Herbst 2011

Das iJUG Magazin

Java aktuell

Java aktuell
Magazin der Java-Community

Java überall

- Neu: Java SE7
- Interview mit Patrick Curran, Vorsitzender des JCP
- Für Android entwickeln
- Testen mit Arquillian
- Suchen mit Apache Solr

Erfahrungen, Ideen und Lösungen für Java-Entwickler

iJUG Verbund

www.ijug.eu D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977 04/2011

Wolfgang Taschner
Chefredakteur Java aktuell

Java ist überall

Die Teilnehmer des Java Forums in Stuttgart waren am 7. Juli 2011 pünktlich um 18 Uhr zusammengekommen, um dem weltweit ersten Launch-Event von Java 7 beizuwohnen. Hoffentlich war es kein schlechtes Omen, dass die Live-Übertragung aus den Oracle-Headquarters wenige Minuten nach der Begrüßung wegen eines starken Gewitters zusammenbrach. Die Rede von Adam Messinger, Oracle Vice President Development Fusion Middleware Group, blieb im Dunkeln. Als die Übertragung etwa eine Stunde später wieder stand, war der Raum leer. Die Zuhörer hatten sich auf den Weg gemacht – wenigstens durfte jeder ein weißes T-Shirt mit der Aufschrift „Moving Java forward" und den Tournee-Orten auf der Rückseite mit nach Hause nehmen.

Bereits im Vorfeld hatte Oracle den Mitgliedern im Interessenverbund der Java User Groups (iJUG) e.V. lokale Java-7-Launch-Events angeboten. Nachdem sich etliche Usergroups zur Teilnahme an einer solchen Veranstaltung bereit erklärt hatten, machte Oracle wieder einen Rückzieher und forderte zu einem erneuten Registrierungsverfahren auf. Als Ergebnis blieben nur noch die Launch-Events in Zürich, Berlin und München übrig. Offensichtlich hatte Oracle das Interesse der Usergroups unterschätzt.

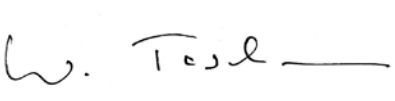
Die Vorstellung von Java 7 ist ein erster Schritt von Oracle, dem hoffentlich viele weitere und vor allem auch größere folgen werden. Noch sind die Gräben zwischen dem Hersteller und der Community überwindbar, auch wenn die anfängliche Skepsis nach der Sun-Übernahme noch nicht abgeklungen ist.

Sich über die Funktionen in Java zu beschweren ist das eine, sich am Community Process zu beteiligen das andere. Einige iJUG-Aktive nutzten im Rahmen des Java Forums in Stuttgart die Gelegenheit, mit Patrick Curran, dem Vorsitzenden des Java Community Process (JCP), zu reden (siehe Interview auf Seite 9). Neben der Arbeitsweise des JCP kam in dem Gespräch vor allem heraus, wie gering das Interesse der Java-Community an einer Mitarbeit am Community Process ist. Dass beim Final Draft eines wichtigen JSRs gerade mal vier Leute Feedback gaben, spricht Bände. Der iJUG hat das Signal verstanden und wird sich um eine Mitgliedschaft im JCP bewerben.

Unsere Titelgeschichte „Java überall" macht die immense Verbreitung dieser Sprache transparent. Neben den zahlreichen Artikeln über Java selbst finden Sie in dieser Ausgabe auch Themen wie „Arquillian", „Solr", „Android" oder „Hibernate". Sie zeigen, auf welche Gebiete Java bereits vorgedrungen ist.

Neu in dieser Ausgabe ist die Rubrik „Update" auf Seite 5. Dort erhalten Sie ab sofort in jeder Ausgabe ein Update über das aktuelle Geschehen in der Java-Community, damit Sie immer auf dem Laufenden sind.

Ich wünsche Ihnen viel Spaß beim Lesen und viel Erfolg mit Ihren Java-Projekten.



Inhalt

Kleiner Exkurs darüber, wo wir Java direkt oder indirekt überall antreffen, Seite 15

Zehn Tage nach der Verabschiedung der Spezifikation hat das JDK 7 den Status „general availability" erreicht, Seite 5

Die lange Reise von Java 7

Markus Eisele, msg systems ag

Vier Jahre, sieben Monate und genau siebzehn Tage – es ist vollbracht. Das neue Java 7 liegt auf vielen Rechnern. Nachdem bereits am vertrieblich deutlich besser geeigneten 7. Juli 2011 rund um die Welt sogenannte „Java 7 Launch Events" stattfanden, gab das Executive Committee für Java SE/EE am 18. Juli 2011 alle vier für Java 7 relevanten JSRs (292, 334, 203 und 336) frei.

Seit Anfang dieses Jahres konnte sich jedermann die Entwickler-Vorschau (Milestone-Build 12) des OpenJDK 7 herunterladen und ausgiebig testen. Sie war bereits funktional vollständig und die Entwicklergemeinde war aufgerufen, die letzten verbliebenen Fehler zu finden. Das OpenJDK bildet dann auch den Kern der offiziellen Java-7-SDK/JRE-Versionen, die aktuell von Oracles Webseiten heruntergeladen werden können [1]. Bei der Bewertung der neuen Funktionen muss man dabei das Wissen um den von Mark Reinhold, Chief Platform Architect der Java-Plattform, vorgeschlagenen Kompromiss „Plan B" im Auge behalten. Schon auf der JavaOne letzten Jahres wurde der noch von Sun geplante Umfang für Java 7 stark eingeschränkt [2]. Wirklich große Erweiterungen wie beispielsweise das Modulsystem Jingsaw [3] oder auch die Lambda-Ausdrücke [4] sollen sich erst im kommenden Java 8 wiederfinden, das für Ende 2012 erwartet wird. Vor diesem Hintergrund kann man Java 7 als konsequente Evolution, aber keinesfalls als Revolution bezeichnen.

InvokeDynamic

Die wohl tiefgreifendste Erweiterung beinhaltet der invokeDynamic 292 [5]. Erstmals seit Java 1.0 wurde ein neuer Bytecode in die JVM eingeführt. Bereits seit Längerem stellt die JVM nicht mehr nur für Java die Basis. Sie konnte immer schon beliebigen Bytecode ausführen. Die eigentliche Programmiersprache, aus der dieser stammte, war und ist egal. Einen geeigneten Bytecode-Compiler haben diverse andere Programmiersprachen so auch schon lange (etwa Python oder Ruby). Der größte Unterschied besteht darin, dass vielfach dynamische Sprachen versuchen, diesen Weg zu gehen. Die eingesetzte dynamische Typisierung stellt ein großes Problem für den Compilerbau dar, da Java und somit auch die JVM auf statische Typisierung ausgelegt sind. Bisher war diese nur durch große Umwege erreichbar und führte vielfach dazu, dass der Bytecode komplex wurde. Darüber hinaus konnten die JVMs kaum Laufzeitoptimierungen durchführen. Beides in Kombination führt zu deutlichen Performance-Einbußen. Dieses Problem soll nun der neue Bytecode „InvokeDynamic" lösen. Für die Mehrzahl der Entwickler wird sich diese Erweiterung nicht auswirken. Lediglich bei der Verwendung anderer JVM-basierter Sprachen ist mit einer deutlichen Performance-Steigerung zu rechnen.

Projekt „Coin"

Ganz anders sieht das bei den Änderungen unter dem Deckmantel des sogenannten Projekts „Coin" aus. Die hier gesammelten Funktionen sind das Ergebnis eines offenen Aufrufs für Verbesserungen an der Sprache Java. Dieser fand zwischen Ende Februar und Ende März 2009 statt. Aus knapp 70 Vorschlägen haben es dann genau sechs in die nächste Java-Version geschafft. Wie der Name schon sagt (Coin = Münze), sind es Kleinigkeiten. Dies wird auch eindrucksvoll durch den prominenten ersten Punkt auf der Liste der sechs Änderungen deutlich. Seit Java7 können Strings in Switch-Statements verwendet werden. Bisher ging das nur mit primitiven Daten-Typen, ihren Wrappern und Enums. So ist nun Folgendes möglich:

```
public String getISSNNumber(String
magazin) {
    String issnNumber = „";
    if (magazin.isEmpty()) {
        return issnNumber;
    }
    switch (magazin.toLowerCase()) {
        case „JavaAktuell":
            issnNumber = „2191-6977";
            break;
        case „DOAGNews":
            issnNumber = „0936-0360";
            break;
        default:
            issnNumber = „";
            break;
    }
    return issnNumber;
}
```

Dabei wird der String im Switch-Ausdruck via „String.equals" mit dem im Case-Fall verglichen und die entsprechende ISSN des Magazins zurückgegeben. Neu ist auch die Möglichkeit, Literale mit einem Unterstrich („_") zur besseren Lesbarkeit zu formatieren. Dabei darf der Unterstrich nicht am Ende oder am Anfang eines Literals sowie an zweideutigen Stellen, an denen die Interpretation gefährdet ist, stehen:

```
2191_6977
2_1_9_1__6_9_7_7L
```

Ein neuer Typ ist ebenfalls dazu gekommen. Es dürfen auch binäre Literale zum Einsatz kommen. Eingeleitet werden sie mit 0b oder 0B:

```
0b0001_0010_0100_1000
```

Statt der leidlichen Aneinanderreihung von Catch-Blöcken ist jetzt das Fangen mehre-

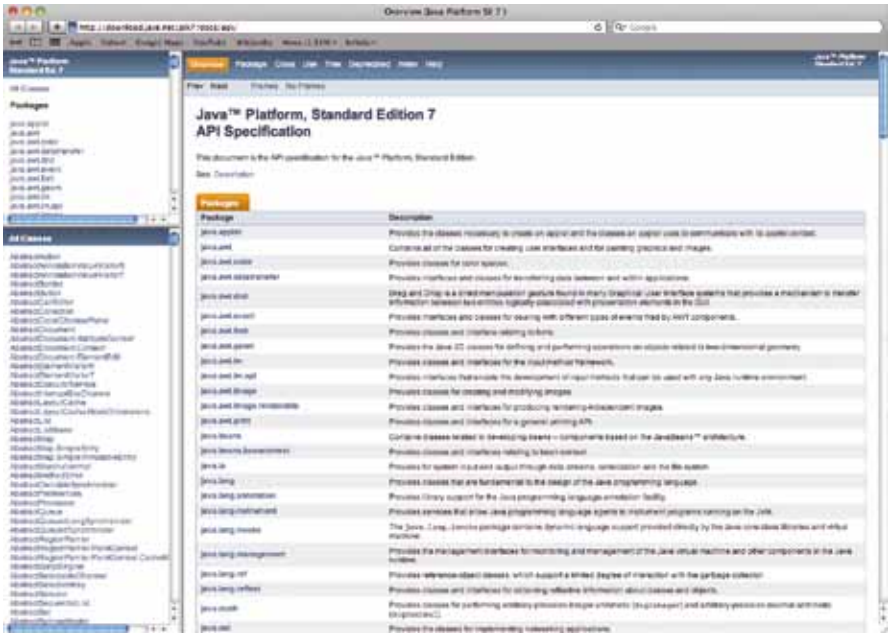


Abbildung 1: SE7 API-Spezifikation

rer Exceptions in einem Catch-Block möglich. Dabei werden die Exception-Typen per OR-„|"-Symbol voneinander getrennt. Für alle Typen wird allerdings der gleiche Exception-Parameter verwendet:

```
try {
 throwAorB();
} catch (ExceptionA | ExceptionB ex) {
   throw ex;
}
```

Eine Verbesserung der Exception-Analyse zur Compile-Zeit kam ebenfalls dazu. Das erneute Werfen einer Exception in einem Catch-Block (re-throw) erzeugt nun ein Objekt vom Typ der ursprünglich geworfenen Exception und nicht mehr vom Typ der gefangenen.

Die Syntax der Java-Generics wurde vereinfacht. Etwaige Redundanzen der Typangaben können jetzt mit dem „Diamond Operator" (<>) vermieden werden. Im folgenden Beispiel entscheidet dabei der Compiler über den Typ der HashMap:

```
Map<Magazin> magazins = new
HashMap<>();
```

Der Ressourcen-Verwaltung ist die nächste Änderung gewidmet. Auch unter der Bezeichnung „Automatic Resource Management" (ARM) bekannt, hat „try-with-resources" Einzug in den Funktionsumfang gehalten. Ein neues Interface („AutoClosable") mit einer Methode („close()") wird dabei automatisch aufgerufen, wenn im „try"-Block Ressourcen vorkommen. Ein explizites Schließen ist damit nicht mehr notwendig. In einem Block lassen sich dabei mehrere Ressourcen gleichzeitig definieren:

```
try (InputStream fis = new
FileInputStream(source);
      OutputStream fos = new
      FileOutputStream(target)){
      // lesen und schreiben
} // Ressourcen automatisch geschlossen.
```

Eine Änderung an den Compiler-Warnungen bei der Verwendung sogenannter „Non-reifiable-Typen" (E, List) in Methoden mit Vararg-Parametern (Variable Anzahl von Parametern) beendet die Liste der neuen Funktionen. Hat man bisher versucht, eine „varargs"-Methode mit Non-reifiable-Typen aufzurufen, wurde eine „unchecked"-Warnung vom Compiler erzeugt, die der Programmierer nicht unterdrücken konnte. Mit Java 7 wartet der Compiler nicht mehr bis zum Aufruf solcher Methoden, sondern erzeugt bereits bei der Definition entsprechende Warnungen. Diese lassen sich jetzt zusätzlich mit dem neuen Attribut („varargs") an der „@SuppressWarnings"-Annotation unterdrücken.

Fork/Join

Mit dem JSR 166y starteten die Überlegungen zum Thema „Nebenläufigkeit" in Java. Bereits mit Java 5 eingeführt und in Java 6 verbessert, finden sich die relevanten Klassen im Package „java.util.concurrent". Als Hilfsklassen zur Erleichterung der fehlerträchtigen Entwicklung von nebenläufigen Applikationen wurde es bisher aber recht wenig genutzt, da sein Einsatz noch vergleichsweise komplex ist. Fork/Join versucht dies zu überwinden und stellt eine Erweiterung zu den Executor-Services dar. Dabei stellt es die Grundlagen für die Implementierung von Algorithmen aus der sogenannten „teile und herrsche"-Familie (devide-and-conquer). Solche Algorithmen zerlegen komplexe Probleme rekursiv in kleinere Unterprobleme, bis ein Abbruchkriterium erreicht ist. Unterprobleme werden mit dem Fork/Join-Ansatz in parallelen Threads abgearbeitet, um deren Handhabung sich ein Entwickler nicht mehr kümmern muss. Die beiden Klassen „RecursiveTask" (mit Rückgabewert) und „RecursiveAction" (ohne Rückgabewert) mit den jeweiligen „compute()"-Methoden sind dabei das Zentrum der Implementierung. Das Lösungsschema ist wie folgt:

```
Ergebnis loese(Problem problem) {
   if (problem is klein)
     löse das Problem direkt
   else {
     teile das Problem in unabhängige Proble-
     me auf
     erstelle einen neuen Untertask um beide
     Teilprobleme zu lösen (fork)
     verbinde beide Untertasks (join)
     erstelle ein Gesamtergebnis aus den
     Teilergebnissen
    }
}
```

Darüber hinaus gibt es noch weitere Neuerungen. Details können unter [7] nachgelesen werden.

New I/O (NIO.2)

Auch die NIO.2-Grundlagen bestehen schon verhältnismäßig lange, wurden jetzt aber unter dem JSR 203 [8] zusammengefasst und offiziell in Java 7 eingebracht. NIO.2 bringt eine komplett neue API mit, die den Umgang mit Dateisystemen erheblich erleichtert. Dabei berücksichtigt sie auch ZIP- und JAR-Files. Zentrale Klas-

sen stellt das Package „java.nio.file“ zur Verfügung. Dabei ist der Nachfolger von „java.io.file“ die zentrale Klasse für neue Funktionen. Mit „java.nio.file.path“ wird nicht nur eine einzelne Datei, sondern ein kompletter Pfad im Dateisystem repräsentiert. Eine enge Integration mit der bisherigen File-Klasse sorgt für Rückwärtskompatibilität:

```
File file = new java.io.File(„mypath/subPath“);
Path path = file.toPath();
```

Hinter der abstrakten Klasse „Path“ liegen diverse Betriebssystem-spezifische Implementierungen (bsp. sun.nio.fs.WindowsPath), die sich letztendlich auf den ebenfalls neu eingeführten FileSystemProvider-SPIs abstützen. Die ebenfalls neue Klasse „Files“ bietet jetzt zentral alle Funktionen zum Arbeiten mit Dateien. Auch Kopieren wird nach fast 15 Jahren endlich auf einfache Art unterstützt:

```
Path path = new java.io.File(„/temp/test.txt“).
toPath();
Path path2 = new java.io.File(„/temp/test2.
txt“).toPath();
Files.copy(path, path2, StandardCopyOption.
REPLACE_EXISTING);
```

Eine gute Übersicht der neuen Funktionen bietet der NIO.2-Teil der Oracle Java Tutorials [9].

Abseits dieser großen Änderungen an der grundlegenden Sprache finden sich noch viele Verbesserungen an Bewährtem. So hat das Nimbus-„Look&Feel“, das bisher nur als proprietäre Erweiterung klassifiziert war, seinen Weg in den Standard geschafft und seine Klassen sind in das Paket „java.swing“ umgezogen. Ein tiefliegendes Deadlock-Problem in Java-Classloader wurde mit Veränderungen an der API beseitigt [10] und etliche Standards auf den neuesten Stand gehoben. Dazu gehört beispielsweise die Unicode-Unterstützung, die jetzt in Version 6.0 enthalten ist und Drei-Buchstaben-Ländercodes verstehen kann. Neben der Unterstützung des Vista-IPv6-Stacks haben auch das Stream Control Transmission Protocol (SCTP), das auf Solaris-Plattformen verwendet wird, sowie das Sockets Direct Protocol (SDP) ihren Weg in den Standard geschafft. JDBC liegt mittlerweile als Version 4.1 zugrunde und neben der Umsetzung der „try-with-resources“-Umfänge ist jetzt eine RowSetFactory vorhanden, mit deren Hilfe RowSet-Objekte erzeugt werden können. Auch FilteredRowSet [11] und CachedRowSet sind neu dabei und vereinfachen die Arbeit. Mit JAXP 1.4.4, JAXWS 2.2 und JAXB 2.2 tragen jetzt auch die aktuellen XML-APIs zu einem

Foto: Markus Eisele

Mark Reinhold auf der JavaOne 2010

runden Bild von Java 7 bei. Optisch haben sich die JavaDocs weiterentwickelt. Neben einer Unterstützung für Cascading-Style-Sheets wurde der Generierungsprozess (javadoc) umgestaltet, um zukünftig deutlich schneller zu arbeiten und valides HTML zu generieren.

Fazit

Die eingeführten Änderungen stellen endlich wieder einen Schritt nach vorne für Java dar. Leider bleiben die wirklich großen Änderungen, die Sun ursprünglich geplant hatte, noch in weiter Ferne. Sie kommen erst mit JDK 8 und 9. Ersteres ist zumindest bereits für Ende 2012 avisiert.

JDK	Veröffentlicht	Kommentar
JDK 1.1	19.02.1997	Große Erweiterungen: Umbau des AWT event models, Spracherweiterungen Inner Classes, JavaBeans und JDBC.
J2SE 1.2	8.12.1998	Reflection, Collections framework, Java IDL (IDL-Implementierung für CORBA-Interoperabilität), Swing graphical API, Java Plug-in, JIT compiler.
J2SE 1.3	08.05.2000	Bundling der HotSpot JVM (im April 1999 Release für J2SE 1.2), JavaSound, JNDI und JPDA.
J2SE 1.4	06.02.2002	Entwickelt als JSR 59, Regular expressions wie in Perl, exception chaining, Integrierter XML parser und XSLT processor (JAXP), Java Web Start.
J2SE 5.0 or 1.5	30.09.2004	Entwickelt als JSR 176, neue Sprach-Features (for-each loop, generics, autoboxing and var-args)
Java SE 6	11.12.2006	Database manager, Scripting languages in der JVM, Visual Basic language support. JSR 269, GUI-Verbesserungen, inkl. native UI-Erweiterungen, Verbesserungen JPDA & JVM Tool Interface
Java SE 7	28.07.2011	Gestartet im August 2006

Tabelle: Die Geschichte der Bohne

Die Produktivität mit Java scheint wieder deutlich mehr an Bedeutung zu gewinnen. Die Einführung von Generics oder auch der Umgang mit Nebenläufigkeit haben in der Vergangenheit viel mehr dazu geführt, dass Java komplexer wurde und die Anforderungen an die Programmierer stiegen. Gerade hier konnten sich andere Sprachen schneller und besser entwickeln. Java und sein „Steward" Oracle haben nun die Aufgabe vor sich, den Rückstand aufzuholen und den geschätzten neun Millionen Java-Programmierern wieder ein modernes, schnelles und leichtgewichtiges Werkzeug an die Hand zu geben. Als solches hat James Gosling es nämlich einmal konzipiert.

Weitere Informationen

[1] http://www.java.com/
[2] http://mreinhold.org/blog/rethinking-jdk7
[3] http://openjdk.java.net/projects/jigsaw/
[4] http://openjdk.java.net/projects/lambda/
[5] http://www.jcp.org/en/jsr/summary?id=292
[6] http://www.jcp.org/en/jsr/summary?id=336
[7] http://g.oswego.edu/dl/concurrency-interest/
[8] http://www.jcp.org/en/jsr/summary?id=203
[9] http://download.oracle.com/javase/tutorial/essential/io/fileio.html
[10] http://openjdk.java.net/groups/core-libs/ClassLoaderProposal.html
[11] http://download.oracle.com/javase/tutorial/jdbc/basics/filteredrowset.html

Markus Eisele
markus.eisele@msg-systems.com

Markus Eisele arbeitet bei der msg systems AG in München. Er betreibt einen Blog zu Themen rund um Enterprise Java unter http://blog.eisele.net.

Von links: Patrick Curran, Andreas Badelt, Tobias Frech und Dr. Michael Paus

„Unbedingt die Specs lesen und Feedback geben …"

Der Java Community Process (JCP) stellt die Weichen für alle neuen Features in Java. Die iJUG-Aktiven Andreas Badelt, Tobias Frech und Dr. Michael Paus sprachen darüber mit Patrick Curran, dem Vorsitzenden des JCP.

Welche Position haben Sie bei Oracle und was sind Ihre Ziele?

Curran: Ich bin Vorsitzender des Java Community Process. Obwohl ich bei Oracle angestellt bin, verbringe ich fast meine gesamte Zeit mit dem JCP. Zu meinen Aufgaben zählt es, den Prozess voranzubringen, die Mitgliedschaft zu managen, Spezifikationen und Experten durch den Prozess zu führen, die Treffen des Executive Committees zu organisieren und die Webseiten des JCP zu betreuen.

Wie ist der derzeitige Stand im Executive Committee?

Curran: Wir haben gerade die im letzten Jahr weggegangenen Leute ersetzt. Es gab lange Diskussionen über Apache und TCK-Angelegenheiten, die jetzt beendet sind, auch wenn nicht alle darüber glücklich sind, dass Apache den JCP verlassen hat. Letztendlich gibt es jetzt wieder einen Fortschritt, nachdem unter Sun in den letzten zwei Jahren nicht mehr viel passiert ist. Oracle hat eine Entscheidung hinsichtlich der JSRs zu Java 7 und Java 8 getroffen und das Executive Committee ist damit einverstanden. Oracle hat auch die brasilianische Java User Group für einen der freien Sitze nominiert, darüber hinaus wurde die London Java Community gewählt, sodass wir zum ersten Mal im Executive Committee eine bedeutende Vertretung aus der Java-Community haben. Darüber hinaus arbeiten wir daran, die Java-Community auf diesem Weg möglichst intensiv in unsere Arbeit einzubeziehen.

Welche Strategie verbirgt sich hinter der Nominierung?

Curran: Nach dem Weggang von Apache ist der Eindruck entstanden, dass wir mehr Repräsentanz aus der Community benötigen. Oracle hingehen ist der Ansicht, dass die Arbeit nicht ohne ein großes Unternehmen erledigt werden kann. Ein Schritt in

Fotos: Wolfgang Taschner

diese Richtung war die Nominierung von Goldmann Sachs, ein Unternehmen, das Tausende von Java-Entwicklern beschäftigt. Als Gegengewicht dazu sollten aber auch die Java User Groups vertreten sein, um der Java-Community entsprechend Einfluss zu geben.

Welche Rolle spielt das Executive Committee im Java Community Process?

Curran: Das Executive Committee hat die Rolle eines Vorstandsgremiums. Es gibt zwei Committees, eines für Java ME und eines für Java SE und EE, die voraussichtlich im kommenden Jahr zusammengeführt werden. Die wichtigste Aufgabe besteht darin, JSRs freizugeben, die den Java Community Process durchlaufen haben. Zudem legt das Executive Committee die Abläufe in der Organisation fest. Der Fokus in den kommenden Jahren liegt auf den nächsten zwei oder drei JSRs. Darüber hinaus soll der Java Community Process transparenter werden und die Leute sollen einfacher daran mitarbeiten können.

Das heißt, es wird bald einen neuen JSR geben?

Curran: Wir arbeiten daran und er wird hoffentlich Mitte nächsten Jahres verfügbar sein.

Was ist der Hintergrund dafür, die beiden Executive Committees künftig zusammenzulegen?

Curran: Wir gehen davon aus, dass die Plattformen in den nächsten Jahren ebenfalls verschmelzen werden. Java ME wird, wie es schon einmal war, eine Teilmenge von Java SE sein. Außerdem erledigen die beiden Committees bereits jetzt schon viele Aufgaben gemeinsam.

Wenn Java ME eine Teilmenge von Java SE werden soll, welche Auswirkungen hat das auf die Strategie eines Java für mobile Endgeräte?

Curran: Die Grenzen zwischen Java ME und SE werden sich auflösen. Wir haben dann eine einzige Plattform, die sich individuell an unterschiedliche Umgebungen anpassen lässt.

Es besteht der Eindruck, dass nicht alle Bereiche von Java über einen JSR laufen, wie zum Beispiel Web Start oder Applets. Wie groß ist daher der Einfluss des Java Community Process auf die Entwicklung von Java als Ganzes?

Curran: JSRs umfassen die Mehrheit aller Features auf der Plattform. Es gibt jedoch Funktionen, die über den Standard hinausgehen und nicht in JSRs enthalten sind.

Wer kümmert sich um diese Features, die nicht Standard-Java sind, aber im Standard-JDK enthalten sind?

Curran: Es gibt keine exakte Spezifikation für den Standard. Ich hoffe, dass es diese für die kommende SE7-Plattform geben wird. Ich werde diesen Punkt mitnehmen und klären.

Gibt es dann Ergänzungen zu Java SE7, die über JSRs spezifiziert werden?

Curran: Der JSR zu SE7 enthält diese. Es ist jedoch schwierig, den gesamten Umfang im JDK abzubilden. Das war zuletzt bei der Version 4 der Fall.

Können Entwickler quasi ein Zertifikat dafür erwarten, dass ihre Applikation eine gültige Java-Implementierung auf Basis des Technology Compatible Kit (TCK) ist?

Curran: Ja, aber es reicht nicht aus, dabei nur auf den TCK zu schauen.

Wie kann man innovative Ideen in den Standardisierungsprozess einbringen?

