum, die fachlichen Inhalte eines Projekts konfigurieren.
3. Die Möglichkeit, einen User in Jira anzulegen, Berechtigungen hinzuzufügen, neue Projekte aufzusetzen und diese Projekte technisch zu konfigurieren, hat ausschließlich der System-Administrator.

Issues sind Vorgänge und repräsentieren die horizontale Auswahl. Jira liefert die vom Entwicklungsleiter bereits erwähnte Kollektion als Standard aus. Es besteht jedoch kein Zwang, genau diese so zu verwenden. Für unseren Rettungsschirm würde ich vorschlagen, wir benutzen drei völlig andere Vorgaben:

- Neukonstruktion
- Modifikation
- Reparatur

Hier haben wir ein konkretes Beispiel für die Aufgabe des System-Administrators, der diese Vorgänge anlegen muss.
Geschäftsführer: Ich verstehe! Sie sprachen gerade von einer horizontalen Auswahl; das impliziert, dass es auch eine vertikale Auswahl gibt?
Müller: Exakt! Das sind die ebenfalls erwähnten Workflows. Jeder Vorgang macht sozusagen eine Zeitreise, durchläuft einen speziellen Lebenszyklus. Ein Wechsel von einer Etappe zur nächsten kann mit Bedingungen und Verantwortlichkeiten verknüpft werden. Ich habe mit der Konstruktion gesprochen und folgende Meilensteine identifiziert, die wir, wenn es um eine Neukonstruktion geht, berücksichtigen müssen:

1. Anlegen (Open)
2. Auftragseingang
3. Anforderungskatalog und Kostenschätzung fertig
4. Konstruktion abgestimmt
5. Fertigung abgestimmt
6. Bestellanforderung generiert
7. Gerät gebaut
8. Dokumentation erstellt
9. Gerät geliefert
10. Auftrag abgeschlossen

Wir können uns einen Ausschnitt des neuen Workflow ansehen (siehe Abbildung 2).
Entwicklungsleiter: Wie genau funktioniert eine Transition?
Müller: Eine Transition definiert einen Übergang von einer Stufe des Workflow zur nächsten. Grundsätzlich gibt es hier mehrere Einstellungen, die konfigurierbar sind:

- *Screen*
  Beim Wechsel kann ein Dialog vorgegeben werden. Für unsere Aufgabe bietet es sich an, einen Kommentar zu schreiben und den nächsten Assignee (Adressaten) über eine Auswahlliste anzubieten.
- *Conditions*
  Hier kann festgelegt werden, welche Voraussetzungen erfüllt sein müssen, damit die Transition ausgeführt werden darf. Wir können Berechtigungen oder Bedingungen definieren, die zu diesem Zeitpunkt geprüft werden.
- *Validators*
  Es besteht weiterhin die Möglichkeit, eine Prüfung der eingegebenen Werte vorzunehmen, bevor die Transition durchgeführt wird. Ein Beispiel wären eigene Plausibilitätschecks. Das ist eine sehr mächtige Funktion, die mit eigenen Plug-ins erweitert werden kann.
- *Post functions*
  Müssen nach der Durchführung der Transition noch weitere Aufgaben erledigt werden, ist das die richtige Stelle. Es wäre möglich, ein Log zu führen, das die Meilensteine manifestiert. Beteiligte Mitarbeiter können über den Fortschritt des Projekts informiert und Statuswerte aktualisiert werden.

Das war jetzt nur ein grober Überblick, hier bedarf es einer detaillierten Planung durch die Entwicklung. Wie es sich aus Sicht eines System-Administrators in Jira darstellt, zeigt Abbildung 3.
Geschäftsführer: Ausgezeichnet. Die Entwicklung wird sich darum kümmern! Mir

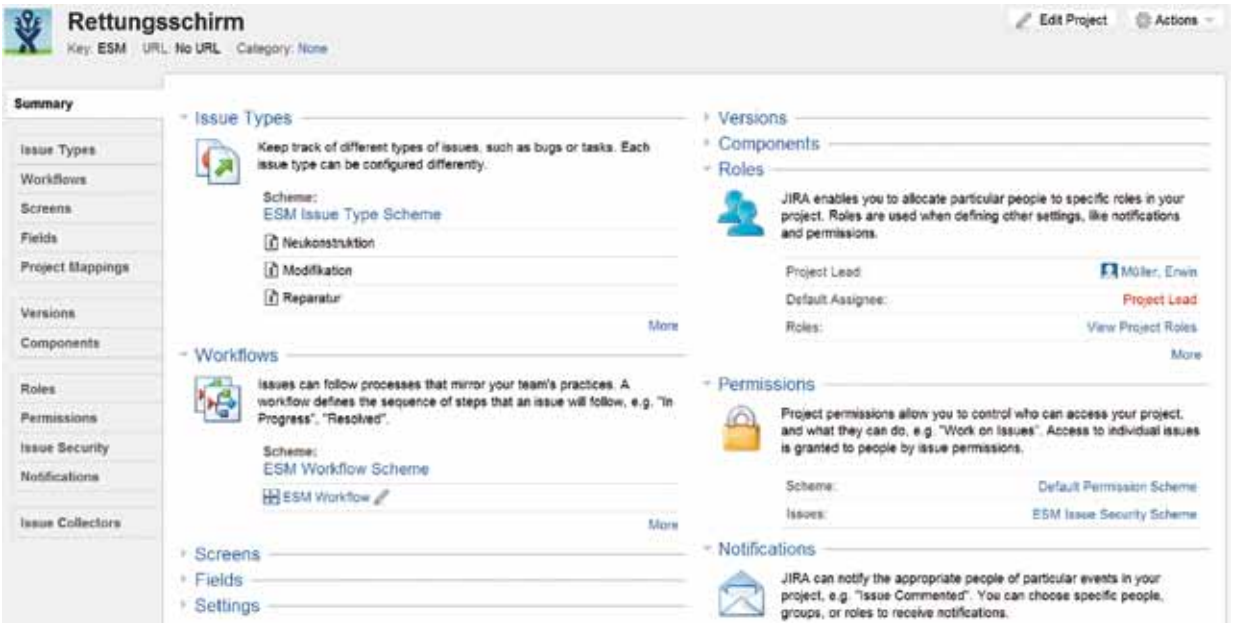


Abbildung 3: Projektkonfiguration aus Sicht des System-Administrators

fiel gerade auf, dass Sie den Begriff „Berechtigungen" erwähnten. Da wir mit externen Dienstleistern zusammenarbeiten, müssen wir verhindern, dass unsere internen Vorgänge nach draußen sichtbar sind. Geht das?

Müller: Die Sicherheit des Systems ist natürlich ein weites Feld! Wenn es um Vorgänge geht, bietet uns Jira eine sogenannte „Issue-Level-Security", die ich exemplarisch herausgreifen möchte (siehe Abbildung 4). Vorab möchte ich bemerken, dass man in Jira fast alles konfigurieren kann, was die Flexibilität auf der einen Seite sehr erhöht, aber mehrere zusätzliche Abstraktionsebenen schafft, die als Schemes realisiert sind. In diesem Fall benutzen wir ein Issue Security Scheme. Es gibt jedoch eine Vielzahl dieser Schemes, über die wir uns zu einem anderen Zeitpunkt unterhalten sollten, da sie die Grundlage des Verständnisses für das Gesamtsystem und eine seiner hervorragendsten Stärken bilden. Doch zurück zum Thema. Was ist nun konkret zu tun, um einen Vorgang abzusichern:

1. Wir legen uns ein neues Issue Security Scheme an und nennen es beispielsweise „ESM Issue Security Scheme".
2. Hier können wir nun zwei Level „intern" und „extern" generieren, wobei letzterer die Rechte der Dienstleister regeln wird.
3. Als Nächstes weisen wir jedem Level die entsprechenden Gruppen, Rollen oder Personen zu, die Zugriffsrechte besitzen sollen.
4. Bisher haben wir noch keine Zuordnung des Schemas zu unserem Projekt getroffen. Mit dieser Zuweisung ist es im letzten Schritt nun möglich, jedem neu angelegten Vorgang einen der beiden Security-Level zuzuweisen. Hierbei kann ein Default-Wert gesetzt werden, um sich unnötige Tipparbeit zu ersparen.

Ich möchte hinzufügen, dass ich für die ganzen Tests einige Überstunden machen musste.

Prokurist: Die waren aber nicht geplant!

Geschäftsführer: Dafür werden wir eine Regelung finden. Das bezog sich ja nun alles auf die Autorisierung. Was machen wir mit der Authentifizierung? Bekommt jeder einen weiteren Account?

Müller: Die Möglichkeit besteht, aber es geht auch einfacher. Wir können uns gegen unser eigenes LDAP anmelden, alles völlig transparent.

Entwicklungsleiter: Das ist ja alles schön und gut, aber die meiste Arbeit steckt doch in den Eingabemasken, über die wir bisher noch gar nicht gesprochen haben. Allein

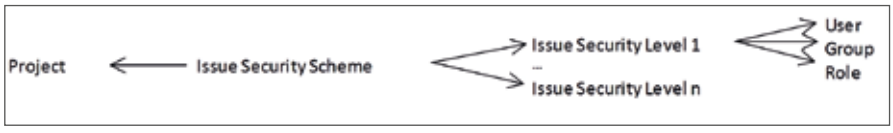


Abbildung 4: Issue-Level-Security

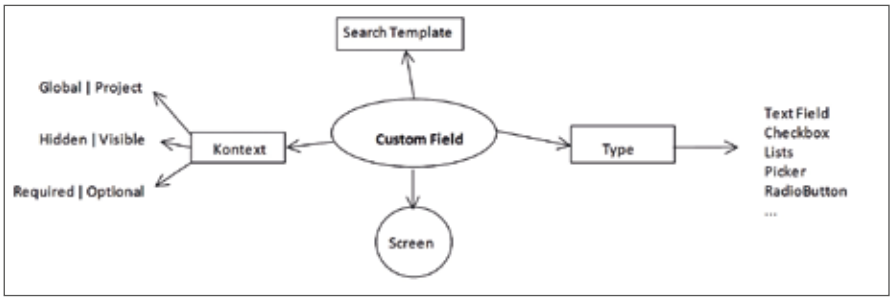


Abbildung 5: Custom Fields

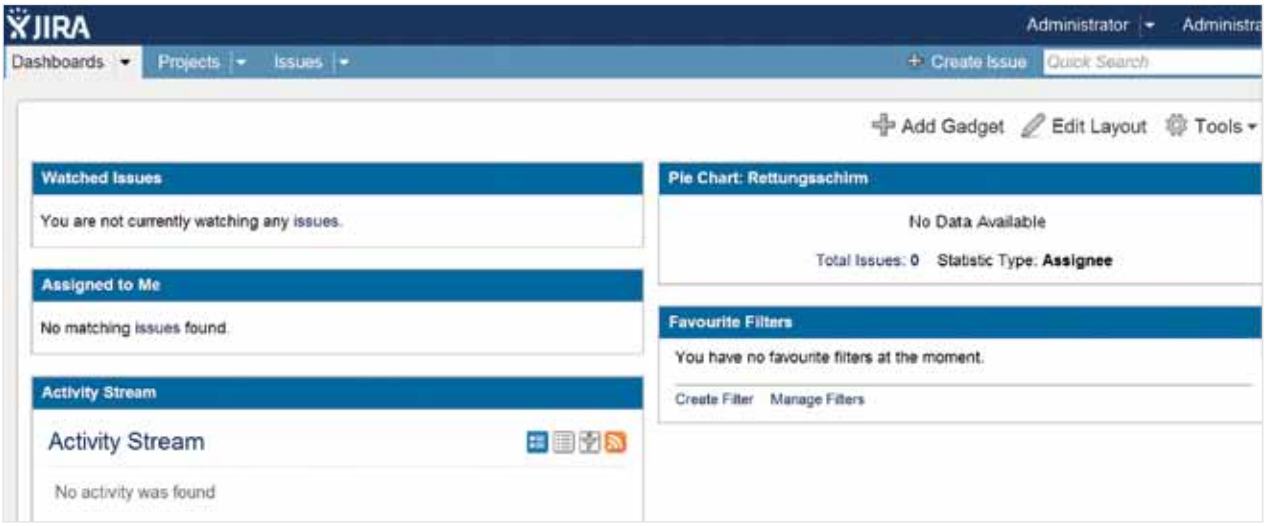


Abbildung 6: Dashboard des ESM-Projekts

hier die Layouts zu entwickeln, wird eine ganze Weile dauern.

Müller: Das ist tatsächlich ein größeres Thema und ich möchte hier auch gar nicht so sehr auf die Details eingehen. Grundsätzlich ist es wichtig, dass die fachliche Struktur bekannt sein muss. Das sollte aber durch die Entwicklung problemlos zu meistern sein. Hier die wichtigsten Punkte, die es zu beachten gilt:

- Es existieren vordefinierte Felder, die sofort benutzt und angezeigt werden können. Hierzu zählen beispielsweise:
  - Type (Issue)
  - Priority
  - Status (Workflow)
  - Label
- Benötigt man eigene Felder, können diese neu angelegt werden. Auch hier gibt es verschiedene Arten von Feldern:
  - (Free) Textfield
  - User-, Date-, Project-, Version-Picker
  - (Multi) Checkboxes
- Die Anordnung der Felder erfolgt auf Screens und die Reihenfolge ist beliebig anpassbar.
- Jedem Feld kann ein sogenanntes „Search Template" zugeordnet werden, um über Filter und Reports danach suchen zu können.
- Das Feld ist entweder einem spezifischen Projekt zugehörig oder von globaler Natur und bezieht sich auf einen oder mehrere Vorgänge. Hier kommen wir jedoch wieder in den Bereich der komplexen Zuordnungen.
- Letztendlich verfügt ein Feld über einen speziellen Kontext, in dem es mit weiteren Eigenschaften assoziiert wird:
  - sichtbar oder unsichtbar
  - optional oder required

Ach ja, so einen Screen kann man auch in verschiedene Reiter unterteilen und die Anzeige darauf abstimmen, ob ein Vorgang neu angelegt oder bearbeitet wird. Im letzten Fall sollen normalerweise nicht mehr alle Felder für eine Bearbeitung zur Verfügung stehen (siehe Abbildung 5).

Entwicklungsleiter: Ähem, ja. Ich muss mir das alles noch mal in Ruhe anschauen.

Geschäftsführer: Eine Frage habe ich noch. Für mich sind ja in erster Linie Auswertungen und Reports wichtig. Herr Müller, können Sie hierzu noch etwas sagen?

Müller: Ich habe mir so etwas gedacht und ein neues Dashboard angelegt. Ein Dashboard (siehe Abbildung 6) gibt einen Überblick über die wichtigsten Daten und Vorgänge eines Projekts. Es kann individuell angepasst oder auch komplett neu angelegt werden, so wie ich es getan habe. Die einzelnen Gadgets können ebenfalls separat konfiguriert angeordnet, hinzugefügt oder entfernt werden. Zur Auswahl stehen unter anderem alle wichtigen Filter, die auch anderen Anwendern, etwa den Kollegen in der Arbeitsgruppe, zur Verfügung gestellt werden können. Es gibt grafische Darstellungen und Übersichten zu den Vorgängen und Schnittstellen zu anderen Systemen wie Confluence, doch das ist ein anderes Thema.

Bezüglich Reports finden wir eine Menge vorkonfigurierter Berichte, wie beispielsweise Workload Pie Chart Report, User Workload Report, Version Time Tracking Report. Das ist nur eine kurze Aufzählung und der Fokus liegt hier auf dem Arbeitsaufwand für einen Anwender in einem Projekt, an einen Vorgang oder für eine spezielle Version.

Prokurist: Da wird der Betriebsrat ein Wörtchen mitzureden haben.

Müller: Vermutlich. Es gibt auch Schnittstellen zur Datenbank und Exportmöglichkeiten nach Excel und somit ist auch diese Aufgabe sehr flexibel lösbar.

Geschäftsführer: Ausgezeichnete Arbeit, Müller. Bevor wir uns in den Details verlieren, sollten wir einen Prototyp des Rettungsschirms erstellen. Sie wissen ja, Europa wartet auf uns!

Literatur/Web

- http://www.atlassian.com/software/jira/overview

Uwe Sauerbrei
uwe.sbr@emcon-partner.de

Uwe Sauerbrei arbeitete als Berater/Entwickler in den Branchen Telekommunikation, Automotive, Logistik und Luftfahrt. 2006 gründete er die „emmert CONSULTING + partner".

Fotos: IBM

Mainframe gestern: das System/360 Model 75

Modernisieren mit Java auf dem Mainframe

Marc Bauer und Tobias Leicher, IBM

Mainframe-Anwendungen sind über Jahrzehnte gewachsen und bedienen meist die kritischen Punkte in einem Unternehmen. Die gewachsene Komplexität dieser Anwendung erschwert Wartungsarbeiten und Änderungswünsche aufgrund neuer Anforderungen. Hier kann Java mit seiner Vielfalt an Frameworks eine Alternative bieten. Dieser Artikel beleuchtet, wie Java-Anwendungen in bestehende Mainframe-Anwendungen im Transaktionsmonitor CICS integriert werden können.

Der Mainframe ist heute noch in vielen Unternehmen eine zentrale Komponente in der IT-Infrastruktur. Im Jahre 1961 wurde das System/360 von einer kleinen Gruppe von IBM-Ingenieuren als das erste universelle Computer-System gebaut. Die damals entworfenen Architekturen gelten noch heute und bilden auch für moderne Computer-Systeme weiterhin eine Grundlage.

Heute ist der aktuelle Nachfolger des Systems/360, die zEnterprise, nach wie vor das Rückgrat in großen Unternehmen, vor allem in der Finanzwelt, wo die hohe Datensicherheit und die durchgehende Verfügbarkeit des Mainframes eine zentrale Rolle einnehmen. Als „Mainframe" verstehen wir in diesem Artikel Systeme auf Basis des Betriebssystems z/OS.

Der größte Teil der Last im Mainframe-Umfeld besteht vorwiegend aus Programmen in den klassischen Transaktionsmonitoren CICS und IMS, die in Sprachen wie COBOL, PL/I oder auch Assembler geschrieben werden. Zudem wird der Mainframe gern für die zentrale Datenhaltung eingesetzt, da er die Notwendigkeit einer verteilten Datenhaltung aufgrund seiner Cluster-Technologien und seiner auf I/O-Operationen optimierten Bauweise minimiert. Üblicherweise werden die Daten in DB2, IMS oder Adabas gehalten und sowohl von Mainframe-Anwendungen als auch von verteilten Anwendungen, die beispielsweise in Java geschrieben wurden, verwendet.

Mainframe-Systeme existieren seit den 1960er-Jahren und haben sich sukzessive weiterentwickelt. Mit zunehmenden Anforderungen wuchs allerdings auch die Komplexität eines Systems und somit müssen nicht selten mehrere Tausend Anwendungen und Module verwaltet, angepasst und verstanden werden. Diese Probleme sind charakteristisch für alle langlebigen Systeme und verlangen nach Modernisierungsmaßnahmen, um den Aufwand bei der Wartung und Weiterentwicklung zu minimieren.

Lange Zeit dachte man, man könne diese Probleme mit der Ablösung durch x86-Architekturen bewältigen, und reduzierte den Aufwand für die Ausbildung von neuem Mainframe-Personal. Heute erkennt man einen großen Personalmangel in der Altersklasse zwischen dreißig und fünfzig, da Ablöseprojekte nicht selten gescheitert sind. Das Wissen über die Systeme geht somit oft mit Kollegen, die in Rente gehen, verloren und junge Kollegen müssen sich an dieser Stelle in jahrelang gewachsene und oft auch nicht gut dokumentierte Anwendungen einarbeiten. Neben dieser Problematik ist auch das Wissen um die traditionellen Sprachen wie Cobol, PL/I und Assembler nur selten Teil der universitären Lehre.

Den notwendigen Modernisierungsmaßnahmen stehen Problemstellungen gegenüber wie komplexe, über die Jahrzehnte gewachsene Systeme und unzureichende Dokumentation über Funktionen und Implementierung einer Anwendung. Der Irrglaube, dass Hardware-Technologien und -Infrastrukturen beliebig austauschbar wären, ist trotz vieler realer Gegenbeweise in vielen Köpfen verankert.

Eine Möglichkeit, diesem Spannungsfeld Herr zu werden, könnte der Einsatz von Java auf dem Mainframe sein. Damit bestünde die Möglichkeit, die neuen Paradigmen der Anwendungsentwicklung mit denen des Mainframes zu vereinen und die notwendigen Modernisierungsmaßnahmen schrittweise einzuleiten. Da von einer einfachen JVM bis zu einem mächtigen JEE Application Server in Form des WebSphere Application Server for z/OS alles vorhanden ist, klingt der Einstieg zunächst einfach. Bestehende Java-Programme und JEE-Anwendungen sind ohne Weiteres lauffähig. Spannender hingegen ist die Frage der Integration von neuen Java-Anwendungen in die Bestandsanwendungen aus Cobol, PL/I oder Assembler, die im Folgenden am Beispiel der Integration von Java in CICS-Anwendungen aufgezeigt werden soll.

```
01 COM-REGION.
   05 JAVA-COMMAREA.
      10 CUST-ID      PIC X(9).
      10 AMOUNT       PIC X(9).
...
   EXEC CICS
      LINK PROGRAM('JAVAPROG')
      COMMAREA(JAVA-COMMAREA)
      LENGTH(18)
   END-EXEC.
...
```

Listing 1: Aufruf des Java-Programms JAVAPROG unter Cobol in CICS

```
public class CICSCommareaTest {
                public static void main(CommAreaHolder cah) {
                        String inputParameter = new String(cah.value);
                        String custID = inputParameter.substring(0, 7);
                        String amount = inputParameter.substring(8,16);
                        System.out.println(custID);
                        System.out.println(amount);
    }
}
```

Listing 2: Auswertung einer COMMAREA durch ein Java-Programm in CICS

```
public class CICSJavaCallOut {
        public static void main(CommAreaHolder cah) {
                try {
                        Program p = new Program();
                        p.setName("CBLPROG");

                        String data = new String("0123456789");
                        p.link(data.getBytes(), data.length());
                } catch(Exception ex) {
                        ex.printStackTrace();
                }
        }
}
```

Listing 3: Ein Java-Programm ruft das Cobol-Programm CBLPROG auf und übergibt eine Zeichenkette, die in einem realen Beispiel noch in eine Cobol-Struktur übersetzt werden müsste

CICS-Anwendungen und Java

Bereits Mitte der 1990er-Jahre setzte IBM einen neuen Kurs im Bereich des Mainframes mit Einführung der heute als „Unix-System-Services" bekannten Komponente, die eine Unix-Implementierung des z/OS anbietet. Diese ist Grundlage für den Einsatz von Java auf z/OS. Um dem Bedarf an eine moderne Entwicklungs- und Administrations-Umgebung nachzukommen, gab es weitere Anpassungen bei der Nutzeroberfläche für z/OS, die bis vor wenigen Jahren noch komplett Terminal-basiert waren. Heute gibt es diverse Eclipse-basierte Browser, die beispielsweise die Administration von z/OS und seiner Subsysteme wie IMS, CICS oder MQ deutlich vereinfachen. Somit wurden die Hürden für neues Mainframe-Personal deutlich verringert.

Um die Modernisierung weiter zu unterstützen, hat man die Integration von Java als Programmiersprache in die klassischen Transaktions-Monitore vorangetrieben. Besonders das aktuelle CICS-Release bietet umfangreiche Unterstützung für Java. Die Version 4.2 enthält neben der Ausführung einzelner Java-Programme auch ein OSGi-Framework, das auf Eclipse Equinox basiert. Ab dieser Version ist es also möglich, komplexe Java-Anwendungen in CICS analog zu den bestehenden Applikationen zu betreiben.

Für die Entwicklung stellt sich das Verhalten von Java auf dem Mainframe neutral dar. Es ist eine einfache JVM, wie auch unter allen anderen Plattformen. Zusätzlich können z/OS-spezifische Frameworks als OSGi-Bundles eingebunden werden. Diese Bibliotheken ermöglichen es, auf alle Funktionalitäten vom CICS zuzugreifen, beispielsweise für den Zugriff auf z/OS-spezifische Dateien oder zum Aufbau von Verbindungen zu DB2 oder MQ. Das API des CICS ist in einer Klassenbibliothek

komplett abgebildet und offen dokumentiert. Die Entwicklung kann beispielsweise mit einem kostenfreien SDK-Plug-in für Eclipse erfolgen.

Die Verwaltung der neuen Java-Anwendung ist wiederum komplett mit CICS-Mitteln abgedeckt und passt sich dadurch vollständig in die vorhandene CICS-Landschaft ein. Eine OSGi-Anwendung muss in ein CICS-Bundle paketiert werden. Dies entspricht der Erstellung einiger zusätzlicher „xml"-Dateien, mit deren Hilfe CICS das Deployment in die OSGi-Runtime steuert. Der OSGi-Lebenszyklus wird über diese CICS-Bundles gesteuert und angezeigt. Somit ist das Entwickeln mächtiger Java-Anwendungen für CICS problemlos möglich und wird von verschiedenen Werkzeugen unterstützt.

Um Java-Anwendungen nun in eine bestehende Anwendung zu integrieren, sind nur wenige Voraussetzungen notwendig. Das Java-Programm muss, wie beim Deployment in CICS üblich, in einer Programm-Definition festgelegt sein und eine Funktion, die die Auswertung der Übergabeparameter vornimmt, ist zu implementieren. CICS-Programme übergeben ihre Parameter in einem Daten-Konstrukt namens „COMMAREA". Der Java-Entwickler kann sich dies durchaus als „ByteArray" vorstellen. Zur Aufbereitung dieser COMM-AREA stehen in dem CICS-API entsprechende Funktionen bereit. Analog kann beim Return aus einem Java-Programm eine COMM-AREA mit den Rückgabewerten gefüllt werden. Diese CICS-Spezifika sind in wenigen Zeilen abbildbar und stehen dem großen Nutzen gegenüber, Java-Anwendungen im CICS einsetzen zu können, ohne die Cobol-Anwendungen verändern zu müssen. Listing 1 zeigt ein Beispiel dafür, wie ein Java-Programm aus Cobol in CICS aufgerufen wird. Die Auswertung der COMM-AREA unter Java ist in Listing 2 zu sehen.

Soll eine bereits implementierte Funktion aus einer der traditionellen Sprachen aufgerufen werden, kann dies mithilfe des CICS-API erfolgen. Es werden weder das Java-Native-Interface noch Connectoren benutzt. Beim Aufruf eines anderen Programms wird die Kontrolle an CICS übergeben, das für die entsprechende Kommunikation sorgt und die in der COMMAREA vorhandenen Daten an das aufgerufene Programm übergibt. Dies gilt auch für den Aufruf eines Java-Programms aus einer bestehenden Anwendung. Im Klartext heißt das, dass die Java-Integration in eine Bestandsanwendung unter CICS ohne Änderung einer Zeile des Bestandsprogramms möglich ist, indem der Programmname der Java-Anwendung einfach per „EXEC CICS LINK" aufgerufen wird (siehe Listing 3).

In der Tat wird dies in der Praxis auch so umgesetzt: Die Übersetzung der Parameter übernimmt das Java-Programm in einer Hilfsfunktion und die Bestandsanwendung bleibt unverändert. Somit ist auch die Integration von Java-Modulen in bestehende Anwendungen unter CICS ohne Weiteres möglich.

Fazit

Die beschriebene Technik lässt sich für verschiedene Szenarien einsetzen. Zum einen können neue Anforderungen in Java implementiert werden, wenn sie besonders großen Nutzen aus den Spracheigenschaften von Java ziehen können. Ein Beispiel wäre die Anforderung, PDFs zu generieren oder XML zu verarbeiten, wofür Java aufgrund vieler Frameworks ein idealer Kandidat ist. Zum anderen kann es für eine langfristige Java-Strategie sinnvoll sein, Module, die größeren Anpassungen unterlegen sind, in Java zu re-implementieren.

Tobias Leicher
tobias.leicher@de.ibm.com

Tobias Leicher arbeitet als IT Spezialist bei der IBM Deutschland GmbH. Er ist Spezialist für CICS und steht als technischer Berater für Kunden im Bereich CICS zur Verfügung, insbesondere im Java und Modernisierungsumfeld.

Marc Bauer
marc.bauer@de.ibm.com

Marc Bauer arbeitet im Software Service der IBM Deutschland GmbH. Er berät und unterstützt Kunden bei Modernisierungen von Mainframeanwendungen mit Hilfe von WebSphere und Java-Applikationen.

Mainframe heute: das System zEC12

Java und das Open Data Protocol

Klaus Rohe, Microsoft Deutschland GmbH

Das Open Data Protocol (http://www.odata.org/), kurz OData genannt, ist eine Spezifikation von Microsoft, die ein Protokoll für die plattformübergreifende Integration von Daten definiert.

OData basiert auf Representational State Transfer (REST) und den Formaten „JavaScript Object Notation" (JSON) und „Atom Publishing Protocol" (Atom Pub). Es unterstützt die Abfrage und Änderung von Daten mit standardisierten Internet-Technologien. Außer für das Microsoft .NET Framework (3.5SP1 und 4.x) gibt es SDKs für Java, JavaScript, Ruby, PHP und ObjectiveC. Microsoft Windows Azure erlaubt den Zugriff auf relationale und nicht-relationale Daten über OData. Auch IBM stellt bereits für die „WebSphere eXtreme Scale"-Plattform einen OData-kompatiblen Daten-Service bereit, und SAP wird mit dem NetWeaver Gateway ebenfalls OData unterstützen.

Der Artikel gibt eine Einführung in das Open Data Protocol und veranschaulicht anhand von existierenden Anwendungen und Beispielen das Einsatzspektrum dieser Technologie. Es wird weiterhin anhand zweier einfacher Beispiele gezeigt, wie man mit dem Open Source Framework odata4j (http://code.google.com/p/odata4j/) einen OData-Consumer und -Producer mit Java implementieren kann.

Daten-Integration über das Internet

Traditionelle Schnittstellen zur Integration von Daten in Applikationen sind im einfachsten Fall die I/O-Schnittstellen, die das Betriebssystem zur Verfügung stellt. Für relationale Daten haben sich Frameworks auf der Basis von ODBC, JDBC oder ADO.NET etabliert. Der Zugriff auf Daten, die in ERP, CRM oder anderen Geschäftsapplikationen liegen, geschieht über spezielle Adapter oder Middleware. Allen diesen Schnittstellen ist gemeinsam, dass sie nur innerhalb des Intranets brauchbar sind (siehe Abbildung 1). Erst einige der sogenannten „NoSQL-Datenbanken" wie Apache CouchDB bieten Schnittstellen auf der Basis von HTTP und sind damit auch über das Internet problemlos nutzbar.

Viele Daten, die für die Öffentlichkeit oder bestimmte Gruppen von Leuten beziehungsweise Unternehmen von großem (legalem) Interesse oder Nutzen wären, sind in Datensilos eingeschlossen, die nur mit erheblichem Aufwand in neue Applikationen – vor allem in die stark zunehmende Anzahl von Applikationen auf mobilen Endgeräten – integrierbar sind.

