03-2013 | Herbst | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Java aktuell

Java trägt Früchte

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

iJUG Verbund

DOAG 2013 Development 19. Juni 2013, Bonn

Eine Konferenz für den Erfahrungsaustausch von Software-Entwicklern

Themen:

- DB Programmierung: PL/SQL, APEX, Spatial
- BPM & Software-Architektur
- Java & Open Source
- Forms, Reports, ADF und BI Publisher

Aussteller: trivadis, t&p software. our profession., JDD Software, XDEV 3 for Java, iJUG Verbund, ORACLE

Im Fokus: Agile and Beyond – Projektmanagement in der Oracle-Software-Entwicklung Wohin geht die Reise? (Part Two)

DOAG
Deutsche ORACLE-Anwendergruppe e.V.

http://development.doag.org

Wolfgang Taschner
Chefredakteur Java aktuell

99 Flaschen Bier

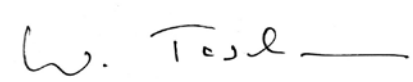
Die Website „http://99-bottles-of-beer.net" basiert auf einer Idee von Tim Robinson. Er begann in den 1990er Jahren, den Text des in Amerika beliebten Lieds „99 Bottles of Beer" in der damals populären Programmiersprache BASIC umzusetzen. Frei übersetzt lautet der Text: „99 Flaschen Bier an der Wand, nimm´ eine herunter und reich´ sie herum, dann sind noch 98 Flaschen Bier an der Wand. 98 Flaschen Bier an der Wand, nimm´ eine herunter …" Das geht so lange, bis alle Flaschen heruntergenommen sind. Bei null angekommen, lautet die Strophe: „Keine Flaschen Bier mehr an der Wand, geh´ in den Laden und kauf´ neue, dann sind 99 Flaschen Bier an der Wand." Dann beginnt wieder alles von vorn.

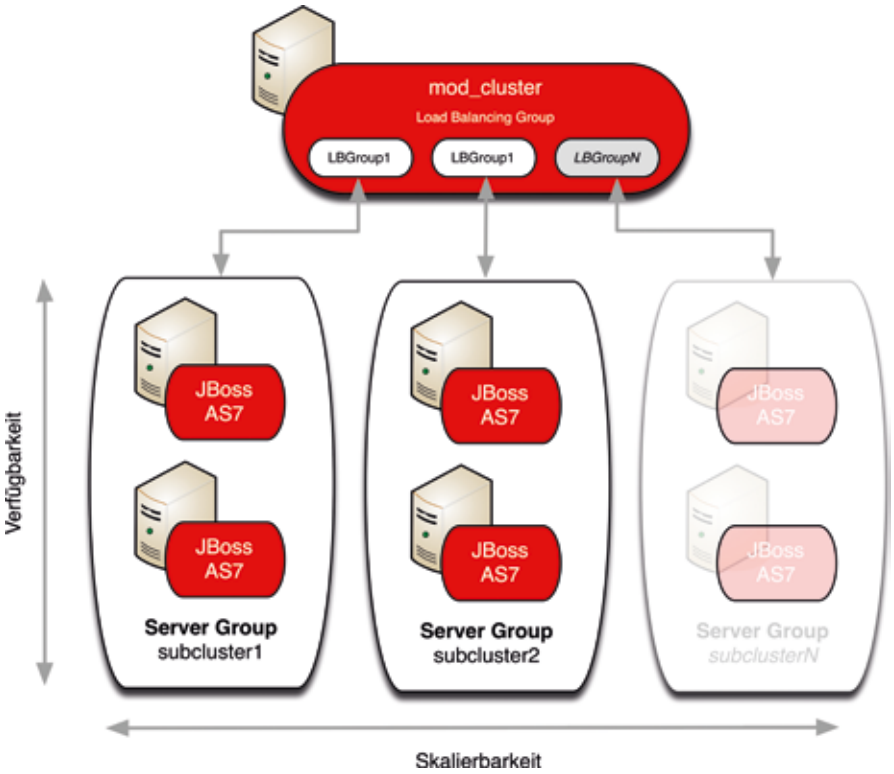
Der relativ einfache Algorithmus hat im Lauf der Jahre zahlreiche Programmierer angespornt – mittlerweile ist das Lied in mehr als 1.500 verschiedenen Programmiersprachen codiert und auf der Website abgelegt worden. Das Spektrum beginnt bei „A+" und endet bei „ZZT". Schon das Schmökern unter den vielen Sprachen ist sehr kurzweilig, allein wenn man den Unterschied zwischen Uralt-Sprachen wie „Cobol" oder dem „Assembler für 8080" und den modernen Sprachen betrachtet. Jeder Besucher der Website kann jede Sprache bewerten und kommentieren. Im Top-Ranking der am besten bewerteten Sprachen sind auf den ersten 25 Plätzen immerhin „Clujure" und „Ruby" vertreten.

„Java" ist natürlich auch unter den 1.500 Sprachen dabei. Die zahlreichen Kommentare zum veröffentlichten Java-Code zeigen, wie sich die Sprache weiterentwickelt hat und wie man das Programm für das Lied unter den verschiedensten Aspekten angehen kann.

Auch in dieser Ausgabe der Java aktuell möchte ich Ihnen eine große Bandbreite an Themen vorstellen, die alle mit Java zu tun haben. Ich hoffe, dass der eine oder andere Artikel Ihnen bei Ihren Projekten hilft, und freue mich wie immer auch über Ihr Feedback an redaktion@ijug.eu.

Ihr





Ein Lastverteiler sorgt dafür, dass Lastverteilung und Failover nur innerhalb eines Subclusters stattfinden. Skalierbare Cluster-Topologien mit dem JBoss AS 7, Seite 11

Wer auf keine Funktion verzichten möchte und schon immer von einer eigenen Rollladensteuerung träumte, findet hier eine Anleitung für den Raspberry Pi, Seite 49

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 ersten Quartal 2013.

21. Januar 2013

Java-Frameworks beeinflussen laut CAST die Software-Qualität

CAST Research on Application Software Health (CAST) hat einen neuen „CRASH"-Report veröffentlicht, in dem der Einfluss von Frameworks auf die Qualität von Software untersucht wurde. Begutachtet wurden 496 Anwendungen mit 152 Millionen Codezeilen von 88 Organisationen aus sechs verschiedenen Branchen; per Daten-Analyse wurden unter anderem die Unterschiede zwischen der Nutzung von Struts, Hibernate und Spring herausgeschält. Nicht überraschend wird festgestellt, dass das Mischen von Java beispielsweise mit C und C++ die Qualität im Durchschnitt mindert – eher überraschend steigt sie aber bei einem Mix etwa mit COBOL oder .NET. Die Nutzung von Struts wiederum führt laut Studie eher zu geringerer, die von Hibernate eher zu höherer Qualität. Framework-Komplexität in Kombination mit mangelndem Training soll hier die Schuld tragen – für diese Aussage allein hätte man aber sicher keine große Studie benötigt. Das Highlight für alle alten, neuen und wiedergewonnen Fans von Java EE: Die Nutzung von purem Java EE ohne andere Sprachen und Frameworks führt zu den höchsten Qualitätswerten. Für alle, die neugierig geworden sind – die Studie kann bezogen werden unter http://www.castsoftware.com/news-events/press-release/press-releases/cast-research-finds-java-frameworks-strongly-influence-security-reliability

31. Januar 2013

JCache nicht in Java EE 7

JCache, der inzwischen wieder aktive JSR 107 (gestartet vor mehr als zwölf Jahren!), hat leider den Sprung unter den Schirm von Java EE 7 verpasst. Die Expertengruppen der beiden Specification Requests haben das gemeinsam entschieden, da JCache zuletzt einige Termine verpasst habe und offensichtlich noch viel zu tun sei. Die Arbeit soll jedoch zügig weitergehen, sodass JCache dann bald als „drop-in"-API genutzt werden kann und auf jeden Fall in Java EE 8 aufgenommen wird, siehe https://blogs.oracle.com/theaquarium/entry/jcache_to_miss_java_ee

1. Februar 2013

Critical Patch Update für Java SE schon da

Das neue Critical Patch Update (SE 7 Update 13 beziehungsweise SE 6 Update 39) ist früher als geplant herausgekommen – offensichtlich unter dem Druck der Sicherheitslücken mit aktiven Exploits, die viele Firmen und Benutzer dazu veranlasst haben, Java im Browser zu deaktivieren beziehungsweise gleich ganz zu deinstallieren. Der iJUG hatte über die Lücken berichtet und versucht, die Diskussionen etwas zu versachlichen: http://www.ijug.eu/index.php?id=1399

19. Februar 2013

Java EE 7 Maven Repository Coordinates

Wer mit Maven arbeitet und schon mal die neuen Java-EE-7-APIs benutzen möchte, wird sich sicher über den GlassFish-Wiki-Eintrag mit den aktuellen Maven-Koordinaten dieser APIs freuen, auf den Oracle-Blogger Reza Rahman dankenswerterweise verwiesen hat: http://wikis.oracle.com/display/GlassFish/Java+EE+7+Maven+Coordinates

Schon wieder ein Critical Patch Update

Da das CPU vom 1. Februar 2013 immer noch bekannte kritische Lücken offengelassen hatte, die nicht rechtzeitig geschlossen werden konnten, hatte Oracle bereits angekündigt, schnell nachzubessern. Das ist nun geschehen, damit stehen wir bei SE 7 Update 15 beziehungsweise SE 6 Update 41.

22. Februar 2013

Der Name „Sun" verschwindet Stück für Stück

Wer schon mal mit Legacy Code gearbeitet hat, weiß, wie hartnäckig sich Namen von Firmen darin halten, die längst nichts mehr mit dem Code zu tun haben oder gar nicht mehr existieren, etwa als Package-Namen. Ein gutes Beispiel ist der Name der Firma, die Java herausgebracht hat, richtig: „Sun". Stück für Stück scheint er nun zu verschwinden – diesmal sind die Namespaces in den Java-EE-7-APIs dran. Den immer noch zahlreichen Sun-Fans lässt das bestimmt das Herz bluten: https://blogs.oracle.com/theaquarium/entry/java_ee_schema_namespace_moved

27. Februar 2013

„Adopt a JSR"-Organisation macht Fortschritte

Im Januar gab es bereits ein erfolgreiches Online-Meeting zu „Adopt a JSR" mit JUG-Repräsentanten. Dazu wurde jetzt ein Follow-up gehalten, in dem es speziell um Tipps für „Hackdays" und ihre erfolgreiche Gestaltung zum Wohle eines JSR ging. Mitte Februar 2013 gab es darüber hinaus ein Meeting für die JSR-Spec-Leads, um ihnen den Sinn des Programms zu erklären und wie sie die Zusammenarbeit mit den JUGs gestalten und nutzen können. Insgesamt scheint das Programm mehr und mehr Fahrt aufzunehmen und sich immer besser zu organisieren. Treffen verpasst oder generell neugierig, was dort besprochen wurde? Dann hilft das Multimedia-Archiv des JCP. Dort finden sich Tonaufnahmen und Folien: http://jcp.org/en/resources/multimedia

Zum (fast) gleichen Thema gibt es am selben Tag einen Blog-Eintrag von Arun Gupta, in dem aufgelistet wird, wie sich die Java-EE-7-JSRs beim Thema „Transparenz", wie vom Java Community Process seit Version 2.8 gefordert, und bei der Beteiligung der Community schlagen. Laut Gupta sind die Transparenz-Vorgaben von allen JSRs vollständig eingehalten. Darüber hinaus gibt es bei den meisten JSRs eine sehr starke Beteiligung von Java User Groups beziehungsweise generell aus der Community. Natürlich sind auch die JSRs beziehungsweise ihre Spec Leads darauf angewiesen, dass ihr Thema auf möglichst viel Begeisterung stößt. Da hat sicher nicht jeder JSR die gleichen Voraussetzungen, aber natürlich sollte ein Spec Lead auch in der Lage sein, Begeisterung zu wecken: https://blogs.oracle.com/arungupta/entry/transparency_and_community_participation_in

5. März 2013

Weiterer Security Patch für Java mit mäßigem Erfolg
Oracle hat einen weiteren Security Patch für Java SE herausgebracht. Wie Heise Online meldet, hat der polnische Sicherheitsexperte Adam Gowdiak, der Oracle bereits auf mehrere Lücken hingewiesen hat, weiterhin Probleme in der neuesten Version feststellen können, http://www.heise.de/open/meldung/Java-trotz-Notfall-Patch-verwundbar-1815992.html

7. März 2013

Von Elstern und gestohlenen Schlüsseln
Im Zusammenhang mit den Security Patches für Java SE passen zwei weitere Meldungen von Heise Online: Erstens, dass bei Java die Prüfung auf zurückgezogene Zertifikate nicht standardmäßig aktiviert ist. Dafür mag es Gründe geben, denn die Prüfung erfordert einen Netz-Zugriff. Aber wenn ein Zertifikat etwa aufgrund eines gestohlenen privaten Schlüssels zurückgezogen wird, sollte das JRE damit signierte Applets nicht mehr akzeptieren und zwar bei allen Nutzern, nicht nur bei denjenigen, die die Prüfung manuell aktivieren. Zweitens haben die ganzen Diskussionen um die Sicherheit inzwischen sogar die Steuerverwaltung geweckt: Ein Sprecher des zuständigen Bayerischen Landesamts für Steuern sagte gegenüber Heise Online: „Die aktuelle Diskussion über Java hat die Steuerverwaltung veranlasst, nach Lösungen zu suchen, die die Nutzung des ElsterOnline-Portals kurzfristig auch ohne Java zulassen, ohne auf die Nutzung von sicherer Zertifikatstechnologie verzichten zu müssen." Das klingt ein bisschen so, als ob nun schnell das Kind mit dem Bade ausgeschüttet werden soll – wir warten gespannt auf die Alternativen.

7. März 2013

Red Hat übernimmt Projektleitung für das OpenJDK 6
Der offizielle Support für das OpenJDK 6 ist ausgelaufen, Bugfixes soll es nur noch für kommerzielle Varianten geben. Jetzt springt allerdings Red Hat in die Bresche und übernimmt laut einer Presseerklärung eine führende Rolle im OpenJDK-6-Projekt. Was das im Detail bedeutet, wird nicht erklärt, aber es geht um „erweiterten Support für die Technologie und ihre Nutzer", was vor dem Hintergrund der jüngst entdeckten Sicherheitslücken sehr wichtig ist. Red Hat nutzt den sich bietenden Raum, um sich einen Namen als verlässlicher Partner im Java-Ökosystem zu machen. SE-6-Nutzer, die noch nicht auf SE 7 umstellen können oder wollen, haben damit weiterhin kostenlosen Support, wenn sie das OpenJDK verwenden.

22. März 2013

TomEE – der Retter von Java EE?
In einem InfoWorld-Artikel wird Apache TomEE als Retter des „in der Zeit eingefrorenen" Java EE beschrieben, mit Zitaten von David Blevins, dem Gründer des Projekts. Dieser antwortet in seinem Blog prompt, dass Java EE keine Rettung benötige, sondern sich mit EE 6 und der ersten öffentlich entwickelten Spezifikation EE 7 ausgesprochen gut entwickle. Die Hauptmotivation für TomEE liege darin, die alte Frage „Tomcat oder Java EE?" überflüssig zu machen, http://blog.dblevins.com/2013/03/saving-java-ee.html

25. März 2013

Apache CloudStack wird Top-Level-Projekt
Die Apache Software Foundation hat angekündigt, dass CloudStack den „Inkubator" verlässt und in Zukunft als Top-Level-Projekt geführt wird. CloudStack, eine Software zum Aufsetzen von „Infrastructure as a Service"-Lösungen, war vor knapp einem Jahr von Citrix an die Apache Software Foundation übergeben worden und wird dort weiter von Citrix-Mitarbeitern, aber auch einem großen und wachsenden Stamm von freien Unterstützern als Open-Source-Projekt weitergeführt, https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces40

4. April 2013

Java EE 7 kurz vor dem Ziel und weitere JSR Updates
Das gesammelte JSR-Update für Februar und März ist auf „blogs.oracle.com" veröffentlicht worden und zeigt jede Menge Bewegung. Der Java-EE-7-Umbrella-JSR sowie viele der darauf referenzierten zu aktualisierenden JSRs wie JAX-RS 2.0 und JSF 2.2 sind in den vergangenen Tagen jeweils als „Proposed Final Draft" veröffentlicht worden. Über sie wird in den nächsten Wochen abgestimmt. Daneben gab es noch eine Reihe von „Maintenance Reviews", etwa für JCA 1.6 und EJB 3.1, „Early Draft Reviews" für die zwei Java-SE-Spezifikationen „Lambda Expressions" und „Annotations on Java Types" sowie einen neuen JSR „Portlet Specification 3.0", https://blogs.oracle.com/jcp/entry/jsr_updates11

8. April 2013

Oracle by Example: Neue Trainings unter anderem zu Java SE 8 Lambda Expressions
Es gibt inzwischen jede Menge freie Trainings zu Java auf der Oracle-Webseite, etwa zu Lambda Expressions. Das volle Programm lässt sich abrufen unter http://www.oracle.com/goto/oll

9. April 2013

Apache Struts 1 ist „End of Life"
Apache Struts 1 ist von Apache offiziell beendet worden. Immerhin war es dreizehn lange Jahre aktiv und eines der ältesten Projekte der Apache Software Foundation. Allerdings liegt das letzte Release auch

schon fünf Jahre zurück. Der Nachfolger Struts 2 ist jedoch weiterhin aktiv, https://blogs.apache.org/foundation/entry/media_alert_the_apache_struts

10. April 2013

Maven-Archetyp für Java EE 7

Eine weitere Erleichterung für Maven-Nutzer, die schon mit der Java-EE-7-Spezifikation arbeiten möchten, ist ein Archetyp, mit dem neue EE-7-Projekte in der richtigen Struktur automatisch angelegt werden können, https://blogs.oracle.com/theaquarium/entry/java_ee_7_maven_archetypes

12. April 2013

OpenJDK: Freie Mitglieder des Governing Board wiedergewählt

Andrew Haley (Red Hat) und Prof. Doug Lea (State University of New York) sind beide als freie („at-large") Mitglieder des OpenJDK Governing Board wiedergewählt worden. Sie ergänzen damit weiterhin die anderen von Oracle („Chair" George Saab, „Lead" Mark Reinhold) und IBM („Vice-Chair" John Duimovich) nominierten Board-Mitglieder als Vertreter der Community. Wahlberechtigt waren alle zu Beginn der Wahlperiode im OpenJDK-Projekt vertretenen Mitglieder. Das sind allerdings nur wenig mehr als hundert, hier kann sich die Java-Community sicher noch aktiver einbringen. Gegenkandidaten gab es nicht, beide wurden einstimmig bei jeweils drei Enthaltungen und 35 beziehungsweise 36 Ja-Stimmen gewählt.

15. April 2013

Java ME 3.3 für ARM Cortex freigegeben

Für den Raspberry Pi steht die Java ME 3.3 Runtime erst als „early access" zur Verfügung. Für den weit verbreiteten ARM-Cortex-M3-Prozessor hat Oracle jetzt die Runtime-Version offiziell freigegeben („general availability"). Terrence Barr von Oracle schreibt in seinem Blog, dass inoffiziell auch die M4-Version unterstützt wird. Für Embedded-Entwickler öffnet sich damit eine große Welt von Mikro-Prozessoren ohne Cross Compilation und ähnlich lästige Dinge, https://blogs.oracle.com/javaspotlight/entry/java_spotlight_episode_125_terrence

17. April 2013

Oracle stopft weitere Sicherheitslücken

Mit dem Update 21 für Java SE 7 (und Update 25 für SE 6) werden nicht nur Fehler im Code behoben, sondern auch einige Dinge generell geändert: Im Java Control Panel gibt es die Sicherheitsstufen „niedrig" und „benutzerdefiniert" nicht mehr. In Abhängigkeit von der gewählten Stufe und der Aktualität des JRE laufen Applets gegebenenfalls gar nicht mehr. Betroffen sind diese, die unsigniert beziehungsweise mit einem Zertifikat signiert sind, das nicht von einer akzeptierten „Certificate Authority" bestätigt ist. Die Sicherheitsdialoge wurden überarbeitet und die Risiken besser herausgestellt. Der Nutzer muss bei steigendem Risiko auch mehr tun, um dies zu akzeptieren. Außerdem gibt es jetzt ein Server-JRE, das kein Java-Plug-in, kein Auto-Update und keinen Installer beinhaltet, http://www.ijug.eu/home-ijug/aktuelle-news/article/java-7-update-21-foerdert-signierte-applets.html

18. April 2013

Java EE 8: Vorschlag zur Verschiebung

Nicht zum ersten Mal muss Mark Reinhold die unangenehme Nachricht verkünden, dass dem „release train" eine Verspätung droht, diesmal für Java SE 8. Er begründet dies hauptsächlich mit der vermehrten Arbeit an Security Fixes und der generell gestiegenen Aufmerksamkeit für „sicheren Code", die auch zu Umstellungen im Entwicklungsprozess geführt hat. Damit hätten einige Features den Meilenstein 6 „feature complete" Ende Januar 2014 nicht einhalten können. Die Alternativen seien nun, entweder den Zug ohne den prominentesten Passagier „Lambda Expressions" weiterfahren zu lassen oder den Fahrplan anzupassen. Reinhold hält die Anpassung für die bessere Alternative. Sein Vorschlag, den er an die offene Mailingliste des OpenJDK schickt, sieht eine Verzögerung von einem halben Jahr vor – statt Anfang September 2013 dann Mitte März 2014, da ein Release um Weihnachten herum keine gute Idee sei. Weihnachten in Deutschland ist in der Regel kürzer, aber vielleicht ist es ja auch nur ein willkommener Grund, noch ein paar Wochen extra herauszuholen. Wenn es denn der Qualität dient … http://mreinhold.org/blog/secure-the-train

26. April 2013

Java EE 8 verzögert

Wie erwartet kündigt Mark Reinhold eine gute Woche nach seinem Vorschlag zur Verschiebung des Java SE 8 Release an, dass diese Variante angenommen wurde. Das neue Release-Datum ist also der 18. März 2014, http://mreinhold.org/blog/hold-the-train

30. April 2013

JSR Updates

Noch ein Update vom JCP: Insgesamt 14 Java-EE-7-JSRs haben in den letzten Wochen den „EC Final Approval Ballot" bestanden und werden nun als finalisierte Releases (Spezifikation, Referenz-Implementierung, Test Compatibility Kit) auf jcp.org veröffentlichen. Hier scheint es also keine weiteren Verzögerungen zu geben, https://blogs.oracle.com/jcp/entry/jsr_updates_april_15

Andreas Badelt
Leiter der DOAG SIG Java

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

Java-API für Batch-Anwendungen

Peter Doschkinow, ORACLE Deutschland B.V. & Co. KG

Eines der neuen und lang ersehnten Features von Java EE 7 ist die Unterstützung von Batch-Anwendungen. Im Rahmen der neuen Teilspezifikation JSR-352 werden die Entwicklung und der Betrieb von klassischen Batch-Applikationen auf der Java-Plattform vereinfacht und standardisiert. Der Artikel führt in die Konzepte und Möglichkeiten dieser neuen Technologie ein, die der Java-Entwickler bei der Implementierung von Batch-Prozessen nutzen kann.

Batch-Anwendungen gehören zu den ersten Computer-Anwendungen, bei denen Programme – auch „Jobs" genannt – automatisch und weitgehend ohne menschliche Intervention im Hintergrund ausgeführt werden. Typischerweise werden dabei große Datenmengen mit oft hohen Anforderungen an Computer-Ressourcen verarbeitet, wie automatisierte Transaktions-Verarbeitung, Datenanalyse- oder „Extract Transform Load"-Data-Warehouse-Anwendungen (ETL). Batch-Anwendungen können sequenziell, parallel, zeitgesteuert oder bei Bedarf ausgeführt werden. Neben Online-Applikationen tragen sie durch unterschiedliche Priorisierung dazu bei, verfügbare Rechenkapazitäten optimal auszunutzen. Historisch bedingt ist der Löwen-Anteil von Batch-Anwendungen noch in Cobol geschrieben. Angesichts der Mächtigkeit der Java-Plattform, der vergleichbar günstigen Verfügbarkeit von Java-Know-how auf dem Arbeitsmarkt und der häufigen Notwendigkeit zur Konsolidierung und Migration auf eine Plattform ist der Bedarf an Unterstützung von Batch-Verarbeitung in Java beträchtlich. In der Java-Welt gibt es bisher nur proprietäre Lösungen dafür, etwa Spring Batch [2] oder WebSphere Batch [3]. Andere Alternativen wie BPMN-/BPEL-basierte Ablaufsteuerung haben einen anderen Fokus und scheinen ein Overkill für das zentrale, aber einfache Read-Process-Write-Muster von Batch-Anwendungen zu sein, die auf die schnelle und effiziente Verarbeitung von Riesen-Datenmengen spezialisiert sind.

Batch-Anwendungen haben gemeinsame Anforderungen wie die Unterstützung von Logging, Checkpoints und Parallelisierung. An ihre Laufzeitumgebung werden ebenfalls gemeinsame Anforderungen gestellt, etwa die Möglichkeit, den Lebenszyklus von Batch-Instanzen zu steuern, zu stoppen und neu zu starten. JSR-352 definiert eine Job Specification Language als Domänen-spezifische Sprache, ein Java-Programmiermodell und eine Laufzeitumgebung für Batch-Anwendungen für Java SE und Java EE, die solchen Anforderungen gerecht werden. Das Ziel ist, ähnlich wie bei anderen Java-Standards wie JPA und CDI, existierende Best-Practices für Batch-Anwendungen zu konsolidieren und zu standardisieren. Nachfolgend sind die Hauptbestandteile von JRS-352 kurz vorgestellt.

Grundkonzepte und die Job-Sprache

In den vergangenen Jahrzehnten hat sich eine Referenz-Architektur für Batch-Anwendungen durchgesetzt, die sich in zahlreichen Implementierungen auf verschiedenen Plattformen wie COBOL/Mainframe, C++/Unix und in letzter Zeit Java/everywhere bewährt hat. Abbildung 1 zeigt eine vereinfachte Darstellung.

Ein Job wird durch die Job-Sprache definiert und kapselt die gesamte Batch-Verarbeitung ein. Er besteht aus mehreren Schritten („Steps"). Jeder Step kapselt eine sequenzielle, unabhängige Verarbeitungseinheit, die nach dem einfachen Muster „Read Process Write" ausgeführt wird und als wiederverwendbare Einheit betrachtet werden kann. Der „ItemReader" repräsentiert das Auslesen von Input-Daten, der „ItemProcessor" kapselt die Geschäftslogik zur Verarbeitung der Datensätze und der „ItemWriter" repräsentiert die Ausgabe der Ergebnisse. Ein „JobOperator" ist die Schnittstelle zur Verwaltung aller Aspekte der Job-Verarbeitung mit Kommandos wie „start", „stop" und „restart" sowie zu „JobRepository", um Informationen über laufende Jobs und Steps zu bekommen.

In JSR-352 gibt es zwei Arten von Steps: „Chunk-Steps" und „Batchlet-Steps". Chunk-Steps sind eine Variation des Step von Abbildung 1 und dienen zur effizienten Massen-Datenverarbeitung mit Unterstützung von Checkpoints. Sie implementieren das am häufigsten genutzte Muster der Verarbeitung in Datenblöcken: Datensätze werden einzeln ausgelesen und verarbeitet, in Datenblöcken akkumuliert und in derselben Transaktion ausgegeben (siehe Abbildung 2).

Batchlet-Steps sind für Task-orientierte, problemspezifische Steps gedacht, für die eine Verarbeitung in Datenblöcken/Chunks nicht geeignet ist. Sie werden einmal ausgeführt, geben einen Exit-Status zurück und müssen einen Stop-Callback implementieren, um gegebenenfalls vom JobOperator storniert zu werden.

Die Job-Sprache, die in JSR-352 definiert wird, spezifiziert eine Batch-Anwendung in der Form eines Jobs, seine Steps

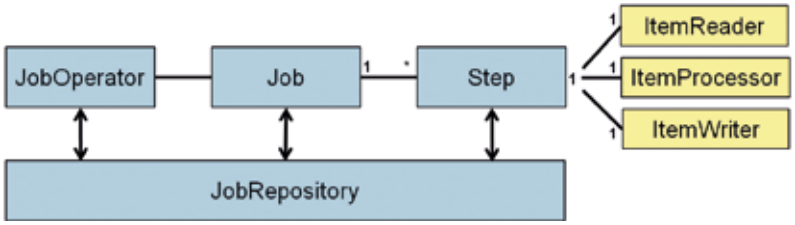


Abbildung 1: Vereinfachte Architektur für Batch-Anwendungen

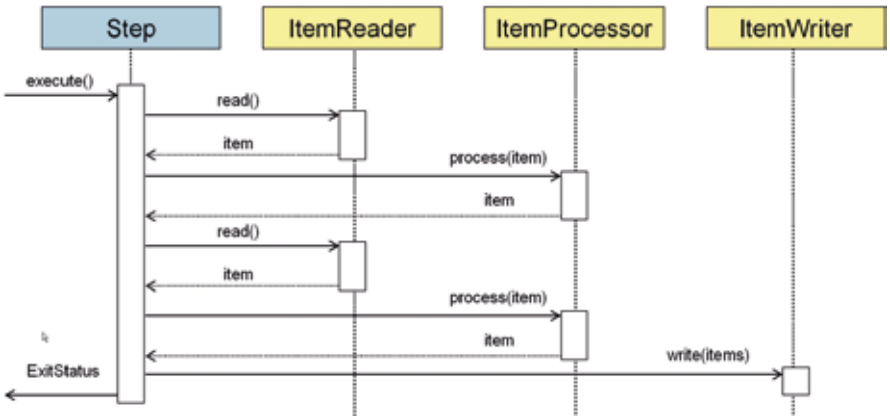


Abbildung 2: Ausführung eines Chunk-Step

und die Reihenfolge ihrer Ausführung. Sie ist in XML implementiert, hat ein zugrunde liegendes XML-Schema und wurde unter anderem von Spring Batch [2] inspiriert. Im folgenden Job-XML-Beispiel werden neben den schon erwähnten Chunk- und Batchlet-Steps auch „Flows" und „Splits" demonstriert (siehe Listing 1).

Ein Flow definiert eine Reihenfolge von Steps, die zusammen als eine Einheit ausgeführt werden, während ein Split eine Menge von Flows definiert, die gleichzeitig ausgeführt werden. Jeder Flow läuft in einem separaten „Thread". Ein Split endet, wenn alle enthaltenen Flows abgeschlossen sind. Die Attribute „reader", „writer" und „processor" im „Chunk"-Element von Chunk-Step „step1" sowie das Attribut „ref" in den „Batchlet"-Elementen der Batchlet-Steps „step2", „step3" und „step4" in Listing 1 referenzieren die entsprechenden Java-Implementierungsklassen.

Alternativ zur Parallelisierung über Splits unterstützt Job XML das gängige Partition-Map-Reduce-Muster, das auch zu extremen Performance-Verbesserungen auf Multi-Core-Systemen führen kann. Dabei werden derselbe Step mehrfach instanziiert und die einzelnen Step-Instanzen in separaten Threads gleichzeitig ausgeführt (Partition-Phase). Zu jeder Step-Instanz gibt es einen „Collector" und einen „Analyzer", die bei ihrer Ausführung ihre Zwischen- und End-Ergebnisse sammeln (Map-Phase).

Nach dem Ablauf aller Step-Instanzen aggregiert der Reducer die Einzel-Ergebnisse zu einem Gesamt-Ergebnis (Reducer-Phase). In [1], Abschnitt 11.3, gibt es eine sehr übersichtliche algorithmische Beschreibung der unterschiedlichen Szenarien der Job-Verarbeitung und der entsprechenden Koordination von Java-EE-Transaktionen, gewähltem Parallelisierungs-Modell und Checkpoint-Behandlung in Chunk-Steps.

```
<job id="job1" xmlns="http://batch.jsr352/jsl">
    <split id="split1" >
        <flow id="flow1" next="flow2">
            <step id="step1" next="step2">
                <chunk reader="MyItemReader"
                       writer="MyItemWriter"
                       processor="MyItemProcessor"/>
            </step>
            <step id="step2">
                <batchlet ref="MyBatchlet2" />
            </step>
        </flow>
        <flow id="flow2">
            <step id="step3" next="step4">
                <batchlet ref="MyBatchlet3" />
            </step>
            <step id="step4" >
                <batchlet ref="MyBatchlet4" />
            </step>
        </flow>
    </split>
</job>
```

Listing 1: Job-XML-Beispiel mit Split, Flows, Chunk- und Batchlet-Steps

Programmiermodell

Das Batch-Programmiermodell ist durch Interfaces und Java-Annotations beschrieben, die den Kontrakt zwischen der Batch-Anwendung und der Batch-Runtime festlegen. Im Job „XML" werden Artefakte referenziert, die einen Teil dieser Interfaces implementieren und zu Laufzeit von der Batch-Runtime instanziiert und genutzt werden. Bezogen auf das Job-XML-Beispiel in Listing 1 bedeutet das, dass der Java-Entwickler jeweils eine Implementierung von „ItemReader", „ItemProcessor" und „ItemWriter" sowie drei Implementierungen von „Batchlet" liefern muss.

Um die Verbindung zwischen referenzierten Namen und Implementierungsklassen herzustellen, kann entweder die CDI-Annotation „@Named" oder die „batch.xml"-Konfigurationsdatei genutzt werden. Eine Implementierung des referenzierten „MyItemProcessor" könnte wie in Listing 2 aussehen und vom Eintrag in „batch.xml" begleitet werden (siehe Listing 3). Eine konsequente Nutzung von CDI- und Java-Annotations würde auch eine reine POJO-Implementierung ohne Einträge in „batch.xml" ermöglichen (siehe Listing 4).

Das Batch-Programmiermodell sieht mehrere Listener-Interfaces vor, die in der Batch-Anwendung als Interceptoren für die einzelnen Phasen ihrer Ausführung dienen können, wie „JobListener", „StepListener", „ChunkListener" und „SkipListener". Context-Annotations wie „@BatchContext" und „@JobContext" sind CDI-Qualifier, die bei Verwendung mit „@Inject" den Zugriff der Anwendung auf die Job- und Runtime-Informationen ermöglichen.

Batch-Anwendungen benötigen kein spezielles Packaging-Format. Sie können im Standard-„jar"-Format oder in jedem anderen Archiv-Format gepackt werden, das von der Ziel-Plattform unterstützt wird, etwa als Web- oder EJB-Modul. Das Konfigurationsfile „batch.xml", das zur Auflösung der im Job XML referenzierten Artefakte dient, soll unter dem Standard-Verzeichnis „META-INF" bei „jar"-Archiven und unter „WEB-INF/classes/META-INF" bei Web-Archiven abgelegt

```
public class MyProcessor implements ItemProcessor<MyInputRecord, MyOutputRecord>{
	…
	@Override
	public MyOutputRecord processItem(MyInputRecord item){
		//process item and return MyOutputRecord
		…
	}
}
```

Listing 2

```
<batch-artifacts xmlns="http://jcp.org.batch/jsl">
	<ref id="MyItemProcessor" class="org.xyz.MyProcessor"/>
	...
</batch-artifacts>
```

Listing 3

```
@Named(„MyItemProcessor")
@ItemProcessor
public class MyProcessor {
	…
	@ProcessItem
	public MyOutputRecord processItem(MyInputRecord item){
		//process item and return MyOutputRecord
		…
	}
}
```

Listing 4

```
JobOperator jobOp = BatchRuntime.getJobOperator();
jobOp.start(„myJob", myJobParameters);
```

Listing 5: Bootstrapping einer Batch-Anwendung

sein. Analog ist für Job-XML-Files das Verzeichnis„META-INF/batch-jobs" vorgesehen.

Laufzeit-Umgebung

Die Batch-Runtime ist für das Laden des Job XML, der dort referenzierten Batch-Artefakte und für seine Ausführung zuständig. Referenzierte Batch-Artefakte können implementierungsabhängig, beispielsweise über CDI, oder mit dem Standard-Archiv- oder ThreadContext-Classloader geladen und instanziiert werden. Das Laden und Starten von Job XML erfolgt über das Interface„JobOperator". Dieser ist die zentrale Schnittstelle für eine interaktive oder automatische Verwaltung aller laufenden Batch-Anwendungen. Die Klasse „BatchRuntime" repräsentiert die JSR-352-Laufzeitumgebung und dient als Factory für den Zugriff auf das „JobOperator"-Interface. Die beiden Zeilen in Listing 5 zeigen das Bootstrapping einer Batch-Anwendung, das beispielsweise in einer asynchronen EJB-Methode oder durch die Nutzung des ManagedExecutorService-API (JSR-236, Concurrency Utilities for Java EE) ausgeführt werden kann.

Die Batch-Runtime definiert auch Interfaces, über die sie ihre Zustands-Informationen und Statistiken in verschiedenen Metriken wie Read-/Write-Count, Commit-Count etc. zur Verfügung stellt. Sie kann direkt auf Java SE oder eingebettet in einem Java EE Application Server laufen. Im letzteren Fall klinkt sie sich über SPI in den Transaktions-Manager des Application-Servers ein, sodass Chunk-Steps in einem transaktionalen Kontext ausgeführt und gegebenenfalls zurückgerollt werden können.

Status und Referenz-Implementierung

Die Experten-Gruppe für JSR-352 wird von Chris Vignola von IBM, früher Chef-Architekt der WebSphere-Batch-Technologie, angeführt und durch die Teilnahme von Oracle, Red Hat, VMware und Credit Suisse verstärkt. Ihre Arbeit erfolgt entsprechend den Richtlinien von JCP 2.8 und ist deshalb von Anfang an öffentlich einsehbar. Die Java-Community kann über die öffentliche E-Mail-Adresse „public@jbatch.java.net" Feedback geben und sich an der Arbeit beteiligen.

Die Spezifikation des Java-API für Batch-Anwendungen ist momentan im Status „Proposed Final Draft", sodass noch mit gewissen API-Änderungen zu rechnen ist [1]. Das Projekt ist unter „jbatch.java.net" beheimatet. Im Unterschied zu anderen JSRs findet man dort noch kein separates Javadoc-Archiv – es kann jedoch vom RI-Archiv [4] extrahiert werden. Die JSR-352-Referenz-Implementierung ist bereits in aktuelle Builds von GlassFish 4.0 integriert. Java-EE-Evangelist Arun Gupta hat in seinem Blog bereits ein Batch-Anwendungsbeispiel veröffentlicht, das auf GlassFish 4.0 b78 läuft [5].

Fazit

Die JSR-352-Spezifikation geht weit über die hier zusammengefassten Features hinaus und beschäftigt sich auch mit der Vererbung von Jobs-, Steps- und Flow-Elementen für ihre bessere Wiederverwendung und mit einer ausführlichen Fehlerbehandlung. Sie ist ein wichtiger Meilenstein auf dem Weg zur Adaption und Migration von Batch-Anwendungen und zur standardisierten Unterstützung ihrer spezifischen Anforderungen in der Java-Welt.

Referenzen

[1] JSR-352, Batch Applications for the Java Platform: http://java.net/projects/jbatch/downloads/download/JSR 352 Proposed Final Draft v1.7.pdf

[2] Spring Batch: http://static.springsource.org/spring-batch/reference/html/index.html

[3] WebSphere Batch: http://www.ibm.com/developerworks/websphere/techjournal/1203_narain/1203_narain.html

[4] JSR-352 RI: http://java.net/projects/jbatch/downloads/download/jsr352-SE-RI-1.0.zip

[5] Arun Gupta Blog: https://blogs.oracle.com/arungupta/entry/chunked_step_using_batch_applications

Peter Doschkinow

peter.doschkinow@oracle.com

Peter Doschkinow arbeitet als Senior Java Architekt bei Oracle Deutschland. Er beschäftigt sich mit serverseitigen Java-Technologien und Frameworks, Web Services und Business Integration, die er in verschiedenen Kundenprojekten erfolgreich eingesetzt hat. Vor seiner Tätigkeit bei Oracle hat er wertvolle Erfahrungen als Java Architect and Consultant bei Sun Microsystems gesammelt.

Skalierbare Cluster-Topologien mit dem JBoss AS 7

Heinz Wilming und Immanuel Sims, akquinet AG

Hohe Verfügbarkeit und hohe Skalierbarkeit sind für viele Anwendungen im Unternehmensumfeld eine unabdingbare Anforderung. Der Einsatz von Clustertechnologien verspricht hier Lösungen, gestaltet sich in der Praxis aber herausfordernder als angenommen. Neben der technischen Komplexität ist auch das Spannungsfeld zwischen Skalierbarkeit und Hochverfügbarkeit aufzulösen.

In der letzten Ausgabe der Java aktuell wurden die grundlegenden Clustering-Konzepte des JBoss Application Server in der Version 7 dargestellt. Dieser Artikel stellt unterschiedliche Cluster-Topologien vor, die auch den Anforderungen an die Skalierbarkeit und die Verfügbarkeit genügen und dabei die Restriktionen der Netzwerk-Infrastruktur und die Besonderheiten größerer Cluster-Umgebungen berücksichtigen.