Curran: Der Standardisierungsprozess ist keine Plattform für Experimente und Innovationen. Deshalb bringen wir hier nur Dinge ein, die bereits erprobt sind. Von daher halte ich einen Open-Source-Development-Process für frühe Phasen des Experimentierens für sinnvoll. Erst wenn sich hier ein Konsens gebildet hat, sollte er in den Standardisierungsprozess eingebracht werden. Momentan gibt es noch keine Vorstellung darüber, welche konkreten Funktionen in der nächsten Java-Generation benötigt werden.

Gibt es definierte Ziele für den JCP?

Curran: Wir wollen hier keine exakten Vorgaben machen und halten uns an die Expertengruppen für die einzelnen Plattformen. Dazu ein Beispiel: Im letzten Jahr war klar, dass eine Modularisierung erfolgen muss. Die Antwort auf das Wie konnte nur von den Plattform-Verantwortlichen kommen, die die entsprechende Expertise haben.

Können Technologien, die in der Community erprobt und bereit für die Standardisierung sind, vom Standardisierungsprozess profitieren?

Curran: Der große Benefit besteht darin, dass sich jeder auf die standardisierte Technologie verlassen kann. Erst dann wird es Implementierungen im größeren Stil geben.

Welchen Zeithorizont hat die Vorbereitung eines neuen JSRs?

Curran: Das hängt etwas vom Umfang des JSRs ab; bei den letzten hat es jeweils etwa neun Monate gedauert.

Gibt es auch eine längerfristige Strategie?

Curran: Oracle ist nun seit eineinhalb Jahren in dieser Position und ist sich seiner Verantwortung sehr bewusst. Wir arbeiten daran, den Java Community Process so umzugestalten, dass auch größere strategische Ziele umgesetzt werden können.

Patrick Curran

Dazu zählt auch, die Entwickler-Gemeinde besser einzubinden.

Hat der JCP auch einen JSR aufgesetzt, der die Java-Plattform als Ganzes betrachtet, um beispielsweise die Konsistenz zwischen verschiedenen Aspekten zu berücksichtigen und die Qualität zu sichern?

Curran: Wir haben darüber diskutiert. Es gab Leute, die dafür waren, andere waren es nicht. Aus meiner Sicht sollte es einen Architektur-Rat geben, der den Blick auf das Ganze hat und sagt, wo wir in fünf Jahren stehen wollen. Die Konsistenz ist allerdings Aufgabe der entsprechenden Executive Committees.

Sollten bestimmte Dinge wie beispielsweise Binding oder Properties, die gerade im Zusammenhang mit Java FX implementiert wurden, nicht Teil des Standards werden, damit Java FX und Java SE nicht weiter auseinanderlaufen?

Curran: Dies ist genau ein Punkt, bei dem uns die Java User Groups unterstützen können, indem sie das Executive Committee auf diese Sachen ansprechen.

Gibt es dafür einen offiziellen Weg?

Curran: Wir haben beim JSR-348 einen Feedback-Mechanismus für das Executive Committee eingeführt. Dies ist eine Möglichkeit, Einfluss zu nehmen.

Kann ein JCP-Mitglied das Executive Committee auffordern, bestimmte Dinge zu diskutieren?

Curran: Momentan können das nur die Mitglieder des Executive Committees tun. Es wird aber künftig offene Meetings des Executive Committees geben, bei denen JCP-Mitglieder bestimmte Punkte auf die Agenda stellen können. Das erste davon wird im Rahmen der JavaOne 2011 in San Francisco stattfinden.

Wie oft trifft sich das Executive Committee?

Curran: Wir kommen durchschnittlich etwa einmal im Monat zusammen.

Macht es Sinn, beispielsweise einen Request bezüglich Swing zu stellen?

Curran: Die richtige Stelle für solche Anfragen sind die Spec Leader des entsprechenden Bereichs auf der Plattform.

Was tun, wenn es keinen Spec Leader gibt, wie beim Service-Interface für Web Start?

Curran: Sie können sich gerne freiwillig melden, um dafür einen JSR zu beginnen.

Haben Sie Wünsche an die Java-Community?

Curran: Sie sollten unbedingt die Specs lesen und Feedback geben. Ich erinnere mich an einen sehr wichtigen JSR vor einigen Jahren. Da haben wir nur fünf Kommentare auf den Early Draft bekommen. Es hilft nichts, wenn sich die Leute über die finale Version beschweren. Sie müssen sich vorher zu Wort melden. Es gibt keine Garantie dafür, dass alle Vorschläge implementiert werden, aber wir beschäftigen uns in jedem Fall damit und geben entsprechend Feedback.

Pressemeldung des iJUG

iJUG wünscht gemeinsame Zukunft mit Oracle

Aus der deutschen Java-Community wurde die Idee an den iJUG e.V. herangetragen, aus dem Apache Harmony-Projekt und einem Fork des OpenJDK eine eigenständige Plattform zu entwickeln, die weitestgehend unabhängig von Oracle und dem „Original-Java“ ist. Nach Abwägung aller Für und Wider tendiert der iJUG unter gewissen Voraussetzungen gegen eine Abspaltung von Oracle.

Der diskutierte Vorschlag hätte in jedem Fall ein Aufspaltung des „Java-Ökosystems“ zur Folge. Ein Teil würde sich dem Aufbau einer neuen Plattform widmen, ähnlich der Java-Plattform, aber prinzipiell unabhängig und stärker Open-Source-getrieben. Der andere Teil bliebe bei der ursprünglichen Plattform, die demgegenüber stärker kommerziell getrieben wäre, insbesondere von den großen Konzernen wie Oracle und IBM.

Inzwischen hat Oracle auch gezeigt, dass es die Roadmap ernst nimmt und die angekündigten Innovationen zügig vorantreibt. Bei allem Streit um einzelne Teile und Projekte sollte die offensichtlich immer noch vorhandene große Energie im Java-Ökosystem möglichst gebündelt und damit zum Vorteil aller genutzt werden.

Weiter auf www.ijug.eu

Das Java-Tagebuch

Andreas Badelt, Leiter SIG Java, DOAG Deutsche ORACLE-Anwendergruppe e.V.

Das Java-Tagebuch wurde in Ausgabe 2/2010 gestartet, um einen Überblick über die wichtigsten Geschehnisse rund um Java zu geben – in komprimierter Form und chronologisch geordnet. Der vorliegende Teil betrifft die Ereignisse im zweiten Quartal 2011.

2. April 2011

ActiveMQ Apollo: ActiveMQ rewrite using Scala 2.8

Der Message Broker Apache ActiveMQ wird mit dem Apollo-Projekt generalüberholt. Für Aufmerksamkeit sorgt, dass Apollo Scala nutzt, um die zentralen Dispatcher-Mechanismen zu implementieren. Die Entwickler sehen die kurze und prägnante Scala-Syntax als deutlichen Vorteil gegenüber einer Implementierung in Java. Sowohl für Scala-Fans als auch für diejenigen, die Sprachen wie Scala bis heute skeptisch gegenüberstehen, dürfte dieser „Härtetest" sehr interessant sein.
http://activemq.apache.org/apollo/versions/1.0-beta1/website/documentation/architecture.html

26. April 2011

Unterstützte Plattformen für das Oracle JDK 7

Henrik Ståhl verkündet in seinem Blog die Plattformen, die das für Juli angekündigte initiale Release des Oracle JDK 7 unterstützen wird:

- Microsoft Windows on x86 and x86-64
- Oracle, Red Hat, SuSE and Ubuntu Linux on x86 and x86-64
- Solaris on x86, x86-64 and SPARC

Ein Termin für den Mac-OS-Port steht noch nicht fest. Die Verzögerung bis zur Verfügbarkeit auf Mac OS ist sicher ein guter Gradmesser dafür, wie ernst die Parteien es mit dem gemeinsamen Projekt meinen. Aber auch andere Plattformen (wie IBM AIX und HP-UX) werden ja vom Oracle JDK nicht unterstützt. Auch hier sind die jeweiligen Hersteller gefordert – was bei IBM (neben Oracle der größter Contributor zum OpenJDK) auch nicht lange dauern dürfte, wenn der Wille da ist – wovon man sicher ausgehen kann.
http://blogs.oracle.com/henrik/entry/supported_platforms_for_jdk_7

Wahlen zum Java SE/EE und zum ME Executive Committee haben begonnen

Heute beginnen die Wahlen zu den vakant gewordenen Sitzen im Java SE/EE und ME Executive Committee. Bis zum 9. Mai 2011 können alle Mitglieder des JCP abstimmen. Für die Neubesetzung gibt es zwei „ratified seats", also Kandidaten, die von Oracle vorgeschlagen werden, aber mehrheitlich angenommen werden müssen – was zuletzt bei Hologic nicht geklappt hatte: Die brasilianische User Group SouJava und Goldman Sachs. Daneben gibt es einen „open election seat", um den sich mehrere Kandidaten bewerben: Portal-Hersteller Liferay, zwei persönliche Kandidaten (George Gastaldi und Siddique Hameed) sowie zwei weitere Usergroups – die London Java Community und die Central Ohio Java Users Group. Bei Java ME ist die Spannung geringer. Einziger Kandidat ist Alex Terrazas.
http://londonjavacommunity.wordpress.com/2011/05/04/why-we-nominated-ourselves-for-the-java-seee-executive-committee/

27. April 2011

Ist der Android-Hype schon vorbei?

Die Verkaufszahlen sagen etwas anderes, aber zumindest unter Entwicklern hat der Android-Hype eine Delle bekommen – das hat zumindest das neueste „Mobile Developer Survey" von Appcelerator und IDC ergeben. Die Begründungen der „Zauderer" stützen sich weniger auf den Rechtsstreit zwischen Oracle und Google und eventuell daraus entstehende Unsicherheiten als vielmehr auf die Zersplitterung der Android-Landschaft. Mehr Einheitlichkeit innerhalb der Android-Plattform über die verschiedenen Geräte hinweg wäre zu begrüßen. Aber noch schöner wäre es, wenn Android nach einer Einigung zwischen Google und Oracle in die Java-Plattform integriert werden würde. Man wird ja noch träumen dürfen …
http://www.appcelerator.com/company/survey-results/mobile-developer-report-april-2011/s

4. Mai 2011

Apache soll im Android-Rechtsstreit aussagen

Die Apache Software Foundation (ASF) wird ganz offiziell in den Rechtsstreit zwischen Oracle und Google hineingezogen. Oracle hat beantragt, die ASF als Zeugen zu vernehmen. Es geht um das Open Source JDK „Apache Harmony", auf dem Googles Android basiert. Insbesondere die jahrelangen, vergeblichen Bemühungen Apaches, das Java Test Compatibility Kit (TCK) ohne Einschränkungen zu erhalten, werden im Vordergrund stehen. Sowohl Sun als auch Oracle haben das TCK nie für die generelle Nutzung freigegeben – Java-Implementierungen für mobile Geräte waren immer ausgeklammert, weil schon für Sun der mobile Sektor eine wichtige Einnahmequelle war. Eine Lizenz ohne Einschränkungen hätte schon Sun sicher nur für viel Geld herausgegeben. Eine der

Fragen in Oracles Argumentation wird nun lauten: „Wie konnte Google auf dieser Basis eine Plattform für mobile Geräte entwickeln?“

Hudson und Eclipse

Nach der 180-Grad-Wende folgt nun eine weitere scharfe Kurve – hoffentlich wird den Verantwortlichen bei Oracle nicht schwindlig. Das Hudson-Projekt war seinem Bruder Jenkins kurz nach der Trennung ja noch auf die GitHub-Plattform gefolgt – was die berechtigte Frage gegenüber Oracle aufwarf, warum es überhaupt zur Trennung kommen musste. Der Streit hatte ja erst begonnen, als Oracle sich gegen den Umzug ausgesprochen und mit der Keule „Namensrechte“ gedroht hatte. Aber es wird wohl nicht der letzte Umzug bleiben: Hudson soll jetzt ein Eclipse-Projekt werden. Mit der Übergabe an Eclipse wird Oracle auch noch die Rechte an der Marke Hudson und am Domänennamen „hudson-ci.org“ aufgeben.

10. Mai 2011

Wahlen zum Java SE/EE und zum ME Executive Committee beendet

Die Wahlen zu den JCP Executive Committees für SE/EE und ME sind beendet. Für die beiden „ratified seats“ sind diesmal beide von Oracle vorgeschlagenen Kandidaten akzeptiert worden: SouJava und Goldman Sachs. Den über eine „open election“ vergebenen dritten Sitz hat die London Java Community mit deutlichem Abstand vor Liferay Inc. und der Central Ohio Java Users Group gewonnen. Damit sind nun zwei Usergroups in diesem Committee vertreten. Für Java ME wurde Alexis Terrazas gewählt, der auch der einzige Kandidat war.

17. Mai 2011

Oracle plant die Zukunft des JCP

Oracle leitet eine Reform des Java Community Process ein – ein großer Evolutionsschritt, aber keine Revolution: Sie geben nicht gleich die Kontrolle aus der Hand und führen ein unabhängiges Standardisierungsgremium ein, wie es im Jahr 2007 noch ein großer Konzern von Sun gefordert hatte. Wie hieß der noch gleich? Richtig: Oracle. Java ist wohl zu viel wert, um es gleich ganz der Allgemeinheit zu spenden. Immerhin: Der JCP soll deutlich verbessert werden. Die Ansätze werden schon seit Längerem innerhalb der Executive Committees diskutiert, jetzt sollen endlich Taten folgen: JSR-348 „Towards a new version of the Java Community Process“ (kurz: „JCP.next JSR1“) wurde heute eingereicht und soll noch bis zum Herbst fertiggestellt werden. Die Themen sind Transparenz, Beteiligung, Agilität/Geschwindigkeit und Steuerung („transparency, participation, agility, governance“). Zunächst sollen einfache Maßnahmen zur Erreichung dieser Ziele beschlossen werden wie zum Beispiel öffentliche Treffen der Executive Committees oder die Nutzung öffentlicher Mailinglisten durch die Expert Groups – diese sind bereits dazu aufgefordert, aber bislang nicht verpflichtet. Generell soll die Community stärker eingebunden und JSRs sollen beschleunigt werden – inklusive Konsequenzen für JSRs beziehungsweise Expert Groups, die zu wenig Aktivität zeigen. Der Punkt „Governance“ drückt ein größeres Vorhaben aus: „JCP.next JSR1“ soll eine Maßnahme lediglich vorbereiten, die dann ein Folge-JSR umsetzt: die Zusammenlegung der beiden Executive Committees, um dem Zusammenwachsen der Bereiche Rechnung zu tragen. Hinweis: Mehr zu diesen Themen und zum JCP insgesamt im Interview mit dem JCP-Vorsitzenden Patrick Curran auf Seite 8.

18. Mai 2011

JRockit wird freigegeben

Oracle hat die JRockit VM freigegeben. Das Produkt, das früher nur zusammen mit dem Bea beziehungsweise Oracle Weblogic Server zu kaufen war, kann nun kostenlos auch für kommerzielle Zwecke genutzt werden. Ein paar Features, insbesondere die erweiterten Monitoring-Funktionen (JRockit Mission Control etc.), sind allerdings auch in Zukunft kostenpflichtig. Welche Features das genau sind, lässt sich hier nachlesen:
http://www.oracle.com/technetwork/java/javase/terms/products/index.html

24. Mai 2011

Hudson und Jenkins

Ein weiteres Postskriptum aus der Hudson-und-Jenkins-Geschichte: Die Entwickler des Jenkins-Projekts denken über eine Möglichkeit nach, unter den Schirm einer anderen Organisation zu schlüpfen, um die Namensrechte an Jenkins, Spenden etc. abzusichern. Die Idee einer Wiedervereinigung mit Hudson wurde aber abgelehnt – vielleicht auch wegen persönlicher Differenzen, aber auf jeden Fall aufgrund der damit einhergehenden technischen und organisatorischen Einschränkungen. Die Entwickler wollen sich insbesondere keinem formalen Prozess unterwerfen, wie ihn ein Übergang zur Eclipse Foundation erfordern würde. Wichtigstes Element dabei scheinen die wöchentlichen Release-Zyklen zu sein.
Damit werden die Projekte wohl weiter auseinanderlaufen. Aber mehr Vielfalt auf dem CI-Markt kann ja auch etwas Positives sein …
https://wiki.jenkins-ci.org/display/JENKINS/Possible+Jenkins+Umbrella+Foundationshttp://meetings.jenkins-ci.org/jenkins/2011/jenkins.2011-05-24-16.09.log.html

25. Mai 2011

„Requirements of a standard Java module system“

Das Projekt „Jigsaw“ – ein Modulsystem als Bestandteil der Java-Plattform (geplant für Java SE 8) hat viele Diskussionen hervorgerufen. Wichtige Kritikpunkte sind beispielsweise die Überschneidung mit OSGi und die nötige technische Abstimmung zwischen Java ME und SE. Daher hat sich auf Veranlassung der JCP Executive Committees eine Gruppe von Experten aus verschiedenen Bereichen zu einem „Modularity Summit“ getroffen. Die Ergebnisse hat Mark Reinhold, Oracles Chef-Architekt für die Java-Plattform, im Entwurf eines Anforderungsdokuments zusammengetragen (siehe http://openjdk.java.net/projects/jigsaw/doc/draft-java-module-system-requirements-12). Auch wenn hier viele Themen nur gestreift werden, ist dies zumindest ein guter Anfang und für die

Community natürlich eine exzellente Gelegenheit, Feedback zu geben – also los!
http://mreinhold.org/blog/module-system-requirements

27. Mai 2011

JavaFX Beta

Die Beta-Version von JavaFX 2.0 ist veröffentlicht. Die umgekrempelte UI-Plattform wird jetzt mit einer reinen Java-API ausgeliefert - JavaFX Script wurde ja von Oracle aufgegeben. Damit können nun auch andere auf der Java-Plattform laufende Sprachen JavaFX nutzen.
http://blogs.oracle.com/java/entry/javafx_2_0_beta_now

28. Mai 2011

Orace JDK: JavaDoc erhält einen neuen Anstrich

JavaDoc war in den vergangenen Java-Releases ein ruhender Pol während andere Plattformen wie Android mit deutlich schickeren und flexibleren Varianten herauskommen. Mit dem Release 7 des Oracle JDK (vormals Sun) wird sich nun erstmals etwas bewegen, wie Jonathan Gibbons („Tech. lead of the Java Language Tools Team") in seinem Blog schreibt: Die Verarbeitung wurde komplett umgestellt – statt direkt serialisierten Output zu schreiben, wird nun erst mal ein Document Tree erstellt. Das erhöht die Flexibilität und garantiert, dass gültiges HTML erzeugt wird. Darüber hinaus wurde unter anderem Unterstützung für CSS eingebaut, sodass das Layout an individuelle Bedürfnisse angepasst werden kann. Weitere tiefgreifende Änderungen wie die Integration von JavaScript für Suchfunktionen sollen folgen.
http://blogs.oracle.com/jjg/entry/what_s_up_javadoc

31. Mai 2011

JCP.next akzeptiert

Der „Meta-JSR" zur Änderung des Java Community Process ist von beiden Executive Committees angenommen worden. Es gab keine Gegenstimmen und keine Enthaltungen, allerdings hat eine Reihe von Mitgliedern, insbesondere im Executive Committee für ME, gar nicht an der Abstimmung teilgenommen. Es ergibt keinen Sinn, darüber zu spekulieren – die geplanten Änderungen am JCP sollen jedenfalls „Anreize" vorsehen, um die Beteiligung an solchen Abstimmungen zu erhöhen. Eine Enthaltung, am besten mit Kommentar (dieser wird ja auf der JSR-Seite veröffentlicht), ist allemal mehr wert als einfach nicht zu wählen.

6. Juni 2011

Oracle und die „böse Alternative" für Java

Oracle hat nach dem Kauf von Sun tatsächlich die Option erwogen, Java „privat" zu machen. Henrik Ståhl sagte dazu im Java Spotlight Podcast: „Wir hielten es nicht für eine gute Idee." In einem Blog-Eintrag von Roger Brinkley (http://groups.google.com/group/javaposse/browse_thread/thread/754e9bb31adf13f5#) ist seine Anfrage an Henrik Ståhl nachzulesen – ob das denn tatsächlich möglich gewesen wäre. Ståhls Antwort: Oracle ist Eigentümer oder Miteigentümer des gesamten geistigen Eigentums („Intellectual Property") des OpenJDK und könne damit auch rückwirkend jede beliebige Lizenz anwenden. Aber dies hätte zu einem Community-Fork geführt, der zu viele Nachteile mit sich gebracht hätte. Daher wurde diese Idee verworfen.

Java SE 7: „Public Review"-Phase erfolgreich abgeschlossen

Der JSR zu Java SE 7 hat die „Public Review"-Hürde im JCP genommen. Einige Mitglieder des Executive Committees nutzen die abschließende Abstimmung aber für Protestnoten. Google wendet sich erneut gegen die Lizenzbedingungen (die TCK-Lizenz) und stimmt deswegen sogar gegen den JSR. Werner Keil beklagt mangelnde Transparenz im „Umbrella" JSR sowie in einigen Detail-JSRs und enthält sich. Einige andere schlagen in die gleiche Kerbe, auch die neu ins EC gewählten Usergroups. Bis auf Google und Werner Keil stimmen aber dann doch alle mit „Ja". Eine Lösung für den Lizenzstreit ist nicht in Sicht; aber was das Thema „Transparenz" angeht, wird Oracle hoffentlich kurzfristig reagieren. Schließlich hat der neue Java-Eigner selbst den JSR „JCP.next" eingereicht – und damit gewisse Erwartungen geweckt.

7. Juni 2011

Oracle und Google

Es gibt neue Details aus dem Rechtsstreit um Android. Googles Anwälte reagieren auf eine Analyse des entstandenen Schadens durch den Bostoner Ökonomie-Professor Iain Cockburn im Auftrag von Oracle. Das Ergebnis der Analyse scheint für Google so unglaublich zu sein, dass sie allein schon die Weitergabe an die Jury verhindern wollen. Der Brief der Google-Anwälte ist öffentlich einsehbar, wurde allerdings an einigen Stellen geschwärzt, insbesondere dort, wo es um die Höhe der Streitwerte geht. FOSS-Aktivist Florian Müller analysiert diesen Brief in seinem Blog und sieht Google hier in einer schwachen Position. Die Berechnung des Schadens durch Oracles Gutachter beruht wohl insbesondere auf zwei Säulen: den Einnahmen, die Google mit Android bislang erzielt hat (wohl basierend auf Werbe-Einnahmen, von Lizenz-Einnahmen ist keine Rede), und dem Schaden, der Oracle durch die Zersplitterung der Java-Plattform entstanden sein soll. Müller hält es nicht für ausgeschlossen, dass Google am Ende sogar mehr als die Einnahmen zahlen muss, wenn die Patentrechtsverletzungen als vorsätzlich angesehen werden. Ein weiteres interessantes Detail: Der Brief bestätigt Gerüchte, dass Google tatsächlich bereits mit Sun Lizenzverhandlungen geführt, das Angebot damals aber abgelehnt hat. Das dürfte sich nun als Fehler herausstellen – eine Einigung mit Oracle oder eine Niederlage vor Gericht wird sicher deutlich teurer werden.
http://fosspatents.blogspot.com/2011/06/oracle-wants-huge-cut-of-googles-mobile.html

Erich Gamma geht zu Microsoft

Einer der führenden Köpfe kehrt Java den Rücken: Erich Gamma, Mitglied der „Gang of Four" („Design Patterns"), viele Jahre lang Entwicklungsleiter und quasi das Gesicht von Eclipse. Nun geht er ausgerechnet zu Microsoft. Aber Microsoft hat

ihm wohl ein sehr interessantes Angebot unterbreitet: Gamma kann in Zürich bleiben und dort ein Team aufbauen, das die Entwicklung von Visual Studio unterstützt. Die Bestürzung, die die Meldung bei einigen Bloggern auslöst, ist wohl ein bisschen übertrieben. Durch den Weggang einer einzelnen Person wird das Java-Ökosystem sicher keinen Schaden nehmen. Und da Java und .Net schon einiges voneinander abgeschaut haben, warum soll der Transfer nicht auch auf dieser Ebene stattfinden?
http://blogs.msdn.com/b/jasonz/archive/2011/06/06/welcome-erich-gamma-to-the-visual-studio-team.aspx

Java SE 7 ist „Feature Complete"
In der Java-Community setzt sich mehr und mehr der Gedanke durch, dass JavaFX 2.0 doch relativ rasch Swing ersetzen könnte. Dokumentiert ist das unter anderem in diesem Blog-Eintrag und den referenzierten Beiträgen: http://groups.google.com/group/javaposse/browse_thread/thread/a53ecf0618bfe604. Allen Bedenken zum Trotz ist die JavaFX 2.0 API – der „pure Java"-Ersatz für das JavaFX-Script – recht gelungen und mit Java SE 8 und Closures wird es auch „schlank" und elegant aussehen. Ein Kommentar zum erwähnten Blog bringt die eigentliche Frage aber auf den Punkt: JavaFX ist die Zukunft von Java auf dem Desktop – aber ob Java generell eine Zukunft auf dem Desktop hat, ist etwas anderes.

JDeveloper
Und wieder ein Hinweis auf eine IDE, deren Unterstützung beziehungsweise Weiterentwicklung Oracle in absehbarer Zeit offenbar nicht einstellen wird: Das Zwischenrelease JDeveloper 11g R2 ist verfügbar, unter anderem mit Unterstützung für OSGi-basierte Erweiterungen und für JSF 2.0 inklusive grafischem Editor.
http://www.oracle.com/us/corporate/press/406155?rssid=rss_ocom_pr

8. Juni 2011

Der „Eclipse Community Survey 2011"
Wer sich für Trends in der Entwicklergemeinde interessiert, wird den Eclipse Community Survey interessant finden. Die Trends beziehen sich zwar nur auf die Nutzer der Eclipse-Plattform, aber das ist immerhin eines der lebendigsten und wichtigsten Biotope im Java-Ökosystem. Nachzulesen ist dort unter anderem, dass Windows als Betriebssystem für die Entwickler gegenüber Linux wieder leicht zugelegt hat oder dass Git(Hub) und Mercurial sich gegenüber Subversion und CVS steigern konnten – wobei Subversion aber immer noch mit über 50 Prozent Anteil weit vorne liegt, gefolgt vom (in IT-Maßstäben) uralten CVS mit 13 Prozent. Interessantes Detail am Rande: Die größte Teilnehmergruppe (18 Prozent) kam aus Deutschland, noch vor den USA!
http://ianskerrett.wordpress.com/2011/06/08/trends-from-the-eclipse-community-survey-2011/

9. Juni 2011

Das OpenJDK wird Referenz-Implementierung für Java SE 7
Henrik Ståhl verkündet in seinem Blog, dass die Referenz-Implementierung für Java SE 7 vollständig auf dem OpenJDK basiert und unter der GPL Open-Source-Lizenz veröffentlicht wird. Bislang war immer das Sun JDK die Referenz-Implementierung und damit quasi automatisch Standardkonform. Dies brachte jedoch Nachteile mit sich: zum einen, weil der Sourcecode der Referenz-Implementierung nicht für alle verfügbar war – zum Beispiel für Projekte, die eine eigene Implementierung anstrebten. Zum anderen, weil das Sun JDK neben der Implementierung des Standards auch proprietäre Erweiterungen hat, was häufig für Verwirrung darüber gesorgt hat, was nun eigentlich Bestandteil des Standards ist und was nicht.
http://blogs.oracle.com/henrik/entry/moving_to_openjdk_as_the

10. Juni 2011

Entwurf für die OpenJDK-Verfassung steht
Der finale Entwurf der OpenJDK-„bylaws" steht. Diese Regeln sollen in Zukunft die Prozesse rund um das OpenJDK steuern. Dazu werden Gruppen (als Untereinheiten der OpenJDK Community zu bestimmten Themen), Projekte (die ein bestimmtes Artefakt schaffen sollen) sowie verschiedene Rollen definiert. Aus den Projekt-„Committern" und den Gruppenmitgliedern können „OpenJDK Members" ernannt werden (erfolgt durch bereits existierende Members). Über allem thront das „Governing Board". In diesem sind zwei freie Mitglieder, aus und von den Members gewählt, ein von Oracle ernannter „Lead" sowie der „Chair" (Oracle) und der „Vice Chair" (IBM). Das Gründungs-Board existiert bereits: Es besteht aus Adam Messinger und Mark Reinhold von Oracle, Jason Gartner von IBM und den freien Mitgliedern Doug Lea (State University New York) und Mike Milinkovich (Eclipse). Mit Basis-Demokratie und auch mit reiner Meritokratie (Herrschaft derjenigen, die sich Verdienste erworben haben, wie die besten und fleißigsten Entwickler) hat das natürlich nicht viel zu tun. Dass eine Mischung aus Tyrannei und Meritokratie erfolgreiche Open-Source-Projekte hervorbringen kann, hat unter anderem Linux bewiesen.