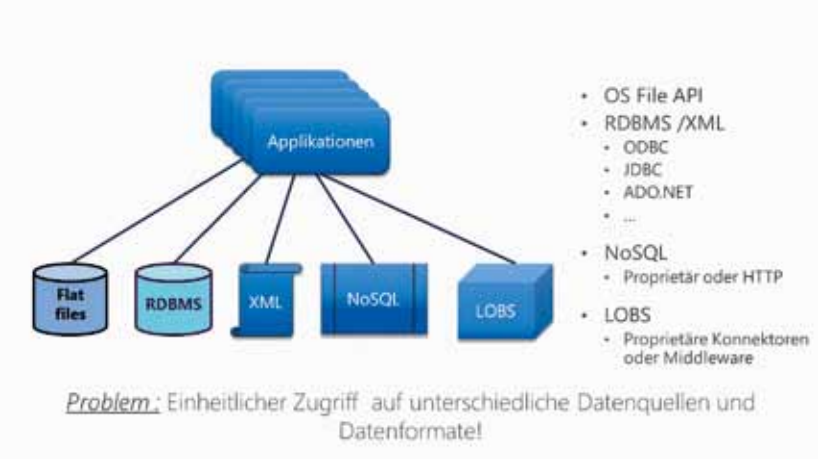


Abbildung 1: Traditionelle Datenintegrations-APIs

Dies drückt das folgende Zitat von der OData-Homepage (www.odata.org) sehr prägnant aus: „There is a vast amount of data available today and data is now being collected and stored at a rate never seen before. Much, if not most, of this data however is locked into specific applications or formats and difficult to access or to integrate into new uses."

Für eine Daten-Integration über das Internet sind, wie schon gesagt, die traditionellen Daten-Integrations-APIs ungeeignet. Außerdem bieten sie keinen einheitlichen Zugriff auf die unterschiedlichen Datenformate und -quellen.

Etabliert und bewährt hat sich im Internet das HTTP-Protokoll. HTTP ist einfach aufgebaut und damit auch gut zu implementieren, sodass es auf allen gängigen Applikationsentwicklungs-Plattformen durch entsprechende Bibliotheken und Frameworks unterstützt wird. Die Adressierung und Identifizierung von Ressourcen im Internet geschieht mithilfe von URIs, deren Syntax im RFC 3986 spezifiziert wurde.

Für den Datenzugriff (Daten-Integration) über das Internet wäre also ein Mechanismus ideal, der auf HTTP in Verbindung mit URIs aufbaut und für die Abfrage und Manipulation von Daten ähnlich flexibel ist wie die Sprache SQL für relationale Datenbanken. Die hier beschriebenen Anforderungen werden durch das Microsoft Open Data Protocol erfüllt.

Das Open Data Protocol (OData)

Das Open Data Protocol wurde als Microsoft-internes Projekt unter dem Code-Namen „Astoria" gestartet. Daraus entstanden die ADO.NET-Data-Services und mit .NET 4.0 die WCF-Data-Services. Es zeigte sich, dass die Architektur, das Protokoll und die Datenformate, die den WCF-Data-Services zugrunde liegen, sehr allgemein

Abbildung 3: Ein OData-Service-Dokument

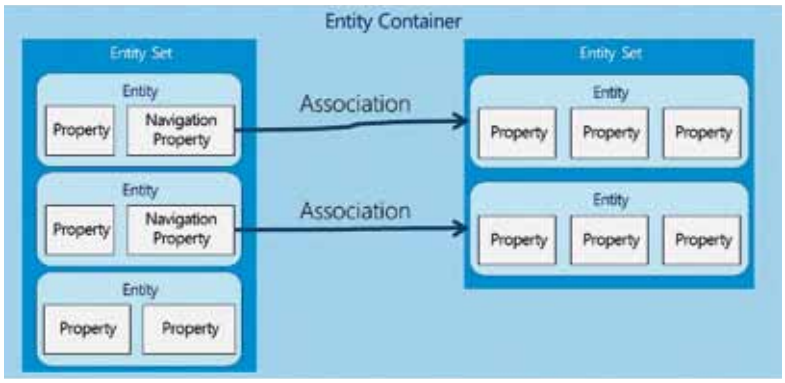


Abbildung 2: Das Entity Data Model

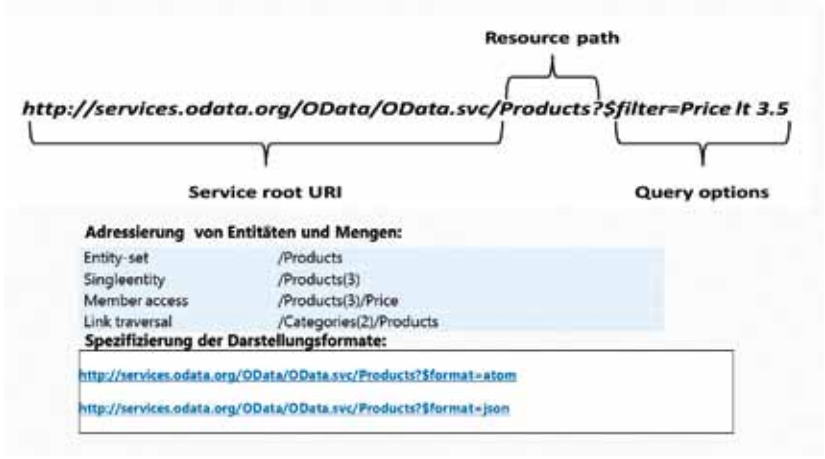


Abbildung 4: Beispiele für OData-URI-Konventionen

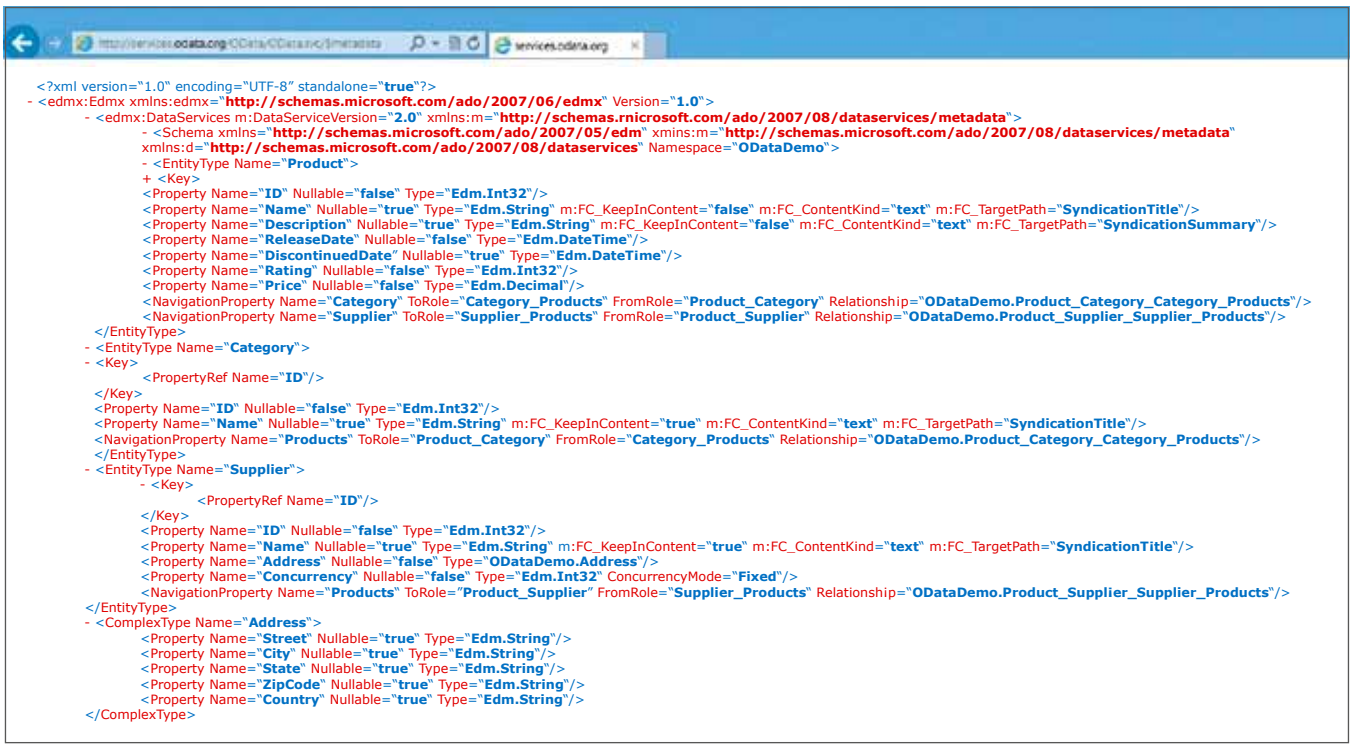
sind und in eine allgemeine Spezifikation gegossen werden konnten, die eben als „Open Data Protocol" bezeichnet wurde und die auf einer eigenen Webseite www.odata.org veröffentlicht wurde. Diese Spezifikation ist unter dem „Microsoft Open Specification Promise" freigegeben worden. OData ermöglicht es, unterschiedlichste Clients wie PCs, Macs, Windows Phone, iPhone, iPad, Android und Internet Browser über das Internet in entsprechende Backend-Systeme zu integrieren. Ebenso kann man Backend-Systeme selber über OData koppeln.

Die grundlegende Architektur von OData ist REST, das Protokoll ist HTTP, die Daten können in den Formaten „Atom Pub" oder „JSON" dargestellt sein. Die Metadaten werden durch einen XML-Dialekt beschrieben, der „Conceptual Schema Definition Language" (CSDL) heißt und der seinen Ursprung im ADO.NET-Entity-Framework hat [1].

Mit OData lassen sich also Schnittstellen zu Daten implementieren, denen man die Eigenschaft „RESTful" zuschreiben kann. Lesen und Ändern der Daten geschieht mit den bekannten HTTP-Verben. Durch Verwendung von HTTP werden auch Caching und Proxys ausgenutzt. Die Daten werden als Ressourcen betrachtet, die über eine einheitliche URL-Syntax adressierbar sind. Die Repräsentation der Daten kann im Atom-Pub- oder im JSON-Format durchgeführt werden.

Das Datenmodell, auf dem OData aufbaut, besteht aus Entitäten und den Assoziationen zwischen ihnen. Die Entitäten sind Ressourcen, die Assoziationen werden durch Links (siehe CSDL) dargestellt. Das Entity Data Model (EDM) hat sehr große Ähnlichkeit mit dem Entity-Relationship-Modell von Peter Chen, das er 1976 veröffentlicht hat (siehe: http://de.wikipedia.org/wiki/Entity-Relationship-Modell). Die Beschreibung dieses Datenmodells geschieht über die oben genannte Conceptual Schema Definition Language (CSDL). Diese wurde ursprünglich für das ADO.NET-Entity-Framework entwickelt, um das konzeptionelle Datenmodell zu beschreiben.

Das ADO.NET-Entity-Framework ist ein Werkzeug zur Objekt-relationalen Abbildung (ORM), die mit .NET 3.5 eingeführt wurde. Ein Beispiel für eine CSDL-Beschreibung wird bei der Vorstellung der OData-Metadaten gezeigt. Details zu CSDL



```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
- <edmx:Edmx xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx" Version="1.0">
    - <edmx:DataServices m:DataServiceVersion="2.0" xmlns:m="http://schemas.rnicrosoft.com/ado/2007/08/dataservices/metadata">
        - <Schema xmlns="http://schemas.microsoft.com/ado/2007/05/edm" xmins:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
        xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" Namespace="ODataDemo">
        - <EntityType Name="Product">
        + <Key>
        <Property Name="ID" Nullable="false" Type="Edm.Int32"/>
        <Property Name="Name" Nullable="true" Type="Edm.String" m:FC_KeepInContent="false" m:FC_ContentKind="text" m:FC_TargetPath="SyndicationTitle"/>
        <Property Name="Description" Nullable="true" Type="Edm.String" m:FC_KeepInContent="false" m:FC_ContentKind="text" m:FC_TargetPath="SyndicationSummary"/>
        <Property Name="ReleaseDate" Nullable="false" Type="Edm.DateTime"/>
        <Property Name="DiscontinuedDate" Nullable="true" Type="Edm.DateTime"/>
        <Property Name="Rating" Nullable="false" Type="Edm.Int32"/>
        <Property Name="Price" Nullable="false" Type="Edm.Decimal"/>
        <NavigationProperty Name="Category" ToRole="Category_Products" FromRole="Product_Category" Relationship="ODataDemo.Product_Category_Category_Products"/>
        <NavigationProperty Name="Supplier" ToRole="Supplier_Products" FromRole="Product_Supplier" Relationship="ODataDemo.Product_Supplier_Supplier_Products"/>
      </EntityType>
    - <EntityType Name="Category">
    - <Key>
        <PropertyRef Name="ID"/>
      </Key>
      <Property Name="ID" Nullable="false" Type="Edm.Int32"/>
      <Property Name="Name" Nullable="true" Type="Edm.String" m:FC_KeepInContent="true" m:FC_ContentKind="text" m:FC_TargetPath="SyndicationTitle"/>
      <NavigationProperty Name="Products" ToRole="Product_Category" FromRole="Category_Products" Relationship="ODataDemo.Product_Category_Category_Products"/>
      </EntityType>
    - <EntityType Name="Supplier">
        - <Key>
            <PropertyRef Name="ID"/>
        </Key>
        <Property Name="ID" Nullable="false" Type="Edm.Int32"/>
        <Property Name="Name" Nullable="true" Type="Edm.String" m:FC_KeepInContent="true" m:FC_ContentKind="text" m:FC_TargetPath="SyndicationTitle"/>
        <Property Name="Address" Nullable="false" Type="ODataDemo.Address"/>
        <Property Name="Concurrency" Nullable="false" Type="Edm.Int32" ConcurrencyMode="Fixed"/>
        <NavigationProperty Name="Products" ToRole="Product_Supplier" FromRole="Supplier_Products" Relationship="ODataDemo.Product_Supplier_Supplier_Products"/>
    </EntityType>
    - <ComplexType Name="Address">
        <Property Name="Street" Nullable="true" Type="Edm.String"/>
        <Property Name="City" Nullable="true" Type="Edm.String"/>
        <Property Name="State" Nullable="true" Type="Edm.String"/>
        <Property Name="ZipCode" Nullable="true" Type="Edm.String"/>
        <Property Name="Country" Nullable="true" Type="Edm.String"/>
    </ComplexType>
```

Abbildung 5: Ausschnitt aus dem Metadaten-Dokument von http://services.odata.org/OData/OData.svc/$metadata

findet man in der Dokumentation und in Büchern zum ADO.NET-Entity-Framework (etwa unter [1]) sowie in der Spezifikation des Open Data Protocol. Abbildung 2 zeigt den schematischen Aufbau des EDM, wie es in OData genutzt wird. Es gibt einen „Entity Container", der „Entity Sets" enthält, die aus einzelnen Entitäten bestehen. Die Entitäten enthalten „Properties" und eventuell „Navigation Properties", die sie in Beziehung zu anderen Entitäten setzen.

Zum Ausführen von Lese- und Änderungsoperationen auf den Ressourcen dienen „HTTP GET", „POST", „PUT" und „DELETE". MIT „GET" werden Ressourcen abgefragt, mit „POST" neue Ressourcen angelegt, mit „PUT" verändert und mit „DELETE gelöscht". In anderer Reihenfolge bezeichnet man dies auch als „CRUD-Operationen" (also Create Read Update Delete).

Das OData-Service-Dokument

Der Einstiegspunkt in einen OData-Service ist das sogenannte „Service-Dokument". Es stellt sozusagen die Wurzel dar, von der man zu den einzelnen Entitäten (Ressourcen) verzweigen kann. Das Service-Dokument ist im Atom-Pub-Format. Auf der OData-Website gibt es einen Test-Service, dessen Service-Dokument über die URL „http://services.odata.org/OData/OData.svc" angesprochen wird. Ruft man diese über den Internet Explorer auf, so wird das in Abbildung 3 dargestellte Service-Dokument zurückgegeben. Es besteht in diesem Fall aus einem Workspace mit dem Titel „Default", der drei Collections enthält: „Products", „Categories" und „Suppliers". Das Dokument entspricht der Atom-Pub-Spezifikation RFC 5023 (siehe http://bitworking.org/projects/atom/rfc5023.html).

Wer sich mit dem Microsoft SQL Server auskennt, wird hier die Beispiel-Datenbank „Northwind" erkennen, die unter anderem die Tabellen „Products", „Categories" und „Suppliers" enthält. Das Service-Dokument dient als Ausgangspunkt, um Informationen über die einzelnen „Products", „Categories" etc. zu bekommen. Dies geschieht über Uniform Resource Identifier (URI), deren syntaktischer Aufbau auf der nächsten Folie anhand von Beispielen erläutert wird. Die gesamte Northwind-Datenbank ist über den URI „http://services.odata.org/Northwind/Northwind.svc/" erreichbar.

URI-Konventionen in OData

Damit man bestimmte Ressourcen über OData ansprechen kann, sind Konventionen für die Syntax des URI einzuhalten. Ein URI für einen OData-Service besteht aus drei Teilen (siehe Abbildung 4):

1. Dem „Service Root URI", also dem Einstiegspunkt in den Service, hier „http://services.odata.org/OData/OData.svc/"
2. Dem „Resource path", hier „Products"
3. Den „Query options", in diesem Beispiel ein Filter für alle Produkte mit einem Preis kleiner als 3,50 Dollar

Beispiele für den Zugriff auf einzelne Entitäten und Mengen von Entitäten auf einzelnen Feldern sowie für die Navigation entlang eines Links sind in der Tabelle in der Mitte der Abbildung 4 angegeben:

- „http://services.odata.org/OData/OData.svc/Products" greift auf alle Produkte zu
- „http://services.odata.org/OData/OData.svc/Products(3)" greift auf das Produkt mit dem Schlüssel 3 zu
- „http://services.odata.org/OData/OData.svc/Products(3)/Price" greift auf den Preis des Produktes mit Schlüssel 3 zu
- „http://services.odata.org/OData/OData.svc/Categories(2)/Products" zeigt alle Produkte an, die zur Kategorie mit dem Schlüssel 2 gehören

NetflixODataClt.java

```
1 import java. util. List;
2 import javax.ws.rs.ext.RuntimeDelegate;
3 import org.odata4j.consumer.ODataConsumer;
4 import org.odata4j.core.OEntity;
5 import org.odata4j.core.OProperty;
6 import org.odata4j.jersey.consumer.ODataJerseyConsumer;
7 import org.odata4j.jersey.consumer.ODataJerseyConsumer.Builder;
8
9 public class NetflixODataClt {
10          static {
11                    // ensure that the correct JAX-RS implementation is loaded
12                    RuntimeDelegate runtimeDelegate = new com.sun.jersey.server.impl.provider.RuntimeDelegateImpl();
13                    RuntimeDelegate.setInstance(runtimeDelegate);
14          }
15
16          public static void main(String[] args)  {
17                    Builder builder = ODataJerseyConsumer.newBuilder("http://odata.netflix.com/Catalog/");
18                    ODataConsumer ode = builder.build();
19                    lookUpMoovies (ode);
20          }
21
22          private static void lookUpMoovies(ODataConsumer ode)   {
23                    // locate the netflix id for Morqan Spurlock
24                    int morganSpurlockId = ode.getEntities("People").filter("substringof('Spurlock', Name)").execute().first().getProperty("Id", Integer.class).getValue();
25                    // lookup and print all titles he's acted in
26                    List<OEntity> titlesActedIn = odc.getEntities("People").nav(morganSpurlockId, "TitlesActedIn").execute().toList();
27                    for (OEntity title : titlesActedIn)   {
28                              for (OProperty<?> p : title.getProperties())   {
29                                        System.out.println(String.format("%s: %s", p.getName(), p.getValue()));
30                              }
31                              System.out.println("==- =\n");
32                    }
33                    System.out.println("count: " + titlesActedIn.size());
34          }
35 }
```

Abbildung 6: Quellcode für den Netflix OData Client

OData unterstützt nicht nur das XML-Format „Atom Pub", sondern auch noch JavaScript Object Notation (JSON); „Atom Pub" ist voreingestellt. Unterstützt ein OData-Service auch JSON, so muss man dies über die Formatoption in dem URI spezifizieren, wie in den Beispielen zu sehen ist, wenn man die Daten in diesem Format konsumieren möchte.

Optional kann ein OData-Service ein Metadaten-Dokument zur Verfügung stellen. Die Beschreibung der Daten geschieht dort über die XML-Sprache CSDL (siehe oben). Die Abfrage der Metadaten erfolgt durch Anhängen von „$metadata" an den URI der Service-Root. Als Beispiel für ein Metadaten-Dokument ist der Service „http://services.odata.org/OData/OData.svc/" in Abbildung 5 gezeigt. Das Metadaten-Dokument wird beispielsweise von Entwicklungswerkzeugen zur Generierung von Proxy-Klassen für den vereinfachten Zugriff auf OData-Services benutzt, etwa vom Java Framework „odata4j" (siehe [5]).

Standardisierung von OData

Am 24. Mai 2012 hat Microsoft angekündigt (siehe http://www.microsoft.com/en-us/news/press/2012/may12/05-24ODataPR.aspx), zusammen mit IBM, SAP, Citrix, Progress Software und WSO2 (http://wso2.com/) bei der „Organisation for the Advancement of Structured Information Standards" (OASIS) ein technisches Komitee zur Standardisierung von OData zu gründen. Das Interesse an der Standardisierung rührt daher, dass IBM und SAP Produkte haben, die auf OData basieren, und dass in den USA und Kanada viele öffentliche Einrichtungen OData-Schnittstellen zu Daten bieten, die der Allgemeinheit zur Verfügung gestellt werden sollen. Dies gilt auch für Daten, die die UNESCO über den Azure-Marktplatz anbietet (https://datamarket.azure.com/).

Software Development Kits für OData-Anwendungen

Unter der URL, die in [4] genannt ist, findet man alle aktuell verfügbaren OData-SDKs. Für alle gängigen mobilen Plattformen wie iPad, iPhone, Windows Phone 7 und Android gibt es SDKs, um einfach OData-Services in entsprechende Applikationen zu integrieren. Zur Implementierung von OData-Services gibt es SDKs für .NET 3.5/4.x, für Java, für PHP und Ruby. Damit ist auch auf der Server-Seite ein breites Spektrum an Applikations-Plattformen abgedeckt. Für solche Applikations-Plattformen, für die es kein dediziertes OData-SDK gibt, stellt die Integration von OData-Services kein allzu großes Problem dar, sofern entsprechende HTPP- und XML- beziehungsweise JSON-Frameworks vorhanden sind.

Das Java-Framework „odata4j"

OData4J ist ein Framework zur Implementierung von OData-Consumern und -Producern in Java. Es basiert auf „Jersey" (https://jersey.dev.java.net/), nutzt „Joda Time" (http://joda-time.sourceforge.net/) und unterstützt laut Dokumentation die Entwicklung von OData-Clients auf Android sowie die Implementierung von OData-Anbietern auf der Google App Engine. OData4J ist aktuell in der Version 0.6 verfügbar. Es kann als ZIP-Archiv unter [5] heruntergeladen werden.

Entpackt man das ZIP-Archiv, so entsteht ein Verzeichnis mit dem Namen „odata4j-archive-x.y" (aktuell ist x.y = 0.6) mit zwei Unterverzeichnissen „bundles" und „odata4j". Alle benötigten jar-Dateien sind in den Unterverzeichnissen enthalten. Die Dokumentation befindet sich im Unterverzeichnis „javadoc" von „odata4j". Das Verzeichnis „bundles" enthält die jar-Files „odata4j-bundle-0.6.jar" und „odata4j-clientbundle-0.6.jar". Will man nur OData-

Clients entwickeln, so reicht es, „odata4j-clientbundle-0.6.jar“ in den „CLASSPATH“ aufzunehmen. Das Einfügen von „odata4j-bundle-0.6.jar“ in den „CLASSPATH“ ist hinreichend für die Implementierung von Clients und Anbietern.

Die hier beschriebenen Beispiele und Tests wurden mit dem Java SE Development Kit Version 1.7.0.07 und dem Eclipse Juno Release durchgeführt. Es sind Modifikationen der Beispiele „NetflixConsumerExample“ und „InMemoryProducerExample“, die man auf der odata4j-Website (siehe [5]) findet. Das erste Beispiel ist ein OData-Consumer, der einen OData-Service von Netflix (www.netflix.com) nutzt. Abbildung 6 zeigt den vollständigen Quellcode für dieses Beispiel.

In Zeile 17 und 18 wird der OData-Client für Netflix erzeugt, indem man den „Root URI“ des Service an die Factory-Methode „ODataJerseyConsumer.newBuilder(...)“ übergibt. Daraus wird der eigentliche OData-Consumer in Form einer Instanz der Klasse „OdataConsumer“ mit der „build()“-Methode der Klasse „Build“ erzeugt. Die Klasse „OdataConsumer“ enthält alle Methoden, die zum lesenden und schreibenden Zugriff auf einen OData-Service notwendig sind. In dem Quellcode in Abbildung 6 wird in „lookUpMoovies(...)“ (Zeile 22 – 34) die Methode „getEntities(...)“ der Klasse „OdataConsumer“ benutzt, um alle Filme des Regisseurs Morgan Spurlock aus dem Netflix-Katalog aufzulisten.

Das zweite Beispiel ist ein OData-Producer, der als lokaler Java-Prozess gestartet werden kann. Er stellt Systemdaten wie die Anzahl der Java Threads, die auf der JVM gerade ausgeführt werden, die Liste der Umgebungsvariablen in seinem Ausführungskontext, die Liste der Dateien in seinem Startverzeichnis etc. als OData-Service zur Verfügung. Der OData-Producer „OdataInMemProducer“ benutzt die Klassen „org.odata4j.jersey.producer.server.JerseyServer“ und „org.odata4j.producer.server.OdataServer“ zum Implementieren des lokalen Servers.

Die Logik zum Starten des lokalen Servers wurde in der Klasse „JerseyServerRuntimeFacade“ gekapselt, die eine Methode „launchODataServer(String baseUri)“ besitzt, um den Server zu starten. Der OData-Service ist dann unter dem als Methoden-Parameter „baseUri“ übergebenen URI erreichbar. Die beiden vorgestellten Beispiele können als Eclipse-Projekte in einer ZIP-Datei verpackt per E-Mail vom Autor bezogen werden.

OData4J macht insgesamt einen brauchbaren Eindruck. Die Dokumentation lässt allerdings noch viele Wünsche offen. Man muss sich aktuell die Anwendung anhand der Beispiele erarbeiten, was machbar ist, da diese sehr übersichtlich sind. Es lohnt sich also, das OData4j-Projekt im Auge zu behalten, wenn man anstrebt, OData-Producer und -Consumer mit Java zu realisieren.

Eine weitere Möglichkeit, OData-Applikationen mit Java zu entwickeln, hat man im Zusammenhang mit dem SAP NetWeaver Gateway. Dieses Gateway ist auf der Grundlage von OData implementiert worden und bietet ein Eclipse-Plug-in für die Entwicklung von OData-Consumern (Clients) an. Dieses Werkzeug kann man sowohl für die Java Standard Edition als auch für Java unter Android nutzen. Details findet man unter der in [6] genannten URL.

Fazit

Das Open Data Protocol nutzt im Internet etablierte und ausgereifte Architekturen, Protokolle, Darstellungsformate und Software-Technologien. Aufgrund dieser Eigenschaften gibt es eine große Anzahl von SDKs für die verschiedenen Applikations-Plattformen. OData hat dadurch exzellente Interoperabilitäts-Eigenschaften und eignet sich damit hervorragend zur Datenintegration über das Internet. Das Interesse von kommerziellen und öffentlichen Anbietern an OData ist groß, sodass es das Potenzial hat, das „ODBC“ des WWW zu werden.

Weiterführende Informationen

[1] ADO.NET Entity Framework: http://msdn.microsoft.com/en-US/data/ef

[2] David Chappel, Introduciong Odata, data acess for the web, the cloud, mobile devices, and more: http://msdn.microsoft.com/en-us/data/hh237663.aspx

[3] Holger Schwichtenberg, Datenpumpe, Datenbasierte Webservices mit dem Open Data Protocol, IX Magazin für professionelle Informationstechnik, 10/2012, Seite 90 - 96

[4] Verfügbare OData SDKs: http://www.odata.org/libraries

[5] odata4j: http://code.google.com/p/odata4j/

[6] SAP NetWeaver Gateway plug-in for Eclipse: http://www.sdn.sap.com/irj/scn/downloads?rid=/webcontent/uuid/b09d414f-f227-2f10-bdbf-ba31c844b432

Klaus Rohe
klaus.rohe@microsoft.com

Klaus Rohe arbeitet bei der Microsoft Deutschland GmbH in der Developer Platform & Strategy Group und berät Software-Hersteller bei der Einführung und Anwendung von neuen Microsoft-Technologien aus dem Bereich Software-Entwicklung.

„Ich bin sehr zufrieden mit der Sprache …“

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

Wie bist du zur Java Student User Group gekommen?
Dorn: Zu Anfang des dritten Semesters auf der Uni wurde die JSUG von den damaligen Mitgliedern in einer Vorlesung kurz vorgestellt. Anfangs war ich skeptisch, bin dann aber doch zum ersten Treffen gegangen.

Wie ist die Java Student User Group organisiert?
Dorn: Wir sind ein eingetragener Verein im Wiener Vereinsregister. Es gibt ein sogenanntes „Core Team“, das die komplette Organisation übernimmt. Wir arbeiten auch viel virtuell per Google-Group und Doodle mit unseren Mitgliedern zusammen, um gemeinsam optimale Entscheidungen zu treffen.

Was zeichnet die Java Student User Group aus?
Dorn: Zuerst einmal haben wir den Rückhalt der TU Wien. Dort bekommen wir Räume zur Verfügung gestellt und kommen auch leichter an neue Mitglieder ran. Viele unserer Mitglieder sind (zum Glück) auch noch nicht so stark in den Arbeitsprozess der Wirtschaft eingespannt, sodass sie die Zeit haben, neue und coole Technologien auszuprobieren – die sie dann bei uns auch vorstellen. Das Wichtigste ist aber vermutlich, dass wir ein „Verein von Studenten für Studenten“ sind. Wir heißen aber auch Java-/JVM-Interessierte, die keine Studierenden sind, herzlich willkommen.

Wie viele Veranstaltungen gibt es pro Jahr?
Dorn: Wir folgen dem Semester-Zyklus der Uni, sprich in den Ferien gibt es keine Treffen, dafür während des Semesters etwa alle drei Wochen. Je nach Einteilung variiert das zwischen zehn und zwölf Treffen pro Jahr.