Einführung

Eine hohe Verfügbarkeit eines Clusters bezeichnet die Fähigkeit einer Cluster-Topologie, bei einem Ausfall einzelner Knoten die Aufgaben nahtlos an andere Knoten der Cluster-Topologie zu übertragen. Hierzu ist es erforderlich, dass sowohl Systemkomponenten als auch transiente Sitzungsdaten redundant vorgehalten werden. Die Verfügbarkeit wird dabei durch Hinzufügen weiterer Knoten erhöht. Dies geht jedoch zulasten der Skalierbarkeit, da die Replikation von Daten mit anderen Knoten der Topologie den Kommunikationsaufwand und Ressourcenbedarf erhöht. Bei einem Cluster mit beispielsweise vier Knoten und jeweils einem Gigabyte Speicher stehen bei einer Replikation auf allen Knoten nicht insgesamt vier, sondern nur ein Gigabyte Speicher für transiente Daten zur Verfügung, da jedes Datum vierfach vorgehalten wird.

Durch das Hinzufügen weiterer Knoten wird der verfügbare Speicher also nicht erhöht, da alle anderen Knoten die Daten des neuen Knotens vorhalten und der neue Knoten die Daten der bereits vorhandenen. Darüber hinaus ist auch ein erhöhter Aufwand für die Verteilung der Daten zu berücksichtigen. Allerdings wird eine höhere Verfügbarkeit durch die Erhöhung der Redundanz erreicht.

Die Schwierigkeit bei der Konzeption eines Clusters ist, sowohl eine hohe Verfügbarkeit als auch eine ausreichende Skalierbarkeit zu erreichen. Die Skalierbarkeit eines Clusters bezeichnet dabei die Fähigkeit der Bewältigung wachsender Last durch Hinzufügen zusätzlicher Ressourcen. Eine gute Skalierung ist beispielsweise beim Betrieb mehrerer voneinander unabhängiger Server gegeben, die keinerlei Kenntnis davon haben, dass sie gemeinsam in einem Cluster arbeiten. Ein Lastverteiler sorgt dafür, dass jeder Knoten gleichmäßig mit Anfragen versorgt wird. Dieses Konzept skaliert sehr gut, aber wenn ein Knoten ausfällt, gehen die transienten Sitzungsdaten verloren – Pech für den Nutzer, der gerade seinen virtuellen Einkaufswagen randvoll geladen hat.

Eine vollständige Replikation und der Betrieb unabhängiger Server sind zwei extreme Konzepte. Es gibt auch Topologien, die sowohl eine gute Skalierbarkeit als auch eine gute Verfügbarkeit aufweisen.

Replikation und Distribution

Im JBoss AS 7 dient der verteilte Cache Infinispan [1] der Replikation transienter Sitzungsdaten. Beispielsweise werden die Sitzungsdaten einer HTTP-Session oder die Daten einer zustandsbehafteten EJB-Komponente in sogenannten „Cache-Containern" in unterschiedlichen Regionen verwaltet. Bei einem Ausfall kann somit der Klient transparent auf einen anderen Server ausweichen und mit den redundant vorgehaltenen Sitzungsdaten weiterarbeiten. Ein Infinispan-Cache-Container kann dabei in verschiedenen Modi betrieben werden. Für Hochverfügbarkeit sind die zwei Varianten „Replikation" und „Distribution" interessant. Replikation wird in den Standard-HA-Konfigurationen des JBoss AS 7 verwendet und entspricht dem, was wir als vollständige Replikation bezeichnet haben (siehe Abbildung 1). Im Distributions-Modus kann im Gegensatz zur Replikation festgelegt werden, wie häufig ein Datum redundant vorgehalten wird (siehe Abbildung 2).

Bei dieser Art der Verteilung wird eine bessere Skalierung erreicht, da die Anzahl der Redundanz unabhängig von der Anzahl der Konten in einem Cluster ist. Dadurch wird es möglich, den Cluster durch das dynamische Hinzufügen beziehungsweise Entfernen von Knoten in den Dimensionen „Speicher" und „Rechenleistung" zu skalieren. Falls ein Knoten ausfällt, reorganisiert sich der Cache, sodass die vorgeschriebene Anzahl von Redundanzen wieder erreicht wird.

Ein Nachteil ist jedoch, dass bei Distribution ein Datum nicht unbedingt auf dem Knoten vorgehalten wird, auf dem es angefragt wurde. Der jeweilige Knoten wird mithilfe einer konsistenten Hash-Funktion [2] ermittelt. Es entsteht eine höhere Netzwerk-Kommunikation. Um diese zu verringern, kann bei Distribution ein First-Level-Cache vor dem eigentlichen Cache eingerichtet sein. Hierbei ist jedoch der zusätzliche Koordinationsaufwand für die Invalidierung von nicht mehr gültigen Daten zu berücksichtigen. Ein solcher Cache kann sich somit auch negativ auf die Skalierbarkeit auswirken.

Synchrone und asynchrone Verteilung

Sowohl im Replikations- als auch im Distributions-Modus können die Daten synchron oder asynchron verteilt sein. Bei einer synchronen Verteilung wird der Aufruf bis zur vollständigen Verteilung blockiert. Im Fehlerfall erfolgt ein Rollback der Transaktion. Bei einer asynchronen Verteilung

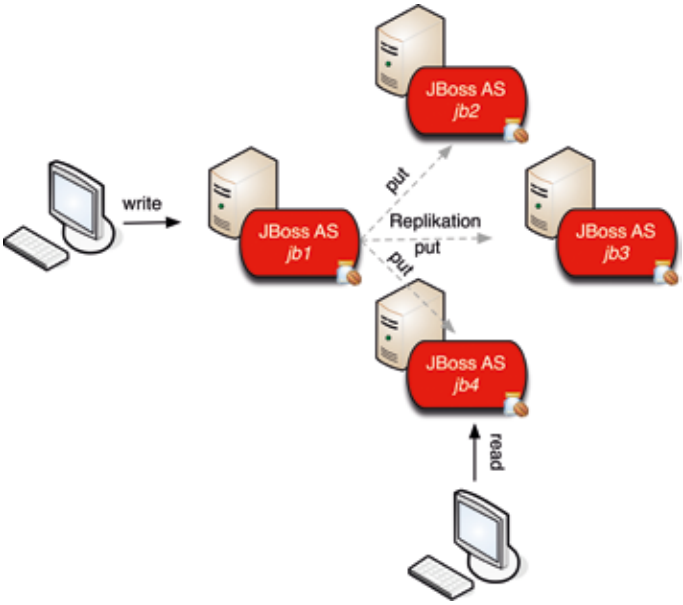


Abbildung 1: Infinispan-Replikation

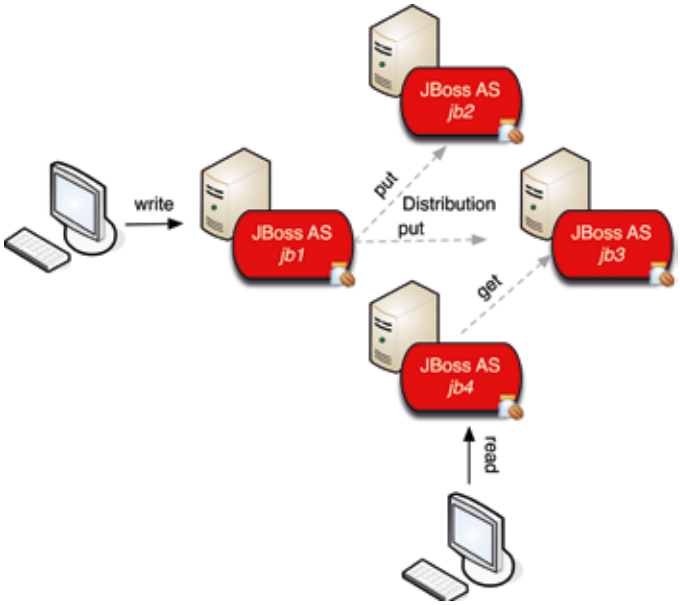


Abbildung 2: Infinispan-Distribution

werden hingegen die jeweiligen Knoten benachrichtigt und der Aufruf nicht blockiert. Kommt es zu einem Fehler während der Verteilung, erfolgt kein Rollback der Transaktion. Fehler werden lediglich protokolliert. Diese Verteilungsart erfordert aufgrund der Asynchronität zudem eine Assoziation der jeweiligen Session mit dem aktuellen Knoten, da Änderungen gegebenenfalls noch in einer internen Queue auf ihre Verarbeitung warten und somit noch nicht auf anderen Knoten verfügbar sind.

Cache-Konfiguration

Der JBoss AS 7 ist bereits für den Betrieb eines Clusters vorkonfiguriert und bietet entsprechende Konfigurationsprofile an. Die Profile „HA“ und „Full-HA“ enthalten im Infinispan-Subsystem unter anderem die Cache-Container „web“, „ejb“ und „hibernate“ (siehe Listing 1). Der letztgenannte agiert als Second-Level-Cache für Hibernate. Hierfür wird der spezielle Modus „Invalidierung“ [2] genutzt.

Der Container „web“ speichert die HTTP-Sitzungsdaten. Im Container „ejb“ werden die transienten Daten von zustandsbehafteten EJB-Komponenten abgelegt. Für „web“ und „ejb“ ist sowohl eine Variante für Replikation („repl“) als auch eine Variante für Distribution („dist“) vorkonfiguriert. Um anstelle des Replikations-Modus den Distributions-Modus zu verwenden, kann das „default cache“-Attribut des jeweiligen Cache-Containers den vorkonfigurierten Distribution Cache referenzieren.

Effiziente Kommunikation mit JGroups

Infinispan nutzt für die Cluster-Kommunikation JGroups [4]. Es ist ebenfalls als eigenständiges Subsystem im JBoss AS 7 eingebunden und unterstützt sowohl Punkt-zu-Punkt-Kommunikation (Unicast) als auch Gruppen-Kommunikation (Multicast). Dabei abstrahiert JGroups von der jeweiligen Netzwerk-Topologie auf Basis eines Transportprotokolls und eines konfigurierbaren Protokollstapels. JBoss AS 7 verwendet standardmäßig UDP als Transportprotokoll. Neben UDP können auch TCP und TUNNEL als Transportprotokoll zum Einsatz kommen. TUNNEL ist, wie der Name suggeriert, zur Vereinfachung der Penetration von Firewalls gedacht, indem die Ver-

```
<subsystem xmlns="urn:jboss:domain:infinispan:1.3">
  ...
  <cache-container name="web" default-cache="dist" ...>
    <replicated-cache name="repl" ...>
      ...
    </replicated-cache>
    <distributed-cache name="dist" ...>
      ...
    </distributed-cache>
    ...
  </cache-container>
  <cache-container name="ejb" default-cache="dist" ...>
    <replicated-cache name="repl" ...>
      ...
    </replicated-cache>
    <distributed-cache name="dist" ...>
      ...
    </distributed-cache>
    ...
  </cache-container>
  <cache-container name="hibernate" ...>
    ...
  </cache-container>
  ...
</subsystem>
```

Listing 1

teilung an die Empfänger über einen Router erfolgt. UDP verwendet IP-Multicasts, TCP versendet an jeden Empfänger einen TCP-Unicast. Bei UDP und TUNNEL muss jeder Multicast vom Server nur einmal versendet werden, während bei TCP jeder Multicast an jeden Clusterknoten gesondert zugestellt wird.

Aufbauend auf dem jeweiligen Transportprotokoll liegen diverse zusätzliche Protokolle im Stapel. Beim Senden durchläuft eine Nachricht den Stapel von oben nach unten, sie passiert also zuletzt das Transportprotokoll. Beim Empfangen wird der Stapel genau umgekehrt durchlaufen. Neben Protokollen zur Cluster-Formierung und dem Auffinden fehlerhafter Knoten befinden sich in der Standard-Konfiguration von JGroups die Protokolle UFC (Unicast-Flow-Control) und MFC (Multicast-Flow-Control). Diese stellen sicher, dass Pakete in einer Frequenz versendet werden, in der die Empfänger diese auch empfangen und verarbeiten können. Ist die Bandbreite des Netzwerks durch hohes Kommunikationsvolumen ausgereizt, aber noch Prozessorkapazität vorhanden, dann kann JGroups den Netzwerkverkehr komprimieren. Hierzu wird der Protokollstapel um das Protokoll COMPRESS erweitert, wie Listing 2 zeigt.

```
<!-- ... -->
<protocol type="FRAG2"/>
<protocol type="COMPRESS"/>
<!-- ... -->
```

Listing 2

Angepasst an das Netzwerk mit JGroups

In manchen Netzwerken, insbesondere in Netzwerken aus verteilten Sub-Netzen, ist IP-Multicast nicht verfügbar. Damit scheidet UDP als Transportprotokoll aus. Eine Alternative ist das TCP-Transportprotokoll. Da JGroups für Gruppennachrichten mit TCP die Kommunikation durch einzelne Verbindungen zu jedem Gruppenmitglied realisiert, wird die Kommunikation in diesem Fall mit steigender Clustergröße immer ineffizienter. In der Praxis sind jedoch viele Switches und Router für TCP optimiert. Deshalb kann TCP trotzdem sowohl eine höhere Nachrichtenrate als auch einen höheren Nachrichtendurchsatz liefern [5]. Allerdings steigt mit der Größe des Clusters die Netzwerklast.

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.1" default-stack="tcp">
  <stack name="udp">
    ...
  </stack>
  <stack name="tcp">
    ...
  </stack>
</subsystem>
```

Listing 3

Im JGroups-Subsystem des JBoss AS 7 ist bereits ein kompletter, auf TCP basierender Protokollstapel bereitgestellt. Der TCP-Protokollstapel wird durch Ändern des „default-stack"-Attributs in Benutzung genommen (siehe Listing 3).

Damit ist die Sache aber leider noch nicht erledigt, denn in diesem Protokollstapel wird zum Auffinden anderer Clusterknoten IP-Multicast über das Protokoll MPING verwendet. Daher muss ein anderes Protokoll als Alternative zum Auffinden anderer Clusterknoten her. JGroups bietet eine große Auswahl an PING-Protokollen an, angefangen bei FILE_PING, das ein gemeinsames Dateisystem nutzt, über Protokolle wie S3_PING und RACKSPACE_PING, die Cloud Storages verwenden, bis hin zu TCPPING. Bei dem Protokoll TCPPING werden die Clusterknoten jeweils Anhand einer statischen Liste initialer Clusterknoten konfiguriert. Über diese Liste versucht jeder Clusterknoten, eine Verbindung zu den konfigurierten Knoten aufzubauen. Sobald eine Verbindung zu einer initialen Mindestanzahl an Knoten vorhanden ist, wird der Cluster formiert. Im folgenden Listing ist eine Beispielkonfiguration für TCPPING gezeigt (siehe Listing 4).

Das Subcluster-Konzept

Wie bereits angerissen, ist die Infinispan-Konfigurations-Distribution eine Lösung, die sowohl Hochverfügbarkeit als auch Skalierbarkeit bietet und mit JGroups eine effiziente Kommunikation auch in größeren Umgebungen ermöglicht. Eine weitere Möglichkeit ist die Unterteilung größerer Cluster-Topologien in mehrere Subcluster. Innerhalb eines jeden Subclusters werden die transienten Sitzungsdaten redundant vorgehalten, aber nicht über die Grenzen eines Subclusters hinweg (siehe Abbildung 3). Somit kann der Cluster gut skalieren, indem Subcluster hinzufügt beziehungsweise entfernt werden.

Die Größe eines Subclusters kann das Maß an Verfügbarkeit bestimmen. Auf einem so aufgebauten Cluster kann auch ein sogenanntes „Rollendes Update" durchgeführt werden, indem jeweils ein kompletter Subcluster aktualisiert wird. Im Gegensatz zu Distribution besteht auch nicht das Problem, dass der Abruf eines Datums häufig über das Netzwerk erfolgen muss, da jeder Subcluster-Knoten alle replizierten Daten vorhält.

Subcluster konfigurieren

Um das Subcluster-Konzept umzusetzen, sind im Wesentlichen zwei Konfigurationen erforderlich. Zu einem muss die Replikation auf bestimmte Knoten begrenzt und zum anderen der jeweilige Lastverteiler so anpasst werden, dass ein Failover nur über die Grenzen eines Subclusters hinweg erfolgt, wenn keine Server des Subclusters mehr verfügbar sind. Die Größe eines

```
<protocol type="TCPPING">
 <property name="initial_hosts">
  1.2.3.4[7600],1.2.3.5[7600]
 </property>
 <property name="num_initial_members">2</property>
 <property name="timeout">2000</property>
</protocol>
```

Listing 4

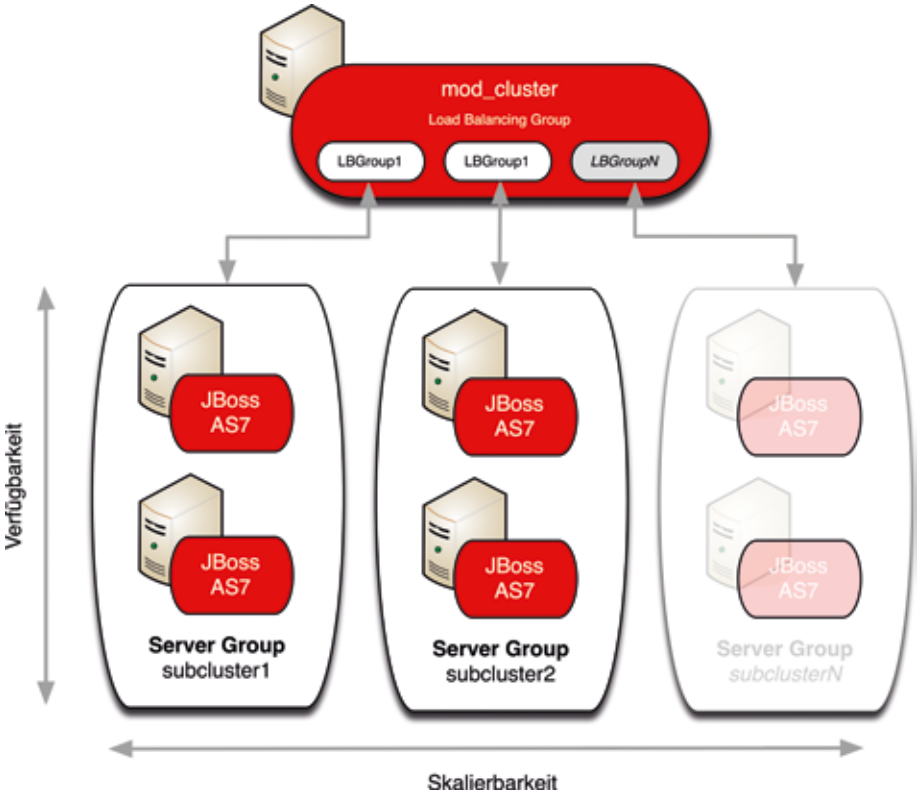


Abbildung 3: Innerhalb eines Subclusters wird Infinispan-Replikation betrieben. Ein Lastverteiler sorgt dafür, dass Lastverteilung und Failover nur innerhalb eines Subclusters stattfinden

Subclusters lässt sich mit der Standard-HA-Konfiguration beschränken. Dazu wird für jeden Subcluster eine gesonderte Multicast-Adresse verwendet. Diese kann sowohl bei der „domain.sh" als auch bei der „standalone.sh" über den Parameter „-u" angegeben werden. Ein Aufruf sieht dann beispielsweise so aus: „./standalone.sh -c standalone-ha.xml -u 230.0.1.4".

Die Lastverteilung lässt sich bei Verwendung von mod_cluster über sogenannte „Lastverteilungsgruppen" realisieren. Lastverteilungsgruppen arbeiten so, dass Lastverteilung und Failover nur innerhalb einer solchen Gruppe erfolgen, es sei denn, es existieren keine Server mehr in dieser Gruppe. Dafür muss jeder Subcluster eine eigene Lastverteilungsgruppe haben. Diese werden auf Seiten des JBoss AS 7 im mod_cluster-Subsystem konfiguriert (siehe Listing 5).

```
<subsystem xmlns="urn:jboss:domain:modcluster:1.1">
 <mod-cluster-config load-balancing-group="LBGroup1" ...>
  ...
 </mod-cluster-config>
</subsystem>
```

Listing 5

Fazit

Mit den grundlegenden Technologien „JGroups" und „Infinispan" für die Cluster-Kommunikation und für die Verteilung transienter Daten sind mit einer etwas angepassten Konfiguration auch größere Cluster-Topologien realisierbar. Die Teilung einer großen Cluster-Topologie in Subcluster ermöglicht neben einer guten Skalierung auch die Aktualisierung einzelner Komponenten des Clusters ohne Ausfallzeiten der gesamten Umgebung. Hierbei müssen jedoch weitere Restriktionen, wie zum Beispiel ein gemeinsames Datenbank-Schema, berücksichtigt werden. Die Beispiele aus diesem Artikel beruhen auf der aktuellen Version des JBoss AS 7. Für den Einsatz in Produktiv-Umgebungen ist jedoch die gesondert qualitätsgesicherte Enterprise Application Platform (EAP 6) von Red Hat empfohlen. Diese ist ebenfalls Quell-offen und basiert auf dem JBoss AS 7 der Community.

Links

[1] http://www.jboss.org/infinispan
[2] http://en.wikipedia.org/wiki/Consistent_hashing
[3] https://docs.jboss.org/author/display/ISPN/Clustering+modes#Clusteringmodes-InvalidationMode
[4] http://www.jgroups.org/manual-3.x/html/index.html
[5] http://www.jgroups.org/perf/perf2006/Report.html

Heinz Wilming
heinz.wilming@akquinet.de

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

Immanuel Sims
immanuel.sims@akquinet.de

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

Lightweight AOP mit CDI und JPA

Prof. Dr. Michael Helbig, Hochschule Rosenheim

Ab Java EE 6 kann man mithilfe von „Contexts and Dependency Injection" (CDI, JSR 299) sogenannte „Cross-Cutting-Concerns" (wie Logging, Transaktionen, Security etc.) elegant von der eigentlichen Business-Logik abspalten, also aspektorientiert programmieren (AOP). Auch das „Java Persistence API" (JPA, JSR 220) bietet mit dem EntityListener eine solche Eigenschaft auf Datensatz-Ebene, etwa relevant für Auditierung.

Der Artikel zeigt an konkreten Code-Beispielen CCCs, AOP, CDI sowie JPA wie man die Struktur bestehender Anwendungen verbessert. Als Erstes werden die verwendeten Begrifflichkeiten erklärt: Cross-Cutting-Concerns (CCC) sind sogenannte „querschnittliche Belange" einer Software. Die zugehörigen Code-Fragmente sind normalerweise quer über den gesamten Code verstreut, da sie durch herkömmliche Programmierung nicht so einfach modularisiert werden können [1]. Beispiele, die fast in jeder Software vorkommen, sind meist nicht-funktionale Anforderungen, die nicht zur eigentlichen Geschäftslogik gehören:

- Logging
- Security-Anforderungen
- Fehlerbehandlung
- Datenbankzugriff
- Transaktionsverwaltung
- Audit-Fähigkeit

Weitere Informationen dazu stehen unter [2]. Um diesen Code an einer Stelle zentral zu formulieren und an den richtigen Stellen automatisch anzuwenden, steht das allgemeine Konzept der aspektorientierten Programmierung (AOP) zur Verfügung [3]. Hier sind diese CCCs von der Geschäftslogik getrennt, was in einem besser lesbaren Code resultiert. Zur konkreten Umsetzung stehen neben dem mächtigen, aber schwergewichtigen AspectJ im Java-EE-6-Standard sowie in Java SE „CDI" und „JPA" zur Verfügung. Zuvor werden CCCs an einem Beispiel vorgestellt.

Kontoverwaltung als Beispiel

Es sind die Konten einer Bank zu verwalten. Man denke an einen Service, der den aktuellen Kontostand speichert (siehe Listing 1). Neben Performance-Logging (a) und Transaktionshandling (b) werden dem Account-Datensatz ein Zeitstempel sowie ein User hinzugefügt (c). Darüber hinaus

```
@RequestScoped
public class AccountService {
 @Inject Logger log;
 @Inject EntityManager em;
 ...

 public void saveAccount(Account account) throws Exception {
  // (a) Performance Logging
  long time = System.nanoTime();
  log.debug("[saveAccount] params=[account="+account+"]");

  // (b) JPA Transaction Handling
  EntityTransaction tx = em.getTransaction();
  try {
   if (!tx.isActive()) { tx.begin(); }

   // (c) Audit
   account.setAcc_last_edit_usr(... .getRemoteUser());
   account.setAcc_last_edit_ts(new Date(System.currentTimeMillis()));

   // BUSINESS LOGIC: save to db
   if (account.getId() == null) { em.persist(account); }
   else { account = em.merge(account); }

   // (d) Write Audit-Log
   AccountAudit accountAudit = new AccountAudit();
   accountAudit.setAcc_balance(account.getAcc_balance());
   ...
   em.persist(accountAudit);

   // (b) JPA Transaction Handling
   if (tx != null && tx.isActive()){ tx.commit(); }
  } catch (Exception e) {
   if (tx != null && tx.isActive()){ tx.rollback(); }
   throw e;
  }

  // (a) Performance Logging
  log.debug("[saveAccount] return - " +
   TimeUnit.MILLISECONDS.convert(System.nanoTime()-time,TimeUnit.NANOSECONDS) +
    " ms");
 }

 public void deleteAccount(Account account) throws Exception { ... }
}
```

Listing 1: Service ohne AOP

```
@InterceptorBinding
@Target({TYPE, METHOD})
@Retention(RUNTIME)
public @interface PerformanceLogging { }
```

Listing 2: Annotation für das InterceptorBinding

wird ein Audit-Log (d) geschrieben. Es wird also eine separate Datenbank-Tabelle mit den für die Auditierungen notwendigen Daten über die durchgeführte Aktion gefüllt. Technisch befinden wir uns in einer CDI-Managed-Bean – der Autor hat bewusst auf eine EJB verzichtet, die viel Funktionalität bereits bietet (siehe Listing 1).

Es ist viel technischer Code vorhanden, der eigentlich nichts mit der Geschäftslogik zu tun hat und den Entwickler behindert. Wir werden diesen jetzt mithilfe von AOP entschlacken. Konkret realisiert wird dies mit CDI und JPA.

Part 1: AOP mit CDI

CDI ist viel mehr als ein Dependency-Injection-Framework. Prominente Implementierungen sind JBoss Weld (Referenz-Implementierung), Apache Open WebBeans und Resin CanDI. In CDI interessieren uns wegen AOP neben „Producers" (realisiert das Factory-Method-Pattern) und „Events" (realisiert das Observer-Pattern, etwa für lokales Messaging) hauptsächlich „Interceptors" und „Decorators". Letztere setzen das Decorator-Pattern um und machen domainnahes AOP möglich – es können also Domain-Objekte mit Zusatzfunktionalität angereichert werden. Dazu im Fazit mehr.

Für unser Beispiel verwenden wir lediglich Interceptors. Diese stehen schon seit Längerem EJBs zur Verfügung – mit CDI jetzt auch „normalen" Klassen, was wichtig ist, falls wir keinen EJB-Container haben (in Tomcat, Jetty etc.). Wir werden jetzt Performance-Logging (a) und Transaktionshandling (b) mit je einem Interceptor herauslösen (siehe Listings 2 bis 5). Einen Interceptor erstellt man in drei Schritten (hier beispielhaft für den „PerformanceLogging"-Interceptor):

1. Annotation für das InterceptorBinding (siehe Listing 2)
2. Interceptor-Implementierung (siehe Listing 3)
3. Aktivierung in META-INF/beans.xml (siehe Listing 4)

```
@Interceptor
@PerformanceLogging
public class PerformanceLoggingInterceptor {

 @AroundInvoke
 public Object performanceLogging(InvocationContext ic) throws Exception {
  // logic before method call
  // (a) Performance Logging
  Logger log = LoggerFactory.getLogger(ic.getMethod().getDeclaringClass().getName());
  long time = System.nanoTime();
  String methodName = "[" + ic.getMethod().getName() + "]";
  log.debug( methodName + " starting... params=" +
    LogUtil.toLogString(ic.getParameters()) });
  try {
   // method call
   Object result = ic.proceed();

   // logic after method call
   // (a) Performance Logging
   log.debug( methodName + " return result=" + result +
    " - " + TimeUnit.MILLISECONDS.convert(System.nanoTime()-time,TimeUnit.NANOSEC-
ONDS) +
     " ms");
   return result;
  } catch (Exception e) {
   log.error( methodName + " ERROR params={}" +
     LogUtil.toLogString(ic.getParameters()),e});
   throw e;
  }
 }
}
```

Listing 3: Interceptor-Implementierung

Bei der Interceptor-Implementierung markiert man die Methode, die den eigentlichen Methodenaufruf abfängt, mit der Annotation „@AroundInvoke". Über den „InvocationContext" können wir uns alle Informationen über die eigentlich aufgerufene Methode besorgen. An diese wird durch „context.proceed()" übergeben.

In Listing 4 ist beim Eintrag in der Datei „beans.xml" die Reihenfolge der Interceptors zu beachten, da diese so wie dort angegeben ausgeführt werden. Außerdem könnte man einfach einen Interceptor durch Löschen des Eintrags quasi ausschalten, ohne die Anwendung neu bauen zu müssen. Den Interceptor „Transactional" für Transaktionshandling (b) kann man auf die gleiche Weise implementieren.

Jetzt folgt der angenehme Teil: „Code Deletion, Teil 1". Unseren Service aus Listing 1 können wir jetzt vereinfachen (siehe Listing 5). Die Code-Fragmente (a) und (b) löschen wir einfach und stellen an deren Stelle die beiden Annotationen „@PerformanceLogging" und „@Transactional". Jetzt können wir diese generische Funktionalität auch auf jede weitere Methode wie etwa „deleteAccount(Account account)" anwenden.

Man könnte jetzt natürlich einwenden: Warum lagern wir den Code nicht einfach in Hilfs-Klassen/-Methoden aus? Man könnte ja beispielsweise die Hilfs-Methoden „beginPerformanceLog(…)" und „endPerformanceLog(…)" definieren. Aber: Diese müsste der Entwickler in jeder Methode

```
<beans>
 <interceptors>
  <class>de.bigdev.aop.PerformanceLoggingInterceptor</class>
  <class>de.bigdev.aop.TransactionalInterceptor</class>
 </interceptors>
</beans>
```

Listing 4: Aktivierung in META-INF/beans.xml (Reihenfolge beachten!)

```
public class AccountListener {

 @PrePersist @PreUpdate
 public void setUserAndTs(Object entity) {
  Account account = (Account) entity;
  // (c) Audit
  account.setAcc_last_edit_usr(... .getRemoteUser());
  account.setAcc_last_edit_ts(new Date(System.currentTimeMillis()));
 }

 @PostPersist @PostUpdate
 public void writeAuditLog(Object entity){
  Account account = (Account) entity;
  // (d) Write Audit-Log
  AccountAudit accountAudit = new AccountAudit();
  accountAudit.setAcc_balance(account.getAcc_balance());
  ...
  // obtain EntityManager via JNDI (in JPA 2.1 use injection)
  EntityManager em = (EntityManager) BeanManagerJNDI.getBeanByName("em");
  em.persist(accountAudit);
 }
}
```

Listing 5: Service mit AOP mittels CDI

zu Beginn und am Ende aufrufen. Er müsste auch geeignete Parameter übergeben (Name der Methode etc.) – also mehr über diese Hilfsmethoden wissen. Mithilfe von CDI passiert dies alles deklarativ (Annotation à la „Mach es", fertig) anstelle von imperativ (schreibe jetzt den Anfangs-Log-Eintrag, schreibe am Ende den zweiten etc.). Dies wäre also nur eine Kompromisslösung.

Jetzt müssen wir nur noch den Account-Datensatz anreichern (c) und das Audit-Log (d) schreiben. Da dies auf Datensatz-Ebene abläuft und von jedem Teil der Anwendung erfolgen soll, fiel der Entschluss, auf einen anderen Mechanismus zurückzugreifen.

Part 2: AOP mit JPA

JPA ist mittlerweile der Standard für das Arbeiten mit relationalen Datenbanken. Der Datenbankzugriff ist mithilfe des EntityManager schon perfekt gekapselt. Will man JPA für AOP nutzen, steht der EntityListener mit folgenden Events zur Verfügung:

- PostLoad
- „After Select Trigger" wie das Filtern von Daten
- Pre-, PostPersist
- „Before/After Insert Trigger"
- Pre-, PostUpdate
- „Before/After Update Trigger" (hier: Dirty Check. Eintreten nur, falls Entity geändert)
- Pre-, PostRemove
- „Before/After Delete Trigger"

```
@PrePersist @PreUpdate
 public void setUserAndTs(Object entity) {
  Account account = (Account) entity;
  // (c) Audit
  account.setAcc_last_edit_usr(... .getRemoteUser());
  account.setAcc_last_edit_ts(new Date(System.currentTimeMillis()));
 }

 @PostPersist @PostUpdate
 public void writeAuditLog(Object entity){
  Account account = (Account) entity;
  // (d) Write Audit-Log
  AccountAudit accountAudit = new AccountAudit();
  accountAudit.setAcc_balance(account.getAcc_balance());
  ...
  // obtain EntityManager via JNDI (in JPA 2.1 use injection)
  EntityManager em = (EntityManager) BeanManagerJNDI.getBeanByName(„em");
  em.persist(accountAudit);
 }
}
```

Listing 6: EntityListener-Implementierung

Man kann bei Eintreten eines dieser Events auf die Entities zugreifen und zusätzliche Logik hinzufügen. Bei uns ist dies Logik (c) und (d) aus Listing 5. Um sie querschnittlich verfügbar auszulagern, müssen wir jetzt konkret folgende Schritte tun:

1. Implementierung einer EntityListener-Klasse (siehe Listing 6)
2. Den EntityListener an der Entity registrieren (siehe Listing 7)

Die EntityListener-Implementierung ist eine ganz gewöhnliche Klasse. Lediglich die Methoden sind durch Annotationen den Events zugeordnet. Diese werden dann bei Auftreten des Events ausgeführt. Hier kann man dann auf die Entity zugreifen, die als Parameter der Methode deklariert ist. Noch eines ist zu bemerken: Leider kann man in einen EntityListener in JPA 2.0 noch nicht injizieren. Hier muss man sich den EntityManager manuell, etwa via JNDI besorgen. Diese mangelnde Zusammenarbeit von JPA mit CDI ist aber in JPA 2.1 behoben.

Alternativ könnte man die Methoden der Entity-Klasse direkt in die Entity stecken. Dies ist eine Geschmacksfrage des zuständigen Architekten. Die hier vorgestellte Vorgehensweise bringt den Vorteil, dass man den EntityListener an mehreren Entities registrieren kann.

Jetzt folgt wieder ein schöner Schritt: „Code Deletion, Teil 2". Wir können unseren Service aus Listing 5 noch weiter vereinfachen und die Code-Fragmente (c) und (d) einfach löschen (siehe Listing 8). Es bleibt nur noch die eigentliche Business-Logik übrig. Der Entwickler der Service-Methode

```
@Entity
@EntityListeners(AccountListener.class)
public class Account { ... }
```

Listing 7: Registrierung des EntityListener an der Entity

```
@RequestScoped
public class AccountService {
 ...
 @PerformanceLogging @Transactional
 public void saveAccount(Account account) throws Exception {
  // BUSINESS LOGIC: save to db
  if (account.getId() == null) { em.persist(account); }
  else { account = em.merge(account); }
 }
 ...
}
```

Listing 8: Service mit AOP mittels CDI und JPA

„saveAccount(…)“ muss überhaupt nichts mehr über das Audit-Log wissen. Dies passiert jetzt automatisch an anderer Stelle.

Wir haben gesehen, dass AOP ein sehr elegantes Mittel ist, um sich auf die eigentliche Business-Logik zu konzentrieren. Sie birgt auf der anderen Seite aber auch die Gefahr, dass der Entwickler nicht mehr weiß, was eigentlich alles passiert, da dies nicht wie in Listing 1 im Code steht. Es stehen an der Service-Methode nur noch eine oder mehrere Annotationen beziehungsweise im Falle von JPA gar nichts.

Als Best Practice rät der Autor, AOP mit Bedacht einzusetzen, um nicht in der „Wartungshölle“ zu landen. Für technische nicht-funktionale Anforderungen, wie hier beschrieben, sei AOP fast ausnahmslos empfohlen, da dies die Lesbarkeit des wichtigen Codes stark erhöht; dagegen wird für fachliche funktionale Anforderungen (Business-Logik) von AOP (realisierbar etwa durch CDI Decorators) eher abgeraten, da man hier besser den sequenziellen Ablauf ohne Seiteneffekte erkennen soll.

Referenzen und weitere Informationen

[1] http://de.wikipedia.org/wiki/Cross-Cutting_Concern
[2] http://architects.dzone.com/articles/example-cross-cutting-concerns
[3] http://de.wikipedia.org/wiki/Aspektorientierte_Programmierung
[4] http://www.bigdev.de

Prof. Dr. Michael Helbig
hel@bigdev.de

Michael Helbig ist Professor für Mathematik und Informatik an der Hochschule Rosenheim. Als Freelancer ist er passionierter Java-EE-Entwickler, Berater und Trainer. Privat betreibt er einen Blog unter http://www.bigdev.de/blog.

Software modular bauen

Ulf Fildebrandt, SAP AG

Modularität wird in der Zukunft eine immer größere Bedeutung bei der Software-Entwicklung spielen. Moderne Systeme verwenden Frameworks wie OSGi und Spring, um die Java-Anteile zusammenzusetzen. Aber Frameworks reichen nicht aus, denn sie gewährleisten allein noch keine gute Architektur. Design Pattern und Regeln sind notwendig, um Systeme zu erstellen, die auch über Jahre weiterentwickelt werden können.

Zu Beginn der Software-Entwicklung entstehen die meisten Programme durch einzelne Entwickler, manchmal arbeiteten auch einige wenige Entwickler zusammen. Im Laufe der Zeit werden die Teams immer größer und auch die Systeme, die erstellt wurden, wachsen. Genau wie bei den Teams Strukturen geschaffen werden, bilden die Entwickler innerhalb der Software Abgrenzungen, um die Komplexität beherrschen zu können. Im schlechten Fall entspricht die Struktur der Software der organisatorischen Struktur, im guten Fall werden fachlich definierte Module erstellt.

Definition der Module

Mit der Modul-Definition ist bereits das Zauberwort gefallen. Ein Modul ist eine abgeschlossene, funktionale Einheit einer Software, bestehend aus einer Folge von Verarbeitungsschritten und Datenstrukturen. Entscheidend für ein Modul ist die Trennung von Schnittstelle und Implementierung.

Die Schnittstelle eines Moduls definiert die Daten-Elemente, die das Modul als Eingabe benötigt und als Ergebnis der Verarbeitung liefert. Die Implementierung enthält den tatsächlichen Programmcode. Module dienen in der Software-Entwicklung zwei Zielen: Erweiterbarkeit und Ersetzbarkeit.

Abbildung 1 zeigt verschiedene Module. Eines davon ist nicht direkt mit den anderen verbunden. Damit soll angedeutet werden, dass dieses Modul die Schnittstellen der anderen verwendet und auf diese Weise mit ihnen interagiert. Der Zugriff auf die interne Implementierung ist nicht gestattet.

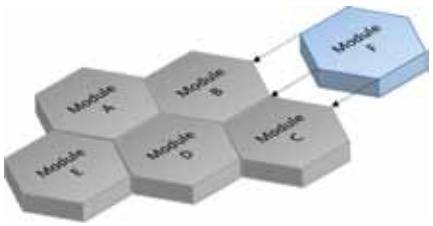


Abbildung 1: Erweiterbarkeit

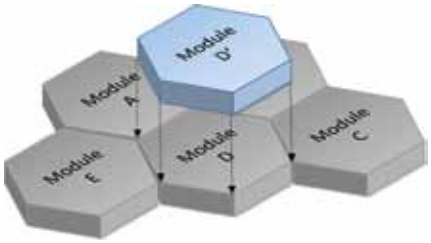


Abbildung 2: Ersetzbarkeit

Das zweite Ziel der Software-Entwicklung besteht darin, Teile der Software wie in Abbildung 2 zu ersetzen, weil das Programm an neue Anforderungen angepasst werden muss und die alte Implementierung nicht mehr umgestellt werden kann.

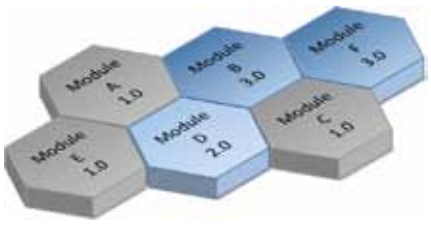


Abbildung 3: Weiterentwicklung eines Systems

Wenn die Teile eines Systems diese Kriterien erfüllen, kann es über die Zeit verändert werden, sodass es niemals komplett neu geschrieben werden muss (siehe Abbildung 3). Gerade in der heutigen Zeit mit den häufig und sich schnell ändernden Anforderungen ist dieses Verhalten sehr erstrebenswert, denn meistens werden Systeme über viele Jahre gewartet, ohne dass sie neu geschrieben werden können. Die Änderungen müssen in Schritten vorgenommen werden und Module mit ihren klar definierten Schnittstellen sind eine Lösung für dieses Problem.

Modularität zieht sich durch die verschiedenen Abstraktions-Ebenen eines Systems hindurch. Entscheidend ist die Entkopplung der Teile auf den Ebenen der Architektur. Auf der untersten Ebene, Coding, muss der Sourcecode so gestaltet sein, dass keine direkten Abhängigkeiten zwischen den Klassen erzeugt werden, zumindest keine unerwünschten. Eine Ebene darüber müssen die Module geeignet definiert sein, um den Anforderungen der Ersetzbarkeit und Erweiterbarkeit zu genügen. Auf der höchsten Ebene muss das gesamte System die Module zusammenbauen, ohne Zyklen und damit Kopplung zu erlauben.

Modulare Laufzeitumgebung

Nach der Definition eines Moduls stellt sich die Frage, wo es ausgeführt wird. Eine normale Laufzeit führt Anweisungen aus, aber um Modularität zu erreichen, bietet es sich an, eine modulare Laufzeitumgebung zu verwenden. Da Module in eine Schnittstelle und eine Implementierung aufgeteilt sind, muss die modulare Laufzeitumgebung diese Aufteilung unterstützen. Die Implementierung ist komplett gekapselt und nicht außerhalb des Moduls erreichbar. Nur die Schnittstelle ist verfügbar. Genau diese Eigenschaften müssen durch die Laufzeitumgebung unterstützt werden.

Im Laufe der letzten Jahre hat sich auf der Java-Seite OSGi als Standard für eine modulare Laufzeitumgebung durchgesetzt. Innerhalb einer OSGi-Laufzeitumgebung werden Bundles ausgeführt. Ein Bundle exportiert Java-Packages, um sie zur Verwendung freizugeben. Abhängigkeiten werden durch importierte Packages definiert. Die gesamte Implementierung

ist nach außen nicht erreichbar. Listing 1 zeigt, wie die Beschreibung eines Bundles in OSGi aussehen könnte.

Dabei ist vor allen Dingen auf die beiden Anweisungen für Packages zu achten, denn über die exportierten Packages wird bekannt gegeben, was nach außen sichtbar ist, also die Schnittstelle. Die importierten Packages müssen von anderen Bundles exportiert werden und definieren, was das Bundle verwenden darf. OSGi in Java benutzt Classloading für die Entkopplung. Jedes Bundle besitzt einen eigenen Classloader, sodass nur Klassen geladen und verwendet werden können, die in der Beschreibung des Bundles spezifiziert wurden.

Für alle weiteren Betrachtungen ist es notwendig, dass eine solche Strukturierung in Modulen vorausgesetzt wird. Die Software, in Java geschrieben, besitzt nicht nur eine Struktur durch die Packages und die damit verbundenen Namenskonventionen, sondern jedes Bundle gruppiert zusammengehörende Code-Teile als Implementierung und eine Schnittstelle für Verwender.

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Basic
Bundle-SymbolicName: Basic
Bundle-Version: 1.0.0.build20120412
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Export-Package: arch.test
Import-Package: org.osgi.framework
```

Listing 1

```
public interface IDataAggregator {
        public List<IDataItem> get();
        public void addDataSource(IDataSource inf);
        public void addDataSource(List<IDataSource> infs);
}
```

Listing 2

Design Pattern für Modularität

Design Pattern sind jedem Software-Entwickler durch das Standardwerk der Software-Entwicklung bekannt (Gamma, Helm, Johnson, Vlissides: Design Pattern). Dabei handelt es sich um Implementierungsvorgaben, mit denen immer wiederkehrende Probleme einheitlich gelöst werden können. Genau wie ein Rad nicht immer wieder erfunden werden muss, können Design Pattern für verschiedene Systeme und in verschiedenen Kontexten angewendet werden. Eine Menge von Design Pattern dient auch dazu, klare Grenzen zwischen den Teilen eines Software-Systems zu ziehen. An dieser Stelle sollen nur zwei genannt werden, die sehr gut geeignet sind, Modularität auf Coding-Ebene zu erreichen. Eine „Façade" ist eine einheitliche und meist vereinfachte Schnittstelle zu einem Teil der Software. Die Ähnlichkeit zur Definition eines Moduls ist sehr offensichtlich. Die Schnittstellen-Definition in Listing 2 beschreibt die Façade eines Daten-Aggregators. Zugriffe darauf erfolgen nur noch über die Schnittstelle.