28. Juni 2011

OpenJDK: Verfassung angenommen
Die OpenJDK-„bylaws" werden von der Wählerschaft mit 70 Ja-Stimmen, 9 Enthaltungen und ohne Gegenstimme angenommen. Die Wählerschaft war in diesem Fall vom Gründungs-Governing-Board festgelegt worden: Es bestand aus allen „Contributors", die in den vergangenen zwei Jahren mindestens fünf unterschiedliche, nicht-triviale Change-Sets im OpenJDK Repository erzeugt haben. Wenn man sich diese Liste anschaut, fällt auf, dass die Mehrheit davon Oracle-Angestellte sind. Ob strategisch geplant oder nicht – die Wahl war demnach ein Heimspiel für Oracle.

Andreas Badelt
Andreas.badelt@doag.org

Andreas Badelt ist Berater und Softwareentwickler / -Architekt bei der CGI Information Systems and Management Consultants (Deutschland) GmbH. Daneben organisiert er seit 2001 die Special Interest Group (SIG) Development bzw. SIG Java der DOAG.

Java überall

Oliver Szymanski, Source-Knights.com, stellv. Vorstandvorsitzender des iJUG

Als Java unter dem Namen „The Green Project" in seiner ersten Version 1991 entwickelt wurde, die auch als „Oak" (Object Application Kernel) bekannt ist, konnte die Welt nicht ahnen, wohin dies führen sollte. Doch Java ging bedächtig seinen Weg, wie auch die aus ihm entstandenen Java-Applikationen, von denen es immer hieß, wie langsam sie seien. Sie wurden schneller und Java eroberte die Welt der IT. Der Artikel gibt einen kleinen Exkurs (ohne den Anspruch auf Vollständigkeit) darüber, wo wir Java direkt oder indirekt antreffen.

Java war von Beginn an nicht als reine Sprache, sondern als eine voll funktionierende Betriebssystem-Umgebung auf einer virtuellen CPU gedacht. Java sollte unterschiedlichste Geräte steuern und dabei lag ein Hauptaugenmerk immer auf interaktivem Austausch von Java mit Benutzern. Dieser Schwerpunkt auf Interaktion trat einige Zeit in den Hintergrund, als Applets gegen ihren schlechten Ruf verloren hatten und Java mehr oder weniger zur Backend-Technologie wurde. Über den Umweg der Browser-Frontend-Entwicklung und der Methode, HTML-Clients per Java-Technologien dynamisch zu erzeugen und auszuliefern, näherte man sich der Client-Entwicklung. Mit der Wiederbelebung von Applets, neueren Technologien wie JavaFX, diversen Verbesserungen an Swing, Rich-Client-Plattformen wie die von Eclipse oder NetBeans und dergleichen wurde der Client-Markt langsam wieder direkt angesprochen.

Viele Frameworks entstanden während der Geschichte von Java. Eine Vielzahl an Firmen wurde über die Jahre im Java-Bereich tätig, teils im Open-Source-Umfeld, das Java nach Linux wohl sehr stark prägte, teils im kommerziellen Bereich. Entwickler lernten Java als Sprache und über die Jahre der Arbeit mit dieser Sprache die Unmengen an Tools, APIs und Frameworks. Obwohl die Sprache Java vergleichsweise schnell erlernbar ist, kann man die Technologien, die damit verbunden sind und die die eigentliche Java-Welt ausmachen, nur durch viele Jahre damit verbundener Arbeit kennen und nutzen lernen.

Java ist somit eine Technologie-Plattform aus virtueller Maschine (mit vielen darauf lauffähigen Sprachen), einer speziellen Java-Sprache, zahlreichen Basis-APIs und Bibliotheken sowie einer schieren Unmenge an Software von vielen Herstellern. Betroffene von Java sind somit der Hersteller von Java selbst beziehungsweise der sogenannte Besitzer (Oracle), die Nutzer (Unternehmen und Einzelpersonen), unabhängige und abhängige Hersteller der Entwicklungsumgebungen, Tools, Bibliotheken, Frameworks – und angestellte sowie selbstständige Entwickler. Alle ziehen auf die eine oder andere Weise ihren Nutzen aus Java und stehen in einem wie auch immer gearteten Abhängigkeitsverhältnis. Würde Java von heute auf morgen zugrunde gehen, würde das für diese Beteiligten einem ökonomischen Fiasko gleichkommen.

JVM und die Sprachen

Java hat mit seiner zunehmenden Verbreitung virtuelle Maschinen salonfähig gemacht. Wurde diesen früher noch nachgesagt, extrem langsam zu sein, führte die Einführung von Hotspot und Just-In-Time-Kompilierung mit adaptiver Optimierung zu einer deutlichen Performance-Steigerung bei Java. Diese Technologie wurde immer weiter verbessert, sodass heutzutage Java-Applikationen nicht mehr unter den Performance-Problemen von früher leiden. Damit wurde das Konzept der virtuellen Maschine etabliert und seit wenigen Jahren entsteht der Trend, weitere Sprachen auf der virtuellen Maschine laufen zu lassen. Frei nach dem Motto „Vor dem

Bytecode sind alle gleich“ können wir uns heute aussuchen, in welcher Sprache wir Programme schreiben. Hoffentlich führt das nie zu einem babylonischen Sprachwirrwarr.

Garbage Collector

Automatische Müllbereinigung – ein tolles Konzept und für mich eines der absolut wichtigsten Dinge, die Java der Welt als funktionierendes Konzept gegeben hat. Kaum jemand, der einmal die wahre Schönheit der Garbage-Collection-Algorithmen zu schätzen gelernt hat, möchte wieder Speicherplatz allokieren, verwalten und freigeben müssen. Auch wenn es mehrere Algorithmen dazu mit verschiedenen Konfigurationen für unterschiedliche Einsatzzwecke gibt, so ist jeder von ihnen besser, als keinen im Einsatz zu haben. Nur höchst selten wird man einen triftigen Grund haben, die Speicherverwaltung in eigene Hände zu nehmen.

Vom Backend zum Client

Obwohl Java mit Applets eigentlich für den Einsatz auf Clients bestimmt war, hatten die Java-Servlets mehr Erfolg und bereiteten unter anderem nach und nach den Weg für Java-Systeme als Backends und Middleware. Es folgten Spezifikationen für Enterprise-Systeme wie EJB, JEE (oder früher auch J2EE). Selbst auf dem Großrechner ist Java verfügbar. Viele große Unternehmen setzen Java-Systeme in ihrer IT ein.

Mobilgeräte

Auf mobilen Geräten (Handys, PDAs etc.) lief Java als Java Micro Edition. Ich mag unrecht haben, aber ich möchte sagen: nicht sonderlich erfolgreich. Doch als Google sein Betriebssystem bzw. die Software-Plattform Android veröffentlichte, gab dies auch Java einen gewaltigen Schub, denn Java-Technik ist der Ursprung wichtiger Bestandteile von Android wie der virtuellen Maschine Dalvik sowie bestimmter Java-Klassenbibliotheken (unter anderem von Apache Harmony, Apache Commons). Wie der noch von Sun geplante Einstieg über die JavaFX-Technologie auf Multimedia-Geräten wie Fernsehern mit eigenem Online-Geschäft zum Erwerb und Download der Applikationen weiterlaufen wird, bleibt nach der Übernahme durch Oracle abzuwarten.

Software-Entwicklungsprozesse

Dass gerade auch große Projekte mit einer Vielzahl an beteiligten Entwicklern mithilfe von Java realisiert werden können, hat Java mit den unzähligen Tools gefördert, die es in der Community gibt. Diese unterstützen beinahe jeden Software-Entwicklungsprozess. Sei es JUnit, um die Softwareartefakte zu testen, Ant, Maven, Ivy, um den Build-Prozess zu steuern und Software-Abhängigkeiten zu verwalten und aufzulösen, CheckStyle, um den Codierungstil der Entwickler zu prüfen, oder CruiseControl, Hudson/Jenkins, um Continuous Integration zu ermöglichen und zu optimieren. Alles ist denkbar, vieles ist machbar. Zum Glück gibt es dann noch Tools für Software-Metriken, um herauszufinden, dass die heutigen Lösungen zu viel Komplexität beherbergen. Aber darauf schaut man ja maximal am Ende eines Projekts.

Java-Community

Durch den großen Erfolg und die Popularität von Java haben es die Java User Groups nicht nur geschafft, immer noch zu bestehen, sondern auch, weiterhin aktiv zu sein. Ob eigenständig oder in der Vereinigung des Interessenverbunds der Java User Groups e.V. versuchen sie, das Wissen über Java zu verbreiten, den Weg, den Java nimmt, zu beobachten und mit Rat und Tat zu kommentieren oder aktiv die Gegenwart und Zukunft mit zu gestalten. Diese verschiedenen Zusammenschlüsse der Java-Interessierten sind das Sprachrohr gegenüber Herstellern, denn ihre Mitglieder sind es letztendlich, die Applikationen auf Basis der Java-Plattform und der damit verbundenen Technologien schaffen – ohne sie wäre es ein sterbender Markt.

Java überall

Java dürfte beinah überall sein, könnte man meinen. Das ist sicher nicht der Fall. Aber auf dem Handy der Person neben uns, auf dem nächstbesten Computer, in all den großen Unternehmen, deren Namen man so kennt, sicher. Wahrscheinlich ist Java involviert, wenn wir einen Versicherungsantrag stellen und dieser seine Bearbeitungskette durchläuft. Java spielt eine Rolle, wenn wir im Laden Punkte sammeln und diese später wieder ausgeben wollen. Java war im Hintergrund tätig, wenn wir einen der neuen EU-Personalausweise bekommen. Firmen basieren auf Java wie auch die Entwickler, die ausschließlich im Java-Bereich tätig sind. Ihre Existenz ist somit stark mit der Zukunft von Java verbunden. Selbst die Leute, die Java nur kennen, weil regelmäßig auf ihrem Desktop ein Fenster aufgeht und „irgend so ein Java ein Update machen will“, kennen den Namen „Java“ und nutzen es eventuell, ohne es zu wissen.

Die Übernahme von Sun hat vielfach Diskussionen darüber ausgelöst, ob man nicht ein „freies“ Java braucht. Es gab Anregungen in der Community, sich endgültig von einem Hersteller zu lösen. Vielleicht wäre dies eine schöne Alternative: Java vollständig frei von der Willkür eines „Besitzers“, bestimmt durch die Community.

Ich weiß nicht, wohin der Weg gehen wird und was die bessere Wahl ist. Aber bei all dem muss man beachten, was auf dem Spiel steht. Und wer alles. Und beides ist nicht wenig. Man überlege sich an dieser Stelle einmal selber, welche Firma direkt Konkurs gehen würde, wenn Java von jetzt auf gleich nicht mehr verwendbar wäre, und welche wirtschaftlichen Schäden dies weiteren Firmen hervorrufen würde.

Oliver Szymanski
oliver.szymanski@source-knights.com
http://www.source-knights.com

Oliver Szymanski (Dipl. Inform., Univ.) ist als Software-Architekt / Entwickler, Berater und Trainer in den Bereichen Java, .NET und Mobile-Development tätig. Parallel dazu arbeitet er als Schriftsteller (siehe auch Seite 53)

„Java muss sich neuen Einsatz-Szenarien wie Cloud und Mobile Computing stellen …“

Für Java aktuell ist Tobias Frech, Beisitzer der Java User Group Stuttgart, im Gespräch mit Dr. Mark Little, Senior Director Engineering und Middleware Engineering bei Red Hat.

Welche Ziele hat Red Hat bei Java?

Dr. Little: Red Hat ist der zweitgrößte Förderer von OpenJDK und wird auch in Zukunft Java sowie die Java Enterprise Edition (EE) unterstützen. Red Hats JBoss-Portfolio verdeutlicht das starke Bekenntnis zur Weiterentwicklung von Java; wir sind mit diesem Portfolio führend im Markt für Java Application Server. Unsere Ziele für die Zukunft bauen alle auf der weiteren Investition in Java und seine Communities.

Bei der zunehmenden Verbreitung von Cloud Computing und der wachsenden Bedeutung von mobilen Endgeräten spielt Java eine entscheidende Rolle. Wir haben vor, auch in diesen beiden Bereichen eine Führungsrolle einzunehmen.

Wie beurteilen Sie die Lage im Enterprise-Java-Markt?

Dr. Little: Es gibt nur wenige große Player in diesem Markt und mit dem JBoss-Portfolio gehört Red Hat zu einem der drei Marktführer. Außer diesen dreien hat kein anderer Hersteller einen Stack, der vollständig genug wäre, alle Anforderungen von Unternehmen hinsichtlich der Breite zu erfüllen. Wettbewerb ist gut, denn Anwender erhalten damit eine bessere Auswahl und gleichzeitig spornt es uns an, Projekte wie beispielsweise JBoss Application Server 7 und Infinispan zu den besten Lösungen ihrer Klasse zu machen.

Dr. Mark Little

Wie ist das Verhältnis zwischen Red Hat und Oracle?

Dr. Little: Unsere Beziehung zu Oracle ist gut. Das bedeutet natürlich nicht, dass wir immer gleicher Meinung sind, aber aufgrund unserer engen Zusammenarbeit mit Oracle und den anderen im JCP beteiligten Partnern glauben wir, dass das Java-Ökosystem noch eine große Zukunft vor sich hat.

Was erwarten Sie von Oracle in Bezug auf Java?

Dr. Little: Wir wünschen uns, dass Oracle mit gutem Beispiel vorangeht und die Java-Communities offener gestaltet, als Sun dies tun konnte. Der neue Java Specification Request JSR-348, bei dem Red Hat eine wichtige Rolle spielt, ist ein Beispiel dafür, wie sich die Situation verbessern kann. Der Java Community Process (JCP) arbeitet innerhalb des JSR-348 daran, die Regeln für die Mitwirkung bei der Weiterentwicklung der Java-Standards neu zu definieren – um diese Regeln offener und nachvollziehbarer zu machen. Unserer Meinung nach ist dies ein guter erster Schritt und wir wünschen uns, dass Oracle ähnliche Initiativen unterstützt.

Welche Chancen sehen Sie für Android in den kommenden Jahren?

Dr. Little: Wie andere auch glauben wir, dass Android einer der dominanten Player im Markt sein wird. Durch die starke Verbindung mit der Java-Welt werden sich große Chancen für Java ergeben, die künftige Mobile-Computing-Landschaft maßgeblich zu beeinflussen. Auch wir bei Red Hat sehen einen starken Schwerpunkt im Mobil-Bereich, wobei wir uns aber nicht exklusiv auf Android konzentrieren.

Was erwartet Red Hat in Zukunft von OpenJDK? Was soll sich bei OpenJDK ändern?

Dr. Little: Wir würden es gern sehen, wenn OpenJDK offener und demokratischer wäre. Wir hoffen, dass die jüngsten Satzungsänderungen durch Oracle kein Zeichen dafür sind, dass sich die Dinge in die entgegengesetzte Richtung entwickeln. Red Hat macht die meisten OpenJDK-Kontributionen nach Oracle, und wir beschäftigen eine ganze Menge Mitarbeiter, die sich nur mit OpenJDK befassen. Wir haben schon seit einigen Jahren ein zum Test Compatibility Kit (TCK) konformes Java, das mit Red Hat Enterprise Linux ausgeliefert wird. Diese Tradition soll fortgesetzt werden und wir unterstützten OpenJDK vorbehaltlos. Red Hat ist fokussiert darauf, sowohl die Spezifikation als auch die Referenz-Implementierung zu stärken und sicherzustellen, dass alle grundlegenden Anforderungen aus dem Markt darin wiederzufinden sind.

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

Dr. Little: Aus unserer Sicht ist es immer eine gute Entscheidung, ein Open-Source-Projekt zur Referenz-Implementierung zu machen.

Was halten Sie von Harmony?

Dr. Little: Harmony war ein guter Ansatz, um eine breitere Masse von Open-Source-Anwendern zu erreichen. Wir waren enttäuscht, dass Oracle es Apache nicht – wie ursprünglich angekündigt – erlaubte, Harmony die benötigten Standard-Edition-Lizenzen auszustellen. Bedauerlicherweise hat Apache daraufhin das JCP Executive Committee verlassen und dadurch erschien Oracle in einem ungünstigen Licht. Wir wollen unseren Teil dazu beitragen, dass sich die Dinge in Zukunft verbessern.

Was erwarten Sie von der Community in Bezug auf Java?

Dr. Little: Java wird auch künftig die dominierende Programmiersprache bleiben. Es gibt einige neue Nicht-Java-Sprachen wie zum Beispiel Scala und Clojure, die auf der JVM laufen können. Vermutlich werden noch mehr Sprachen von dieser Art entstehen, während sich neue, Domänen-spezifische Einsatzbereiche auftun, für die die Java-Sprache nicht unbedingt die beste Lösung ist, aber man die Vorteile der JVM nutzen möchte. Wir denken jedoch nicht, dass eine dieser Sprachen in den nächsten fünf Jahren Java von der Führungsposition verdrängen kann. Wir glauben eher, dass im Ergebnis die Java-Community wachsen wird – indem die Definition der Community nun auch diese Sprachen einschließt.

Wohin sollte Java sich entwickeln?

Dr. Little: Java muss sich neuen Einsatz-Szenarien wie Cloud und Mobile Computing stellen. Die JVM und die Sprachen sollten sich in Bereichen wie Multi-Tenancy, Modularität und Concurrency verbessern. So finden sich beispielsweise heute in allen Notebooks und Servern Mehrkern-Prozessoren, und dennoch ist es für Java-Entwickler nicht einfach, deren Vorteile auch voll auszunutzen. Parallel arbeitende Anwendungen zu entwickeln ist aufwändig und schon seit Jahrzehnten Gegenstand der wissenschaftlichen Forschung und Entwicklung. Es existieren jedoch gute Ansätze dazu, die ihren Weg auch in die JVM finden können. Im Hinblick auf Cloud Computing muss Java kompakter werden – weniger Ressourcen beanspruchen, damit mehr Instanzen auf einer einzelnen Maschine (oder einem Image) laufen können. Ferner muss es möglich sein, dass mehrere Applikationen isoliert auf der gleichen JVM laufen können (Multi-Tenancy).

Tobias Frech hilft Unternehmen als Trainer und Coach, ihre Java-Anwendungen performanter und stabiler zu betreiben. Er ist Boardmitglied der Stuttgarter Java User Group, SENS-Experte und leitet (mit anderen) die SIG JBoss bei der JUGS.

Arquillian

Frederik Mortensen

Der Artikel befasst sich mit Unit-Tests im Enterprise-Application-Bereich unter Verwendung von Arquillian. Dazu werden Quellcode-Beispiele aufgeführt, die Maven als Build-Management-Tool sowie JBoss und Glassfish als mögliche Beispiel-Application-Server verwenden. Grundlegende Vorkenntnisse im Bereich der serverseitigen Java EE-Entwicklung (Enterprise Java Beans EJB3.1) sowie eines Unit-Test-Frameworks sind daher empfehlenswert.

Um den Erfolg und die Notwendigkeit von Arquillian verstehen zu können, muss man sich mit der Geschichte von Java EE im Allgemeinen befassen. Dessen Final-Release wurde am 10. Dezember 2009 veröffentlicht. Betrachtet man die vielen Veränderungen und Verbesserungen, die im Laufe der Versionen Einzug in die Plattform hielten, wird klar, dass vieles aus dem Wunsch heraus entstanden ist, Java noch weiter zu vereinfachen und zu verbessern. Die Praxis hat gezeigt, dass der übermäßige Einsatz von Interfaces (siehe EJB 2.1), Deployment-Deskriptoren und abstrakten Bean-Klassen bald auch belastend und unübersichtlich werden kann und einfach unnötig ist. Nicht zuletzt beeinflusst durch Lösungsansätze wie das Spring-Framework entstand mit Java EE 6 eine leistungsfähige, stabile, aber vor allem auch leicht handhabbare Plattform, die sich sehen lassen kann. Frei nach dem Motto „Convention over Configuration" promoten Java-Champions wie Adam Bien auf Veranstaltungen, wie einfach es ist, Transaktions-Handling, Persistenz, Security oder RESTful-Web-Services einzubinden. Dazu sind oft nur wenige Annotationen erforderlich. Java EE 6 wurde schnell von der Community akzeptiert. Mehr und mehr Anwendungen werden migriert, der TIBOE-Index meint es gut mit Java.

Die Frage nach dem Warum

Dass Java EE 6 sich schnell zu einer Erfolgsgeschichte entwickelt hat, wissen wir nun. Doch ist es perfekt? Es stellt sich die Frage: „Welche Lücke füllt Arquillian?"

Keine professionelle Software wird heute noch ohne eine absichernde Teststrategie auskommen. Idealerweise macht man sich zu Anfang Gedanken über die Schnittstelle und schreibt dann zuerst einen Unit-Test. Das Implementieren der wirklichen Geschäftslogik folgt erst danach. Ist die Entwicklung abgeschlossen, lässt man die Unit-Tests laufen und sieht recht schnell, ob das gewünschte Ergebnis auch erreicht wurde. Grundgedanke dieses Testansatzes war immer: Funktionieren die einzelnen Bestandteile, so funktioniert auch die Software als Ganzes. Eine möglichst hohe Abdeckung von Quellcode mit Tests war und ist wünschenswert. Doch hielt man sich bisher an diesen guten Brauch, merkt man in Enterprise-Anwendungen schnell, dass hier etwas nicht stimmt. Es schleicht sich das Gefühl ein, dass die bisherigen Testmethoden unzureichend sind. Es entsteht ein Code, der nicht ohne Weiteres testbar ist. Themen wie „Dependency Injection" und „Session Beans" machen schnell deutlich, dass mit einfachen Mockup-Objekten hier nicht gearbeitet werden kann. Wo in klassischen Client-Anwendungen einfache Unit-Tests ausreichen, muss man sich in einer verteilten Architektur mit der Testbarkeit sowie dem Zusammenspiel verschiedener Komponenten Gedanken machen. Es bestehen Abhängigkeiten untereinander. Genau hier kommt Arquillian ins Spiel.

Die Einbürgerung der Integration

„Der Begriff „Integrationstest" bezeichnet in der Software-Entwicklung eine aufeinander abgestimmte Reihe von Einzeltests, die dazu dienen, verschiedene voneinander abhängige Komponenten eines komplexen Systems im Zusammenspiel miteinander zu testen. Die erstmals im gemeinsamen Kontext zu testenden Komponenten haben jeweils einen Modultest erfolgreich bestanden und sind für sich isoliert fehlerfrei funktionsfähig" (aus Wikipedia).

Die Referenz-Dokumentation beschreibt Arquillian mit der Überschrift „An integration testing framework for Containers". Es geht also um sogenannte „Integrationstests". Der verwendete Begriff macht deutlich, dass die angewandte Testmethode sich von herkömmlichen Unit-Tests unterscheidet. Im Gegensatz zum Testansatz einzelner Klassen werden hier ganze Abläufe simuliert durchlaufen. Es findet eine Kombination von Modulen statt und die erwähnten Abhängigkeiten entstehen. Die bei Unit-Tests getroffene Annahme, dass das Gesamtbild passt, wenn die Einzeltests funktionieren, trifft hier also nicht zu. Die Kernkompetenz von Integrationstests besteht darin, Erfolg oder Misserfolg im Zusammenspiel der Module zu prüfen. Nimmt man beispielsweise in einer Web-

Hinweis: Die Listings zu diesem Artikel finden Sie unter http://src.ijug.eu/ja/1104/arquillian.zip

anwendung den Vorgang des Registrierens eines neuen Benutzers als Vorlage, so ist dieser Vorgang erst dann erfolgreich abgeschlossen, wenn folgende Bedingungen erfüllt sind:

- Die Benutzereingabe validiert
- Der Benutzer persistiert
- Eine E-Mail an den Benutzer versandt
- Die entsprechenden Rechte gesetzt

Dabei können für das Versenden der Nachricht und das Persistieren des Benutzers schon verschiedene Services beansprucht worden sein. Ist einer dieser Services fehlerhaft oder nicht verfügbar, scheitert die Registrierung im Ganzen. Ein Integrationstest würde hier also bereits mehrere Abhängigkeiten einfordern.

Hello Arquillian

Die graue Theorie klingt vielleicht etwas herausfordernd. Nun der Schritt in die Praxis: Ein Beispielprojekt veranschaulicht, wie man mit Arquillian als Framework sein Ziel recht einfach erreicht. Dabei spielt die fachliche Logik eine eher untergeordnete Rolle. Die angeführten Beispiele sind mehr als technische Vorlage gedacht und zu verstehen. Das Beispiel demonstriert einen ersten Einsatz unter Verwendung von JBoss. Der Applications-Server liegt derzeit als Release-Candidate in Version 7.0.0.CR1 vor, wir nutzen jedoch vorerst noch die 6.0.0.Final-Version. Downloadbar ist er unter http://www.jboss.org/jbossas/downloads. Als IDE wurde NetBeans 7.0 verwendet, die Beispiele sind aber natürlich auch in jeder anderen IDE lauffähig (Maven vorausgesetzt). Sind IDE und Application-Server also installiert, muss ein EJB3 Projekt angelegt werden. Listing 1 zeigt eine Beispiel-POM.