Was bedeutet Java für dich?
Dorn: Java ist für mich mittlerweile ein Allround-Tool. Man kann nahezu alles damit machen – Web-Anwendungen, Rich-Clients und natürlich den „Mobile“-Sektor. Das ausgeprägte Ökosystem mit vielen vorhandenen Bibliotheken und guten Entwicklungsumgebungen machen die Arbeit angenehm.

Welchen Stellenwert besitzt die Java-Community für dich?
Dorn: Hmm … eine gute Frage. Ich habe verschiedenste Kontakte quer durch die Java-Community. Viele dieser Kontakte treffe ich auch im „realen“ Leben und zähle sie zu meinen Freunden und guten Bekannten. Nachdem die zahlreichen Bibliotheken und Tools ohne die Community nicht möglich wären, ist sie für mich das Wichtigste im Java-Ökosystem.

Was hast du bei der Übernahme von Sun durch Oracle empfunden?
Dorn: Anfangs war ich sehr skeptisch, dann aber auch hoffnungsvoll. Sun war in den letzten Monaten sehr angeschlagen und nahezu handlungsunfähig. Die Übernahme durch Oracle versprach frisches Kapital und das Ende des Stillstands. Es war dann teilweise etwas frustrierend, wie der Prozess ewig von verschiedensten Behörden verzögert wurde. Als die Übernahme dann endlich abgeschlossen war, hat Oracle in meinen Augen leider sehr viele Fehler gemacht (Jenkins, OpenOffice, JCP etc.) Ich hoffe, sie haben wenigstens etwas daraus gelernt.

Wie sollte sich Java weiterentwickeln?
Dorn: Ich bin prinzipiell sehr zufrieden mit der Sprache, so wie sie aktuell ist. Die neuen Sprach-Features von Java 7 finde ich zwar nett, lebe aktuell aber immer noch gut mit der 1.6er Version. Was ich vermisse, sind Features aus der funktionalen Programmierung, dafür gibt es aber ja zum Glück andere Byte-Code-kompatible Sprachen auf der JVM.

Wie sollte Oracle deiner Meinung nach mit Java umgehen?
Dorn: Oracle sollte die eigenen, profitorientierten Manager möglichst weit weg entfernen und jemanden engagieren, der wirklich den Spirit der Community versteht. Die aktuelle Führungsetage hat vieles in der Community mit sturem Kapitalismusdenken kaputt gemacht und viel gutes Personal weggetrieben.

Wie sollte sich die Community gegenüber Oracle verhalten?
Dorn: Wir sollten uns immer vor Augen halten, dass Oracle ein großer, alter und klobiger Konzern ist, der sich sehr schwer mit Veränderungen tut. Es ist, wie wenn man in einen neuen Betrieb einsteigt, Schwächen erkennt und gleich damit beginnt, zig Änderungsvorschläge zu machen – man wird sehr viel Widerstand ernten. Wir als Java-Community müssen versuchen zu verstehen, wie Oracle tickt. Wenn wir eine Änderung wollen, müssen wir ein kleines Paket daraus schnüren und eine hübsche Verpackung dazu basteln, bei der Oracle gleich den Mehrwert (in Form von $$$-Zeichen) sieht.

Dominik Dorn
dominik.dorn@jsug.at

Dominik Dorn studiert aktuell Software & Information Engineering an der Technischen Universität Wien. Neben dem Studium arbeitet er an verschiedenen Unternehmen im Web-Umfeld mit. Seit 2010 führt er die Java Student User Group als Obmann.

Foto: Fotolia

Garbage Collection im Java-Umfeld

Mathias Dolag, iSYS Software GmbH; Prof. Dr. Peter Mandl und Christoph Pohl, Hochschule München

Ein oft wenig beachteter, jedoch essentieller Bestandteil der Java Virtual Machine (JVM) ist der Garbage Collector. Er ist für das Entfernen nicht mehr benötigter Objekte aus dem Heap-Speicher (Heap) zuständig.

Der Garbage Collector (GC) nimmt dem Entwickler eine fehleranfällige Arbeit ab, sodass er sich auf sein eigentliches Spezialgebiet konzentrieren kann. Doch wie arbeitet der Garbage Collector? Welche Möglichkeiten haben der Entwickler beziehungsweise Administrator, um Einfluss auf ihn zu nehmen? Diesen Fragen wurde im Rahmen einer Untersuchung an der Hochschule München in Kooperation mit der iSYS Software GmbH nachgegangen. Der Artikel stellt die Ergebnisse vor. Im ersten Teil wird auf die Funktionsweise ausgewählter Garbage-Collector-Algorithmen der HotSpot JVM eingegangen. Der zweite Teil beschreibt die grundlegenden Aspekte des Garbage-Collection-Tunings. Zum Abschluss werden Ergebnisse durchgeführter Leistungs-Messungen vorgestellt. Alle dargestellten Informationen und Ergebnisse beziehen sich – sofern nicht explizit anders ausgewiesen – auf die HotSpot JVM Version 6.

Allgemeines

Alle Objekte werden bei Java vom Allokator im Heap erzeugt. Dessen Größe steuert man beim Starten der JVM durch die Parameter „-xms“ (initiale Heap-Größe) und „-xmx“ (maximale Heap-Größe). Sind diese mit identischen Werten gesetzt, wird bereits beim Start der JVM die gesamte Kapazität reserviert, andernfalls bei Bedarf dynamisch vergrößert oder verkleinert. Es ist festzuhalten, dass es bei Java nicht möglich ist, direkt Heap-Adressen anzusprechen oder Objekte an einer fest definierten Speicheradresse anzulegen. Daher kann man ein Objekt im Heap nur erreichen, falls darauf eine Referenz existiert. Indirekt muss es dabei von einer „Root-Referenz“ aus erreichbar sein. Darunter versteht man Referenzen, die beispielsweise von den einzelnen Applikations-Threads in deren Thread-Stacks gehalten werden. Jede Root-Referenz dient dabei als Einstiegspunkt in den „Objektgraphen“, welcher das Gerüst aller erreichbaren Objekte darstellt. Die Summe aller Root-Referenzen nennt man ein „RootSet“.

Durch Verfolgen aller Objekt-Referenzen (ausgehend von den Root-Referenzen) lassen sich vom Garbage Collector beziehungsweise der Applikation aus alle lebendigen Objekte erreichen. Alle nicht erreichbaren Objekte stellen im Sinne der Applikation „Müll“ (engl. Garbage) dar und können vom Garbage Collector entfernt werden, sodass deren Speicherbereich zur Allokation neuer Objekte wieder zur Verfügung steht. Man spricht bei solchen Objekten auch von „toten“ Objekten. Würden die, nicht mehr von einer Root-Referenz aus erreichbaren Objekte, nicht aus dem Heap entfernt, könnte dies irgendwann zu einer „OutOfMemory-Exception“ führen, da sich keine neuen Objekte mehr im Heap allokieren lassen. Vorausgesetzt ist hierbei eine aktive Verwendung der Applikation.

Mark-and-Sweep

Nachfolgend ist die Vorgehensweise von „Mark-and-Sweep“, dem einfachsten GC-Algorithmus, beschrieben. Wie dem Namen zu entnehmen ist, läuft dieser in zwei Phasen ab. Phase 1 geht den Objekt-Graphen durch und markiert alle erreichten Objekte. Dies kann auch parallel durch mehrere Threads durchgeführt werden. Jeder Thread erhält dabei eine Root-Referenz, deren Referenzen darüber verfolgt

werden. Hierdurch verkürzt sich die Markierungs-Zeit.

In Phase 2 werden anschließend alle nicht markierten Objekte aus dem Heap entfernt und die freigegebenen Speicherbereiche einer Liste des Allokators (FreeList) angehängt. In dieser Liste verwaltet der Allokator alle freien Speicher-Bereiche sowie deren Größe. Für die Allokation eines neuen Objekts wird dann ein passender Speicherbereich in der FreeList gesucht.

Mark-and-Sweep fällt in die Kategorie der blockierenden Algorithmen, da während der Markierung der/die GC-Thread(s) uneingeschränkten Zugriff auf den Heap benötig(t)en. Die eigentlichen Applikations-Threads werden dabei unterbrochen. Man spricht bei der entstehenden Pausenzeit auch von einer „Stop-the-World-Pause".

Durch das reine Entfernen von Objekten wird der Heap auf Dauer unterschiedlich stark fragmentiert. Man nennt dies folglich „Heap-Fragmentierung". Durch diese wird die Allokation neuer Objekte erschwert und verlangsamt, da für jedes neue Objekt eine passende Position im Heap gefunden werden muss. Daher stellt zunehmende Heap-Fragmentierung ein Problem dar. Um diese zu vermeiden beziehungsweise möglichst gering zu halten, wurden komplexere Algorithmen und Konzepte entwickelt, die nachfolgend besprochen sind.

Generational Garbage Collection

Alle heutigen GC-Algorithmen basieren auf Generationen beziehungsweise auf Regionen. Man spricht daher von der „Generational Garbage Collection". Dieser Ansatz resultiert aus der Tatsache, dass Objekte unterschiedliche Lebensdauern haben. Es existieren in einer Java-Applikation dabei meist sehr viele kurzlebige Objekte (etwa Objekte, die nur innerhalb einer Methode oder Expression im Zugriff sind, wie Iteratoren oder StringBuilder), weniger mittellang lebende (etwa Session-spezifische Variablen) und noch weniger langlebige Objekte (wie Singletons oder Datenbank-Verbindungen). Dies wurde nach [2, Seite 6] durch die „weak generational hypothesis" wissenschaftlich belegt. Diese Hypothese ist dabei unabhängig von einer bestimmten Programmiersprache zu sehen. Ebenfalls wurde festgestellt, dass ältere Objekte nur selten Referenzen auf jüngere Objekte halten.

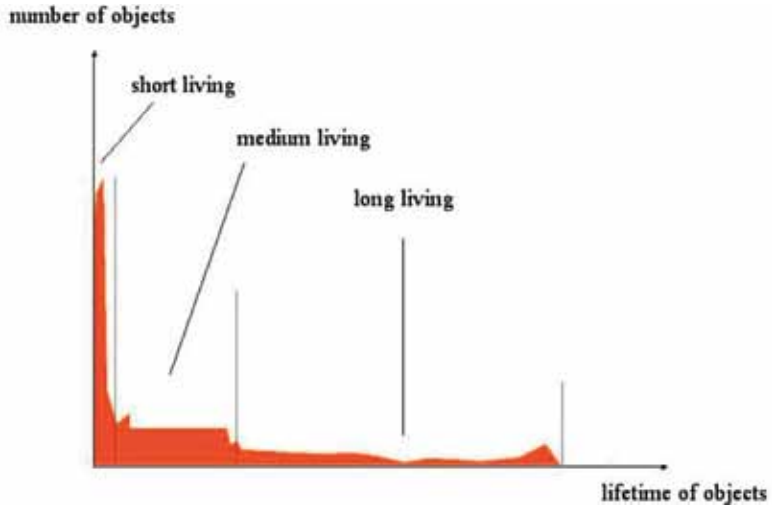


Abbildung 1: Typische Objekt-Population einer Java-Applikation (Quelle: [1, S. 151])

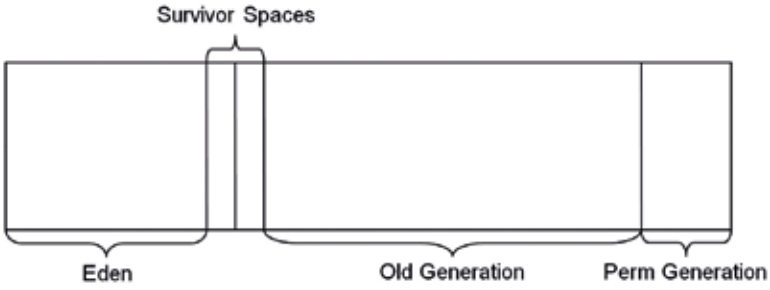


Abbildung 2: JVM-Heap-Aufteilung (Quelle: in Anlehnung an [1, S. 153])

Abbildung 1 zeigt die typische Objektpopulation einer Java-Applikation mit einer Verteilung der Objekte nach deren Lebensdauer. Daraus lässt sich ableiten, dass es nicht vorteilhaft ist, bei einer GC jeweils den gesamten Heap nach nicht mehr referenzierten Objekten zu durchsuchen. Dies würde mit zunehmender Heap-Größe zu immer längeren GC-Pausenzeiten führen und die eigentliche Applikation dadurch massiv behindern.

Der Heap wird in der HotSpot JVM daher in Generationen eingeteilt. Junge beziehungsweise neu allokierte Objekte sind in der „Young Generation" gespeichert, ältere Objekte in der „Old Generation". Die Young Generation ist dabei noch feiner, in „Eden" und zwei gleich große „Survivor Spaces" namens „From-Space" und „To-Space" unterteilt. Diese spielen eine entscheidende Rolle zur Steigerung der GC-Performance (siehe hierzu Kapitel „Mark-and-Copy"). Daneben existiert noch die „Perm(anent) Generation", die unter anderem die geladenen Class-Files beinhaltet. Sie spielt für die GC jedoch nur eine untergeordnete Rolle, auf die an dieser Stelle nicht weiter eingegangen wird. Abbildung 2 zeigt die Aufteilung des Heaps in der HotSpot JVM.

Alle neuen Objekte werden in Eden allokiert, wohingegen die Survivor-Spaces nur Objekte beinhalten, die bereits mindestens einen GC-Durchlauf überlebt haben. Eine GC wird ausgelöst, sobald Eden keine neuen Objekte mehr aufnehmen kann beziehungsweise sobald die Old Generation einen festgelegten Füllgrad erreicht hat. Dieser kann entweder dynamisch durch die JVM bestimmt oder manuell durch einen JVM-Parameter festgelegt sein.

Eine GC kann nun entweder nur die Young Generation (Minor Collection) oder die Young- und die Old Generation (Major Collection) bereinigen. Major Collections sind während der Laufzeit einer Applikation deutlich seltener und bereinigen meist auch nicht mehr Speicherplatz als eine reine Minor Collection. Dabei haben

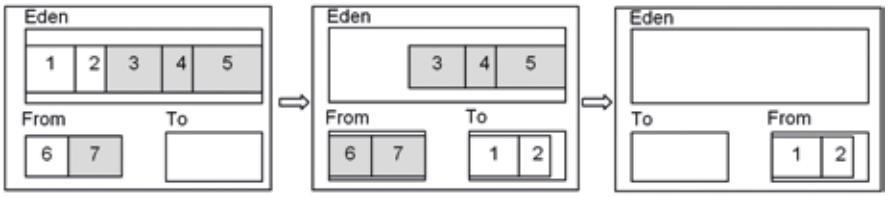


Abbildung 3: Mark-And-Copy (Quelle: In Anlehnung an [8])

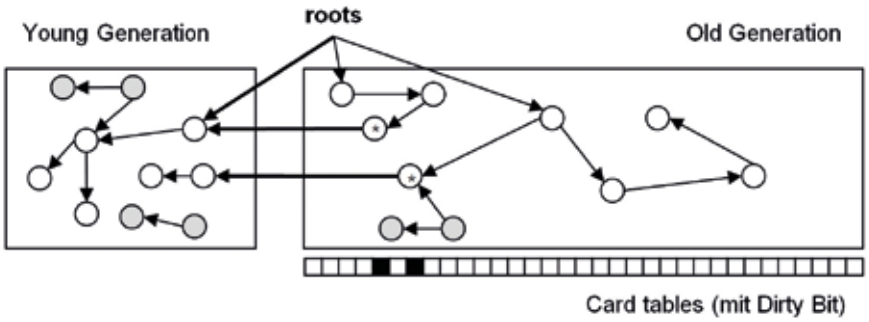


Abbildung 4: Intergenerational References (Quelle: In Anlehnung an [1, S. 168])

Major Collections jedoch meist eine deutlich längere Laufzeit, da der zu bereinigende Bereich wesentlich größer ist.

Mark-and-Copy

Ein rein für die Young Generation verwendbarer Algorithmus ist „Mark-and-Copy". Er ist aktuell auch der Standard-Algorithmus für die Young Generation. Mark-and-Copy läuft in zwei Phasen ab, einer Markierungs-Phase und einer Kopier-Phase. Die Markierung läuft wie bereits im Kapitel „Mark-and-Sweep" beschrieben ab. Es werden dabei jedoch nur Root-Referenzen verfolgt, die auf Objekte in der Young Generation verweisen. Beim Kopieren spielen nun die Survivor-Spaces eine entscheidende Rolle. Zu Beginn einer GC ist der To-Space leer, wohingegen der From-Space Objekte aus früheren GCs enthalten kann. Alle überlebenden Objekte aus Eden und dem From-Space werden nun in den To-Space kopiert. Für den Fall, dass der To-Space nicht alle Objekte aufnehmen kann, werden direkt alle überlebenden Objekte in die Old Generation verschoben. Eine Ausnahme existiert hierbei für Objekte, welche bereits eine festgelegte Anzahl an GCs überlebt haben. Diese werden in die Old Generation verschoben. Dieser Vorgang heißt „Promotion". Der Schwellwert, nach wie vielen überlebten GC-Durchläufen ein Objekt verschoben wird, wird als „Tenuring Threshold" bezeichnet. Dieser kann entweder über einen JVM-Parameter eingestellt oder automatisch durch die JVM gesteuert werden. Nach Abschluss des Kopiervorgangs wird der nun leere From-Space zum To-Space für die nächste GC und der To-Space zum From-Space.

Abbildung 3 verdeutlicht nochmals den Kopiervorgang. Die grau hinterlegten Objekte sind nicht mehr über eine Root-Referenz erreichbar und werden während des GC-Durchlaufs entfernt. Objekt Nr. 6 hat den Tenuring Threshold erreicht und wird nicht in den To-Space, sondern in die Old Generation verschoben.

Während der Markierung ist bei einer Minor Collection noch ein Sonderfall zu beachten. Prinzipiell müssen alle noch referenzierten Objekte innerhalb der Young Generation gefunden werden. Hierzu werden alle Root-Referenzen, die auf Objekte der Young Generation zeigen, verfolgt und alle von dort erreichten Objekte als „lebendig" markiert. Root-Referenzen in die Old Generation werden nicht verfolgt, da nur die Young Generation aufgeräumt werden soll und daher die Suche nach lebenden Old Generation-Objekten einen nicht notwendigen Mehraufwand darstellt. Es kann jedoch die Situation vorliegen, dass ein Objekt in der Young Generation nicht über eine Root-Referenz, jedoch über eine Referenz eines Old Generation-Objekts erreicht werden kann. Man spricht in diesem Fall von „Intergenerational References (IR)". Dies kann beispielsweise durch Promotion entstehen, indem das jetzige Old-Generation-Objekt bei einer vorangegangen GC verschoben wurde, das referenzierte Objekt jedoch in der Young Generation verblieben ist. Eine zweite Möglichkeit wäre, dass die Applikation dem Old Generation-Objekt explizit eine Referenz auf das Young Generation-Objekt gegeben hat und dies mittels einer „Write Barrier" kennzeichnet. Die Quell-Objekte der IRs sind dabei in einem „Remembered Set" hinterlegt. In beiden Fällen ist es erforderlich, diese Referenz zu erkennen und das referenzierte Young-Generation-Objekt nicht zu entfernen. Daher sind während der Markierungs-Phase zusätzlich vorhandene IRs zu verfolgen. Damit der Garbage Collector diese IRs erkennt, kommen „Card Tables" zum Einsatz. Dabei wird die Old Generation in Speicherbereiche (Cards) aufgeteilt. Wird nun in einer dieser Cards eine IR erzeugt, so wird der Bereich durch ein „Dirty Bit" markiert. Während der Markierungs-Phase werden diese IRs dann zusätzlich zu den Root-Referenzen verfolgt.

Abbildung 4 zeigt ein Beispiel für IRs. Die mit „*" markierten Objekte verfügen über IRs. Während der Markierungsphase werden die fett markierte Root-Referenz sowie die ebenfalls fett markierten IRs verfolgt. Unter der Old Generation sind exemplarisch die Cards dargestellt. Man sieht, dass das „Dirty Bit" der Cards mit den IRs gesetzt ist. Durch die Verfolgung der IRs werden weitere lebendige Objekte in der Young Generation gefunden. Die grau markierten Objekte in der Young Generation stellen nicht mehr referenzierte Objekte dar und sind nach der GC nicht mehr vorhanden.

Dieser Mechanismus bringt zwei Nachteile mit sich. Erstens kosten die Card Tables zusätzlichen Speicherplatz. Zweitens entstehen Zusatzaufwände durch das Setzen des Dirty Bit sowie während der GC durch das Suchen nach IRs.

Je mehr IRs existieren, desto länger wäre die GC und umso mehr würden die erwähnten Vorteile einer Generational GC schwinden. Aufgrund der im Kapitel „Generational Garbage Collection" angesprochenen „weak generational hypothesis",

die besagt, dass nur wenige Referenzen von der Old Generation in die Young Generation bestehen, ist dieser Zusatzaufwand jedoch begrenzt. Aus diesem Grund ist die Generational-GC auch der Standard bei der HotSpot JVM.

Allgemein kann man festhalten, dass Mark-and-Copy ein effizienter Algorithmus ist. Man spricht bei diesem Garbage Collector auch vom „Throughput Collector", da er für das Erreichen eines hohen Durchsatzes verwendet wird. Das Ziel der Durchsatz-Optimierung wird später erläutert. Da das Verschieben von Objekten nicht ohne Anhalten der Applikations-Threads durchgeführt werden kann, entstehen auch durch Mark-and-Copy Stop-the-World-Pausenzeiten. Diese halten sich auf Grund der geringen Größe der Young Generation jedoch in Grenzen.

Um die Effizienz des Algorithmus weiter zu steigern, kann auch eine parallelisierte Variante von Mark-and-Copy verwendet werden. Hierbei wird das Kopieren parallel durch mehrere GC-Threads durchgeführt. Dazu ist der To-Space in sogenannte „Parallel Local Allocation Buffers" (PLAB) aufgeteilt. Jeder Thread bekommt einen spezifischen PLAB zugewiesen, in den er (ohne Synchronisation mit den anderen Threads) die überlebenden Objekte verschieben darf. Ist ein PLAB voll, so bekommt der Thread einen neuen zugewiesen. Ein gewisses Maß an Fragmentierung ist hierbei nicht zu vermeiden. Dies wird allerdings durch die kürzer ausfallenden Stop-the-World-Pausen in Kauf genommen.

Lessons learned: Wie dargestellt ist Mark-and-Copy ein effizienter Algorithmus zur Erreichung eines hohen Durchsatzes. Ebenfalls werden nur geringe Stop-the-World-Pausen verursacht. Er eignet sich daher besonders für interaktive Systeme wie zum Beispiel Web-Shops. Für die Young Generation ist er aktuell der meist verwendete Collector.

Mark-and-Compact

Ein aufwendiger Algorithmus, der das Problem der Heap-Fragmentierung durch Zusammenschieben (compacting) aller überlebender Objekte zu beheben versucht, ist „Mark-and-Compact". Dieser kann für den gesamten Heap eingesetzt werden, kommt jedoch meist nur auf der Old Generation zum Einsatz.

Mit vier Phasen ist der Algorithmus relativ komplex. Zu Beginn werden erneut alle überlebenden Objekte markiert. Anschließend ist dem Garbage Collector bekannt, welche Heap-Bereiche frei werden, da alle erreichbaren Objekte markiert wurden. Für die überlebenden Objekte werden nun neue Positionen im Heap bestimmt. Dabei wird versucht, die Objekte mit möglichst geringer Fragmentierung neu anzuordnen. Nachdem für jedes Objekt eine neue Position bestimmt wurde, werden vom Kollektor die Referenzen auf diese an die neuen Heap-Positionen angepasst. Abschließend werden die Objekte innerhalb des Heaps verschoben.

Durch die Beschreibung wird deutlich, dass es sich um einen blockierenden Algorithmus mit möglicherweise langen Pausenzeiten handelt. Um diese zu verkürzen, steht auch der Mark-and-Compact in einer parallelisierten Variante zur Verfügung.

Lessons learned: Für einen hohen Durchsatz ist Mark-and-Compact vorteilhaft. Durch eine sehr geringe Heap-Fragmentierung kann die eigentliche Applikation nach einer GC möglicherweise lange ohne Unterbrechung ausgeführt werden. Er eignet sich unter anderem für Batch-Systeme oder Daten-Sicherungen, bei denen die Pausenzeiten nur eine untergeordnete Rolle spielen. Für interaktive Systeme hingegen ist er meist ungeeignet. Hierfür eignet sich dagegen der im kommenden Kapitel vorgestellte Concurrent-Mark-and-Sweep.

Concurrent-Mark-and-Sweep

Wie bereits beschrieben können durch Major Collections lange Pausenzeiten entstehen. Grund hierfür ist die Größe der Old Generation. Es wurde daher für die Old Generation ein Algorithmus entwickelt, welcher nebenläufig (concurrent) zur eigentlichen Applikation ausgeführt werden kann – der „Concurrent-Mark-and-Sweep" (CMS). Wie dem Namen zu entnehmen ist, handelt es sich dabei jedoch nur um einen Sweep-Algorithmus, bei dem sich das Problem der Heap-Fragmentierung nicht vermeiden lässt. Da Objekte, sobald sie einmal die Old Generation erreicht haben, jedoch selten ihre Verbindung zu einer Root-Referenz verlieren, hält sich dieses Problem in Grenzen und kann auf Grund des Performance-Vorteils toleriert werden.

Zu Beginn des CMS werden in einer kurzen Stop-the-World-Pause alle Old-Generation-Objekte bestimmt, die von den Root-Referenzen erreicht werden können. Ausgehend von diesen Objekten werden dann nebenläufig zur Applikation alle lebendigen Objekte im Heap bestimmt. Da die Applikation nebenbei jedoch weiter auf dem Heap arbeiten kann, ist nicht sichergestellt, dass alle als lebendig markierten Objekte nach Abschluss der nebenläufigen Markierung auch tatsächlich noch lebendig sind.

Eine ehemals gesetzte Referenz kann nach der Objekt-Markierung entfernt worden sein. In diesem Fall spricht man von „Floating Garbage". Dieser hält sich meist in Grenzen und stellt kein weiteres Problem dar, da er einfach bei der nächsten GC entfernt wird. Ein größeres Problem jedoch stellt die Allokation von Objekten in einem bereits gescannten Heap-Bereich dar. Diese Objekte müssen zwangsläufig als lebendig erkannt werden, sofern sie über eine Root-Referenz erreichbar sind. Daher erfolgt nach der nebenläufigen Markierung eine nicht nebenläufige „Re-Markierung". In dieser werden alle vom Allokator als geändert (dirty) markierten Bereiche erneut auf lebendige Objekte gescannt. Nach Abschluss der Re-Markierung sind dem Kollektor definitiv alle lebendigen Objekte bekannt. Diese können dann in der abschließenden, nebenläufig ausgeführten, Sweep-Phase entfernt werden.

Um die Pause zu verkürzen, die durch die nicht nebenläufige Re-Markierung entsteht, kann optional zwischen der nebenläufigen Markierung und der Re-Markierung eine Zusatzphase durchgeführt werden. Während dieser als „Preclean(ing)" bezeichneten Phase werden nebenläufig die als „dirty" markierten Heap-Bereiche auf lebendige Objekte gescannt. Diese Zusatzphase hat dabei eine konfigurierte maximale Laufzeit (Standardwert 5000 ms) und ist auch iterativ möglich.

Wie alle bisherigen GC-Algorithmen ist auch der CMS parallel durch mehrere Threads ausführbar. Die Besonderheit hierbei ist, dass nur nicht nebenläufige Phasen durch mehrere Threads abgearbeitet werden. Um die Applikation während der nebenläufigen Phasen möglichst wenig zu beeinträchtigen, werden diese nur von einem einzelnen Kollektor-Thread bearbeitet.

Auf Grund der Nebenläufigkeit wird der GC-Durchlauf des CMS bereits gestartet, sobald ein definierter Füllgrad der Old Generation vorliegt. Man bezeichnet diesen auch als „OccupancyFraction". Dies soll sicherstellen, dass während der GC weiterhin genügend Platz zur Allokation neuer Objekte in der Old Generation zur Verfügung steht. Den größten Nachteil des CMS stellt die mögliche Heap-Fragmentierung dar. Durch diese kann auch die Minor Collection verlangsamt werden, da bei der Promotion erst eine passende Lücke in der Old Generation gefunden werden muss. Die freien Speicherbereiche in der Old Generation werden wiederum mit einer FreeList verwaltet. Diesem Nachteil gegenüber steht der große Performance-Vorteil durch die deutlich kürzeren Stop-the-World-Pausen.

Lessons learned: Der CMS empfiehlt sich speziell für Systeme, deren Pausenzeiten kurz gehalten werden müssen. Hierzu zählen unter anderem Webserver und interaktive Systeme. Eine gute und häufig verwendete Kollektor-Kombination ist Mark-and-Copy für die Young Generation und CMS für die Old Generation.

Garbage First

Der neueste Garbage Collector hat die Bezeichnung „Garbage First" (G1). Er wurde mit Java 6 Update 14 in einer experimentellen Version eingeführt. Seit Java 7 liegt er in einer finalen Version vor. Langfristiges Ziel ist die Ablösung des CMS. Der G1 nutzt ebenfalls die Vorteile der heutigen Hardware und versucht möglichst viele Arbeiten parallel durch mehrere Threads abzuarbeiten. Einige Aufgaben können auch nebenläufig zur eigentlichen Applikation durchgeführt werden. Allgemein lässt sich beim G1 die maximale Pausenzeit durch einen JVM-Parameter vorgeben. Die Einhaltung dieser Zeit wird jedoch nicht garantiert. Daneben kann man auch das Intervall zwischen dem Start zweier GCs vorgeben.

Anders als bei den bisher beschriebenen Algorithmen ist der Heap beim G1 nicht in Generationen, sondern in „Regionen" eingeteilt. Eine Region ist dabei zwischen 1 MB und 32 MB groß. Die Größe ist konfigurierbar, muss jedoch eine „2er"-Potenz sein. Abbildung 5 zeigt die unterschiedlichen zur Verfügung stehenden Regionen-Typen. Diese sind mit den bereits bekannten Generationen vergleichbar. Der Typ einer Region ist dabei nicht fixiert, sondern bestimmt sich dynamisch aus den aktuell vorliegenden Anforderungen.

Der GC-Durchlauf beim G1 kann in zwei Modi ablaufen zwischen denen dynamisch gewechselt werden kann. Der „Fully Young Mode" bereinigt alle „Young Regions" (Allocation- und Survivor-Regions), wohingegen beim „Partially Young Mode" zusätzlich auch ausgewählte „Old Regions" bereinigt werden. Welche Regionen nun während einer GC bereinigt werden, bestimmt sich aus dem „Müllanteil" der jeweiligen Region. Daher resultiert auch der Name dieses Kollektors, da Regionen mit hohem „Müllanteil" (engl. garbage first) vorrangig bereinigt werden.