Eine „Factory" definiert eine Schnittstelle zur Erzeugung einer Familie von Objekten, wobei die konkreten Klassen der zu instanziierenden Objekte nicht näher festgelegt werden. Durch diese Abstraktion entkoppelt eine Factory die Verwendung eines Objekts von der Erzeugung. Im Rahmen der Modularität sind diese beiden Funktionen nach der Einführung einer Factory voneinander getrennt. Die Anweisung in Listing 3 zeigt die Verwendung einer Factory.

Beide Design Pattern bieten eine Schnittstelle an. Sowohl die Erzeugung von Objekten als auch der Zugriff auf Objekt-Instanzen wird somit im Coding bereits von Schnittstellen strukturiert.

SOLID

Sind die Design Pattern im Prinzip auf Coding festgelegt, gibt es eine Gruppe von Prinzipien, die Robert Martin definiert hat: SOLID. Jeder Anfangsbuchstabe steht für ein Prinzip: Single Responsibility, Open-Closed, Liskov, Interface Segregation und Dependency Injection. Genau wie die Design Pattern sind diese Prinzipien für Coding sehr sinnvoll, sie lassen sich aber auch eine Ebene höher auf Module anwenden. Zunächst einmal sollen jedoch diese Prinzipien erklärt werden.

Dependency Injection ist ein Prinzip, bei dem die direkte Abhängigkeit zwischen zwei Objekten aufgehoben wird. Bei schlechtem Coding würde ein Objekt ein anderes unmittelbar verwenden. Die beiden Objekte wären auf alle Zeiten miteinander gekoppelt. Wenn man Dependency Injection anwendet, dann wird eine Schnittstelle im verwendenden Objekt definiert. Das zu verwendende Objekt implementiert diese Schnittstelle. Die beiden Objekte haben also eine Schnittstelle ausgehandelt.

Ein Framework sorgt dafür, dass eine Instanz des zu verwendenden Objekts erzeugt und in das andere Objekt injiziert wird. Durch die Schnittstelle sind die beiden Implementierungen nicht mehr eng miteinander gekoppelt. Sie können ausgetauscht werden, solange sie sich an die Schnittstelle halten.

Dependency Injection hat sich in den letzten Jahren durchgesetzt und wird von vielen Frameworks angeboten. Spring war sicher eines der ersten Frameworks, die Dependency Injection zum Prinzip erhoben. JEE 5 hat diese Art der Pflege von Abhängigkeiten übernommen.

Um das Prinzip von Dependency Injection zu verdeutlichen, wird an dieser Stelle kurz eine solche Deklaration gezeigt. Zunächst einmal geht es darum, das verwendende Objekt zu definieren, im vorliegenden Fall eine Instanz der Klasse „DisplayServlet". Diese Klasse besitzt eine Property „dataAggregrate". Der entscheidende Punkt ist, dass durch diese Deklaration das Framework die Aufgabe übernimmt, die Property mit einer Instanz zu belegen (siehe Listing 4). Die Instanz mit der Referenz „DataAggregate" wurde auf genau dieselbe Weise deklarativ erzeugt (siehe Listing 5).

An dieser Stelle soll nur kurz angemerkt sein, dass Spring und OSGi sich voneinander unterscheiden. Spring kann als Framework innerhalb einer bestehenden Lösung verwendet werden, OSGi legt den Grundstein

```
IDataAggregator aggregator =
        DataAggregatorFactory.getInstance(tableObject.getType());
```

Listing 3

```
<bean id="DataDisplay" class="arch.datadisplay.ui.DisplayServlet">
        <property name="dataAggregate" ref="DataAggregate"/>
</bean>
```

Listing 4

```
<bean id="DataAggregate"
        class="arch.dataaggregate.DataAggregate">
</bean>
```

Listing 5

für die Implementierung. OSGi ist demnach invasiver und bietet auch noch andere Eigenschaft wie die Versionierung von Modulen. Im Beispiel eines Bundle-Deskriptors war die Versionsnummer enthalten, während Spring die Beans nicht versioniert.

Es ist möglich, beide Frameworks gleichzeitig zu verwenden. Hier wurde Spring gewählt und darauf verwiesen, weil es sich dabei um das bekannteste Dependency-Injection-Framework handelt. In OSGi gibt es ähnliche Frameworks zur Deklaration von Abhängigkeiten. „Blueprint" ist die Einbettung der Spring-Konzepte in OSGi, davor wurden „Declarative Services" verwendet. Entscheidend ist aber die Verwaltung der Abhängigkeiten durch das Framework, nicht mehr durch das Coding selbst. Die Objekte werden voneinander entkoppelt.

Der zweite Grundsatz, der einen näheren Blick wert ist, ist das Liskov-Prinzip. Darunter versteht man, dass ein Objekt einer Klasse durch ein Objekt einer Subklasse ersetzt werden kann. Durch die Beziehung der Klasse zu ihrer Subklasse wird festgelegt, dass jede Instanz der Klasse durch eine Instanz der Subklasse ersetzt werden kann. Vereinfacht gesagt, definiert die Klasse wieder einen Vertrag, der von den abgeleiteten Klassen eingehalten werden muss. Ersetzbarkeit ist zum Prinzip erhoben worden, zunächst auf Klassen-, aber auch auf Modulebene.

Schichten

Nachdem durch die Design Pattern und SOLID hauptsächlich Prinzipien für das Coding und Module angesprochen wurden, ist es an der Zeit, sich über das System an sich Gedanken zu machen. Die Definition von Modulen ist ein notwendiger Schritt für die modulare Software-Entwicklung, die besten Module nützen allerdings nichts, wenn sie beliebig miteinander interagieren können.

Um diesen Gedanken zu verdeutlichen, stelle man sich einfach eine Menge von Modulen vor, bei der jedes Modul wieder mit den anderen verbunden ist. Durch die klar definierten Schnittstellen wäre es möglich, jedes Modul auszutauschen, in der Realität jedoch gestaltet sich ein solches Vorgehen als sehr schwierig, da eine größere Anzahl von Abhängigkeiten dazu führt, dass die Module nicht mehr einzeln betrachtet werden können. Sie erscheinen in ihrem Verhalten wie ein einzelnes Modul.

Daher ist man in komplexen Systemen dazu übergegangen, die Module zu gruppieren und geeignet anzuordnen. Das gruppierende Element ist ein Subsystem. Ein Subsystem bündelt eine Menge von semantisch zusammengehörenden Modulen in dem zu entwickelnden System. Sollte ein neues System entwickelt werden, kann die Zusammenstellung der Module selbstverständlich auch wieder neu definiert werden.

Die Subsysteme werden jetzt verschiedenen Schichten innerhalb der Architektur eines Systems zugewiesen. Ein Subsystem darf nur Subsysteme tieferer Schichten verwenden. Auf diesem Weg soll eine zu große Kopplung der Module und damit der Subsysteme vermieden werden.

Je tiefer ein Modul in einer Schicht angeordnet wird, desto größer ist normalerweise der Grad der Wiederverwendung. Daraus ergibt sich umgekehrt, dass ein Subsystem so hoch wie möglich angeordnet sein sollte, um ungewünschte Abhängigkeiten zu vermeiden. Je tiefer ein Subsystem angeordnet ist, desto mehr Subsysteme dürfen es verwenden.

Abbildung 4 verdeutlicht dieses Prinzip. In diesem Beispiel existieren fünf Schichten. Die unterste Schicht bietet eine Schnittstelle an, um Datenquellen allgemein zu definieren. Auf der Schicht darüber sind die zwei Datenquellen angesiedelt. Nochmal eine Schicht oberhalb existiert eine Anwendung zur Darstellung der Daten.

Zyklen

Ein spezieller Aspekt von Schichten in der Architektur besteht darin, dass Zyklen verboten sind. Abhängigkeiten von Subsyste-

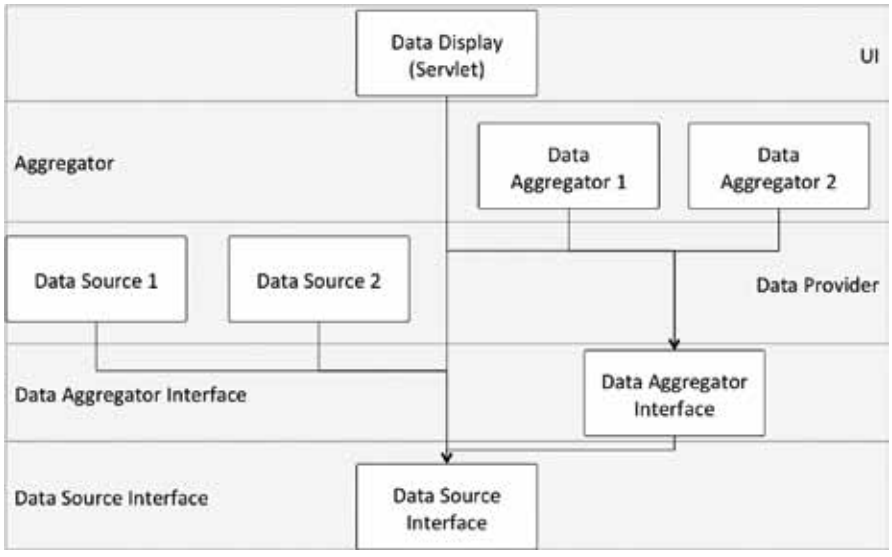


Abbildung 4: Einfaches Beispiel für Schichten

men aus einer tieferen in höhere Schichten sind nicht erlaubt, um die Kopplung zwischen den Subsystemen minimal zu halten. Jede weitere Abhängigkeit eines Subsystems führt dazu, dass es schwieriger wird, es auszutauschen oder zu erweitern.

Dieses Prinzip gilt nicht nur für die Ebene der Subsysteme, sondern genauso für Coding. Java-Klassen, die sich gegenseitig referenzieren, führen dazu, dass die Klassen schwerer weiterzuentwickeln sind, weil eine Änderung zu unerwarteten Folgen in den anderen Klassen führen kann.

Wenn man Software modular entwickeln will, sollte man Zyklen auf allen Ebenen der Architektur vermeiden. Durch Zyklen werden die entsprechenden Teile der Software gekoppelt und können danach als ein Modul gelten, auch wenn sie formal in zwei Modulen oder Klassen abgelegt sind. Sie sind nicht mehr unabhängig voneinander zu entwickeln. Es handelt sich nicht mehr um modulare Software.

Metriken

Obwohl man es anders vermuten könnte, ist Modularität nicht nur durch Prinzipien zu erreichen, sondern sie kann auch direkt im Coding geprüft werden. Wie bei allen anderen Tatsachen, die geprüft werden sollen, muss man Messbarkeit erreichen – dafür existieren Metriken. An dieser Stelle sollen nur ein paar beschrieben werden, um das Prinzip zu verdeutlichen.

Die Kopplung („coupling“) eines Moduls wird durch die Abhängigkeiten zu anderen Modulen bestimmt. Dabei sind die ausgehenden („efferent coupling“ = Ce) und eingehenden („afferent coupling“ = Ca) Abhängigkeiten zu unterscheiden. Die Instabilität berechnet sich über „I = Ce / (Ca + Ce)“. Je mehr Module das betrachtete Modul verwenden, desto instabiler wird das Modul selbst. Als zweite Metrik soll die Kapselung vorgestellt werden. Sie gibt das Verhältnis aller Typen (T) im Vergleich zu den privaten Typen (pt) an. Damit berechnet sich die Kapselung über „ep = pt / T“.

Je mehr sich die Zahl der privaten Typen der Zahl aller Typen annähert, desto mehr nähert sich die Kapselung dem Wert 1. Wenn die Kapselung falsch verstanden wird und einfach die gesamte Implementierung offengelegt wird, dann geht pt gegen 0 und somit auch ep. Daraus folgt unmittelbar, dass schmale Schnittstellen eines Moduls direkt erkennbar werden. Es existieren weitere Metriken, um Module und Coding zu vermessen. Anmerkung: Die beiden beschriebenen Metriken dienen nur als Beispiel.

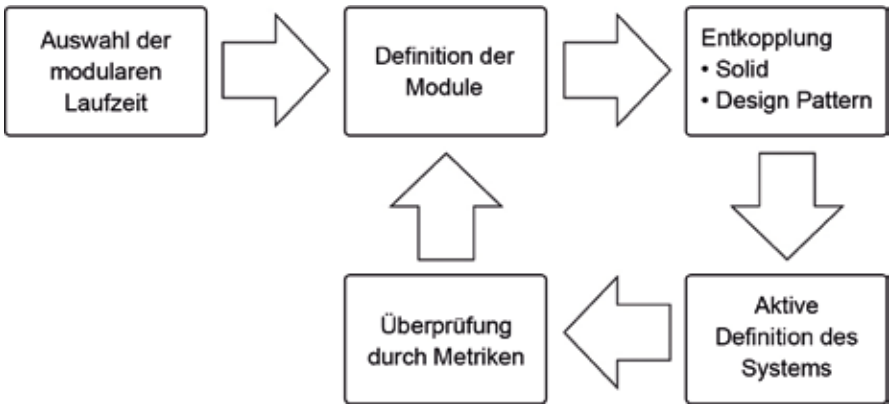


Abbildung 5: Verfahren zur modularen Software-Entwicklung

Anwendung der Verfahren

Nach der Vorstellung der Konzepte und Prinzipien stellt sich die Frage, wie man diese anwenden muss, um ein System modular aufzubauen (siehe Abbildung 5). Als Grundbedingung wird eine modulare Laufzeitumgebung angenommen. Für das System wird ein Framework ausgewählt, um Module auszuführen. Danach besteht der erste Schritt darin, die Module passend zu den Anforderungen zu definieren. Der Zweck eines Moduls muss klar festgelegt sein und dementsprechend auch die Schnittstelle dieses Moduls. Für die Definition der Schnittstelle sind Design Pattern wie „Factory“ und „Façade“ notwendig.

Für das Modul selbst müssen Dependency Injection und auch das Liskov-Prinzip beachtet werden. Ein Modul und das Coding darin sollten nie direkte Abhängigkeiten aufbauen, das Zusammenspiel der Module sollte vielmehr über Dependency Injection gewährleistet sein. In der weiteren Implementierung muss darauf geachtet werden, dass das Liskov-Prinzip eingehalten wird. Die Schnittstelle sollte aussagekräftig genug sein, um die Ersetzbarkeit des Moduls zu garantieren.

Wenn das implementierte Modul diese Konzepte erfüllt, dann kann man an die Zusammenstellung des Systems gehen. Die Module werden zu Subsystemen zusammengefügt und diese werden den Schichten zugewiesen. An dieser Stelle findet eine weitere Überprüfung der Abhängigkeiten von Modulen statt. Unerwünschte Abhängigkeiten können durch Dependency Injection vermieden werden.

Am Ende wird das System vermessen. Man ermittelt die Metriken für die Module und analysiert, ob Kopplung und Kapselung der Vorgabe entsprechen. Die Überprüfung dieser Metriken lässt sich in einen automatischen Build integrieren, sodass die Architektur eines Systems automatisch getestet und jeder Verstoß gemeldet wird. Die Weiterentwicklung eines Systems läuft jetzt auf die iterative Anwendung dieses Verfahrens hinaus.

Ulf Fildebrandt
ulf.fildebrandt@sap.com

Ulf Fildebrandt arbeitet für SAP seit 1998 in verschiedenen Bereichen als Development Architect. Dort hat er Java in vielen Bereichen wie Eclipse und auf dem Server verwendet. Während der letzten Jahre war er verantwortlich für verschiedene Produkte im SOA-Umfeld auf Basis von Java und OSGi.

Software – weniger kaufen, mehr bauen

Erik Dörnenburg, ThoughtWorks Deutschland GmbH

Wenn ein Unternehmen eine neue IT-Lösung braucht, möglicherweise, weil sich die Geschäftsbedingungen ändern oder weil ein existierender manueller Prozess automatisiert werden soll, so fragen sich die Umsetzungsverantwortlichen üblicherweise schnell: „Sollen wir die Lösung selbst bauen oder sollen wir sie kaufen?" Lange Zeit war die gängige Auffassung, dass es grundsätzlich besser sei, eine Lösung zu kaufen. Selbst zu entwickeln kam nur dann in Frage, wenn es keine passende Lösung auf dem Markt gab.

In diesem Artikel werden die Gründe für die Präferenz von Kauf-Lösungen untersucht. Vor allem will der Autor die vorherrschende Meinung hinterfragen und dabei erklären, warum er glaubt, dass die Veränderungen der letzten zehn Jahre in der Software-Entwicklung dafür sorgen, dass sich die Antwort auf die „Kaufen oder Bauen"-Frage weg vom Kaufen und mehr in Richtung „Bauen" bewegt. Dabei geht der Autor nicht so weit zu behaupten, dass Bauen die neue Standard-Antwort sein sollte, aber er meint, Software selbst zu entwickeln sollte öfter in Erwägung gezogen werden, sogar dann, wenn passende Lösungen auf dem Markt existieren.

Seiner Erfahrung nach wird die Entscheidung zum Kauf selten auf Basis von Argumenten getroffen, die zeigen, warum der Kauf eine gute Idee ist. Üblicherweise geht die Analyse eher dahin zu belegen, warum selbst zu entwickeln eine schlechte Idee ist. Diese Tatsache spricht in gewisser Weise für sich selbst. Es klingt, als wäre die Eigenentwicklung die eigentlich logische Antwort und als machten es Probleme damit unausweichlich, den Kaufansatz zu wählen. Daher stellt sich die Frage, welche Probleme diejenigen sind, die die Entscheider dazu bringen, lieber eine fertige Lösung zu kaufen.

Software-Entwicklung ist riskant

Das am meisten genannte Argument gegen die Eigenentwicklung ist die Annahme, dass Software-Entwicklung risikoreich sei. Es gibt beliebig viele Geschichten von gescheiterten Software-Projekten, von Projekten, die den Zieltermin nicht einhalten, oder von solchen, die das Budget nicht nur um ein paar Prozentpunkte überschreiten, sondern um ein Vielfaches der ursprünglich veranschlagten Kosten. Eine Möglichkeit, diese Risiken zu vermeiden, ist, die Software zu einem vorher festgelegten Preis von einem Anbieter zu kaufen. Das verlagert das Risiko von Budgetüberschreitungen direkt zum Anbieter. Bei Standard-Software ist es zusätzlich üblicherweise so, dass diese bereits fertiggestellt ist, bevor der Kauf getätigt wird. In dem Fall besteht nicht einmal das Risiko, dass die Software zum benötigten Zeitpunkt nicht fertig ist.

Die offensichtliche Frage ist, ob Software-Entwicklung wirklich so grundlegend risikobehaftet ist, dass viele Projekte scheitern mussten. Auf der einen Seite ist die Software-Entwicklung natürlich eine junge Disziplin, zum Bespiel im Vergleich zur Architektur oder zur industriellen Produktion von Gütern. Aus diesem Grund gibt es weniger Erfahrung und viele Ideen und Konzepte sind noch nicht in wiederholbaren Prozessen umgesetzt. Auf der anderen Seite jedoch ist Software flexibel und unendlich formbar. Mit den richtigen Techniken sollte es möglich sein, nicht nur exakt das zu produzieren, was benötigt wird, sondern es sollte genauso möglich sein, Teile der Software bereits vor der Fertigstellung des Gesamten zu nutzen. Zusätzlich sollte es möglich sein, die Soft-

ware veränderten Geschäftsbedingungen anzupassen. All das würde das Risiko hoher Investitionsaufwendungen verringern.

Aus Unternehmensperspektive ist die Fragestellung noch eine andere: Wessen Risiko wird durch den Software-Kauf eigentlich reduziert? In vielen Unternehmen gibt es eine IT-Abteilung, die für den Kauf von Software und für Eigenentwicklungen verantwortlich ist. Sie ist jedoch in aller Regel nicht die Abteilung, die die Software nutzt. Gemessen wird die IT-Abteilung oft eher an der pünktlichen Lieferung und der Einhaltung gegebener Budgets als an der Gesamtlösung oder dem realisierten Potenzial. Es gibt für sie also wenig bis gar keinen Anreiz, Software selbst zu entwickeln, denn ein Kauf verlagert das gesamte für die IT-Abteilung relevante Risiko zum externen Software-Anbieter.

Für das Business hingegen sieht die Situation etwas anders aus. So besteht nicht nur die Möglichkeit, dass eine Eigenentwicklung kostengünstiger wäre und dadurch früher den ROI erreicht hätte, es ist genauso möglich und nicht unüblich, dass die gekaufte Standard-Software nicht exakt das bietet, was die Fachabteilung braucht. Das wiederum führt zu verringerter Produktivität und ungenutzten Chancen. Selbstverständlich sollten diese Punkte mit in den Entscheidungsprozess einfließen. Unglücklicherweise wird dieser jedoch normalerweise von der IT-Abteilung durchgeführt und diese ist mehr an der Minimierung ihrer eigenen Risiken interessiert. Entsprechend ist es nicht unwahrscheinlich, dass der Auswahl- und Entscheidungsprozess zwar das Risiko der IT-Abteilung minimiert, jedoch nicht die beste Lösung für das Unternehmen als Ganzes findet.

Ein gutes Beispiel für diesen Konflikt ist die Einführung einer CRM-Lösung bei Australiens größtem Telekommunikationsanbieter. Hier kaufte man ein System von einem der Marktführer, was eindeutig das Risiko der IT-Abteilung minimierte. Wie jedoch eine Zeitung berichtete, meinen die Händler an der Vertriebsfront des Unternehmens, dass die unpraktische Kunden- und Abrechnungsplattform im Zentrum der mehrere Milliarden schweren IT-Transformation sie einiges an Geschäftspotenzial kostet [1]. Für die IT-Abteilung minimiertes Risiko bedeutete in dem Fall nicht die optimale Lösung für das Unternehmen.

Wenn nun also der Fachabteilung mit einer Eigenentwicklung besser geholfen werden kann, ist es dann möglich, der IT-Abteilung das Umsetzungsrisiko abzunehmen? Zunächst einmal haben im letzten Jahrzehnt bedeutende Durchbrüche in den Entwicklungsmethoden dafür gesorgt, dass Software-Entwicklung viel vorhersagbarer geworden ist. Prozesse, die um das agile Wertesystem gebaut wurden, liefern funktionierende Software in kurzen Iterationen. Sie liefern dabei nicht nur inkrementell Funktionalität, sondern bieten sowohl dem Business die Chance, zeitnahes Feedback zu geben, als auch den Entwicklern die Möglichkeit, die Software bei Bedarf anzupassen. Das verringert das Risiko immens, dem Business eine Software zu liefern, die nicht seinen eigentlichen Bedürfnissen entspricht. Die Fähigkeit, iterativ zu entwickeln und auf Veränderungen schnell zu reagieren, wird den Software-Entwicklungsteams mit Techniken und Praktiken wie zum Beispiel Extreme Programming und Continuous Delivery gegeben.

Risiko-Minimierung

Noch interessanter ist die Möglichkeit der Risiko-Minimierung durch Verlagerung der Ziele. Was, wenn das Hauptziel nicht mehr darin besteht, eine große Menge an Function Points in vorgegebener Zeit und mit festgesetztem Budget zu liefern? Was, wenn das Kernziel wäre, die zum jeweiligen Zeitpunkt am dringendsten benötigte Funktionalität so effizient und effektiv wie möglich zu liefern? Auch hier würden agile Praktiken ganz klar helfen, aber wie weltfremd wäre das denn? 1982 eröffnete Tom DeMarco sein einflussreiches Buch zum Thema „Software-Entwicklung“ mit dem Satz: „Was man nicht messen kann, … kann man nicht kontrollieren.“ Interessanterweise schreibt er im Jahr 2009 in einer IEEE-Veröffentlichung [2], dass wir uns in den vergangenen vierzig Jahren mit unserer Unfähigkeit, ein Software-Projekt pünktlich und innerhalb des vorgegebenen Budgets zu beenden, gequält hätten, dass das aber nie das primäre Ziel hätte sein sollen. Viel wichtiger sei Transformation; Software zu kreieren, die die Welt oder Unternehmen verändert oder die verändert, wie ein Unternehmen sein Geschäft betreibt. Das hätte ich nicht besser ausdrücken können.

Ein anderes gängiges Argument gegen Eigen-Entwicklungen ist, dass die Implementierung eine Menge Zeit und Aufwand erfordert. Eine Geschäftsfunktion, die auf wenigen Seiten beschrieben werden kann, erfordert mehrere Hundert Zeilen Code. Um diese zu schreiben und zu testen, braucht ein Entwickler Wochen und es dauert sogar noch länger, um sie in eine Produktionsumgebung auszurollen.

Komplexe Systeme, die nur auf mehreren Hundert Seiten beschrieben werden können, erfordern sogar Jahre, um sie zu entwickeln. Eine offensichtliche Antwort wäre, mit der Ineffizienz zu leben, aber zumindest die Umsetzungszeit zu reduzieren, indem ein größeres Team an die Aufgabe gesetzt wird. Unglücklicherweise ist es inzwischen sehr klar, dass die Geschwindigkeit bei der Software-Entwicklung nicht mit der Menge der Entwickler im Team skaliert. Ab einem bestimmten Punkt übersteigt der wachsende Kommunikations- und Koordinations-Aufwand jede weitere Brainpower.

Genauso, wie der Kauf von Standard-Software das Risiko auf den Anbieter verlagert, verlagert er auch das Problem effizienter Lieferung auf den Anbieter. Der Käufer muss nur noch einen Ausschreibungs- und Beschaffungsprozess durchlaufen und die Installation der Software überwachen. Noch viel besser: Kaufen mehrere Parteien dieselbe Software, kann der Anbieter die Entwicklungskosten über alle Käufer verteilen. Er verteilt damit auch die Auswirkungen von Ineffizienz auf alle Käufer. Auf die Installation selbst kommen wir später zu sprechen.

Aber ist Software-Entwicklung wirklich so langwierig und ineffizient, wie die Befürworter von Kauflösungen behaupten? Was kann getan werden, um sie effizienter zu machen? Fred Brooks erklärte in seinem Paper aus dem Jahre 1987, warum keine einzige Entwicklung in Technologie oder in Management-Techniken eine Produktivitätssteigerung welcher Größenordnung auch immer verspricht [3]. Er fügte jedoch hinzu, dass viele ermutigende Innovationen in Gange seien und dass die disziplinierte, fortlaufende Anstrengung, diese Innovationen zu entwickeln, zu verbreiten und zu nutzen, am Ende tatsächlich zu einer messbaren Verbesserung führen würde. Der Autor glaubt, dass er Recht hatte und dass in den fünfundzwanzig Jahren,

nachdem er dieses Paper geschrieben hat, substanzielle Verbesserungen in der Entwicklerproduktivität erzielt wurden.

Brooks macht eine wichtige Unterscheidung zwischen essenzieller und nebensächlicher Komplexität. Essenzielle Komplexität ist die Komplexität des Geschäftsproblems, das implementiert wird. Es mag so aussehen, als könnte das IT-Team nichts tun, um diese essenzielle Komplexität zu verringern. Das Team kann und sollte Business-Anforderungen nicht ändern. Interessanterweise gibt es in Projekten mit langer Laufzeit (Jahre) jedoch die Tendenz der Fachbereiche, sehr spekulativ zu sein und von vornherein jede nur erdenkliche Funktionalität zu fordern. Dies geschieht aus lauter Angst, auf die Entwicklung neuer Anforderungen ansonsten sehr lange warten zu müssen. Mit einem priorisierten, iterativen Entwicklungsansatz kann das Business hingegen ein Projekt stoppen, wenn alle wirklich benötigten Features fertiggestellt sind. Das reduziert natürlich nicht die essenzielle Komplexität, aber es reduziert die Menge der implementierten Funktionalität meiner Erfahrung nach erheblich.

Zurück zu Brooks´ Unterscheidung. Nebensächliche Komplexität ist diejenige, die durch die bei der Entwicklung verwendeten Tools und Techniken erzeugt wird. Diese Komplexität würde ohne die Implementierung nicht existieren. Wie von Brooks vorhergesagt, hat es keine Reduzierung von nebensächlicher Komplexität im Bereich von Größenordnungen gegeben. Endnutzer-Entwicklung oder automatische Programmierung, wie Brooks sie nannte, ist nicht sehr verbreitet. Die einzige Ausnahme bisher sind Excel-Tabellenkalkulationen. In Zukunft allerdings haben Domain-spezifische Sprachen (DSLs) das Potenzial, hier doch noch einen Durchbruch zu schaffen.

Etwas, das Brooks unterschätzt hat, sind integrierte Entwicklungsumgebungen (IDEs). Die Möglichkeiten, die sie im Hinblick auf Code Navigation und automatisches Code Refactoring bieten, resultieren in beeindruckenden Produktivitätsgewinnen. Automatisches Refactoring in Verbindung mit Test Driven Development ermöglicht es Entwicklern, mit weniger Aufwand hohe Software-Qualität zu erzeugen und dadurch nicht nur die Effizienz, mit der Code geschrieben wird, zu steigern, sondern auch die Entwicklungsgeschwindigkeit bei wachsender Größe der Code-Basis aufrechtzuerhalten.

Entwickler-Workstations sind wesentlich schneller geworden und erlauben damit den Einsatz komplexerer IDEs und anderer Werkzeuge. Verbesserte Compiler zum Beispiel erzeugen besser optimierten Code, ohne dass die Entwickler sich über Details wie Register-Zuteilungen Gedanken machen müssen. Der entscheidende Produktivitätsschub wird aber mit einer schnellen Internetverbindung erreicht. Statt zu experimentieren oder Probleme in Mailing-Listen oder Newsgroups diskutieren zu müssen, können Entwickler jetzt auf Online-Communities zugreifen, die die Antwort auf die meisten Fragen innerhalb von Minuten verfügbar machen, sei es, weil frühere Diskussionen zum Problem durchsuchbar sind, oder, weil Kollegen in Echtzeit via Internet Relay Chat (IRC) antworten.

Die Hardware-Verbesserungen wirken sich nicht nur auf Entwickler-Workstations aus, sondern haben natürlich auch große Auswirkungen auf Produktionsumgebungen. In vielen Fällen können Entwickler jetzt etwas mehr Großzügigkeit beim Ressourcen-Verbrauch zur Laufzeit gegen eine erhöhte Produktivität während der Entwicklung eintauschen. So ist es zum Beispiel möglich, statt viel Aufwand in das Schreiben von Code zur Speicherverwaltung zu stecken, mit dem Einsatz eines automatischen Garbage Collectors die nebensächliche Komplexität, die mit manueller Speicherverwaltung verbunden ist, zu verringern. Es wird sogar argumentiert, dass automatische Speicherverwaltung die Effizienz auf moderner Hardware steigert, da sie lokalere Referenzen und damit verbesserte Cache-Nutzung bietet. Aber selbst wenn die automatische Speicherverwaltung langsamer ist, lohnt sich mit Ausnahme von ganz speziellen Einzelfällen der Einsatz eines Garbage Collectors.

Brooks und viele andere haben in den frühen Neunzigern einen wachsenden Markt für Komponenten-Lösungen vorhergesagt. Diese sollten einen Mittelweg zwischen dem Kauf kompletter Software-Pakete und der Eigenentwicklung darstellen. Das hat sich allerdings im Großen und Ganzen nicht bewahrheitet. Es gibt Ausnahmen, aber allgemein ist die Idee durch etwas noch Besseres ersetzt worden: Open-Source-Frameworks. Viele Entwickler und Unternehmen haben festgestellt, dass es dem eigenen Unternehmen nicht schadet, Quell-Code öffentlich verfügbar zu machen, sondern dass es vielmehr nutzt. Unternehmen profitieren von der frei verfügbaren Entwicklungskapazität anderer Mitstreiter. Ein weiterer Vorteil von direkt von und für Anwendungsentwickler geschriebenen Frameworks ist die Tatsache, dass ihr Funktionsumfang von echten Anforderungen bestimmt ist und nicht von Funktionschecklisten. Die Innovation findet auch auf den öffentlichen Portalen, auf denen die Frameworks bearbeitet werden, statt. Es entstehen immer flexiblere und effektivere Wege, Code zu teilen.

Software-Entwicklung ist nicht unsere Kernkompetenz

In den letzten dreißig bis vierzig Jahren haben Software-Lösungen mehr und mehr Geschäftsbereiche erobert. Es ist jedoch wichtig, sich daran zu erinnern, dass sie ursprünglich als Werkzeuge für operative Effizienz eingesetzt wurden. Vermutlich glauben aus genau diesem Grund viele Unternehmen, dass Software-Entwicklung nicht zu ihren Kernkompetenzen gehört. Einige beziehen sich hier auf Beispiele aus dem frühen zwanzigsten Jahrhundert, als Fabriken noch ihre eigenen Elektrizitätswerke hatten. Die Analogie ist klar: Die Fabriken wechselten sofort zu Strom aus dem Netz, als er verfügbar war, denn selbst Elektrizität herzustellen, gehörte nicht zu ihrem Kerngeschäft. Ihre Kernkompetenz lag woanders, zum Beispiel in der Produktion von Autos. Warum sollte also heute zum Beispiel ein Unternehmen, das sich auf das Verwalten von Versicherungen spezialisiert hat, auch Kompetenzen in der Software-Entwicklung haben? Tatsächlich stellt sich im Zusammenhang mit Cloud Computing sogar die Frage, ob Unternehmen überhaupt IT-Kompetenzen haben oder ob sie nicht stattdessen alles in die Cloud auslagern sollten. Genau wie die Fabriken ihr Strom-Problem in das Netz ausgelagert haben.

Dieser Punkt unterscheidet sich von den beiden oben diskutierten. Die Beurteilung von Risiko und Effizienz ändert sich aufgrund der Vorteile in der Art und Weise, wie Software entwickelt und geliefert wird.

Die Kompetenzfrage wird stark davon beeinflusst, wie sich die Unternehmen ändern. Software-Lösungen sind nicht länger mehr nur Werkzeuge, die es ermöglichen, bestimmte Geschäftsprozesse effizienter zu machen. Software-Lösungen ermöglichen überhaupt völlig neue Wege, Geschäfte abzuwickeln.

Vor etwa fünfundzwanzig Jahren profitierten Zeitungen von der Einführung des elektronischen Desktop Publishing. Das war jedoch nur ein Weg, das Layouting von Seiten effizienter zu machen. Es gab kaum einen Grund, dafür eigene Software zu schreiben. Heutzutage, da sich ein Großteil der Geschäftsgrundlage ins Internet verlagert hat, ist die Software hinter einer Zeitungs-Website strategisch entscheidend für den Geschäftserfolg der Zeitung. So zählt zum Beispiel nicht mehr nur der loyale Zeitungsleser zum Publikum. Genauso wichtig ist der Nutzer, der über eine Suchmaschine zum Artikel gelangt. Bietet die Site nun leichten Zugriff auf verwandte Inhalte, so bleibt der Nutzer länger, sieht mehr Anzeigen und erhöht damit den Umsatz der Zeitung. Die richtigen Inhalte zu finden, ist immer noch eine Angelegenheit von Editoren und Autoren. Sie zum richtigen Zeitpunkt im richtigen Zusammenhang verfügbar zu machen, ist allerdings eine Funktion der Software.

Das Zeitungsbeispiel ist nur eines von vielen. Der Trend, dass Software zum strategischen Erfolgsfaktor für die Geschäftsentwicklung wird, ist in vielen Bereichen und Vertikalen zu finden. Wenn nun also ein Stück Software strategisch wichtig für das Geschäft ist, sollte klar sein, was es bedeutet, dieselbe Standard-Software wie die Konkurrenz zu kaufen: Man hat wenig bis gar keinen Wettbewerbsvorteil. Wie vorhin diskutiert, mag es weniger risikoreich für die IT-Abteilung sein, Standard-Software zu kaufen, aber es macht es für das Unternehmen definitiv schwieriger, einen Vorteil gegenüber den Mitbewerbern zu erlangen. Diese können schließlich aus denselben Standard-Lösungen wählen. Schlimmer noch, es dauert viel länger, auf Marktveränderungen zu reagieren oder eine brillante neue Idee umzusetzen, wenn ein externes Produkt mit externem Lieferanten involviert ist. Die Cycle-Time – also die Zeit von der Idee bis zur Live-Stellung – kann erheblich reduziert werden, wenn die IT-Abteilung gut funktionierende Entwicklungskapazität hat, die auf agile Methoden und Continuous Delivery setzt.

Es scheint, dass, selbst wenn Software-Entwicklung in den meisten Unternehmen keine Kernkompetenz ist, es doch strategisch immer wichtiger wird, sie zu einer solchen zu machen. Das mag schmerzvolle Veränderungen und einen Wandel in der Unternehmenskultur erfordern, aber neue Realitäten zu ignorieren, hat noch nie geholfen.