Viele Geheimnisse birgt diese Konfiguration erst einmal nicht. Unter den Eigenschaften (properties-Block) wird die aktuelle Arquillian-Version 1.0.0-Alpha5 als Variable hinterlegt. Demnächst erscheint der Release-Candidate 1. Ist dies der Fall, ist nur in dieser Eigenschaft die Versionsnummer zu ändern, um die neue Abhängigkeit herunterladen zu lassen. Als Nächstes sind die benötigten Repositories zum Download der Bibliotheken angegeben. Darauf folgend sind die Dependencies für das erste Beispiel gelistet. Im vorliegenden Fall wird ein Test mittels des JUnit-Frameworks auf einem vorinstallierten JBossAS per Remote-Aufruf ausgeführt. Daraus ergeben sich die gelisteten, zu verwendenden Abhängigkeiten. Beim ersten Clean und Build des Projekts kann es etwas dauern, bis alle Abhängigkeiten aus dem entsprechenden Repository heruntergeladen worden sind.
Um erfolgreich den Remote-Aufruf absetzen zu können, wird eine weitere Datei benötigt, die wie folgt aussieht und im „src/test/resources"-Ordner abgelegt sein muss:

```
java.naming.factory.initial=org.jnp.interfaces.
NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.
naming:org.jnp.interfaces
java.naming.provider.url=jnp://localhost:1099
```

„Localhost" ist unter der Annahme eingetragen, dass der zu nutzende Applications-Server auf dem Entwicklungs-PC lokal installiert ist. Zuletzt sind noch die zu testende Java-Klasse und deren Testklasse erforderlich, hinterlegbar jeweils unter „src/main/java" beziehungsweise „src/test/java". Um den Fokus erstmal auf Arquillian zu belassen, fällt das Beispiel sehr simpel aus, es werden nur zwei Zahlen addiert und das Ergebnis zurückgegeben (siehe Listings 2 und 3).

Das erste Mal findet hier Arquillian Erwähnung, wenn man es der JUnit-Annotation „@RunWith" als „TestRunner"-Parameter übergibt. Eine Instanz der zu testenden Klasse „IntCounter" wird dann per „@Inject" unter Zuhilfenahme von CDI instanziiert. Richtig spannend wird es mit der Annotation „@Deployment". Hier kommt ein erster Vorgeschmack auf die Arbeitsweise von Arquillian auf. Mittels des „ShrinkWrap" genannten Deployment-Mechanismus (ebenfalls ein Unterprojekt von jboss.org) ist es möglich, programmatisch und voll automatisiert ein .JAR-, .WAR- oder .EAR-Archiv zu erzeugen, um dieses dann in einem eingebetteten Server zu nutzen oder per Serialisierung remote auf den Zielserver zu übertragen. Wichtig hierbei ist, anzugeben, welche Klassen letztendlich im Archiv aufgenommen werden sollen und wie der Archivname lauten soll. In fortgeschrittener Nutzung wird es auch möglich sein, ganze Archive statt einzelner Klassen einzubinden. Sinnvoll ist dies etwa in Maven-Projekten, wo man eventuell sein „domain model" (also alle Entity-Klassen) als Ganzes übergeben möchte. Die ebenfalls angelegte, leere Datei „beans.xml" wird für die Verwendung von CDI benötigt und sollte so übernommen werden. Ein erster Testlauf des anzustoßenden Unit-Tests sollte folgenden Consolen-Output liefern:

```
Tests run: 1, Failures: 0, Errors: 0, Time
elapsed: 7,438 sec
```

Die embedded-Variante mit EJB

Die oben erwähnten 7,438 Sekunden für eine einfache Addition an sich sind vielleicht nicht unbedingt überzeugend. Man sollte jedoch bedenken, dass im Hintergrund unter anderem das Erzeugen und Löschen des Testarchives per „ShrinkWrap" stattfindet sowie der Remote-Aufruf auf dem Application-Server. Wenn wir jetzt auf einen Embedded-Test umstellen, wird sich die Zeit für einen einzelnen Test sogar noch erhöhen, weil das Hochfahren und Beenden des Servers dann noch zur Laufzeit hinzukommt. Allerdings handelt es sich ja, wie bereits erwähnt, um Integrationstests, die im Gegensatz zu Unit-Tests nicht bei jedem Build-Vorgang angestartet werden müssen. Ein Zeitgewinn entsteht dann, wenn in einer Testklasse mehrere Einzeltests hintereinander erfolgen. Dabei ist das Hochfahren des Embedded-Servers natürlich nur einmal am Anfang des ersten Tests nötig und der Server wird nach Durchlaufen des letzten Tests schließlich wieder beendet.

Ein zweites ähnliches Beispiel zeigt die vollen Vorzüge Arquillians. Denn wer möchte schon dauerhaft einen Testserver laufen lassen, wenn Arquillian das Starten, Deployen und Stoppen des Servers komplett für einen übernehmen kann. Versprochen wurde auch bereits, dass EJBs (allein und als Abhängigkeiten untereinander) ebenfalls testbar sind.

Ziel dieses Beispiels ist das Einbinden von EJB3 sowie der Einsatz von Glassfish im Embedded-Modus. Wichtig hierbei ist, dass im Gegensatz zum vorherigen Beispiel keine „jndi.properties"-Datei angelegt werden darf. Listing 4 zeigt die POM.

Wie man sehen kann, ist ein neues Repository hinzugekommen. Grund ist die neue, benötigte Abhängigkeit zu Glassfish. Neben der POM-Datei werden nun noch die EJB- und die Test-Klasse erstellt (siehe Listings 5 und 6). Der angelegte TimeService gibt die aktuelle Jahreszahl als String zurück. Die Annotation „@EJB“ wurde – wie bei Session-Beans üblich – dazu verwendet, um den Service mittels Dependency Injection zu laden. Der schlanke Quellcode lässt erahnen, wie viel Arbeit einem Arquillian erspart und übernimmt. Ist man durch diese ersten Beispiele neugierig geworden, findet man unter https://github.com/arquillian/arquillian-exampleshttps://github.com/arquillian/arquillian-examples weiterführende, eingecheckte Quellcode-Beispiele.

Die Einfachheit der Tests bedeutet keinesfalls, dass man als Entwickler hiermit schon an die Grenzen des Machbaren stößt. Mittels hinterlegbarer Konfigurationsdateien wie „arquillian.xml“ oder „domain.xml“ (unter GlassFish) kann man noch einiges an Feinjustierung vornehmen.

Fazit

Arquillian hat eine vernünftige Basis geschaffen. Der Einsatz ist durchaus auch in größeren, produktiven Umgebungen möglich. Trotzdem sollte man natürlich beachten, dass sich das Produkt noch in der Alpha-Phase befindet. Erfreulich ist es daher, zu hören, dass ein erster Release-Candidate schon in Arbeit ist. Die aufgezeigten Quellcodes können natürlich nur einen Vorgeschmack auf das liefern, was Arquillian an Funktionalität bietet. Es lohnt sich, tiefer in die Materie einzudringen, indem man sich die Referenz-Dokumentation näher ansieht. Neben diversen Konfigurationsmöglichkeiten für die möglichen Applikations-Server findet man dort Informationen über Remote-Debugging, erweiterte Anwendungsfälle, Hintergründe und Tipps zu ShrinkWrap, dem Erstellen von Archiven, Hinterlegen von DataSources und mehr. Das Ausprobieren lohnt sich. Die volle Unterstützung von JPA2 beispielsweise in Unit-Tests zu sehen, lässt einen den anfänglichen Einarbeitungsaufwand schnell vergessen. Es wäre schön, wenn dieser Artikel einen ersten Anreiz zum Umdenken in so manchem Projekt führen könnte, und es bleibt zu hoffen, dass Arquillian weiterhin Anwendung findet.

Weiterführende Quellen

http://docs.jboss.org/arquillian/reference/1.0.0.Alpha5/en-US/html/
http://de.wikipedia.org/wiki/Integrationstest

Frederik Mortensen
mortensen.frederik@googlemail.com

Frederik Mortensen ist gelernter Anwendungsentwickler, Sun-zertifiziert (SCJP) und seit zehn Jahren im Bereich der Java-Entwicklung in verschiedenen Tätigkeitsfeldern aktiv. Derzeit ist er mitverantwortlich für die Weiterentwicklung des Java-Backends einer Domain-Verwaltungssoftware der Firma InternetX in Regensburg. In seiner Freizeit beschäftigt er sich neben der Entwicklung von Web-Anwendungen auf JSF2- und EJB3.1-Basis auch mit der Entwicklung eines Open-Source-Projekts im Java-3D-Bereich.

Suchen mit Apache Solr

Peter Karich, Pannous GmbH

Apache Solr ist ein auf der Volltextsuchbibliothek Lucene aufsetzender Suchserver und konnte sich aufgrund der leichten Bedienbarkeit und guten Performance neben kommerziellen Produkten einen festen Platz in der Enterprise-Suche sichern. Dieser Artikel führt in die Grundlagen zum Aufsetzen und Benutzen von Apache Solr ein und zeigt verschiedene Anwendungsmöglichkeiten auf.

Jeder kennt Beispiele von Webseiten mit einer schlechten oder langsamen Suche. Sie alle zeugen davon, dass beim Umsetzen der Suche viele Fehler gemacht werden können und auch gemacht wurden. Viele Webseiten- oder Shopbetreiber setzen dann auf Google, jedoch ist dies nicht immer die beste Lösung und meist auch nicht die wirtschaftlichste. Wie im Anhang aufgelistet, gibt es mittlerweile viele Open-Source-Projekte die sich dem Problem der Suche angenommen haben. Apache Lucene [1] war eines der ersten dieser Projekte und wurde im Jahr 2000 durch Doug Cutting auf Sourceforge veröffentlich und ist später zu einem Top-Level Apache-Projekt geworden. Der Suchserver Solr [2] wurde 2004 von Yonik Seeley entwickelt und 2006 ein Tochterprojekt von Lucene. Mit Apache Nutch [3] steht gleichzeitig ein mächtiger Webcrawler zur Verfügung, der mit Solr gut zusammenarbeitet.

Eine Einführung

Apache Solr ist ein Open-Source-Suchserver, der auf der Volltext-Suchbibliothek Lucene basiert. Solr ist also vereinfacht gesprochen ein HTTP-Proxy, mit dem ein invertierter Index in Form von Lucene angesprochen wird. Einen invertierten Index kann man sich als Liste von Wörtern vorstellen, die auf die Dokumente verweisen in denen sie vorkommen, zum Beispiel ist jeder Wortindex am Ende eines Buchs ein solcher invertierter Index. Durch diese Speicherungsart können für eine einfache Anfrage die passenden Dokumente sofort – also in O(1) – zurückgegeben werden, ohne dass die Dokumente durchsucht werden müssen. Damit dieser invertierte Index aufgebaut werden kann, muss jedes Dokument den Indizierungsprozess durchgehen.

Wann sollte nun Solr eingesetzt werden und wann Lucene? Meist wird man sich für Solr entscheiden, da einem mit Solr der Anfang leichter gemacht wird, Facetten unterstützt werden (dazu später mehr) und man von Java unabhängiger ist. Es existieren Open-Source-Bibliotheken, sogenannte „Clients“ für alle gängigen Programmiersprachen für Solr. Einer der ersten Schritte sollte das offizielle Tutorial [4] sein, dass die wichtigsten Schritte einer jeden Suche abdeckt:

1. Installation des JDK (~70MB) und Solr (~80MB)
2. Indizieren von Dokumenten: Füge Daten beliebiger Form, etwa über http, hinzu. Andere Möglichkeiten: JSON, XML oder CSV über Java direkt oder über eine SQL-Datenbank
3. Abfragen von Dokumenten: Frage den Suchserver mit einem erweiterten Lucene-Query an, um die jeweils relevantesten Dokumente zu bekommen.

Im Produktionsbetrieb ist noch ein geeigneter Webserver auszuwählen. Falls dieser Tomcat ist, sollte unbedingt „UTF-8“ als URI-Encoding in der Datei „conf/server.xml“ eingetragen sein. Dazu wird im korrekten Connector-Element ein Attribut hinzugefügt: URIEncoding=“UTF-8“.

Indizieren von Dokumenten

Im offiziellen Tutorial werden XML-Dokumente über die HTTP-Schnittstelle in den Solr-Index geschrieben. Dieser Prozess heißt im Englischen „Indexing“ oder „Feeding“. Es gibt noch andere Möglichkeiten, Dokumente zu indizieren:

- Mit dem Data Import Handler (DIH) eine sprachunabhängige Lösung umsetzen
- Sprachabhängige Lösungen für Java (SolrJ), Python etc.

Vor dem Indizieren ist genau zu überlegen, welche Felder durchsuchbar gemacht werden sollen, und auch, wie diese gesucht werden können. Wenn ein Feld als HTML vorliegt, dann sollte man zunächst folgende Aktionen durchführen:

1. Irrelevante Zeichen entfernen
2. Das Feld in einzelne durchsuchbare Wörter (sogenannte „Terme“) aufteilen, beispielsweise mit dem Tokenizer „WhitespaceTokenizerFactory“

3. Die Terme in Kleinschreibung transformieren, etwa mit dem Filter „LowerCaseFilterFactory“
4. Außerdem können die Terme noch auf ihren Wortstamm gekürzt werden, um die Trefferquote zu erhöhen. Dies kann mit einem sprachspezifischen Filter erfolgen [5].

Der letzte Schritt ist mit Bedacht durchzuführen, da er stark von der Sprache abhängt und meist zur Verringerung der Relevanz, jedoch zur Vergrößerung der Trefferanzahl führt. Weiterhin sei erwähnt, dass das Projekt „Apache Tika“ [6] das Indexieren von HTML und anderen Dokumenten wie PDF vereinfacht und andere interessante Eigenschaften wie Sprachermittlung bietet.

In der „schema.xml“-Datei von Solr finden sich noch weitere Standardtypen wie „text“, „string“ oder „float“. „string“- und „float“-Felder werden nicht tokenisiert und sind damit zum Beispiel für Filter in der Anfrage geeignet.

Beim Design eines Suchindizes ist zu beachten, dass das betreffende Dokument, das gesucht wird, auch indiziert ist. Wenn beispielsweise Hotelzimmer verschiedener Hotels durchsuchbar gemacht werden sollen, so darf nicht jedes Hotel als ein Dokument aufgefasst werden, sondern die Daten sind zu denormalisieren und jedes Zimmer muss für sich indiziert und noch mit einem string-Feld versehen sein, in dem der Hotelname gespeichert ist. Es wird also alles Wichtige in einem Dokument untergebracht, auch wenn dabei Daten doppelt gespeichert sind – in diesem Beispiel die Hotelnamen.

Abfragen von Dokumenten

Es ist sehr leicht, eine Abfrage für Solr zu erstellen und zu testen: Einfach eine URL in den Browser eingeben und das Resultat direkt im Browser als XML anschauen (siehe Listing 1). Falls Firefox benutzt wird, können auch unwichtige Elemente eingeklappt werden:
http://localhost:8983/solr/select?q=car.

Die Ausgabe kann auch über das Velocity-Plugin erfolgen. Damit ist es seit der Version 3.1. möglich, direkt in Solr das Ergebnis als HTML zu rendern und so einen schnellen Prototypen zu entwickeln. Die

```
−<response>
 −<lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">13</int>
   −<lst name="params">
      <str name="q">car</str>
    </lst>
  </lst>
 −<result name="response" numFound="2" start="0">
  −<doc>
   −<arr name="cat">
      <str>electronics</str>
      <str>connector</str>
    </arr>
   −<arr name="features">
      <str>car power adapter, white</str>
    </arr>
    <str name="id">F8V7067-APL-KIT</str>
    <bool name="inStock">false</bool>
    <str name="manu">Belkin</str>
    <date name="manufacturedate_dt">2005-08-01T16:30:25Z</date>
    <str name="name">Belkin Mobile Power Cord for iPod w/ Dock</str>
    <int name="popularity">1</int>
    <float name="price">19.95</float>
    <str name="store">45.17614,-93.87341</str>
    <float name="weight">4.0</float>
  </doc>
```

Listing 1: Ausschnitt der HTTP-Antwort als XML

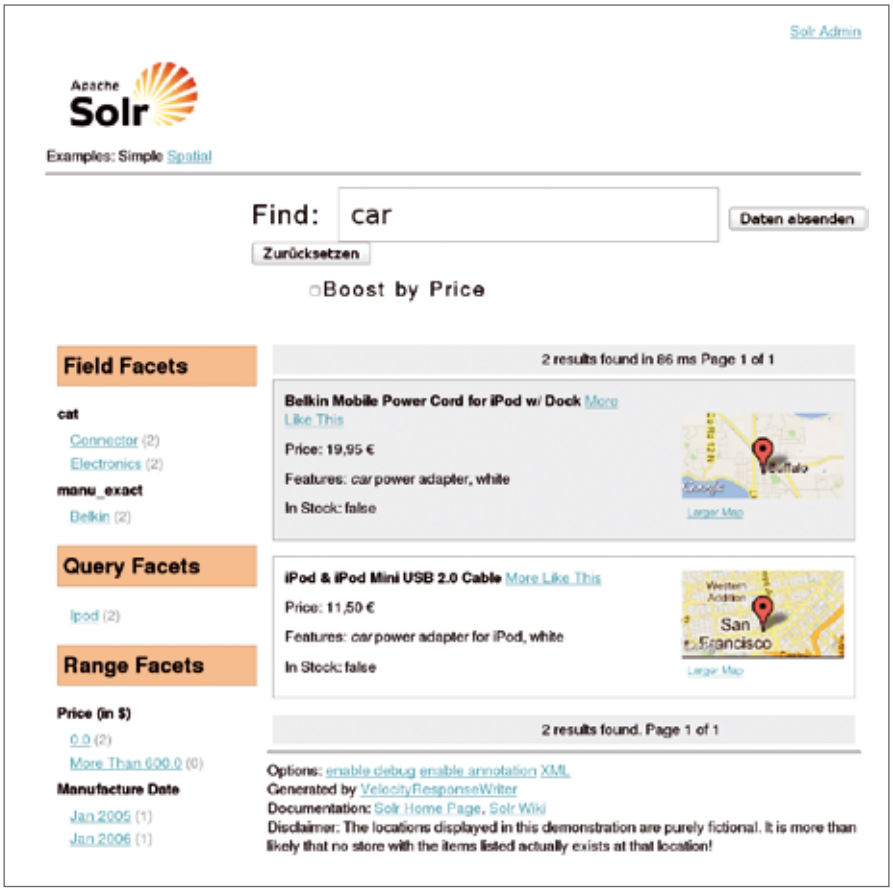


Abbildung 1: HTTP-Antwort in der Standard HTML-Ansicht

Anfrage ist hier nur leicht verschieden:
http://localhost:8983/solr/browse?q=car

Als Anfragemodule seien hier der Dismax-Queryhandler (oder auch edismax) und der Standard-Queryhandler erwähnt. Der Dismax-Queryhandler ist besser für direkte User-Eingaben geeignet und erleichtert Mehrfeldanfragen, wo Felder wie „Titel“ und „Text“ verschiedene Gewichtung erfahren sollen:

```
q=superman&fq=type:book&sort=price asc&rows=10&start= 0
```

q (query): Es wird nach ‚superman' gesucht
fq (filter query): Als Typ sind nur Bücher zugelassen
sort : Der kleinste Preis soll zuerst erscheinen
rows, start: Anzahl der Dokumente und Startdokument zum Paginieren

Es sind viele andere Anfragekonzepte möglich, eines davon ist „Boosting". Um beispielsweise Dokumente, die ‚superman' im Titel enthalten, zu bevorzugen (boost von 2), kann man es mit folgender Anfrage versuchen:

```
q=title:superman^2 OR subject:superman
```

Diese Anfrage kann mit dem Dismax-Queryhandler vereinfacht werden:

```
q=superman&qf=title^2 subject
```

Eine weitere mächtige Anfragemöglichkeit ist das „Faceting", die im Velocity-Beispiel per default schon eingeschaltet ist. Facets ermöglichen es dem Entwickler, die Anzahl der Dokumente zu bestimmen, die unter verschiedenen Filtern vorzufinden wären, und das gleichzeitig zur eigentlichen Anfrage. Ein naheliegendes Beispiel ist, die verfügbaren Produkttypen für eine Anfrage zu bestimmen:

```
http://localhost:8983/solr/
select?q=car&facet=true&facet.field=cat
```

Listing 2 zeigt im XML ein Teilstück für die Facets (links).

Die nächste Anfrage bezieht die Filterquery „fq=cat:connector" ein, die resultierende Dokumentenzahl liegt dann bei „2". Abbildung 2 zeigt, wie mithilfe von Facets und Filterqueries eine Navigation im Kategoriebaum implementiert wurde.

Abbildung 2: Navigation durch Kategoriebaum mit Hilfe von Facets

Es gibt noch folgende weitere Anfragekonzepte:

- Fuzzy search ermöglicht es auch, Dokumente mit nur „ähnlichen" Wörtern herauszufinden, wobei die Ähnlichkeit z.B. auf dem Levenshtein-Abstand basiert sein kann, was gut bei Rechtschreibfehlern von Benutzern ist.
- Results-Grouping oder Field-Collapsing ermöglicht es, die Dokumente basierend auf einem ihrer Felder in Gruppen einzuordnen. Dies kann in einer Websuche benutzt werden, um nur Webseiten von verschiedenen Domains anzuzeigen.

```
-<lst name="facet_counts">
   <lst name="facet_queries"/>
  -<lst name="facet_fields">
   -<lst name="cat">
      <int name="connector">2</int>
      <int name="electronics">2</int>
      <int name="camera">0</int>
      <int name="copier">0</int>
      <int name="graphics card">0</int>
      <int name="hard drive">0</int>
      <int name="memory">0</int>
      <int name="monitor">0</int>
      <int name="multifunction printer">0</int>
      <int name="music">0</int>
      <int name="printer">0</int>
      <int name="scanner">0</int>
      <int name="search">0</int>
      <int name="software">0</int>
    </lst>
   </lst>
   <lst name="facet_dates"/>
   <lst name="facet_ranges"/>
 </lst>
```

```
-<lst name="facet_counts">
   <lst name="facet_queries"/>
  -<lst name="facet_fields">
   -<lst name="cat">
      <int name="connector">2</int>
      <int name="electronics">2</int>
    </lst>
   </lst>
   <lst name="facet_dates"/>
   <lst name="facet_ranges"/>
 </lst>
```

Listing 2: Links das XML-Teilstück der Facets, rechts die gleiche Anfrage um den Parameter „facet.mincount=1" erweitert

Drupal-Integration

Das Drupal-Solr-Modul zeigt, wie gut Solr auch in Java-fremde Projekte integriert werden kann. Generell stehen bei einer Integration die drei verschiedenen APIs zur Verfügung (HTTP, JSON, XML). Auch andere PHP Projekte wie Typo3, Magento und Oxid besitzen ein Solr-Suchmodul.

Autovervollständigung

Für die Implementierung einer Autovervollständigung gibt es hauptsächlich zwei Ansätze: Der erste geht über Facets und spezielle Prefix-Facet-Filter. Beim zweiten Ansatz wird eine NGramFilterFactory beim Indexieren des Feldes verwendet.

Spatiale Suche

Um die spatiale Suche in Solr zu nutzen, muss ein Feld mit den Koordinaten (Longitude, Latitude) in jedem Dokument existieren:

```
<field name="position">45.17614,-93.8734
1</field>
```

Ein Geo-Filterquery, der im Umkreis des Punktes „45.15,-93.85" und einem Radius von 5 km sucht, benutzt nun die spezielle Syntax mit der lokalen Variablen „geofilt":

```
&fq={!geofilt pt=45.15,-93.85 sfield=position
d=5}
```

Anstatt der „geofilt"-Variablen kann auch die meist schnellere Variante „Boundingbox" mit „bbox" genutzt werden.

DuckDuckGo

DuckDuckGo ist eine alternative Web-Suchmaschine, bei der viele Open-Source-Werkzeuge zum Einsatz kommen. Solr hilft dabei, das sogenannte „zero click"-Feature mittels Dismax-Query-Handler umzusetzen. Gabriel Weinberg, der Gründer von

DuckDuckGo, hat Solr dafür ein wenig trainieren müssen:

1. Tunen des Dismax-Handlers und des Schemas
2. Eigener Relevanzfilter, der unter Umständen die Ergebnisse komplett unterdrückt

Clustering mit Carrot2

Carrot2 [7] ist ein Solr-Plugin und unterstützt Clustering, auch Cluster-Analyse genannt: „Unter Cluster-Analyse versteht man Verfahren zur Struktur-Entdeckung in Datenbeständen. Die so gefundenen Gruppen von „ähnlichen" Objekten werden als Cluster bezeichnet, die Gruppenzuordnung als Clustering.", so Wikipedia. Dies ist nicht zu verwechseln mit Klassifikationsverfahren, wo die Gruppen schon vorher feststehen. Im Solr-Tutorial kann man dieses Plugin sehr leicht ausprobieren. Einfach Solr wie folgt starten:

```
java -Dsolr.clustering.enabled=true -jar start.
jar
```

Dann bekommen die Ergebnisse in der Velocity-Ansicht in der linken Spalte den Eintrag „Clusters" (siehe Abbildung 3). Diesmal werden alle Dokumente einbezogen:

```
http://localhost:8983/solr/browse?q=*
```

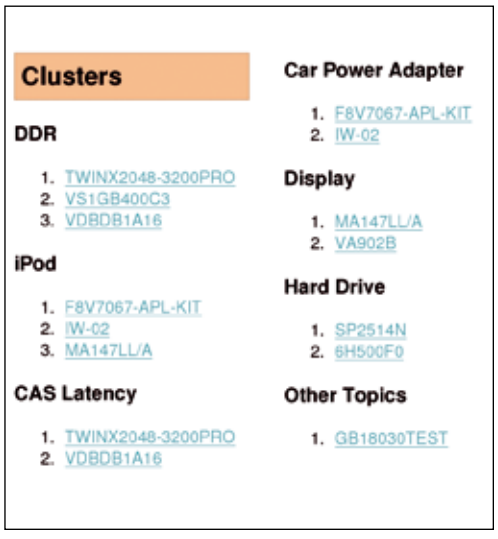


Abbildung 3: Clustering-Eintrag der Velocity-Ansicht

NoSQL-Lösung

Solr lässt sich als Haupt-Datenspeicher nutzen. Dies hat den Vorteil, dass nicht zwei parallele Systeme gepflegt und synchronisiert werden müssen. Dabei bestehen allerdings noch folgende Einschränkungen, etwa im Vergleich zu ElasticSearch:

1. Solr ist momentan noch nicht (Near) Real-Time. Damit ein Dokument direkt nach dem Indexieren abfragbar ist, ist eine teure Refresh-Operation erforderlich.
2. Der Index kann korrupt werden und ein Re-Indexieren wäre unmöglich. Dadurch ist immer ein Backup anzuraten oder zumindest ein einfach zu installierendes Replikat (siehe „Replication" in der Dokumentation).
3. Aber auch mit dieser Limitation wurde Solr erfolgreich im Produktiv-Einsatz als Datenbank-Ersatz benutzt.

Performance

Um die Suchgeschwindigkeit zu steigern, kann Solr den Lucene-Index komplett im RAM vorhalten, was nicht nur im Falle einer Autovervollständigung nützlich ist. Ein weiterer Performance-Gewinn wird durch Autowarming-Queries erreicht. Dabei sendet Solr automatisch Anfragen, sobald ein neuer Index-Leseprozess startet. So ist der Cache gefüllt und die ersten Benutzer landen nicht in einem Timeout.

Fazit

Solr hat sich als bewährte Suchlösung im Enterprise-Bereich etabliert. Die Anwendungsgebiete sind vielfältig und gehen manchmal sogar über das reine Suchen hinaus. Daher ist es für jeden Programmierer und Software-Architekten heutzutage unerlässlich, Solr oder ähnliche Suchlösungen zu beherrschen.