Jede Region verfügt über ein sogenanntes „Remembered Set", das alle Referenzen aus anderen Regionen in diese Region enthält. Für jede Änderung dieses Sets wird ein „Update Task" erstellt, der abgearbeitet wird, sobald genügend CPU-Zeit zur Verfügung steht. Nachfolgend ist ein GC-Durchlauf mit Markierung und Evakuierung beschrieben.

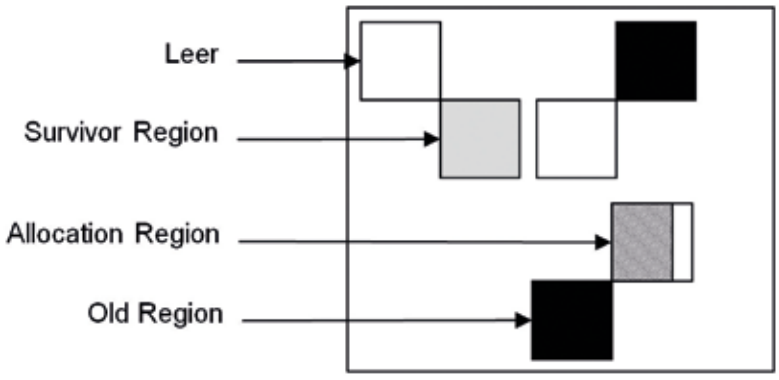


Abbildung 5: G1-Heap-Aufteilung (Quelle: In Anlehnung an [1, S. 232])

Wie beim CMS läuft auch beim G1 die Markierung nebenläufig zur Applikation ab (Concurrent Marking). Dabei entsteht ein „Snapshot-at-the-Beginning". Dieser enthält alle lebendigen Objekte zu Beginn der Markierung. Hieraus wird ermittelt, ob die Objekte, die in den Remembered Sets vermerkt sind, noch leben. Darüber hinaus wird in einer „Count-Phase" die Größe der erreichten Objekte festgehalten. Anhand dieser Information werden die zu bereinigenden Regionen in einem Garbage-First-Ansatz ausgewählt. Objekte, die während der nebenläufigen Markierungs-Phase von der Applikation erstellt werden, sind automatisch als lebendig gekennzeichnet.

Die Evakuierung überlebender Objekte findet während einer Stop-the-World-Pause statt. Alle dabei anfallenden Aufgaben werden in relativ kleine Sub-Tasks zerlegt, die parallel von allen zur Verfügung stehenden CPUs abgearbeitet werden, um die Pause so kurz wie möglich zu halten beziehungsweise um das angegebene Pausenziel zu erreichen. In dieser Pause werden drei Phasen durchlaufen. Zuerst erfolgt die Ausführung der noch nicht durchgeführten Updates der Remembered Sets. Danach werden durch Referenzverfolgung alle lebendigen Objekte innerhalb des „Collection Sets" bestimmt. Die Referenzen eines Objekts werden dabei nur verfolgt, wenn dieses Quell-Objekt lebendig ist. Zusätzlich werden alle Referenzen, die in das Collection Set verweisen, auf lebendige Objekte überprüft. Zu diesen Referenzen zählen beispielsweise RootSet-Referenzen sowie Referenzen aus den Remembered Sets anderer Regionen. Drittens können „Mark-Stack-Referenzen" existieren, die durch Unterbrechung einer nebenläufigen Markierung entstehen, was in der Praxis jedoch selten anzutreffen ist.

Die letzte Phase der GC-Pause ist das eigentliche Bereinigen der Regionen. Dabei werden alle überlebenden Objekte der Regionen in eine neue Region (To-Space) kopiert und die bereinigten Regionen freigegeben. Die überlebenden Objekte aus unterschiedlichen Regionen können dabei in einer neuen Region zusammengefasst und ohne Fragmentierung in dieser angeordnet werden. Auch Regionen, welche überhaupt keine leben-

digen Objekte mehr enthalten, werden freigegeben.

Lessons learned: Der G1 ist ein Allround-Kollektor, welcher im Vergleich zu den etablierten Kollektoren einen neuen Ansatz der Heap-Bereinigung wählt. Der wichtigste Ansatz des G1 ist die Heap-Bereinigung über die Laufzeit der Applikation. Durch die vorgebbaren Pausenzeiten werden nach und nach die einzelnen Regionen bereinigt. Ein Test des G1 empfiehlt sich daher auf allen Systemen.

Garbage-Collection-Tuning

Bei einer zu langsamen Applikation ist eventuell eine Optimierung der GC-Konfiguration anzustreben. Dabei ist festzuhalten, dass ein GC-Tuning nur bei einem Problem angestrebt werden soll, das auf einen fehlerhaft konfigurierten Garbage Collector zurückzuführen ist. Andernfalls wird ein dementsprechendes Tuning nicht den erhofften Effekt bringen. Daher sind vor einem GC-Tuning zum Beispiel Implementierungsfehler in einer Applikation auszuschließen. Ein möglicher Implementierungs-Schwachpunkt könnte eine statische Collection sein, zu der Objekte zwar hinzugefügt, jedoch nicht wieder entfernt werden, wenn sie nicht mehr benötigt werden. Über die statische Referenz der Collection kann man dann auch alle sich in der Collection befindlichen Objekte erreichen und als lebendig markieren. Allgemein ist der Einsatz von statischen Variablen zu überprüfen.

Die Feststellung, ob Applikations-Probleme (etwa zu hoher Speicherverbrauch oder zu lange Antwortzeiten) durch GC verursacht werden, kann durch die Analyse von „GC-Traces" erfolgen. Dies sind die Logfiles des Garbage Collectors, in denen sich dessen Aktivitäten inklusive der zeitlichen Dauer ablesen lassen. Die Detail-Tiefe der Logfiles lässt sich mit JVM-Parametern einstellen. Aufgrund der Größe und der zumeist sehr detaillierten Aussagen über freien Speicher, bereinigten Speicher, Zeiten für Speicherbereinigung etc. ist eine manuelle Auswertung sehr langwierig. Hierzu ist alternativ das OpenSource-Tool „GCViewer" einsetzbar, um einen GC-Trace einzulesen. Das Tool stellt die Informationen anschließend grafisch aufbereitet und in Tabellenform dar. Lange Pausenzeiten oder ein schlechter Durchsatz lassen sich so schnell erkennen. Korrespondieren die auftretenden Applikations-Probleme mit langen GC-Pausenzeiten, so kann man mit einer recht hohen Wahrscheinlichkeit annehmen, dass eine fehlerhafte GC-Konfiguration vorliegt und eine Optimierung anzustreben ist.

Einen wichtigen Aspekt, weshalb eine optimal konfigurierte GC für die Performance der Applikation unabdingbar ist, stellt die heutige leistungsfähige Hardware dar. Je mehr Rechenkerne gleichzeitig an der Allokation neuer Objekte beteiligt sind, desto schneller wird der Heap voll und eine GC ausgelöst. A. Langer und K. Kreft beschreiben dies in [1, S. 208] wie folgt: „Für das Erzeugen von Objekten ergibt sich also, was sonst im Allgemeinen nicht gilt: Die Anzahl neuer Objekte steigt (fast) linear mit der Anzahl der zur Verfügung stehenden Cores." Daraus lässt sich schließen, dass mit zunehmender Anzahl an CPUs die Anzahl der GC-Durchgänge steigt und der Durchsatz der Applikation sinkt. Daher sollte bei Erhöhung der CPU-Anzahl der Heap ebenfalls vergrößert werden, um einen zu schnellen Speichermangel zu verhindern.

Wie lässt sich die GC-Performance nun messen? Im Rahmen der Untersuchung konnten hierfür zwei wichtige Kennzahlen ermittelt werden. Diese sind auf der einen Seite der Durchsatz (Throughput) und auf der anderen Seite die verursachten Stop-the-World-Pausenzeiten. Daneben existieren weitere Kennzahlen, wie die Häufigkeit einer GC (Frequency of Collection) oder die Zeitspanne, in welcher der Speicherplatz eines Objekts nach Entfernen der Verbindung zu einer Root-Referenz wieder verfügbar wird (Promptness). Bei älteren Systemen oder Systemen mit beschränktem Arbeitsspeicher kann auch der unnötige Verbrauch von Arbeitsspeicher eine wichtige Kennzahl sein.

Durchsatz

Unter Durchsatz wird der Prozentsatz an Zeit verstanden, welchen eine Applikation nicht mit GC beschäftigt ist. Für ein aussagekräftiges Ergebnis sollte dieser über einen längeren Zeitraum ermittelt werden, am besten über die Gesamtlaufzeit der Applikation. Der Kehrwert, also die prozentual mit GC verbrachte Zeit, wird als GC Overhead bezeichnet.

Die Ermittlung des Durchsatzes kann aus dem GC-Trace erfolgen. Hierzu sind die fett markierten Bereiche des nachfolgenden Beispiel-Traces aus [1, S.200] relevant: 176.532: [GC 56794K -> 17083K (129792K), 0.1041413 sec.]. Der erste markierte Wert ist dabei der Zeitstempel (gemessen in Sekunden ab Start der Applikation), an welchem die GC begonnen wurde. Der zweite markierte Wert stellt die verursachte Stop-the-World-Pause dar. In die Durchsatz-Berechnung geht die Gesamtzeit des Trace (gz) und die Summe über alle Stop-the-World-Pausen (zmgc) ein. Der Durchsatz (in Prozent) ist nach folgender Formel zu berechnen: (100 * (1 - zmgc / gz)). Ein Nachteil ist jedoch, dass die CPU-Zeiten der nebenläufigen GC-Aktivitäten nicht berücksichtigt werden. Dies liegt unter anderem daran, dass diese nicht explizit im GC-Trace ausgewiesen sind.

Werte über 95 Prozent werden als sehr guter Durchsatz angesehen. Bei vielen Systemen ist eine Optimierung des Durchsatzes praktisch eher irrelevant (Ausnahme: Batchbetrieb), da sehr lange Pausenzeiten entstehen können. Der Aufwand, den Durchsatz beispielsweise um 10 Prozent zu steigern, ist häufig sehr hoch. Alternativ kann eine CPU mit mehr Leistung eingesetzt werden, wodurch der Durchsatz zwar vorerst nicht erhöht, die Performance der Applikation durch schnellere Verarbeitung der GC-Aktivitäten dennoch gesteigert wird. Der Grund hierfür ist in den kürzer ausfallenden Stop-the-World-Pausen zu sehen.

Stop-the-World-Pausen

Die Reduzierung von GC-Pausenzeiten ist ein häufig angestrebtes Ziel und oft der Anlass, sich näher mit dem GC-Tuning zu beschäftigen. Wie bereits dargestellt, sind GC-Pausen unvermeidbar und stellen tendenziell auch kein Problem dar. Werden sie jedoch zu lang, sodass sie den Betrieb der Applikation beeinträchtigen (etwa durch zu lange Reaktionszeiten bei einer interaktiven Applikation), rücken sie in den Fokus einer Optimierung. Auch hierbei muss vor einer GC-Optimierung sichergestellt sein, dass der Fehler durch einen fehlerhaft konfigurierten Garbage Collector ausgelöst wird. Häufig sind auch hier Design- und Implementierungsfehler eher der Grund für eine langsame Applikation.

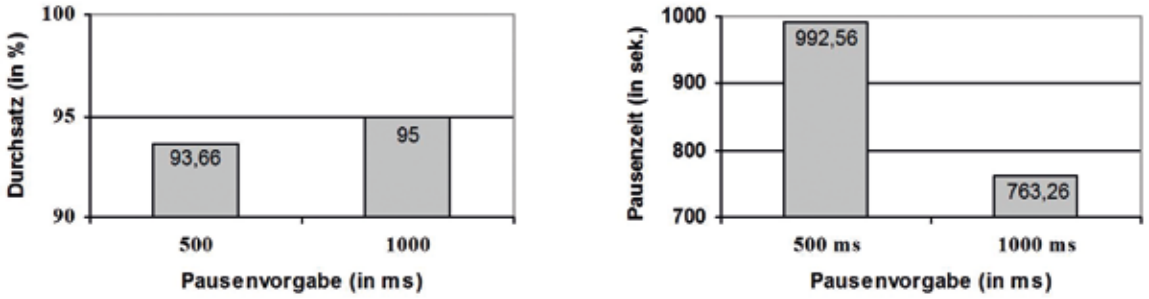


Abbildung 6: Lasttest CMS I – Durchsatz (links); Summe der Pausenzeiten (rechts)

Können diese ausgeschlossen werden, so lassen sich zum Beispiel die GC-Traces nach langen Pausenzeiten durchsuchen. Korrespondieren diese mit dem Auftreten des Problems, kann im nächsten Schritt ein GC-Tuning hinsichtlich der Reduzierung der Pausenzeiten begonnen werden.

Lessons learned: Es wurde dargestellt, dass der Durchsatz und die Stop-the-World-Pausenzeiten sehr gute Kennzahlen für die Ermittlung der GC-Performance sind. Diese lassen sich relativ leicht durch Hilfsmittel wie GCViewer aus den GC-Traces ermitteln. GC-Traces können dabei automatisch durch die JVM erstellt werden, sofern dies mittels Parametern eingeschaltet ist.

Konkrete GC-Tuning-Maßnahmen

Es lässt sich festhalten, dass ein GC-Tuning ein iterativer und möglicherweise langwieriger Prozess ist. Nach jeder durchgeführten Änderung sollte überprüft werden, ob man dem gesetzten Ziel näher gekommen ist. Auch müssen Ziele während des Tunings eventuell angepasst oder aufgegeben werden. Für die gewählten Ziele ist wichtig, dass diese realistisch und zu erreichen sind. So ist beispielsweise ein Ziel „Keine Pausenzeiten bei 100 Prozent Durchsatz" weder realistisch noch erreichbar. Es empfiehlt sich in jedem Fall vor einer GC-Konfiguration die Standardwerte der JVM zu testen. Oftmals lassen sich gänzlich ohne eigene Konfiguration bereits respektable Ergebnisse erzielen. Mit jedem Upgrade der JVM ist jedoch ein neuer Test notwendig, da sich die Standard-Einstellungen ändern können.

Der erste Schritt bei einem GC-Tuning ist die Wahl des richtigen Garbage Collectors. Für hohen Durchsatz empfiehlt sich der Einsatz des parallelen Mark-and-Copy für die Young- sowie der parallele Mark-and-Compact für die Old Generation. Um eine Optimierung hinsichtlich kurzer Pausenzeiten anzustreben, kann für die Old Generation alternativ der CMS gewählt werden.

Im zweiten Schritt sind die gewählten Kollektoren an die Objektpopulation der zu tunenden Applikation anzupassen. Dabei ist zu beachten, dass sich die Objektpopulation über die Laufzeit der Applikation mit hoher Wahrscheinlichkeit verändert. Webshops haben beispielsweise Zeiten, in denen sie stärker belastet werden und somit mehr Objekte erzeugen. Das Tuning ist in jedem Fall an die Lastspitzen des Systems anzupassen. Falls ein dementsprechendes Tuning für den Regelbetrieb zu viele Ressourcen benötigt, können auch individuelle Tunings auf unterschiedlichen JVMs für einzelne Betriebszeiten durchgeführt werden.

Die wichtigste Tuning-Maßnahme ist die Anpassung der Größenverhältnisse zwischen Young- und Old Generation sowie zwischen Eden und den Survivor Spaces. Ziel bei der Anpassung sollte sein, Objekte möglichst in der Young Generation sterben zu lassen, um langwierige Major Collections zu vermeiden.

Für das Verhältnis Young zu Old Generation steht der JVM-Parameter „-XX:NewRatio=<value>" zur Verfügung. Wird dieser beispielsweise auf „3" gesetzt, hat dies ein Größenverhältnis von „1:3" zwischen Young und Old Generation zur Folge. Die Anpassung zwischen Eden und den Survivor Spaces wird über den Parameter „-XX:SurvivorRatio=<value>" durchgeführt. Wird für diesen der Wert „6" gewählt, entspricht ein Survivor-Space ein Achtel der Young-Generation-Größe.

Eine weitere Möglichkeit, die Promotion in die Old Generation zu verhindern, ist die Erhöhung des Tenuring Threshold, dem Schwellwert ab wie viel überlebten GCs ein Objekt in die Old Generation übertragen wird. Objekte mit mittellanger Lebenszeit können so länger in den Survivor Spaces gehalten werden, was wiederum Aufwände für die GC der Old Generation sparen kann. Zur Ausgabe des aktuell eingestellten Tenuring Thresholds sowie detaillierter Altersangaben von Objekten der Young Generation ist der Parameter „PrintTenuringDistribution" zuständig. Hiermit lässt sich der Lebenszyklus eines Objekts nachverfolgen.

Zur Verkürzung der Pausenzeiten empfiehlt sich ein Kollektor, welcher möglichst viele GC-Aufgaben nebenläufig ausführt. Hierzu kann zum Beispiel der CMS gewählt werden. Allerdings besteht bei diesem das Problem der Heap-Fragmentierung, welches im schlechtesten Fall eine extrem lange Pausenzeit verursacht. Der Grund hierfür ist ein Rückfall in den seriellen Mark-and-Compact, falls in der Old Generation keine Objekte mehr allokiert werden können. Um dies zu verhindern, kann man einerseits die Old Generation extrem vergrößern. Zu einem für die Applikation unkritischen Zeitpunkt kann dann durch den Aufruf von „System.gc()" manuell eine GC ausgelöst werden. Eine andere Anpassungsmöglichkeit beim CMS ist die Einstellung der „OccupancyFraction". Ziel dabei ist, dass der CMS weder zu oft noch zu selten läuft. Falls kein expliziter Wert für den Füllgrad gesetzt wird, wird dieser von der JVM während der Laufzeit dynamisch angepasst. Zu Grunde liegen dabei Informationen, welche von der JVM automatisch bei jeder durchgeführten GC gesammelt werden.

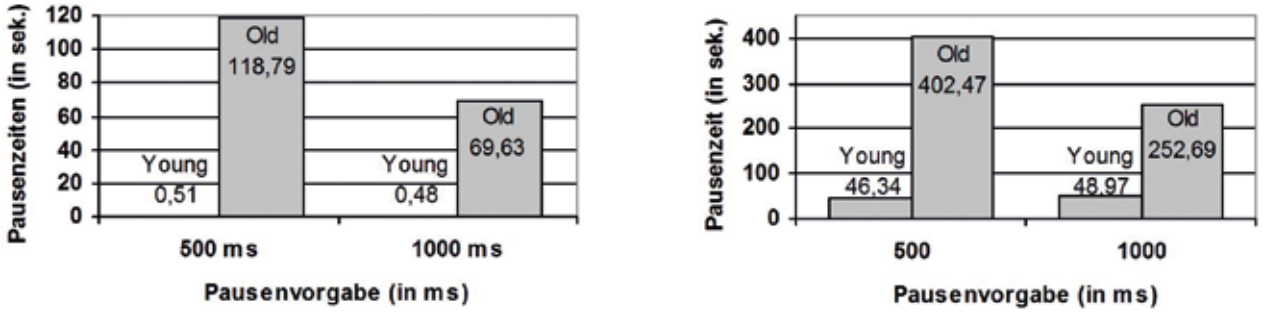


Abbildung 7: Lasttest CMS II – Ø Pausenzeit (links); maximale Pausenzeiten (rechts)

Lessons learned: GC-Tuning bedarf eines iterativen Tunings. Der Schwerpunkt einer GC-Optimierung liegt meist auf Anpassung von Größenverhältnissen zwischen den einzelnen Heap-Bereichen. Hauptziel ist ein Verhindern der Promotion von Objekten in die Old Generation. Die Größenverhältnisse sind dementsprechend so anzupassen, dass Objekte mit kurzer und mittellanger Lebenszeit möglichst in der Young Generation (inklusive den Survivor Spaces) verbleiben bis sie entfernt werden.

Leistungs-Tests

Im Rahmen der Untersuchung wurden Leistungs-Tests mit unterschiedlichen GC-Konfigurationen durchgeführt. Als Testsystem stand dabei ein Clusterknoten (in Kopie) eines bereits produktiv eingesetzten B2B-Webshop-Systems zur Verfügung. Um realistische Testbedingungen zu schaffen und vergleichbare Testresultate zu erhalten, wurden am produktiven System „Requests" aufgezeichnet, die am Testsystem beliebig häufig wiedergegeben werden konnten. Die Lasttests erfolgten mit zwei unterschiedlichen GC-Konfigurationen. Jeder Test wurde sechsmal wiederholt. Das Testsystem wurde pro Durchlauf rund sieben Stunden mit Requests belastet und die dabei erstellen GC-Traces mittels GC-Viewer analysiert und ausgewertet. Anschließend wurden aus allen durchgeführten Testdurchläufen Mittelwerte gebildet, um einmalige Schwankungen auszugleichen. Diese Mittelwerte sind in den Abbildungen 6 und 7 dargestellt.

Im ersten Testfall kam die GC-Kombination Parallel Mark-and-Copy (Young Generation) und CMS (Old Generation) unter die Lupe. Die Heap-Größe betrug 16 GB, das Verhältnis zwischen Young- und Old Generation „1:5". Insgesamt wurden 30 parallele GC-Threads verwendet. Als maximale Pausenzeit für den CMS wurden einmal 500 ms und einmal 1000 ms vorgegeben. Ziel dieses Tests war die Überprüfung, ob es sich bei der gewählten Konfiguration um eine realistisch einsetzbare Konfiguration handelt.

Abbildung 6 (links) zeigt, dass der Durchsatz bei diesem Testfall mit durchschnittlich 95 Prozent sehr hoch war. Korrespondierend dazu war auch die Gesamtsumme der Pausenzeiten (Abbildung 6, rechts) moderat. In Abbildung 7 (links) ist hingegen zu sehen, dass die vorgegebenen maximalen Pausenzeiten im Durchschnitt nicht erreicht wurden. Speziell die Major Collections konnten diese ziemlich deutlich nicht einhalten. Auch die maximalen Pausenzeiten (Abbildung 7, rechts) waren extrem hoch. Dies führte in Summe zu einer hohen Anzahl an Request-Timeouts während des Tests. Als Fazit dieses Testfalls zeigte sich, dass die Kombination Mark-and-Copy und CMS eine gute Wahl für einen hohen Durchsatz darstellt. Die Verringerung der Pausenzeiten wären der nächste Ansatzpunkt für weitere Optimierung. Auf Grund verschiedener Einschränkung konnte im Rahmen der Untersuchung hier jedoch keine weitere Optimierung vorgenommen werden.

Beim zweiten Lasttest kam der G1 zum Einsatz. Um eine finale Version dieses Kollektors zu verwenden, wurde für diesen Testfall Java 7 eingesetzt. Die Heap-Größe betrug erneut 16 GB und es kamen 30 parallele GC-Threads zum Einsatz. Als maximale Pausenzeiten wurden wie beim vorherigen Test 500 ms bzw. 1000 ms gewählt.

Abbildung 8 (links) zeigt, dass auch mit dem G1 ein sehr hoher Durchsatz-Wert zu erzielen ist. Doch, wie die durchschnittlichen Pausenzeiten in Abbildung 8 (rechts)

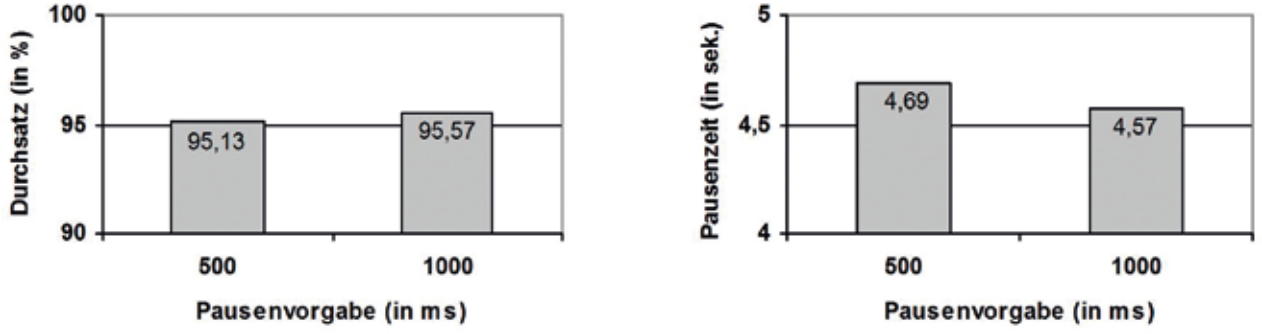


Abbildung 8: Lasttest G1 – Durchsatz (links); durchschnittliche Pausenzeiten (rechts)

zeigen, wurden auch mit dem G1 die vorgegebenen Pausenziele deutlich nicht eingehalten. Ein weiteres Ergebnis dieses Testfalls war, dass das vorliegende Testsystem bei Verwendung des G1 sehr viele Request-Timeouts hatte. Ein genauer Grund hierfür konnte nicht ermittelt werden. Daher konnte keine Empfehlung zum produktiven Einsatz des G1 gegeben werden. Die Leistungsfähigkeit dieses Kollektors müsste mit weiteren Tests und Konfigurationsanpassungen überprüft werden. Aufgrund unterschiedlicher Einschränkungen konnte dies im Rahmen der Untersuchung jedoch ebenfalls nicht bewerkstelligt werden.

Fazit

Bei der Untersuchung der Thematik „Garbage Collection im Java-Umfeld" wurde festgestellt, dass es sich um ein breites, komplexes Themengebiet handelt. Jeder zur Verfügung stehende Garbage Collector bietet eine Summe an Einstellmöglichkeiten. Für die GC-Konfiguration an sich konnten keine allgemeingültigen Empfehlungen gegeben werden. Es wurde sich daher darauf beschränkt Richtlinien zu erarbeiten, welche für die meisten Applikationen gelten sollten:

- Der JVM sollte so viel wie möglich Arbeitsspeicher zugewiesen werden (Heap-Größe). In einem offiziellen Oracle-Dokument ([6, Abschnitt 4]) wird hierzu folgende Richtlinie gegeben: „Unless you have problems with pauses, try granting as much memory as possible to the virtual machine."
  Bei der Zuweisung von Arbeitsspeicher an die JVM ist jedoch darauf zu achten, dass es sich um echten physikalischen Speicher handelt, um „Paging" zu vermeiden. Um mehr Speicher allokieren zu können, ist eventuell auch ein Umstieg von einer 32-Bit-JVM auf eine 64-Bit-JVM in Erwägung zu ziehen.
- Das Verhältnis zwischen Young und Old Generation ist so zu wählen, dass nur wirklich langlebige Objekte in die Old Generation promoviert werden. Hierzu können neben der Young Generation-Größe auch die Größe der Survivor-Spaces sowie der Tenuring Threshold angepasst werden.
- Eine GC der Old Generation kann durch eine Vergrößerung des ihr zustehenden Heaps hinausgezögert werden. Sobald die Applikation weniger belastet ist, kann eine GC manuell ausgelöst werden.
- Die Wahl eines passenden Kollektors muss an die angestrebten GC-Ziele angepasst werden. Je nach Kollektor und dessen Konfiguration kann entweder der Durchsatz erhöht und/oder die Pausenzeit minimiert werden.
- Eine physikalische Erweiterung der Ressourcen kann unter Umständen eine stärkere Wirkung haben als (langwierige) Parameter-Optimierungen. Hierzu zählt die CPU-Leistung, der verfügbare Arbeitsspeicher sowie die benutze Variante der JVM (32-Bit/64-Bit).
- Bei Erhöhung der CPU-Anzahl sollte auch die Heap-Größe erweitert werden. Hintergrund ist hier, dass die Allokation neuer Objekte sehr gut parallel ablaufen kann. Um den Durchsatz konstant zu halten, muss folglich auch die GC parallel durchgeführt werden, da die Anzahl der GCs vermutlich ansteigen wird.

Im Rahmen der Untersuchung wurde ebenfalls eine Übersicht der wichtigsten GC-JVM-Parameter erstellt. Diese ist bei den Autoren erhältlich.

Quellen

1. Langer, Angelika; Kreft, Klaus (2011): Java-Core-Programmierung. Memory Model und Garbage Collection. Frankfurt: entwickler.press.
2. Memory Management in the Java HotSpot Virtual Machine: http://www.oracle.com/technetwork/java/javase/tech/memorymanagement-whitepaper-1-150020.pdf
3. Java HotSpot VM Options: http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html
4. Java HotSpot Garbage Collection: http://www.oracle.com/technetwork/java/javase/tech/g1-intro-jsp-135488.html
5. Bacon, David F.; Diwan, Amer; Detlefs, David; Flood, Christine; Heller, Steve; Printezis, Tony (2004): Garbage-first garbage collection. In: Proceedings of the 4th international symposium on Memory management - ISMM ‹04: ACM Press, S. 37 - 48.
6. Java SE 6 HotSpot Virtual Machine Garbage Collection Tuning: http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html
7. Java Memory Leaks und andere Übeltäter (2.Akt): http://blog.codecentric.de/2011/03/java-memory-leaks-und-andere-ubeltater-2-akt/
8. Wie funktioniert der Java Garbage Collector: http://it-republik.de/jaxenter/artikel/2452

Mathias Dolag
m.dolag@isys-software.de

Mathias Dolag hat 2012 seinen Bachelor of Science im Fachbereich Wirtschaftsinformatik an der Hochschule München abgeschlossen. Seit 2010 arbeitet er als Software-Entwickler in den Bereichen Projekt-/Personal-Management, eCommerce und Web-Entwicklung bei der iSYS Software GmbH in München

Newsticker:

Oracle präsentiert die Java Embedded Suite 7.0

Die neue Lösung erleichtert die Erstellung von Anwendungen in vielen Embedded-Systems, darunter Netzwerk-Appliances, Geräte im medizinischen Bereich, Home Gateways und Router sowie große Peripheriegeräte wie Multifunktions-Drucker. Sie basiert auf Oracle Java Platform, Standard Edition (Java SE) Embedded 7, Java DB, Versionen von GlassFish for Embedded Suite und dem Jersey-Web-Services-Framework mit optimiertem Speicherbedarf. Weitere Informationen unter www.oracle.com/us/technologies/java/embedded/suite

Oracle bietet kostenlose ADF-Version

Detlef Müller, ORACLE Deutschland B.V.& Co. KG

Das Application Development Framework (ADF) ist Oracles Plattform für den Aufbau moderner, JEE-basierter Web-Anwendungen. Es gibt nun eine lizenzkostenfreie Version mit der Bezeichnung „ADF Essentials", um Entwicklern die Möglichkeit zu geben, ADF-Anwendungen auch lizenzkostenfrei in einem JEE-Application-Server eigener Wahl bereitzustellen. Ziel dieser Oracle-Offerte ist nach eigenen Angaben die Aufhebung von Lizenzbarrieren für die Adaption von ADF-Technologien. Den Kunden wird es freuen.

Bei Oracle ADF handelt es sich um eine Bündelung von Standard-basierten Technologien und Frameworks, mit denen man verschiedenste Arten von Web-Anwendungen auf Basis des „MVC Design Pattern" effizient und vor allem schnell entwickeln kann. ADF ist seit Ende September 2012 nun auch in einer lizenzkostenfreien Version mit der Bezeichnung „ADF Essentials" erhältlich.