Quellen

[1] Mitchell Bingemann, Dealers still fuming at 'clumsy' Telstra system. The Australian, 2 June 2009, http://www.theaustralian.com.au/news/dealers-still-fuming-at-clumsy-telstra-system/story-e6frgal6- 1225719947511

[2] Tom DeMarco, Software Engineering: An Idea Whose Time Has Come and Gone? IEEE Software, July/August 2009, http://www.computer.org/portal/web/computingnow/0709/whatsnew/software-r

[3] Frederick Brooks, No Silver Bullet: Essence and Accidents of Software Engineering. Computer, April 1987, http://www.cs.nott.ac.uk/~cah/G51ISS/Documents/NoSilverBullet.html

Übersetzung aus dem Englischen:
Elke Bethke,
ThoughtWorks Deutschland GmbH

Erik Dörnenburg
erik@thoughtworks.com

Erik Dörnenburg ist Software-Entwickler, Berater und Head of Technology bei ThoughtWorks. In Projekten hilft er Kunden bei der Entwicklung von Enterprise Software sowie der Optimierung agiler Praktiken. Häufig mit übermäßig komplexer Software konfrontiert, entwickelte er ein starkes Interesse an einfachen Architekturen und an Möglichkeiten, Software zu visualisieren, um sie dadurch verständlicher zu machen. In den letzten zehn Jahren hat er auf vielen internationalen Konferenzen Vorträge gehalten, Kapitel zu verschiedenen Buchprojekten beigetragen und einige Open-Source-Projekte geleitet.

NetBeans

NetBeans IDE 7.3 ist verfügbar

Die NetBeans Integrated Development Environment (IDE) ist nun in der Version 7.3 allgemein verfügbar und bringt eine bessere Unterstützung von HTML5, JavaScript und CSS, um die Entwicklung von Rich-Web-Applications und mobilen Anwendungen zu vereinfachen.

Netbeans bietet nun eine bessere Unterstützung der Java-Plattform. So enthält der Editor neue Hints und Refactorings. Verbessert wurde auch die Unterstützung der FXML-Layout-Dateien in JavaFX-Projekten. Was HTML5 angeht, so unterstützt NetBeans nun die jüngsten HTML5-Elemente. Bei einem neuen HTML5-Projekt können sich Developer die Arbeit erleichtern und direkt aus einer Liste von beliebten Online-Templates ein Projekt erzeugen. Diese definieren in diesem Fall die Dateien, Bibliotheken und Struktur des Projekts.

Auch die Einbindung vom Framework jQuery wurde auf Vordermann gebracht. Zudem gibt es jetzt für JavaScript im Übrigen auch einen neuen Editor und Debugger, der auf dem Nashorn-Projekt basiert. Auch neue CSS-Regeln werden nun von NetBeans berücksichtigt. Developer können übrigens jetzt in Real Time sehen, wie sich ihre Änderungen im CSS-Style-Fenster auf das Aussehen der Webseite auswirken. Darüber hinaus ermöglicht eine bidirektionale Integration von Google Chrome und dem internen WebKit-basierten Browser eine vereinfachte Live-Code- und Web-Page-Synchronisierung.

Ende März 2013 ist bereits der erste Patch für NetBeans 7.3 veröffentlicht worden. Um diesen einzuspielen, muss zuvor NetBeans 7.3 installiert sein.

Weitere Informationen unter:
https://netbeans.org

Diagramm-Bibliotheken für Android

Falko Krische, buschmais GbR

Die visuelle Aufbereitung geschäftlicher Daten auf mobilen Endgeräten stellt häufig eine große Herausforderung dar. Der Artikel befasst sich mit fünf Bibliotheken für Android, die dieser Aufgabe begegnen.

Internet, Kommunikation, Unterhaltung – das, wofür man bis vor ein paar Jahren noch einen Desktop-PC oder Laptop benötigte, kann man heute in der einen Hand halten und mit der anderen bedienen. E-Mails lesen auf dem Smartphone, Surfen im Internet und Chatten auf dem Tablet gehört heute zum Alltag. Eine Vielzahl von Apps steht dafür bereit.

Anders als im Consumer-Bereich sind professionelle Anwendungen noch rar gesät. Warum überwachen Administratoren nicht mobil ihre Server-Landschaft, prüfen Sales-Manager nicht die Verkaufszahlen der letzten Tage oder sehen Fußballtrainer nicht den aktuellen Trainingsstand ein? Damit diese Anwendungsfälle realisiert werden können, sind Bibliotheken zur visuellen Aufbereitung von Daten erforderlich. Diese Bibliotheken werden nachfolgend für Android unter die Lupe genommen.

Ausgepackt, angeschaut, ausprobiert

Die Untersuchung der Bibliotheken geschieht unter rein pragmatischen Gesichtspunkten. Dabei geht es weitestgehend um Fragen wie: „Ist das API umgänglich?", „Ist die JavaDoc verständlich?", „Kann ich die gezeichneten Diagramme relativ einfach auf meine Bedürfnisse anpassen?" oder „Wo finde ich schnelle Hilfe?"

Als Szenario für die Demo-Applikation dient folgendes Bild: Ein Wartungsingenieur läuft durch eine Halle mit Maschinen und lässt sich die Ausfälle (also Fehler) einer jeden Maschine in einem bestimmten Zeitraum anzeigen. Alle Demo-Applikationen basieren auf der gleichen Datenbasis, um die Vergleichbarkeit zu gewährleisten. Die Demo-Applikationen sind als Download verfügbar (siehe Absatz „Beispiele zum Herunterladen" am Ende des Artikels).

Übersicht

Nachdem die prinzipielle Verwendungsweise der Bibliotheken klar ist, wird die Anpassbarkeit der erzeugten Diagramme untersucht. Dazu werden folgende Punkte betrachtet:

- *Titel*
  Lässt sich der Diagrammtitel anpassen?
- *Achsen*
  Lassen sich die Diagrammachsen anpassen, die Farben ändern, die Achsenskalierungen ändern, die Skalierungsbeschriftungen (Labels) anpassen?
- *Achsentitel*
  Lassen sich diese anpassen?
- *Datenbereich*
  Kann man den Hintergrund ändern? Ist eine Zoom- beziehungsweise Scroll-Funktion bereits in der Bibliothek implementiert? Kann das Gitternetz geändert werden?
- *Datenbeschriftung*
  Kann man einzelne Messpunkte beschriften und wenn ja, kann diese Beschriftung angepasst werden?
- *Datendarstellung*
  Lassen sich Linien, Balken, Punkte etc. in Größe und Aussehen ändern?
- *Legende*
  Lässt sich die Legende anpassen?
- *Implementierungsspezifika*
  Wie lassen sich die Diagramme darstellen und wie werden sie deklariert?

Tabelle 1 zeigt das Ergebnis der Untersuchung.

AChartEngine

AChartEngine ist eine Open-Source-Bibliothek, deren Quelltext bei Google-Code gehostet ist. Die Bibliothek unterstützt alle gängigen Diagramm-Typen wie Linien-, Balken- und Torten-Diagramme (siehe Abbildungen 2 bis 4). Sie steht unter der Apache License 2.0, als minimale Android-Version wird 1.6 gefordert. Die aktuelle Version ist 1.0.0.

Der Start mit dieser Bibliothek geht relativ schnell vonstatten, obwohl keine Tutorials oder kommentierte Beispiele auf der projekteigenen beziehungsweise auf der Google-Code-Seite vorhanden sind. Die JavaDoc ist in den meisten Fällen ausreichend. Glücklicherweise gibt es ein Beispiel-Projekt, dessen Quellen man einsehen kann. Diese hilft die Bibliothek zu verstehen und zu benutzen.

Im Issue-Tracker der Google-Code-Seite gibt es sporadische Anfragen, die von den Entwicklern auch beantwortet werden. Falls man auf Probleme mit der Bibliothek oder Bugs in ihr stoßen sollte, kann man hier auf Unterstützung hoffen.

AndroidPlot

Die aktuelle Version von AndroidPlot ist 0.5.2. Diese Bibliothek ist ebenfalls ein Open-Source-Projekt. Sie bietet nur die vier verschiedenen Diagramm-Typen (Linien-, Balken-, Streuungs- und Schritt-Diagramme). AndroidPlot ist ebenfalls ab Android-Version 1.6 einsetzbar und steht auch unter der Apache License 2.0 (siehe Abbildungen 5 und 6).

Auf der projekteigenen Webseite gibt es einen Quickstart-Guide, der den Einstieg in diese Bibliothek vereinfacht. Auf dieser Seite stehen noch weitere Tutorials, die die grundlegenden Funktionen veranschaulichen. Die JavaDoc ist leider sehr mangelhaft. Viele der Klassen beziehungsweise Methoden sind gar nicht oder nur rudimentär dokumentiert. Es gibt jedoch auf der Bitbucket-Seite des Projekts weitere Beispiele mit Quellcode, um tiefer in die Materie vorzustoßen. Glaubt man der Projekt-Webseite, so wird AndroidPlot schon in mehr als fünfhundert Apps eingesetzt. Es gibt auf der Seite auch ein Forum mit reger Beteiligung.

AFreeChart

AFreeChart ist ebenfalls ein Open-Source-Projekt. Die aktuelle Version ist 0.0.4. Einsetzbar ist es ab Android-Version 2.1. Die Bibliothek wird unter der GNU-Lesser-GPL

	AChartEngine	AndroidPlot	AFreeChart	aiCharts	TeeChart
Titel ändern					
Größe	x	-	x	-	-
Farbe	-	-	-	x	x
Position	-	-	-	-	-
einblenden/ausschalten	x	- (1)	x	x	x
Achsen					
Skalierung ändern	x	-	-	-	-
Skalierung umbenennen (statt 1, 2 ... a, b)	x	x	x	x	x
Farbe anpassen	x	x	-	x	x
beschriften	x	x (1)	x	x	x
Label-Größe anpassen	x	-	x	-	-
Label-Farbe anpassen	x	-	-	x	x
Labels rotieren (90°)	x	-	x	x	-
Achsentitel					
Größe anpassen	x	-	x	-	-
Farbe anpassen	-	-	-	x	-
Position ändern (Abstand von Achse)	-	-	x	-	-
Datenbereich					
Hintergrundfarbe ändern	x (2)	x	-	-	x
Hintergrundbild hinzufügen	-	-	-	-	-
hinein-/herauszoomen	x	-	x (3)	x	x
Bei Start Teilbereich darstellen	x	-	x (3)	x	
Datenbereich verschieben (rechts/links oder hoch/runter)	x	-	x (3)	x	x
Gitternetz hinzufügen/ausblenden	x	x	x	x	x
Gitternetzfarbe anpassen	x	x	x	x	-
Gitternetzstil anpassen	-	-	x	x	-
Datenbeschriftung					
verfügbar	-	x	-	x	x
Größe anpassen	-	-	-	-	-
Farbe anpassen	-	x	-	x	-
Datendarstellung					
Farbe anpassen	x	x	x	x	x
Linienbreite etc. anpassen	x	x	x	x	x
Linienstil etc. anpassen	x	x	x	x	x
Legende					
verfügbar	x	x	x	x	x
Größe anpassen	x	-	x	-	-
Farbe anpassen	-	-	x	-	-
Position ändern	-	-	-	-	x
Implementierungsspezifika					
Vollbild (kompletter Bereich)	x	x	x	x	x
eingebettet (neben anderen View-Elementen)	x	x	x	x	x
Deklaration per XML (in Layout-XMLs)	-	x	-	x	-
Deklaration per Java-Code	x	x	x (4)	x	x

Tabelle 1: Die Anmerkungen dazu stehen im Kasten auf Seite 29

Anmerkungen zur Tabelle:

(1) Wenn nichts angegeben, wird Default-Text angezeigt

(2) Funktionsaufrufe sind da, zeigen aber keine Wirkung

(3) Durch Anpassungen des sichtbaren Bereichs und durch Überschreiben der on-TouchEvent(MotionEvent ev) -Funktionen der View, die das Diagramm zeichnet

(4) Überschreiben der onDraw()-Methode einer View

ausgeliefert. Die Entwickler haben es sich zur Aufgabe gemacht, JFreeChart auf die Android-Plattform zu portieren. Dabei haben sie eine Vielzahl der Funktionen von JFreeChart übernommen, jedoch nicht alle. AFreeChart unterstützt unter anderem Balken-, Linien-, Kuchen- und Streuungs-Diagramme (siehe Abbildungen 7 bis 9). Die Bibliothek wird bei Google-Code gehostet.

Der Einstieg in die Bibliothek von AFreeChart gestaltet sich im Vergleich zu den anderen Bibliotheken relativ schwierig. Es existiert weder ein Quickstart-Guide noch ein Tutorial. Die Beispiel-Applikation, die sich auf der Google-Code-Seite herunterladen lässt, zeigt eine Vielzahl möglicher Diagramme. Jedoch ist der Beispiel-Quellcode, der sich ebenfalls auf dieser Seite einsehen lässt, nicht der, der zu der Applikation gehört. Dessen Umfang ist um einiges geringer. Das verwirrt noch mehr.

Eine Brücke lässt sich bei dieser Bibliothek zu JFreeChart schlagen. Da die meisten Klassen, die von dort portiert wurden, ihre Namen behalten haben, kann man auf der JFreeChart-Seite nach Verwendung und Hilfe suchen. Andererseits ist aber die JavaDoc sehr gut und hilft auch beim Vertiefen der Bibliothek. Leider gibt es offensichtlich keinerlei Feedback der Entwickler auf Probleme der Nutzer. Im Issue-Tracker auf der Google-Code-Seite sind noch Beiträge von Anfang 2011 offen und unkommentiert.

aiCharts

aiCharts der Firma ArtfulBits ist eine Bibliothek, deren Quellcode nicht offen ist und für die man zur Nutzung einen Entwickler-Account und eine einjährige Subscription benötigt. Es besteht jedoch die Möglich-

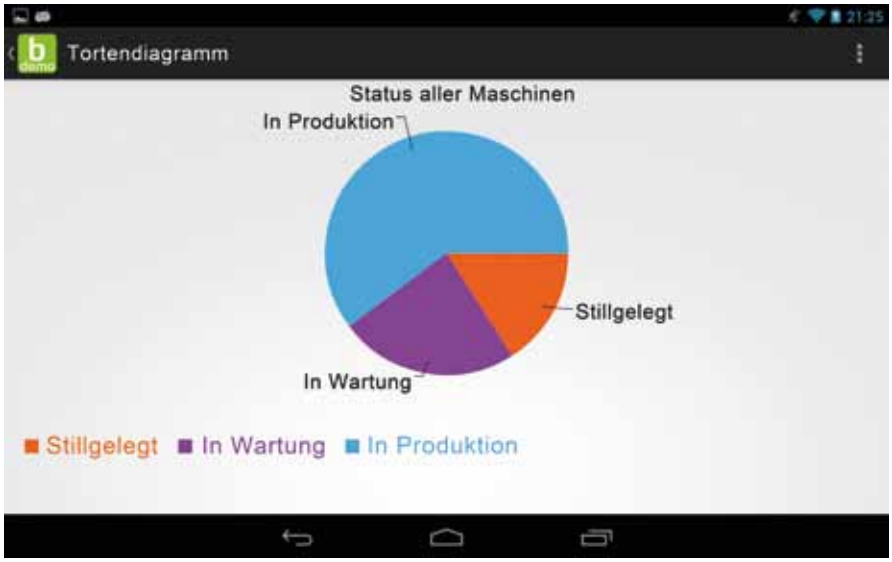


Abbildung 2: AChartEngine – Torten-Diagramm

Abbildung 3: AChartEngine – Linien-Diagramm

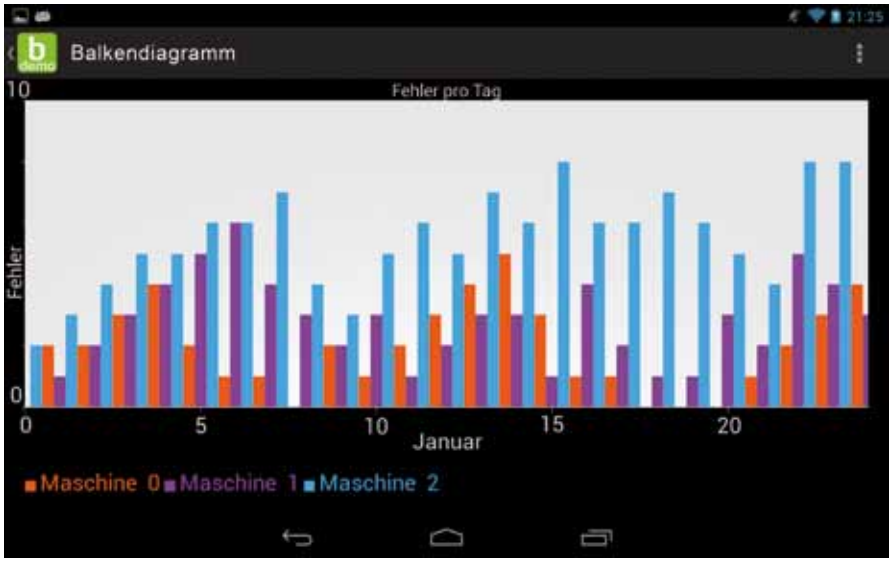


Abbildung 4: AChartEngine – Balken-Diagramm

Abbildung 5: AndroidPlot – Linien-Diagramm

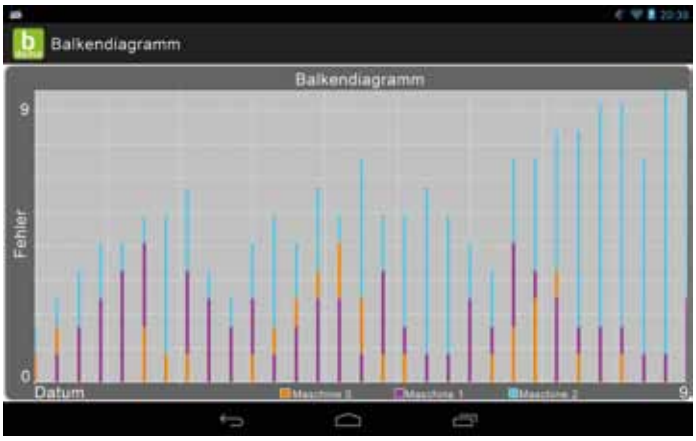


Abbildung 6: AndroidPlot – Balken-Diagramm

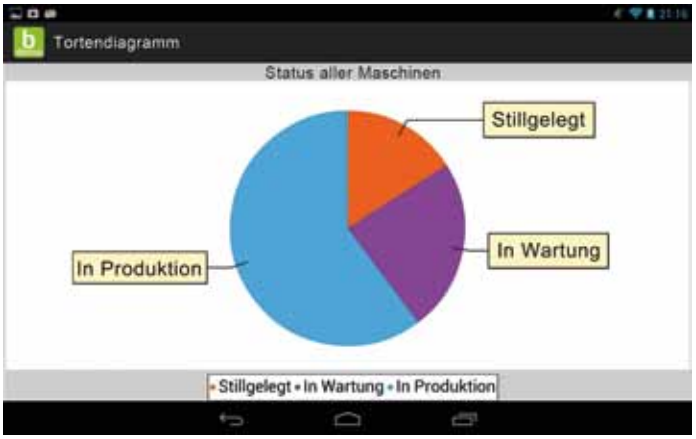


Abbildung 7: AFreeChart – Torten-Diagramm

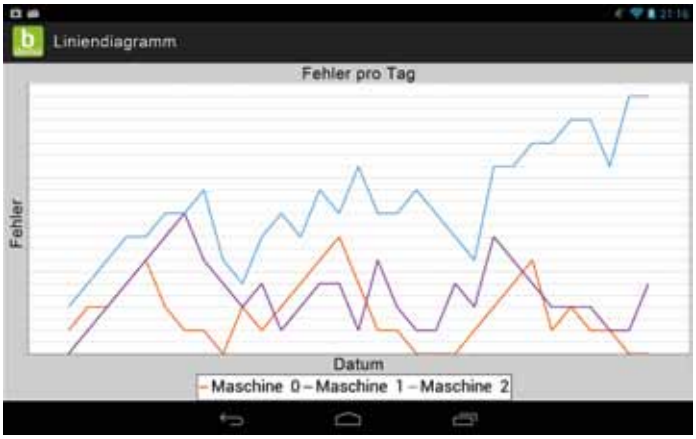


Abbildung 8: AFreeChart – Linien-Diagramm

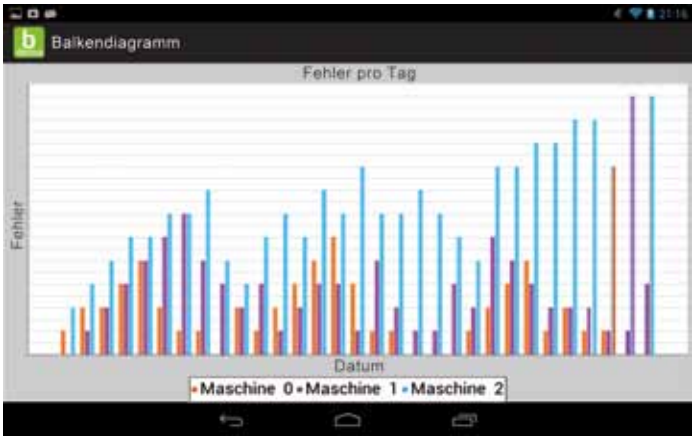


Abbildung 9: AFreeChart – Balken-Diagramm

keit, einen Trial der Versionen 1.7 (stable) beziehungsweise 2.0 (beta) herunterzuladen. Die Bibliothek ist ab Android-Version 1.5 einsetzbar. Die Palette der Bibliothek ist weit gefächert.

Durch den im Download-Paket mitgelieferten Quick-Start-Guide und die ausführlichen Quellcode-Beispiele ist der Einstieg in diese Bibliothek schnell gemacht. Auch das weitere Arbeiten mit der Bibliothek gestaltet sich dadurch relativ einfach. Die JavaDoc ist sehr gut. Es finden sich sogar in wichtigen Klassenbeschreibungen Bilder und Beispiele, wie man etwas implementieren sollte. So kennt man es auch von der JavaDoc des Android-API. ArtfulBits bietet neben dem Support, den man sich mit der Subscription sichert, noch ein Forum, in dem die Entwickler der Bibliothek detailliert auf Fragen der Benutzer eingehen.

TeeChart

TeeChart ist eine Diagramm-Bibliothek der Firma Steema Software. Auch hier benötigt man eine Entwickler-Lizenz und eine jährliche Subscription. Die Bibliothek unterstützt alle gängigen Diagrammtypen wie Linien-, Balken-, Häufigkeiten- und Schnittmengen-Diagramme. Die aktuelle Version ist 2012, TeeChart lässt sich ab Android-Version 2.1 einsetzen.

Das Download-Paket umfasst – wie auch schon bei aiCharts – eine Menge an Tutorials und Beispiel-Anwendungen nebst Quellcode, sodass der Einstieg in die Bibliothek relativ leicht fällt und schnell vonstattengeht. Leider ist die JavaDoc nicht ganz so hilfreich wie gehofft. Die Klassen-Beschreibungen bestehen nur aus Copyright-Informationen und viele Beschreibungen der Funktionen scheinen generiert.

Neben dem Support, den Subscription-User von Steema bekommen, gibt es auf der Webseite auch ein Forum, in dem gestellte Fragen recht schnell beantwortet werden. Als einzige der untersuchten Bib-

liotheken kann TeeChart Diagramme auch in 3D darstellen.

Weitere Bibliotheken

Es gibt natürlich noch weitere Bibliotheken, mit denen sich Diagramme zeichnen lassen. Dazu gehören unter anderem RChart, DroidCharts, Android Chart und Snowdon. Diese wurden jedoch aus der Betrachtung herausgenommen, da deren letzte Aktivität (hinsichtlich Quellcode-Commits oder Foren-/Wiki-Einträgen) mehr als zwei Jahre zurückliegt.

ChartDroid, dessen letzte Aktivität auch im November 2011 zu verzeichnen ist, bietet noch einen weiteren Ansatz der Diagramm-Erstellung. Statt die Bibliothek in die zu erstellende Applikation einzubinden, wird von der Applikation vorausgesetzt, dass eine weitere Applikation – in diesem Fall ChartDroid – auf dem Zielgerät vorhanden ist. ChartDroid wird dann per Intent gerufen und stellt das Diagramm anschließend bereit. Dieser Ansatz scheint jedoch nicht praktikabel genug, um ihn weiter zu verfolgen.

Fazit

Alle Bibliotheken tun genau das, wozu sie entwickelt wurden. Und das nicht mal schlecht. Abgesehen von AndroidPlot decken alle eine weite Palette an Diagramm-Typen ab. Benötigt man einen sehr speziellen Typ, so muss man sich die Beschreibungen der Bibliotheken ansehen, um zu erkennen, ob diese Bibliothek den gewünschten Typ unterstützt. Um zu entscheiden, welche Bibliothek man einsetzen möchte, stellt sich hier nur die Frage, inwieweit man die aufgelisteten Bibliotheken beziehungsweise deren erstellte Diagramme an die eigenen Wünsche anpassen kann.

Ein anderes Entscheidungskriterium könnte die Art und Weise sein, wie die Diagramme zur Anzeige gebracht werden. So bietet zum Beispiel AFreeChart keine Views für die verschiedenen Diagramm-Typen, sondern der Entwickler muss die „onDraw()"-Methode einer Kind-View überschreiben. In dieser wird dann das Diagramm auf den View-Bereich gezeichnet. Diese Art der View-Darstellung passt eigentlich nicht in das Android-Umfeld und ist sicher dem Umstand geschuldet, dass diese Bibliothek von einer Swing-Bibliothek portiert wurde. Andere Bibliotheken (AChartEngine, AndroidPlot, aiCharts und TeeChart) machen es dem Benutzer da schon einfacher: Sie bieten Views an, die in manchen Fällen sogar per Layout-XML deklariert werden können.

Beispiele zum Herunterladen

- achartengine-demo-debug.apk und Quellen: http://www.buschmais.de/wp-content/uploads/2013/02/achartengine-demo-debug.apk und http://www.buschmais.de/wp-content/uploads/2013/02/achartengine-demo-src.zip
- androidplot-demo-debug.apk und Quellen: http://www.buschmais.de/wp-content/uploads/2013/02/androidplot-demo-debug.apk und http://www.buschmais.de/wp-content/uploads/2013/02/androidplot-demo-src.zip
- afreechart-demo-debug.apk und Quellen: http://www.buschmais.de/wp-content/uploads/2013/02/afreechart-demo-debug.apk und http://www.buschmais.de/wp-content/uploads/2013/02/afreechart-demo-src.zip

Links

- AchartEngine: http://code.google.com/p/achartengine und http://www.achartengine.org
- AndroidPlot: http://androidplot.com
- AfreeChart: http://code.google.com/p/afreechart
- aiCharts: http://www.artfulbits.com/products/android/aicharts.aspx
- TeeChart: http://www.steema.com/teechart/mobile

Falko Krische
falko.krische@buschmais.com

Falko Krische ist Berater bei der buschmais GbR. Sein Schwerpunkt liegt auf der Entwicklung von Unternehmensanwendungen im eCommerce- und Enterprise-Bereich. Die Entwicklung von Android-Anwendungen ist sein großes Steckenpferd.

Die erste Java-EE-7-Spezifikationen sind final

Die nächste Version der Java-EE-Spezifikation biegt auf die Zielgrade ein. Die ersten vier Spezifikationen sind bereits Ende März 2013 erfolgreich durch den „Final Approval Ballot" des Java Community Process (JCP) gegangen und damit inhaltlich final. Die Vorreiter sind:

- Java Message Service 2.0 (JSR 343, http://www.jcp.org/en/jsr/detail?id=343)
- Bean Validation 1.1 (JSR 349, http://www.jcp.org/en/jsr/detail?id=349)
- Batch Applications for the Java Platform (JSR 352, http://www.jcp.org/en/jsr/detail?id=352)
- Java API for JSON Processing (JSR 353,http://www.jcp.org/en/jsr/detail?id=353)

Mit Ausnahme des Batch JSR 352 wurden alle Einstimmig befürwortet. Werner Keil, als einziges persönliches Mitglied im Executive Committee (EC), äußerte Bedenken im Hinblick auf die nicht einheitliche Lizensierung der diversen Ergebnisse des JSRs. So finden sich im JavaDoc Hinweise wie „Copyright © 2012. All Rights Reserved", obwohl Technology Compatibility Kit (TCK) und Referenzimplementierung (RI) laut original JSR eigentlich einheitlich Apache 2.0 lizensiert sein sollten. Die Interpretation mag valide sein und formell ist dies sicherlich nicht in Ordnung. Bleibt abzuwarten, ob für die kommenden Versionen hier Abhilfe entsteht.

Die aktuell noch in Entwicklung befindliche Version der Java EE 7 konformen Referenz-Implementierung ist der GlassFish 4.0. Die verfügbaren promoted builds (http://dlc.sun.com.edgesuite.net/glassfish/4.0/promoted/) enthalten die vier Spezifikationen bereits. Jeder Interessierte ist eingeladen, diese bereits heute auszuprobieren und eventuelle Fehler über den GlassFish Bugtracker (http://java.net/jira/browse/GLASSFISH) zu melden.

WebID: Das Web als Social Network

Angelo Veltens, http://datenwissen.de

Mit Linked Data lassen sich Daten im Web verbinden. Weitet man das Konzept auf Personen und deren Interaktion aus, entsteht ein „Social Web of Data". Anstelle Dutzender Accounts bei unzähligen Diensten tritt eine globale Identität, die WebID. Sie dient Angeboten im WWW als Quelle von Profil-Informationen und zur Authentifizierung. Auch als Single Sign-on im Unternehmens-Intranet kann WebID eine Option sein.

Wer heute im Web aktiv ist, muss eine Vielzahl von Nutzer-Accounts bei unterschiedlichen Diensten anlegen und verwalten. Dutzende Profile wollen ausgefüllt und gepflegt werden, verlangen Nutzernamen und Passwörter. Dies alles scheint im Social Web nötig zu sein, um Nutzer zu identifizieren und sich mit anderen Nutzern in Kontakt bringen zu können.

Doch die soziale Interaktion ist in der Regel auf die jeweiligen Dienste begrenzt und lässt sich bestenfalls auf sogenannte Social Networks wie Facebook, Twitter und Google+ ausweiten. Letztere werden auch oft als Authentifizierungsdienst herangezogen. Dies führt zu einer Zentralisierung des eigentlich dezentralen Social Webs zu monolithischen Social-Network-Plattformen. Dienste, die sich zur Authentifizierung auf diese Plattformen verlassen, werden von deren API und ihren Beschränkungen abhängig.

Mit Linked Data und dem WebID-Protokoll ist ein Ausweg aus dem Dilemma in Sicht. Statt unzähliger Accounts erhalten die Nutzer eine Identität im Web, die sie bei Bedarf selbst hosten können und über die sie den genutzten Diensten beliebige Profil-Informationen bereitstellen können. Dadurch wird das Web selbst zu einem dezentralen sozialen Netzwerk.

Rückblick: Daten verlinken

In den letzten Ausgaben wurde Linked Data thematisiert. Das WWW ist bereits heute ein Netz aus miteinander verlinkten Dokumenten. Linked Data weitet dieses Konzept der Links auf Daten aus, indem es nicht nur Dokumente, sondern auch Dinge über URIs identifiziert und miteinander in Beziehung setzt. Dazu dient das Datenmodell RDF, das Daten als „Tripel" abbildet. Ein solches Tripel besteht aus Subjekt, Prädikat und Objekt. Subjekt und Objekt werden über das Prädikat miteinander verlinkt.

Im Social Web bestehen vielerlei Beziehungen: Nutzer knüpfen digitale Freundschaften, veröffentlichen Fotos, Kommentare und viele andere Inhalte. Linked Data ist wie dafür geschaffen, diese Beziehungen abzubilden. Im Grunde genommen erzeugt jeder Klick auf einen „Gefällt mir"-Button ein RDF-Tripel. Dessen „Subjekt" ist der klickende Nutzer, „gefällt" ist das Prädikat und die dem Nutzer gefallende Seite ist das Objekt (siehe Abbildung 1).

Freunde von Freunden

Die Linked-Data-Prinzipien von Tim Berners-Lee [1] verlangen zunächst, dass „Dinge" – nicht nur Dokumente – mittels URI identifiziert werden. Auch Personen können dabei durch einen URI identifiziert werden. Sobald dies geschehen ist, können sie als Subjekt oder Objekt in einem RDF-Tripel verwendet werden. Mein URI lautet etwa „http://me.desone.org/person/aveltens#me" und mit folgendem RDF-Tripel sage ich aus, dass ich Tim Berners-Lee kenne (siehe Listing 1).

In dem Beispiel wird die FOAF-Ontologie [2] verwendet, um die Bekanntschafts-

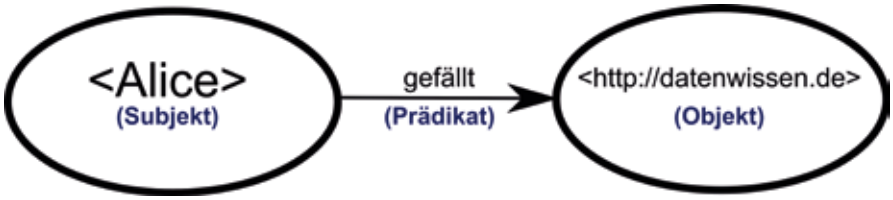


Abbildung 1: „Gefällt mir"-Beziehung als RDF-Graph

```
<http://me.desone.org/person/aveltens#me>
        foaf:knows
                <http://www.w3.org/People/Berners-Lee/card#i>.
```

Listing 1

```
<http://me.desone.org/person/aveltens#me>
 a foaf:Person;
 foaf:familyName "Veltens";
 foaf:givenName "Angelo";
 foaf:homepage <http://datenwissen.de/>;
 foaf:basedNear <http://sws.geonames.org/2945024/>.
```

Listing 2

beziehung auszudrücken. FOAF steht für „Friend of a Friend" und ist eine weit verbreitete Ontologie, die strukturierte Aussagen über Personen und deren Beziehungen trifft. In Listing 2 ist ein Auszug aus meinem FOAF-Profil in der RDF-Syntax „Turtle" zu sehen:

In der ersten Zeile steht der URI des Subjekts, über das Aussagen getroffen werden. Es folgen Prädikat-Objekt-Paare, durch Semikolon getrennt. Aus den Daten geht hervor, dass es sich um eine Person handelt („a foaf:Person"). Auch Vor- und Nachname sind im Profil hinterlegt. Da es sich bei der Homepage um eine Web-Ressource handelt, wird sie über ihren URI verlinkt und nicht wie die Namen als String-Literal hinterlegt. Die große Stärke von Linked Data liegt auch hier in der Verlinkung von beliebigen Ressourcen im Web of Data. „foaf:basedNear" verweist daher auf den URI der Stadt Braunschweig beim Dienst „geonames.org". Folgt man diesem Link, erhält man weitere Informationen zur Stadt, beispielsweise deren Einwohnerzahl.

Natürlich ist man in einem Profil nicht auf FOAF beschränkt, sondern kann alle im Web verfügbaren Ontologien nutzen und bei Bedarf eine eigene entwickeln. Für die Bedürfnisse von Anwendungen im Social Web ist in Ergänzung zu FOAF die SIOC-Ontologie [3] geeignet. Ruft man „http://me.desone.org/person/aveltens#me" im Browser auf, erhält man eine menschenlesbare HTML-Seite. Verlangt man im HTTP-Accept-Header jedoch „application/rdf+xml", antwortet der Server mit maschinenlesbarem RDF/XML. Das RDF-Dokument hat auch einen eigenen URI und lässt sich über „http://me.desone.org/person/aveltens.rdf" direkt aufrufen.

Im Web of Data werden Dinge von Dokumenten unterschieden. Eine Person ist etwas anderes als deren Profilseite im Web. Daher benötigen beide einen eigenen URI. Während ich als Person durch „http://me.desone.org/person/aveltens#me" (Man beachte den Fragment-Identifier „#me", der Teil des URI ist) identifiziert werde, lautet der generische URI meiner Profilseite „http://me.desone.org/person/aveltens" (ohne „#me"). Letzterer identifiziert ein Dokument, der erste eine Person (siehe Abbildung 2). Über „http://me.desone.org/person/aveltens.rdf" und „http://me.desone.org/person/aveltens.html" werden zudem zwei verschiedene Repräsentationen des generischen Dokuments identifiziert (RDF/XML und HTML).

Wie man sieht, ist es leicht, ein Social-Network-Profil mithilfe von Linked Data zu erzeugen und mit anderen Profilen und Web-Ressourcen zu verlinken. Aber das allein reicht noch nicht, um soziale Interaktionen im WWW zu ermöglichen. Mit bewährten, asynchronen Verschlüsselungsmechanismen und dem WebID-Protokoll eignet sich ein FOAF-Profil auch zu Authentisierung im Web.

Das WebID-Protokoll

Authentifizierung im Social Web bedeutet, nachweisen zu können, dass einem Nutzer ein bestimmtes Profil gehört. Bei zentral organisierten Social-Network-Plattformen kann dies durch die Eingabe des entsprechenden Benutzernamens und des zugehörigen Passworts erfolgen. In einem dezentralen Social Web aus verlinkten FOAF-Profilen soll eine Nutzerin nachweisen können, dass das unter einem bestimmten URI abrufbare Profil das ihre ist. Dieser Nachweis soll zudem gegenüber beliebigen Diensten im Web möglich sein.

Das WebID-Protokoll erreicht dies durch geschickte Kombination bewährter Web-Standards. Neben einem persönlichen URI und einem darunter verfügbaren FOAF-Profil benötigt ein Nutzer zusätzlich ein X.509-Zertifikat und den dazu gehörigen privaten Schlüssel. Alle heutigen Browser enthalten eine Zertifikatsverwaltung, in der diese bequem hinterlegt werden können. Auch die Authentifizierung mit

Abbildung 2: Dank Content-Negotiation kann unter dem Profil-URI sowohl ein menschen- (HTML, links) als auch maschinenlesbares Profil (RDF/XML, rechts) ausgeliefert werden

solchen Zertifikaten gegenüber einem Web-Server ist keine neue Erfindung, sondern eine seit Langem funktionierende Technologie. Der neue Weg, den WebID einschlägt, ist die Verknüpfung dieser Zertifikate mit FOAF-Profilen.

Zum einen wird im Zertifikat der URI des Nutzers als „Subject Alternative Name" hinterlegt. Ein Server, der die Authentifizierung durchführt, kann den URI auslesen und die dort hinterlegten Daten abrufen. Um zu beweisen, dass dieses Profil tatsächlich dem Nutzer gehört, fügt dieser neben den gewünschten Profildaten auch seinen öffentlichen Schlüssel dort ein. Die dazu nötigen Prädikate stellt die Cert-Ontologie [4] zur Verfügung (siehe Listing 3).

Der öffentliche Schlüssel ist vom Typ „cert:RSAPublicKey" und wird über das Prädikat „cert:key" mit der entsprechenden Person verlinkt. „cert:exponent" und „cert:modulus" beinhalten die entsprechenden Daten des Schlüssels. Optional kann der Schlüssel auch über ein „rdfs:label" benannt werden. Die Zusammenhänge sind in Abbildung 3 schematisch dargestellt.

Authentisiert sich nun eine Nutzerin über ihr Zertifikat an einem Web-Server, kann dieser das Profil abrufen und die digitale Signatur mithilfe des dort hinterlegten öffentlichen Schlüssels prüfen. Das WebID-Protokoll setzt hier also auf bewährte Public-Key-Authentifizierung, um die Zugehörigkeit des FOAF-Profils zum Nutzer nachzuweisen.

Diese Vorgehensweise ist nicht nur sicher, sondern auch benutzerfreundlich. Ein Klick auf einen Link genügt, um sich gegenüber beliebigen Diensten im Web zu authentisieren. Weder müssen sich Nutzer komplizierte Passwörter merken, noch müssen sie ihren URI eintippen. Denn letzterer ist bereits im Zertifikat enthalten.

Digitale Freundschaften schließen

Nachdem nun Nutzerdaten im FOAF-Profil bereitstehen und die Authentifizierung über das WebID-Protokoll gewährleistet ist, bleibt noch die Frage offen, wie die Kommunikation zwischen WebID-Nutzern ablaufen kann. Wie kann ich beispielsweise jemandem mitteilen, dass ich ihn oder sie als Freund hinzufügen möchte. Natürlich kann man einfach ein entsprechendes RDF-Tripel im eigenen Profil ergänzen, doch möchte man sein Gegenüber vielleicht über diesen Link informieren, ihr die Gelegenheit geben, die Freundschaft zu bestätigen und einen Backlink zu setzen.

Dazu gibt es einen „Pingback"-Mechanismus, ähnlich wie man ihn schon von Blogs kennt. Verlinkt eine Ressource auf eine andere, kann sie die Information darüber über HTTP-POST an einen bestimmten URI senden. Welcher URI das ist, kann die verlinkte Ressource wiederum selbst bekannt geben. Auch dafür gibt es eine passende Ontologie, das Semantic Pingback Vocabulary [5]. Das folgende Listing gibt über das Prädikat „ping:to" an, dass sich der Pingback-Dienst von „my-profile.eu" um eingehende „Pings" zu meinem Profil kümmert (siehe Listing 4).