Es gibt auch auf dem Markt der Open-Source-Suchlösungen viele Projekte, die sich ein ähnliches Ziel wie Solr oder Lucene gesetzt haben:

- ElasticSearch
  Ein direkter Konkurrent von Apache Solr. Ein Suchserver, basierend auf Lucene, der das Suchen besser skalierbar und leichter mandantenfähig macht – Apache Lizenz 2.
- Open search server
  Komplettes Software-Paket für Suchlösung, inklusive Web-Crawler und Suchserver, der auf Lucene basiert – GPLv3 Lizenz.
- Sphinx
  Bibliothek zur Volltextsuche, wird meist in Verbindung mit MySQL benutzt – GPLv2 Lizenz.
- Xapian
  Bibliothek zur Volltextsuche – GPLv2 Lizenz.
- Search, Network, Analytics
  Basierend auf Lucene, kein Suchserver, aber mit Zoie eine interessante Realtime-Erweiterung für Lucene – Apache Lizenz 2.

Referenzen

[1] http://lucene.apache.org
[2] http://lucene.apache.org/solr/
[3] http://nutch.apache.org/
[4] http://lucene.apache.org/solr/tutorial.html
[5] http://wiki.apache.org/solr/LanguageAnalysis#German
[6] http://tika.apache.org/
[7] http://project.carrot2.org/

Peter Karich
pkarich@pannous.info

Peter Karich ist Diplom-Physiker und arbeitete seit April 2008 zunächst freiberuflich bei Esemos, Pannous und Indv GmbH (Enterprise Suche, Web-Services etc.) sowie in einer Halbtagsstelle als wissenschaftlicher Mitarbeiter an der Universität Bayreuth (Datenbanken, Prozesse). Seit Mai 2009 ist er festangestellt bei der Pannous GmbH (Enterprise Suche mit Solr, Lucene und ElasticSearch). Sein Blog ist unter http://karussell.wordpress.com/.

Michael Hüttermann, Java User Group Köln

„Ich denke, die Java-Community ist wie eine große Familie …“

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

Wie bist du zur Java User Group Köln gekommen?

Michael Hüttermann: Ich bin vor etwa zehn Jahren nach Köln gezogen. Die Suche nach einer regionalen aktiven Java-Community lieferte keine Ergebnisse. Deshalb startete ich mit der Organisation monatlicher Stammtische und später mit abendlichen Vortragsveranstaltungen. Am Anfang saß ich beim Stammtisch auch mal allein da, jetzt nach all den Jahren ist es eine vitale Community.

Wie ist die Java User Group Köln organisiert?

Michael Hüttermann: Die leichtgewichtige Organisation sieht so aus, dass ich die Events organisiere, und dabei aus der Community immer wieder recht unterschiedliche Unterstützung bekomme.

Was zeichnet die Java User Group Köln aus?

Michael Hüttermann: Die Java User Group Köln zeichnet sich durch Vitalität aus sowie eine schlanke Organisation von hochkarätig besetzten, kostenlos besuchbaren Events. Die besondere Zielvorstellung ist dabei primär die Mischung, weltweit bekannte, in ihren Domänen führende Experten für Beiträge nach Köln zu locken, und regionalen Experten, oder solchen, die es werden wollen, eine Plattform zu bieten.

Wie viele Abendveranstaltungen gibt es pro Jahr?

Michael Hüttermann: Das ist recht unterschiedlich. Zu Stoßzeiten waren es phasenweise zwei hochkarätige Abendveranstaltungen im Monat. Mittlerweile halte ich für die JUG Köln etwa acht Vortrags-Events im Jahr für optimal, wobei an einem Abend nicht selten mehrere Beiträge Platz finden.

Was motiviert dich besonders, als Vorstand die Java User Group Köln zu führen?

Michael Hüttermann: Es ist der besondere Reiz, der Community etwas zurückzugeben. Community-Arbeit bedeutet, selbst zu investieren, vornehmlich Zeit, ohne dafür eine Gegenleistung zu erwarten.

Was bedeutet Java für dich?

Michael Hüttermann: Java ist die führende Plattform und Sprache sowie ein umfassendes Ökosystem. Auch wenn in Projekten nicht selten auch andere Sprachen und Plattformen eine Rolle spielen oder zumindest Artefakt-Typen wie Skripte, Datenbankelemente, ist Java doch häufig die führende Plattform. Wenn man so wie ich Tag und Nacht mit Java unterwegs ist beziehungsweise sein Hobby zum Beruf gemacht hat, nimmt Java natürlich eine dominante Stellung ein.

Welchen Stellenwert besitzt die Java-Community für dich?

Michael Hüttermann: Eine große. Ich denke, die Java-Community ist wie eine große Familie.

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

Michael Hüttermann: Es hat mich gewundert, warum es so lange dauerte, bis ein adäquates Unternehmen Sun übernommen hat. Schließlich hat Sun mit Java noch nie wirklich Geld verdient, das Fehlen eines Business Case war meiner Meinung nach schließlich der Grund, warum die Selbstständigkeit verlorenging. Ferner ist Java ein großes Asset, das für viele Unternehmen eine interessante Beimischung darstellt, so auch für Oracle, schließlich basieren Oracle-eigene Produkte auf Java, gerade im Bereich der Middleware.

Wie sollte sich Java weiterentwickeln?

Michael Hüttermann: Ich denke, wir sind auf dem richtigen Weg und haben mittlerweile von der .NET-Welt erfolgreich adaptiert, Java mehr als Plattform zu begreifen, denn als Sprache. Das Spannungsfeld zwischen zu viel Innovation und zu wenig, siehe die Closure-Debatte, tut unterm Strich der Weiterentwicklung von Java gut. Mit

Java 7 steht nun ein spannendes neues Release zur Verfügung. Mehr noch als einzelne neue Features für Java 8 wären in meinen Augen eine Stabilisierung einer zügigeren Release-Taktung und insgesamt noch mehr Transparenz zielführend.

Wie sollte Oracle deiner Meinung nach mit Java umgehen?

Michael Hüttermann: Oracle sollte Java weiter vorantreiben, was das Unternehmen bereits erfolgreich tut. Verbesserung bei der Transparenz ist auf dem Weg, siehe JCP, auch die Weiterentwicklung des JCP ist ein Prozess. Da Oracle Sun kaufte, und somit auch Java und Patente, darf man sich über eine entschlossene Verfolgung von Interessen nicht wundern, siehe die Google-Patent-Debatte oder die weiterhin sperrige TCK-Situation.

Wie sollte sich die Community gegenüber Oracle verhalten?

Michael Hüttermann: Geduldig. Oracle ist offen für Anregungen und Verbesserungen. Alle sollten sich einbringen in eine konstruktive Diskussion. Beispielsweise steht eine Mitarbeit im JCP jedem offen.

Michael Hüttermann
michael@huettermann.net

Zur Person: Michael Hüttermann

Dipl.-Wirt.-Inf. Michael Hüttermann (Java Champion, SCJA, SCJP, SCJD, SCWCD) ist freiberuflicher Entwickler, Architekt, Berater, Coach, Autor und Dozent für Java/JEE, ALM/SCM und agile Softwareentwicklung. Zur proaktiven Verfolgung seiner Work-Life-Balance schreibt er Bücher und spricht auf Konferenzen. Weitere Infos unter http://huettermann.net.

iJUG Verbund

Die iJUG-Mitglieder auf einen Blick

Java User Group Deutschland e.V.
http://www.java.de

DOAG Deutsche ORACLE Anwendergruppe e. V.
http://www.doag.org

Java User Group Stuttgart e.V. (JUGS)
http://www.jugs.de

Java User Group Köln
http://www.jugcologne.eu

Java User Group München (JUGM)
http://www.jugm.de

Java User Group Metropolregion Nürnberg
http://www.source-knights.com

Java User Group Ostfalen
http://www.jug-ostfalen.de

Java User Group Saxony
http://www.jugsaxony.org

Sun User Group Deutschland e.V.
http://www.sugd.de

Swiss Oracle User Group (SOUG)
http://www.soug.ch

Der iJUG möchte alle Java-Usergroups unter einem Dach zu vereinen. So können sich alle interessierten Java-Usergroups in Deutschland, Österreich und der Schweiz, die sich für den Verbund interessieren und ihm beitreten möchten, gerne beim iJUG unter office@ijug.eu melden.

Android – Java macht mobil

Andreas Flügge, object systems GmbH

Trotz des schwelenden Patentstreits zwischen Oracle und Google erfreut sich die Plattform Android einer weiterhin stark wachsenden Verbreitung. Im Juni 2011 bezifferte Google die Anzahl der täglich aktivierten Android-Mobiltelefone auf 350.000 Stück. Gartner prognostiziert, dass 2015 jedes zweite Smartphone unter Android laufen wird. Ähnlich schnell wächst auch der Android-Market, der Mitte 2011 bereits drei Milliarden Downloads verzeichnen konnte. Zu diesem Zeitpunkt waren dort mehr als 200.000 Anwendungen verfügbar. Bei den kostenlosen Apps ist das Angebot mittlerweile größer als in Apples App Store. Die vielen angekündigten Android-Tablets werden einen weiteren Schub auslösen. Es lohnt sich also, einen Blick hinter die Kulissen von Android zu werfen.

Als der Finne Linus Torvalds 1991 die erste Version des Linux-Kernels im Internet veröffentlichte, hatte er gewiss keine Mobiltelefone oder Tablet-Computer als Zielplattform im Auge. Die Portabilität, die der Linux-Kernel im Laufe der Zeit erlangte, und die Entwicklung kleiner, stromsparender, aber trotzdem leistungsfähiger Prozessoren, schafften im Laufe der Zeit die Voraussetzungen, um Systeme wie Android auf eine Vielzahl von modernen mobilen Geräten zu bringen. Etwa zum gleichen Zeitpunkt begann ein kleines Team um James Gosling im Auftrag von Sun Microsystems mit der Entwicklung einer Betriebsumgebung mit eigener Programmiersprache, die für unterschiedliche Anwendungszwecke geeignet sein sollte.
Im Gegensatz zu Linux sollte das System unter anderem Kleingeräte, angeblich sogar Kaffeemaschinen steuern. Da der Name „Oak" (Eiche), der James Gosling wegen des Baumes vor seinem Bürofenster für das System vorschwebte, rechtliche Schwierigkeiten bereitete, einigte sich das Team schließlich auf „Java", in Anlehnung an die Kaffeesorte, die für ihren bevorzugten Espresso verwendet wurde. Im Mai 1995 wurde die erste Java-Version offiziell der Öffentlichkeit vorgestellt. Zehn Jahre später, nachdem Java sich längst besonders im Enterprise-Umfeld etabliert hatte, übernahm Google im Jahr 2005 die auf Software für Mobiltelefone spezialisierte Firma Android.

Nur kurze Zeit danach kündigte Google 2007 an, zusammen mit den Mitgliedern der Open Handset Alliance (darunter HTC, Motorola und Samsung) ein Betriebssystem mit einer Java-Laufzeitumgebung für mobile Geräte zu entwickeln. Im Oktober 2008 war die erste Version von Android offiziell verfügbar.

Bis zur heute aktuellen Version 2.3 („Gingerbread") beziehungsweise der für Tablets optimierten Version 3.0 („Honeycomb") ist aus Android mit hohem Innovationstempo ein produktives, stabiles System entstanden. Seit Version 1.5 („Cupcake") tragen die Versionen den Namen einer Süßspeise.

Architektur

Die Grundlage für Android bildet ein Linux Kernel der Version 2.6 mit den Kernfunktionen unter anderem für das Speicher- und Prozessmanagement, den Netzwerkstack und dem Gerätetreibermodell. Der Kernel abstrahiert von den Hardwaredetails und implementiert Optimierungen für die bei mobilen Geräten besonders wichtigen Aspekte, wie Ressourcen-Management und Energieverbrauch (siehe Abbildung 1).

Jede Android-Applikation wird in einer speziellen Laufzeitumgebung zur Ausführung gebracht. Diese Umgebung besteht aus zwei wesentlichen Teilen: den Core

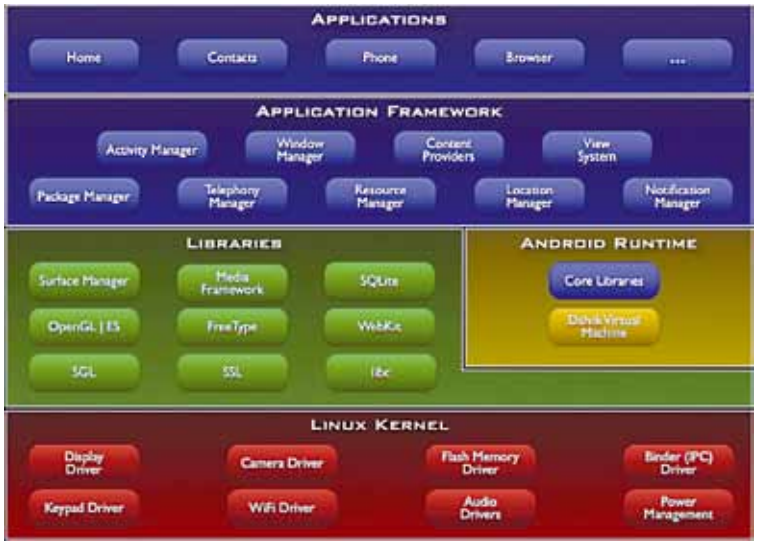


Abbildung 1: Die Android-Architektur (Quelle: Google)

Libraries und der virtuellen Maschine (Dalvik-VM). Während die Core Libraries sprachspezifische Java-Bibliotheken bereitstellen, kommt der Dalvik-VM eine besondere Bedeutung zu, auf die später noch genauer eingegangen wird.

Neben der Android-Laufzeitumgebung existieren die Standard-Bibliotheken direkt unterhalb des Application Frameworks. Sie sind für unterschiedliche Funktionalitäten wie die Darstellung und Verarbeitung von Multimedia-Inhalten und 3D-Grafiken, Datenbankzugriffe, Webfunktionen oder Security-Aspekte zuständig. Sie sind zu großen Teilen in C/C++ implementiert und werden von einer Android-Anwendung in der Regel nicht direkt angesprochen. Das Application Framework abstrahiert von der Laufzeitumgebung und den Standard-Bibliotheken und bildet den Anwendungsrahmen, auf dem jede Android Applikation aufgebaut wird.

Dalvik-VM

Die Dalvik-VM für Android ist das Gegenstück der Standard-JVM, wie man sie aus dem Java Runtime Environment kennt. Sie bringt den aus Java-Quellcode übersetzten Bytecode zur Ausführung. Warum ist aber eine spezielle virtuelle Maschine notwendig? Dafür gibt es im Wesentlichen zwei Gründe.

Die virtuelle Maschine muss in einem mobilen Gerät dafür Sorge tragen, dass dessen Ressourcen möglichst effizient verwendet werden. Die CPU sollte so wenig wie möglich benutzt („Time is Accu") und der knappe Speicher bestmöglich verwaltet werden. Die Standard-VM ist diesbezüglich leider nicht in geeigneter Weise optimiert. Die Android-Entwickler haben deshalb mit der Dalvik-VM eine neue, hochoptimierte Variante einer virtuellen Maschine implementiert.

Diese Dalvik-VM ist nicht als Stack-Maschine konzipiert, sondern arbeitet registerorientiert, ähnlich wie bei einer klassischen CPU. Zudem wurde die Code-Einheit des Bytecodes von 8 Bit auf 16 Bit verdoppelt. Mit diesen Maßnahmen ist die Dalvik-VM auf den meisten Prozessoren deutlich näher an der Hardware, was sich in einer gesteigerten Effizienz und Performance bemerkbar macht.

Da jede Android-Anwendung eine eigene VM startet und damit alle Anwendungen aus Sicherheits- und Stabilitätsgründen separiert (Sandbox-Prinzip), spielt das Thema effiziente Speichernutzung eine große Rolle. Aus diesem Grund wird beim Systemstart eine einzelne Instanz der Dalvik-VM gestartet (eine sogenannte „Zygote"), die bei der Erzeugung jeder weiteren Instanz in einen separaten logischen Adressraum geklont wird.

Die Speicherverwaltung des Kernels sorgt zunächst dafür, dass der Klon den gleichen physikalischen Speicher referenziert wie die Zygote. Erst wenn sich Daten der Applikation (beispielsweise Variableninhalte) ändern, werden neue Speicherbereiche belegt und neu referenziert („Copy on Write"). So ist sichergestellt, dass wesentliche und in hohem Maße statische Teile des Systems (wie Klassen) nur einmalig im Speicher vorgehalten werden müssen.

Lizenzfragen

Der zweite Grund für die Entwicklung einer eigenen virtuellen Maschine war vermutlich die Vermeidung von Lizenzabgaben. Dadurch, dass die Dalvik-VM, die laut Google auf dem Apache Harmony-Projekt basiert, nicht mehr bytecode-kompatibel zur Standard-VM ist, ist ein spezieller Übersetzer (dx-Tool) erforderlich, der die .class-Bytecode-Dateien in das Dalvik-VM-Format (.dex) konvertiert. Da der Bytecode und die virtuelle Maschine von Java lizenzrechtlich geschützt sind und beides auf dem Android-Gerät keine Verwendung findet, sollten keine Lizenzabgaben fällig sein.

Oracle – durch die Übernahme von Sun mittlerweile Eigentümer der Java-Rechte – sieht das freilich anders. Die seitens Oracle eingereichte Klage soll eine Klärung herbeiführen und könnte weitreichende Konsequenzen nicht nur für Google und Android, sondern für die gesamte Java-Community haben, die den Streit mit Unbehagen hinsichtlich der zukünftigen Offenheit von Java beobachtet.

Entwicklungsumgebung

Das Android Software Development Kit (SDK) und eine aktuelle Eclipse-Version sind alles, was man für die Entwicklung von Android-Applikationen benötigt. Es existiert zwar noch ein natives Kit (NDK), das die Entwicklung Performance-kritischer Anwendungen in C/C++ erlaubt, allerdings gibt man damit die Plattform-Unabhängigkeit teilweise auf. Außerdem verfügt Android seit der Version 2.2 über einen Just-In-Time-Compiler, der den Performance-Vorteil relativiert.

Das SDK, das von Google frei zur Verfügung gestellt wird, beinhaltet die Android Development Tools (ADT) als Eclipse Plugin. Damit lassen sich sowohl Projekte für die passende Android-Plattform erzeugen als auch virtuelle Devices (siehe Abbildung 2) anlegen und verwalten.

Virtuelle Devices sind emulierte Geräte, auf denen eine Android-Applikation zunächst ohne ein physikalisches Endgerät getestet werden kann. Damit die Emulation unter annähernd realistischen Bedingungen erfolgt, lassen sich die virtuellen Geräte hinsichtlich Bildschirmgröße, Speicherausbau, Dateisysteminhalt und diverser anderer Merkmale konfigurieren. Eine in der Entwicklung befindliche Anwendung kann so direkt aus Eclipse her-

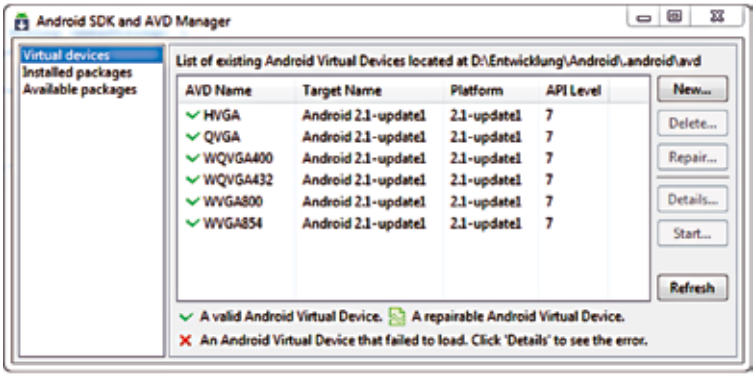


Abbildung 2: Virtuelle Devices

aus gebaut und dann auf dem virtuellen Gerät eingesetzt und getestet werden. Der Emulator (siehe Abbildung 3) ist Bestandteil des SDK.

Abbildung 3: Der Emulator

Im Emulator lässt sich die zu testende Applikation wie gewohnt debuggen. Der Dalvik-Debug-Monitor (siehe Abbildung 4) ermöglicht außer der Ansicht diverser Statusinformationen (Heap, Threads etc.) auch die Simulation verschiedener Situationen wie Telefonaktionen, Netzstatus-Wechsel oder die Einspeisung unterschiedlicher GPS-Daten. Ein physikalisches Endgerät ist über den USB-Port angebunden. Die Treiber liegen ebenfalls dem SDK bei. Danach können Applikationen direkt auf das Gerät eingespielt, getestet und debugged werden.

Ausblick

Google macht es Entwicklern leicht, Anwendungen für Android zu programmieren. Dank der Verwendung von Java existiert ein sehr großes Potenzial in der Entwicklergemeinde und die zur Verfügung gestellten Tools bilden eine effektive und qualitativ hochwertige Basis zur Erstellung von Android-Anwendungen. Allerdings entwickelt sich genau wie Android auch das SDK zurzeit noch in sehr hohem Innovationstempo. Entwickler müssen sich deshalb auf Veränderungen in den APIs und des Frameworks einstellen.

Traut man den Prognosen, ist der Siegeszug von Android im Mobile-Bereich trotzdem nicht aufzuhalten. Die Zukunft bleibt auch wegen des aufkommenden Tablet-Booms spannend. Alles gute Gründe, sich genauer mit dieser interessanten Technologie zu befassen.

Andreas Flügge
info@ordix.de

Andreas Flügge ist seit 2002 Senior Consultant bei der Object Systems GmbH, einer Tochtergesellschaft der ORDIX AG. Seine Kenntnisse umfassen Middleware, Netzwerke, zahlreiche Programmiersprachen, Protokolle/Schnittstellen,Reporting Tools sowie Programmierung.

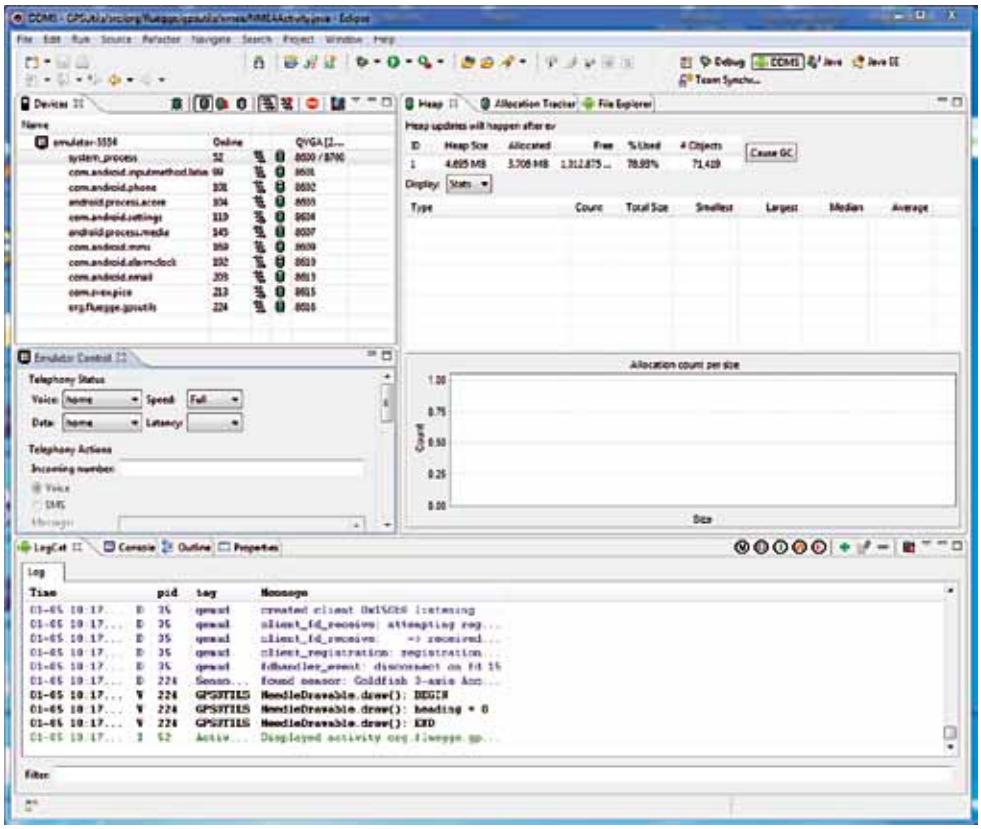


Abbildung 4: Der Debug-Monitor

Hibernate im Projekteinsatz

Dirk Mahler, buschmais GbR

Der Einsatz eines O/R-Mappers gehört heute zu den Selbstverständlichkeiten in Java-EE-Projekten. Die seit vielen Jahren am Markt verfügbaren Produkte gelten als ausgereift und erprobt. Im Alltag kann es trotzdem unangenehme Überraschungen geben, wie am Beispiel von Hibernate verdeutlicht werden soll.

Ein neues Projekt beginnt und ich kann endlich einmal etwas auf der grünen Wiese anfangen. Aus dem Java-Universum suche ich mir die aktuellen Versionen meiner Lieblingsframeworks zusammen. Dabei taucht ein entscheidendes Problem auf: Ich habe eine relationale Datenbank und reines JDBC ist sicherlich nicht mehr up to date. Wenn in anderen Java-Projekten die Themen „Persistenz" und „relationale Datenbank" auf den Tisch kommen, dann fallen immer wieder die Begriffe „Java Persistence API" (JPA) und „Hibernate". Die Gründe liegen auf der Hand:

- JPA ist ein Standard für Objekt-/Relationale Persistenz, der im Rahmen des Java Community Process (JCP) entstanden ist und breite Akzeptanz besitzt.
- Auf die JPA-Spezifikation haben die Hibernate-Entwickler – namentlich Gavin King – großen Einfluss ausgeübt. Ich kann davon ausgehen, dass Kompatibilität mit dem Standard kein Problem darstellen sollte.
- Hibernate ist seit vielen Jahren am Markt und der wohl populärste O/R-Mapper schlechthin, also ein ausgereiftes Produkt.
- Es existiert eine große Community rund um das Produkt. Lösungen für die gängigsten Probleme stehen mit Sicherheit in Foren, FAQs und Blogs.
- Es gibt eine Menge interessanter Frameworks aus dem JBoss-Umfeld, die sich eng mit Hibernate integrieren und teilweise selbst Basis für weitere Standardisierungen im JCP sind, beispielsweise Hibernate Validator und JBoss Seam. Das hinterlässt einen sehr innovativen Eindruck.
- Das Beste zuletzt: Hibernate steht unter der Lesser GNU Public License (LGPL), ist also Open-Source und verursacht keine Lizenzkosten.

Da kann doch gar nichts schiefgehen! Ich werfe einen Blick ins Maven-Repository und finde aktuelle Hibernate-Artefakte – die Anpassung der Build-Deskriptoren ist schnell erledigt.