Bisher war die Entwicklung von Anwendungen im JDeveloper kostenfrei. Lediglich für die Laufzeit-Umgebung, in der die ADF-Anwendungen eingesetzt werden, musste eine gültige Laufzeit-Lizenz vorliegen. Mit Ausnahme der „iAS Forms/Reports"-Edition war die ADF-Runtime-Lizenz automatisch in der Oracle-Application-Server-Lizenz enthalten. Für Nicht-Oracle-JEE-Server war bis dato eine explizite ADF-Lizenz erforderlich. Mit ADF Essentials ist diese nun nicht mehr grundsätzlich nötig. Sowohl in der Entwicklung als auch in der Laufzeit sind folgende ADF-Kerntechnologien lizenzkostenfrei nutzbar:

ADF Faces Rich Client Components

Ein Framework basierend auf Java-Server-Faces-Technologien mit mehr als 150 vorkonfektionierten, funktional ausgestalteten UI-Komponenten. Hinzu kommt eine umfangreiche Palette von Komponenten zur Datenvisualisierung.

- *ADF Controller*
  Eine Erweiterung des Java-Server-Faces-Controllers mit dem Fokus auf Kapselung und Wiederverwendung von Controller Flows sowie der Möglichkeit, Seitenregionen mit dynamischen Inhalten zu verarbeiten.
- *ADF Binding*
  Eine technische Abstraktionsschicht zur Verbindung von Business-Logik einer Anwendung im Model Layer mit der Ebene der Darstellung im View Layer.
- *ADF Business Components*
  Ein Framework zur Vereinfachung des Aufbaus von Business-Logik in einem bestehenden Modell aus relationalen Datenbanken.

Technisch betrachtet unterscheiden sich die ADF-Essentials-Anwendungen nicht von bisherigen ADF-Anwendungen. Der Einsatz dieser ADF-Anwendungen ist offiziell für WebLogic Server 11g, GlassFish 3.1 und WebSphere 7 unterstützt. Bezüglich der Lizenzfreigabe durch ADF Essentials gibt es aber keine Restriktionen für andere JEE-Server. Der JEE-Entwickler ist damit aus kommerzieller Sicht frei in der Wahl der Application-Server, solange er ausschließlich die in ADF Essentials freigegebenen Komponenten beziehungsweise Technologie-Bausteine einsetzt.

Wird ADF zu Open Source?

Nein, ADF ist mit Essentials nicht Open Source. Die ADF-Framework-Sourcen bleiben wie gehabt bei Oracle und werden auch von Oracle stets weiterentwickelt. Unterstützung wird im Rahmen des Community-Supports, also hauptsächlich über die Oracle-Foren, geleistet. Ein offizieller Produkt-Support steht über eine gültige ADF-Lizenz zur Verfügung. Alternativ kann auch Support speziell für ADF Essentials erworben werden.

Der JDeveloper ist und bleibt Oracles primäre Entwicklungsplattform für den Aufbau JEE-basierter Web-Anwendungen. Es gibt nach wie vor auch die Möglichkeit, mit dem Oracle Enterprise Pack for Eclipse (OEPE) ADF-Artefakte in Eclipse zu erstellen. Auch an dieser Strategie hat sich mit ADF Essentials nichts geändert.

Mit ADF Essentials ist übrigens keine spezielle JDeveloper-Version verknüpft. Das heißt, man arbeitet wie gewohnt im aktuellen JDeveloper-Release, in dem auch andere Frameworks aus der Fusion-Middleware-Produktwelt eingebettet werden. Es gibt auch keine spezielle Filterung für ADF Essentials im JDeveloper.

Die Paketierung von ADF Essentials hat keinen Einfluss auf Oracles Festhalten an der zentralen Entwicklungsstrategie pro ADF. ADF ist und bleibt die strategische Entwicklungsplattform für JEE-Web-Anwendungen und ist bereits heute die Basis sowohl für Fusion Applications als auch für fast alle Fusion-Middleware-Produkte wie Enterprise Manager, Oracle SOA Suite, Oracle WebCenter etc. Viele Tausend Kunden setzen heute Oracle ADF als Basis-Technologie für ihre Web-Anwendungen ein. Von daher ist die Veröffentlichung von ADF Essentials auch als Bestätigung für das Festhalten an ADF zu bewerten.

Detlef Müller
detlef.mueller@oracle.com

Detlef Müller (Dipl. Ing., Univ.) ist Leitender Systemberater der ORACLE Deutschland B.V. & Co. KG und seit 1998 in den Bereichen „Portale" und „ADF" beratend tätig.

Der Tiger im Tank: PL/SQL-Logik in Java-Anwendungen optimal nutzen

Björn Christoph Fischer

Business-Anwendungen auf Basis von Oracle Forms enthalten mitunter einen beachtlichen Anteil von PL/SQL in der Oracle-Datenbank. Eine solche Architektur, die in der Business-Logik in der Datenbank implementiert ist, wird auch als „datenbankzentriert" bezeichnet. Im Unterschied dazu ist die Architektur von webbasierten Java-Enterprise-Anwendungen meist mittelschichtzentriert, da hier die Business-Logik im Application-Server implementiert ist.

Folgende Fragen beschäftigen viele IT-Verantwortliche, die eine Forms-Migration planen: „Schließen sich diese beiden Architektur-Ansätze/Paradigmen gegenseitig aus?" und „Bedeutet es, dass bei der Migration einer datenbankzentrierten Oracle-Forms-Anwendung die vorhandene PL/SQL-Logik in der Datenbank komplett zu verwerfen ist und im Application-Server neu implementiert werden muss?"

Ausgangslage, Voraussetzungen und Schlüssel-Anforderungen

Die vorangegangenen Fragen stammen aus einem Migrationsprojekt, bei dem die Triestram & Partner GmbH ihr Forms-basiertes Standard-Software-Produkt „lisa.lims" auf eine Java-Enterprise-Architektur umgestellt hat. Dabei handelt es sich um ein Labor-Informations- und Managementsystem, das bis zur Version 9 auf den klassischen Oracle-Technologien basierte, nämlich auf der Datenbank sowie Forms und Reports. Das System ist nach den Architektur-Empfehlungen von Oracle gestaltet worden, das heißt die Domänen beziehungsweise die Business-Logik steckt in Form von PL/SQL-Logik in der Datenbank. Dies zeigt sich in folgendem Mengengerüst:

- 150 Forms
- 250 Reports
- 400 PL/SQL-Datenbank-Packages

Für die Version 10 wurde lisa.lims nach Java migriert. Dabei wurden die Oracle-Datenbank beibehalten und Oracle Forms und Reports durch Java-Open-Source-Komponenten ersetzt:

- Die Eclipse Rich Client Platform (RCP) beziehungsweise die Eclipse Rich Ajax Platform (RAP) für die Benutzer-Oberfläche
- Die Eclipse Business Intelligence and Reporting Tools (BIRT) für die Berichte

Investitionsschutz

Bei der Auswahl der Migrations-Strategie spielte der große Umfang der vorhandenen PL/SQL-Logik in der Datenbank eine entscheidende Rolle, denn sie stellt eine beachtliche Investition von etwa 80 bis 150 Personen-Jahren dar. Um diese Investition zu schützen, war es nicht sinnvoll, die Business-Logik einer Big-Bang-Migration zu unterziehen und sie „auf der grünen Wiese" in Java auf der Mittelschicht neu zu implementieren. Das wäre wirtschaftlich und zeitlich nicht sinnvoll gewesen. Daher hatte die Wiederverwendung des vorhandenen PL/SQL eine zentrale Bedeutung in der Liste der Anforderungen an die migrierte Anwendung. Die folgenden Abschnitte zeigen, welche technische Rolle die vorhandene PL/SQL-Logik in der alten und der migrierten Anwendung spielt.

Mehrbenutzer-Fähigkeit und PL/SQL

lisa.lims ist ein Mehrbenutzer-System, in dem jeder einzelne Nutzer dediziert an der Datenbank angemeldet wird (siehe Abbildung 1). Die PL/SQL-Logik in der Datenbank verwendet den Session-Kontext des angemeldeten Nutzers, unter anderem, um

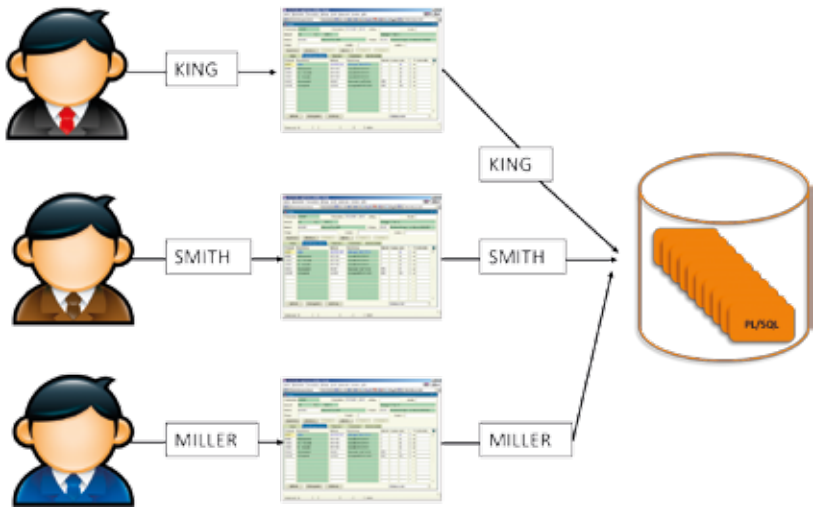


Abbildung 1: PL/SQL und Mehrbenutzerfähigkeit

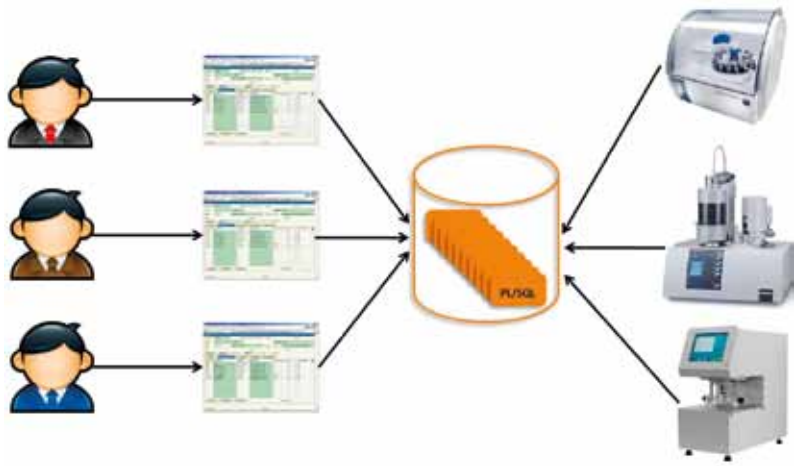


Abbildung 2: PL/SQL als Schnittstelle für Sub-Systeme

mittels Package-Variablen die Datenbank-Änderungen nachzuverfolgen, die der Nutzer durchgeführt hat. Um diesen Mechanismus weiterhin nutzen zu können, muss das migrierte System ebenfalls in der Lage sein, jedem angemeldeten Benutzer eine individuelle Datenbank-Verbindung zuzuweisen.

Anbindung von Sub-Systemen an die Datenbank

Die vorhandene PL/SQL-Logik dient unter anderem als Schnittstelle, über die Labor-Geräte schreibend auf die Datenbank zugreifen und so dort ihre Messdaten speichern. Damit dient das vorhandene PL/SQL in der Datenbank auch als API für den schreibenden Zugriff auf die Tabellen: Es stellt die Konsistenz von Schreibzugriffen auf die Datenbank sicher, auch von solchen, die außerhalb der Applikation erfolgen, wie zum Beispiel externe Geräte mit Datenbank-Schnittstelle (siehe Abbildung 2).

Erfolgen zusätzlich schreibende Zugriffe direkt auf die Tabellen der Datenbank, also etwa vom Application-Server aus, dann ist unbedingt sicherzustellen, dass dabei die betroffenen Datensätze für die Dauer der Transaktion gesperrt werden (Pessimistic Locking), damit die Konsistenz der Daten insgesamt gewahrt bleibt.

Technische Anforderungen an die migrierte Anwendung in Bezug auf PL/SQL

Aus den geschilderten Rahmenbedingungen ergeben sich folgende technische Anforderungen an die migrierte Anwendung in Bezug auf das vorhandene PL/SQL in der Datenbank. Die migrierte Anwendung muss folgende Aufgaben erfüllen:

- In der Lage sein, in der Datenbank gespeichertes PL/SQL aufzurufen
- Jedem Applikationsbenutzer eine exklusive Datenbank-Verbindung zuteilen, die er über die Dauer der Anwendungssitzung beibehält
- Den Benutzer unter seinem spezifischen Account in der Datenbank anmelden
- Pessimistisches, exklusives Sperren von Datensätzen ermöglichen, wenn schreibende Zugriffe auf die Daten erfolgen

Schwierigkeiten bei der Einbindung von PL/SQL in einer Java-Enterprise-Architektur

In einer Vorstudie wurde unter anderem zunächst die Java-Enterprise-Technologie (JEE) evaluiert. Als weit verbreitete Plattform kam dabei der JBoss Applikations-Server mit Hibernate als OR-Mapper zum Einsatz. Nach der Entwicklung eines ersten Prototypen wurde außerdem versucht, ihn auf dem GlassFish Applikations-Server zu betreiben. Es stellte sich heraus, dass die Umsetzung der genannten Anforderungen mittels der Java-Enterprise-Architektur aus folgenden Gründen nicht möglich war:

1. *Inkompatibilität der Schichten-Trennung*
   Der JEE-Ansatz sieht vor, dass die Geschäftslogik grundsätzlich auf der Mittelschicht im Application-Server implementiert wird – und zwar im EJB-Container durch Sessions-Beans beziehungsweise Entity/Persistent Beans, deren Geschäftsdaten mithilfe von Container Managed Transactions in die Datenbank geschrieben werden. Dieser mittelschichtzentrierte Architektur-Ansatz steht im grundsätzlichen Gegensatz zum datenbankzentrierten Ansatz, bei dem Geschäftslogik ganz oder teilweise in der Datenbank-Schicht implementiert ist. Dennoch wurde die Machbarkeit der Migration unter Verwendung der JEE-Plattform geprüft, nicht zuletzt deshalb, weil JEE bereits weit verbreitetet ist und es sich um einen etablierten Standard handelt. Weitere Probleme führten jedoch letztendlich zu der Entscheidung, JEE für die Migration der Software nicht zu verwenden.
2. *Container Managed Persistence*
   JEE sieht für die Geschäftslogik in Session Beans normalerweise vor, dass Datenbank-Transaktionen durch den EJB-Container verwaltet werden. Das bedeutet, dass die Anwendung keine Kontrolle darüber hat, wann genau Daten-Änderungen in die Datenbank geschrieben werden. Das erschwert die Synchronisation mit schreibenden Zugriffen auf die Datenbank von außerhalb des Applikations-Servers, etwa durch PL/SQL in der Datenbank selbst. Dies kann die Daten-Konsistenz gefährden. Außerdem sind dadurch lang laufende Transaktionen kaum umsetzbar, die durch den Anwender mit Commit oder Rollback abgeschlossen werden. Eine Alternative zur Container Managed Persistence bieten die sogenannten „User-Transactions". Hier kontrolliert die Anwendung beziehungsweise der Anwender die Transaktion explizit – und zwar so, dass bei schreibenden Operationen an Daten die entsprechenden Datensätze in der Datenbank auch gesperrt werden. Dies würde ermöglichen, dass Schreibzugriffe auf die Datenbank vom Applikations-Server aus und von den PL/SQL-Modulen in der Datenbank erfolgen, ohne dass dabei die Gefahr von Daten-Inkonsistenzen besteht. Bei den meisten Applikations-Servern war es zum Zeitpunkt der Technologie-Studie möglich, die automatische Transaktionsverwaltung des Servers abzuschalten und User-Transactions zu verwenden. Jedoch war diese Konfiguration mit Nebenwirkungen behaftet (etwa Fehler bei der Persistierung von Session Beans) und zudem herstel-

lerspezifisch, was die Portierbarkeit der Anwendung stark einschränkt und somit einen wesentlichen Vorteil von Java EE verspielt.

3. *Connection Pooling*
   In der JEE-Welt basiert die Verbindung zur Datenbank standardmäßig auf dem Prinzip des „Connection Pooling“: Der Applikations-Server startet eine definierte Anzahl von Verbindungen zur Datenbank, die ständig offen gehalten werden, um den Overhead des Verbindungsaufbaus zu minimieren. Führt der Benutzer in der Anwendung eine Aktion in der Datenbank aus, so nutzt der Applikations-Server dazu eine der Verbindungen aus dem Pool und gibt die Verbindung am Ende der Aktion wieder in den Pool zurück, wo sie dem nächsten Applikations-Benutzer zu Verfügung steht. Dieses Konzept hat zwei Folgen:

   - Alle Benutzer der Applikation nutzen ein- und denselben Datenbank-Benutzernamen. PL/SQL-Logik in der Datenbank, die den Session-Kontext des Datenbank-Benutzers verwendet oder Änderungsverfolgung mithilfe der SQL-Funktion „USER“ betreibt, ist damit nicht mehr verwendbar. Eine potenziell aufwändige Anpassung der Packages wäre die Folge.
   - Um die vorhandene PL/SQL-Logik in der Datenbank wiederverwenden zu können, kommen für lisa.lims nur solche Applikations-Server in Frage, die in der Lage sind, eine Datenbank-Funktion in der gerade aktiven Datenbank-Session des Benutzers auszuführen. Jedoch nicht alle evaluierten Server waren dazu in der Lage, was von der Implementierung der Java-Persistence-API (JPA) des jeweiligen Servers abhängt. Ein Austausch der JPA-Implementierung ist zwar oft möglich, wird jedoch dann nicht mehr durch den Hersteller-Support abgedeckt.

4. *Herstellerspezifische Erweiterungen des Applikations-Servers*
   Die Spezifikation der Java-Enterprise-Architektur bezieht sich auch auf den Applikations-Server. Oracle/Sun hat mit GlassFish eine Referenz-Implementierung vorgelegt. Andere Hersteller von Applikations-Servern weichen von diesem Standard ab, teilweise dadurch, dass sie spezifische Erweiterungen der Funktionalität bieten. Für die migrierte Anwendung relevante Spezifika fanden sich beim JBoss Application Server zum Beispiel im Bereich der Datensatz-Sperren. Da die migrierte Applikation jedoch auf möglichst vielen Applikations-Servern lauffähig sein soll, ist die Nutzung solcher Hersteller-Spezifika strikt zu vermeiden.

Am Ende der Technologie-Studie zeigte sich deutlich: Eine JEE-konforme Architektur unter Verwendung eines JEE-konformen Applikations-Servers stand einer der Schlüssel-Anforderung bei der Migration im Wege, nämlich der Wiederverwendung der vorhandenen PL/SQL-Logik in der Datenbank der migrierte Anwendung. So entstand der Plan, einen eigenen kleinen „Application Server“ auf Basis des Spring-Frameworks und des Objekt-relationalen Mappers EclipseLink zu konstruieren. Spring dient in dieser Architektur als Dependency-Injection-Framework und der Spring-HTTP-Invoker-Mechanismus als Er-

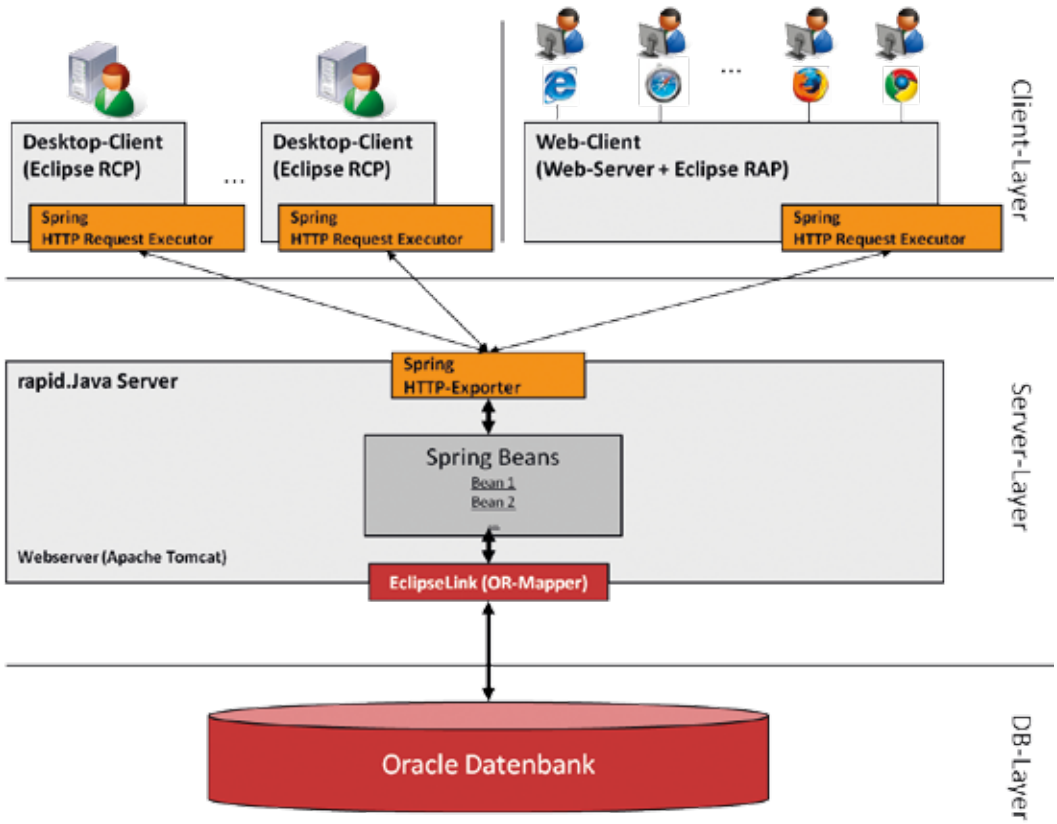


Abbildung 3: Remoting mithilfe des Spring-Frameworks

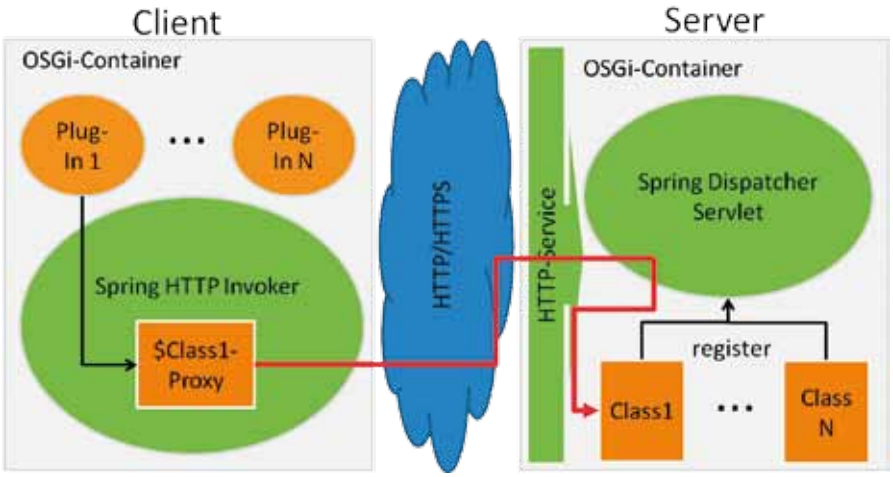


Abbildung 4: Remote-Zugriff auf Logik im Application-Server mittels Spring

satz für die dann nicht mehr verwendbare JEE-Remoting-Technologie (JNDI).

Aufgrund der beschriebenen Ergebnisse der Technologie-Studie hat sich t&p für folgenden Ansatz entschieden (siehe Abbildung 3). Dabei implementieren die Spring-Beans auf der Mittelschicht die Zugriffe auf die Datenbank. Das Dispatcher-Servlet des Spring-Frameworks bindet die einzelnen Spring-Beans dediziert an die einzelnen HTTP-User-Sessions, die in einem Servlet-fähigen Webserver laufen. Der jeweilige Client greift auf seine Web-Session mithilfe des Spring-HTTP-Invokers zu. So gelang es, auf der Server-Seite eine der EJB-Technologie ähnelnde Struktur zu schaffen, die das Veröffentlichen von Fach-Objekten für den Aufruf auf der Client-Seite ermöglicht (siehe Abbildung 4).

Verwendung des Objekt-relationalen Mappers EclipseLink

EclipseLink, zurzeit in der Version 2.2 in Verwendung, ermöglicht folgende Funktionen:

- Datensätze in der Datenbank pessimistisch zu sperren
- PL/SQL in der Datenbank aufzurufen
- jedem Applikations-Benutzer eine exklusive Datenbank-Verbindung zuzuteilen und den Benutzer unter seinem spezifischen Account in der Datenbank anzumelden

Damit erfüllt EclipseLink alle technischen Anforderungen, die erforderlich sind, um das vorhandene PL/SQL in der neuen Java-Anwendung wiederzuverwenden. Darüber hinaus implementiert EclipseLink die JPA-Spezifikation, sodass auch viele nützliche JPA-Features genutzt werden können. Für die JPA-Spezifikation 2.0 ist EclipseLink die Referenz-Implementierung. Da EclipseLink auf dem OR-Mapper „Toplink" von Oracle basiert, ist es sogar möglich, einige herstellerspezifische Funktionen der Oracle-Datenbank zu nutzen. Da eine der Grund-Anforderungen darin bestand, die vorahandenen PL/SQL-Packages in der Datenbank unverändert weiter zu nutzen, stand ein Wechsel der Datenbank-Plattform beziehungsweise Datenbank-Unabhängigkeit außer Frage. Damit war EclipseLink für dieses Vorhaben die erste Wahl.

Fazit

Der Ansatz verleiht der migrierten Java-Anwendung die datenbankrelevanten Eigenschaften, die in der Welt von Oracle-Forms-Anwendungen etablierter und geschätzter Standard sind: die Mehrbenutzer-Fähigkeit inklusive des pessimistischen DML-Sperrverhaltens und der individualisierten Anmeldung an der Datenbank sowie die Integration von vorhandener PL/SQL-Logik. Diese Features sind für eine Datenbank- beziehungsweise OLTP-Anwendung grundlegend.

Die Kombination von EclipseLink mit dem Remoting des Spring-Frameworks ermöglicht es, einen Applikations-Server zu implementieren, der die zuvor genannten Datenbank-Anforderungen erfüllt und dabei ohne JEE-Overhead auskommt. Diese Lösung zeichnet sich dadurch aus, dass sie leichtgewichtig und äußert flexibel ist: Der Server nutzt OSGi als Laufzeit-Umgebung und kann damit sowohl Stand-Alone laufen, als auch in jeden Servlet-fähigen Web- beziehungsweise Applikations-Server als Standard-Webanwendung (WAR) eingesetzt werden. Somit ist die gewählte Architektur unabhängig von den Spezifika einzelner Applikations-Server, es wird lediglich ein Servlet-Container benötigt.

Die Integration von vorhandener PL/SQL-Datenbank-Logik bedeutet einen Schutz der bereits getätigten Investition und einen geringeren zeitlichen und finanziellen Aufwand bei der Migration.

Der Ansatz ermöglicht eine sogenannte „smart migration" – und zwar im Sinne eines schrittweisen, fließenden Übergangs von der alten auf die neue Technologie-Plattform. Die Grundlage hierfür ist die Wiederverwendung der vorhandenen PL/SQL-Business-Logik. Einzelne, besonders zentrale Dialoge können dabei bereits auf die neue Technologie für die Benutzer-Oberfläche umgestellt werden, während andere Dialoge zunächst weiter auf Oracle Forms basieren. So können Alt und Neu koexistieren, bis die gesamte Migration abgeschlossen ist. Anschließend kann die Neu-Implementierung der Business-Logik in Java ins Auge gefasst werden. Somit verbindet der vorgestellte Ansatz technisch das Beste aus zwei Welten – nämlich die Beibehaltung der Transaktions-orientierten Features auf Seiten der Oracle-Datenbank mit einer gleichzeitig plattformunabhängigen, leichtgewichtigen und flexiblen Java-Architektur.

Björn Christoph Fischer
info@t-p.com

Björn Christoph Fischer ist als Senior Software Engineer spezialisiert auf Java-Technologien und -Lösungen. Er ist einer der Architekten des „rapid.Java development framework" – ein Java-Framework für das Rapid Application Development und die schrittweise Migration von Oracle-Forms-Anwendungen nach Java.

Oracle Forms 11*g* – Debugging, Statusmeldungen und Maskensteuerung durch eine Erweiterung des Java-Timers

Frank Christian Hoffmann, Cologne Data GmbH

Die Java-Bean „timeout.jar" auf den Download-Seiten von Forms 11.1.2 unter der Rubrik „Demos" wurde im Jahre 2003 von Frank Nimphius als Ersatz für die alte Timer-Funktion der Client/Server-Forms-Version entwickelt. Erweitert um zwei weitere Custom-Events, lässt es sich hervorragend zur Entwicklung einiger neuer und praktischer Funktionen nutzen.

Um bei der Administration von mehreren Hundert Masken, Berichten und Anwendern sowie beim Zugriff auf verschiedene Datenbanken nicht den Überblick zu verlieren, kann es sehr hilfreich sein, wenn im laufenden Betrieb wichtige Parameter protokolliert werden können. Dazu zählen Anmeldeparameter (Wer hat mit welcher Java-Version und von welchem Forms-Server auf welche Datenbank zugegriffen?), Zugriffsverhalten (Welche Masken werden aufgerufen?), angezeigte Fehlermeldungen sowie Reports-Parameter und -Laufzeiten.

Darüber hinaus ist der Eingriff in den laufenden Betrieb denkbar. Anforderungen könnten Wartungshinweise, Hinweise zur Benutzung der Anwendung oder gezielte personifizierte Meldungen sein. Denkbar ist auch ein automatischer Abbruch aller Sessions, um Module zu aktualisieren, eine neue Version einzuspielen oder den Zugriff auf eine vorgegebene Zeit zu beschränken (etwa bei Regelarbeitszeiten von 9 –18 Uhr).

Der Timeout sollte den Benutzer in seiner Session nicht mit einer kryptischen Oracle-Fehlermeldung überraschen. Eine kleine Statuszeile, die rechtzeitig auf den Timeout hinweist, gibt ähnlich wie bei Online-Banking-Applikationen einen guten Überblick über die verbleibende Zeit (siehe Abbildung 1). Es ist auch sinnvoll, wenn der Benutzer, der einen lang laufenden Report im Hintergrund startet, bei Fertigstellung einen Hinweis bekommt.