Logge ich mich mit meiner WebID bei „my-profile.eu" ein, bekomme ich eingehende „Pings" in der Web-Oberfläche angezeigt. Im Artikel „Friending on the Social Web" haben Henry Story, Andrei Sambra und Sebastian Tramp semantische Pingbacks näher beschrieben [6].

Daten-Silos überwinden

Mit Linked Data und WebID kann es gelingen, die abgeschotteten Daten-Silos des Web 2.0 zu öffnen und ein echtes Social Web zu spinnen. Da sowohl Personen als auch Inhalte über ihre URIs global verlinkbar sind, entfällt die Notwendigkeit eines zentralen Social-Network-Dienstes. Wie auch im dokumentenbasierten WWW Links zu beliebigen anderen Seiten möglich sind, können auf Linked Data basierende Social-Web-Anwendungen Links zu Daten anderer Dienste setzen. Abbildung 4 zeigt, wie das Profil der WebID-Nutzerin „Alice" mit verschiedenen Inhalten im Social Web verlinkt ist. In diesem fiktiven Beispiel hostet Alice ihr FOAF-Profil selbst und hat sich mit ihrer WebID bei einem Bilderdienst eingeloggt. Nach einem Klick auf den dortigen „Gefällt mir"-Button wird der in Alices Profil angegebene Pingback-Dienst benachrichtigt, der daraufhin ein RDF-Tripel im FOAF-Profil ergänzt. Alice hat außerdem im Blog eines Freundes einen Kommentar hinterlassen. Auch dieser wurde mit ihrem Profil verlinkt, ebenso mit dem Blogartikel, auf den er sich bezieht.

Eine Vielzahl von kleinen, auf eine Kernaufgabe spezialisierten Diensten kann sich so zu einem Social Web of Data vernetzen. In der Summe bieten diese Dienste ihren Nutzern weit mehr Möglichkeiten als zentralisierte Social-Network-Plattformen und dennoch entfällt die Notwendigkeit, für diese Vielfalt mehrere Accounts zu führen. Die Herausforderung ist jedoch noch, dass viele Dienste WebID unterstützen müssen und die Handhabung ähnlich einfach werden muss wie die von Facebook & Co.

Semantische Zugriffskontrolle

Ein FOAF-Profil muss keinesfalls komplett öffentlich sein. Über semantische Access Control Lists (ACL) ist es möglich, genau zu definieren, wer welche Informationen sehen darf. Diese ACL funktionieren ähnlich, wie man es bereits von Dateisystemen kennt, nur dass nun Personen und Gruppen über URIs identifiziert werden. Angenommen, Alice wird durch „http://me.example.org/person/alice#me" identifiziert und stellt unter diesem URI einige unkritische Informationen über sich öffentlich. Gleichzeitig will sie aber engen Freunden auch Zugriff auf ihr Geburtsdatum und ihre Telefonnummer geben. Dazu hinterlegt sie diese Daten im Dokument „http://me.example.org/person/alice/private" und verlinkt es in ihrem eigentlichen Profil (siehe Listing 5). Man beachte, dass in diesem Beispiel der relative URI „private"

```
<http://me.desone.org/person/aveltens#me>
 cert:key <#publickey>.

<#publickey>
 a cert:RSAPublicKey;
 rdfs:label "Angelos Public Key";
 cert:exponent 65537;
 cert:modulus "ACE0[...]"^^<http://www.w3.org/2001/XMLSchema#hexBinary>.
```

Listing 3

```
<http://me.desone.org/person/aveltens#me>
 ping:to <https://my-profile.eu/pingback.php>.
```

Listing 4

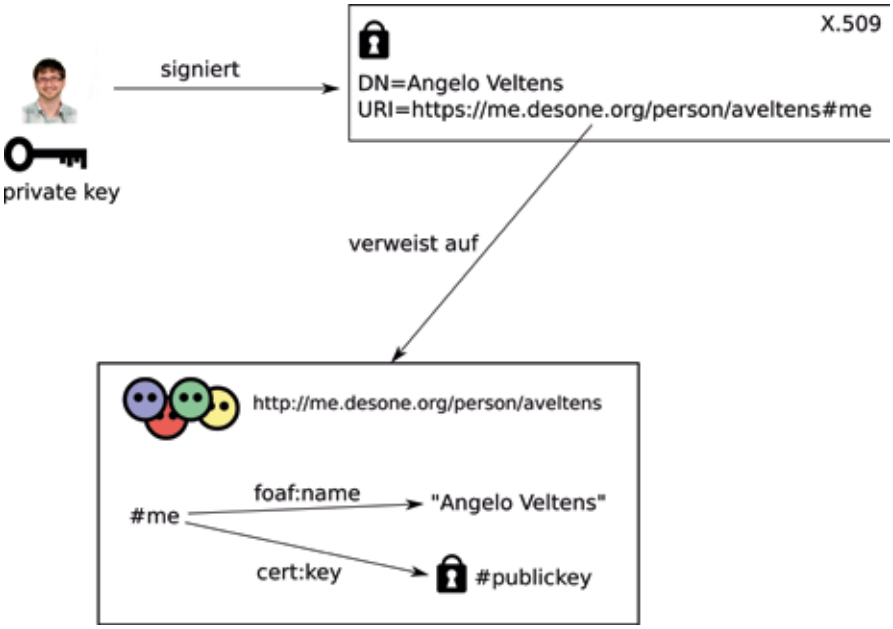


Abbildung 3: Schematische Darstellung der Beziehungen zwischen Nutzer, Zertifikat und FOAF-Profil

genügt, da das Dokument in der Pfadhierarchie direkt unterhalb des öffentlichen Profil-Dokuments liegt. Nun gilt es noch, den Zugriff auf die privaten Informationen zu schützen. Dazu hinterlegt Alice eine Autorisierung in der ACL-Ontologie [7] in ihrem privaten Dokument (siehe Listing 6).

Das Beispiel nutzt nicht mehr die Turtle-Syntax, sondern die ähnliche, aber mächtigere Notation 3. Da die ACL selbst keinen eigenen URI benötigt, ist sie in den ersten fünf Zeilen in eckigen Klammern als sogenannter „blank node" hinterlegt. Das Prädikat „acl:accessTo" teilt mit, welche Ressource durch die ACL geschützt wird. Das leere Klammernpaar „<>" stellt im Grunde wieder einen relativen URI dar, verweist also auf das aktuelle Dokument „http://me.example.org/person/alice/private". Über „acl:mode acl:Read" wird ausgesagt, dass es um den Lesezugriff geht.

„acl:agentClass" schließlich legt fest, wer zugriffsberechtigt ist. Das Prädikat verweist auf einen Typ, dem eine Ressource angehören muss, um Zugriff zu erlangen. Über „acl:agentClass foaf:Person" könnte man beispielsweise allen Personen Zugriff auf eine Ressource gestatten. Alice will ihre Daten jedoch nur engen Freunden zugänglich machen und gruppiert diese daher in einem eigenen Typ „#closeFriend". Ein Web-Server oder eine Web-Anwendung, die das Profil von Alice beherbergt, kann nun bei versuchtem Zugriff auf die private Ressource den Nutzer über das WebID-Protokoll authentifizieren, die ACL auswerten und anschließend den Zugriff verweigern oder gestatten.

Wer WebID ausprobieren möchte, kann sich leicht bei „http://my-profile.eu" ein Profil anlegen. Dabei wird ein Zertifikat im Browser erzeugt und installiert. Das zugehörige FOAF-Profil wird bei „my-profile.eu" gehostet. Anschließend kann man sich mit seiner WebID beispielsweise bei „http://picserv.desone.org" anmelden und dort ein Bild hochladen. Die nächste Ausgabe wird den praktischen Einsatz von WebID intensiver behandeln und zeigen, wie eine Anwendung Nutzer über WebID authentifizieren kann.

Linked Data und WebID im Unternehmensnetz

WebID kann auch für den Einsatz im Unternehmens-Intranet eine Option sein. Zum einen eignet es sich als Single-Sign-on-Verfahren (SSO). Selbst wenn Mitarbeiterdaten über mehrere Systeme verteilt sind, können diese als Grundlage für ein unternehmensweites SSO dienen. Die Systeme müssen dazu lediglich die Daten, wie oben beschrieben, als RDF bereitstellen. Ein zentraler Authentifizierungsdienst mit Kenntnis aller Legitimationsdaten ist jedoch nicht notwendig.

Insbesondere, wenn konsequent auf Linked Data gesetzt wird, können Unternehmen darüber hinaus auch von verbessertem Wissensmanagement profitieren. Mit Linked Data und semantischen Technologien lässt sich Wissen organisieren und besser auffinden. Oft steht ein solches Wissensmanagement jedoch vor dem Problem, dass Informationen nicht in den geeigneten Formaten vorliegen.

Identifiziert man Mitarbeiter über WebID, können RDF-Tripel als Nebenprodukt der täglichen Arbeit entstehen. Angenommen, eine Mitarbeiterin liest im Unternehmens-Wiki die Dokumentation über ein Web-Framework. Sie bekommt dort einen Button „kenne mich aus" zur Verfügung gestellt. Ein Klick darauf verlinkt ihre WebID entsprechend mit dem Web-Framework. Projekte, die gerade planen, das Framework einzusetzen, sehen diese Beziehung und können das Fachwissen der Mitarbeiterin bei Bedarf zu Rate ziehen. Die dazu nötigen Kontaktdaten sind im FOAF-Profil ersichtlich.

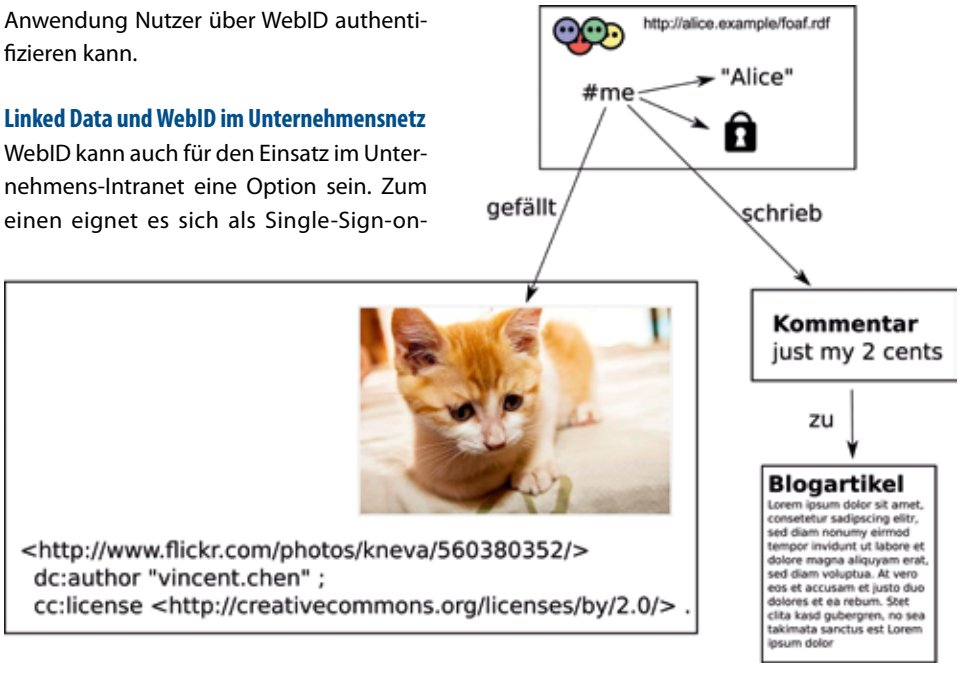


Abbildung 4: Verlinkung des WebID-Profils von Alice mit Inhalten im Social Web

```
# In http://me.example.org/person/alice/private
<http://me.example.org/person/alice#me>
 foaf:birthday "11-30";
 foaf:phone <tel:+49-123-456-789>.

# In http://me.example.org/person/alice
<http://me.example.org/person/alice#me>
 foaf:givenName "Alice";
 rdfs:seeAlso <private>.
```

Listing 5

```
[
 acl:accessTo <>;
 acl:mode acl:Read;
 acl:agentClass <#closeFriend>
].

<#closeFriend> is rdf:type of
 <http://example.net/profile/bob#i>,
 <http://trudy.example/foaf#i>.
```

Listing 6

Fazit

Während Linked Data das Web aus Dokumenten zu einem Web aus Daten erweitert, bringt WebID die soziale Komponente in dieses Web. Nicht nur Dinge werden durch URIs identifiziert, sondern auch Personen. Durch Links lassen sich die URIs dieser Personen mit den Dingen im Web in eine Beziehung setzen. Die sozialen Beziehungen der Nutzer untereinander lassen sich ebenso abbilden wie die Verbindungen zu den Inhalten, die sie im Web erzeugen und teilen. Die Möglichkeiten, die uns heute auf Social-Network-Plattformen geboten werden, lassen sich so auf das gesamte offene Web ausweiten. Es entsteht eine direkte Interaktion zwischen den Profilen der Nutzer, die sie bei Bedarf selbst hosten können, und der Vielzahl an Diensten im Web. Die Notwendigkeit für eine zentrale Plattform entfällt ebenso wie das ständige Anlegen von Accounts bei neuen Diensten.

Referenzen

[1] http://www.w3.org/DesignIssues/LinkedData.html
[2] http://xmlns.com/foaf/spec
[3] http://sioc-project.org
[4] http://www.w3.org/ns/auth/cert#
[5] http://purl.org/net/pingback
[6] http://bblfish.net/tmp/2011/05/09
[7] http://www.w3.org/ns/auth/acl

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.

Vermittlungsinstanzen sind gefragt

Tobias Hartwig, Red Hat GmbH

Cloud-Computing, Software für mobile Endgeräte und Big Data rücken die Middleware in das Blickfeld von Entwicklern. Open-Source-basierte Middleware vereinfacht die Erstellung der dazu benötigten anspruchsvollen Anwendungen.

In großen Unternehmen ist die IT-Infrastruktur historisch gewachsen und meist heterogen. Sie setzt sich aus unterschiedlichen Hardware- und Betriebssystem-Plattformen, verschiedenen Datenbanken und File-Systemen sowie einer Mischung aus Standard-Applikationen und Individual-Software zusammen. Viele Geschäftsprozesse greifen dabei auf mehrere Anwendungen zu, die koordiniert werden müssen. Für Unternehmen wird es auch immer wichtiger, sich schnell und zielgerichtet den dynamischen Marktbedingungen anpassen zu können. Die Herausforderungen können dabei sehr unterschiedlich sein – neue IT-Lösungen, die unternehmenskritische Geschäftsprozesse unterstützen sollen, erfordern einen hohen Grad an Flexibilität, um heutigen und künftigen fachlichen Anforderungen optimal gerecht werden zu können.

Die meisten Unternehmen kämpfen auch heute noch mit einer Vielzahl isolierter Anwendungs- und Informationssilos, die eine Falle für Geschäftsdaten darstellen und komplexe Abstimmungen zwischen den Anwendungen erforderlich machen. Middleware schafft Abhilfe und sorgt für die Interoperabilität von Anwendungen, ermöglicht den Aufruf von Diensten, die in den unterschiedlichsten Programmiersprachen implementiert wurden und liefert Endgeräten jeder Art Verzeichnis- und Vermittlungsdienste zum Zugriff auf entfernte Clients. Damit vereinfacht Middleware den Datenzugriff und senkt die Entwicklungs- und Administrationskosten. Die wichtigsten Vorteile aus Business-Sicht: Middleware bietet vielfältige Möglichkeiten, um die Produktivität der Entwickler zu steigern und den Markteintritt von Produkten und digitalen Dienstleistungen zu beschleunigen. Darüber hinaus kann Middleware entscheidend dazu beitragen, Geschäftsprozesse zu optimieren, zu automatisieren, zu dynamisieren und damit effizienter zu gestalten.

Innovation und Entwickler-Produktivität

Das Einsatzspektrum von Middleware hat sich seit einiger Zeit schon deutlich erweitert, nachdem eine Vielzahl neuer interner und externer Datenquellen sowie Anwendungen auf den unterschiedlichsten mobilen Endgeräten und aus den verschiedenen Cloud-Modellen dazugekommen sind. Als Folge hat die Komplexität massiv zugenommen. Web- und Java-Entwickler benötigen flexible Entwicklungs-Plattformen, um anpassungsfähige Anwendungen zu erstellen. Die Entwickler-Produktivität spielt hierbei auch eine entscheidende Rolle. In diesem Zusammenhang hat der Open-

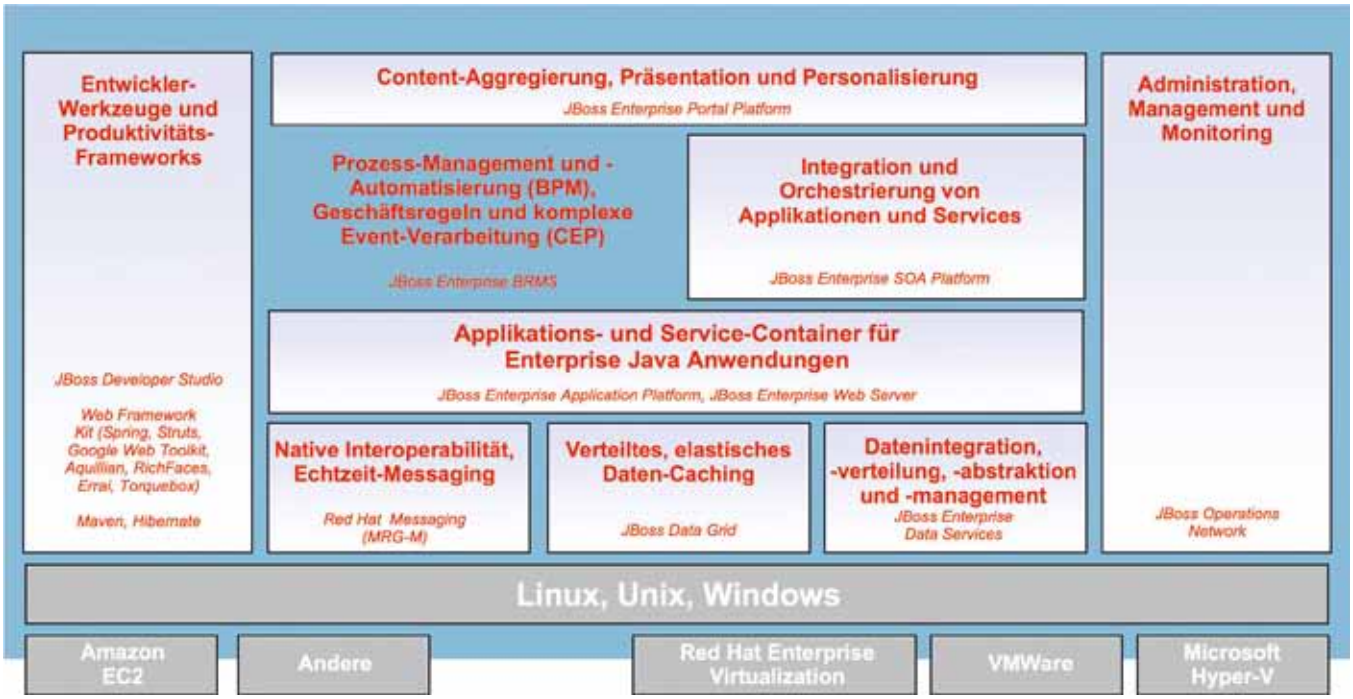


Abbildung 1: Das Produktportfolio von JBoss Enterprise Middleware auf einen Blick

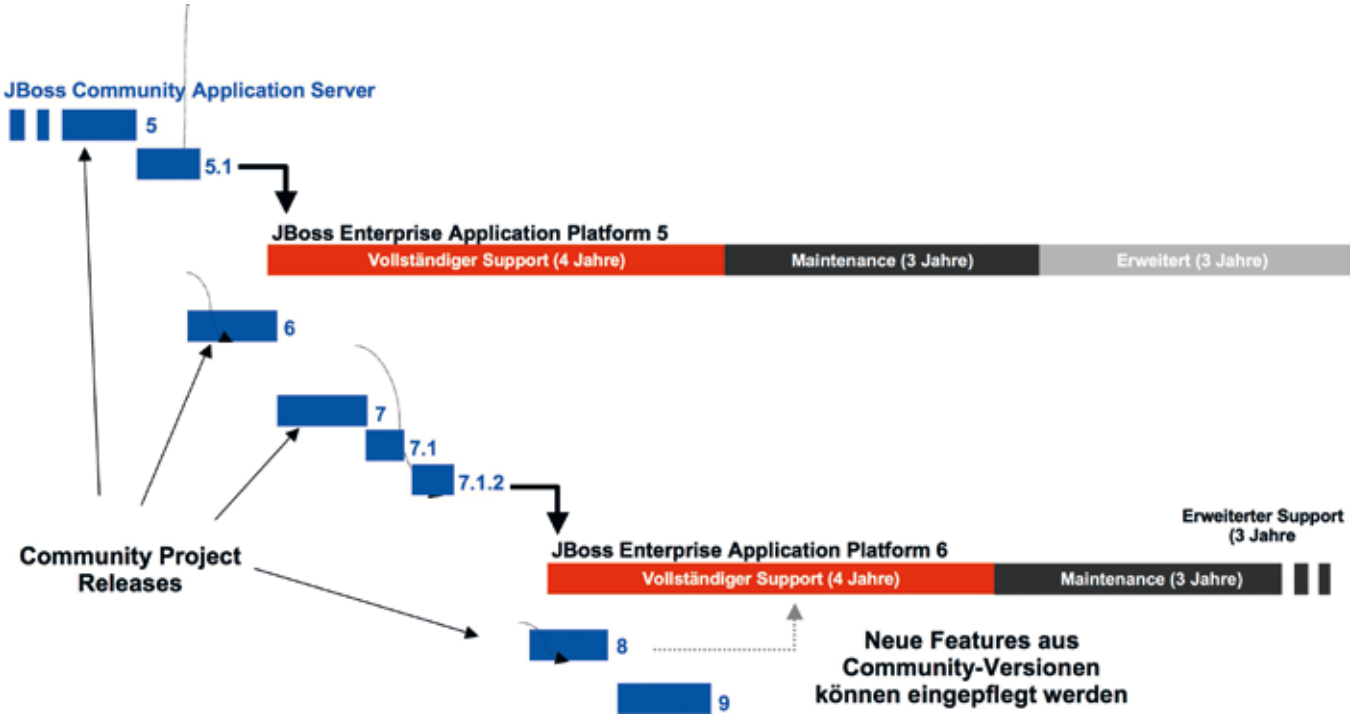


Abbildung 2: Open-Source-Middleware für Unternehmen muss zuverlässig und stabil laufen

Source-Ansatz eine besondere Bedeutung, denn er erweist sich als zentraler Innovationstreiber. Die Entwicklung wird in Community-Projekten durch eine Vielzahl von Beteiligten vorangebracht, die aktuelle Themen und Trends sehr schnell aufgreifen und in Software umsetzen können. Plattformen wie Red Hats JBoss Enterprise Middleware (siehe Abbildung 1) unterstützen aktuelle Technologien sowie offene Standards und ermöglichen eine schnelle und produktive Entwicklung. Open-Source-Plattformen sind damit auf die neuen Herausforderungen durch mobile Anwendungen, Cloud Computing und Big Data gut vorbereitet.

Anforderungen durch mobile Anwendungen

Die Zahl mobiler Anwendungen ist enorm gestiegen. Immer mehr Anwender wollen von unterwegs genauso einfach auf ihre gewohnten Daten und Applikationen zugreifen wie an ihrem Schreibtisch. Entsprechende Apps für das iOS von Apple und für die Android-Welt gibt es bereits und nahezu täglich kommen neue hinzu. Zum einen integrieren immer mehr Unternehmen mobile Anwendungen in komplexe Backend-Applikationen wie CRM- oder ERP-Systeme. Zum anderen fächert sich das Angebot an Mobilgeräten in Zukunft weiter auf. Zu den Smartphones und Tablet-PCs gesellen sich in Zukunft weitere Gerätetypen, darunter beispielsweise auch neue Formen von Sensoren, mit denen sich automatisch laufende Prozesse überwachen lassen. Um die zunehmend heterogenen Mobilgeräte bedienen zu können, werden Entwickler verstärkt auf Open-Source-fähige Middleware und Frameworks zurückgreifen, die offene Standards unterstützen und vom grundlegenden Java-Paradigma „write once, run anywhere“ profitieren (siehe Abbildung 2).

Middleware und die Cloud

Beim Cloud-Computing sind vor allem die Platform-as-a-Service-Lösungen (PaaS) auf Middleware angewiesen. PaaS-Dienstleister stellen in der Cloud eine Plattform für die Entwickler von Web-Anwendungen zur Verfügung. Sie bieten ihren Kunden eine Laufzeitumgebung, die sich einfach administrieren lässt und die den gesamten Lebenszyklus einer Software vom Design über die Entwicklung, den Test bis hin zum Betrieb der Anwendungen in der Cloud umfasst. Parallel dazu geht es darum, nicht nur den reinen Applikationsserver über die Cloud zur Verfügung zu stellen, sondern ihn auch mit anderen Anwendungen zu verknüpfen und mit einem Workflow- und Prozess-Management auszustatten. Gefragt sind flexible und vor allem zukunftssichere Lösungen, um heute schon unterschiedlichste Deployment-Paradigmen unterstützen zu können – physisch, virtuell, in privaten, öffentlichen oder hybriden Clouds, und das, ohne die Anwendungen dafür jeweils anpassen zu müssen.

Unternehmen wollen zudem das Know-how der Entwickler, das sie beim Aufbau einer Applikation für den internen Einsatz von Applikationen benötigen, auch bei Cloud-Anwendungen einsetzen. Dazu bietet sich Middleware an, die im Web die gleichen Funktionen bereitstellt wie im lokalen Netzwerk.

Entwickler müssen hier vor allem die hohen Anforderungen an die Sicherheit der Daten berücksichtigen. Behörden, aber auch Finanzdienstleister und Unternehmen aus dem Gesundheitswesen, die neue Technologielösungen für interne und externe User einsetzen wollen, legen großen Wert auf die Common-Criteria-Zertifizierung. Die JBoss Enterprise Application Platform (EAP) beispielsweise erfüllt den internationalen Common-Criteria-Sicherheitsstandard für Behörden und Unternehmen auf einer der höchsten Stufen, dem Evaluation Assurance Level 4+.

Big Data

Neben mobilen Anwendungen und Cloud-Computing ist Big Data momentan das drit-

Neue Standards für Middleware

Standards vereinfachen die Software-Entwicklung im Bereich Middleware und sorgen dafür, dass die Kommunikation zwischen den Applikationen reibungslos funktioniert. Dabei sind folgende Standards von besonderer Bedeutung:

- **Java EE 6**
  In der Spezifikation der Java Enterprise Edition werden Standards definiert, die die Interoperabilität von Softwarekomponenten und Diensten sicherstellen soll, die primär in der Programmiersprache Java erstellt werden. Dies ermöglicht die Realisierung verteilter, mehrschichtiger Applikationen. Die aktuelle Version Java EE 6 führte einige Neuerungen zur Erhöhung der Entwicklerproduktivität ein. Beispiele sind erweiterte Annotationen, die Unterstützung eines einfachen Objektmodells („POJO") oder eine vereinfachte Paketierung.

- **BPMN 2.0**
  Immer mehr Unternehmen nutzen Business Model and Notation 2, kurz BPMN 2.0. Dahinter steckt eine grafische Spezifikationssprache, mit der sich Geschäftsprozesse in Diagrammen darstellen lassen. Die entstehenden Grafiken haben große Ähnlichkeit mit Flussdiagrammen. Das Ziel von BPMN ist es, einen grafischen Standard zu entwickeln, den alle an einem Geschäftsprozess Beteiligten verstehen.

- **HTML 5**
  Der aktuelle Standard der Hypertext Markup Language vereint Technologien, die den Browser von einem Dokument-Betrachter in eine Anwendungs-Plattform für Webapplikationen verwandeln – und zwar auf allen Plattformen. Das heißt, die Anwendungen sind nicht proprietär, sondern laufen auf allen mobilen Geräten und PCs. HTML 5 ist damit vor allem für mobile Anwendungen von Bedeutung.

- **Advanced Messaging Queuing Protocol (AMQP)**
  Das AMQP ist noch verhältnismäßig neu und wurde erst im Herbst 2011 in der Version 1.0 verabschiedet. Beteiligt an der Entwicklung waren neben IT-Unternehmen wie Red Hat, VMware und Microsoft auch die IT-Abteilungen diverser Finanzinstitute wie etwa JPMorgan, Bank of America oder die Deutsche Börse. Bei AMQP handelt es sich um ein herstellerneutrales, freies und offenes Protokoll, das den Nachrichtenaustausch zwischen unterschiedlichen Systemen ermöglicht. AMQP kommt primär im Finanzsektor zum Einsatz.

te große Thema der IT-Welt. Marktforschern wie IDC zufolge wächst das Datenvolumen in den Unternehmen um jährlich bis zu 60 Prozent. Aufgabe von Big-Data-Anwendungen ist es, Informationen aus einer Vielzahl unterschiedlichster Datenquellen am besten in Echtzeit aufzubereiten und zu analysieren. Solche Anforderungen entstehen beispielsweise im Finanzwesen beim Überwachen von Börsengeschäften, bei Telekommunikationsunternehmen, bei Internet-Suchmaschinen oder in großen Online-Shops, bei denen jeden Tag für Tausende von Kunden riesige Datenmengen anfallen.

Vielfach geht es darum, sowohl große Mengen strukturierter als auch unstrukturierter Information einbeziehen zu müssen. Die Aufgabe der Middleware besteht darin, die gewonnenen Informationen über eine stabile und an offenen Standards ausgerichtete Prozess- und Analyseplattform an anderen Applikationen verfügbar zu machen oder weiterzugeben.

Skalierbarkeit ist gefragt

Auch dann, wenn Unternehmen schnell wachsen und ihre geschäftlichen Aktivitäten kontinuierlich ausweiten, weil beispielsweise neue Märkte erschlossen werden und neue Vertriebskanäle hinzukommen, müssen Middleware technisch skalierbar und gleichzeitig die betriebswirtschaftlichen Kosten transparent kalkulierbar sein. Nicht nur technisch, auch unter Kostenaspekten kann Open-Source-Middleware hier punkten. Unterstützt eine Middleware-Lösung offene Standards, ist es einfach und kostengünstig möglich, bestehende Lösungen zu ergänzen, zusätzliche Bausteine mit vorhandenen zu verbinden und nahezu beliebig miteinander zu kombinieren.

Fazit

Middleware vereinfacht die Arbeit der Programmierer und steigert deren Produktivität, da sie den Aufwand bei der Erstellung von Applikationen reduziert. Deutliche Vorteile zeigen sich beim Testen, bei der Integration in bestehende Umgebungen sowie bei der Skalierung und der Verwaltung. Mit moderner Middleware, die sich an offenen Standards orientiert, können sich Entwickler voll auf die inhaltlichen Anforderungen einer Applikation konzentrieren und dabei vom Vorteil des „write once, run anywhere" profitieren. Middleware beschleunigt damit die Entwicklung anspruchsvoller und komplexer Applikationen vor allem bei Cloud-Computing, Software für mobile Endgeräte und Big Data. Leistungsfähige Middleware unterstützt den Einsatz der Applikationen in physischen und virtuellen Umgebungen, aber auch in privaten, öffentlichen oder hybriden Clouds.

Tobias Hartwig
thartwig@redhat.com

Tobias Hartwig ist Business Unit Manager EMEA für JBoss Enterprise Middleware bei Red Hat.

Anspruchsvolle Web-Anwendungen mit Wicket 6

Jochen Mader, Senacor Technologies AG

Wicket 6 stellt mit seiner starken Ausrichtung an objektorientierten Prinzipien nahezu einen Exoten unter den Web-Frameworks dar. Strikte Trennung von Markup und Logik, transparente Ajax-Integration und ausgezeichnete Unit-Test-Unterstützung sind nur ein paar der Highlights, die in diesem Artikel beleuchtet werden.

Komponentenorientierung ist ein wichtiger Bestandteil moderner Rich-Client-Frameworks wie Eclipse oder NetBeans. Viele der gängigen Web-Frameworks fühlen sich im Gegensatz dazu etwas altbacken an. Oft arbeitet man an den Konzepten der Objektorientierung vorbei und „Copy & Paste" wird plötzlich wieder zum beliebten Entwicklungsmuster. Seit ein paar Jahren hat hier ein Umdenken eingesetzt. Verschiedene Web-Frameworks haben damit begonnen, die Konzepte der Komponentenorientierung zu adaptieren. Die herausragendsten Vertreter dieser Gattung sind nach Ansicht des Autors Wicket und sein geistiger Vater Tapestry. Ihr Ziel ist es, die Entwicklung von Web-Anwendungen wieder näher an den Entwickler zu bringen und Web-Anwendungen zu dem zu machen, was sie eigentlich sind: Objektorientierte Anwendungen, die HTML produzieren.

Quickstart

Nach dem Ausführen von „mvn archetype:generate -DarchetypeGroupId=org.apache.wicket -DarchetypeArtifactId=wicket-archetype-quickstart -DarchetypeVersion=6.6.0 -DgroupId=com.senacor -DartifactId=demo-DarchetypeRepository=https://repository.apache.org/ -DinteractiveMode=false" hat man eine vollständig lauffähige Wicket-Anwendung erzeugt. Besonders faule Menschen können sie sich auch auf der Wicket-Homepage erzeugen lassen. Diese lässt sich entweder direkt per „mvn jetty:run", über die im Test-Ordner hinterlegte „Start"-Klasse oder als Web-Anwendung in der IDE starten.

Ein Quickstart eignet sich beispielsweise als Ausgangspunkt für eigene Anwendungen, als Anhang für einen Bug-Report oder um mal kurz etwas außerhalb der eigentlichen Anwendung ausprobieren zu können. Abbildung 1 zeigt den Inhalt der frisch erzeugten Anwendung. Im Verlauf des Artikels werden verschiedene Beispiele anhand des Quickstarts aufgebaut.

Abbildung 1: Quickstart-Struktur

Konfiguration

Auf XML-Konfigurationsdateien wird vollständig verzichtet. Alle relevanten Konfigurationsoptionen werden in der „WicketApplication" festgelegt. Wie heutzutage üblich regiert auch hier das Prinzip von „Convention over Configuration", Anpassungen sind also nur in Ausnahmefällen notwendig. Falls man doch mal etwas machen muss, kann man in der „init"-Methode via „Getter" auf die verschiedenen Interfaces aus „org.apache.wicket.settings" zugreifen. Außerdem lassen sich hier Page-Mounts definieren, mit denen Pages auf feste URLs gelegt werden können. Ein Blick in „JavaDoc" zu den „mount"-Methoden verrät die Details.

Unmanaged

Wicket ist ein „Unmanaged"-Web-Framework. Statt den Entwickler zur Verwendung eines bestimmten Glue-Layers (Spring, Guice, CDI) zu zwingen, wird ein API bereitgestellt, das eine Integration erleichtert. Natürlich kann man auf vorgefertigte Module zurückgreifen („wicket-spring", „wicket-guice" und „wicket-cdi"). Diese sind Teil der Kern-Distribution, jedoch grundsätzlich von Wicket losgelöst. Somit hat man nun die Qual/Freude der Wahl. Eines wird jedoch vorausgesetzt: Ein Servlet-2.5-Container sollte es schon sein.

Komponenten

Komponenten bilden die Basis einer Vielzahl von UI-Frameworks, angefangen von Eclipse RCP über NetBeans RCP bis hin zu JavaFX. Seit einigen Jahren hat sich hier auch eine kleine Gruppe von Web-Frameworks etabliert, mit ihren bekanntesten Vertretern Tapestry, JSF und eben auch Wicket, die genau diesem Ansatz folgen.

Was Wicket von diesen abhebt, ist die strikte Trennung von Markup und Logik. Während man bei beinahe allen Konkurrenten die Möglichkeit hat, verschiedene Formen von Skriptsprachen (JUEL, Groovy etc.) direkt im Template zu verwenden, sieht Wicket diese Funktion nicht vor. Im Gegenteil: Man bemüht sich, diese Arbeitsweise komplett zu verhindern, indem ein Template ein simples XHTML-Dokument mit erweitertem Namespace ist. Mehr dazu später.

Egal, ob eine vollständige Page oder ein Label – alle Wicket-Komponenten unterliegen einer festgelegten Struktur. Abbildung 2 zeigt den entsprechenden Aufbau. Hier gibt es die eigentliche Komponentenklas-

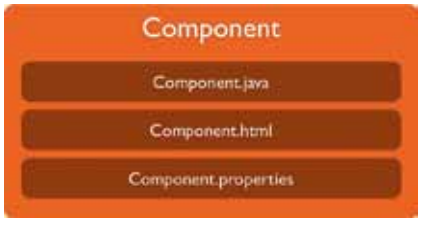


Abbildung 2: Komponente

```
public class HomePage extends WebPage {
        public HomePage(final PageParameters parameters) {
        ...
        final IModel<String> input1 = Model.of("default1");
        final IModel<String> input2 = Model.of("default2");
        Form testForm = new Form("testForm") {
                @Override
                protected void onSubmit() {
                        System.out.println("Input1 "+input1.getObject());
                        System.out.println("Input2 "+input2.getObject());
                }
        };
        TextField<String> inputField1 = new TextField<String>("inputField1", input1);
        TextField<String> inputField2 = new TextField<String>("inputField2", input2);

        add(testForm
                .add(inputField1)
                .add(inputField2)
        );
        }
}
```

Listing 1: HomePage.java

```
...
        <form wicket:id="testForm">
                <input type="text" wicket:id="inputField1"/>
                <inputtype="text" wicket:id="inputField2"/>
                <input type="submit" value="submit"/>
        </form>
</div>
<div id="ft">
</div>
```

Listing 2: HomePage.html

se „Component.java", neben ihr liegen im Klassenpfad die zugehörige Template-Datei „Component.html" und ihre Properties „Component.properties". Die Vorteile dieses Konstrukts liegen auf der Hand: Alle Bestandteile einer Komponente befinden sich an der gleichen Stelle im Klassenpfad und sind auf einen Blick ersichtlich.

Komponenten-Baum und DOM-Tree

Der eingangs erwähnte Quickstart enthält eine simple Startseite in „HomePage.java", die im Folgenden um ein Formular erweitert wird. Hierzu ergänzt man die bestehende Klasse um den Code in Listing 1. Gleichzeitig ist auch das zugehörige Markup-Template in „HomePahe.html" zu ergänzen (siehe Listing 2).

Was sofort auffällt, ist die entstehende Baum-Struktur. Im Java-Code werden der Page eine Form-Komponente samt ihrer zwei Eingabefelder hinzugefügt und im DOM-Tree entsprechend ein „form"- und zwei von ihm eingeschlossene „input"-Tags. Somit sind der Wicket-Komponenten-Baum und der DOM-Tree äquivalent. Die Beziehung zwischen Java und HTML wird dabei durch das XML-Attribut „wicket:id" hergestellt.

Dieses wird durch die Verwendung des entsprechenden Namespace im HTML-Tag bereitgestellt: „<html xmlns:wicket="http://wicket.apache.org">". Daneben gibt es eine Vielzahl weiterer Tags und Attribute, die den Rahmen dieses Artikels sprengen würden. Viel wichtiger ist es zu verstehen, was hierdurch erreicht wird. Wicket-Templates beschreiben nur, was der Java-Code mit ihnen tun soll.

IModel

Auch die schönste Komponente muss irgendwann Daten mit der Welt austauschen. Wicket setzt auf ein simples, aber effektives Konzept, um diese vom Service-Layer zu entkoppeln. Schlüssel dazu ist das „IModel"-Interface (siehe Listing 3).

Jede Komponente hat im Normalfall einen Konstruktor der Form „public Label(final String id, IModel<?> model". Damit wird der Komponente beschrieben, wie sie an ihre Daten kommt und Eingaben zurückgibt. Ein Label wird „IModel.getObject()" erst bei seiner Darstellung aufrufen. Gleichzeitig werden Eingaben via „IModel.setObject(Object)" entgegengenommen. Dieses simple Interface hat gleich mehrere Vorteile.

Der wichtigste ist, dass das Laden von Daten entspannt erfolgt. Daten werden nur dann abgerufen, wenn sie tatsächlich notwendig sind, und der entsprechende Aufruf geschieht zum Zeitpunkt der Verwendung. Der zweite Vorteil liegt in der Möglichkeit, Modelle zu verketten. Listing 4 zeigt dieses Konzept. Das „userModel" greift auf einen Service zu, um eine bestimmte User-Entität zu laden. Das „passwordField" verwendet ein „PropertyModel", das eine „PropertyExpression" benutzt, um selbige am verketteten User-Model zu finden. Das „getObject" liefert hier den String aus der „password"-Property des Users. Ein „setObject", das durch eine Eingabe im „TextField" ausgelöst wird, schreibt in eben diese. Der wichtigste Aspekt dieses Konstrukts ist aber die vollständige Trennung der Beschaffung von Daten und ihrer Verwendung. Eine Komponente weiß nur, was sie mit einer Entität machen soll, jedoch nicht, woher sie kommt.

Der letzte Bestandteil des Komponentisierungs-Konzepts sind die sogenannten „Behaviors". Diese erlauben es, eine Komponente dynamisch um zusätzliche Funktionen zu erweitern. So können AJAX-Funktionalitäten, Event-Verhalten oder das Markup ergänzt werden. Am besten lässt sich das mit einem Beispiel erklären.