Das Modell

Auf dem Reißbrett ist ein Domänenmodell entstanden. Es handelt sich um Personen, deren Adressen sowie deren private Daten. Abbildung 1 zeigt das Klassendiagramm. Folgende Beziehungen sind zwischen den Entitäten vorgesehen:

Abbildung 1: Das Objektmodell

- Eine Person besitzt eine Menge von Adressen (1..*)
- Eine dieser Adressen stellt die primäre Adresse der jeweiligen Person dar. Diese Beziehung ist obligatorisch (1..1)
- Eine Person kann optional private Daten besitzen (0..1)

Mithilfe von JPA-Annotationen erzeuge ich die Mappings und stelle dafür folgende Regeln auf:

- Kompositionen werden zwecks Navigierbarkeit bidirektional mit Rückbeziehung gemappt. Ausgehend von der besitzenden Seite sind alle Kaskadierungen aktiviert (CascadeType.ALL). So erreiche ich, dass beim Löschen einer Person sämtliche abhängige Objekte unkompliziert entsorgt werden.
- Weitere Beziehungen erhalten den CascadeType.PERSIST. Das soll es mir erleichtern, Objektgraphen zu persistieren, indem ich ein beliebiges Wurzelelement verwenden kann.
- Im Falle einer Komposition, die eine 1:1-Beziehung darstellt (Person#privateData), soll der Fremdschlüssel mit einem Unique-Constraint auf der Seite des abhängigen Objekts definiert werden. Damit ist eine Sicherung der Datenintegrität auf Ebene der Datenbank möglich.

- Für jede X:1-Beziehung ist zu spezifizieren, ob sie optional oder obligatorisch ist (optional=true|false). So kann ich am Mapping erkennen, wie mit Werten aus dieser Beziehung umzugehen ist – die Domänen-Objekte werden selbsterklärend. Außerdem erhält der O/R-Mapper Informationen über die Art zu generierender Joins.
- Alle Beziehungen sind mit dem FetchType.LAZY versehen. Eager-Fetching ist, wenn notwendig, in den Anwendungsfällen durch Abfragen zu realisieren. Das soll ermöglichen, Objekte unabhängig voneinander auch in großen Mengen zu laden.

Das Ergebnis der Mapping-Arbeit spiegelt sich in den Listings 1 bis 3 wider.

```
@OneToMany(      mappedBy = „person“,
cascade = CascadeType.ALL,
          fetch = FetchType.LAZY)
private List<Address> addresses = new
ArrayList<Address>();

@OneToOne(optional = false, cascade =
CascadeType.PERSIST,
          fetch = FetchType.LAZY)
private Address primaryAddress;

@OneToOne(mappedBy = „person“, optio-
nal = true,
          cascade = CascadeType.ALL,
          fetch = FetchType.LAZY)
private PrivateData privateData;
```

Listing 1: Person.java

```
@ManyToOne(      optional = false, cascade
= CascadeType.PERSIST,
          fetch = FetchType.LAZY)
private Person person;
```

Listing 2: Adress.java

```
@OneToOne(      optional = false, cascade
= CascadeType.PERSIST,
          fetch = FetchType.LAZY)
private Person person;
```

Listing 3: PrivateData.java

Ein einfacher Auftrag

Mein erster Anwendungsfall erzeugt eine neue Instanz einer Person. Sie erhält von mir zwei Adressen, eine davon wird als primäre Adresse referenziert. Der gesamte Objektgraph soll persistiert werden; ich rufe entityManager.persist() mit der Referenz auf die Person als Parameter auf.

```
org.hibernate.PropertyValueException: not-
null
property references a null or transient value:
com.buschmais.sample.model.Person.prima-
ryAddress
```

Autsch! Ich überfliege die Fehlermeldung und mir fällt ein, dass ich zwar die Beziehungen der Person zu ihren Adressen aufgebaut, aber die Rückbeziehungen nicht gesetzt habe. Ich korrigiere meinen Fehler, starte meine Anwendung:

```
org.hibernate.PropertyValueException: not-
null
property references a null or transient value:
com.buschmais.sample.model.Person.prima-
ryAddress
```

Wie bitte? Dieses Mal lese ich mir die Meldung – entgegen meinen Gewohnheiten – genauer durch und stelle fest, dass Hibernate die Referenz der primären Adresse moniert. Aber die ist doch gesetzt! Ich habe bestimmt etwas übersehen, setze einen Breakpoint auf die Zeile des Persistierens und debugge meine Anwendung. Der Objektgraph ist an diesem Punkt vollständig und konsistent, das verstehe ich nicht …
Ich ziehe die Möglichkeit in Erwägung, dass ich über einen Bug gestolpert bin. Wozu wurden eigentlich Suchmaschinen erfunden? Ich finde schließlich ein Ticket im Issue-Tracker von Hibernate, dessen Betreff mir passend erscheint. In der Tat: Hier ist mein Problem beschrieben [1].

Die kurze Geschichte eines Tickets

Meine Freude weicht schnell: Der Bug wurde bereits vor zwei Jahren berichtet, der Status lautet „resolved“ und die Resolution besagt „rejected“. Voller Verwunderung schaue ich mir die Historie des Falls an. Der einzige Kommentar eines Hibernate- Entwicklers vermeldet lapidar „you're missing some cascade“. Der Ersteller des Tickets stellt in seiner Antwort klar, dass das Hinzufügen sämtlicher Kaskadierungen nichts an der Fehlersituation ändert. Ich gebe eine Stimme für das Ticket ab, trage mich als Beobachter ein und füge einen Kommentar hinzu, dass das Problem in der Tat existiert und das Ticket wieder geöffnet werden sollte – mehr kann ich nicht machen. Glücklicherweise ist ein Workaround beschrieben; die Deklaration der Beziehung für die primäre Adresse als optional schafft Abhilfe:

```
@OneToOne(optional = true,
          cascade = CascadeType.
PERSIST,
          fetch = FetchType.LAZY)
private Address primaryAddress;
```

Ich bin wenig begeistert von dieser Idee. Obwohl mein Domänen-Objekt damit nicht mehr selbsterklärend ist, probiere ich es trotzdem aus. Und tatsächlich: Nun endlich funktioniert das Persistieren.

Übertroffene Erwartungen

Ich gebe mich zufrieden und widme mich meinem nächsten Anwendungsfall: Eine Abfrage soll mir sämtliche Personen (> 10.000) in der Datenbank liefern; die dazugehörige JPQLAbfrage ist einfach: „SELECT p FROM Person p“. Hibernate übertrifft meine Erwartungen deutlich, zumindest was die Menge geladener Objekte betrifft. Neben des erwarteten SQL-Statements „SELECT … FROM Person“ wird pro gefundener Person ein weiteres Statement zum Laden der optionalen Referenz „privateData“ ausgelöst. Wahrscheinlich habe ich vergessen, die Beziehung mit „FetchType.LAZY“ zu versehen. Doch mich erwartet eine weitere Überraschung: Die Deklaration ist korrekt und ich verstehe erst einmal nichts mehr.

Ich greife wieder zur Suchmaschine. Die Stichwörter „hibernate lazy loading“ weisen mich in die richtige Richtung. Zwei Klicks weiter zeigt sich des Rätsels Lösung: Der von Hibernate standardmäßig verwendete Lazy-Loading-Mechanismus ist der gesuchte Schuldige. Er basiert auf zur Laufzeit erzeugten, dynamischen Proxies, die anstelle der referenzierten Objekte verwendet werden. Der Haken daran: Um zu entscheiden, ob nach dem Laden des referenzierenden Objekts (Person) die optionale Beziehung (privateData) mit „null“ oder einem Proxy initialisiert werden muss, erfolgt eine Prüfung, ob das referenzierte

Objekt (PrivateData) in der Datenbank existiert, und dazu ist ein SQL-Statement nötig. Zu diesem Verhalten finde ich keinen Hinweis in der Hibernate-Dokumentation.

Die Community hilft

In Foren und Blogs finden sich diverse – teils originelle – Hinweise, wie ich mit diesem Problem umgehen soll. Sie reichen von „alle Beziehungen als „EAGER" definieren" über „Ersatz der „@OneToOne"-Beziehung durch eine „@OneToMany"-Beziehung, die aber nur ein Element beinhaltet" über „manuelle Implementierung der Bytecode-Instrumentation" bis hin zu „Buildtime-Bytecode-Instrumentation" [2][3]. In meinem Kopf regt sich etwas: Moment mal, waren es nicht die Hibernate-Macher, die immer wieder hervorhoben, dass Instrumentation unnötig und lästig sei … Ich entscheide mich trotzdem dafür; es scheint mir die sauberste Lösung zu sein. In einem Foren-Eintrag finde ich einen Code-Schnipsel zum Aufbohren meines Maven-Deskriptors und eine lange Odyssee beginnt.

Zunächst schlägt die Instrumentierung wegen fehlender Bibliotheken fehl. Nach einigem Hin und Her und einem Foren-Hinweis bekomme ich die Instrumentierung mit einer älteren Hibernate-Version zum Laufen. Ein wenig Trickserei mit diversen Bibliotheken (unter anderem CGLIB) ist notwendig. Mein bereits unwohles Bauchgefühl verbessert sich nicht, als beim Hochfahren der Anwendung eine Meldung erscheint, dass die CGLIB-Unterstützung „deprecated" sei. Die Nachlade-Operation erscheint immer noch, aber ich kann mich nicht geschlagen gegeben. Mein Debugger verrät mir, dass in der persistence.xml noch eine Hibernate-Property gesetzt werden muss, welche die CGLIB-Unterstützung aktiviert. Das Nachladetatement erscheint aber immer noch … In einem weiteren Foren-Eintrag findet sich der entscheidende Hinweis, dass die entsprechenden Beziehungen zusätzlich mit „@LazyToOne" annotiert werden müssen.

Nach einer gefühlten Ewigkeit bin ich am Ziel: Mein Anwendungsfall verhält sich wie erwartet. Trotz oder gerade wegen des steinigen Wegs und auf das Prinzip Hoffnung bauend, belasse ich es bei dieser Lösung. Zugegebenermaßen schwingt dabei ein wenig Stolz mit.

Wer hoch steigt …

Wenige Tage später stelle ich fest, dass Fetch-Joins in Criteria-Abfragen aus einem unerfindlichen Grund nicht mehr funktionieren und die Anwendung mit NullPointerExceptions beim Zugriff auf die entsprechenden Attribute reagiert. Die Ursache ist zunächst unklar. Aber nach dem Zurückverfolgen der letzten Änderungen gerät die mühevoll eingebaute Bytecode-Instrumentierung unter Verdacht. Dieser bestätigt sich auch schließlich. Ich baue alles zurück und die Nachlade-Operationen sind wieder da. Frustriert entscheide ich mich für eine andere Lösung: Die Beziehung privateData erhält nun den FetchType „EAGER", sie wird ab sofort immer und grundsätzlich per Join geladen:

```
@OneToOne(mappedBy = „person", optional
= true,
        cascade = CascadeType.ALL,
        fetch = FetchType.EAGER)
private PrivateData privateData;
```

Der Weg ist nicht immer das Ziel

Ich beschließe, meine Erfahrungen mit anderen Entwicklern auszutauschen. Sie erzählen mir, dass diese und ähnliche Geschichten auch von ihnen hätten stammen können. Einige Thesen fallen mir ein:

- Das Siegel eines Standards stellt keine Aussage über die Qualität der Umsetzung dar. Im Beispiel kann dies an der Unterstützung optionaler, inverser Beziehungen nachvollzogen werden.
- Jedes Produkt trägt die Handschrift seiner Entwickler. Hibernate ist ein Framework, das von Use-Case zu Use-Case gewachsen und in seinen Ansätzen pragmatisch geblieben ist. Dieser Eindruck wird durch Einblicke in den Quellcode bestätigt. .
- Die offizielle Dokumentation ist sehr gut geeignet, um einen schnellen Start zu ermöglichen. Sie konzentriert sich aber auf Dinge, mit denen Einfachheit, Mächtigkeit und Innovationskraft bewiesen werden sollen – unliebsame Themen werden nur gestreift oder bleiben außen vor. Das ist nicht überraschend, da die Dokumentation kommerzieller Open-Source-Produkte eine Marketing-Funktion erfüllt.
- Ein Eindruck, der sich leider zu oft bestätigt: Hibernate-Entwickler neigen dazu, Probleme in Foren oder im Issue-Tracker wegzuwischen oder zu ignorieren. Fehler werden als unabänderliches oder gewolltes Verhalten dargestellt, Tickets ohne nähere Prüfung geschlossen.
- Das Internet bietet eine Vielzahl von Problemlösungen an. Oftmals ist deren Qualität unbefriedigend: Sie sind größtenteils durch Ausprobieren seitens der Anwender entstanden. Damit stellen diese „Lösungen" später nicht selten die Ursache für weitere Probleme dar, für die sich dann wieder Workarounds finden.
- Der offene Quellcode ermöglicht das Auffinden von Ursachen für unerklärliches Verhalten. Damit wird der Ehrgeiz der Entwickler angestachelt, die Probleme selbst zu lösen. Es passiert aber nicht selten, dass dies zum Selbstzweck wird und die Arbeit am eigentlichen Projekt aus dem Fokus gerät [4].

Fazit

Auch wenn sich Hibernate in meinem Projekt trotz aller Ärgernisse immer noch ganz tapfer schlägt, haben sich viele der eingangs aufgezählten Erwartungen stark relativiert. Eine Evaluation des Produkts vor dessen Einsatz hätte die zu Tage getretenen Schwächen wahrscheinlich offenbart, ebenso die Qualität der Unterstützung durch Entwickler beziehungsweise durch die Community.

Quellen

[1] https://jira.jboss.org/browse/EJBTHREE-686
[2] http://community.jboss.org/wiki/Someexplanationsonlazyloadingone-to-one
[3] http://justonjava.blogspot.com/2010/09/lazy-one-to-one-andone-to-many.html
[4] http://in.relation.to/Bloggers/Hibernate-MovesToGitGitTips-AndTricks

Dirk Mahler
dirk.mahler@buschmais.com

Dirk Mahler ist als Senior Consultant auf dem Gebiet der Java-Enterprise-Technologien tätig. In seiner täglichen Arbeit setzt er sich mit Themen rund um Software-Architektur auseinander, kann dabei eine Vorliebe für den Aspekt der Persistenz nicht verleugnen. Er ist Gesellschafter der buschmais GbR, einem Beratungshaus mit Sitz in Dresden.

Semantisch-orientierte Programmierung mit Java

Oliver Böhm

Eine Einführung in eine neue Art der Programmierung, bei der die Datenbeschreibung direkt mit Java-Mitteln ausgedrückt und damit Teil der Dokumentation wird – ohne die Gefahr, dass Dokumentation und Code auseinanderlaufen.

Vor einiger Zeit wurde ich gefragt, ob man Semantische Programmierung mit Aspektorientierter Programmierung (AOP) vergleichen könne. Während ich Letzteres sehr gut kenne, immerhin habe ich ein Buch über AspectJ verfasst, einer AOP-Erweiterung für Java, war Ersteres noch Neuland für mich. Aber wozu gibt es Wikipedia: „Semantic-oriented programming (SOP) is a programming paradigm in which programmers express instructions directly in semantic meanings most suitable to reflect the task at hand."
Viel half mir dieser Artikel nicht weiter. Auch andere Artikel im Netz zeigten mir, dass Vieles noch im Forschungsstadium liegt und auch unterschiedliche Auffassung von SOP existieren. Mit Soplets (siehe http://www.soplets.org) existiert immerhin für Java ein Ansatz, der Folgendes für die Software-Entwicklung verspricht:

- Einfachheit (Simplicity)
- Wartbarkeit (Maintainability)
- Transparenz (Transparency)

Think Different

Was bietet sich für den Einstieg in neue Themen an? Richtig, das seit Kernighan & Ritchie so beliebte Hello-World-Beispiel. Hier in der klassischen Java-Version:

```
public class World {
    public static void main(String[] args) {
        System.out.println(„Hello World!");
    }
}
```

Da der Schwerpunkt bei den Soplets auf den Daten liegt, hilft uns dieses Beispiel nur bedingt weiter. Daher erweitern wir es und geben den Gruß als Sammlung von Worten aus. Dabei definieren wir die erlaubten Worte als enum:

```
public enum Word {
    HELLO,
    WORLD;
}
```

Damit ändert sich die Ausgabe-Anweisung:

```
System.out.println(Word.HELLO + „ „ +
Word.WORLD);
```

Den Nachteil, dass die Ausgabe nur in Großbuchstaben erfolgt, kann man durch eine geeignet „toString()"-Methode in der Word-Klasse beheben. Aber darum geht es hier gar nicht. Der eigentliche Clou beim Soplet-Gedanken ist, dass man dieses Modell, das durch diese Enumeration repräsentiert wird, mit Meta-Informationen in Form von Annotationen anreichert:

```
@Retention(RetentionPolicy.RUNTIME)
public @interface WordInfo {
    String textEN();
    String textDE();
    String description();
    String shortform() default „";
    boolean isNoun() default true;
}
```

Diese Annotation lässt sich von der Word-Enumeration nutzen, um die aufgelisteten Wörter näher zu beschreiben und z.B. in der „toString()"-Implementierung auszunutzen (siehe Listing 1).

```
public enum Word {

    @WordInfo(
            textEN = „Hello",
            textDE = „Hallo",
            description = „some kind of greeting",
            shortform = „Hi",
            isNoun = false
    )
    HELLO,
    ...

    public String getText() {
        try {
            WordInfo type = this.getClass().getField(this.name()).
            getAnnotation(WordInfo.class);
            if (Locale.getDefault() == Locale.GERMAN) {
              return type.textDE();
            }
            return type.textEN();
        } catch (Exception e) {
            throw new RuntimeException(„internal error",
            e);
        }
    }

    @Override
    public String toString() {
        return getText();
    }
}
```

Listing 1

Die Spracheinstellung (Locale) wurde dazu genutzt, um den englischen oder deutschen Text für die Ausgabe zu verwenden. Natürlich könnte man dies auch konventionell über ResourceBundles lösen, aber der Vorteil über Annotationen ist folgender:

- Der Text steht direkt vor dem betroffenen Wort und muss nicht in einer Resource-Datei gesucht werden
- Ein Text kann erzwungen werden, sonst gibt es einen Compile-Fehler
- Der Ansatz ist robuster gegen Änderungen (zum Beispiel Refactorings)

Es ist schwierig, anhand dieses einfachen Beispiels die Stärken und Schwächen dieses Ansatzes beurteilen zu können. Für eine bessere Einschätzung nehmen wir ein Projekt aus der Praxis, das wir nach diesem Ansatz umbauen. Als Beispiel dient die OpenSource-Bibliothek gdv.xport (siehe https://github.com/oboehm/gdv.xport), die den Im- und Export des GDV-Datensatzes erleichtert. Der GDV-Datensatz ist ein standardisiertes Datenformat zum Austausch zwischen Versicherungsunternehmen.

Abbildung 1 zeigt einen Ausschnitt zum Satz 200 (Allgemeiner Vertragsteil) aus dem Handbuch zum GDV-Datensatz (http://www.gdv-online.de/vuvm/bestand/best_2007.htm). Der GDV-Datensatz ist mehr oder weniger eine Sammlung verschiedener Sätze, die wiederum aus mehreren Teildatensätzen bestehen können. Das Ganze erinnert an das gute alte Lochkartenformat. Die derzeitige Abbildung des Datensatzes in Java sieht zurzeit so aus:

```
public class AllgemeinerVertragsteil extends
Datensatz {
    ...
    private void setUpDatenfelder() {
        ...
        add(new Datum(HAUPTFAELLIGKEIT,
        60));
        add(new Zeichen(ZAHLUNGSWEISE,
        68));
        add(new Zeichen(VERTRAGSSTATUS,
        69));
        add(new
        AlphaNumFeld(ABGANGSGRUND,
        2, 70));
        ...
    }
}
```

Dies ist der Ausschnitt, der vom Konstruktor aufgerufen wird, um den Satz aufzubauen. Er entspricht dem abgebildeten Ausschnitt aus dem GDV-Handbuch, ist aber ohne weitere Erklärung (beziehungsweise Lesen des Codes) nicht so ohne weiteres zu verstehen. Zum Vergleich schauen wir uns die jetzige SOP-Lösung an:

Gesamtverband der Deutschen Versicherungswirtschaft e.V.
GDV-Branchennetz
Versicherungsunternehmen - Vermittler

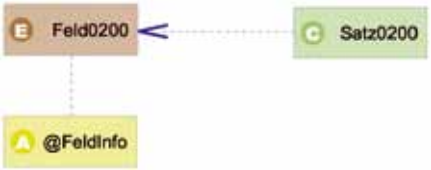
GDV

Kapitel:	**Allgemeine Daten**			Stand:	**15.05.2007**
Abschnitt:	**Allgemeiner Vertragsteil**			Satzart:	**0200**
				Version:	**2.2**
Nr.	Bezeichnung	Darst. AN/N	Anz. Bytes	Byte-Adr.	Inhalt / Erläuterung
11	Hauptfälligkeit	N	8	60	Die jeweils folgende Hauptfälligkeit des Vertrages zum Zeitpunkt der Lieferung. Sollten Tag und/oder Monat nicht vorhanden sein, muss "00" geschlüsselt werden. Tag/Monat/Jahr (TTMMJJJJ)
12	Zahlungsweise	AN	1	68	blank = Rechtschutz / Verkehrsservice / Kredit (In den Sparten Rechtschutz und Verkehrsservice kann das Datenfeld "Zahlungsweise" in der Satzart 0200 blank sein, die Zahlungsweise wird dann auf Risikoebene in der Satzart 0210, spätestens in Satzart 0220 angegeben. Bei Einzel- und / oder Umsatzanmeldungen muss das Datenfeld "Zahlungsweise" in der Satzart 0400 blank sein.) siehe Anlage 14
13	Vertragsstatus	AN	1	69	siehe Anlage 24

Abbildung 1: Der GDV-Datensatz

```
public enum Feld0200 {
    ...

    @FeldInfo(
            teildatensatz = 1,
            nr = 11,
            type = Datum.class,
            anzahlBytes = 8,
            byteAdresse = 60
    )
    HAUPTFAELLIGKEIT,

    @FeldInfo(
            teildatensatz = 1,
            nr = 12,
            type = Zeichen.class,
            byteAdresse = 68
    )
    ZAHLUNGSWEISE,

    @FeldInfo(
            teildatensatz = 1,
            nr = 13,
            type = Zeichen.class,
            byteAdresse = 69
    )
    VERTRAGSSTATUS,

    @FeldInfo(
            teildatensatz = 1,
            nr = 14,
            type = AlphaNumFeld.class,
            anzahlBytes = 2,
            byteAdresse = 70
    )
    ABGANGSGRUND,

    @FeldInfo(
            teildatensatz = 1,
            nr = 15,
            type = Datum.class,
            anzahlBytes = 8,
            byteAdresse = 72
    )
    ABGANGSDATUM,
    ...

}
```

Hier kann man jetzt schon eher den Zusammenhang zwischen Code und dem Auszug aus dem GDV-Handbuch erkennen. Allerdings sind noch weitere Klassen für die gleiche Funktionalität notwendig:



Annotation	Annotationen bieten die Möglichkeit, zusätzliche Meta-Informationen an Daten, Methoden oder Klassen anzuhängen. Sie können zur Laufzeit ausgewertet werden und wurden mit Java 5 eingeführt.
AOP	Aspekt-orientierte Programmierung: stellt Sprachmittel zur Verfügung, um Querschnitts-Belange, die mit OO-Konzepten kaum gekapselt werden können (z.B. Logging, Transaktionen, Security etc.), in Aspekten herauszuziehen.
Enum	Basis-Klasse für einen eigenen Aufzählungs-Typ, der mit Java 5 eingeführt wurde.
SOP	Semantisch-orientierte Programmierung: ein Programmier-Ansatz, der auf Enumerationen und Annotationen beruht.
Soplet	Ein Java-Coding-Pattern, das auf SOP beruht (s.a. http://www.soplets.org).

Tabelle 1: Glossar

Die Satz0200-Klasse greift dabei auf die Enumeration Feld0200 zu, um die internen Teildatensätze mit Hilfe der Meta-Informationen aus der FeldInfo-Annotation aufzubauen:

```
public class Satz0200 extends Datensatz {
    ...
    private void setUpDatenfelder() {
        Feld0200[ ] felder = Feld0200.values();
        for (int i = 0; i < felder.length; i++) {
            add(felder[i]);
        }
    }
}
```

Anhand der FeldInfo-Annotation, die die benötigte Information trägt, lassen sich damit relativ einfach die internen Datenfelder anlegen. Als Zwischenresultat lässt sich folgendes Fazit ziehen:

Vorteil:

- Code wird sehr viel lesbarer und wartbarer (Code ist gleichzeitig Dokumentation)
- Code ist leichter erweiterbar (zusätzliche Infos können in der FeldInfo-Annotation untergebracht werden)
- Es gibt eine höhere Typsicherheit für den Zugriff auf Felder
- Dokumentation kann erzwungen werden (über FeldInfo-Annotation)

Nachteil:

- Etwas komplexer (drei Klassen statt einer)
- Höherer Schreibaufwand
- Schlechtere Performance
- Keine Vererbung bei Enum-Klassen möglich (warum eigentlich?)

Die fehlende Vererbung stört dann, wenn man beispielsweise Sätze hat, die ähnlich aufgebaut sind und den gleichen Anfang besitzen. Man kann sich zwar durch Komposition und andere Tricks behelfen, aber natürlich wäre hier der Einsatz von Vererbung schöner.
Durch die Verwendung von Reflexion und Auslesen von Meta-Informationen verlangsamte sich der Konstruktor um den Faktor 3 – während der ursprüngliche Konstruktor zwischen 3 und 5 Millisekunden benötigte, brauchte er jetzt zwischen 10 und 13 Millisekunden. Dies lässt sich aber durch Caching oder Pooling auf unter eine Millisekunde drücken, sollte dies ein Problem sein.

Fazit

Inzwischen wurde das gdv.xport-Framework mit Version 0.6 komplett auf den SOP-Ansatz umgestellt. Dadurch hat sich die Definition neuer GDV-Datensätze, die noch nicht direkt durch das Framework unterstützt werden, weiter vereinfacht und die Datenstrukturen sind jetzt selbsterklärend – ein großer Vorteil dieses Ansatzes. Nachteilig war, dass der bestehende Code umgestellt und erweitert werden musste, um mit den neu eingeführten Enumerationen umgehen zu können. Aber auch hierfür gibt es Unterstützung in Form eines Eclipse-Plugins, das über www.soplets.org erreichbar ist. Insgesamt haben die Vorteile überwogen, da jetzt die Datenbeschreibung direkt mit Java-Mitteln ausgedrückt wird und damit Teil der Dokumentation wird – ohne die Gefahr, dass Dokumentation und Code auseinanderlaufen.

Oliver Böhm
ob@jugs.org

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

KURZMELDUNG

Rückt eine Einigung zwischen Oracle und Google näher?

Die vorgerichtlichen Verhandlungen gehen voran. Google hat beim United States Patent and Trademark Office (USPTO) die Überprüfung der sieben Patente verlangt, die Bestandteil der Klage sind. In vier Fällen hat die Behörde das Patent insgesamt oder einen Teil der Forderungen, die Oracle daraus konstruiert hat, für ungültig erklärt. Dies ist allerdings keine endgültige Entscheidung, sondern kann wiederum von Oracle angefochten werden – was in der Vergangenheit häufig erfolgreich war. Gleichzeitig schlug der Richter vor, die Streitpunkte aus Effizienzgründen auf den wesentlichen Kern zu reduzieren – was Google natürlich entgegenkommt. So könnte weiterhin alles auf einen Showdown vor Gericht ab dem 31. Oktober 2011 hinauslaufen. Aber jetzt hat Google wohl zum ersten Mal die Bereitschaft zu einer außergerichtlichen Einigung gezeigt: Bezugnehmend auf die Reduzierung der Streitpunkte schreiben die Google-Vertreter, dass dies auch eine „informelle Lösung" der Angelegenheit erleichtern würde.
http://fosspatents.blogspot.com/2011/07/google-blinks-in-oracle-patent-case.html

Slice – Unterstützung für verteilte, partitionierte und heterogene Datenbanken mit OpenJPA

Bernd Müller, Ostfalia Hochschule für angewandte Wissenschaften,
sowie Harald Wehr, MAN Truck & Bus AG

Die JPA-Provider arbeiten an Lösungsansätzen, von denen der Artikel exemplarisch Slice von OpenJPA vorstellt.