Denkbar ist auch ein Security-Feature. Man könnte für jeden Benutzer ein Profil erstellen und ihm nur bestimmte Module zuordnen. Wenn Masken-Aufrufe protokolliert werden, lassen sich noch vor Anzeige der Daten die Rechte prüfen und Zugriffe verhindern. Theoretisch kann aber auch ein Datenbank-Job die Datenbank-Sessions mit den Forms-Sessions in der Protokoll-Tabelle vergleichen und unerwünschte Zugriffe auf die Datenbank unterbinden oder davor warnen.

Eine Hotline wird dankbar sein, wenn aktuelle Zugriffe, Fehler und Probleme jederzeit abrufbar sind. Die Möglichkeit, präventiv zu agieren, soll die Qualität der Anwendung steigern. Hinzu kommt die Analyse von Ausfallzeiten. Möglicherweise können Kapazitäts-Engpässe auftreten, wenn eine bestimmte Anzahl von Nutzern eine bestimmte Maske aufruft. Solche Szenarien lassen sich bequem mit den Protokoll-Tabellen analysieren.

Lösungsansatz und praktische Umsetzung

Für die Protokollierung bei Maskenaufruf und Applikationsstart lässt sich die bekannte Triggerlogik von „PRE-FORM" oder „WHEN-NEW-FORM-INSTANCE" nutzen. Für alle Aktionen, bei denen in einem regelmäßigen Zeitintervall Aktionen gestartet werden sollen, bietet sich die modifizierte Timeout-Bean an. Der Autor hat diese so umprogrammiert, dass sie jede Minute eine Aktion auslöst – unabhängig von der Gesamtzeit, in der sie läuft. Die Umsetzung sollte von einem erfahrenen Entwickler durchgeführt werden. Die nun folgenden Codebeispiele verstehen sich als eine Anregung zur Umsetzung. Es beginnt mit der Aktivierung der neuen Funktionen aus „webforms.cfg".

Debugging-Funktionen sollten an- und ausschaltbar sein. Dafür können neue Applet-Parameter in die Konfigurationsdatei „webutiljpi.html" (siehe Listing 1) sowie im Embedded-Bereich (siehe Listing 2) eingestellt werden. Listing 3 zeigt, wie die Funktionen in „webforms.cfg" gesteuert werden.

Der Parameter „Timer-Debug" ermöglicht ein Debugging der wichtigsten Aktionen des „timeout.jar" über die Java-Konsole. Diese Debugging-Funktion hat Frank Nimphius in die Timeout-Java-Bean integriert. Für das Debugging bieten sich drei Level an:

Abbildung 1: Hinweis auf den Timeout

```
<PARAM NAME="report_server_name" value="%report_server_name%">
<PARAM NAME="forms_session_timeout" value="%forms_session_timeout%">
<PARAM NAME="timer_debug" value="%timer_debug%">
<PARAM NAME="forms_debug" value="%forms_debug%">
```

Listing 1

```
report_server_name="%report_server_
name%"
forms_session_timeout="%forms_sessi-
on_timeout%"
timer_debug="%timer_debug%"
forms_debug="%forms_debug%"
```

Listing 2

```
report_server_name=rptsvr_srv11gr1_as-
inst_1
forms_session_timeout=50
forms_debug=level_2
timer_debug=true
```

Listing 3

```
// NEW FH 09.2011
public static final ID ceInActivityExceeded =
ID.registerProperty(„INACTIVITY_EXCEE-
DED");
public static final ID ceActivityExceeded   =
ID.registerProperty(„ACTIVITY_EXCEE-
DED");
```

Listing 4

```
ON-ERROR-TRIGGER:
num_errcod NUMBER := ERROR_CODE;
vch_errtyp VARCHAR2(3) := ERROR_TYPE;
vch_errtxt VARCHAR2(80) := ERROR_TEXT;
BEGIN
 IF (num_errcod in (41355,41351))
            THEN null;
 ELSE
   Message(vch_errtyp||'-'||to_char(num_err-
cod)||': ,||vch_errtxt);
   RAISE Form_Trigger_Failure;
 END IF;
END;
```

Listing 6

```
if webutil_global.get_logout_parameter = 'J'
then
   exit_form(no_validate);
end if;
```

Listing 7

```
// NEW FH 09.2011
// Customevent "ActivityExceeded" fires if a useraction occured during the sleeptime
//

if ((current_time - last_user_action) < (60/1000))
  {
  try{
  // raise custom Forms event
  CustomEvent ce = new CustomEvent(mHandler,ceActivityExceeded);
  dispatchCustomEvent(ce);
  }
  catch (Exception e)
  {
    write_message("Error occured when raising custom event = e.getMessage()");
  }
  write_message("Timer not expired .. Useractivity recognized");
  }
else
  //
  // NEW FH 09.2011
  // Customevent "NoActivityExceeded" fires if no Useraction occured during the sleeptime
 //
 {
  try{
  // raise custom Forms event
  CustomEvent ce = new CustomEvent(mHandler,ceInActivityExceeded);
  dispatchCustomEvent(ce);
  }
  catch (Exception e)
  {
   write_message("Error occured when raising custom event = e.getMessage()");
  }
  write_message("Timer: No Useractivity recognized");
  }
```

Listing 5

```
-- Timeout Activation, zum sofortigen Auslö-
sen des ersten Timerevents
-- Der Timer wird auf 1 Sekunde gesetzt um
bei Aufruf der Maske
-- sofort ausgelöst zu werden
set_custom_property(,WEBUTIL.WEBU-
TIL_TIMER_FUNCTIONS',1,'RECORDING_
EVENTS','all');
set_custom_property(,WEBUTIL.WEBU-
TIL_TIMER_FUNCTIONS',1,'ENABLE_
DEBUGGING',p_debug);
set_custom_property(,WEBUTIL.WEBU-
TIL_TIMER_FUNCTIONS',1,'TIMER_SLEEP_
TIME','1'); --sek
set_custom_property(,WEBUTIL.WEBU-
TIL_TIMER_FUNCTIONS',1,'START_TI-
MER','1'); --min
```

Listing 8

```
-- Standard Message
 webutil_separateframe.settitle
(,Informationssystem ,
  || , | Angemeldet: ,
  || get_application_property(USERNAME)
|| , auf ,
  || upper(webutil_local_parameter.db_apps-
rv)
  || , | Datum: , || to_char(sysdate,'DD.
MM.YYYY HH24:MI')
  || ' | Die Anwendung schließt bei Inaktivität
in '
  || to_char(num_timeout_par) || ,
Minute(n)');
```

Listing 9

- *Level_0*
  ausgeschaltet
- *Level_1*
  Protokoll von Aufrufparametern und Fehlermeldungen
- *Level_2*
  Protokoll aller Meldungen

Die Applet-Parameter lassen sich zur Laufzeit aus Forms über „webutil_browser.getappletparameter(‚forms_debug')" abfragen.

Technische Modifikation des Timeout-JAR

Neben dem bestehenden Custom-Event

```
if pdr_webutil_blobtab.f_exist_erstellte_dok(user)
  then
    -- MDI-Icon anzeigen
    set_menu_item_property(,MDI_TOOLBAR.MDI_DOKUMENTEN_ABLAGE‘,
                           ENABLED,PROPERTY_TRUE);
    set_menu_item_property(,MDI_TOOLBAR.MDI_DOKUMENTEN_ABLAGE‘,
                           VISIBLE,PROPERTY_TRUE);
  else
    -- MDI-Icon ausblenden
    set_menu_item_property(,MDI_TOOLBAR.MDI_DOKUMENTEN_ABLAGE‘,
                           VISIBLE,PROPERTY_FALSE);
end if;
```

Listing 10

„MAX_INACTIVITY_EXCEEDED“ hat der Autor seine modifizierte Version um zwei weitere Events ergänzt: „INACTIVITY_EXCEEDED“, und „ACTIVITY_EXCEEDED“. Dafür wurden zwei zusätzliche Variablen angelegt (siehe Listing 4), in die Ausführungsschleife der Ursprungs-Bean ein paar Zeilen Code ergänzt und hinter „if ((current_time – last_user_action) < (max_inactivity))“ Code eingefügt (siehe Listing 5).

Damit werden in Forms drei verschiedene Zustände ausgelöst. Wenn der Timer startet, wird neben dem bisherigen Event alle 60 Sekunden eine Information an die Aktivitäten des Anwenders übermittelt. Hat der Anwender keine Maus-, Fenster- oder Tastatur-Bewegungen vollzogen, meldet die Java-Bean „INACTIVITY EXCEEDED“. Hat er in den letzten 60 Sekunden eine Aktion vollzogen, erfolgt die Meldung „ACTIVITY_EXCEEDED“. Diese neue „60-Sekunden-Heartbeat“-Logik lässt eine kontinuierliche Kommunikation zur Applikation zu.

Es ist sinnvoll, die zusätzlichen Objekte in eine OBJEKT-Bibliothek (OLB-File) zu legen und als Unterklasse in die Module einzubinden. In einer Bibliothek (pll) sollten alle aufzurufenden Prozeduren und Funktionen hinterlegt und dokumentiert sein. Somit lassen sich Anpassungen später strukturiert vornehmen. Der Aufwand für die Integration der neuen Features ist überschaubar. Vier Trigger und ein Parameter sind zu erstellen beziehungsweise anzupassen:

1. Eine Forms-Parameter-Variable als Zähler für den Timeout.
2. Ein Trigger „WHEN-CUSTOM-ITEM-EVENT“ für die neue Timeout-Java-Bean mit Aufruf der gewünschten neuen Feature-Funktionen (Beispielcode siehe unten)

```
-- Ermittlung ob Nachrichten für den Anwender vorliegen
  select count(*) counter
    into num_message_cnt
    from webutil_msg
   where msg_to in (USER,‘ALL‘)
     and sysdate between msg_actiontime_
from and msg_actiontime_to
     and msg_message_id not in
     (select message_id
        from webutil_log
       where user_session_id =
userenv(,SESSIONID‘)
         and message_id = msg_message_id);
```

Listing 11

3. Ein Error-Handling, um die Forms-Fehlermeldungen „41355“ und „41351“ auszublenden. Dies ist notwendig, um bei der Nutzung von „CALL_FORM“ die Fokusfehler des „timeout.jar“ zu unterdrücken (siehe Listing 6).
4. Ein Trigger „WHEN-WINDOW-ACTIVATED“, um nach Maskenabbruch ein potenzielles Handling für die Vorgängermaske zu aktivieren. Das ist wichtig, um einen Gesamt-Applikationsabbruch möglich zu machen (siehe Listing 7).
5. Die für die Startmaske(n) bei der Nutzung von „webutil“ übliche(n) Verzögerung(en) zum Laden der Java-Beans (siehe „webutil“-Demomaske).
6. Die Initialisierung des Java-Timers über den Trigger „WHEN-NEW-FORM_INSTANCE“ (siehe Listing 8).

Code-Beispiele für neue Funktionen

Jede Minute lassen sich aktualisierte Statusmeldungen mit User, Datenbank, Forms-Server, Uhrzeit und Inaktivitäts-Timer anzeigen. Voraussetzung ist, dass die Masken über die Einstellung in „formsweb.cfg“ mit „separateFrame=true“ laufen. Die Einbindung erfolgt über den Block und das Item, in dem die Timeout-Java-Bean mithilfe des Triggers „WHEN-CUSTOM-ITEM-EVENT“ integriert wurde (siehe Listing 9).

„webutil_local_parameter.db_appsrv“ wurde als globale Package-Variable für die Anmelde-Informationen angelegt. Die Variable „num_timer_par“ ist die lokale Variable für die Restzeit des Timer-Parameters.

Ein weiteres Beispiel ist die Ausgabe von Reportstatus-Informationen über erstellte Langzeitreports. Voraussetzung für die minütliche Abfrage zur Anzeige von Langzeitreports ist, dass die Oracle-Report-Statustabellen in der Datenbank betrieben werden. Die Anzeige eines fertigen Reports erfolgt über ein Icon in der MDI-Statusbar (siehe Listing 10).

```
Insert into webutil_msg
(
msg_to,
msg_txt,
msg_actiontime_from,
msg_actiontime_to
)
values
(
‘ALL’,
‘Heute steht das Forms-System ab 18:00 wegen Wartungsarbeiten nicht zur Verfügung!’,
to_date(‘21.02.2012 09:00’,’DD.MM.RRRR
HH24:MI’),
to_date(‘21.02.2012 20:00’,’DD.MM.RRRR
HH24:MI’)
);
Insert into webutil_msg
(
msg_to,
msg_action
msg_txt,
msg_action,
msg_actiontime_from,
msg_actiontime_to
)
values
(
‘ALL’,
‘Abbruchsignal’,
‘EXIT’,
to_date(‘21.02.2012 18:00’,’DD.MM.RRRR
HH24:MI’),
to_date(‘21.02.2012 20:00’,’DD.MM.RRRR
HH24:MI’)
);
```

Listing 12

```
if get_msg.msg_action = 'EXIT'
  then
    webutil_global.set_logout_parameter('J');
    exit_form(no_validate);
end if;
```

Listing 13

In diesem Beispiel wurden die Dokumente über das Report-Plug-in „BLOBDestination" in die Tabelle „webutil_blobtab" geschrieben. Eine Package-Funktion greift über eine Funktion auf diese Tabelle zu und prüft sie auf nicht abgeholte Berichte.

Das dritte Beispiel betrifft die Anzeigen von personalisierten oder globalen Nachrichten alle 60 Sekunden (siehe Listing 11). Um Nachrichten nur einmal pro Session anzuzeigen, wird in der LOG-Tabelle geprüft, ob die freigegebene Nachricht schon abgerufen wurde.

Ein möglicher Einsatz wäre, die Wartung einer Applikation oder einer Maske anzukündigen und danach die Maske zu schließen oder zu sperren (siehe Listing 12).

Über das Abbruch-Signal „EXIT" kann auch der Abbruch aller Forms-Sessions erfolgen. Die globale Package-Variable sorgt dafür, dass auch alle über „CALL_FORM" geöffneten Masken beendet werden (siehe Listing 13).

```
vch_eventName      := name_in('system.custom_item_event');

   -- Das Event "MAX_INACTIVITY_EXCEEDED" wird ausgelöst,
   -- wenn in den letzten 60 Sekunden auf dem Applet
   -- KEINE Benutzer-Aktivitäten wie Mausbewegungen, Klicks,
   -- Tastatureingaben oder Fensterbewegungen
   -- registriert wurden und der Timer mit der Restzeit auf
   -- mehr als 1 Minute stand.

IF (vch_eventName='MAX_INACTIVITY_EXCEEDED') THEN … END IF;
IF (vch_eventName='ACTIVITY_EXCEEDED') THEN … END IF;

-- zum Aktivieren des nächsten EVENTS
set_custom_property('WEBUTIL.WEBUTIL_TIMER_FUNCTIONS',1,'TIMER_SLEEP_TIME','60');
-- Sekunden
set_custom_property('WEBUTIL.WEBUTIL_TIMER_FUNCTIONS',1,'START_TIMER','60'); --
Minuten
```

Listing 14

Zum Abschluss ein Beispiel dafür, wie die Aktionen über den Trigger „WHEN-CUSTOM-ITEM-EVENT" verankert an die Timeout-Java-Bean umgesetzt werden könnten. Dieser Code sollte in einer Bibliothek (PLL) hinterlegt und aus allen Masken referenziert sein (siehe Listing 14).

Der nichtsignierte Java-Code des neuen „timeoutPJV.jar" können nach Erscheinen über die Webseite www.cologne-data.de/oracle.html abgerufen werden.

Frank Christian Hoffmann
fch@cologne-data.de

Frank Christian Hoffmann ist Dipl.-Ing. Informationsverarbeitung an der GH Paderborn. Seine Schwerpunktthemen sind Oracle-Datenbanken und 4GL-Software-Entwicklung. Er war von 1994 bis 1999 Berater beziehungsweise Seniorberater bei Opitz Consulting und Oracle Deutschland und ist seit 1999 Geschäftsführer der Cologne Data GmbH.

Linked Data – ein Web aus Daten

Angelo Veltens, http://datenwissen.de

Wer heute im Web Daten abrufen möchte, hat es nicht leicht. Obwohl hinter vielen Diensten große Datenbanken stehen, beschränkt sich das Surfen im Web meist noch auf menschenlesbare Webseiten. Entwickler können bestenfalls den mühsamen Umweg über proprietäre APIs gehen. Mit Linked Data werden die Daten Teil des Webs und das dokumentenbasierte World Wide Web erweitert sich um ein Web aus Daten.

Trotz aller modernen Web-Anwendungen und -Services ist das World Wide Web (WWW) heute immer noch ein Web aus Dokumenten. Diese mögen dynamisch vom Server erzeugt sein und Megabytes an JavaScript enthalten, sie mögen XML-Daten oder JSON-Objekte beinhalten – dennoch sind es Dokumente. Dabei ist das WWW voller Daten. Hinter praktisch jeder Webseite verbirgt sich heute eine Datenbank, sei es das private Wordpress-Blog oder ein großer Online-Shop. Wir sitzen auf wahren Datenschätzen. Doch weil wir sie hinter menschenlesbaren Web-Seiten und heterogenen Web-Services verstecken, bleibt ihr Nutzen verhältnismäßig gering. Sie sind gefangen in Daten-Silos und nur über die vom Betreiber festgelegten Möglichkeiten abrufbar. Mithilfe von Linked Data können wir sie aus ihren Silos befreien und ein globales Web aus Daten schaffen, das einen grundlegend neuen Umgang mit Daten ermöglicht.

Von Daten im Web …

HTML-Dokumente liegen nicht einfach „im Web" – sie sind das Web. Sie bilden es dadurch, dass wir sie miteinander über ihre URLs verlinken. Wir könnten stattdessen Textdateien in ZIP-Archive verpacken und auf einem FTP-Server zum Download anbieten. Das wäre jedoch kein Web. Kurioserweise tun wir aber genau das mit unseren Daten. Wir verpacken sie fein säuberlich und bieten sie zum Download an oder verstecken sie in HTML-Dokumenten. Der Weg über einen Web-Service ist der erste Schritt in die richtige Richtung, stellt uns allerdings dennoch vor einige Probleme.

Angenommen, jemand arbeitet an einer Anwendung, die auf mehrere verschiedene Web-Services zugreifen soll. Nehmen wir als Beispiel einen Dienst, der Nutzern die Bewertung von Hotels ermöglichen soll. Die Anmeldung soll über einen Facebook-Account möglich sein. Welche Hotels es gibt, ermitteln wir über einen Web-Service („Hotel-Service") und Informationen über den Urlaubsort, an dem das Hotel steht, fragen wir über einen weiteren Web-Service ab („Geo-Service").

Um diese Anwendung zu realisieren, müssen wir gegen drei Web-Services programmieren, also gegen drei verschiedene APIs mit drei verschiedenen Datenformaten. Wir müssen drei verschiedene API-Dokumentationen verinnerlichen und unsere Anwendung auf diese APIs zuschneiden. Hinzu kommen die Probleme der Verlinkung: Angenommen, der Hotel-Service liefert uns einen JSON-Datensatz zu einem Hotel (siehe Listing 1).

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

Listing 1

Wie gelangen wir, ausgehend von diesem Datensatz, zu den im Geo-Service verfügbaren Informationen über den Urlaubsort? Es gibt keinen Link. Der Datensatz enthält lediglich die Zeichenkette „Berlin". Die passende Anfrage an den Geo-Service müssen wir als Entwickler mithilfe unseres menschlichen Hintergrundwissens und des Studiums der API-Dokumentation selbst herstellen. Eine automatische Verknüpfung der Daten ist nicht möglich.

Die Daten liegen zwar im Web, sind jedoch nicht Teil des Webs. Sie sind abrufbar, aber nicht miteinander verlinkt. Ihr Kontext geht nicht aus ihnen hervor und ihre Bedeutung erschließt sich nur über die jeweilige API-Dokumentation. Ein Entwickler muss sich jedem der drei Dienste separat widmen und die Verknüpfungen anschließend selbst herstellen. Und warum sollte er sich überhaupt auf drei Dienste beschränken? Warum kann er nicht den gesamten öffentlich im WWW verfügbaren Datenbestand für seine Anwendungen nutzen?

… zu einem Web aus Daten

Dieser Schritt zu einem Web aus Daten, durch das wir maschinell ähnlich browsen können, wie wir als menschlicher Nutzer heute das Web aus Dokumenten durchstöbern, wird durch Linked Data möglich. Das Konzept, Daten miteinander zu verlinken, stammt von Tim Berners-Lee, dem Erfinder des WWW. Er hat schon 2006 vier Grundprinzipien von Linked Data aufgestellt [1]. Diese lauten:

1. Nutze Uniform Resource Identifier (URI) als Name für Dinge
2. Nutze HTTP URIs, sodass man diese Namen nachschlagen kann
3. Wenn jemand einen URI abruft, stelle nützliche Informationen mittels Standards (RDF, SPARQL) bereit
4. Füge Links zu anderen URIs ein, sodass man weitere Dinge entdecken kann

Berücksichtigen wir diese Regeln, werden unsere Daten Teil des Webs. Sie sind einfach und verlangen dennoch ein grundlegendes Umdenken im Umgang mit Daten und den

Dingen im Web. Werfen wir nun also einen näheren Blick auf diese Prinzipien.

Alles bekommt einen URI

Ein URI ist eine globale ID. Während im heutigen Web vornehmlich Dokumente – seien es Web-Seiten oder XML-Dokumente – über sie identifiziert und auffindbar gemacht werden, verlangt Linked Data, dass wir alle möglichen Dinge über URIs identifizieren. Dieser Begriff des „Dings" ist essenziell für das Verständnis von Linked Data. Es ist nicht möglich, Produkte, Städte, Personen, Unternehmen und andere „Dinge" im Web zu veröffentlichen. Es ist jedoch möglich, diesen Dingen einen Namen in Form eines URI zu geben.

Wollten wir obiges Beispiel mithilfe von Linked Data realisieren, wären sowohl die Hotels als auch die Urlaubsorte durch einen URI identifizierbar. Wichtig ist, dass diese URIs tatsächlich die Hotels und Orte selbst identifizieren und nicht etwa die Website eines Hotels oder Urlaubsorts. Im „Web of Data" gibt es nicht mehr nur Dokumente, sondern auch „Dinge". Jedes dieser Dinge bekommt einen eigenen URI. Ein Hotel und seine Website sind zwei verschiedene Dinge und somit wird beides durch unterschiedliche URIs identifiziert. Statt von Dokumenten spricht man auch von Informations-Ressourcen, während Dinge, die keine Dokumente sind, als Nicht-Informations-Ressourcen bezeichnet werden.

Das zweite Prinzip verlangt die Verwendung von HTTP-URIs. Dies hat den pragmatischen Hintergrund, dass HTTP-URIs über das Domain-Name-System auflösbar und somit abrufbar sind. Dies gilt beispielsweise nicht für ISBN-URIs, die möglicherweise zur Identifikation von Büchern in Betracht gezogen werden könnten. Da diese URIs jedoch nicht abrufbar sind, ist es nicht möglich, das dritte Prinzip von Linked Data zu erfüllen, nützliche Informationen bei Abruf des URI bereitzustellen.

Dinge und Dokumente

Um überhaupt Informationen bereitstellen zu können, benötigen wir ein Dokument, das Daten enthält. Dieses müssen wir bei Abruf des URI einer Nicht-Informations-Ressource bereitstellen, da letztere sich nicht selbst im Web befinden kann. Nehmen wir an, unter „http://hotels.example/fiktiva" sind Informationen über das Hotel „Fiktiva" abrufbar. Das Hotel selbst muss, da es ein vom Dokument zu unterscheidendes „Ding" ist, über einen eigenen URI identifiziert werden. Um dennoch auf die im Dokument bereitgestellten Informationen zugreifen zu können, empfiehlt sich die Verwendung sogenannter „Hash-URIs" zur Identifikation von Nicht-Informations-Ressourcen. Dabei wird der Fragment-Identifier eines URI genutzt, um die im Dokument beschriebenen Dinge zu identifizieren.

Unser Hotel „Fiktiva" bekäme bei dieser Herangehensweise etwa den URI „http://hotels.example/fiktiva#it". Durch den Fragment-Identifier „#it", der Teil des URI ist, haben wir eine Unterscheidung zur Informations-Ressource mit dem Namen „http://hotels.example/fiktiva" geschaffen. Alternativ könnten wir das Hotel durch „http://hotels.example/hotel/fiktiva" oder einen beliebigen anderen URI identifizieren und mittels HTTP-Status-Code „303" (See Other) eine Weiterleitung zur Informations-Ressource bewirken. Die Verwendung von Hash-URIs hat jedoch den Vorteil, dass sie ohne diese Weiterleitung auskommt, da der Client automatisch den Fragment-Identifier abstreift, bevor er die Anfrage an den Server sendet, und somit selbstständig die Informations-Ressource anfordert.

Aber was sind nützliche Informationen? Nützlichkeit liegt im Auge des Betrachters. Während ein Mensch großen Nutzen an aufbereiteten, interaktiven HTML-Seiten hat, benötigen Dienste, die automatisiert auf ein Angebot zugreifen, maschinenlesbare Daten. Diesem Umstand können wir durch Content-Negotiation gerecht werden. Mithilfe dieser Technik lässt sich abhängig von der Anforderung des Clients eine andere Repräsentation einer Ressource ausliefern. Bleiben wir beim Hotel „Fiktiva" mit dem URI „http://hotels.example/fiktiva#it": Beim Abruf streift der Client den Fragment-Identifier ab und fordert das Dokument „http://hotels.example/fiktiva" an. Weiterhin setzt der Client den HTTP-Accept-Header auf den von ihm bevorzugten MIME-Type, um auszudrücken, welche Art der Repräsentation für ihn nützlich ist. Ein Browser wird beispielsweise „text/html" anfordern, während eine andere Anwendung möglicherweise „text/json" oder „application/rdf+xml" bevorzugt. Der Server versucht den Wünschen des Clients entsprechend, die passende Repräsentation des Dokuments auszuliefern (siehe Abbildung 1).

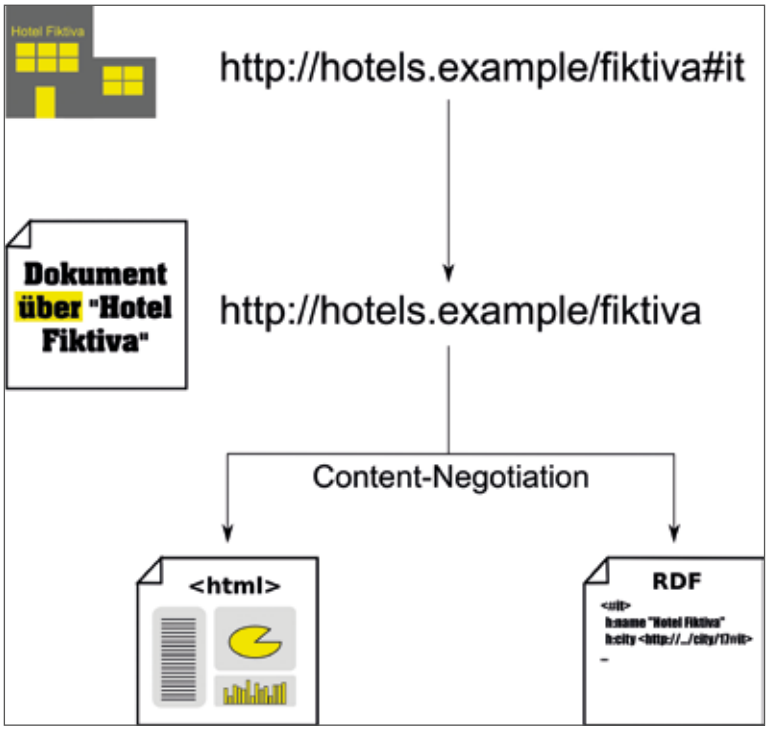


Abbildung 1: Abruf einer Nicht-Informations-Ressource und Content-Negotiation

RDF – die Sprache des „Web of Data“

Mittels Content-Negotiation ist sichergestellt, dass sowohl menschen- als auch maschinenlesbare Informationen bereitgestellt werden können, wenn man den URI einer Nicht-Informations-Ressource abruft. Wir werfen nun einen näheren Blick auf die nützlichen, maschinenlesbaren Daten im Sinne der Linked-Data-Prinzipien. In deren Zentrum steht das Resource Description Framework (RDF). Wie der Name schon sagt, wird es verwendet, um Ressourcen zu beschreiben. Diese können sowohl Informations- als auch Nicht-Informations-Ressourcen sein.

RDF organisiert die Informationen in sogenannten „Tripeln“, bestehend aus Subjekt, Prädikat und Objekt, ähnlich wie in einem einfachen natürlich-sprachlichen Satz. Subjekt ist die Ressource, die beschrieben wird, das Prädikat drückt aus, welche Aussage über diese Ressource getroffen wird, und das Objekt ist das Datum, das durch das Prädikat dem Subjekt zugesprochen wird. Wollen wir zum Beispiel ausdrücken, dass das Hotel „Fiktiva“ 100 Zimmer hat, könnte ein Tripel in der RDF-Turtle-Syntax wie in Listing 2 aussehen.

Der URI des Hotels ist in diesem Beispiel das Subjekt der Aussage, „hotel:anzahlZimmer“ ist das Prädikat, das die Anzahl der Zimmer ausdrückt, und „100“ der Wert der Aussage, also die tatsächliche Anzahl an Zimmern, die wir mit diesem Tripel mitteilen möchten. Ebenso leicht können wir mehrere Aussagen zu diesem Hotel treffen (siehe Listing 3).

Die Aussagen zu einem Subjekt werden durch Semikolon getrennt, die letzte Aussage durch einen Punkt abgeschlossen. Durch Kommas getrennt, können sogar mehrere Objekte einem Prädikat zugeordnet werden. Durch die Unterscheidung zwischen Ding und Dokument ist es sehr leicht, auch Metadaten zu hinterlegen, indem wir Aussagen über den URI des Dokuments (ohne „#it“) treffen (siehe Listing 4).

Dieses Tripel sagt aus, wann das Dokument zuletzt geändert wurde, und nicht etwa, wann das Hotel baulich verändert wurde. Durch die Unterscheidung zwischen Ding und Dokument wird also eine leichte Trennung von Daten und Metadaten möglich.