Das beste Web-Framework ist heutzutage nur so viel wert wie seine AJAX-Integration. Im Falle von Wicket ist diese vollkommen transparent. In den allermeisten Fällen wird man sich mit den JavaScript-Details der AJAX-Interaktion nicht auseinandersetzen müssen. Es wird zum natürlichen Teil der Entwicklung.

Wer beispielsweise seine Komponente um die Möglichkeit eines automatischen Submit nach einer Eingabe erweitern will,

```
public interface IModel<T> extends IDetachable
{
        T getObject();
        void setObject(final T object);
}
```

Listing 3: IModel

muss nichts weiter tun, als das entsprechende Behavior hinzuzufügen. In Listing 5 wird für diesen Zweck das von Wicket mitgelieferte „AjaxFormSubmitBehavior" verwendet.

Einzig das JavaScript-Event, auf das die Komponente reagieren soll, wird dem Behavior übergeben. Schon kann auf der Konsole beobachtet werden, wie „Form.onSubmit()" aus Listing 1 ausgeführt wird, sobald man Text im entsprechenden Eingabefeld eingibt. Das Ganze erfolgt ohne Neuladen der Seite. Dank der Wicket-AJAX-Bibliothek, die vollständig auf JQuery aufgebaut ist, wird nur der Inhalt des Formulars an den Server gesendet.

Natürlich hat Wicket noch mehr zu bieten. So lassen sich Komponenten mithilfe des „AjaxSelfUpdatingTimerBehavior" zyklisch neu rendern oder via „AjaxEventBehavior" um eine serverseitige Reaktion auf beliebige JavaScript-Events erweitern. Das funktioniert alles ohne eine Zeile JavaScript. In Verbindung mit Events wird es aber erst richtig spannend.

Events

Komponenten existieren nicht um ihrer selbst willen. An bestimmten Punkten wird man mit dem Rest des Komponentenbaums kommunizieren müssen. In früheren Wicket-Versionen (<=1.4) war dies per Default nur via Callbacks möglich. Je nach Tiefe des Komponentenbaums konnte dies zu extrem komplexen Konstrukten führen, da man gezwungen war, ein Callback vom Ursprung bis zum Verwender durchzureichen. Bald fingen Entwickler an, nach Alternativen zu suchen.

Mit Wicket 1.5 wurde dann endlich ein moderner EventBus eingeführt. Jede Komponente implementiert seither die Schnittstellen „IEventSource" und „IEventSink" und kann Nachrichten senden und empfangen. Die so erzielte lose Kopplung erleichtert sowohl die Verwendung als auch die Wiederverwendung von Komponenten. Gleichzeitig wird der benötigte Boiler-Plate-Code auf ein erträgliches Maß reduziert.

Ein gutes Beispiel für diesen Event-Mechanismus ist „Ajax Default Event". Nehmen wir das Beispiel von vorhin und fügen im „HomePage"-Konstruktor den Code aus Listing 6 hinzu. Das Markup wird um die Zeile in Listing 7 ergänzt.

Bei einer Eingabe in „inputField1" löst unser Behavior einen Submit aus und überträgt die Daten in einem Ajax-Request zum Server. Wicket erzeugt nun ein Objekt vom Typ „AjaxRequestTarget", das als Event an alle Komponenten im Baum der aktuellen Seite geschickt wird. Auch das neu hinzugefügte Label wird via „onEvent" informiert und entscheidet sich, an diesem Request teilzunehmen. Es fügt sich selbst durch den Aufruf von „add" hinzu, wird als Teil der AJAX-Antwort gerendert und dann an den Browser zurückgeschickt.

```
IModel<User> userModel = new LoadableDetachableModel<User>() {
        @Override
        protected User load() {
                return userService.loadUserById(1);
                }
        };
TextField<String> passwordField = new TextField<String>("passwordField", new PropertyModel<
String>(userModel,"password"));
```

Listing 4: IModel-Chaining

```
add(testForm
        add(inputField1.add(new AjaxFormSubmitBehavior("change") {
}))
        add(inputField2)
);
```

Listing 5: AJAX-Self Submit

```
add(new Label(„input1Label", input1) {
        @Override
        public void onEvent(IEvent<?> event) {
                if(event.getPayload() instanceof AjaxRequestTarget) {
                        ((AjaxRequestTarget)event.getPayload()).add(this);
                }
        }
});
```

Listing 6: Update mit Default Event

```
<span wicket:id="input1Label">content</span>
```

Listing 7: Update mit Default Event Markup

```
public class TestHomePage
{
        private WicketTester tester;

        @Before
        public void setUp()
        {
                tester = new WicketTester(new WicketApplication());
        }

        @Test
        public void homepageRendersSuccessfully()
        {
                tester.startPage(HomePage.class);
                tester.assertRenderedPage(HomePage.class);
        }
}
```

Listing 8: Wicket-extensions

Unit-Tests

Es gibt wohl kaum einen Bereich des TDD, der so lange vernachlässigt wurde wie das Testen der UI. Diese sind sehr schwer zu

```
@Test
public void checkContentAfterSubmit() {
  tester.startPage(HomePage.class);
  FormTester formTester = tester.newFormTester("testForm");
  formTester.setValue("inputField1","test");
  formTester.submit();
  assertEquals("test", formTester.getTextComponentValue("inputField1"));
}
```

Listing 9: Formulartest

```
<dependency>
  <groupId>org.apache.wicket</groupId>
  <artifactId>wicket-extensions</artifactId>
  <version>${wicket.version}</version>
</dependency>
```

Listing 10: Wicket-Extensions

testen und in vielen Fällen musste man sich bisher der Hilfe von Tools wie Selenium bedienen, um überhaupt einen gewissen Grad automatisierter Tests bereitstellen zu können. Natürlich ist Selenium ein großartiges Tool, für Unit-Tests ist es aber denkbar ungeeignet. Wicket nahm sich dieses Problems an, indem es das einzige Hinderniss zu echten Unit-Tests aus dem Weg räumte: den Servlet-Container.

Beim Blick in die Wicket-Quellen fällt auf, dass man mittlerweile so gut wie keine direkten Abhängigkeiten zum Servlet-API hat. Da Wicket auch die komplette Request-Verarbeitung über einen Filter an Wicket-eigene Klassen delegiert, gibt es hier eine Sollbruchstelle. Es war möglich, Komponenten mit einigen kleinen Tricks komplett außerhalb eines Containers zu verwenden. Die notwendigen Tricks sind im „WicketTester" zusammengefasst, der das Rückgrat der Test-Unterstützung bildet. Natürlich gibt es auch für dessen Verwendung ein entsprechendes Beispiel im Quickstart. In „TestHomePage.java" (siehe Listing 8) findet sich ein Beispiel für den grundlegendsten aller Komponenten-Tests, den Render-Test.

Der „homepagerenderSuccessfully"-Test stellt sicher, dass unsere „HomePage" fehlerfrei gerendert wird. Dafür wird sie im „WicketTester" gestartet und am Ende das gerenderte Ergebnis überprüft. Dies hört sich simpel an, verhindert aber bereits viele zeitintensive Compile-Deploy-Browser-Roundtrips. Darüber hinaus steckt noch viel mehr drin, als dieses einfache Beispiel hergibt. Deshalb erweitern wir es um einen eigenen Unit-Test für unser Formular von zuvor (siehe Listing 9).

Der „WicketTester" kapselt verschiedene Helper-Funktionen im sogenannten „FormTester". Nachdem man diesen erzeugt hat, können Eingabefelder befüllt oder das Formular abgeschickt werden. Mit dem passenden „assertEquals" überprüfen wir dann, ob der eingegebene Wert auch im Model angekommen ist. Mittlerweile ist der Autor dazu übergegangen, seine Komponenten fast vollständig ohne den Einsatz des Browsers zu entwickeln. Die Zeitersparnis ist beträchtlich.

Basis-Komponenten

Wicket bietet einen sehr umfangreichen Satz an Basis-Komponenten, von Formularen über Labels bis hin zu Buttons. Durch das Einbinden der „wicket-extensions"-Dependency in der „pom.xml" (siehe Listing 10) erhält man Zugriff auf weitere, deutlich komplexere Komponenten wie fortgeschrittene Tabellen mit Paging-Unterstützung („DefaultDataTable") oder Wizards.

Darüber hinaus gibt es eine ganze Reihe spezialisierter Komponenten, die einen automatischen „NoScript"-Fallback liefern. Dies sind notwendig, da es tatsächlich User gibt, die JavaScript deaktivieren oder Geräte ohne selbiges nutzen. Es soll auch vorkommen, dass Entwickler zur Unterstützung von IE6 gezwungen sind. Für alle diese Fälle gibt es Komponenten wie „AjaxFallbackButton" oder „AjaxFallbackDefaultDataTable". Diese degenerieren im „NoScript"-Fall automatisch zu entsprechenden Alternativen. Somit lassen sich auch Anforderungen wie die Zugänglichkeit der Anwendung leicht umsetzen.

Fazit

Das Unternehmen des Autors setzt Wicket in einer Vielzahl verschiedener Projekte ein, von Intranet-Anwendungen bis hin zum vollständigen Onlinebanking. Seine starke Ausrichtung an objektorientierten Prinzipien erleichtert vor allem erfahrenen Java-Entwicklern den Einstieg.

Erwähnt seien auch die kurzen Reaktionszeiten, wenn es um Issues/Exploits geht. Gepaart mit einer starken Community und dem Support älterer Versionen erledigen sich auch die Bedenken vieler Architekten hinsichtlich seiner Zukunftssicherheit. Zudem gibt es noch verschiedene Aspekte der Anwendungsentwicklung mit Wicket wie die interessanten Themen „Leicht lesbare Stacktraces", „Markup-Vererbung" oder „Verwaltung von Ressourcen wie CSS und JavaScript" (siehe auch http://code.google.com/p/wicket-guide).

Jochen Mader
jochen.mader@senacor.com
twitter:@codepitbull

Jochen Mader ist als Chief Developer bei der Senacor Technologies AG tätig, wo er sich seit mehreren Jahren mit der Entwicklung und Optimierung von anspruchsvollen Web-Anwendungen auf der JVM beschäftigt.

„Java ist Grundlage für meine Software-Projekte …“

Usergroups bieten vielfältige Möglichkeiten zum Erfahrungsaustausch und zur Wissensvermittlung unter den Java-Entwicklern. Wolfgang Taschner, Chefredakteur von Java aktuell, sprach darüber mit Dirk Dittert von der Java User Group Erlangen-Nürnberg.

Wie ist die JUG Nürnberg organisiert?
Dittert: Wir sind Mitglied im Interessenverbund der Java User Groups e.V. (iJUG). Intern sind wir hingegen eher lose organisiert. Es ist also keine Vereinsmitgliedschaft notwendig, um an unseren Treffen teilzunehmen.

Was zeichnet die JUG Erlangen-Nürnberg aus?
Dittert: Unsere Speaker und unsere Teilnehmer bei den Treffen motivieren mich stark, mich dort einzubringen. Der Erfahrungsaustausch unter den Mitgliedern steht im Mittelpunkt. Das Java-Ökosystem ist so vielfältig, dass eine einzelne Person unmöglich den Überblick über all die spannenden Weiterentwicklungen behalten kann. Ich möchte durch unsere Treffen dazu beitragen, dieses Wissen in der Region zu verbreiten und so die Community zu stärken.

Wie viele Veranstaltungen gibt es pro Jahr?
Dittert: Wir treffen uns eimal im Monat.

Was bedeutet Java für dich?
Dittert: Java ist die Grundlage für meine Software-Projekte und darüner hinaus die Plattform mit dem besten Tool-Support. Diese Programmiersprache begleitet mich seit vielen Jahren ständig.

Was hast du bei der Übernahme von Sun durch Oracle empfunden?
Dittert: Unter der Führung von Sun hat sich ein aktives Ökosystem um die Sprache Java und den zugehörigen Technologien herausgebildet. Anfangs hatte ich Bedenken, ob unter Oracle die Offenheit der Java-Plattform leiden würde. Diese wurden jedoch durch die kontinuierliche Weiterentwicklung und der klaren Kommunikation – wie der Release-Pläne des JDK – vollständig ausgeräumt. Ich hatte aber auch von Anfang an die Hoffnung, dass die Übernahme frischen Wind in die Entwicklung des JDK bringen würde. Es ist nun an Oracle, dies durch regelmäßige Releases zu beweisen.

Wie sollte sich Java weiterentwickeln?
Dittert: Java wird oft vorgeworfen, dass die Syntax sehr ausschweifend ist und die eigentlichen Programm-Strukturen versteckt. Hier sollte Java von anderen Sprachen wie beispielsweise Scala lernen und uns Entwicklern mehr Schreibarbeit abnehmen. Die für Java 8 geplanten Closures sind ein erster Schritt in die Richtige Richtung. Aber auch die Typinferenz bei der Variablen-Deklaration würde Java gut bekommen.

Modularisierung ist ein wichtiges Thema bei großen Systemen. Die momentan verfügbaren Sichtbarkeiten gehen mir hier nicht weit genug. Ich halte es hier aber für wesentlich wichtiger, sich endlich auf eine allgemein akzeptierte Technologie zur Modularisierung auf der Java-Plattform zu einigen, als das JDK selbst zu modularisieren. Und wenn ich schon am wünschen bin: Es wäre schön, endlich mal das komplette JDK in einem einheitlichem Stil neu zu formatieren.

Wie sollte Oracle deiner Meinung nach mit Java umgehen?
Dittert: Oracle sollte die Entwicklungen anderer Sprachen genau beobachten und innovative Konzepte, die zu Java passen, dann integrieren. Mir ist eine nachhaltige Weiterentwicklung wesentlich wichtiger, als sofort auf alle Hype-Themen aufzuspringen oder gar die Sprache Java aufgrund von kurzfristigen Überlegungen zu verbiegen.

Bei den Technologien sehe ich Oracle eher in einer integrierenden Position. Viele innovative Konzepte kommen aus der Community. Hier wünsche ich mir, dass Oracle diese Bemühungen aktiv unterstützt und zügig die nötigen Grundlangen in der Plattform schafft. Ein Beispiel hierfür ist die Unterstützung von „invokedynamic“, das gerade bei den dynamischen Programmiersprachen Performance-Vorteile bringt.

Welchen Stellenwert besitzt die Java-Community für dich?
Dittert: Eine aktive Community ist für mich bei jeder Technologie-Entscheidung ein wichtiges Kriterium. Nur so kann man sicherstellen, dass die Anforderungen aus der Praxis wirklich zur Weiterentwicklung der verwendeten Technologien führen und man nicht auf Luftschlösser setzt. Dieser Aspekt ist jedoch nicht nur für die Community wichtig, sondern bringt auch für Oracle Vorteile: Software wie Spring, Scala, IntelliJ IDEA, Yourkit oder JRebel erlauben effizientes und komfortables Arbeiten – wie kaum auf einer anderen Plattform.

Wie sollte sich die Community gegenüber Oracle verhalten?
Dittert: Die Community sehe ich als aktives Sprachrohr gegenüber Oracle. Es ist unsere Aufgabe, unsere Wünsche und Anforderungen aus dem täglichen Arbeitsalltag in gebündelter Form weiterzugeben.

Dirk Dittert
dirk.dittert@24objects.de

Zur Person: Dirk Dittert
Wie entwickle ich Software richtig? Diese Frage steht bei ihm seit den ersten Programmier-Erfahrungen im Mittelpunkt. Dirk Dittert ist seit JDK 1.2 in der Java Community aktiv. Für seine Firma berät er verschiedene Kunden in den Bereichen „Software-Architektur“ und „Test Driven Development“.

Nachgefragt …

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 hierbei auf Atlassians Jira. Inzwischen sind die Vorbereitungen in vollem Gange und die Entwicklungsabteilung ist vollauf mit der Planung des Prototyps beschäftigt.

Nachdem in einer ersten Gesprächs- und Evaluierungsrunde das Pro und Kontra besprochen wurde (siehe Ausgabe 01/2013), steht die Entscheidung fest, einen ersten Prototyp mit Jira zu implementieren. Dazu wurden exemplarisch die zukünftigen Vorgänge definiert und die dazugehörigen Workflows entwickelt. Während der Geschäftsführer von einer Gewinn-Maximierung träumt und keinen Zweifel hat, dass seine Mitarbeiter das Projekt stemmen werden, klopft es an der Tür des Projektleiters Erwin Müller, dem die Organisation und Umsetzung auferlegt wurde.

Müller: Ja, herein!

Der Entwicklungsleiter, der gegen den Einsatz der neuen Software die meisten Bedenken geäußert hatte, aber überstimmt wurde, betritt das Büro.

Müller: Ach, was verschafft mir denn die Ehre, Sie in meinem Büro begrüßen zu dürfen?

Entwicklungsleiter: Nun ja, es ist mir auch etwas unangenehm. Aber Sie erwähnten bei unserem Gespräch über die Masken und Felder, dass man hier die Sichtbarkeit konfigurieren könnte und auch die Verbindung zu den Vorgängen einstellbar ist. Ich hatte bisher noch nicht die Zeit, mir das im Detail anzuschauen. Meine Mitarbeiter möchten nun implementieren, haben aber noch einige Fragen und vielleicht könnten Sie etwas Zeit erübrigen, die Zusammenhänge aufzuzeigen?

Müller: Das muss Ihnen nicht unangenehm sein. Wir sollten keine Zeit verlieren, trommeln Sie Ihre Leute zusammen!

Ein paar Stunden später ist der Schulungsraum gut gefüllt, der Beamer läuft, Herr Müller hat sein Notebook angeschlossen und auf der Leinwand ist der Browser zu sehen.

Müller: Kollegen, lassen Sie uns gleich einsteigen. Jira ist ein formularbasiertes System. Es lebt davon, dass Felder gefüllt und beim Durchlaufen des Workflows aktualisiert werden. Jira bringt von Haus aus eine ganze Menge „globaler Felder" mit (siehe Abbildung 1). In dem nachfolgenden Screenshot sind das beispielsweise der „Type", der den Namen des Vorgangs anzeigt, der „Status", der den aktuellen „Standort" im Workflow wiedergibt, und die „Resolution", die eine Lösungsmenge definiert.

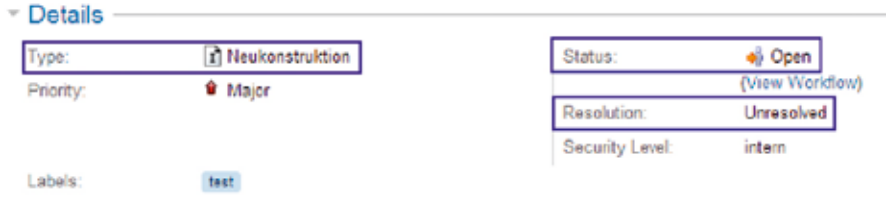


Abbildung 1: Beispiel für globale Felder

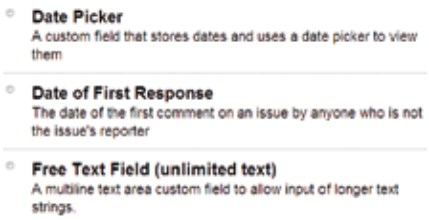


Abbildung 2: Beispiel für Feldtypen

Von diesen Feldern gibt es eine ganze Reihe, die man sich einfach mal in Ruhe anschauen sollte. Reichen diese Felder nicht aus, kann man sich auch eigene definieren, sogenannte „Custom Fields". Dafür steht eine Vielzahl von Feldtypen zur Verfügung, wie man auf diesem kleinen Ausschnitt erahnen kann (siehe Abbildung 2).

Diese Felder können nun auf einem Screen angeordnet werden. Einschränkend muss ich jedoch hinzufügen, dass es die Möglichkeit gibt, „Reiter" hinzuzufügen, jedoch können die Felder, außer in ihrer Reihenfolge, nicht positioniert werden.

Anonymer Zwischenrufer: Was ist mit Pflichtfeldern?

Müller: Kann man definieren. Damit betreten wir jetzt das weite Feld der Konfigurationen. Speziell handelt es sich hier um „Field Configurations". Wie für alle Konfigurationen gibt es hier eine Default-Einstellung (siehe Abbildung 3).

Wenn wir das Feld „Description" betrachten, so befindet sich das Feld auch auf dem Default Screen. Wir können nun festlegen, ob das Feld ein Pflichtfeld sein soll, indem wir diese Funktion mit „required" aktivieren. Damit wird geprüft, ob das Feld einen Wert enthält, bevor die Daten gespeichert werden können.

Anonymer Zwischenrufer: Was bedeutet „hide"? Wenn ich ein Feld nicht sehen will, kann ich es doch einfach löschen?

Müller: Das ist grundsätzlich richtig. Aber gesetzt den Fall, wir haben zwei Vorgänge: Konstruktion und Wartung. Beide haben Formulare mit vielen Feldern und unterscheiden sich im Feld „Dienstleister". In der Wartung wird das Feld angezeigt und ist ein Pflichtfeld, in der Konstruktion ist es ausgeblendet und optional. Natürlich können wir jedem Vorgang eine eigene Maske spendieren. Aber noch einfacher ist es, für beide Vorgänge die gleiche Ansicht zu nutzen und mittels einer speziellen „Field Configuration" die überzähligen Felder auszublenden. Vereinfacht könnte man sagen, man macht bestimmte Eigenschaften der Felder „Name", „Sichtbarkeit", „Pflichtfeld" und „Renderer" abhängig vom aktuellen Vorgang (siehe Abbildung 4).

Entwicklungsleiter: „Renderer"?

Müller: Für mehrzeilige Textfelder kann ein sogenannter „Wiki-Renderer" gesetzt sein, der es erlaubt, den Inhalt mit Wiki-Syntax auszuzeichnen. Auf gut deutsch: Sie können dort Überschriften und Tabellen einfügen, Listen generieren, Abschnitte unterstreichen etc.

Anonymer Zwischenrufer: Angenommen ich möchte beim Anlegen eines Vorgangs Schlüssel vergeben, die ich danach nicht mehr ändern darf. Kann ich diese Felder schützen?

Müller: Also, wenn es hier um einen Schreibschutz geht, den gibt es von Haus aus nicht. Das heißt allerdings nicht, dass es dafür keine Lösung gibt.

Anonymer Zwischenrufer: Ein Schema?

Müller: Genau, „Screen Schemes". Das Prinzip ist recht einfach. Wir nehmen eine fertige Maske, sagen wir für die Konstruktion, kopieren sie dreimal und benennen sie folgendermaßen:

Abbildung 3: Default Field Configuration

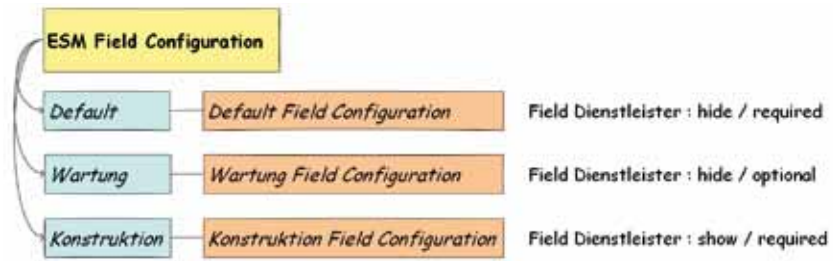


Abbildung 4: Field Configuration Scheme

- Konstruktion Create Screen
- Konstruktion Edit Screen
- Konstruktion View Screen

Diese Masken werden nun den entsprechenden Aktionen „Create" und „Edit View" zugeordnet, wobei man hier auch einen Default-Screen vereinbaren kann. Ich habe das mal in einer Grafik skizziert (siehe Abbildung 5).

Jeder Aktion wird nun eine eigene Bildschirmmaske zugeordnet. Diese kann nun jeweils individuell konfiguriert werden. Das bedeutet, wir können einzelne Felder entfernen und hinzufügen. Jetzt müssen wir jedoch noch vereinbaren, welcher Vorgang mit dieser Kombination an Screens verbunden werden soll. Dafür gibt es dann die sogenannten „Issue Type Screen Schemes". Sie assoziieren auf der nächsten übergeordneten Ebene die Vorgänge (Issues) mit den „Screen Schemes". Das hört sich alles ein wenig kompliziert an, ist es auch. Hier kann ich nur empfehlen, es einfach auszuprobieren.

Ich erläutere dass nochmal am Beispiel der Konstruktion. Das Feld der Betriebsmittelnummer wird nur dem Create- und dem View-Screen zugeordnet. Damit wird es beim Anlegen einmal zur Verfügung gestellt – sollte dort auch ein Pflichtfeld sein – und dann nur noch in der View zur Anzeige gebracht. Eine weitere Änderung ist nun nicht mehr möglich.

Entwicklungsleiter: Man muss sich offensichtlich einige Gedanken im Vorfeld machen.

Müller: Allerdings!

Entwicklungsleiter: Wir haben bisher noch nicht darüber gesprochen, wie die Workflows da hineinspielen. Ich würde mal vermuten, es wird noch komplexer?

Müller: Ich fürchte, ja. Aber sie bieten auch eine Menge weiterer Möglichkeiten.

Entwicklungsleiter: Aha, ist das für uns relevant? Was ich damit sagen will, sollten wir uns in diesem Rahmen auch damit näher beschäftigen?

Müller: Auf jeden Fall.

Entwicklungsleiter: *Seufzt.* Fahren Sie fort.

Müller: Was ist ein Workflow? In erster Linie geht es um Zustandsänderungen, die ein Vorgang durchlebt. Jira bringt von Haus aus einen Standard-Workflow mit. Grundsätzlich kann er sofort verwendet werden, aber das ist jetzt gar nicht der Punkt, auf

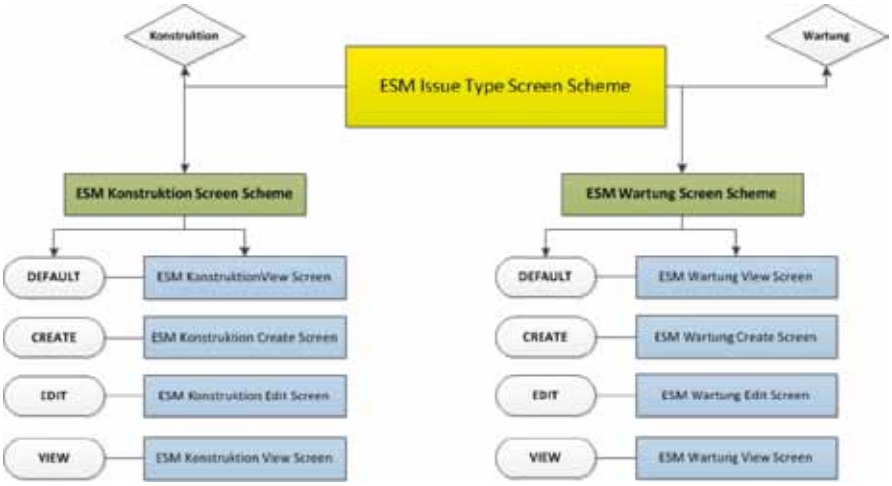


Abbildung 5: Screen Scheme

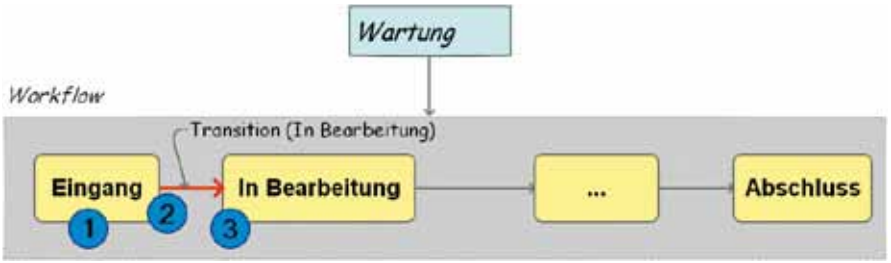


Abbildung 6: Workflow Transition

Abbildung 7: Schaltzentrale der Transition-Funktionen

den es mir ankommt. Einen Workflow im grafischen Editor anzupassen, ist trivial. Ich bevorzuge allerdings die Text-Variante, die mir etwas übersichtlicher und robuster erscheint.

Den Wechsel von einem Zustand in einen anderen nennt man eine „Transition". Ich habe das am Beispiel der Statuswerte „Eingang" und „In Bearbeitung" aufgezeichnet (siehe Abbildung 6). Als Bedingung für einen Übergang lassen sich „Conditions" definieren. Darüber hinaus gibt es „Validierungen" und „Post Functions".

Wir können hier nun an der Oberfläche kratzen. Für alle Bedingungen gibt es sowohl zusätzliche Plug-ins als auch die Möglichkeit, Scripting einzusetzen. Darüber können wir uns später mal unterhalten. So, hier mal ein kurzer Abriss, was wir hier alles machen können (siehe Abbildung 7).

Conditions

Ich muss zugeben, dass sich mir die Conditions rein praktisch nicht so recht erschließen. Es ist möglich, hier Prüfungen zu definieren, die den Übergang in einen anderen Status sozusagen freischalten. Konkret bedeutet das: Ist eine Bedingung nicht erfüllt, steht die Schaltfläche für den Übergang erst gar nicht zur Verfügung. Ich habe allerdings keine Möglichkeit gefunden, wie man die Information darüber, was genau fehlt, dem Anwender vermittelt. Es kann also passieren, dass man keine Ahnung hat, warum es nicht weitergeht.

Anonymer Zwischenrufer: Was kann hier überhaupt geprüft werden?

Müller: Die Bandbreite ist ziemlich groß, aber es gibt hier schon ein paar Schwerpunkte:

- Prüfung von Berechtigungen. Beispielsweise dürfen nur der Bearbeiter oder der Reporter den Statuswechsel vornehmen.
- In der Softwareentwicklung kann ein Review von Code-Änderungen erzwungen werden.
- Es kann ein bestimmter Inhalt von Feldern (Stati, Versionen etc.) abgefragt werden.

Eigene Erweiterungen sind mit Skripten möglich, in denen beispielsweise ganz spezifische JQL-Abfragen getätigt werden können. Um Ihren Fragen zuvorzukommen, JQL ist das Mittel der Wahl, eigene Filter und Reports zu definieren.

Validators

Validatoren sind recht schnell erklärt und intuitiv einsetzbar. Wie der Name bereits sagt, können Felder inhaltlich geprüft werden. Schlägt so eine Prüfung fehl, erscheint eine Fehlermeldung, die genau darauf hinweist. Das ist ziemlich praktisch in einer Situation, in der ein Feld nur für diesen Übergang „required" ist. Man kann hier beliebige Felder prüfen lassen. Sonst gibt es noch:

- Felder können miteinander verglichen werden
- Der übergeordnete Vorgang muss sich in einem speziellen Status befinden, womit auch Abhängigkeiten zwischen Haupt- und Subtasks verifiziert werden können.
- Natürlich spielen auch Berechtigungen der Anwender und Skripte eine große Rolle. Gerade hier existieren eine Reihe vorgefertigter Groovy-Snippets, die einen kleinen Eindruck davon vermitteln, was alles in diesem Bereich möglich ist. Dass würde allerdings den heutigen Rahmen sprengen. Ich muss auch zugeben, dass ich das noch nicht bis in die letzte Tiefe ausgelotet habe.

Hier ist das Template, das beliebig erweitert werden kann (siehe Abbildung 8). Übrigens ist das Forum von Atlassian sehr aktiv und hier findet man eine Vielzahl von Beispielen, wie solche Anfragen aussehen können.

Post Functions

In vielen amerikanischen Serien gibt es ja Sweeper-Teams, die das Chaos aufräumen,

```
1 cfValues['SomeCustomField'] == 'Some Value'
```

Abbildung 8: Einfache Groovy-Abfrage

das die SWAT-Teams hinterlassen haben. So ähnlich ist das mit den „Post Functions". Sie sind das „Mädchen für alles" und hier kann man all das erledigen, was auf direktem Wege nicht so ohne Weiteres möglich ist.

Grundsätzlich sind bereits einige Funktionen vordefiniert. Interessant ist hier besonders auch der „Update Change", der als Zeitstempel für das Erreichen eigener Meilensteine verwendet werden kann. Auch hier kann ich nur eine Idee der sehr umfangreichen Funktionalität vermitteln:

- Die nachgelagerten Aktionen erlauben es uns, Feldwerte zwischen Haupt- und Unteraufgaben zu synchronisieren. Das hat den Charme, dass eine Unteraufgabe relevante Daten von ihrer übergeordneten Aufgabe übernehmen kann, ohne dass der Anwender sie erneut eingeben muss.
- Mit der Transition kann eine Zuweisung der Verantwortlichkeit automatisiert werden. Beispielsweise gibt es für einen bestimmten Status immer eine Rolle, die hier verantwortlich zeichnet.
- Werte für eigene und Standard-Felder können gesetzt werden. Angenommen, der Vorgang soll abgebrochen werden, dann kann die „Resolution" automatisch durch das System zugewiesen werden.
- In den Post Functions finden wir die umfangreichsten Skripting-Funktionen überhaupt. Sie ermöglichen es uns auch, auf sehr flexible Art und Weise „Notifications" in speziellen Situationen zu senden, die durch das allgemeine „Notification"-Konzept nicht abgedeckt sind.

Anonymer Zwischenrufer: Das hört sich alles nach „Eitel Sonnenschein" an. Da lauert doch bestimmt noch irgendwo ein „Aber"?
Müller: Na ja, es gibt da schon ein paar Einschränkungen. Ein großes Problem ist die Dokumentation. Wie halte ich alle Entscheidungen fest, die im Vorfeld der Konfiguration für das Projekt und später während der Implementierung getroffen werden? Auch bedingen sich manche Abhängigkeiten und führen zu seltsamen Effekten, in denen Felder mal nicht sichtbar sind oder als erforderlich angemahnt, aber gar nicht in der Maske enthalten sind. Zumeist lässt sich das mit einer gründlichen Analyse beheben. Auch sind klare Grenzen im „Look & Feel" gesetzt, was schnell zum Frust führen kann, wenn die Bildschirmseite einfach nicht gut aussieht. Es ist halt sehr wichtig, sich intensiv mit dem System zu beschäftigen und damit zu arbeiten, und bei Fragen stehe ich natürlich immer gerne zur Verfügung. Wenn es keine weiteren Fragen gibt, bedanke mich für Ihre Aufmerksamkeit!
Entwicklungsleiter: Herr Müller, ich bin beeindruckt, wie schnell Sie sich in dieses Thema eingearbeitet haben und auf jede Frage eine fundierte Antwort haben. Darf ich auf Sie zählen, wenn wir im weiteren Verlauf des Projekts an Grenzen stoßen?
Müller: Ja, natürlich.
Entwicklungsleiter: Ausgezeichnet, danke!
Nachdem der Entwicklungsleiter den Raum verlassen hat, bleiben nur noch der anonyme Zwischenrufer und Müller zurück.
Anonymer Zwischenrufer: Na Herr Müller, wie war ich?
Müller: Es hat tatsächlich funktioniert! Ich hatte befürchtet, dass Sie meine Handschrift auf dem Fragenzettel nicht lesen könnten. Es musste halt alles ziemlich schnellgehen. Wir sind ein gutes Team, vielen Dank!

Literatur/Web

[1] http://www.atlassian.com/software/jira/overview
[2] https://answers.atlassian.com/

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".

Fernbedienung verbunden mit Raspberry Pi

Den Rollladen steuern mit Pi

René Jahn, SIB Visions GmbH

Die Steuerung von Rollläden kann heutzutage bequem per Fernbedienung oder durch eine zentrale Steuereinheit erfolgen. Der Einsatz von zentralen Steuereinheiten automatisiert wunderbar alltägliche Abläufe wie zeitgesteuertes Öffnen und Schließen. Doch nicht jeder Haushalt mit elektronischen Rollläden besitzt auch eine zentrale Steuereinheit, denn diese sind meist recht teuer und erfüllen äußerst selten die gewünschten Anforderungen. Wer auf keine Funktion verzichten möchte und schon immer von einer eigenen Rollladensteuerung träumte, findet hier eine Anleitung.

Wer bisher seine Rollläden mit einer Fernbedienung steuert, wird folgende Probleme nur allzu gut kennen. Man fährt in den Urlaub und lässt die Rollläden entweder unten, oben oder einen Spalt geöffnet, damit die Pflanzen auch etwas Licht bekommen. Im Idealfall kümmert sich eine Nachbarin um das Problem, doch die optimale Lösung ist das nicht. Es kann aber auch schon einmal vorkommen, dass ganz einfach vergessen wurde, die Rollläden zu schließen, weil die Zeit etwas knapp war. Man bemerkt das üblicherweise erst, wenn es zu spät ist. In diesem Fall wäre es natürlich wünschenswert, die Rollläden irgendwie runterfahren zu können. Ohne zentrale Steuereinheit lassen sich die Probleme jedoch nicht lösen. Denn nur damit kann festgelegt werden, wann die Rollläden hoch- oder runterfahren.

Es gibt auch praktische Erweiterungen, etwa für die tageslichtabhängige Steuerung oder spezielle Smartphone Apps. Doch selbst damit gibt es Einschränkungen und als Software-Entwickler möchte man ja auf keinen Komfort verzichten. Vor allem nicht, wenn man es selbst besser machen könnte. Genau hier beginnt es spannend zu werden, denn wir entwickeln einfach unsere eigene zentrale Steuereinheit, die noch dazu kostengünstig ist.

Das Ziel ist klar definiert: Wir wollen unsere Rollläden per SMS öffnen oder schließen. Um das zu realisieren, benötigen wir jedoch mehr als nur ein Stück Software. Denn irgendwie müssen wir die Rollläden auch ansteuern. Doch welche Hardware kann das? Es scheint ja eigentlich ganz trivial zu sein, denn eine Fernbedienung schickt bestimmte Signale zu den Empfängern der Rollläden. Es sollte also ausreichen, diese Signale aufzuzeichnen, um sie später wieder senden zu können. Dazu wären nur ein Sender für Funksignale und ein Oszilloskop erforderlich, um die Signale aufzuzeichnen. Natürlich muss man die richtigen Signale am Ende noch softwaretechnisch erzeugen. Doch so einfach und günstig, wie sich der Autor das vorgestellt hatte, war es nicht. Es war schon nicht möglich, einen geeigneten Sender für die benötigte Frequenz zu organisieren, ohne gleich eine komplette Schaltung zu bauen.

Weil das Ganze auch ohne notwendiges Expertenwissen nicht umsetzbar gewesen wäre, musste eine andere und vor allem einfache Lösung her. Am einfachsten erschien es, Software-gesteuert einen Tastendruck auf der Fernbedienung durchzuführen. Die

Abbildung 1: Fernbedienung mit aufgesetzter Taster-Steuerung

Fernbedienung war sowieso schon vorhanden und so musste nur überlegt werden, wie eine Taste gedrückt werden konnte und vor allem, wodurch? Dieses Problem war relativ leicht lösbar, da jede Taste auf der Fernbedienung lediglich einen Stromkreis schließt.

Um einen Stromkreis zu schließen reicht es aus, einen Schalter umzulegen, und genau das ist die Aufgabe von Relais. Somit war dieses Problem gelöst und es fehlte nur noch eine Lösung für die Ansteuerung von Relais. Eine übliche Lösung wäre der Einsatz einer seriellen Schnittstelle (RS232) oder eines „USB zu RS232"-Adapters. Für die Ansteuerung der seriellen Schnittstelle existiert mit RxTx [1] eine praxiserprobte Open-Source-Lösung. Bei seriellen Schnittstellen kommt jedoch schnell der Gedanke hoch, dass ein Notebook oder Desktop PC notwendig ist. Die Lösung sollte aber auf keinen Fall mehr Strom verbrauchen als unbedingt nötig, daher musste eine schlanke Alternative her.

Der Raspberry Pi

Spätestens seit der JavaOne 2012 sind Embedded Devices in aller Munde. Allen voran der Raspberry Pi. Ein kreditkartengroßer Computer mit ausreichend Rechenpower und allen notwendigen Schnittstellen wie USB, LAN und HDMI. Auf der Konferenz wurde jedoch vor allem die Verwendbarkeit von JavaFX auf Embedded Devices hervorgehoben. Das beeindruckte einerseits und verhalf andererseits JavaFX zu mehr Akzeptanz. Seit diesem Event haben JavaFX und der Raspberry Pi auch den Autor in ihren Bann gezogen.