Die Cloud ist gegenwärtig das zentrale Thema der IT und auch Gegenstand von Arbeiten innerhalb des Java Community Process (JCP). Der Java Specification Request für Java-EE 7 (JSR 342) benennt die Cloud explizit als zentrales Thema (siehe Seite 54). Neben allgemeinen Diensten ist die Ablage von Daten in der Cloud ein wichtiges Ziel aktueller Entwicklungen. Die strikte Trennung der Daten verschiedener Kunden ist dabei selbstverständlich. Der JSR für JPA 2.1 (JSR 338) beinhaltet als einen der Schwerpunkte die Mandantenfähigkeit von JPA-Anwendungen (Multitenancy), die diese Trennung realisiert. Die Arbeitsgruppen des JCP arbeiten zwar mittlerweile unter der Ägide von Oracle sehr öffentlich, es gibt jedoch noch keine konkreten Vorschläge bezüglich der genannten Thematik. Als Beispiel für die Arbeit der JPA-Provider dient Slice von OpenJPA.

Slice arbeitet als Schicht zwischen JPA und JDBC. Aus Sicht von JPA wird eine virtuelle Datenbank genutzt. Diese Basisannahme der JPA-Spezifikation konkretisiert sich durch die explizite Benennung der Persistenzeinheit in der JPA-Konfigurationsdatei „persistence.xml". Dies bleibt bei der Verwendung von Slice erhalten, wird aber auf die tatsächlich verwendeten Datenbanken verallgemeinert. Ziel ist es, das JPA-Programmiermodell unverändert zu übernehmen. Die Slice-Schicht wird daher auf Konfigurationsebene in der „persistence.xml" definiert und konfiguriert. Zusätzlich erlauben Call-Back-Mechanismen die Steuerung der Verteilung von Daten und Anfragen auf die verwendeten Datenbanken.

Der Artikel stellt zunächst die zentralen Konzepte von Slice im Überblick vor, geht dann detailliert auf die Konfigurationsoptionen ein und beschreibt abschließend einige ausgesuchte Konzepte ausführlicher, da eine vollständige Darstellung der Möglichkeiten von Slice den Rahmen sprengen würde.

Einordnung

Der JSR 342 (Java EE 7) enthält als großes Arbeitsthema die Cloud. Java EE ist bereits als Standardplattform zur Entwicklung von Unternehmensanwendungen etabliert und soll in der üblichen Cloud-Charakterisierung zur Platform as a Service (PaaS) ausgebaut werden. Als übergeordnete Dachspezifikation enthält Java EE unter anderem JPA. Für Java EE 7 wird JPA 2.1 als JSR 338 entwickelt. Hier ist im Spezifikationsaufruf „Support for multitenancy" explizit genannt.

OpenJPA ist neben EclipseLink und Hibernate einer der bekannteren JPA-Provider. Das Teilprojekt Slice von OpenJPA stellt Möglichkeiten zur Verfügung, um eine horizontal partitionierte Datenbankumgebung zu realisieren. Man kann damit JPA-Anwendungen mandantenfähig (multi-tenant) machen. Slice wurde jedoch nicht mit diesem (expliziten) Ziel entwickelt und andere Einsatzgebiete sind möglich und auch sinnvoll.

Zentrale Konzepte

Slice teilt die einer JPA-Persistenzeinheit zugrunde liegende logische Datenbank in eine Menge horizontal partitionierter realer Datenbanken auf. Die übliche JPA-Funktionalitäten (CRUD und Transaktionalität) werden auf diese Datenbanken verteilt, wobei über zusätzliche Klassen die Verteilung gesteuert werden kann.

Ein Hauptziel von Slice ist der Verzicht auf Änderungen am Anwendungs-Code, um die Verteilung zu steuern. Die Anwendungslogik kann bei der Verwendung von Slice unverändert bestehen bleiben und nur die die Verteilung steuernden Klassen mit entsprechenden Call-Back-Methoden sind hinzuzufügen.

Konfiguration

Slice ist eine OpenJPA-Erweiterung, sodass die Konfiguration mit Provider-spezifischen Properties und nicht mit den durch JPA 2.0 definierten Properties (Präfix „javax.persistence") erfolgt. Der folgende Ausschnitt aus der „persistence.xml" zeigt die Definition der Java-SE-Persistenzeinheit „bank" und die Aktivierung von Slice über das Property „openjpa.BrokerFactory" (siehe Listing 1).

```
<persistence ...
  <persistence-unit name="bank">
    <class>...</class>
    <properties>
      <property name="openjpa.BrokerFactory" value="slice" />
      ...
```

Listing 1

Diese Aktivierung von Slice weist das OpenJPA-Laufzeitsystem an, eine virtuelle Datenbank, bestehend aus mehreren realen Datenbanken, zu verwenden, die wir in Listing 2 konfigurieren.

```
<property name="openjpa.slice.Names"
        value="bank1,bank2,bank3" />
<property name="openjpa.slice.Master"
value="bank1" />
<property name="openjpa.slice.Lenient"
value="true" />
```

Listing 2

Hier werden die Slices „bank1", „bank2" und „bank3" deklariert. Master-Slice ist „bank1". Dieser wird zur Primärschlüsselgenerierung für alle Slices verwendet. Das Property „Lenient" erlaubt der Anwendung, auch ohne Verbindung zu allen Slices zu arbeiten. Die Verbindungsdaten zu den einzelnen Datenbanken können global (Präfix „openjpa") oder für einen Slice (Präfix „openjpa.slice") angegeben sein (siehe Listing 3).

```
<property name="openjpa.ConnectionDri-
verName"
        value="org.postgresql.Driver" />
<property name="openjpa.ConnectionUser-
Name" value="..." />
<property name="openjpa.ConnectionUser-
Password" value="..." />
```

Listing 3

Das Beispiel verwendet die globale Version, sodass alle verwendeten Datenbanken PostgreSQL-Datenbanken mit derselben Benutzername/Passwort-Kombination sein müssen. Die verwendeten Datenbanken unterscheiden sich dann lediglich in ihren Verbindungs-URLs (siehe Listing 4).

```
<property name="openjpa.slice.bank1.
ConnectionURL"
        value="jdbc:postgresql://localhost/
bank1" />
<property name="openjpa.slice.bank2.
ConnectionURL"
        value="jdbc:postgresql://localhost/
bank2" />
<property name="openjpa.slice.bank3.
ConnectionURL"
        value="jdbc:postgresql://localhost/
bank3" />
```

Listing 4

Soll eine alternative Datenbank verwendet werden, so sind die alternativen Informationen für diesen Slice anzugeben. Falls die Benutzername/Passwort-Kombination von oben auch für den vierten Slice gilt, so reduziert sich die explizite Information auf den Treiber und das Verbindungs-URL (siehe Listing 5).

```
<property name="openjpa.slice.bank4.Con-
nectionDriverName"
        value="com.mysql.jdbc.Driver" />
<property name="openjpa.slice.bank4.
ConnectionURL"
        value="jdbc:mysql://localhost/bank4"
/>
```

Listing 5

Im Property „openjpa.slices.Names" muss der Slice ebenfalls aufgenommen sein.

Allgemeine Verwendung

Bestehende Entity-Klassen können unverändert mit Slice verwendet werden. Für einen Bankkunden mit folgender Klasse kann auch die Möglichkeit der Tabellen-Generierung genutzt werden:

```
@Entity
public class Kunde implements Serializable {
  @Id @GeneratedValue
  private Integer id;
  private String vorname;
  private String nachname;
  @Temporal(TemporalType.DATE)
  private Date geburtsdatum;
```

Mit der obigen Konfiguration wird dann jeweils eine Kundentabelle in den drei PostgreSQL- und der MySQL-Datenbank erzeugt. Beim Einfügen von Entities („em.persist()") werden diese zufällig auf die zur Verfügung stehenden Slices verteilt.

Explizite Verteilung

Die zufällige Verteilung bewirkt lediglich eine Verteilung der Last beziehungsweise des Volumens. Soll eine logische Verteilung stattfinden, so ist das Interface zu implementieren und in der „persistence.xml" zu konfigurieren:

```
public interface DistributionPolicy {
    String distribute(Object entity,
                      List<String> slices,
                      Object context);
}
```

Der erste Parameter ist das zu persistierende Objekt, der zweite die Liste der aktiven Slices und der dritte ein für spätere Erweiterungen vorgesehener Kontext, der im Augenblick mit dem aktuellen Persistenzkontext belegt aber noch nicht verwendet wird. Im folgenden Beispiel erfolgt eine Verteilung bezüglich des Anfangsbuchstabens des Bankkunden (siehe Listing 6).

```
public class DistributionPolicyNachname
          implements DistributionPolicy {
  private static Random random;
  public DistributionPolicyNachname() {
    random = new Random();
  }

  @Override
  public String distribute(Object entity,
      List<String> slices, Object context) {
    if (entity instanceof Kunde) {
      char anfangsbuchstabe = ((Kunde)entity).
getNachname()
                        .toCharArray()[0];
      if ('A' <= anfangsbuchstabe && anfangs-
buchstabe < 'I') {
        return „bank1";
      } else if ('I' <= anfangsbuchstabe
              && anfangsbuchstabe < 'M') {
        return „bank2";
      } else if ('M' <= anfangsbuchstabe
              && anfangsbuchstabe < 'T') {
        return „bank3";
      } else {
        return „bank4";
      }
    } else {
      return „bank1";
    }
  }
}
```

Listing 6

Die Konfiguration dieser Distribution-Policy erfolgt in der „persistence.xml" über:

```
<property name="openjpa.slice.Distribution-
Policy"
      value="de.jpainfo.DistributionPolicy-
      Nachname"/>
```

Die einfache statistische Verteilung kann in eine Partitionierung bezüglich eines Mandanten erweitert werden. Soll etwa ein Online-Banking für mehrere Banken, beispielsweise die Tochtergesellschaften eines Konzerns, erstellt werden, so wird das Entity „Kunde" um ein Property „mandant" erweitert und dieses über die Anwendung vergeben. Mögliche Werte des Properties sind die Slice-Namen (siehe Listing 7).

```
@Entity
public class Kunde implements Serializable {
  ...
  @Column(nullable = false)
   private String mandant; // Werte: bank1, ...,
  bank4
  ...
}
```

Listing 7

Die Distribution-Policy wird ebenfalls angepasst und verteilt nun auf Basis des Properties „mandant" (siehe Listing 8).

```
public class DistributionPolicyMandant ...

  public String distribute(Object entity,
            List<String> slices, Object context)
{
    if (entity instanceof Kunde) {
      return ((Kunde) entity).getMandant();
    } else {
      ...
    }
  }
}
```

Listing 8

Bei der Verwendung der Annotationen „@OneToOne", „@OneToMany", „@ManyToOne" und „@ManyToMany" realisiert JPA über das „cascade"-Attribut eine transitive Persistenz: Das Speichern des Wurzelobjekts impliziert automatisch auch das Speichern aller assoziierten Objekte, falls „PERSIST" als Cascadierungsoption gewählt wurde. Slice weicht in diesem Fall von der Distribution-Policy ab und speichert alle assoziierten Objekte im selben Slice. JPA-Queries, auf die wir später noch eingehen, können damit effizient in einer Datenbank joinen. Ein Join über mehrere Datenbanken wird von Slice nicht unterstützt.

Andere Verteilungs-Policies

Die DistributionPolicy wird durch drei weitere Policies ergänzt, die erste ist ReplicationPolicy (siehe Listing 9).

```
public interface ReplicationPolicy {
  String[ ] replicate(Object entity,
                List<String> slices, Object
context);
}
```

Listing 9

Die „ReplicationPolicy" wird verwendet, um Daten in verschiedenen Datenbanken zu replizieren. Dies ist etwa für Aufzählungstypen („Enums" wie „Nationalitäten" und „Währungen") sinnvoll, die von mehreren Entities verwendet werden. Die Interface-Methode „replicate()" der „ReplicationPolicy" entspricht der „distribute()"-Methode der „DistributionPolicy", gibt jedoch ein Array von Slice-Bezeichnern zurück. Die zu replizierenden Klassen werden in der „persistence.xml" für das Property „ReplicatedTypes" aufgezählt, beispielsweise „Nationlitaet" und „Waehrung" und in die von „getTargets()" zurückgegebenen Slices repliziert:

```
<property name="openjpa.slice.Replicated-
Types"
      value="de.jpainfo.Nationalitaet,de.
      jpainfo.Waehrung"/>
```

Beim Lesen von Daten aus der Datenbank muss zwischen der EntityManager-Methode „find()" und JPA-Queries unterschieden werden. Dies geschieht mit den zwei Policies (siehe Listing 10).

```
public interface FinderTargetPolicy {
  String[ ] getTargets(Class<?> cls, Object
oid;
                List<String> slices, Object
                context);
}
public interface QueryTargetPolicy {
  String[ ] getTargets(String query,
  Map<Object,Object> params,
                String language, List<String>
                slices,
                Object context);
}
```

Listing 10

Die „getTargets()"-Methode der „FinderTargetPolicy" erhält als ersten Parameter die Entity-Klasse und als zweiten Parameter die Objekt-Id analog zur „find()"-Methode. Dritter und vierter Parameter sind die bereits beschriebenen Übergabewerte. Die „getTargets()"-Methode der Query"TargetPolicy" erhält als ersten Parameter den JPA-Query-String, als zweiten Parameter eine Map von Query-Parametern und ihren Werten und als dritten Parameter die Anfragesprache (für spätere Erweiterungen, im Augenblick nicht verwendet). Der vierte und fünfte Parameter wurden bereits diskutiert.

Beispiele

Zu Beginn des Artikels haben wir ein Kunden-Entity mit Vorname, Nachname und Geburtsdatum definiert. Anfragen ohne die Verwendung einer „QueryTargetPolicy" werden gegen alle aktiven Slices ausgeführt, was wir nun in einem Beispiel demonstrieren wollen. Bei der Standard-JPA-Query wird eine Select-Anfrage gegen alle aktiven Datenbanken (in unserem Fall drei PostgreSQL und eine MySQL-Datenbank) abgesetzt (siehe Listing 11).

```
TypedQuery<Kunde> query =
em.createQuery(„select k from Kunde k
order by k.nachname",
            Kunde.class);
List<Kunde> list = query.getResultList();
```

Listing 11

Die Anfrage enthält das entsprechende „Order By" von SQL, das heißt es wird in den jeweils beteiligten Datenbanken sortiert. Die sortierten Kundenlisten sind von Slice in Java verschmolzen. Es kommt ein Merge-Algorithmus zum Einsatz, sodass die Komplexität des Merge (O(n)) die Komplexität des Sortierens in den Datenbanken (O(n log(n)) insgesamt nicht negativ beeinflusst. Im nächsten Beispiel wird der jüngste Kunde gesucht. Auch hier kommt eine Standard-JPA-Query zum Einsatz:

```
Date gebu = (Date) em.createQuery(
    „select max(k.geburtsdatum) from Kunde
    k")
    .getSingleResult();
```

Es wird das jeweilige Maximum in den vier aktiven Datenbanken berechnet und anschließend von Slice in Java das Maximum dieser vier Werte verwendet. Auch hier ist die Komplexität (O(n)) nicht negativ beeinflusst.

Ausblick

Die beschriebene OpenJPA-Erweiterung Slice ist noch nicht im aktuellen Release 2.1 von OpenJPA enthalten, aber in den Snapshots für 2.2. Die Implementierung scheint noch nicht auf Produktionsniveau.

Das von uns entwickelte Beispiel des maximalen Geburtsdatums war fehlerhaft ausgeführt. Nach Meldung des Fehlers wurde dieser innerhalb eines Tages gefixt, was für die Agilität des Projekts spricht. Bei anderen Problemen befinden wir uns im Augenblick in der Klärungsphase.

EclipseLink arbeitet an ähnlichen Features. Kurz vor Drucklegung dieses Artikels wurde Release 2.3 von EclipseLink veröffentlicht, das sogenannte „Composite Persistence Units" ermöglicht, die ebenfalls mehrere Datenbanken in einer logischen Persistence Unit vereinigen. Die Annotation „@Multitenant" (und weitere) erlaubt die explizite Verwendung von Mandanten auf Quell-Code-Ebene und erscheint uns als vielversprechender Ansatz für dieses Problem. Entsprechende Entwicklungen bei Hibernate sind uns nicht bekannt.
In der JSR Expert-Group arbeiten Entwickler von OpenJPA, EclipseLink und Hibernate mit. Man darf gespannt sein, welche Ideen bezüglich Mandantenfähigkeit und Cloud letztendlich den Einzug in JPA 2.1 und damit Java-EE 7 schaffen.

Bernd Müller
bernd.mueller@ostfalia.de
Harald Wehr
harald.wehr@gmail.com

Bernd Müller ist seit März 2005 Professor für Software-Technik an der Fakultät Informatik der Hochschule Braunschweig/Wolfenbüttel, nachdem er sieben Jahre Professor für Wirtschaftsinformatik an der Hochschule Harz war. Praktische Erfahrung hat er zuvor im Wissenschaftlichen Zentrum der IBM in Heidelberg sowie bei HDI Informationssysteme in Hannover gesammelt.

Harald Wehr ist seit 2005 bei der MAN Gruppe in Salzgitter als Java-Entwickler beschäftigt. Seit 2008 befasst er sich verstärkt mit der SAP HR Anwendungsentwicklung und Beratung. Zusammen mit Bernd Müller verfasste er das Buch „Java-Persistence-API mit Hibernate". Die Autoren arbeiten derzeit an einer Neuauflage des Buches, das im kommenden Frühjahr beim Hanser-Verlag erscheinen wird. Gegenstand ist JPA in der Version 2.0 bzw. 2.1. Als Provider werden EclipseLink, Hibernate und OpenJPA verwendet.

NetBeans Platform 7

gelesen von Jürgen Thierack

Über den Autor erfährt man auf dem Umschlag: „Heiko Böck ist Master of Science in Informatik und Experte im Bereich der professionellen Software-Entwicklung mit Java. Für seine Arbeit mit NetBeans wurde er ins NetBeans Dream Team gewählt."

Das Buch ist zur Version 7.0 der NetBeans-Plattform erschienen, die noch vor JDK 7 herauskam. Eine Version 7.01 der NetBeans folgte Anfang August 2011 mit Berücksichtigung der endgültigen Java-Version und vielen hundert Bugfixes. Für November 2011 stehen mit Version 7.1 neue Features auf dem Plan.

Was im Titel nicht zum Ausdruck kommt: Der Schwerpunkt liegt eindeutig bei der Rich-Client-Entwicklung, nicht bei NetBeans als allgemeines Entwicklungswerkzeug für Java, PHP, C++ und HTML. So werden beispielsweise die neuen Features des HTML-Editors in der Version 7.0 nicht besprochen.

Überhaupt ist der Autor von Rich-Client-Plattformen absolut überzeugt. „Die vermeintliche Komplexität der Plattform-Konzepte ist einer der Hauptgründe, warum sich Rich-Client-Plattformen noch nicht als Quasi-Standard bei der Client-Anwendungsentwicklung durchgesetzt haben. Tatsächlich hat ein Entwickler zu Beginn den Eindruck, vor einem Berg von APIs und Konzepten zu stehen. Sind diese jedoch erst einmal erlernt beziehungsweise verstanden, so ergeben sich immense – zu Beginn vielleicht nicht erahnte – Synergien und Erleichterungen, welche die anfängliche Lernphase schnell wieder ausgleichen." Das ist richtig und trifft auf Eclipse-RCP zu.

Das Buch stellt die modulare Struktur der NetBeans vor, die APIs, Datenbank-Anbindungen und natürlich die Netbeans-typischen Swing-Oberflächen. Und wer von Eclipse zur NetBeans migrieren will findet dafür viele Tipps. Als umfangreiches „lerning by doing"-Beispiel wird ein MP3-Manager entwickelt.

Zwei Kapitel beschäftigen sich damit, wie man in Eclipse und in der IntelliJ-IDEA-NetBeans-Plattform Anwendungen entwickeln kann. Doch dahinter stehen nur Anleitungen, wie man von diesen Entwicklungswerkzeugen aus Maven veranlassen kann, ein als Maven-Projekt bereits fertig existierenes NetBeans-RCP-Projekt zu bauen. Es ist aber nun mal nicht sinnvoll, ohne die NetBeans für die NetBeans zu arbeiten.

Fazit: Das Buch ist gut geeignet für Leute, die NetBeans-RCP-Anwendungen programmieren möchten.

Jürgen Thierack ist in der Software-Branche freiberuflich unterwegs. Seit 10 Jahren liegt sein Schwerpunkt bei Fragen der Finanzmathematik, darunter der Preisfindung für Optionen und Optionsscheine. Aktuelle Thematik: Trendfolgesysteme.

Titel:	NetBeans Platform 7
Autoren:	Heiko Böck
Verlag:	Galileo Computing
Umfang:	670 Seiten
Preis:	49,90 €
ISBN:	978-3-8362-1731-6

XPages – Ein neues Framework zur Entwicklung von Web-Anwendungen

Dr. Rolf Kremer, PAVONE AG

XPages sind auf einem IBM-Lotus-Domino-Server lauffähig und können von jedem Webbrowser und dem auf Eclipse RCP-basierten IBM-Lotus-Notes-Client ausgeführt werden.

Eine XPage basiert auf XSP, einem XML-Dialekt, der bei der Ausführung in Java-Code umgewandelt wird. Die Entwicklung erfolgt mit dem IBM-Lotus-Domino-Designer, einem kostenlos erhältlichen, auf Eclipse IDE-basierten Entwicklungswerkzeug. Objekte lassen sich hiermit per Drag & Drop auf eine XPage platzieren und mit Java oder JavaScript erweitern. JavaScript-Frameworks wie das Dojo-Toolkit, Prototype oder jQuery lassen sich problemlos einbinden.

Bereits seit einigen Jahren gibt es eine Open-Source-Community namens „OpenNTF.org". Dort sind viele Anwendungen unter der Apache-V2-Lizenz oder GPL V3 verfügbar, die sich umgehend ausführen oder in eigene Projekte einbinden lassen. Der Artikel gibt einen Einblick in die Entwicklung mit XPages für Java-Entwickler.

Lotus Notes/Domino unterscheidet zwei zentrale Komponenten: den Server, der sich „Lotus Domino" nennt, und den properitären Client mit dem Namen „Lotus Notes". Zudem gibt es noch den Lotus-Domino-Designer, mit dem sich Anwendungen für den Lotus-Notes-Client und Webbrowser entwickeln lassen und den Domino-Administrator, das Administrationswerkzeug zum Verwalten des Domino Servers.

Vergangenheit

Die erste Version von Lotus Notes erschien bereits im Jahr 1989. Nach aktuellen Schätzungen gibt es heute weltweit mehr als 120 Millionen Anwender [1] und etwa zehn Millionen Anwendungen [2], die auf Lotus Notes/Domino basieren. Nachdem im Jahr 1995 die Lotus Development Inc. von IBM übernommen wurde, haben sich Client und Server stark in Richtung Java und Java EE weiterentwickelt. Anwendungen ließen sich mithilfe des Prototyping-Ansatzes schnell entwickeln und mit der eigenen @-Formelsprache LotusScript – einem Visual-Basic-Derivat – C/C++ oder Java anreichern. Während die Unterstützung der @-Formeln und LotusScript sowie die C/C++-Anbindung gut waren, kam die Java lange Zeit zu kurz. Dies lag zum einen an der mangelhaften Unterstützung von Java im Entwicklungswerkzeug Domino Designer, der geringen Debugging-Möglichkeiten und zum anderen auch an der veralteten Java-Unterstützung im properitären Lotus Notes Client. Erst seit dem Release 8.5, das Anfang 2009 erschienen ist, wird wie vom Domino Server die Java-Version 1.6.0 unterstützt.

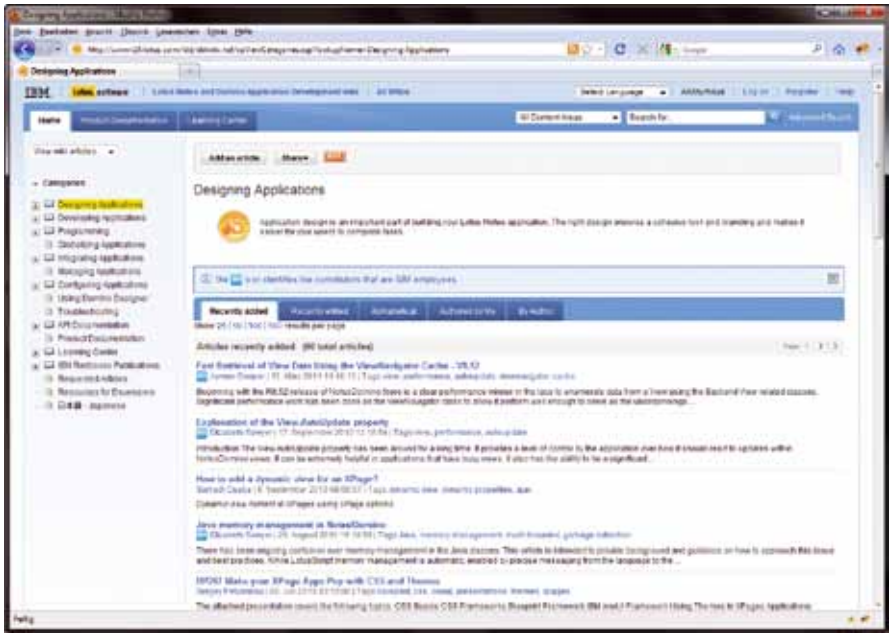


Abbildung 1: Benutzerschnittstelle einer typischen XPages-Anwendung (Wiki) [4]

Gegenwart

Mit dem aktuellen Release 8.5 wurde mit XPages eine neue Technologie zur Entwicklung von Anwendungen eingeführt. XPages basiert auf Java Server Faces (JSR 127) der Version 1.1, wobei IBM einige Änderungen der Versionen 1.2 und 2.0 ebenfalls in XPages integriert hat [3]. XPages werden als Bestandteil des Domino Servers sowie des Domino Designer ausgeliefert. Eine XPage ist ein Gestaltungselement einer Lotus-Notes/Domino-Anwendung, in der CSS, JavaScript, HTML und XSP-Tags kombiniert sind, sodass daraus eine eige-

ne Website-Komponente entwickelt werden kann (siehe Abbildung 1). Eine XPage wird als XML-Datei gespeichert und bei der Ausführung in Java übersetzt. Waren die XPages zu Anfang noch ausschließlich für die Verwendung im Webbrowser konzipiert, wurden sie mit dem Release 8.5.1 im Oktober 2009 auch für die Nutzung im Lotus-Notes-Client freigegeben.

Domino Designer

Die Entwicklung von XPages-Anwendungen erfolgt mit dem Domino Designer Version 8.5 (siehe Abbildung 2), einer Entwicklungsumgebung basierend auf der Eclipse IDE (Ganymede-Release). Gegenüber der Standard Eclipse IDE enthält der Domino Designer eine Vielzahl weiterer Features und Plugins, etwa zur Entwicklung der XPages.