Zu betonen ist, dass RDF kein Datenformat, sondern tatsächlich ein Modell ist, das keine Syntax vorschreibt. RDF-Tripel können durch unterschiedliche Syntaxen dargestellt werden. Neben der für Menschen leicht zu lesenden, hier verwendeten Turtle-Syntax, gibt es zum Beispiel auch eine XML-Notation. Da es sich bei RDF-Daten letztlich um einen gerichteten Graphen handelt, lassen sie sich auch als solcher darstellen (siehe Abbildung 2). Man beachte, dass die Aussage „hotel:ort“ nicht auf ein Objekt-Literal, sondern auf einen anderen URI verweist. Wir setzen dadurch einen Link zu einer anderen Ressource und verlinken Daten.

```
<http://hotels.example/hotel/fiktiva#it>
  hotel:anzahlZimmer “100“.
```

Listing 2

```
<http://hotels.example/hotel/fiktiva#it>
  hotel:name “Hotel Fiktiva“;
  hotel:anzahlZimmer “100“;
  hotel:ort <http://sws.geonames.
org/2945024/>.
```

Listing 3

```
<http://hotels.example/hotel/fiktiva>
  dc:modified “2012-09-30“.
```

Listing 4

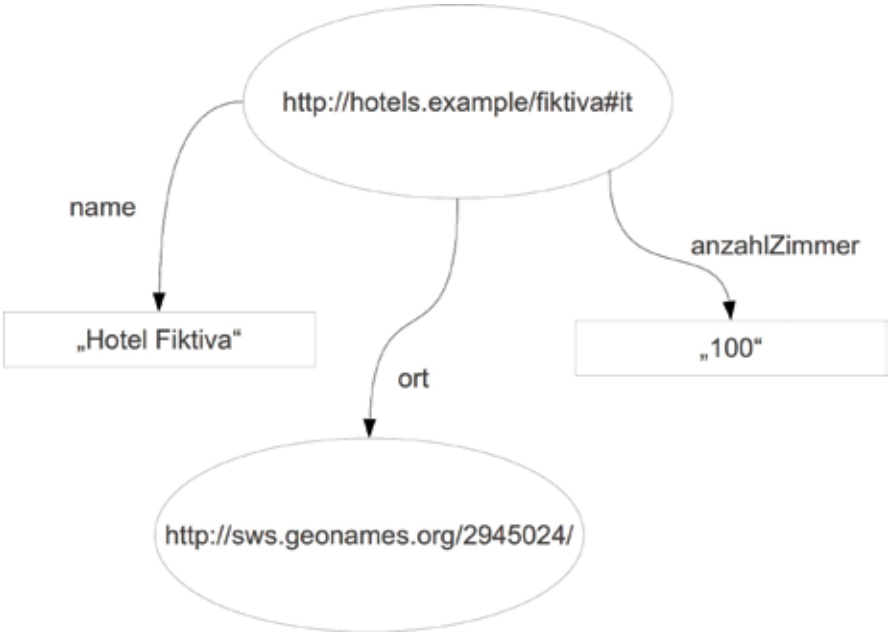


Abbildung 2: RDF-Graph

Daten verlinken

Die genannten Links sind der entscheidende Schritt zu einem Web aus Daten. Dadurch, dass wir Dinge über URIs identifizieren, ist es egal, wo wir die Informationen über diese Dinge hosten. Mithilfe dieser globalen ID sind sie auffindbar, wie jede Seite im WWW auch. Im obigen Beispiel verweist „hotel:ort“ zum Beispiel auf den URI „http://sws.geonames.org/2945024/“. Er ist die ID der Stadt Braunschweig beim Dienst „geonames.org“. Ruft man den URI ab, erhält man weitere Informationen über die Stadt, die in unserem ursprünglichen Datensatz gar nicht vorhanden sind, zum Beispiel die Einwohnerzahl. Dieser Datensatz verlinkt wiederum auf andere Dinge irgendwo im Web. Wir können den Links folgen, um weitere Daten und Zusammenhänge zu entdecken. Aber nicht nur die Subjekte und Objekte können über URIs identifiziert werden: Auch die verwendeten Prädikate selbst sind URIs. Sie sind Teil einer oder mehrerer Ontologien, die in der Turtle-Syntax vorab durch @prefix eingebunden werden (siehe Listing 5).

Hinter dem Präfix „hotel:“ verbirgt sich der URI „http://hotels.example/ontologie/“

```
@prefix hotel: <http://hotels.example/onto-
logie/>
<http://hotels.example/hotel/fiktiva#it>
  hotel:anzahlZimmer "100".
```

Listing 5

```
PREFIX hotel: <http://hotels.example/onto-
logie/>
SELECT ?name ?anzahl
WHERE {
  ?hotel hotel:name ?name;
        hotel:anzahlZimmer ?anzahl;
        hotel:ort <http://sws.geonames.
org/2945024/>.
  FILTER (?anzahlZimmer > 100)
}
```

Listing 6

und das Prädikat „hotel:anzahlZimmer" lautet vollständig „http://hotels.example/ontologie/anzahlZimmer". Ruft man den URI des Prädikats ab, erhält man wiederum RDF-Daten, die die Bedeutung des Prädikats beschreiben. Die Dokumentation der verwendeten Ontologien ist somit selbst Bestandteil des Webs aus Daten und maschinell auswertbar.

Das Web als globale Datenbank

Linked Data bildet ein Web aus maschinell auswertbaren Daten und macht das WWW so zu einer globalen Datenbank. Diese kann mit einer speziellen Abfragesprache namens SPARQL sogar abgefragt werden. Folgende Beispiel-Abfrage könnte alle Hotels in Braunschweig mit mehr als 100 Zimmern finden (siehe Listing 6).

Variablen wird ein Fragezeichen vorangestellt. Die SELECT-Clause beschreibt, welche dieser Variablen von Interesse sind. In der WHERE-Clause werden RDF-Tripel beschrieben, mit denen der Datenbestand abgeglichen wird. Das Ergebnis kann über Filter weiter eingeschränkt werden.

Während Suchmaschinen wie Google heute oft nur Fragen, aber keine Antworten finden, können semantische Suchen auf maschinell auswertbaren Daten operieren und exakte Ergebnisse finden. Eine solche Suchmaschine ist zum Beispiel „sindice.com". Die SPARQL-Schnittstelle [2] ist zwar noch in der Beta-Phase, lädt aber dennoch bereits zu Experimenten ein.

Linked Data in der Praxis

Linked Data kommt heute bereits an zahlreichen Stellen zum Einsatz. Eines der größten Projekte ist „Dbpedia" [3], das Informationen aus Wikipedia extrahiert und als Linked Data bereitstellt. Mit Wikidata [4] arbeitet die Wikimedia Foundation derzeit selbst daran, Daten strukturiert und sogar als Linked Data anzubieten. Die dezentrale Microblogging-Plattform „identi.ca" veröffentlicht Nutzerprofile als Linked Data und Facebook experimentiert mit Open Graph ebenfalls in diese Richtung [5].

Ein großer Vorreiter in Sachen Linked Open Data ist die New York Times, die seit dem Jahr 2009 Daten über rund 10.000 Subjekte veröffentlicht und unter einer Creative-Commons-Lizenz bereitstellt [6]. Die Bibliotheksverbunde Bayern und Berlin-Brandenburg haben gemeinsam ihre riesigen Bestandskataloge als Linked Data verfügbar gemacht und damit Daten von über 23 Millionen Medien online gestellt [7]. Das Web of Data wächst, und je größer es wird, desto mehr nutzt es uns allen. Wie wir selbst Linked Data veröffentlichen und verarbeiten können, werden wir in der nächsten Ausgabe betrachten.

Chancen und Ausblick

Mit Linked Data werden Daten ein natürlicher Bestandteil des Webs. Datenbanken werden Teil des Webs und das Web wird eine riesige, offene Datenbank. Statt unterschiedliche Datenformate über proprietäre APIs abzufragen und miteinander zu vereinbaren, können wir URIs abrufen, um strukturierte Informationen zu den Dingen zu erhalten, die sie identifizieren. Wir können den dort enthaltenen Links folgen, um weitere Informationen zu erhalten, ohne dass wir ein zusätzliches Web-Service-API erlernen und implementieren müssen.

Linked Data bietet darüber hinaus große Chancen auf vielen Gebieten: Indem wir unsere Daten global miteinander verlinken, können wir Zusammenhänge entdecken, die zuvor verborgen blieben. Forschung und Wissenschaft sind nicht mehr nur in der Lage, Forschungsdokumente zu veröffentlichen, sondern auch ihre Rohdaten untereinander zu verknüpfen und so ungeahnte Beziehungen zu entdecken. Die Verbraucher profitieren von einer höheren Markt-Transparenz, wenn als Linked Data veröffentlichte Produkt-Merkmale leicht miteinander verglichen werden können. Unternehmen wird die Marktforschung erleichtert, wenn das Feedback der Verbraucher mit den Produkten und Dienstleistungen, auf die es sich bezieht, verlinkt ist.

Auf Grundlage von Linked Data kann ein „Social Semantic Web" entstehen. Ein Klick auf einen Like-Button erzeugt de facto ein RDF-Tripel mit mir als Subjekt, dem Prädikat „like" und der Sache, die ich mag, als Objekt. Letztlich könnten alle Handlungen, die wir im Web tätigen, als Linked Data abgebildet werden, sei es die Veröffentlichung eines Blog-Artikels, das Hinterlassen eines Kommentars, das Taggen eines Bilds oder das Bewerten eines Produkts. Allein durch die Nutzung des Webs können wir ein Web aus Daten erzeugen.

Dazu ist es notwendig, dass wir Linked Data in unseren Web-Projekten unterstützen. Wie das geht, werden wir in der nächsten Ausgabe zeigen. Wer sich bis dahin näher mit Linked Data befassen möchte, dem steht der Autor mit seinem Blog unter „http://datenwissen.de" zur Verfügung.

Referenzen

[1] http://www.w3.org/DesignIssues/LinkedData.html
[2] http://sparql.sindice.com/
[3] http://dbpedia.org/
[4] http://meta.wikimedia.org/wiki/Wikidata/de
[5] http://developers.facebook.com/docs/opengraph/
[6] http://data.nytimes.com/
[7] http://lod.b3kat.de/doc

Angelo Veltens
angelo.veltens@online.de
http://datenwissen.de

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

Der Herbstcampus 2012 aus Sicht eines Besuchers

Steven Schwenke, msg systems ag

Der jährlich in Nürnberg stattfindende Herbstcampus bietet Vorträge zu Java-, Scala- und Software-Entwicklungsthemen und eignet sich aufgrund seiner Größe sehr gut zum Networking und direkten Kontakt mit Vortragenden. Der Autor beschreibt seine Eindrücke von den Vorträgen und seine Wahrnehmung der Veranstaltung.

Der jährlich stattfindende Herbstcampus mit rund 140 Teilnehmern wird von der MATHEMA Verwaltungs- und Service-Gesellschaft mbH in der Georg-Simon-Ohm-Hochschule für angewandte Wissenschaften in Nürnberg organisiert. Auf meinen ersten Besuch hatte ich mich bereits lange vor der Konferenz gefreut. Meine Erwartungen beschränkte ich darauf, interessanten Menschen zu begegnen und einiges zu lernen.

Schon vor dem ersten Vortrag hatte sich die Sicht auf meinen Alltag verändert. Besonders in den Gesprächen mit den anderen Teilnehmern sah ich mich und meine Arbeit schnell von weiter außen und fing an, diese zu reflektieren. Es war wie ein Rückzug aus dem Arbeitstrott. Noch kurz vor der Konferenz hatte ich den Gedanken, die Zeit lieber mit dem Vertiefen der in meinem Projekt erforderten Technologie zu verbringen. Das wäre ein großer Fehler gewesen.

Bereits die erste Keynote „Dauerhaft schlank – Kein Lean ohne Automatisierung“ von Hans Dockter weckte den mir bisher unbekannten Gedanken, dass die heutige Software-Entwicklung, wenn nicht ganz am Anfang, dann doch stark in der Veränderung steht. Themen wie „Automatisierung“ oder „Software-Testing“ sind noch lange nicht allgemein anerkannt und vereinheitlicht. Für Berufseinsteiger wie mich ist das hochinteressant, da sich viele Entwicklungsmöglichkeiten bieten und man sein angelesenes Wissen stets infrage stellen sollte.

Ein wesentlich besseres Verständnis meiner eigenen technischen Domäne vermittelte mir Dalibor Topics Vortrag „Java SE 8 & Beyond“. Mit Dingen wie Releases einer neuen Java-Version beschäftigt man sich als Entwickler nicht alltäglich, bisher nahm ich die Entwicklung von Funktionen und Sprach-Features ungefragt als gegeben hin. Nachzuvollziehen, wie sich das tagtäglich genutzte Werkzeug entwickelte, lässt eine höhere Identifikation mit dem eigenen Schaffen zu und gibt mir ein besseres Gefühl für das, was ich mache.

Nicole Rauch und Andreas Leidig bestätigten mit ihrem Vortrag „Beispielhaft – Specification by Example“ einen mir schon lange bekannten Gedanken: Entwickler müssen mit Anwendern reden! Ich habe es in meinem kurzen Berufsleben bereits häufig erlebt, dass Entwickler wenig oder gar nicht mit Anwendern reden beziehungsweise versuchen, deren Probleme zu verstehen. Dabei müssen sie genau das tun, um deren Probleme zu lösen. Damit hat ein Entwickler zwei Rollen: Einerseits ist er für die technische Umsetzung, andererseits für die fachliche Beratung zuständig. Diesem Gedanken war ich bereits vor der Konferenz eher zufällig gefolgt. So hatte ich mir, obwohl ich zugegebenermaßen wenig an Autos interessiert bin, eine Automobil-Zeitschrift abonniert. Ich möchte in dem Bereich mitreden können, in dem ich als Entwickler arbeite. Solche Einsichten machen den Besuch der Konferenz an sich schon lohnenswert.

Bis zur Mitte des ersten Tages hatte ich durch das Gehörte reichlich Gelegenheit, eigene Fehler als solche zu erkennen. Andererseits merkte ich in Vorträgen wie „Fählerprevention» von Joachim Hofer, dass ich auch schon vieles richtig machte. Der Einsatz von David Allens GTD, das Vermeiden von Multitasking, Arbeitslisten und die Nutzung des vielbeschrienen „Flows“ während der Arbeit sind einige Dinge, die Fehler in der Software-Entwicklung vermeiden und damit die Kosten erheblich senken.

Am Ende des ersten Tages nahm ich nach inspirierenden Vorträgen an vielen Diskussionen teil, die enormen technischen Tiefgang hatten. Für mich waren insbesondere die Nutzung von immutablen Objekten sowie die Vermeidung von Void-Methoden interessant. Diese beiden Themen seien als umstrittene Standpunkte stellvertretend für viele interessante Denkanstöße genannt, die mich noch lange beschäftigen werden.

Die Konferenz bot durch die Tutorials am ersten Tag Möglichkeiten sowohl zur Vertiefung einzelner Themen als auch zu einem breiten fachlichen Überblick. So habe ich durch den Vortrag von Eberhard Wolff viel über den aktuellen Stand der Cloud-Technologien, die Entwicklung JavaScript und die Tendenzen in der Branche gelernt.

Ein besonderes Lernerlebnis war der Vortrag „Build-Dreikampf – Ein Vergleich zwischen Ant, Maven und Gradle“ von Carl Anders Düvel und Sven Bung. Ich habe nur wenig Erfahrung im Schreiben von Ant-Skripten, Maven kenne ich nur vom Prinzip her und Gradle ist mir zugegebenermaßen recht unbekannt. Ich ging also ziemlich ahnungslos in den Vortrag hinein und kam mit einem guten Überblick über die Technologien heraus. Natürlich kann ich dadurch noch immer nicht die Syntax einer „pom.xml“ im Detail, aber ich habe durch den sehr schönen Vergleich der drei Build-Tools im Vortrag einen Überblick bekommen.

Denken in eine andere Richtung

Was mich sehr überraschte, war die Eröffnung ganz neuer Problemklassen. Dr. Elmar Jürgens hielt einen extrem interessanten Vortrag über den Umgang mit Redundanz in Software-Artefakten und

brachte mir damit die mir neue Metrik der „Clone Coverage“ nahe. Ich weiß nicht, warum ich das nicht schon längst als wichtiges Thema wahrgenommen hatte, manchmal braucht es wohl einen Botschafter. Im Anschluss an die Konferenz habe ich mir die vorgestellte Software angesehen und konnte sie direkt auf mein aktuelles Projekt anwenden. Auch der wundervoll vorgetragene Talk von Stefan Zörner über Architektur-Dokumentation traf den Nerv einiger aktueller Überlegungen und Probleme in meinem Umfeld. Sehr praktisch, wenn man direkt im Vortrag bereits Lösungsansätze für die eigenen Probleme bekommt.

Herbstcampus

Selbst als Referent aktiv

Auch wenn die gehörten Vorträge interessant waren, war ich nicht nur als Besucher nach Nürnberg gekommen. Zusammen mit Jens Schauder von der Java User Group Ostfalen hielt ich einen Vortrag über unser seit über einem Jahr laufendes Mentoring. Auch wenn es nicht der erste Vortrag war, den ich seit Abschluss meines Studiums gehalten habe, war es doch ein ganz besonderer. Die Zuhörer waren sehr interessiert, stellten viele Fragen und waren willens, das Gesagte umzusetzen. Auch nach dem Talk vertieften wir das Thema gemeinsam mit einigen Zuhörern. Der Vortrag und diese Gespräche waren ein Highlight meines Besuches in Nürnberg und ich empfehle allen Entwicklern, das einmal selbst auszuprobieren.

Der eng gepackte Zeitplan des Herbstcampus ließ auch den zweiten Tag wie im Flug vergehen, doch nicht nur in den Vorträgen konnte man seinen Horizont erweitern. Gerade in der Zeit dazwischen bestand die Möglichkeit, im persönlichen Austausch Themen zu vertiefen oder Kontakte zu knüpfen. Dies war insbesondere auf dem Herbstcampus sehr gut möglich: Es gab nicht so viele Teilnehmer, sodass man sich mehrfach über den Zeitraum der Veranstaltung sah und nach einem Talk den Speaker sprechen konnte. Die Konferenz war jedoch besucht genug, um ausreichend Auswahl zwischen interessanten Personen zu haben. Gespräche wurden zusätzlich durch die räumliche Dichte gefördert. So waren alle Vortragssäle über einen gemeinsamen breiten Flur verbunden, in dem ständig Getränke und Snacks angeboten wurden. Ich habe auch mit Leuten über Themen gesprochen, die man auf einer solchen Konferenz eher nicht vermuten würde: „Bio-Food“, „Weinanbau“ und „Regenerative Energien“ sind auch bei Software-Entwicklern ernstzunehmende Themen.

Fazit

Rückblickend kann ich sagen, dass der Herbstcampus ein extrem interessantes, fesselndes und dichtes Event war, das ich auf jeden Fall erneut besuchen möchte. Ein Entschluss steht bereits jetzt fest: Im nächstem Jahr reiche ich mindestens einen technologisch-orientierten und einen Softskill-Vortrag ein.

Steven Schwenke
steven@stevenschwenke.de

Steven Schwenke arbeitet seit drei Jahren im Automotive-Umfeld mit Java. Er hält regelmäßig Vorträge und schreibt Artikel über seine Erfahrungen in der IT-Branche.

Rückblick auf die Source Talk Tage 2012

Daniel van Ross, Java User Group Deutschland e.V.

Am 28. und 29. August 2012 fanden zum achten Mal die Source Talk Tage im Mathematischen Institut der Universität Göttingen statt. Die bundesweit einmalige Veranstaltung führte Studierende, Berufseinsteiger und Unternehmer im direktem Austausch zusammen.

Die Java User Group Deutschland e.V. setzte als Mitveranstalter einen der Schwerpunkte auf Vorträge rund um das Thema „Java“. Darüber hinaus boten renommierte Referenten aus dem IT-Bereich in anspruchsvollen Trainings sowohl Studierenden als auch Profis neueste Informationen und die Möglichkeit zum Dialog und zum gegenseitigen Kennenlernen.

Mehr als 200 TeilnehmerInnen fanden Ihren Weg nach Göttingen. Themen wie „Clean Code“, „HTML5“ und „NetBeans“ erfreuten sich großen Zuspruchs. Für 90 Studierende aus ganz Deutschland war der Besuch der Trainings kostenfrei. Sie nahmen damit Trainingsleistungen im Wert von rund 50.000 Euro in Anspruch. Die Anbieter, zum Teil regionale IT-Unternehmen, schätzten den engen Kontakt, der in der kleinen Gruppe im Laufe eines Tages aufgebaut werden konnte. Die Veranstaltung ist damit inzwischen zu einer etablierten Schnittstelle zwischen Universität und Arbeitswelt geworden.

20 Jahre Java, 20 Folien, 20 Sekunden

Oliver Böhm, Java User Group Stuttgart

Am 21. September 2012 startete die Java User Group Stuttgart (JUGS) ein Experiment: Kann es gelingen, das Pecha-Kucha-Format (PeKu), das hauptsächlich im architektonischen und künstlerischen Bereich eingesetzt wird, auch auf technische Vorträge zu übertragen?

Das Pecha-Kucha-Format wurde im Jahre 2003 in Tokio von den Architekten Astrid Klein und Mark Dytham als Antwort auf das „Death-by-Powerpoint"-Syndrom vorgestellt: 20 Folien á 20 Sekunden ergibt 6:40 Minuten Vortragszeit. Für die Referenten in Stuttgart wurde mit Pecha-Kucha als Vortragsformat Neuland betreten. Umso erfreulicher war es, dass sich einige dieser Herausforderung gestellt und sich folgenden Spielregeln unterworfen haben:

- Der Vortrag besteht aus 20 Folien á 20 Sekunden
- Jede Folie wird exakt 20 Sekunden lang gezeigt. Die Folien wechseln automatisch – der Vortragende hat keinen Einfluss darauf
- Die Reihenfolge der Folien legt der Vortragende fest
- Die Reihenfolge der Vortragenden legt der Veranstalter fest

Folgenden Tipp bekamen die Referenten dabei mit auf den Weg: „Beschränke dich auf das Wichtige – 20 Sekunden sind kurz!"; das war aus Sicht des Autors auch das Schwierigste: Unnötiges wegzulassen und sich auf das Wesentliche zu beschränken. Aber dies machte gerade den Reiz aus, wie viele Teilnehmer hinterher bestätigten. Eine weitere Herausforderung für den Veranstalter war die Namensgebung. Der Begriff „Pecha Kucha Night" für Veranstaltungen ist geschützt, sodass die Entscheidung letztendlich auf „20 Jahre Java" fiel.

20 Jahre Java

Bevor jemand den Taschenrechner aus der Schublade kramt und nur auf 17 Jahre kommt, denn Java wurde im Jahr 1995 auf der SunWorld angekündigt, hier die Auflösung: Die Ursprünge von Java gehen auf das Green-Projekt zurück, das bereits 1992 einen PDA-Device mit Touch-Screen und Duke als Assistenten hervorgebracht hat. Die Software war in Oak geschrieben, das später aus lizenzrechtlichen Gründen in „Java" umbenannt wurde.

Das Jahr 1992 war aus Sicht der damals noch nicht vorhandenen Java User Group Stuttgart (sie entstand erst fünf Jahre später) ebenfalls erfreulich: Der VfB Stuttgart war deutscher Fußball-Meister (vor Borussia Dortmund).

20 Folien, 20 Sekunden

Für die Zuhörer war der Abend kurzweilige Unterhaltung – für die Vortragende Stress, zumal manche Folien nicht in der aktuellsten Version aufgelegt wurden. In der Pause und im gemütlichen Teil konnte man dann in entspannter Atmosphäre mit den Referenten in Kontakt treten und all die Fragen loswerden, die während des Vortrags (aus Zeitgründen) unterbunden worden waren.

Die Themen waren bunt gemischt und für jeden war etwas dabei. Es gab diverse Rückblicke und Ausblicke rund um die Entwicklung des Java-Universums, diverse Einblicke in das, was man lieber nicht in/mit Java machen sollte, aber auch Themen wie „Spring Roo", „EHCache", „Workflow-Engines" oder „Eclipse Code Recommenders". Zitat: „Entwickler, die diese Methode verwendet haben, haben auch diesen Code eingebaut …"

Dem Autor hat vor allem die Vorstellung von „Specification by Example" (Vortrag: „From System.out to Excecutabe Specifiations") gefallen, weil er dadurch einige Anregungen für künftige Entwicklungsvorhaben mitnehmen konnte. In der anschließenden Diskussion mit dem Referenten konnten die Teilnehmer Erfahrungen austauschen und Tipps darüber einholen, wie man mit fehlenden Tests umgehen und Mehrwert aus Dokumenten herausziehen kann. Damit konnte der Autor wieder Energie tanken, um all die Dinge anzugehen, die in der Vergangenheit aus Zeitgründen vernachlässigt wurden.

Generell war festzustellen, dass viele Teilnehmer die Gelegenheit genutzt haben, sich mit den Referenten und anderen Teilnehmern auszutauschen. Und es gab viel zu diskutieren – schließlich war die Informationsdichte pro Vortrag recht hoch und machte Appetit auf mehr Informationen zu dem einen oder anderen Thema.

Weiterführende Links

- http://www.jugs.org
- http://jugs.org/2012-09-21.html
- http://www.pecha-kucha.org/
- http://www.java-forum-stuttgart.de/
- http://oli.blogger.de/20120923/

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.

Scrum & Kanban in der Praxis

Martin Dilger, PENTASYS AG

Dieser Artikel erzählt einige Episoden aus dem Alltag eines überwiegend fiktiven Scrum-Teams und dessen üblichen Anforderungen, Problemen und Erfolgen. Der Autor ist derzeit Entwickler und Scrum-Master in einem agilen Team.

Das Scrum-Team, das wir die nächsten paar Tage begleiten, besteht aus Karl, Markus und Georg (Java-Entwickler), Hans (Tester) und Arne (Scrum-Master). Das Team entwickelt einen Online-Shop und arbeitet mit Scrum. Hierbei handelt es sich um die wahrscheinlich bekannteste agile Vorgehensweise, die es erlaubt, Produkte inkrementell in kleinen Schritten zu entwickeln.

Das Team arbeitet in Sprints, die üblicherweise eine Länge von zwei Wochen haben. Es vereinbart mit dem Kunden (den wir gleich kennenlernen werden), was am Ende jedes Sprints geliefert werden soll. Geliefert werden die mit dem Kunden vereinbarten Teil-Funktionalitäten, diese aber tatsächlich fertig. Das bedeutet für unser Team: Unit-Tests vorhanden, Peer Review durchgeführt, dokumentiert, Acceptance-Test auf definierter Testumgebung durchgeführt und jeder im Team ist so zufrieden mit der durchgeführten Arbeit, dass man sie sich am liebsten zu Hause an den Kühlschrank pinnen möchte.

10:30 Uhr: Das wichtigste Meeting des Tages

Der erste und wichtigste Termin des Tages eines jeden Scrum-Teams ist das Daily Scrum. Es dauert höchstens 15 Minuten und startet pünktlich um 10:30 Uhr. Zu diesem Zeitpunkt sind auch die nachtaktiven Teammitglieder ausgeschlafen und können erste klare Gedanken fassen. Das Daily Scrum ist deshalb so wichtig für das Team, weil es diese 15 Minuten nutzt, um sich für den Tag zu synchronisieren.

Heute startet das Daily Scrum mit Karl. Er erzählt, dass er seit gestern an einem Feature zur Validierung von Kundendaten im Frontend des Online-Shops gearbeitet hat. Das beantwortet Frage 1: „Was habe ich seit gestern geschafft?“ Anschließend erzählt er, dass er dieses Feature gerne zusammen mit Hans, dem Tester, auf eine Testumgebung deployen und ausführlich testen möchte. Das beantwortet Frage 2: „Was möchte ich heute tun?“

Zuletzt erzählt Karl, dass er dringend auf eine Zulieferung aus der Grafikabteilung wartet, um die Validierungsfehler im Frontend korrekt darstellen zu können. Bis die Grafikabteilung die fehlenden Stylesheets liefert, kann Karl das Feature nicht abschließen. Damit beantwortet Karl Frage 3: „Was behindert mich aktuell?“ Als Karl diese Frage beantwortet, wird sofort Arne, der Scrum-Master, hellhörig, denn genau das ist seine Aufgabe. Der Scrum-Master löst Probleme. Er sorgt dafür, dass das Team zu jedem Zeitpunkt produktiv arbeiten kann.

Was wird Arne also machen? Er wird direkt nach dem Daily Scrum nach Möglichkeit nicht anrufen und auch keine E-Mail schreiben, sondern der Grafikabteilung ein paar Büros weiter einen persönlichen Besuch abstatten und versuchen, die dringend benötigte Lieferung zu beschleunigen.

Diese 15 Minuten sind unglaublich wichtig für das Team, weil damit sichergestellt ist, dass jeder zumindest grob die Aufgaben der anderen Teammitglieder kennt und der Fortschritt des Projekts täglich besprochen wird.

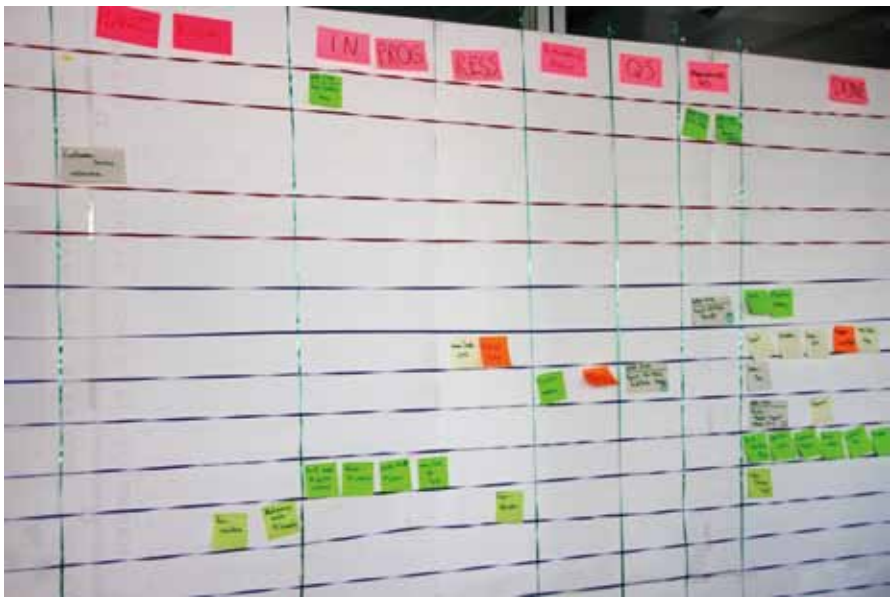


Abbildung 1: Das Board

Das wichtigste Werkzeug

Wir haben das wichtigste Meeting eines Scrum-Teams kennengelernt, jetzt werden wir das wichtigste Werkzeug kennenlernen – das Board. Es visualisiert die Arbeit des Teams (siehe Abbildung 1). Am Board ist jederzeit für jeden (auch beispielsweise für den Kunden) ersichtlich, welche Features aktuell umgesetzt werden, welche

bereits abgeschlossen sind und was noch für den restlichen Sprint geplant ist.

Das Board des Teams kann gar nicht groß genug sein. Es ist mehr als vier Meter lang und bedeckt eine komplette Bürowand. Es ist unterteilt in mehrere Spalten, die die einzelnen Schritte repräsentieren, die eine Anforderung bis zur Fertigstellung durchläuft (siehe Abbildung 2).

Gibt der Kunde eine neue Anforderung in das Team, landet diese zunächst als „Post-it"-Zettel in der „TODO"-Spalte (siehe Abbildung 3). Üblicherweise verbraucht ein Scrum-Team Unmengen dieser „Post-its".