Aus diesem Grund war auch für die Rollladensteuerung ein Raspberry Pi (RasPi) als Hardware gesetzt. Ein großer Vorteil war

```
public static void main(String[] pArgs) throws Exception
{
 System.out.println("== GPIO control Test ==");
 GpioController gpio = GpioFactory.getInstance();
 GpioPinDigitalOutput pinDown;
 GpioPinDigitalOutput pinUp;
 GpioPinDigitalOutput pinStop;
 pinDown = gpio.provisionDigitalOutputPin(
        RaspiPin.GPIO_01, "Down", PinState.LOW);
 pinUp = gpio.provisionDigitalOutputPin(
       RaspiPin.GPIO_02, "Up", PinState.LOW);
 pinStop = gpio.provisionDigitalOutputPin(
        RaspiPin.GPIO_03, "Stop", PinState.LOW);
 System.out.println("--> GPIO down ON");
 pinDown.high();
 Thread.sleep(1000);
 pinDown.low();
 System.out.println("--> GPIO down OFF");
 Thread.sleep(5000);
 System.out.println("--> GPIO up ON");
 pinUp.high();
 Thread.sleep(1000);
 pinUp.low();
 System.out.println("--> GPIO up OFF");
 Thread.sleep(5000);
 System.out.println("--> GPIO down ON");
 pinDown.high();
 Thread.sleep(1000);
 pinDown.low();
 System.out.println("--> GPIO down OFF");
 Thread.sleep(2000);
 System.out.println("--> GPIO stop ON");
 pinStop.high();
 Thread.sleep(1000);
 pinStop.low();
 System.out.println("--> GPIO stop OFF");
 Thread.sleep(5000);
 System.out.println("--> GPIO down ON");
 pinDown.high();
 Thread.sleep(1000);
 pinDown.low();
 System.out.println("--> GPIO down OFF");
}
```

Listing 1

```
SerialModemGateway modem;
modem = new SerialModemGateway(
   "modem","/dev/ttyUSB0", 115200,"Cinterion","EGS3");
modem.setSimPin("0000");
modem.setInbound(true);
modem.setSmscNumber("");
Service serv = Service.getInstance();
serv.addGateway(modem);
serv.startService();

//check incoming messages
List<InboundMessage> msgList = new ArrayList<InboundMessage>();
Service.getInstance().readMessages(msgList,
          InboundMessage.MessageClasses.ALL);
String sText;
for (InboundMessage msg : msgList)
{
 sText = msg.getText();

 if ("UP".equalsIgnoreCase(sText))
 {
  up();
 }
 else if ("DOWN".equalsIgnoreCase(sText))
 {
  down();
 }
}
serv.stopService();
```

Listing 2

GPIO Pin	Funktion
1	AB
2	AUF
3	STOPP

Tabelle 1

auch, dass der RasPi problemlos mit dem USB-Port eines WLAN-Routers betrieben werden kann. Der Stromverbrauch ist minimal. Zudem besitzt der RasPi mit den GPIO-Pins eine sehr gute Möglichkeit, digitale Signale zu schalten. Die serielle Schnittstelle beziehungsweise ein USB-Port musste somit nicht belegt werden, um die Relais anzusteuern. Nachdem alle Hardware-Probleme für die Rollladensteuerung gelöst waren, wurde mit etwas Geschick und einem Lötkolben die Hardware zusammengesetzt (siehe Abbildung 1).

Als Relais kamen G6L-1P (3V) von Omron zum Einsatz und für die Verbindung mit dem RasPi wurde ein USB-Port aufgelötet. Für die Steuerung von „Auf", „Ab" und „Stopp" wurde je ein Relais verbaut. Ein USB-Kabel stellte abschließend die Verbindung zum RasPi her. Das Praktische an der Lösung ist, dass die Fernbedienung auch weiterhin manuell bedient werden kann.

Die Software-Ansteuerung

Die Hardware ist eine sehr wichtige Komponente für die Steuerung. Doch ohne passende Software funktioniert auch unsere Lösung nicht. Der erste Test erfolgt mithilfe eines Java-Programms (siehe Listing 1).

Zur Ausführung am RasPi ist ein Java-Runtime-Environment für ARM-Prozessoren erforderlich. Das kann eine JavaSE-Embedded-Version oder auch die JDK8-Early-Access-Version [2] mit Hard-Float-Unterstützung sein. Zusätzlich wird die Open-Source Bibliothek „Pi4J" [3] benötigt, die

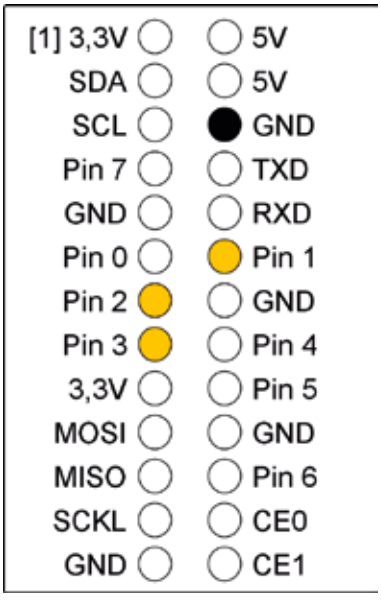


Abbildung 2: Raspberry Pi (Model B), GPIO für Pi4J

Abbildung 3: JavaFX-Applikation zur Steuerung

den Zugriff auf die GPIO-Pins mit einem einfachen API bietet. Das Testprogramm prüft, ob die Rollläden nach unten beziehungsweise oben gefahren und gestoppt werden können. Es setzt eine definierte GPIO-Pin-Belegung voraus (siehe Tabelle 1). Abbildung 2 zeigt die vollständige GPIO-Pin-Belegung, angepasst an die Verwendung mit Pi4J.

Die größten Hürden sind überwunden und die Steuerung der Rollläden kann bereits durch unsere Software erfolgen. Ein entscheidender Punkt fehlt allerdings noch: Wir wollten SMS-Nachrichten für die Bedienung einsetzen. Damit unsere Software SMS verarbeiten kann, ist weitere Hardware notwendig. Es eignen sich sowohl Mobiltelefone mit integriertem Modem als auch spezielle GSM-Module, wie sie beispielsweise Cinterion anbietet. Es sollte jedoch darauf geachtet werden, dass die eingesetzte Java-Bibliothek etwa mit „SMSLib" [4] die GSM-Hardware unterstützt.

Die GSM-Hardware wird über eine serielle Schnittstelle (RS232) oder einen „USB zu RS232"-Adapter angeschlossen. Die Ansteuerung mit Java erfolgt unter Verwendung von SMSLib (siehe Listing 2).

Je nachdem, welche GSM-Hardware zum Einsatz kommt, kann es zu Problemen bei der Kommunikation kommen. In unserem Fall mussten zusätzliche System-Properties gesetzt sein, damit die Hardware mit dem RasPi funktionierte: "-Dsmslib.serial.polling -Dsmslib.at.wait=500 -Dsmslib.nocops=1"-.

Das definierte Ziel ist damit erreicht. Auch wenn es sich noch nicht um eine vollständige Applikation handelt, sollte es nicht sehr schwierig sein, die einzelnen Teile zusammenzusetzen.

Erfahrungsbericht

Die beschriebene Lösung steuert Rollläden des Autors seit Beginn des Jahres im 7x24-Betrieb und hat bereits den ersten Urlaub erfolgreich gemeistert. Natürlich wurden zahlreiche Erweiterungen integriert, etwa das Setzen der Öffnungs- und Schließzeiten per SMS, das Ein-/Ausschalten der automatischen Steuerung, ein Wochenendmodus etc. Zusätzlich wurde eine JavaFX-Applikation zur Steuerung umgesetzt (siehe Abbildung 3).

Aufgrund der integrierten Effektmöglichkeiten von JavaFX war die Umsetzung des Bewegungsablaufes ein Leichtes und die App bewegt den Rollladen zeitgleich zur Realität.

Fazit

Die Entwicklung der Lösung bereitete großen Spaß, da gerade das Zusammenspiel von Hard- und Software ein spannendes Thema war. Vor allem die Verwendung des RasPi brachte enorme Zufriedenheit und bietet tolle Möglichkeiten zur (Heim-) Automatisierung zu einem unschlagbaren Preis.

Links

[1] RxTx: http://rxtx.qbang.org
[2] JDK8 Early Access: http://jdk8.java.net/fxarm-preview/index.html
[3] Pi4J: http://www.pi4j.com
[4] SMSLib: http://smslib.org/doc/compatibility

René Jahn
rene.jahn@sibvisions.com

René Jahn ist Mitbegründer der SIB Visions GmbH und Head of Research & Development. Er verfügt über langjährige Erfahrung im Bereich der Framework- und API-Entwicklung. Sein Interessenschwerpunkt liegt auf der Integration von State-of-the-Art-Technologien in klassische Business-Applikationen. Unter anderem betreut er die Open-Source-Sparte bei SIB Visions und veröffentlicht regelmäßig Artikel im Unternehmensblog.

ADF Essentials Mobile – Ein unlösbares Problem?

Markus Klenke, TEAM GmbH

Anwendungsentwicklung für mobile Endgeräte ist für Oracle in den letzten Jahren immer weiter in den Vordergrund gerückt. Mit der jüngsten Version des Oracle JDevelopers sind zum ersten Mal die neuen ADF-Mobile-Komponenten verfügbar.

Um auch ohne große Lizenzkosten die Vorzüge der ADF-Entwicklung kennenzulernen, hat Oracle zusätzlich ADF Essentials veröffentlicht, eine Version von ADF, die auch auf lizenzkostenfreien Servern wie dem GlassFish 3.1 aufgespielt werden darf. Im gleichen Atemzug, in dem die mobilen ADF-Komponenten von Oracle angekündigt wurden, wurde jedoch bekannt gegeben, dass die mobilen Anteile in den Essentials nicht enthalten sind. Bedeutet das: „Keine mobile ADF-Entwicklung für kleinere Unternehmen?"

TEAM sagt „Nein!". Auch ohne die ADF-Mobile-Elemente war es schon möglich, mobile Anwendungen in ADF zu erstellen. Anhand des TEAM iConfGuide, mit dem Besucher der DOAG-Konferenz ihr eigenes Konferenzprogramm zusammenstellen und verwalten können, stellt der Artikel einige interessante Konzepte und Möglichkeiten vor, wie ADF-Entwicklung für Mobilgeräte auch ohne die ADF-Mobile-Komponenten aussehen kann.

ADF Essentials? Nie gehört.

Grob formuliert könnte man sagen, dass in ADF Essentials die Basics von ADF enthalten sind. Das impliziert Oracle-Elemente wie ADF Business Components, den ADF Model Layer, ADF Task Flows sowie mehr als 150 Komponenten aus den ADF Faces. Letztere erweitern den Java-Standard der Java Server Faces. Somit können mit den Essentials viele Übungsapplikationen erstellt werden, beispielsweise aus dem Oracle-by-Example-Bereich. Um allerdings eine produktiv einsetzbare Applikation zu erstellen, bedarf es weiterer Funktionalitäten. Dazu gehören Sicherungskomponenten für eine Anwendung, die Bereitstellung von Schnittstellen für zusätzliche Datenquellen und Komponenten wie Web-Services oder Business Intelligence, aber auch ein Metadaten-Speicher, um persönliche Einstellungen des Benutzers abzulegen. Viele der ADF-spezifischen Ausprägungen der oben genannten Funktionalitäten sind aus den ADF Essentials herausgenommen. Dazu zählen zum Beispiel:

- Oracle ADF Security
- Oracle ADF Web Service Data Control
- Oracle Meta Data Storage
- Oracle Business Component Service Interfaces
- Oracle Remote ADF Task Flow Call

Durch Java-Herkunft und den durchgängigen Einsatz des MVC-Design-Patterns im ADF Framework ist es jedoch möglich, diese Komponenten durch Open-Source-Elemente zu ersetzen. So kann statt der ADF Security beispielsweise ein individuelles Security HTTP Servlet zusammen mit einer Benutzer-JavaBean eingebunden werden, um User-Management für die Applikation bereitzustellen.

Die Theorie klingt überzeugend

Nachfolgend wird der iConfGuide von TEAM als Anschauungsmaterial für die Entwicklung einer mobilen ADF-Applikation ohne ADF-Mobile-Komponenten und mit den genannten optionalen Austauschmöglichkeiten betrachtet. Der iConfGuide (siehe Abbildung 1) ist seit 2010 für die unterschiedlichen DOAG-Konferenzen im Einsatz. Die Applikation ermöglicht es den Benutzern, einfach und schnell auf die Informationen der Aussteller sowie auf die Termine der Vorträge zugreifen zu können und sich diese als Favoriten markiert zu speichern.

Im Kern ist die Applikation über ADF Business Components an eine Oracle-Datenbank angebunden. Sie benutzt das ADF Model und setzt dann auf das allgemeine JSF Controlling und die allgemeinen Trinidad-JSF-Oberflächen-Elemente auf. In den nächsten Abschnitten wird der Aufbau der vorliegenden Applikation als Beispiel für ähnlich gelagerte, mobile Entwicklungsvorhaben vorgestellt.

Gibt es einen Entwicklungsfahrplan?

Wie bei der Entwicklung einer typischen ADF-Applikation wurde auch beim iConfGuide das Augenmerk sehr stark auf den Anwendungsfluss und die Benutzer-Navigation gelegt. Durch die begrenzte Oberfläche auf mobilen Geräten hat man auf Features wie „Partial Page Refresh" oder aufwändiges Layouting verzichtet. Daher waren Komponenten wie „ADF Bounded Task Flows" und Oberflächen-Elemente wie „ADF-Regionen" erforderlich. Man konnte mit dem gegebenen Java-Server-Faces-Controller-Standard arbeiten, da dieser die notwendigen Java Bean Scopes („session" und „request") bereitstellt. So kann mit

Abbildung 1: Ausstellerauswahl im iConfGuide

gerade einmal drei verschiedenen Komponenten der gesamte Applikationsfluss dargestellt und der Benutzer auf einfache, visuelle Art und Weise durch die Anwendung geführt werden. Das Navigationsverhalten der Applikation kann also rein deklarativ über einen Graphen dargestellt werden (siehe Abbildung 2), wobei von jeder Seite über einen einfachen String-Aufruf ein anliegender Navigationsfall eingeleitet werden kann.

Da in den ADF Essentials das Benutzen mancher Komponenten der Task Flows (wie zum Beispiel das Aufrufen externer Task Flows) untersagt ist, stellt die Nutzung des allgemeinen JSF Controllers anstelle des ADF Controllers eine sehr simple, aber dennoch effiziente und übersichtliche Methode dar, den Controller Layer einer eventuell komplexen Applikation zu entwerfen.

Das Handy als Schlüssel für Metadaten-Speicher

Der ursprüngliche Ansatz, dem Benutzer Individualisierungen in der Applikation zu ermöglichen, beruhte auf dem Oracle Meta Data Storage (MDS) und der Anmeldung des Benutzers via HTTP Servlet. Wegen der hohen Anzahl der unterschiedlichen Nutzer, der früheren Instabilität des Oracle MDS und des umständlichen Anmeldevorgangs per Handy wurde nach einer anderen Lösung für die favorisierten Einstellungen des Benutzers gesucht. Als eine bequeme Möglichkeit bietet sich der Einsatz von Cookies an. Möchte der Benutzer seine Favoriten anschauen, so wird über einen gesetzten Cookie eine User ID ausgelesen, die als Suchwert in einer Favoriten-Tabelle der Datenbank fungiert. Ist dieser Cookie nicht gesetzt, wird ein neuer angelegt (siehe Listing 1).

Über diese ID lassen sich weitere Funktionalitäten wie Mehrsprachigkeit oder ein benutzerabhängiges Tracing realisieren. Sofern der Benutzer diesen Cookie speichert, ist ihm über die gesamte Zeitspanne der DOAG-Konferenz ein eindeutiger Benutzer zugewiesen. Auf diese Weise können die für die Applikation wichtigen Elemente, wie die favorisierten Vorträge, ohne den Oracle Meta Data Storage gespeichert werden.

Für weitere Updates der Software und für eventuelle Security-Implementierungen lassen sich in die ADF- und Trinidad-Mobile-Komponenten natürlich auch Erweiterungen im HTML5-Kontext einbinden. So kann die in den ADF Essentials fehlende ADF Security beispielsweise auch über die schon genannte HTTP Security, eine Web-Socket-Verbindung und den Handshake, der den Client mit dem Server bekannt macht, ersetzt werden und über einen Session-gebundenen Schlüssel eine Autorisierung des Benutzers in der Applikation stattfinden.

Das Rote oder das Grüne?

Die erste Version des iConfGuide wurde, wie man dem Namen vielleicht entnehmen kann, für die Benutzung auf dem iPhone optimiert. Gefordert war, dass die Applikation sich perfekt in das Look and Feel des iPhones integrieren lässt. Die heutigen ADF-Mobile-Komponenten werden im JSF-Rendering-Prozess automatisch auf den jeweiligen Browser optimiert und ein Großteil vorgegebener Skins liegt im JDeveloper als Erweiterungsgrundlage vor. In unserem Fall wurde zu Beginn der Entwicklungsphase über einen Styleguide festgelegt, welche Trinidad-Komponenten für die Entwicklung genutzt werden, und daraufhin ein CSS-Sheet entworfen, das den Ansprüchen und den Anforderungen genügt.

Durch die kontinuierliche Weiterentwicklung der mobilen Systeme, deren jeweiliger Betriebssysteme und Browser wurden weitere Stylesheets notwendig, um jeden Benutzer zu bedienen. Um die Größe der einzelnen Stylesheets übersichtlich zu halten, bietet es sich an, eine Skin Family für Mobile-orientierte Anwendungen bereitzustellen. So lassen sich auf Basis einer Default-mobile.css-Erweiterung Sheets für die jeweiligen Browser erstellen. Über einen HTTP Servlet Request, der wiederum durch den Java-Server-Faces-Kontext der Anwendung gestellt wird, kann erfragt werden, welcher Browser vom System genutzt wird und wo das jeweilige .css-File für die Anwendung geladen werden kann, um somit das gewünschte Look and Feel bereitzustellen (siehe Listing 2).

Da hier auf reinen Trinidad-Komponenten gearbeitet wird, ohne auf den ADF-Standard zuzugreifen, sind die Regularien

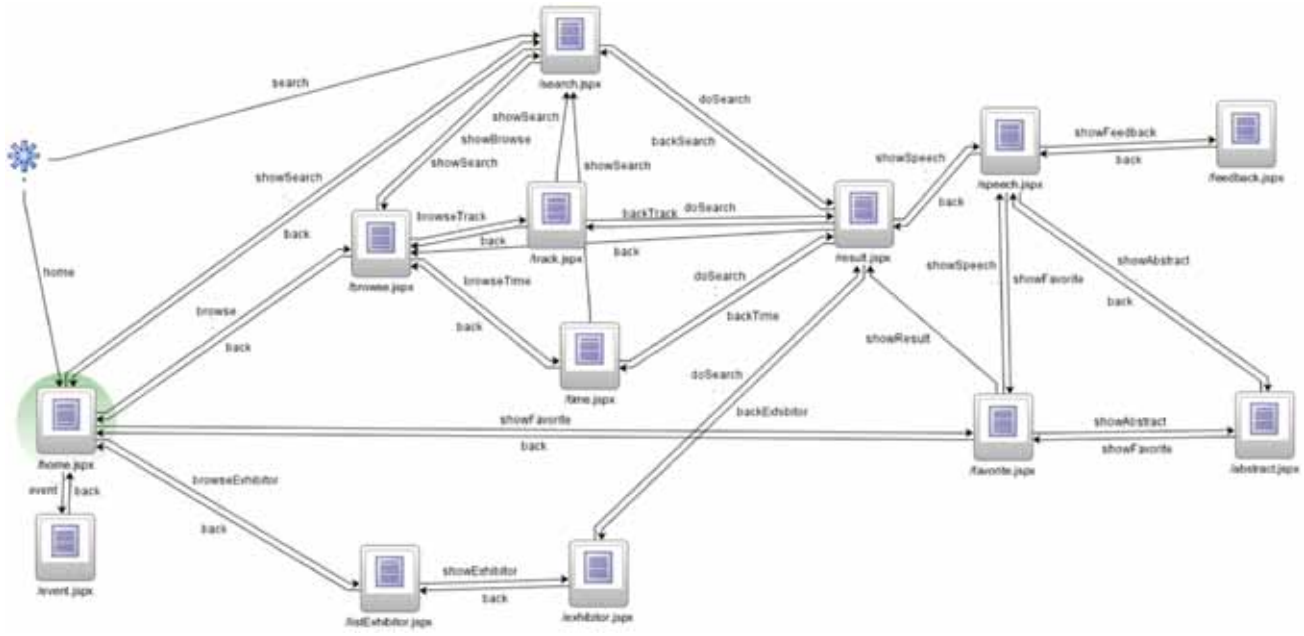


Abbildung 2: Navigationsstruktur des iConfGuide mithilfe von JSF Controlling

```
String getUser()
{
        if cookieFound{
                cookieValue = getCookie(USERID_COOKIE_NAME);
        }
        else{
                cookieValue = getRequest().getSession().getID();
                addCookie(USERID_COOKIE_NAME,
                        cookieValue,ID_COOKIE_MAXTIME);
        }
        return cookieValue;
}
```

Listing 1

```
Skin getPhoneBrowser() {
        FacesContext fc = FacesContext.getCurrentInstance();
        HttpServletRequest req =
        (HttpServletRequest)fc.getExternalContext().getRequest();
        String agent = req.getHeader("User-Agent");
        Skin skin;
        if (agent != null && agent.indexOf("iPhone") > -1) {
                skin = Skin.IPHONE_SKIN;
        }
        else if (…)
        {….}
}
```

Listing 2

von ADF Essentials weiter erfüllt. Somit darf die Applikation auf einem lizenzkostenfreien Server gestellt und betrieben werden.

ADF Essentials vs. Open Source

Wie man an den letzten drei Abschnitten erkennen kann, ist die Entwicklung von mobilen ADF-Anwendungen auch zu bewerkstelligen, wenn man nur die in den ADF Essentials bereitgestellten Komponenten nutzt und weitere Open-Source-Elemente hinzufügt. Das ADF Framework erlaubt einen fließenden Übergang zwischen verschiedenen Ausprägungen der Applikationsschichten. Man kann sich allerdings die Frage stellen, wieso überhaupt auf das ADF Framework aufgesetzt wird. Die Entwicklungen der Java-Standards wie EJB und JSF schreiten weiter voran und auch die Entwicklung geeigneter IDEs, um beispielsweise deklarativ zu entwickeln, ist zu erkennen. Zudem könnten manche der durch JSF-Standards gelösten Problemstellungen mit anderen Frameworks eleganter gelöst werden. Dennoch sind JDeveloper und ADF im Vorteil, wenn es sich bei den zugrunde liegenden Technologien um Oracle-Produkte (Oracle-Datenbank, WebLogic-Server) handelt, da in dem Framework der Übergang der einzelnen Schichten so einfach wie möglich gemacht wird (siehe Abbildung 3).

Die ADF Business Components bieten beispielsweise vor allem Entwicklern mit viel Datenbank- beziehungsweise Forms-Erfahrung eine bekannte Schnittstelle. Die notwendigen ADF-View-Objekte werden mithilfe von SQL-Queries definiert und so wird ein typischer Arbeitsablauf aus dem Datenbank- und Forms-Umfeld generell nur in eine andere Applikationsschicht verlegt. Der ADF Model Layer bietet eine sehr hilfreiche abstrakte Schnittstelle zu Datenanbietern, sodass auch ein optionaler Wechsel an dieser Stelle einfach zu realisieren ist.

Darüber hinaus sind auch die ADF Data Bindings sowie der ADF Controller mit Open-Source-Elementen nutzbar und bieten somit eine Schnittstelle für beliebig erweiterbare Komponenten. Die ADF Essentials beinhalten, wie eingangs erwähnt, jede dieser Komponenten. Dadurch, dass die Einschränkungen jeweils nur die einzubindenden Elemente betreffen, nicht aber die Schnittstellen oder die Abstraktionsebenen selbst, kann man die Stärke des Frameworks auch dann nutzen, wenn beispielsweise die ADF-UI-Komponenten selbst (wie im Beispiel des iConfGuide) nicht genutzt werden, sondern eine alternative Oberflächenlösung gewählt wurde.

Fazit

Oracle ADF ist ein flexibles Framework mit schier endlosen Variationsmöglichkeiten. Als Einsteiger kann man von der reinen Masse schnell überwältigt werden und diese Vielfalt kostet seinen Preis. Mithilfe der ADF Essentials bietet Oracle nicht nur eine Möglichkeit, sich einen Überblick über Technologie und die Komponenten zu verschaffen, sondern eine wirkliche Alternative für kleinere Unternehmen und Entwicklungsteams, die bis dato vor den Lizenzkosten eines WebLogic-Servers zurückgeschreckt sind.

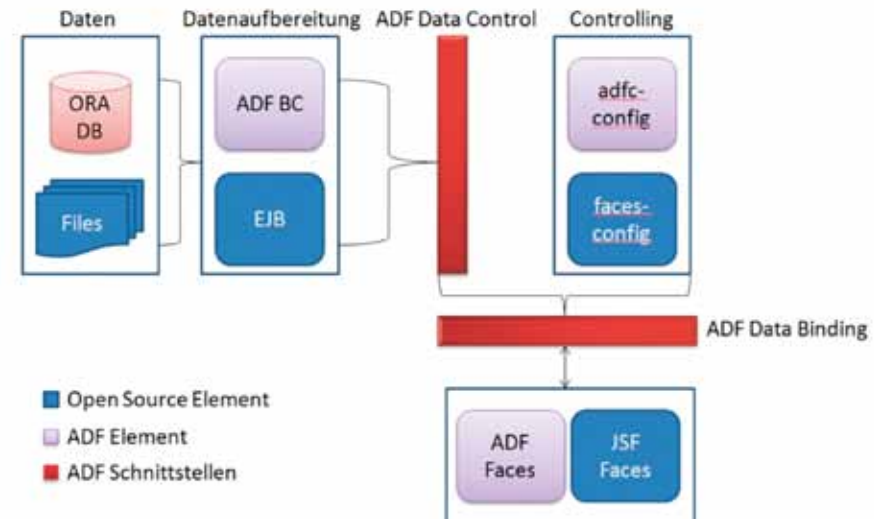


Abbildung 3: MVC Pattern mit ADF-Komponenten

Markus Klenke
mke@team-pb.de

Markus Klenke ist als Softwareentwickler spezialisiert auf die Entwicklung von Oracle-ADF-Anwendungen und Integrationsumgebungen. Durch Framework-Kenntnis und eine mathematisch abstrakte Denkweise unterstützt er als Consultant Unternehmen bei der Migration von Oracle Forms zu ADF.

Performance-Falle beim Logging: die Zeitstempel-Formatierung

Jürgen Lampe, A:gon Solutions GmbH

Ein Beispiel aus dem Projektalltag zeigt, wie mit relativ geringen Kosten und kleinen Modifikationen einer sich abzeichnenden Überlast vorgebeugt werden kann. Beim Profiling wurden Laufzeit und Allokationsverhalten gleichwertig untersucht, um kritische Code-Bereiche zu identifizieren. Die Erfahrung, dass das Einsparen von Objekt-Erzeugungen die Gesamtleistung überproportional verbessern kann, wurde erneut bestätigt.

Irgendwann ist jede erfolgreiche Applikation so gewachsen, dass ihre Performance durch zunehmende Nutzung und Funktionserweiterungen an Grenzen stößt. Wenn die relativ günstigen Möglichkeiten zum Upgrade der Hardware ausgereizt sind, bleibt nichts anderes übrig, als das gesamte System einer kritischen Schwachstellen-Analyse zu unterziehen.

In dem hier betrachteten Fall handelte es sich um einen großen kommerziellen Applikationsserver mit rund hundert Anwendungen, der bereits auf einer sehr elaborierten Hardware lief. Wie so oft hatten sich die Performance-Probleme schleichend entwickelt. Updates mit neuen Funktionen brachten zusätzliche Belastungen, andererseits konnten durch Änderungen in der Konfiguration auch immer wieder Verbesserungen erreicht werden, ohne allerdings den langfristigen Trend zu höherer Systembelastung dauerhaft zu brechen.

Bei Beginn der Untersuchungen war bereits bekannt, dass die sich abzeichnenden Probleme mit hoher Wahrscheinlichkeit auch durch das Logging verursacht wurden. Indikatoren für diese Vermutung waren:

- Eine beobachtete und mit der Serverlast korrelierende Verzögerung bei der Ausgabe der Log-Ereignisse
- Die Tatsache, dass Änderungen in der Verwaltung der Log-Dateien (schnelleres Filesystem) bereits eine Verbesserung gebracht hatten

Aufgrund rechtlicher Bedingungen, die eine lückenlose Nachvollziehbarkeit aller relevanten Aktionen erfordern, müssen im laufenden Produktionsbetrieb sehr umfangreiche Logs geschrieben werden. Das für den Debug-Modus bekannte Anti-Pattern, das darauf hinweist, „log.debug(…)"-Anweisungen in ein „if(log.debugEnabled()) {log.debug(…);}" einzuschließen, war also nicht anwendbar, weil es bereits beachtet wurde und die Last durch Info-Level-Ausgaben verursacht wird, die nicht abgeschaltet werden dürfen.

Performance-Messung

Es ist ein bewährter Grundsatz, Code-Optimierungen nur auf der Basis von Messungen vorzunehmen und den Erfolg ebenfalls durch Messungen zu verifizieren. Das ist leichter gesagt als getan. Die heute üblichen Prozessoren mit mehreren Kernen, Berechnungs-Pipelines und hierarchischen Cache-Strukturen machen es nahezu unmöglich, reproduzierbare Daten zu gewinnen. Java-Entwickler sehen sich einer weiteren Schwierigkeit gegenüber, die durch die Just-in-time (JIT) kompilierenden virtuellen Maschinen (VM) verursacht wird. Die bekannte HotSpot-VM führt außerdem Optimierungen wie Inlining (Ersetzung durch Einkopieren) von Methoden-Aufrufen oder Loop-Unrolling auf der Basis statistischer Daten aus, die nicht nur zu stark variierenden Ausführungszeiten führen, sondern auch Profiler bei der Datengewinnung beeinträchtigen, da beispielsweise einkopierte Methoden-Aufrufe praktisch unsichtbar werden.

Weitere Probleme ergeben sich, wenn – wie häufig und auch im vorliegenden Fall – das Profiling nicht direkt in der Produktionsumgebung möglich ist. Ein erster Ansatz kann dann darin bestehen, auf dem Testsystem die JIT-Funktion zu deaktivieren, um auf diese Weise wenigstens die Aufruf-Hotspots zu identifizieren. Mit etwas Glück liefert eine anschließende Code-Inspektion der gefundenen Bereiche Ansatzpunkte für Optimierungen. Man sollte diesen Weg aber nur dann weiterverfolgen, wenn die erkannten Defizite wirklich gravierend sind und prinzipiell nicht durch die VM behoben werden können. Ansonsten besteht die große Gefahr, Zeit und Kraft in Arbeiten zu stecken, deren Auswirkungen minimal sind – oder gleich null, wenn nämlich im Produktionsbetrieb eine gleichwertige Beschleunigung durch die JIT-VM bereits erreicht wird.

Im vorliegenden Fall wiesen die durch Profiling gewonnenen Daten auf einen Schwerpunkt bei der Aufbereitung der Log-Ereignisse hin, insbesondere bei der Formatierung, ohne jedoch klare Favoriten für die weitere Analyse auszuweisen.

Objekt-Erzeugung analysieren

In dieser Situation half es, sich an Erfahrungen mit früheren Anwendungen, die unter Speicherplatznot litten, zu erinnern. Als überraschendes Ergebnis hatte es sich dabei immer wieder gezeigt, dass Verbesserungen der Speicherökonomie gleichzeitig das Laufzeitverhalten verbessern. Das ist ein gewisser Widerspruch zu der einst verbreiteten Überzeugung, dass sich Speicherbedarf und Laufzeit antagonistisch verhalten und somit die Verbesserung des einen auf Kosten des anderen geht. Zumindest für Java-Programme gilt diese Regel nicht, was leicht erklärbar ist. Erstens kostet die Objekt-Erzeugung Lauf-

zeit und ein Objekt kommt selten allein; oft müssen weitere (Member-)Objekte erzeugt werden. Zweitens bedeuten mehr Objekte auch mehr Arbeit für den Garbage-Collector (GC) – schon während ihrer Lebenszeit sind mehr erreichbare Objekte zu traversieren. Gleichzeitig wird der Heap schneller gefüllt, was häufigere GC-Aktivitäten verursacht.

Eine erfolgversprechende Strategie besteht deshalb darin, Hotspots der Objekt-Erzeugung zu suchen. Das ist wesentlich einfacher als die oben beschriebene direkte Laufzeitanalyse, weil Hardware und VM praktisch keinen Einfluss auf die Objekt-Erzeugung nehmen. Sollten irgendwelche kurzlebigen Objekte tatsächlich im Stack angelegt werden und uns entgehen, so wäre das eine begrüßenswerte Optimierung, aber für diese Analyse unerheblich.

Ein besonderes Augenmerk haben bei dieser Analyse String-Objekte verdient. Die bequeme Handhabung von Zeichenketten ist einer der großen Vorzüge von Java, hat andererseits jedoch ihren Preis. Die Erzeugung von Strings erfordert, außer bei Teilstrings, jeweils die Allokation eines internen „char"-Arrays sowie das Kopieren der Zeichen in dieses Feld. Letzteres ist eine Operation, deren Kosten mit der Länge des Strings zunehmen. Unglücklicherweise enthalten zahllose Methoden-Signaturen String-Parameter, obwohl „CharSequence" als Typ vollkommen ausreichend wäre.

Formatierung

Um auf das genannte Problem zurückzukommen – auch die Durchsicht der Allokations-Profile deutete auf hohe Last beim Formatieren des Zeitstempels der „LoggingEvents" innerhalb des verwendeten Log4j-Frameworks hin [1]. Wobei es hier nicht um eine bestimmte Software geht, sondern darum zu zeigen, welche Auswirkungen kleine, scheinbar belanglose Unterschiede haben können.

Um die Vermutung zu bestätigen, wurde für die Formatierung des Zeitstempels spezieller Code benutzt. Die Installation eines eigenen Konverters ist recht einfach. Listing 1 zeigt die beiden benötigten Klassen „FastDatePatternParser" und „FastDatePatternLayout", die fast nur aus Verwaltungsaktionen bestehen. Die eigentliche Funktion verbirgt sich ganz unten in der Methode „convert".

```
public class FastDatePatternLayout extends PatternLayout {
 public FastDatePatternLayout() {
  this(DEFAULT_CONVERSION_PATTERN);
 }

 public FastDatePatternLayout(String pattern) {
  super(pattern);
 }

 @Override
 public PatternParser createPatternParser(String pattern) {
  if (pattern == null) {
   return new FastDatePatternParser(DEFAULT_CONVERSION_PATTERN);
  } else {
   return new FastDatePatternParser(pattern);
  }
 }
}
////////////////////////////////////////////////////////////
public class FastDatePatternParser extends PatternParser {
 public FastDatePatternParser(String pattern) {
  super(pattern);
 }
 @Override
 protected void finalizeConverter(char c) {
 if (c == ,d') {
  String dateFormatStr= "yyyy-MM-dd HH:mm:ss,SSS";
  String dOpt= extractOption();
  if ("ABSOLUTE".equals(dOpt)) {
   dateFormatStr= "HH:mm:ss,SSS";
  } else if ("DATE".equals(dOpt)) {
   dateFormatStr= "dd MMM yyyy HH:mm:ss,SSS";
  } else if ("ISO8601".equals(dOpt)) {
  } else if (dOpt != null) {
   dateFormatStr= dOpt;
  }
  PatternConverter pc= new FastDatePatternConverter(formattingInfo, dateFormatStr);
  currentLiteral.setLength(0);
  addConverter(pc);
 } else {
  super.finalizeConverter(c);
 }
 }
 private static class FastDatePatternConverter extends PatternConverter {
  final private String format;

  FastDatePatternConverter(FormattingInfo formattingInfo, String format) {
   super(formattingInfo);
   this.format= format;
  }
  @Override
  public String convert(LoggingEvent event) {
   return CurrentDateProvider.formatTimeMillis( event.getTimeStamp(), format);
  }
 }
}
```

Listing 1

Damit dieser Konverter verwendet wird, muss er in die Konfiguration aufgenommen werden, zum Beispiel durch die folgenden Zeilen in der Datei „log4j.xml" innerhalb eines Appender-Elements (siehe Listing 2).

Das aktuelle Datum

Wenn man die Aufbereitung eines Zeitstempels betrachtet, ist leicht erkennbar, dass ein relativ großer Aufwand getrieben werden muss, um aus dem Millisekunden-Wert das immer wieder gleiche Tagesda-

```
<layout class="de.agon.util.log.FastDatePatternLayout">
        <param name="ConversionPattern" value="%d - %m%n"/>
</layout>
```

Listing 2

tum zu berechnen. Deshalb liegt es nahe, zunächst dort anzusetzen. Im vorliegenden Fall existierte mit der Klasse „CurrentDateProvider" bereits eine Lösung, da auch an anderen Stellen das aktuelle Datum als Zeichenkette, mal mit und mal ohne Uhrzeit, häufig benötigt wird.

Listing 3 zeigt die hier interessierenden Teile des Codes. Für den aktuellen Tag wird jeweils eine Instanz angelegt, die die Millisekunden-Werte für den Tagesbeginn („intervalStart") und das Tagesende („intervalEnd" = Tagesbeginn des Folgetags) sowie die numerischen Zeichen des Datums („dateText") enthält. Daneben wird aus der Tageslänge noch ein Indikator für die beiden Tage der Sommer-Winterzeit-Umschaltung bestimmt („isDSTswitchDay" mit den möglichen Werten „-1", „0" und „+1").

Die Formatierung des Zeitstempels reduziert sich so auf das Kopieren des Datums. Lediglich die Uhrzeit muss noch jeweils neu konvertiert werden. Das leisten wenige Ganzzahl-Operationen. Zugunsten der Performance wurden die möglichen Formate auf rein numerische eingeschränkt. Außerdem muss der Formatstring genauso viele Zeichen enthalten wie das erwartete Ergebnis. Für den Einsatz beim Logging ergab sich daraus kein Hindernis. Die Methode ist wiedereintrittsfähig (re-entrant) und erzeugt beim Aufruf nur ein einziges Objekt – das Ergebnis-Array.

Eine weitere Verbesserung für das Logging wäre eventuell dadurch zu erreichen, dass „CurrentDateProvider" um eine Methode ohne Format-Parameter ergänzt wird, die nur das im Log benötigte Format liefert. Anmerkung: In der Praxis wird dies fast nie geändert, weil es Änderungen an den Skripten zur Auswertung etc. zur Folge hätte.

Allerdings kann die Anzahl der erzeugten Objekte dadurch nicht mehr reduziert werden und die zu erwartende minimale Beschleunigung dürfte den Aufwand kaum rechtfertigen. Bei einer Neuimplementierung sollte dieser Ansatz aber in Erwägung gezogen werden.

Über den hier gezeigten Code hinaus enthält diese Klasse noch eine Reihe von Convenience-Methoden, die beispielsweise Instanzen des aktuellen Datums in verschiedenen Formaten aus dem Cache liefern. Außerdem wird beim Tageswechsel jeweils eine neue Instanz angelegt – die Implementierung dieser Aufgabe ist unproblematisch, sodass hier auf ihre Darstellung verzichtet wird.

Ergebnis

Bei der Bewertung des Ergebnisses sind die Ausgangsbedingungen in Betracht zu ziehen. Da sie nach Erfahrungen des Autors nicht untypisch sind, seien sie kurz beschrieben. Die Entwicklung einschließlich der Entwickler- und Integrationstests erfolgt auf Standard-PCs. In Produktion läuft die Anwendung auf gänzlich anderen Systemen bei einem externen Hoster. Die Überwachung der Leistung erfolgt anhand regelmäßig gelieferter und vertraglich vereinbarter Auswertungen der Logs. Daraus lassen sich Hinweise auf kritische Punkte gewinnen, ein zielgerichtetes Profiling unter Produktionsbedingungen ist jedoch unmöglich. Natürlich sollen sich eventuell abzeichnende Probleme so rechtzeitig beseitigt werden, dass eine Eskalation, die andererseits wiederum die Voraussetzung dafür wäre, den Hoster mit dem Profiling zu beauftragen, vermieden wird.

In dieser Lage bleibt nur, auf der Basis von Erfahrungen und Tests auf den Entwicklungs- beziehungsweise Test-Systemen nach Verbesserungen zu suchen. Mit dieser Strategie wurde das beschriebene Logging-Problem angegangen. Die vorgestellten Veränderungen zeigten sowohl im Performance- als auch im Speicher-Profiling sichtbare Ergebnisse und wurden demzufolge in die Produktion übernommen. Solch ein Schritt ist immer mit einem gewissen Risiko verbunden. Allerdings war angesichts der Tatsache, dass die Anzahlen der auszuführenden Bytecode-Instruktionen und der erzeugten Objekte verringert wurden, zumindest keine Verschlechterung zu erwarten. Tatsächlich sind die Auswirkungen auf die Gesamtanwendung noch erfreulicher als erwartet. Die Serverlast hat sich messbar reduziert und Verzögerungen zwischen „LoggingEvent"-Generierung und der Protokollierung treten so gut wie nicht mehr auf. Mittlerweile ist dieser Fix über ein Jahr erfolgreich und völlig unauffällig im Einsatz.

Damit hat sich erneut gezeigt, dass Sparsamkeit beim Erzeugen von Objekten einen deutlich über die unmittelbar betroffenen Programmteile hinausgehenden positiven Effekt hat. Die Verschlechterung der Code-Dichte durch angepasstes Ausprogrammieren von eigentlich in Bibliotheks-Methoden vorhandenen Funktionen konnte dabei auf einen sehr kleinen und gut austestbaren Bereich beschränkt werden („CurrentDateProvider"). Tatsächlich war dieser Code ja sogar schon vorhanden.