Die „TODO"-Spalte wird vom Kunden priorisiert. Je weiter oben dort eine Anforderung steht, desto wichtiger ist dem Kunden die Umsetzung und desto früher wird die Anforderung vom Team bearbeitet. Sobald sich das Team entscheidet, eine neue Anforderung in Bearbeitung zu nehmen, wandert diese aus der „TODO"- in die „DEV"-Spalte. Damit ist sofort ersichtlich, dass aktuell an dieser Anforderung gearbeitet wird. Ist die Entwicklung abgeschlossen, wandert die Anforderung weiter in die „QS"-Spalte. Diese Spalte ist ein Entwicklungsschritt, der vom Team definiert wurde, um die Qualität zu verbessern. Eine Anforderung, die hier steht, ist bereits fertig entwickelt und wird nochmals von einem Entwickler überprüft, der nicht an der Umsetzung beteiligt war.

Irgendwann ist auch dieser Entwickler zufrieden und die Anforderung wandert erneut eine Spalte weiter in die „TEST"-Spalte. Hier wird Hans aktiv und testet die Anforderungen gegen die Akzeptanzkriterien des Kunden. Ist Hans zufrieden, gibt auch er grünes Licht und die Anforderung wandert wieder einen Schritt weiter in die „DEPLOYMENT"-Spalte.

Eine Anforderung in dieser Spalte ist fertig implementiert, vom Kunden abgenommen und auf dem Weg in das Produktiv-System. Dieser Schritt wird leider nicht direkt vom Team selbst durchgeführt, sondern von einer externen Abteilung, die aber im gleichen Gebäude ist. Das bedeutet aber keinesfalls, dass das Team aus der Pflicht genommen wird, ein schnellstmögliches Deployment sicherzustellen.

Das Team unterstützt das Deployment und fordert ständig Informationen zum aktuellen Status ein. Erst dann, wenn die Anforderung komplett umgesetzt, Review und Test erfolgreich waren und die Anforderung im Livesystem von echten Kunden verwendet wird, wandert die Anforderung in die „DONE"-Spalte und ist somit abgeschlossen (siehe Abbildung 4).

Der Kunde

Wir haben bereits öfter vom Kunden gesprochen. Wer ist nun dieser Kunde? Das Team arbeitet für den Product-Owner. Der Product-Owner des Teams ist Henning. Henning war früher Projektmanager und ist heute Product-Owner mit Leib und Seele.

Er entwickelt die Vision des Produkts und weiß genau, was vom Team wann umgesetzt werden muss, um den Online-Shop zum Erfolg zu führen. Er definiert die Anforderungen, die er dem Team übergibt.

Wie sieht jetzt eine typische Anforderung in Scrum aus? Eine Anforderung in Scrum wird „User-Story" genannt. Sie hat idealerweise die folgende Form: „Als X möchte ich Y um Z." Das beantwortet die drei wichtigsten Fragen, die das Team braucht, um eine Anforderung umzusetzen. „Wer möchte etwas haben?", „Was möchte derjenige haben?" und „Warum wird diese Funktionalität gebraucht?"

Abbildung 5 zeigt ein Beispiel für eine ausformulierte User-Story. Dieser eine Satz genügt einem gut eingespielten Scrum-Team, um eine Anforderung umzusetzen. Alle weiteren Fragen werden in separaten Planungs-Meetings geklärt.

Ein neuer Tag, ein neuer Sprint

Überspringen wir einige Tage. Das Team hat den laufenden Sprint abgeschlossen und sowohl Henning als auch das Team sind sehr zufrieden mit dem aktuellen Status des Projekts. Für den neuen Sprint ist das Sprint-Backlog (die „TODO"-Spalte) gut gefüllt (ähnlich Abbildung 3). Das Team ist hochmotiviert und beginnt direkt mit der Arbeit und den ersten Stories, die auch direkt in die „DEV"-Spalte wandern.

Karl übernimmt Anforderung „A" („Als Kunde möchte ich meine Kundendaten ändern, um nach einem Umzug trotzdem Bestellungen an die richtige Adresse geliefert zu bekommen"), Markus die Anforderung „B" („Als Kunde möchte ich alle Links im Online-Shop mit neuen Styles gerendert bekommen, damit ich diese besser finden kann.").

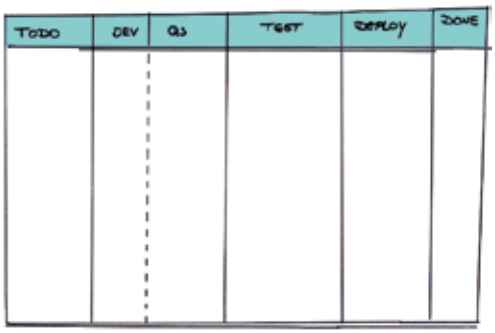


Abbildung 2: Das Board des Teams

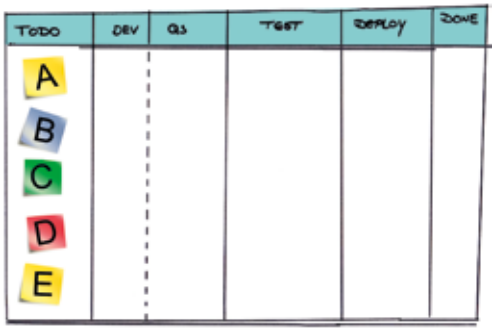


Abbildung 3: Anforderungen auf dem Board

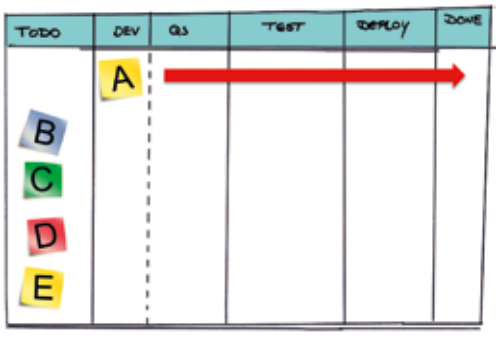


Abbildung 4: Anforderungen wandern auf dem Board von links nach rechts

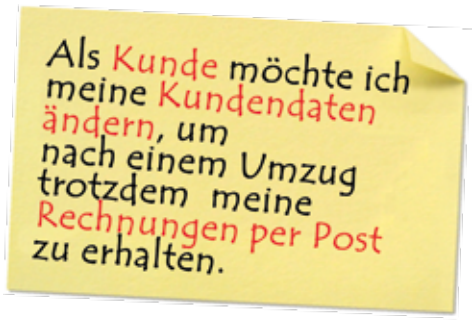


Abbildung 5: Eine typische User-Story

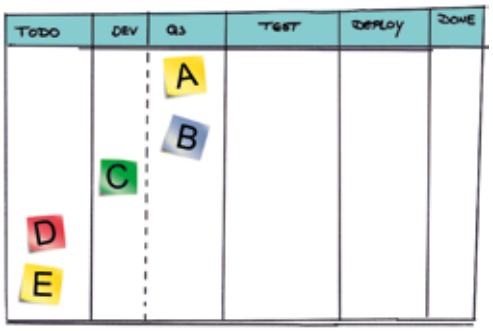


Abbildung 6: Das Scrum-Board in Aktion

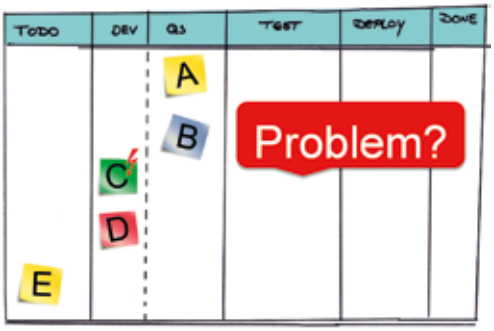


Abbildung 7: Ein Problem zeichnet sich ab

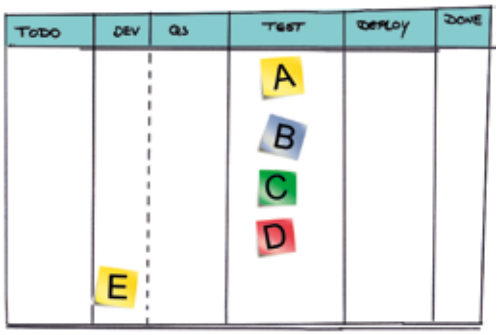


Abbildung 8: Stau auf dem Board

Abbildung 9: Das Team misst die Lead-Time

Anforderung „B" ist von Markus sehr schnell umgesetzt und wandert weiter in die „QS"-Spalte, wo Georg sie nochmals überprüft. Markus nimmt inzwischen bereits die Anforderung „C" („Als Administrator möchte ich tägliche Reportings über eingegangene Bestellungen erhalten, um schnell auf Lagerengpässe reagieren zu können") in Bearbeitung, denn er will keine Zeit verschwenden. Am Ende des ersten Tages ist auch Karl mit der Anforderung „A" fertig, sodass sich auf dem Board ein neues Bild ergibt (siehe Abbildung 6).

Leider muss Markus sehr schnell feststellen, dass die Anforderung „C" nicht umgesetzt werden kann, da die Schnittstelle zur Anbindung an das Lagerhaltungs-System noch nicht bereitsteht. Die Story ist also blockiert. Markus weist Arne, den Scrum-Master, auf dieses Problem hin, markiert die blockierte Story auf dem Board mit einem roten Blitz und nimmt sofort die Anforderung „D" in Bearbeitung (siehe Abbildung 7). Arne betrachtet das Board und kann nur seinen Kopf schütteln. Sein Bauchgefühl sagt ihm, dass zu viele Stories parallel in Bearbeitung sind.

Ein Scrum-Board dient zur Visualisierung der Arbeit des Teams. Die Stories auf dem Scrum-Board verhalten sich wie Autos auf einer Autobahn. Eine Autobahn hat keinen anderen Zweck, als auf ihr möglichst schnell von Punkt „A" nach Punkt „B" zu gelangen. Analog dazu möchten die Stories auf dem Board möglichst schnell von der „TODO"- in die „DONE"-Spalte. Erst wenn ein Feature live ist und von echten Kunden verwendet wird, ist dadurch auch ein echter Wert für den Kunden, also für Henning, entstanden.

Was passiert auf einer Autobahn, wenn zu viele Autos gleichzeitig unterwegs sind? Es kommt zum Stau. Genau dasselbe geschieht auf dem Board. Sind zu viele Stories gleichzeitig in Bearbeitung, werden die Stories nicht schneller, sondern viel langsamer abgeschlossen. Team-Mitglieder blockieren sich gegenseitig und die Re-Integration von Features wird komplizierter, beispielsweise durch Konflikte beim Mergen verschiedener Änderungen.

Ein weiteres Problem, das sofort jedem im Team auffällt ist: Hans, der Tester, legt bisher die Beine hoch und dreht Däumchen. Denn bisher hat es keine Story bis in die „TEST"-Spalte geschafft. Das Schlimmste, was Hans jetzt passieren kann ist, dass alle Anforderungen gleichzeitig fertiggestellt werden (siehe Abbildung 8). Hans wird mit so viel Arbeit überhäuft, dass nun die „TEST"-Spalte das Board blockiert und er keine weiteren Stories aufnehmen kann. Nach wie vor ist das Team dann weit davon entfernt, eine Story bis ins Live-System gebracht zu haben.

Der Scrum-Master wird aktiv

Arne hat kürzlich ein Buch über Kanban gelesen. Kanban wird von vielen oft als Alternative zu Scrum verstanden. Arne versteht Kanban aber eher als ein Set von Spezial-Werkzeugen, um die Produktivität des Teams zu steigern. Eine erste Maßnahme, die Arne dem Team vorstellt, ist das Messen der „Lead-Time". Das ist eine hilfreiche Metrik, die sehr einfach zu erfassen ist und üblicherweise sehr schnell zu ersten „Aha"-Momenten führt. Das Team misst die Lead-Time für eine erste Story (siehe Abbildung 9).

Die Lead-Time bezeichnet die Zeit, die eine Story braucht, um von der „TODO"- in die „DONE"-Spalte zu gelangen. Sie wird gemessen, indem auf der jeweiligen Story am Board jedes Mal das Datum vermerkt wird, wenn die Story eine Spalte weiter wandert. In unserem Beispiel wurde die Anforderung am 01.04. vom Product-Owner ins Team gegeben, am 02.04. in Bearbeitung genommen und am 08.04. an Hans, den Tester, übergeben. Am Ende des Sprints kann über die Lead-Time aller Stories ein Mittelwert gebildet werden, der nichts anderes als eine Metrik für die Geschwindigkeit des Teams darstellt.

Insbesondere Engpässe lassen sich so sehr schnell erkennen. Dauert der Sprung von der „DEV"- in die „TEST"-Spalte jedes Mal sehr lange, wird dies am Board sofort sichtbar und man kann hinterfragen, wieso das so ist und was dagegen getan werden muss. Das Team ist begeistert und nimmt sich vor, die Messung der Lead-Time zukünftig für alle Stories durchzuführen.

WIP-Limits für das Team

Allein durch die Einführung der Lead-Time wird dem Team schon viel klarer, wo die Engpässe liegen. Die Messung der Lead-

Time macht Probleme sichtbar, löst diese aber nicht.

Arne macht einen weiteren Vorschlag. Kanban bietet viele Werkzeuge, die zum Ziel haben, Arbeit fokussiert und konzentriert zum Abschluss zu bringen. Das vielleicht wichtigste Werkzeug hierfür sind „Work in Progress"-(WIP)-Limits. Mit deren Definition limitiert das Team die Anzahl an Stories oder auch Tasks, die parallel in Bearbeitung sein dürfen. WIP-Limits sind kein intuitives Werkzeug. Es braucht zu Anfang ein wenig Zeit, dieses Werkzeug effektiv einzusetzen.

Das Team und Arne haben das Problem auf dem Board zuvor schon richtig erkannt. Es wird zu viel Arbeit parallel gemacht. Arne schlägt vor, einen Sprint lang mit WIP-Limits zu arbeiten um zu sehen, wie sich dieses Werkzeug auf die Produktivität auswirkt. Dies geschieht ganz einfach dadurch, dass auf dem Board für jede Spalte oder auch zusammengefasst für mehrere Spalten definiert wird, wie viele Stories gleichzeitig in Bearbeitung sein dürfen (siehe Abbildung 10).

Das Team setzt sich zusammen und definiert, dass in der „DEV"- und „QS"-Spalte nicht mehr als zwei Stories gleichzeitig in Bearbeitung sein sollten. Arbeitet Karl an Story „A" und Markus an Story „B", ruht Georg sich in der Zwischenzeit aus? Keinesfalls. Da durch das definierte WIP-Limit nicht mehr als zwei Anforderungen gleichzeitig bearbeitet werden dürfen, wird das Team gezwungen, zusammenzuarbeiten. Georg unterstützt seine Teammitglieder um möglichst schnell eine Anforderung mindestens in die „TEST"-Spalte zu bringen. Dies schafft einen freien Slot und das Team kann Story „C" in Bearbeitung nehmen. Das Team definiert, dass Hans der Tester nur eine Story gleichzeitig bearbeiten sollte, um diese möglichst schnell an das Deployment-Team weiterreichen zu können (siehe Abbildung 11).

Hans ist mit dem Testen von Story „B" beschäftigt. Die Stories „A" und „C" befinden sich bereits in der „QS"-Spalte. Das Board blockiert, denn das definierte WIP-Limit für die „TEST"-Spalte gestattet es nicht, weitere Stories aufzunehmen. Können sich die Entwickler nun zurücklehnen? Keinesfalls, denn um einen freien Slot zu schaffen und damit Story „D" möglichst schnell in Bearbeitung nehmen zu können, unterstützen die Entwickler Hans bei technischen Problemen (Test-Umgebung funktioniert nicht), versorgen ihn mit Know-how über die Implementierung (erleichtert das Testen) oder bringen ihm einfach frischen Kaffee, um ihn wach zu halten.

Schon nach wenigen Tagen im Sprint zeichnet sich ab, dass die Lead-Time der Stories viel kleiner ist, als noch vor der Einführung von WIP-Limits. WIP-Limits zwingen also das Team, fokussiert an wenigen Anforderungen zu arbeiten und diese zum Abschluss zu bringen.

Scrum oder Kanban oder Scrumban?

Was ist nun besser und sollte verwendet werden? Scrum oder Kanban? Für das Team ist dies keine „Entweder/oder"-Entscheidung, sondern idealerweise verwendet man das Beste aus beiden Welten. Das Team arbeitet weiterhin mit festen Sprintlängen, was Henning die Planung erleichtert, und macht weiterhin Daily-Scrums. Zusätzlich misst das Team die Lead-Time und arbeitet mit WIP-Limits.

Natürlich bieten sowohl Scrum als auch Kanban noch viel mehr Ansätze, Werkzeuge und Ideen, als in diesem Artikel vorgestellt werden konnten. Der Autor arbeitet derzeit als Entwickler und Scrum-Master in einem agilen Team. Dort werden die Vorteile aus beiden Welten und alle Werkzeugen, die sinnvoll erscheinen, genutzt.

Fazit

Es empfiehlt sich, nicht dogmatisch, sondern pragmatisch vorzugehen. Agil arbeitet man nicht schwarz oder weiß. Man kann aus einer Vielzahl an Werkzeugen wählen und diejenigen nutzen, die sinnvoll erscheinen. Wichtig ist das Experimentieren. Stellt sich nach einem oder zwei Sprints heraus, dass ein Werkzeug nicht das erhoffte Ergebnis bringt, nimmt man das Thema in der Retrospektive auf und diskutiert darüber.

Das Ziel ist immer das gleiche: „Schaffe Mehrwert für den Kunden". Das geht nur durch produktives Arbeiten und ständige Verbesserung.

Vielen Dank an Wolfgang Wiedenroth für die wertvollen Tipps.

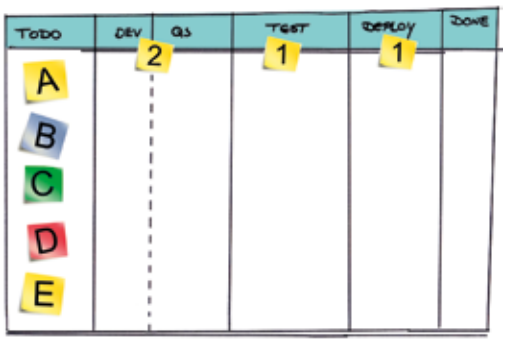


Abbildung 10: WIP-Limits begrenzen die parallele Arbeit

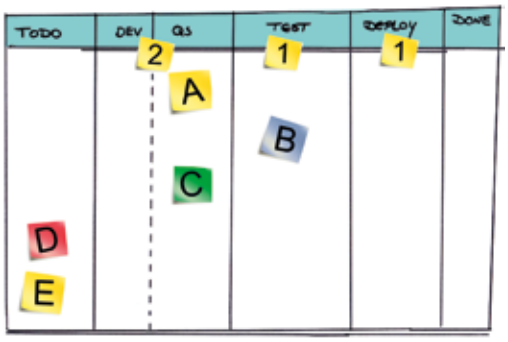


Abbildung 11: WIP-Limits fördern Team-Work

Martin Dilger
martin.dilger@googlemail.com

Martin Dilger ist Senior Consultant bei der PENTASYS AG, Autor und bloggt regelmäßig unter http://splitshade.wordpress.com

Geronimo 3.0 – modulare Hybride fahren gut

Frank Pientka, MATERNA GmbH

Inzwischen sind einige Applikationsserver für Java EE 6 zertifiziert und der Nachfolger Java EE 7 ist für 2014 geplant. Nach GlassFish war Geronimo der zweite Open-Source-Applikationsserver, der die Tests, sowohl das volle Profil als auch das Web-Profil betreffend, erfüllt. Es hat fast ein Jahr gebraucht, bis die finale Version 3.0 erschienen ist. Der Artikel stellt deren hybride Architektur und die neuen Funktionen vor.

Apache Geronimo setzt seit dem Jahr 2005 auf viele seit Jahren bewährte Komponenten wie Tomcat/Jetty, Axis2/CX, Derby, ActiveMQ, OpenEJB, MyFaces, OpenWebBeans, Karaf und Aries auf. Der Applikationsserver hat bisher jede Zertifizierung von J2EE 1.4 bis Java EE 6 erfüllt. In letzter Zeit ist es um den „Indianer" etwas still geworden und er hat mit TomEE inzwischen Enkel bekommen.

Mit einem rein auf OSGi basierenden Kernel ist er noch flexibler geworden. Damit verabschiedet er sich von seinem allerdings immer noch intern verwendeten Modul-Konzept „GBean". Das hat den Vorteil, dass er als Kommandozeilen-Konsole „Apache Karaf" und als OSGi-Blueprint-Container „Apache Aries" einsetzen kann.

Der Nutzer zieht daraus einige Vorteile: Es ist möglich, sowohl Java-EE- als auch OSGi-Anwendungen auf derselben Infrastruktur einzusetzen. Damit kann der Übergang von der alten, monolithischen Welt in die dynamische und modulare „OSGi-Welt" einfacher erfolgen. Intern verwaltet Geronimo alle eingesetzten Einheiten als OSGi-Bundle, sodass diese von beiden Seiten genutzt werden können. Sie sind zentral im Geronimo-Repository abgelegt, das sich an die Maven-Repository-Struktur anlehnt.

Geronimo bietet insbesondere 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 mit Geronimo einen verlässlichen und innovativen Begleiter. Für die hybride Entwicklung von OSGi- und Java-EE-Anwendungen ist Geronimo eine gute Wahl, 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 sich große Unterstützter des JCP wie IBM und JBoss immer mehr bei Apache-Projekten engagieren und so die Open-Source-Fahne weiter hochhalten.

Maßanzug oder Server von der Stange?

Geronimo wird entweder als voll zertifizierte, als Web-Profile zertifizierte oder als Little-G-Variante mit dem integrierten Tomcat heruntergeladen und installiert. Für die Entwicklung kann die Framework-Variante verwendet oder mit SVN und Maven der Server selbst übersetzt werden. Da bisher kein binäres Release von Geronimo 3.0 mit Jetty als Web-Container existiert, müsste man hier selbst Hand anlegen.

Der Server wird am besten mit „geronimo.[bat|sh] run" gestartet, da man sich dann mit „Tab" die 218 Kommandozeilen-Befehle anzeigen lassen kann. Durch die modulare Architektur ist es durch die Werkzeug-Unterstützung in der Web-Konsole „http://localhost:8080/console" oder auf der Kommandozeile mit dem Befehl „assemble-server" recht einfach möglich, sich eine für die eigenen Bedürfnisse maßgeschneiderte Version zu erstellen. Als Minimum sollte man über die funktionale Auswahl gehen und dann folgende Punkte auswählen:

- Assembly groupId: DAOG
- Assembly artifactId: TestServer
- Assembly version: 0.1
- Folgenden Plug-ins
  - Geronimo Plugin Group: Framework
  - Geronimo Plugin Group: WAB Tomcat
  - Geronimo Plugins, System Database: System Database

Anschließend findet sich unter „<GERONIMO_HOME>\var\temp\assembly\TestServer-0.1-bin.zip" das Assembly-Paket. Aktuell funktioniert das jedoch nur, wenn der Server unter Java 6 läuft. Gleiches gilt für den umgekehrten Weg, wenn man eine installierte Anwendung als Plug-in exportieren oder ein Plug-in aus einem Repository-Verzeichnis aus dem Internet installieren möchte, wie für die Beispielanwendungen unter „http://geronimo.apache.org/plugins/samples-3.0.0/" (siehe Abbildung 1).

Nachdem das Assembly-Paket ausgepackt ist, kann man den Server wie bisher starten. Wer das parallel zu einem weiteren Server tun möchte, sollte in der „config/config-substitutions.properties"-Datei mit der Variable „PortOffset = 10" die Portnummern erhöhen, um Port-Konflikte zu vermeiden. Der neue Server ist dann beispielsweise unter „http://localhost:8090" erreichbar. Stoppen lässt er sich entweder aus der Konsole über „stop-server" oder aus dem „shutdown.[bat|sh]"-Skript.

Hello-OSGi

OSGi-Bundles lassen sich wie Java-EE-Anwendungen entweder über das Hot-Deployment-Verzeichnis „deploy", die Web-Konsole oder über die Kommandozeile installieren. Die Web-Konsole bietet die meiste Kontrolle. Andererseits kann man über die Kommandozeile wiederholbar mehrere Geronimo-Instanzen auch „remote" bestücken. Dazu lädt man einfach ein OSGi-Bundle über die Web-Konsole hoch und startet es. Über den OSGi-Enterprise-Standard werden die beiden neuen Archiv-Formate Web Application Bundle (WAB) und Enterprise Bundle Archive (EBA) definiert.

Die Blueprint-Erweiterung im „OSGi Service Plattform Release 4 Version 4.2" unterstützt die Java-EE-Standards „JTA", „JPA", „JDBC", „JNDI" und „JMX". Um diese zu testen, kann man aktuell entweder

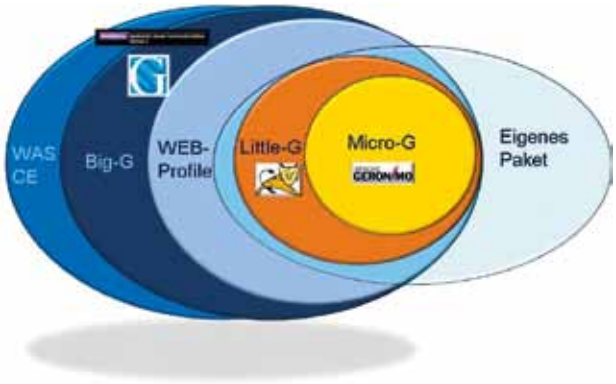


Abbildung 1: Bau dir deinen eigenen Server

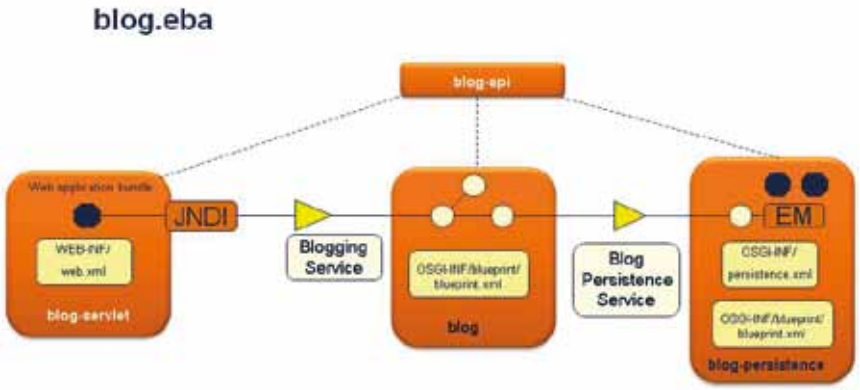


Abbildung 2: Beispiel OSGi-Bundle mit JNDI, JPA und Blueprint

Apache Aries, Eclipse Gemini oder Apache Geronimo verwenden. Zu einem späteren Zeitpunkt werden sie sicher auch die aktuelle OSGi-Version 5.0 unterstützen. Um zu zeigen, dass sich eine für Apache Aries entwickelte Anwendung ohne Änderungen in Geronimo einsetzen lässt, dient das Blog-Beispiel von Aries. Es existiert sowohl mit der Nutzung von JPA und OpenJPA als auch mit der direkten JDBC-Verwendung.

Dazu legt man die vom Beispiel benötigte Datenquelle und die dazugehörigen Tabellen in Geronimo an. Das geht entweder manuell über die Web-Konsole oder über die Kommandozeile mit dem Befehl „deploy deploy tranql-connector-derby-embed-xa-1.7.rar aries-datasource.xml". Danach installiert man auf der Kommandozeile mit dem Befehl „deploy deploy org.apache.aries.samples.blog.jdbc.eba-1.0.0.eba" beziehungsweise „deploy deploy org.apache.aries.samples.blog.jpa.eba-1.0.0.eba deploy" die Beispielanwendung. Anschließend wird die Blog-Anwendung mit „http://localhost:8080/blog" aufgerufen. Abbildung 2 zeigt die Aufteilung der Anwendung in abhängige Bundles mit den dazugehörigen Beschreibungsdateien.

Die Befehle „osgi:info" und „osgi:headers" zeigen die Informationen der „METAINF.MF"-Datei der installierten OSGi-Bundles an. Da die OSGi-Bundles in einem Datei-Cache abgelegt sind, kann es manchmal erforderlich sein, den Cache entweder manuell oder beim Start des Geronimo-Servers mit „geronimo.[bat|sh] run –c" zu löschen.

Zusammenfassung

Da Geronimo von Anfang an Java-EE-zertifiziert und damit abwärtskompatibel ist, sind die Voraussetzungen für einen Applikationsserver-Wechsel gut. Mit der aktuellen Version können sowohl OSGi- als auch Java-EE-Anwendungen parallel betrieben werden. So hat man nicht nur die Möglichkeit, seine Altanwendungen zu migrieren, sondern diese auch schrittweise zu modernisieren. Da Geronimo – typisch Apache – anders als seine Open-Source-Konkurrenten einen sehr langen Support-Zeitraum von acht Jahren (fünf plus drei extended) anbietet, kann man entweder bei der weiterhin unterstützten Version 2.1.8 für Java EE 5 bleiben oder auf die stark verbesserte Version 3.0 und Java EE 6 oder deren Nachfolgeversionen 2.1.9 bzw. 3.0.1 wechseln.

Fazit

Geronimo baut durch Aries eine Brücke zwischen der OSGi- und der Java-EE-Welt. Trotz allem handelt es sich sowohl bei OSGi-Enterprise als auch bei Geronimo um Nischenprodukte. Das könnte sich jedoch ändern, da zum Thema „OSGi EE" schon die ersten Bücher und Erfahrungen vorhanden sind. Die Apache-Lizenz und -Community garantieren, dass Geronimo kontinuierlich weiterentwickelt wird. Ein Vorteil von Geronimo ist seine Wahlfreiheit, was den Web-Container, den Webservice-Stack oder den OSGi-Container angeht. OSGi EE unterstützt die wichtigsten Java-EE-Standards und ist ein wichtiger Helfer im Kampf gegen die zunehmende Komplexität. Wenn die Spezifikation und das TCK für Java EE 7 verfügbar sind, wird auch Geronimo 4.0 wieder diese Spezifikation unterstützen.

Links

- http://geronimo.apache.org
- https://issues.apache.org/jira/browse/GERONIMO
- https://cwiki.apache.org/confluence/display/GMOxPMGT/Documentation
- https://cwiki.apache.org/confluence/display/GMOxDOC30/Customizing+server+assemblies
- https://cwiki.apache.org/GMOxDEV/apache-aries-samples-running-in-geronimo-30.html
- http://aries.apache.org/modules/samples/blog-sample.html
- http://geronimo.apache.org/apache-geronimo-v30-xml-schemas.html
- http://publib.boulder.ibm.com/wasce/Front_de.html
- https://svn.apache.org/repos/asf/aries/tags/samples-1.0.0/

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.