Nachtrag

Das bisher beschriebene Vorgehen zeigt die Arbeit im Projektalltag, wo es vorrangig darum geht, ein gestecktes Ziel mit beschränktem Aufwand zu erreichen. Das ist gelungen, ohne dabei wissenschaftliche Standards bei Analyse und Ergebnisbewertung einzuhalten.

Bei der Vorbereitung dieses Artikels wurde deshalb versucht, die erwähnten Erfahrungen und Heuristiken etwas besser zu begründen und den seinerzeit erzeugten Code kritisch zu inspizieren. Die Ergebnisse werden in diesem zweiten Teil präsentiert. Diese Form ist bewusst gewählt worden, um hier die Teile der Analyse nachzuliefern, für die normalerweise leider meist die Zeit fehlt.

Im Folgenden werden vier Varianten der Formatierung verglichen. Als Ausgangspunkt mit dem Namen „NORMAL" dient die Implementierung, die jeden Zeitstempel mithilfe von „java.text.SimpleDateFormat" aufbereitet. Das ist die Log4j-Standard-Implementierung, wenn im „ConversionPattern" explizit ein Muster für die Zeitstempel-Formatierung angegeben wird, „%d" also nicht allein steht, sondern etwa in der Form „%d{dd.MM.yyyy}".

Unter dem Namen „FAST" wird die oben beschriebene Variante betrachtet. Aufgrund der erläuterten speziellen Implementierung ist es dabei unerheblich, in welcher Form das Format spezifiziert wird.

Bei genauem Lesen der Log4j-API-Dokumentation findet sich ein wichtiger Hinweis: „For better results it is recommended to use the log4j date formatters. These can be specified using one of the strings

```
/** Tagesgrenzen */
private final long intervalStart, intervalEnd;
/** Cache fuer das aktuelle Datum in der Form ddMMyyyy */
private final char[] dateText = new char[8];

protected final char[] formatToCharArrayIntern(long time, String format) {
 if (time < intervalEnd && time >= intervalStart) {
  int millisecs= (int) (time - intervalStart);
  if (isDSTswitchDay != 0) {
   if (isDSTswitchDay < 0 && millisecs > 7200000L) {
       millisecs+= 3600000L;
   } else if (isDSTswitchDay > 0 && millisecs > 10800000L) {
    millisecs-= 3600000L;
   }
  }
  char[] charArray= format.toCharArray();
  int yearCount= 0;
  int monthCount= 0;
  int dayCount= 0;
  int hours= millisecs / 3600000;
  millisecs%= 3600000;
  int minutes= millisecs / 60000;
  millisecs%= 60000;
  int secs= millisecs / 1000;
  int millis= millisecs % 1000;

  char[] dateAsText= dateText;
  for (int i= 0, len= charArray.length; i < len; i++) {
   switch (charArray[i]) {
    case 'y':
     if (yearCount < 2) {
      charArray[i]= dateAsText[6 + yearCount++];
     } else if (yearCount++ == 2) {
      charArray[i - 2]= dateAsText[4];
      charArray[i - 1]= dateAsText[5];
      charArray[i]= dateAsText[6];
     } else {
      charArray[i]= dateAsText[7];
     }
     break;
    case 'M':
     charArray[i]= dateAsText[2 + monthCount++];
     break;
    case 'd':
     charArray[i]= dateAsText[0 + dayCount++];
     break;
    case 'H':
     charArray[i]= (char) ('0' + hours / 10);
     hours= hours % 10 * 10;
     break;
    case 'm':
     charArray[i]= (char) ('0' + minutes / 10);
     minutes= minutes % 10 * 10;
     break;
    case 's':
     charArray[i]= (char) ('0' + secs / 10);
     secs= secs % 10 * 10;
     break;
    case 'S':
     charArray[i]= (char) ('0' + millis / 100);
     millis= millis % 100 * 10;
     break;
    default:
   }
  }
  return charArray;
 } else return null;
}
```

Listing 3

„ABSOLUTE", „DATE" and „ISO8601"…" [2]. Da das ISO8601-Format dem teilweise vor der Optimierung verwendeten entspricht (es ist das Default-Muster, wenn „%d" allein steht), wurde die entsprechende Konvertierung als ISO8601 in den Vergleich aufgenommen. Der zugehörige Konverter hält nicht nur den Datumsteil des formatierten Zeitstempels in einem Cache vom Typ „char[ ]", sondern alle Zeichen außer den Sekundenbruchteilen, die bei jedem Aufruf neu konvertiert werden. Zusammen mit den Zeichen steht der dazugehörige Sekundenwert im Cache, um entscheiden zu können, wann eine neue Konvertierung des vorgehaltenen Teils notwendig ist.

Diese Arbeitsweise ist für die Formatierung sehr schnell aufeinanderfolgender Zeitstempel günstig, verliert ihre Vorteile aber, wenn die Zeitstempel ungeordnet sind oder nur wenige pro Sekunde vorliegen. Nebenbei bemerkt, ergab ein kleine (nichtrepräsentative) Umfrage, dass unter den Log4j-Anwendern vielen unbekannt ist, dass zwischen den Formaten „%d" beziehungsweise „%d{ISO8601}" einerseits und „%d{yyyy-MM-dd HH:mm:ss,SSS}" andererseits ein erheblicher Performance-Unterschied besteht. Anmerkung: Im vorliegenden Projekt kamen diese Formen ebenfalls vermischt zur Anwendung. Bisweilen ist Optimierung schon durch die Auswahl der richtigen Schreibweise möglich.

Die abschließende vierte Variante „FAST2" kann wegen Zugriffs-Restriktionen nicht vollständig realisiert werden. Mit ihr soll der Einfluss untersucht werden, den das Einsparen von String-Objekten hat. Dafür wird ein weiterer „PatternConverter" definiert, der diesmal die „format"-Methode von „org.apache.log4j.helpers.PatternConverter" überschreibt. Im Wesentlichen wird in dieser Methode die in Listing 1 zu findende „convert"-Methode aufgerufen und deren Ergebnis-String in einen „StringBuffer" kopiert. Listing 4 zeigt die geänderte Methode. Die erste auskommentierte Zeile wurde durch die nachfolgende neue ersetzt. Die zweite auskommentierte Zeile müsste angepasst werden, wird für den Test aber nicht benötigt.

Die aufgerufene statische Methode „CurrentDateProvider.formatTimeMillisToChars" kapselt den Code für das Ermitteln der aktuellen Instanz des Providers und ruft deren „formatToCharArrayIntern"-

Methode (siehe Listing 3) auf. Durch das direkte Kopieren des „char"-Arrays in den „StringBuffer" wird pro Aufruf ein String eingespart. Anmerkung: Diese Optimierung ist generell anwendbar, praktisch aber leider mit einigem Aufwand verbunden, da die für die Konfiguration des „PatternConverter" verwendete Klasse „FormattingInfo" „package-private" ist.

Einfache Messung

Auf die Problematik der Performance-Messung mit modernen Prozessoren und VM wurde bereits hingewiesen. Alle Messungen sind auf einem PC mit Intel 7-2600 CPU/3.40 GHz, 16 GB Hauptspeicher und HotSpot-64-Bit-Server-VM (1.6.0_31) ausgeführt. Tabelle 1 zeigt repräsentative Ergebnisse für jeweils vier aufeinanderfolgende Formatierungen von „LogEvents", gemessen mit „System.nanoTime()". Bei den ersten drei Aufrufen wird der gleiche Event verwendet, beim vierten ein solcher mit einem um mehr als eine Sekunde abweichenden Zeitstempel.

Die Ergebnisse sind wenig überraschend. Die Dauer des ersten Aufrufs wird durch das Laden und Initialisieren der Klassen bestimmt. Da die Varianten von oben nach unten und in einem Lauf ausgeführt werden, profitieren die letzten natürlich davon, dass die meisten benötigten Klassen bereits geladen sind. Klar erkennbar ist die längere Laufzeit von NORMAL, während die anderen Varianten relativ dicht beieinander liegen. Weitere Erkenntnisse sind:

- Die Laufzeiten von NORMAL variieren wesentlich stärker als die der anderen Verfahren.
- Wenn ISO8601 auf den Cache zurückgreifen kann, ist die Laufzeit mit FAST vergleichbar (Läufe 2 und 3).
- Der vierte Aufruf von ISO8601 (Ereignis mit Zeitstempel, der nicht mithilfe des Cache formatiert werden kann) reicht erwartungsgemäß in der Dauer an NORMAL heran.
- FAST2 ist fast immer (ein wenig) schneller als FAST.

Wie zu erwarten war, zeigen die Messungen eine große Streuung, wobei, von einzelnen Ausreißern abgesehen, die aufgeführten Ergebnisse klar erkennbar sind.

```
@Override
public void format(StringBuffer sbuf, LoggingEvent e) {
// String s = convert(e); wird ersetzt durch
 char[] s = CurrentDateProvider.formatTimeMillisToChars(e.getTimeStamp(), format);
 if (s == null) {
  if (0 < min) {
   spacePad(sbuf, min);
  }
  return;
 }
 int len = s.length;
 if (len > max) {
// sbuf.append(s.substring(len - max));
 } else if (len < min) {
  if (leftAlign) {
   sbuf.append(s);
   spacePad(sbuf, min - len);
  } else {
   spacePad(sbuf, min - len);
   sbuf.append(s);
  }
 } else {
  sbuf.append(s);
 }
}
```

Listing 4

```
version instances  B ns linear runtime
  NORMAL     9,00 504 903 ============================
    FAST     7,00 408 431 ==============
  ISO860     9,00 504 536 =================
   FAST2     5,00 312 425 ==============
```

Listing 5

Durchlauf / Variante	1	2	3	4
NORMAL	417461 ns	29280 ns	21130 ns	28977 ns
FAST	897102 ns	12075 ns	9961 ns	10263 ns
ISO8601	55541 ns	14187 ns	10565 ns	20828 ns
FAST2	20224 ns	9659 ns	9659 ns	9357 ns

Tabelle 1

Benchmark

Um einen Eindruck vom Einfluss des JIT-Compiling der VM zu bekommen, werden die Formatierungen mit Googles Benchmark-Framework „Caliper" [3] untersucht. Das verfügbare Release 0.5 ist laut eigener Homepage noch „a little rough around the edges, but we have already found it quite useful". Implementiert wird ein statistischer Ansatz, der sich an Experimenten mit komplexen biologischen Systemen orientiert.

Ein Benchmark kann einfach als Erweiterung der Klasse „SimpleBenchmark" programmiert werden. Die Struktur gleicht der von JUnit-Tests, nur dass die Methoden-Namen mit „time" beginnen. Listing 6 zeigt den Code des Format-Benchmarks. Neben der Laufzeit kann der verbrauchte Speicher protokolliert werden. „Caliper" bietet verschiedene Möglichkeiten, die Ergebnisse darzustellen, darunter auch einen direkten Upload auf einen Web-Server. Im einfachsten Fall erhält man die folgende Ausgabe auf die Konsole (siehe Listing 5).

Die Resultate korrespondieren gut mit denen der Einzelmessungen. Nebenbei

```
private PatternLayout layout;
private LoggingEvent[] events;

@Param Version version;
public enum Version {NORMAL, FAST, ISO860, FAST2}

@Override
protected void setUp() throws Exception {
 Logger logger= LogManager.getLogger(«FormatLogger»);
 switch (version) {
 case NORMAL:
  layout= new PatternLayout(«%d{yyyy-MM-dd HH:mm:ss,SSS} [%t] - %m%n»);
  break;
 case FAST:
  layout= new FastDatePatternLayout(«%d [%t] - %m%n»);
  break;
 case ISO860:
  layout= new PatternLayout(«%d [%t] - %m%n»);
  break;
 case FAST2:
  layout= new FastDatePatternLayout2(«%d [%t] - %m%n»);
  break;
 }
 long ts= System.currentTimeMillis() - 8 * 3600L;
 events = new LoggingEvent[10000];
 for (int i= 0; i < events.length; i++) {
  events[i]= new LoggingEvent(«fgnOf», logger, ts+600*i, Level.INFO, «message», null);
 }
}

public String timePatternLayoutFormat(int reps) {
 String dummy= null;
 for (int i= 0; i < reps; i++) {
   dummy= layout.format(events[i % 10000]);
 }
 return dummy;
}
```

Listing 6

sieht man das erhebliche Beschleunigungspotential der JIT-Technologie. Bei der Interpretation muss man aber beachten, dass die wiederholte Ausführung in Schleifen zu einer besonders aggressiven Optimierung führt. Im tatsächlichen Anwendungsszenario ist die Formatierung jedoch nur ein Schritt in einer umfassenderen Schleife, sodass sich durchaus andere Hotspots ergeben können. Wichtig ist die Erkenntnis, dass keine der untersuchten Varianten Code enthält, der die VM bei ihren Optimierungen behindert.

Fazit

Die zwischen den Varianten ISO8601 und FAST beobachteten Laufzeit-Unterschiede sind – auch wenn man berücksichtigt, dass daneben noch teilweise NORMAL verwendet wurde, – zu gering, um für sich allein die beim Gesamtsystem erreichten Verbesserungen zu erklären. Das gelingt nur, wenn man den Speicherverbrauch in die Betrachtung mit einbezieht. Wie im Benchmark-Versuch dokumentiert, erzeugt ein Aufruf der FAST-Formatierung zwei Objekte weniger (sieben statt neun) und verbraucht dadurch fast 20 Prozent weniger Platz (408 statt 504 Byte). ISO8601 und NORMAL unterscheiden sich in dieser Beziehung nicht voneinander. Interessant wäre nun natürlich die Frage, ob beim Einsatz der FAST2-Variante in der Produktion eine weitere Verbesserung beobachtet werden kann.

Obwohl die Auswirkungen des Speichersparens deutlich spürbar sind, ist das Ausmaß quantitativ kaum exakt erfassbar, denn neben der Verringerung der GC-Aktivität wird unter Umständen durch größere Datenlokalität auch der Effekt des Caching verbessert. Wegen der zahlreichen Einfluss-Faktoren lassen sich solche Effekte jedoch nicht isolieren. Offen bleibt gleichfalls, ob sich die sparsamere Verwendung des Speichers gerade (oder nur) bei Systemen so positiv bemerkbar macht, die vorher bereits eine gewisse Lastgrenze überschritten hatten.

Letztlich haben die zusätzlichen Untersuchungen keine wirklich neuen Fakten an den Tag gebracht. Das sollte niemanden davon abhalten, solche Messungen vorzunehmen. Das Performance-Modell von Prozessoren und VM macht Vorhersagen zunehmend schwieriger. Umso wichtiger ist es, die bewährten Heuristiken immer wieder zu überprüfen, um unangenehme Überraschungen möglichst vermeiden zu können. Im vorliegenden Fall darf man nicht außer Acht lassen, dass Log4j eine Bibliothek mit bereits weitgehend optimiertem Code ist. Jedes andere als das vorgestellte Resultat wäre daher eine echte Überraschung gewesen.

Zusammenfassung

Performance-Verbesserungen sind eine wichtige und schwierige Aufgabe. In ordentlich gebauten Anwendungssystemen ist es oft schwierig, allein durch Laufzeit-Profiling kritische Codestellen zu identifizieren. In diesen Fällen ist die Suche nach Allokations-Hotspots eine vielversprechende Alternative. Die vorliegenden Erfahrungen sprechen dafür, dass insbesondere beim Umgang mit Strings Sorgfalt sinnvoll ist.

Quellen

[1] Apache log4j, http://logging.apache.org/log4j
[2] http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html
[3] Caliper – Microbenchmarking framework for Java: https://code.google.com/p/caliper/

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.

Unbekannte Kostbarkeiten des SDK Heute: Web-Services

Bernd Müller, Ostfalia

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

Nachdem Gartner und andere Analysten einen Web-Service-Hype für die Dekadenmitte vorhergesagt hatten und dieser auch tatsächlich eintrat, sah sich Sun veranlasst, eine Web-Service-Implementierung in das Java SDK zu integrieren, die wir hier kurz vorstellen wollen.

Web-Services und das SDK

Gartner veröffentlichte im Jahr 2001 eine Analyse, die für die Jahre 2005 und folgende eine starke Verbreitung von Web-Services und die Etablierung serviceorientierter Architekturen prognostizierte [1]. Der entsprechende JSR 224 (Java API for XML-Based Web Services (JAX-WS) 2.0) [2] wurde 2006 verabschiedet. Die Referenz-Implementierung [3] stammte von Sun. Diese wurde in das SDK 6 integriert und kann damit ohne weitere Bibliotheken oder Server wie etwa GlassFish oder JBoss-AS als einfache Web-Service-Implementierung ausschließlich mit dem SDK betrieben werden. Wir gehen auf WSDL, SOAP und weitere Themen nicht ein und beschränken uns auf das unbedingt Nötige.

Das obligatorische „Hello World"

Wir entwickeln im Folgenden die typische Hello-World-Anwendung, um sie als Web-Service zu publizieren. Danach bauen wir einen Client für diesen Service. Die Klasse „HelloWorld", die nachfolgend wiedergegeben ist, wird mit „@WebService" annotiert. Diese Annotation ist im JSR 181 [4] definiert und veröffentlicht die Klasse als Web-Service-Endpoint, sodass alle öffentlichen Methoden als Web-Service verfügbar sind. Die Annotation einer Methode mit „@WebMethod" ist damit nicht notwendig (siehe Listing 1).

```
package de.pdbm;
import javax.jws.WebService;
@WebService
public class HelloWorld {
  public String helloWorld(String whom) {
    return "\"hello world\" from " + whom;
  }
}
```

Listing 1

Um den Web-Service veröffentlichen zu können, ist ein Web-Service-Endpoint erforderlich. Dieser ist durch die Klasse „Endpoint" realisiert, sodass zur Vervollständigung unseres Beispiels nur ein einziger Aufruf der „publish()"-Methode dieser Klasse genügt, wie Listing 2 zeigt. Damit ist offensichtlich, dass das SDK auch einen einfachen HTTP-Server enthalten muss, um das Binding für das verwendete URI-Schema bereitzustellen.

Wird die Klasse als Main-Klasse gestartet, so liefern HTTP-Get-Aufrufe für „http://localhost:8080/HelloWorld" und „http://localhost:8080/HelloWorld?wsdl" allgemeine Informationen zum Endpunkt beziehungsweise das entsprechende WSDL-Dokument zurück. Um den Web-Service tatsächlich deployen und nutzen zu können, müssen noch weitere Klassen generiert werden. Hierzu verwendet man das Werkzeug „wsgen", das sich im bin-Verzeichnis des SDK befindet. Wir zitieren dazu die Dokumentation des Werkzeugs: „The wsgen tool generates JAX-WS portable artifacts used in JAX-WS web services. The tool reads a web service endpoint implementation class (SEI) and generates all the required artifacts for web service deployment, and invocation."

Zurück zum Beispiel: Der Aufruf von „wsgen -cp bin -d bin -s generate de.pdbm.HelloWorld" erzeugt für die Web-Service-Klasse „de.pdbm.HelloWorld" zwei neue Klassen, deren Quell-Code im Verzeichnis „generate" und deren Compilate im Verzeichnis „bin" abgelegt werden.

Auch für das Erzeugen der benötigten Client-Artefakte enthält das SDK die entsprechende Unterstützung. Das Werkzeug „wsimport" des SDK erzeugt die für einen Client benötigten Artefakte. Wir zitieren auch hier die Dokumentation: „The wsimport tool generates JAX-WS portable artifacts, such as:

- Service Endpoint Interface (SEI)
- Service
- Exception class mapped from wsdl:fault (if any)
- Async Reponse Bean derived from response wsdl:message (if any)
- JAXB generated value types (mapped java classes from schema types)"

```
import javax.xml.ws.Endpoint;
public class HelloWorldServer {
  public static void main(String[] args) {
    Endpoint.publish("http://localhost:8080/HelloWorld", new HelloWorld());
  }
}
```

Listing 2

Der Aufruf erfolgt durch „wsimport -keep -p de.pdbm.client -d bin -s generate http://localhost:8080/HelloWorld?wsd". Auch hier wird eine Reihe von Klassen erzeugt, die im Verzeichnis „generate" und hier wiederum im Package „de.pdbm.client" abgelegt ist. Als Grundlage der Generierung wird die Service-Beschreibung verwendet, die über den HTTP-Server verfügbar ist. Als letzter Schritt kann nun ein Client entwickelt werden, der die generierten Artefakte verwendet. Die Klasse HelloWorldClient leistet genau dies (siehe Listing 3).

Die verwendeten Klassen „HelloWorldService" und „HelloWorld" wurden zuvor von „wsimport" erzeugt. Wir sind damit am Ende unseres einfachen Beispiels angelangt.

```
package de.pdbm;
import de.pdbm.client.HelloWorldService;
public class HelloWorldClient {
 public static void main(String[] args) {
   HelloWorldService service = new HelloWorldService();
   de.pdbm.client.HelloWorld proxy = service.getHelloWorldPort();
   System.out.println(proxy.helloWorld(„Duke"));
  }
}
```

Listing 3

Zusammenfassung

Mit Version 6 des SDK wurde eine Implementierung von Web-Services in das Java-SE SDK integriert. Damit ist es möglich, einfache Web-Services ohne weitere Bibliotheken und Frameworks zu realisieren. Komplexe Anwendungen auf Basis von Web-Services bleiben natürlich nach wie vor umfassenderen Frameworks wie AXIS2 oder Application-Servern vorbehalten.

Literatur

[1] Gartner: The Hype Is Right: Web Services Will Deliver Immediate Benefits, http://www.gartner.com/id=344028

[2] JSR 224: Java API for XML-Based Web Services (JAX-WS) 2.0, http://jcp.org/en/jsr/detail?id=224.

[3] JAX-WS Reference Implementation: http://jax-ws.java.net

[4] JSR 181: Web Services Metadata for the Java Platform 2.0, http://jcp.org/en/jsr/detail?id=181.

Bernd Müller
bernd.mueller@ostfalia.de

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

Foto: IBM

Integration von Java via JNI in traditionelle Cobol-Batch-Anwendungen

Marc Bauer und Martin Kiefer, IBM Deutschland GmbH

In den letzten Monaten ist die Anzahl an Projekten zur Modernisierung von Altanwendungen beträchtlich gestiegen. Entwickler, die sowohl die traditionelle als auch die Java-Welt kennen, sind gefragt.

Für den neuen Trend gibt es vor allem zwei Gründe: Erstens gehen mehr und mehr Experten für die älteren Programmiersprachen wie beispielsweise Cobol in den Ruhestand und zweitens ist Java schon lange erwachsen und robust genug, um auch in sehr kritischen Einsatzgebieten angewendet zu werden. Die Technologie, um Altsysteme mit neuen Java-Anwendungen zu erweitern und abzulösen, steht bereit. Der Artikel zeigt anhand eines Beispiels, wie leicht Cobol in Batch-Anwendungen um die Funktionalität aus Java-Klassen erweitert werden kann.

Batch-Anwendungen und die Programmiersprache Cobol sind seit vielen Jahren ein fester Bestandteil der Anwendungsentwicklung. Sie charakterisieren sich dadurch, dass sie ohne Benutzer-Interaktion ablaufen und eine sehr große Menge an Eingabedaten verarbeiten. Beispiele wären die monatliche Kreditkartenabrechnung, das Erstellen von Kontoauszügen oder Rechnungsschreiben.

Die Abkürzung „Cobol" steht für „Common Business Oriented Language". Die Syntax ist stark an die natürliche Sprache angelehnt und leicht verständlich. Zahlen werden beispielsweise in Fix-Komma-Notation abgelegt. Gefördert durch die fortwährende Abwärtskompatibilität, wuchs die Komplexität der Batch-Anwendungen an. Systeme mit mehreren Millionen Zeilen Cobol-Code und einem Alter von mehr als dreißig Jahren sind keine Seltenheit. Jedoch ist davon auszugehen, dass in den nächsten Jahren ein Großteil der Cobol-Experten den Markt verlassen wird. Mit ihnen verschwindet auch das Wissen über die historisch gewachsenen und hochkomplexen Bestandssysteme. Somit stehen wir heute vor der Frage, wie wir mit diesen Bestandssystemen in Zukunft umgehen sollen.

Die erste Idee ist die Ablösung der Altanwendungen durch neue Java-Programme. Jedoch sind derartige Projekte aufgrund der hohen Komplexität oft mit viel Risiko behaftet. Des Weiteren fehlt in vielen Fällen eine genaue Funktionsbeschreibung der Altanwendung, wodurch die Gefahr steigt, dass im Rahmen der Neuentwicklung wichtige Anforderungen übersehen werden. Außerdem ist das Neuprogrammieren einer schon bestehenden Anwendung nur schwer zu verkaufen, da die Fachabteilungen, die heutzutage oft die Budgetverantwortung haben, kaum unmittelbaren fachlichen Nutzen erkennen werden.

Eine Alternative kann die Entwicklung einer hybriden Anwendung sein, in der neue Module in Java entwickelt und Bestandsmodule aus Cobol weiterverwendet werden. Auf diese Weise würden die Investitionen in die Altanwendungen geschützt und neue Anforderungen schnell und

unter Verwendung vieler Frameworks mit Java ermöglicht werden.

Eine naheliegende Möglichkeit, eine hybride Anwendung zu erstellen, bietet das Java Native Interface (JNI). Es ist ein standardisiertes API, um von nativen Programmen auf Java-Anwendungen und umgekehrt zugreifen zu können. Somit ist es möglich, Java-Objekte im Cobol-Programmcode zu verwenden.

Am folgenden Beispiel wird aufgezeigt, wie die Interaktion von Cobol und Java mithilfe von JNI erfolgen kann. Dabei bereitet das Cobol-Modul Daten beispielsweise für einen Kontoausdruck auf; sie sollen anschließend als PDF ausgegeben werden. Die PDF-Generierung übernimmt ein Java-Modul mithilfe der iText-PDF-Bibliothek. Zunächst schreiben wir eine Klasse „PDF-Creator", die „itextPdf" verwendet und über ihre Methoden alle Funktionen bereitstellt, die von Cobol zum Generieren eines rudimentären Kontoausdrucks benötigt werden.

Wir übergeben im Konstruktor den Namen des Kunden und dessen Kontonummer. Über die Methode „addTransaction" werden Betreff und Wert von Überweisungen übergeben. Wurden alle Überweisungen eingetragen, kann die Methode „writePdf" aufgerufen werden, um eine PDF-Datei mit Name, Kontonummer und aktuellem Datum abzulegen (siehe Listing 1).

Nun soll aus einer Cobol-Anwendung auf diese Java-Klasse zugegriffen werden. Eine Cobol-Anwendung besteht aus mehreren „Divisions", die wiederum „Sections" enthalten können. Diese sind in „"Paragraphs" unterteilt (siehe Listing 2).

Die „Identification Division" enthält Meta-Informationen über das Programm. Sie muss vorhanden sein, spielt aber für die konkrete Verwendung des JNI keine direkte Rolle.

In der „Environment Division" muss ein Eintrag für jede Java-Klasse angelegt sein, die von der Anwendung verwendet wird. Für das Beispiel wird hier unsere Klasse „PDFCreator" angegeben, darüber hinaus benötigen wir die Klasse „jstring", die einem gewöhnlichen Java-String entspricht, sowie die Klasse „java.lang.Exception", um auftretende Exceptions zu behandeln.

In der „Data Division" sind Programmvariablen und Datenstrukturen definiert. Ihre Working-Storage-Section enthält alle Variablen, die im Programm verwendet werden. Auf Java-Objekte wird über Variablen zugegriffen, die entsprechende Objekt-Referenzen enthalten. Im Beispiel werden hier zunächst Variablen für die Klassen „PDFCreator" und „Exception" benötigt.

Es ist in Cobol üblich, Variablen logisch zu Records zusammenzufassen. Im Beispiel enthält hier der Record „CUSTOMER" die Variablen, die für die Verarbeitung von Name und Kontonummer notwendig sind. Der Datentyp „PIC X(30)" entspricht einer Zeichenkette mit 30 Zeichen, „PIC S9(9)" einer vorzeichenbehafteten Ganzzahl mit neun Stellen, der Zusatz „COMP-5" sorgt dafür, dass die Zahl im Zweierkomplement abgelegt wird. Dieser Datentyp ist kompatibel zum Java-Basisdaten-Typ „int". Zusätzlich enthält der Datensatz eine Variable mit einer Objekt-Referenz auf einen String. Sie wird später dem Konstruktor übergeben, der einen String und einen Integer als Argumente erwartet. Analog enthält der Datensatz Transaction-Variablen für den Aufruf von „AddTransaction". Der Datentyp „COMP-1" ist kompatibel zum Java-Basisdaten-Typ „float". In der Linkage Section wird das Copybook für das JNI importiert. Es enthält Definitionen, die für dessen Verwendung benötigt werden.

Zuletzt folgt in jedem Cobol-Programm die Procedure Division, in der die eigentlichen Programmbefehle angegeben werden. Sie enthält üblicherweise Sections mit Programmanweisungen, die die Anwendung gliedern und mithilfe der Perform- oder Goto-Anweisung angesprungen werden können. Im Gegensatz zur Goto-Anweisung erfolgt bei der Perform-Anweisung ein Rücksprung am Ende der Section.

Im Beispiel initialisieren wir zunächst das JNI, sodass mithilfe des JNI-Environment Pointer-Funktionen aufgerufen werden können. Zuerst werden die Variablen aus dem „CUSTOMER"-Record gesetzt. Hat eine Zeichenkette ein „z" als Präfix, wird sie nullterminiert. Dies wird vom Java-Native-Interface vorausgesetzt, um die Cobol-Zeichenkette in einen Java-String umzuwandeln.

Die Section „CREATE-PDFGEN" wird aufgerufen. Sie verwendet die JNI-Funktion „GetStringPlatform", die aus den Daten hinter einem Zeiger auf eine Cobol-Zeichenkette einen Java-String sowie eine Referenz auf diesen erzeugt. Nun wird ein Objekt der Klasse „PDFCreator" erstellt. Hierzu kommt zum Ausführen von Methoden die „INVOKE"-Anweisung zum Einsatz, die auch zur Ausführung von objektorientiertem Cobol verwendet wird. Dem Aufruf wird zuerst die Klasse beziehungsweise das Objekt übergeben, für das eine Methode ausgeführt werden muss. Es folgen die Parameter der Methode. Ist ein Rückgabewert vorgesehen, wird er in die Variable hinter dem Schlüsselwort „Returning" gespeichert. An dieser Stelle rufen wir die Klasse „PDFCreator" mit dem Schlüssel-

```
public class PdfCreator {
	private Document document;
	private Paragraph paragraph;

	public PdfCreator(String name, int accountId) throws FileNotFoundException, DocumentException {
		document = new Document();
		PdfWriter.getInstance(document, new FileOutputStream("OUTPUT.pdf"));
		document.open();
		paragraph = new Paragraph();
		Chunk chunk = new Chunk("Name:" + name + "Customer ID:" + accountId+ "\n");
		paragraph.add(new Chunk(chunk));
		chunk = new Chunk("--Subject----------Value----------\n");
		paragraph.add(new Chunk(chunk));
	}

	public void writePdf() throws DocumentException{
		document.add(paragraph);
		document.close();
	}

	public void addTransaction(String subject, float value) {
		paragraph.add(new Chunk(subject + "	" + String.format(("%.2f"),value) +"$"  + "\n"));
	}
}
```

Listing 1

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ‚ExTrans‘ RECURSIVE.
AUTHOR. martin.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
    Class PdfCreator      is “PdfCreator“
    Class jstring         is “jstring“
    Class jException      is “java.lang.Exception“.

DATA DIVISION.
WORKING-STORAGE SECTION.
01  ex      OBJECT REFERENCE jException.
01  creator OBJECT REFERENCE PdfCreator.
01 CUSTOMER.
    05 NAME        PIC X(30).
    05 ACCOUNT     PIC S9(9) COMP-5.
    05 jNameOBJECT REFERENCE jstring.
01  TRANSACTION.
    05  SUBJECT    PIC X(30).
    05  TVALUE     USAGE COMP-1.
    05  jSubject   OBJECT REFERENCE jstring.

LINKAGE SECTION.
    COPY JNI.

PROCEDURE DIVISION.
*   Get the JNI Environment Pointer
    SET ADDRESS OF JNIEnv TO JNIEnvPtr
    SET ADDRESS OF JNINativeInterface TO JNIEnv
*   Set customer information
    MOVE z“THOMAS J. WATSON“ TO NAME
    MOVE 314159 TO ACCOUNT
    PERFORM CREATE-PDFGEN.

*   Check for an exception
    PERFORM EXCEPTION-CHECK

*   Create transaction input
    MOVE z“Einkauf“ TO SUBJECT
    MOVE 14.99 TO TVALUE

*   Add the transaction
    PERFORM ADD-TRANSACTION
    PERFORM EXCEPTION-CHECK

*   Write the PDF fiel
    PERFORM WRITE-PDF
    PERFORM EXCEPTION-CHECK

*   Free all Java object references
    PERFORM CLEAN-UP
    GO TO OKEND.
* Progam End

* Create the PDFGenerator for current customer
CREATE-PDFGEN.
*    Convert Cobol name String to Java String
    CALL “NewStringPlatform“ USING BY VALUE JNIEnvPtr
    ADDRESS OF NAME ADDRESS OF jName 0
* Create PdfCreator object
    INVOKE PdfCreator New USING BY VALUE jName ACCOUNT
RETURNING creator.

* Convert Cobol subject string and call addTransaction
ADD-TRANSACTION.
    CALL “NewStringPlatform“
    USING BY VALUE JNIEnvPtr ADDRESS OF SUBJECT ADDRESS
OF jSubject 0
    INVOKE creator “addTransaction“ USING BY VALUE jSubject
TVALUE.

* Call the writePdf method
WRITE-PDF.
     INVOKE creator “writePdf“.

* CLEAN-UP for Garbage Colletion
CLEAN-UP.
    CALL DeleteLocalRef USING BY VALUE JNIEnvPtr jName
    CALL DeleteLocalRef USING BY VALUE JNIEnvPtr jSubject
    CALL DeleteLocalRef USING BY VALUE JNIEnvPtr creator.

* Check for an exception
EXCEPTION-CHECK.
    CALL ExceptionOccurred USING BY VALUE JNIEnvPtr RE-
TURNING ex.
    IF ex NOT = null THEN
    CALL ExceptionClear using by VALUE JNIEnvPtr
    DISPLAY “Caught an exception!“
    INVOKE ex “printStackTrace“
    GO TO ERREND
    END-IF.

* Return code 0 and exit.
OKEND.
    DISPLAY “Everything went fine.“
    MOVE 0 TO RETURN-CODE.
    STOP RUN.

* Return code 8 and complain. And exit.
ERREND.
    DISPLAY “An error occured.“
    MOVE 8 TO RETURN-CODE.
    STOP RUN.
```

Listing 2

wort „New“ auf, um eine Objektreferenz für ein Objekt vom Typ „PDFCreator“ zu erhalten und übergeben dem Konstruktor die eben erzeugte Referenz auf den Namen und die Kontonummer.

Im Anschluss wird die Section „EXCEPTION-CHECK“ aufgerufen. Sie prüft mit einer JNI-Funktion, ob die zuletzt aufgerufene Methode eine Exception verursacht hat, die nicht behandelt wurde. Ist dies der Fall, muss mit der JNI-Funktion „ExceptionClear“ die aufgetretene Exception aufgelöst werden, um weiter Java-Komponenten zu verwenden, andernfalls wird die Anwendung unsanft durch das Betriebssystem beendet. Generell empfiehlt es sich, beim Prüfen auf Exceptions nicht zu sparsam zu sein, denn selbst wenn die Methode keine Exception vorsieht, kann eine unerwartete RuntimeException zu demselben Verhalten führen und bei komplexeren Anwendungen die Datenkonsistenz gefährden.

Im nächsten Programmabschnitt werden Variablen für das „TRANSACTION“-Record für eine Überweisung gesetzt. Die Section „ADD-TRANSACTION“ wird aufgerufen, ihre Implementierung ist weitestgehend analog zu „CREATE-PDFGEN“. Einzig der „INVOKE“-Befehl wird hier dazu verwendet, die Methode „addTransaction“ aufzurufen, sodass anstelle der Klasse und „New“ die Referenz auf das „PdfRecord“-Objekt und der Methodenname übergeben werden.

Im weiteren Programmverlauf wird die Section „WRITE-PDF“ aufgerufen, um die PDF-Datei mit der Funktion „writePdf“ zu

schreiben. Bevor das Programm beendet werden kann, verdient die aufgerufene Section „CLEAN-UP“ besondere Aufmerksamkeit. Vielleicht haben Sie sich bis zu dieser Stelle gefragt, wie der Garbage Collector der JVM nicht mehr benötigten Speicher freigeben kann, wenn sich die Objekt-Referenzen im Cobol-Code seiner Überwachung entziehen. Die Antwort lautet: Der Garbage Collector kann es nicht. Standardmäßig werden von den JNI-Funktionen sogenannte „lokale Referenzen“ zurückgegeben. Ihre Lebensdauer endet mit dem aufgerufenen Cobol-Modul. Es ist oftmals jedoch nicht ausreichend, auf das Ende der Lebensdauer der Variablen zu warten. In Batch-Anwendungen könnten für viele zu verarbeitende Datensätze neue Java-Objekte erzeugt werden. Das kann in einer Situation enden, in der der Speicher der JVM vollständig belegt ist und nicht mehr freigegeben werden kann. Das Programm kann nicht mehr weiter ausgeführt werden. Aus diesem Grund kann zwischen der statischen Speicherverwaltung von Cobol und der automatischen Speicherverwaltung von Java als drittes Konzept manuelle Speicherverwaltung hinzutreten. Mithilfe der JNI-Funktionen „DeleteLocalRef“ beziehungsweise „DeleteGlobalRef“ können Referenzen vorzeitig für den Garbage Collector freigegeben werden.

Fazit

Das Beispiel zeigt, dass das Verwenden von Funktionalität aus Java-Programmen mithilfe von JNI in Cobol realisierbar ist. Das Aufrufen von Methoden mit dem „INVOKE“-Statement ist intuitiv möglich. Dennoch treffen sehr unterschiedliche Eigenschaften der Programmiersprachen aufeinander. Cobol-Anwendungen verwenden traditionell statisches Speicher-Management, bei dem der verwendete Speicher vor Ablauf des Programms feststeht. Demgegenüber steht Java mit seiner dynamischen Speicherverwaltung und dem Garbage Collector. Java-Konzepte wie Exception Handling müssen beachtet werden. Den unterschiedlichen Datentypen in den beiden Programmiersprachen muss beim Austausch Rechnung getragen werden. So werden in Cobol Zahlen oft in Fix-Komma-Notation abgelegt und benötigen somit eine Umwandlung für die Übertragung in Java-Programme. Insbesondere, wenn auf Datenbanken zugegriffen wird und hybride Anwendungen denselben transaktionalen Kontext benötigen, stößt man schnell auf die Grenzen der Technologien und muss unter Umständen zusätzliche Middleware integrieren.

Zukünftige Modernisierungsprojekte werden sich dem Einsatz von Java kaum entziehen können. Der Markt ist gefüllt von Experten, Werkzeugen und Vorgehensmodellen für die Implementierung von Java-Anwendungen. Die großen Herausforderungen werden eher im Bereich der Integration entstehen, da Cobol und Java-Know-how gleichermaßen benötigt werden und Schnittstellen aufgrund der höheren Komplexität gut durchdacht werden müssen. Zudem sind verschiedene Entwicklermentalitäten und Entwicklungsprozesse zu vereinen. Ein spannendes Feld für Architekten und Java-Entwickler gleichermaßen.

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.

Martin Kiefer
martinkiefer@boxm.de

Matrin Kiefer absolviert ein duales Informatikstudium in Kooperation mit der IBM und beschäftigt sich dabei mit den Themen der Anwendungsmodernisierung auf dem Mainframe und der Anwendungsentwicklung mit Java und C++, sowie z/Linux.

Impressum

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

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

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

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

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

Titel, Gestaltung und Satz:
Alexander Kermas
DOAG Dienstleistungen GmbH
Foto Titel © burak çakmak / Fotolia.com
Foto S. 25 © alphaspirit / Fotolia.com
Foto S. 34 © burak çakmak / Fotolia.com
Foto S. 45 © Picture-Factory / Fotolia.com

Anzeigen:
Simone Fischer
anzeigen@doag.org

Mediadaten und Preise:
http://www.doag.org/go/mediadaten

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

Java aktuell
Magazin der Java-Community

www.ijug.eu

JETZT ABO BESTELLEN

Sichern Sie sich 4 Ausgaben für 18 EUR

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

FAXEN SIE DAS AUSGEFÜLLTE FORMULAR AN
0700 11 36 24 39

ODER BESTELLEN SIE ONLINE
go.ijug.eu/go/abo

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

Java aktuell

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

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

ANSCHRIFT

Name, Vorname

Firma

Abteilung

Straße, Hausnummer

PLZ, Ort

GGF. ABWEICHENDE RECHNUNGSANSCHRIFT

Straße, Hausnummer

PLZ, Ort

E-Mail

Telefonnummer

X

Die allgemeinen Geschäftsbedingungen* erkenne ich an, Datum, Unterschrift

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