Der Domino Designer ist kostenlos (siehe http://developer.lotus.com), jedoch nur auf Windows XP, Windows Vista oder Windows 7 ablauffähig. Eine Version für Linux oder Macintosh ist nicht verfügbar.

Eine Lotus Notes/Domino-Anwendung besteht – unabhängig vom verwendeten Client, mit dem auf die Anwendung zugegriffen wird – aus der Benutzerschnittstelle, der Geschäftslogik und der Datenbank. Eine Anwendung hat immer eine .nsf-Datei-Endung („Notes storage facility“) und kann nur über den Lotus-Notes-Client oder den Domino-Server ausgeführt werden. Während in einer Nicht-XPages-Anwendung für Lotus Notes/Domino das Model-View-Controller-Konzept nicht angewendet wird, greift XPages darauf zurück. Die Speicherung der Daten erfolgt nicht mit einem relationalen Datenbankmodell, sondern über ein dokumentenorientiertes Datenbankmodell. Dementsprechend sind die Daten und Gestaltungselemente in Dokumenten gespeichert. Das Layout des Dokuments wird über eine Form oder eine XPage definiert. Dort kann auch Code hinterlegt werden, der verwendet wird, um die Daten zu verwalten. Ansichten listen die Dokumente auf, auf die der Anwender zugreifen kann. Die eigentliche Geschäftslogik wird oftmals in Form von Skriptbibliotheken oder Agenten zumeist mit LotusScript oder Java programmiert. Ein Agent ist im Prinzip ein eigenständiges Programm, mit dem Funktionen entweder zeit- oder ereignisgesteuert ausgeführt werden. Agenten lassen sich sowohl im Hintergrund als auch durch den Anwender im Lotus Notes Client starten. Die Zeitablaufsteuerung kann ohne Programmierung konfiguriert werden.

Im Folgenden wird zur Veranschaulichung eine einfache Anwendung erstellt, mit der Kontakte verwaltet werden können. Als Funktion wird eine Schaltfläche zum Berechnen des Alters der Person hinzugefügt. Dazu wird zunächst eine neue Anwendung im Domino Designer erstellt. Es kann gewählt werden, ob eine komplett neue Anwendung erstellt wird oder die Anwendung auf der Gestaltung einer bestehenden Anwendung (Template) basieren soll. In diesem Fall werden alle Gestaltungselemente aus dem Template in die neue Anwendung kopiert. Die neue Anwendung erhält einen Namen und einen Dateinamen. Ferner wird angegeben, ob sie lokal oder auf einem Domino Server erstellt werden soll. Im Domino Designer ist nun in der linken Leiste die Anwendung mit ihren Gestaltungselementen dargestellt (siehe Abbildung 2).

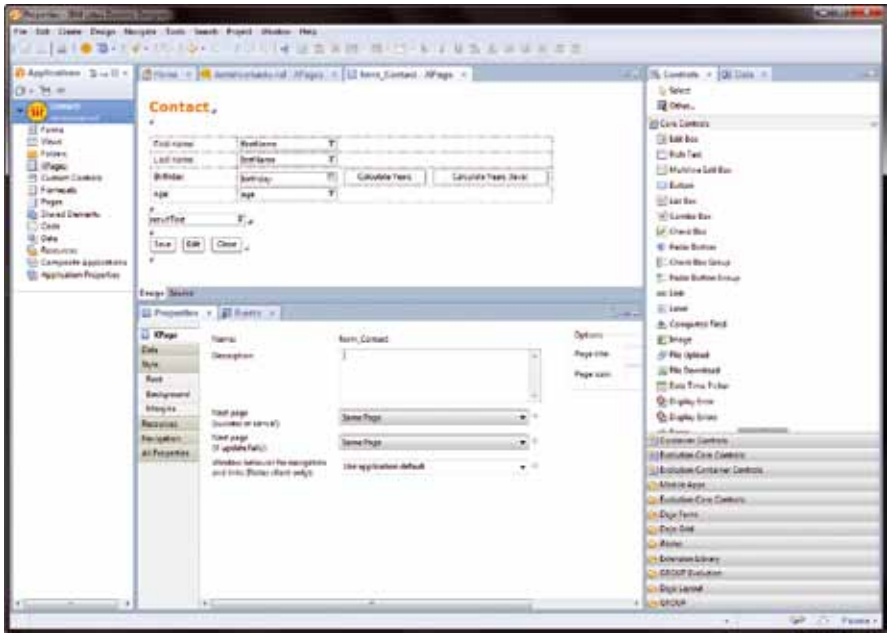


Abbildung 2: Domino Designer mit Entwicklungswerkzeugen für XPages

Innerhalb der Anwendung wird als Erstes eine Form erstellt, die die Felder zum Speichern der Daten enthält. Die Gestaltung der Form spielt keine Rolle, da die Visualisierung später durch eine XPage erfolgt. Über eine neu erstellte Ansicht lassen sich die mit der Form generierten Dokumente auflisten. In der Ansicht sind die Felder der Form als Spalten definiert. Jede Spalte zeigt die Werte des jeweiligen Feldes an. Anschließend wird basierend auf der Form eine XPage für die Anzeige der Dokumente erstellt. Diese erscheint im Editor-Bereich in der Mitte des Domino Designer. Über die Control-Palette, zumeist rechts neben dem Editor angeordnet, können Standard-Controls (wie Label, Field, Button, Table, View) oder selbst erstellte Controls (Custom Controls) per Drag & Drop auf die XPage platziert werden. Im Properties-Bereich unterhalb des Editors befinden sich unter anderem die Eigenschaften, Validierungsformeln und das Data Bindung. Zum Definieren der Funktionalität, etwa einer Schaltfläche, kann Code in Form von Simple Actions (beispielsweise für Standardaktionen, wie Speichern oder Schließen eines Dokumentes) oder über den Script Editor mit JavaScript hinterlegt sein.

Die XPage selbst ist in einem XML-Format gespeichert (siehe Listing 1), das über den Source Editor auch bearbeitet werden kann. Neben den Controls (wie label, inputText, button im Listing) enthält die XPage weitere Tags. Dazu können Data Sources (wie dominoDocument), Containers, View Resources, Converters, Validators, Simple Actions, Client-side Scripting und normale HTML-Tags gehören.

Hinweis: Das Listing 1 finden Sie unter http://src.ijug.eu/ja/1104/listing_xpages.txt

Im nächsten Schritt wird für die definierte Ansicht eine weitere XPage erstellt, in der die Ansicht integriert ist.

Damit die Anwendung im Webbrowser geöffnet werden kann, definiert man unabhängig von der Gestaltung und Programmierung der Anwendung die Zugriffsberechtigung über die in der Anwendung integrierte Zugriffskontrollliste. Hier können sowohl Personen, Gruppen oder Rollen zur Umsetzung des Sicherheitskonzepts der Anwendung angegeben sein. Anschließend kann die Anwendung auf einem Domino Server deployed werden.

Integration von Java

Die Funktionalität der Anwendung soll unter anderem mithilfe von Java-Code erweitert werden. Zum einen kann man Java-Code direkt in einem Server-side JavaScript (SSJS) verwenden. Dieses SSJS wird im Gegensatz zum Client-side JavaScript direkt auf dem Server ausgeführt. Zum anderen lassen sich Java-Klassen aus beliebigen JAR-Dateien oder Java-Bibliotheken verwenden. Listing 2 zeigt den Einsatz von Java direkt in einem SSJS, wo einfache Java-String-Objekte verwendet werden.

```
var resultName = new java.lang.String(„Your
name is „);
var resultAge = new java.lang.String(„ and
your age is „);
var name = getComponent(„firstName“).get-
Value() + „ „ + getComponent(„lastName“).
getValue();
var birthday = String(
getComponent(„birthday“).getValue() );
var year = parseInt(birthday.substring( birth-
day.length - 4));
var now = new Date();
var age = String( now.getFullYear() - year );
getComponent(„age“).setValue( age );
getComponent(„resultText“).setValue( re-
sultName + name + resultAge + age + „.“ );
```

Listing 2: Java-Code eingebunden in SSJS

Über die Eclipse Views lässt sich der Package Explorer öffnen (siehe Abbildung 5, linke Hälfte). Hierüber kann man im Ordner WebContent eigene Java-Klassen erstellen. Diese Java-Klassen können anschließend in die XPage eingebunden werden, beispielsweise für die Ausführung über eine Schaltfläche. Listing 3 zeigt eine einfache Java-Klasse, die im Code einer Schaltfläche aufgerufen wird, wie in Abbildung 5 angegeben ist. Die Java-Klasse wurde zuvor im Verzeichnis WebContent\WEB-INF\src abgelegt. Dieser Pfad muss anschließend im Java Build Path der Anwendung übernommen werden.

```
package com.pavone.demo.contact;

public class Age {

        public String getResultString(String
        name, String age){
            return „Your name is „ + name
            + „ and your age is „ + age +“!“;
        }

}
```

Listing 3: Java-Klasse „Age“

Damit nach dem Ermitteln des Alters das Ergebnis auf der Webseite angezeigt werden kann, ist eine Aktualisierung der Website erforderlich. Hier sind unter „Server Options“ zwei konfigurierbare Varianten der Aktualisierung angeboten. Zum einen ist ein Partial Update möglich, wenn nur ein Element aktualisiert werden muss. Da in dem Beispiel zwei Felder zu ändern sind („age“ und „resultText“), erfolgt ein Full Update.

Das Erstellen der Java-Klasse aus Listing 3 kann entweder direkt im Domino Designer erfolgen oder extern, um sie über eine JAR-Datei zu importieren. Die Erstellung dieser JAR-Datei kann jedoch auch direkt im Domino Designer erfolgen, da dieser die Eclipse IDE-Funktionen zur Verwaltung von Java-Projekten unterstützt. Ein so angelegtes Java-Projekt wird es im Gegensatz zum erzeugten Java-Code nicht in der Anwendung, sondern wie von der Eclipse IDE gewohnt, direkt im Workspace-Verzeichnis gespeichert. Um auf die Notes-Java-Klassen in dem Java-Projekt zugreifen zu können, muss man diese über die Datei „NCSO.jar“ importieren. Diese Bibliothek ist im Verzeichnis „\domino\java“ vorhanden. In dieses Projekt können die zuvor im Domino Designer erzeugten Java-Klassen eingebunden werden. Anschließend kann man daraus eine JAR-Datei generieren. Diese wird in der XPages-Anwendung im „WEB-INF\lib“-Ordner importiert, wodurch sie wieder direkt in der XPages-Anwendung (und nicht im lokalen Workspace-Verzeichnis) gespeichert ist. Sinnvoll ist dieses Vorgehen natürlich nur, wenn die JAR-Datei bereits existiert und nicht mehr angepasst werden muss, andernfalls sind die Änderungen zu aufwändig.

Die Open-Source-Community OpenNTF.org

Durch die Unterstützung von Custom Controls in XPages wuchs die Open-Source-Community OpenNTF.org, die seit 2009 stärker vom Hersteller IBM unterstützt wurde, schnell an. Mittlerweile findet sich dort neben eigenständigen Anwendungen auch eine Vielzahl sofort einsetzbarer Custom-Controls, die in eigene Anwendungen integriert werden können. Auch die Veröffentlichung vieler Komponenten auf Basis der Apache V2-Lizenz oder GPL 3-Lizenz trägt zur steigenden Beliebtheit der OpenNTF.org-Community bei. IBM stellt mittlerweile frühzeitig neue Entwicklungen ein, wie beispielsweise die Xpages-Extension-Library, die häufig benötigte Out-of-the-box-Elemente beinhaltet, welche im aktuellen Release 8.5.2 des Domino Designer integriert wurde. Als weitere Neuerung ist eine Erweiterung der Extension Library verfügbar, in der REST-Services für Domino und XPages enthalten sind. Ebenso wurde ein Plugin für die Anbindung von Apache Subversion Repositories bereits veröffentlicht, das im nächsten Release 8.5.3 des Domino Designer (angekündigt für Q3/2011) integriert sein soll [6].

Fazit

Bisher war die Entwicklung von Lotus-Notes/Domino-Anwendungen vornehmlich für Entwickler interessant, die sich schon längere Zeit damit beschäftigt haben. Durch das XPages-Framework sowie die kostenlose Verfügbarkeit des Domino-Designer, die neuen Möglichkeiten des Domino-Servers (unter anderem OSGI-Fähigkeit) und die weiterhin einfache Erstellung von Anwendungen basierend auf dem Rapid-Prototyping-Konzept ist die Entwicklung nun auch für Java- und Java EE-Entwickler interessant. Auch die mittlerweile stetige

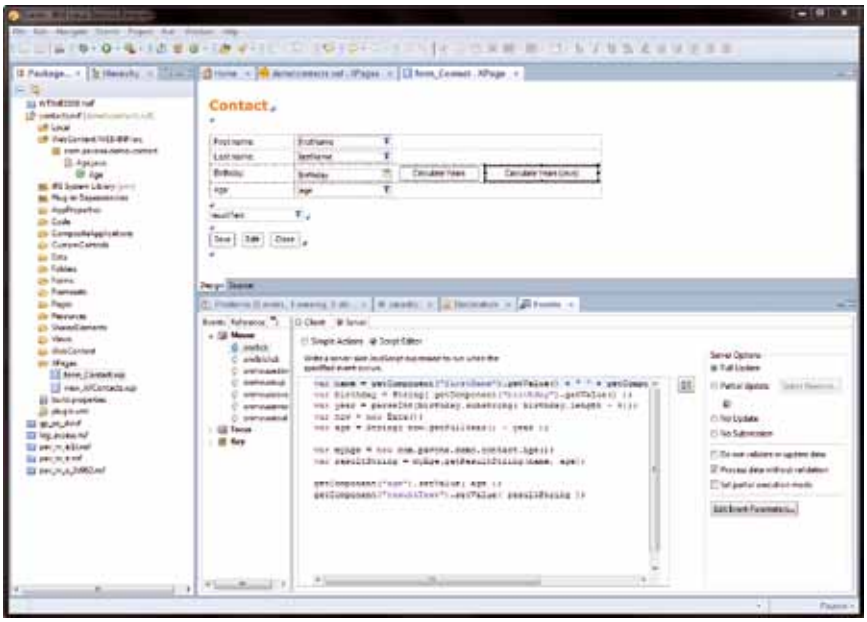


Abbildung 5: Aufruf der Java-Klasse „Age" (siehe Listing 3) in SSJS

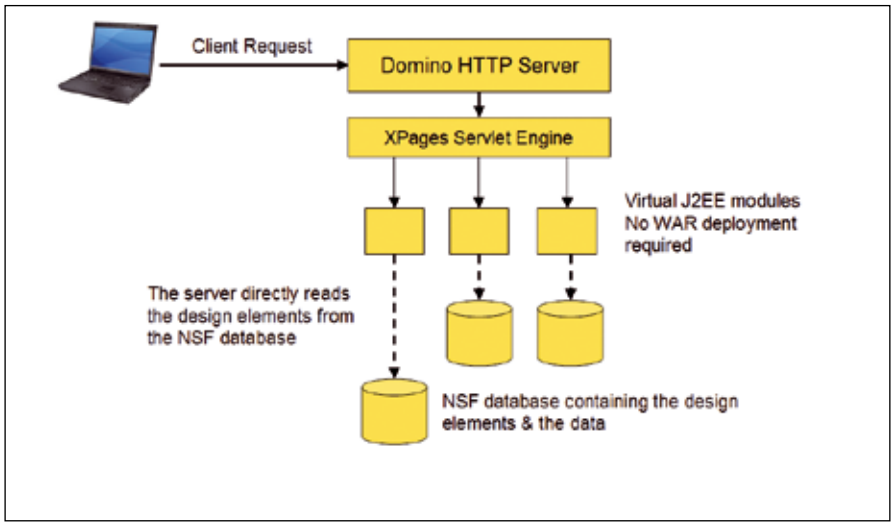


Abbildung 6: Architektur der XPages-Unterstützung [5]

Bereitstellung von zukünftigen Release-Erweiterungen von IBM auf OpenNTF.org unterstützt die Entwickler-Community. Ein guter Startpunkt ist die mit vielen Beispielen, Tutorials und Ressourcen angereicherte Webseite „XPages.info".

Weiterführende Links

- OpenNTF.org Community: http://www.openntf.org
- Informationsmaterialien und Demo-Anwendungen: http://www.xpages.info
- Developer Materialien von IBM: http://www-10.lotus.com/ldd/ddwiki.nsf
- Developer Works: http://developer.lotus.com
- Podcasts zu XPages: http://thexcast.net/
- Blog-Verzeichnis zu Lotus Notes/Domino-Themen: http://www.planetlotus.org

Literatur:

[1] Wikipedia:http://de.wikipedia.org/wiki/Lotus_Notes, 2011

[2] GBS: GBS Transformer brings Lotus Notes Client applications to the Web, http://www.gbs.com/de/press/gbs-en--launch-event-Transformer20, 2011

[3] Martin Donnelly, Mark Wallace, Tony McGuckin: Mastering XPages, IBM Press, 2011

[4] Lotus Notes and Domino Application Development Wiki: http://www-10.lotus.com/ldd/ddwiki.nsf, 2011

[5] John Head, Philippe Riand: Xpages: http://www.slideshare.net/johnhead/domino-x-pages-85, Seite 19, 2009

[6] Developer Works: Notes/Domino Fix List, http://www-10.lotus.com/ldd/r5fixlist.nsf/%28Progress%29/853, 2011

Dr. Rolf Kremer
rolf.kremer@de.gbs.com

Dr. Rolf Kremer ist Leiter der Produktentwicklung bei der PAVONE AG, eine Division der GBS (http://www.pavone.de). Er ist zertifizierter IBM Domino Developer und Administrator der Versionen 4 bis 8.5. Seit 1994 beschäftigt er sich mit der Entwicklung von Software auf Basis von IBM Lotus Notes/Domino und seit einigen Jahren mit Java EE und JavaScript-Frameworks. Er führt ein privates Weblog unter http://www.r-k.net

KURZMELDUNG

Zuwachs für die Sprachfamilie

Erst vor einigen Wochen ist Gavin King (der geistige Vater von Hibernate) mit dem neuen Projekt „Ceylon" an die Öffentlichkeit getreten. Das Projekt wird seit Längerem von einem Team bei Red Hat vorangetrieben. Es soll ein neues SDK plus Programmiersprache schaffen, lauffähig auf der Java-Plattform, aber wohl nicht vollständig kompatibel mit Java-Code. Jetzt kündigt JetBrains (die Firma hinter IntelliJ Idea, TeamCity etc.) das „Projekt Kotlin" an, eine statisch typisierte, Java-kompatible Sprache. Die Vielfalt der Sprachen für die Java-Plattform wird langsam unübersichtlich. Aber spannend ist es auf jeden Fall, das Rennen um die Java-Nachfolge zu verfolgen.
http://in.relation.to/Bloggers/Gavin
http://confluence.jetbrains.net/display/Kotlin/FAQ

„Bereits jetzt zählen wir zu den führenden Java-Magazinen im deutschsprachigen Raum …“

Die Zeitschrift Java aktuell feiert mit dieser Ausgabe ihr einjähriges Jubiläum. Chefredakteur Wolfgang Taschner sprach darüber mit Fried Saacke, dem Vorstandsvorsitzenden des Interessenverbunds der Java User Groups e. V. (iJUG).

Mit welchen Zielen wurde der iJUG gegründet?

Nach der Sun-Übernahme durch Oracle vor rund eineinhalb Jahren haben sechs Java-Usergroups aus Deutschland zusammen mit der DOAG Deutsche ORACLE-Anwendergruppe e.V. den Interessenverbund der Java User Groups e.V. (iJUG) gegründet. Mittlerweile sind die Java User Group Ostfalen, die Sun User Group Deutschland sowie die Schweizer Oracle User Group hinzugekommen; weitere Mitgliedschaften sind in Vorbereitung. Ziel des iJUG ist die umfassende Vertretung der gemeinsamen Interessen der Java-Anwendergruppen sowie der Java-Anwender im deutschsprachigen Raum, insbesondere gegenüber Entwicklern, Herstellern, Vertriebsunternehmen sowie der Öffentlichkeit. Zudem werden die einzelnen iJUG-Mitgliedergruppen insbesondere in ihren regionalen Aktivitäten unterstützt. Bei Wahrung der Eigenständigkeit der Mitglieder findet neben dem Informations- und Erfahrungsaustausch eine gemeinsame Öffentlichkeitsarbeit statt und die Netzwerkbildung wird gefördert. Eine der ersten Aktivitäten war die Herausgabe der Zeitschrift Java aktuell.

Wie hat sich die Java aktuell innerhalb des ersten Jahres entwickelt?

Bereits jetzt zählen wir zu den führenden Java-Magazinen im deutschsprachigen Raum. Der Verkauf am Kiosk entwickelt sich prächtig, bei den Abonnements wollen wir noch weiter zulegen. Am meisten freut es mich, dass wir viele gute Autoren gewinnen konnten, sodass die Zeitschrift für alle Java-Entwickler sehr interessante Inhalte bietet.

Wie ist das redaktionelle Konzept der Java aktuell?

Wesentliches Ziel von Java aktuell ist es, den Leser stets aktuell über die Entwicklungen im Umfeld von Java zu informieren. Dafür pflegt der iJUG enge Beziehungen zu wichtigen Herstellern wie Oracle, IBM, Red Had oder SAP. Wir legen besonderen Wert darauf, dass der Leser gut und effizient durch das Magazin geführt wird und schnell die wesentlichen Informationen findet. Alle eingereichten Artikel sind in einem Wiki abgelegt. Ein Redaktionsteam, bestehend aus mehreren iJUG-Mitgliedern, bewertet die Texte vor der Veröffentlichung. Bei Bedarf arbeiten die Autoren entsprechend nach. So ist gewährleistet, dass die Artikel eine hohe Qualität erreichen.

Gibt es neben der Herausgabe der Zeitschrift noch weitere Aktivitäten des iJUG?

Der iJUG ist auf großen Java-Veranstaltungen wie der Jazoon in Zürich, dem Java Forum in Stuttgart, den Source Talk Tagen in Göttingen sowie der DOAG 2011 Konferenz + Ausstellung in Nürnberg mit einem Stand vertreten. Dort greifen wir die Stimmung in der Community auf, bündeln die Probleme und kommunizieren sie gegenüber Oracle. Das hat eine ganz andere Wirkung, als wenn sich jede Java-Usergroup einzeln an den Hersteller wenden würde. Wir bringen unsere Anliegen auch in die Öffentlichkeit, beispielsweise durch Pressemeldungen zu Themen wie „Spaltung der Java-Community“, „Qualität der JavaOne“ oder „Mangelnde Informationspolitik seitens Oracle“.

Wie kann ein Anwender seine Erfahrungen in der Java aktuell veröffentlichen?

Er braucht nur vorab einen Abstract an redaktion@ijug.eu zu schicken und kann mit dem Schreiben loslegen, sobald er grünes Licht für den Artikel erhalten hat.

Fried Saacke, Vorstandsvorsitzender des Interessenverbunds der Java User Groups e.V. (iJUG)

Leichtgewichtige Authentifizierung mit OpenID

Sebastian Glandien, Acando GmbH

OpenID ist ein Standard zur dezentralen Anmeldung an Web-Anwendungen. Wurde bisher die Überprüfung einer Anmeldung immer durch die eigene Anwendung vorgenommen, delegiert OpenID dies an einen vertrauenswürdigen Dritten. Somit kann sich die Nutzerbasis eines Partners schnell in Synergie-Effekten auszahlen. OpenID schreibt die Ausstellung einer beglaubigten Nutzeranmeldung, deren Überprüfung und die Abfrage von zusätzlichen Nutzerattributen fest. Das offene Protokoll ist eine leichtgewichtige Variante zur einfachen Zusammenarbeit, zur Interaktion und zum Datenaustausch über Anwendungsgrenzen hinweg.

OpenID beschreibt die dezentrale Anmeldung an Webdiensten mittels einer URL-basierten Identität. Das offene Protokoll wurde im Jahr 2005 von Brad Fitzpatrick in der Version 1.0 entwickelt. Der Fokus für OpenID 1.0 lag vorrangig auf der Unterstützung der Authentifizierung von Nutzern und deren Registrierung. Erst in der Version 2.0, die im Dezember 2007 festgeschrieben wurde, ist der Austausch von Identitäts-Attributen mit aufgenommen worden. Im Jahr 2007 wurde ebenfalls die OpenID Foundation gegründet und übernimmt seitdem die Vermarktung von OpenID. Der aktuelle Vorschlag zur Weiterentwicklung ist OpenID Connect. OpenID wird dabei auf der Basis von OAuth 2.0 aufgesetzt, sodass die Implementierung vereinfacht und zusätzlich die Anmeldung in Desktop-Anwendungen möglich ist. Weiterhin soll zukünftig auch die eigene E-Mail-Adresse als Identität genutzt werden können.

Vorangetrieben durch Firmen wie Yahoo, Google, IBM, Myspace und Facebook gewinnt der Standard immer mehr an Verbreitung. Durch sie wird der Standard eingesetzt und die Nutzerbasis stetig erweitert. Aktuell kommt OpenID in mehr als 50.000 Websites zum Einsatz. Unterschiedliche Webportale profitieren somit von den rund einer Milliarde benannten OpenID-Identitäten der Global Player (siehe Login in Abbildung 1).

Die technischen Möglichkeiten, eine OpenID eines anderen Anbieters zu verwenden beziehungsweise eine selbst bereitzustellen, sind denkbar einfach. Für verschiedene Entwicklungsumgebungen stehen verschiedene Bibliotheken, Module oder deployfähige Apps bereit, etwa für Apache 2, C#, Coldfusion, Haskell, Java, Perl, PHP, Python, Ruby und Squeak/Smalltalk. Für die folgenden Beispiele wurde die OpenID 2.0 Java Bibliothek openid4java verwendet.

Abbildung 1: Login einer Web-Anwendung mit OpenID-Unterstützung (Quelle: http://openid.net/)

Anmeldung mit einer OpenID

Am OpenID-Anmeldeprozess sind der Nutzer als Akteur, identifiziert durch die OpenID, und die beiden Dienste, der OpenID-Consumer oder auch als Relying Party bezeichnet und der Identity-/OpenID-Provider, beteiligt. Der Nutzer steuert den gesamten Anmeldeprozess. Er gibt die gültige OpenID an, meldet sich gegebenenfalls mit seiner Nutzername/Passwort-Kombination an, autorisiert den gewählten OpenID-Provider und bestätigt bei Bedarf die Abfrage von Nutzerattributen. Jeder Anmeldeschritt ist dabei transparent und durch den Nutzer einzeln steuerbar. Der OpenID-Consumer bezeichnet die Web-Anwendung, welche die OpenID-Anmeldung nutzen möchte. Sie fordert die Eingabe der OpenID ein, initiiert den Anmeldeprozess und überprüft dessen Ergebnis. Der OpenID-Provider stellt die OpenID URL-basiert bereit. Auf Seiten des Providers erfolgen die Validierung der Anmeldeinformation (etwa Nutzernamen/Passwort) und die Übermittlung des signierten Ergebnisses an den OpenID-Consumer.

Der ideale OpenID-Anmeldevorgang beginnt mit einer Authentifizierungsanfrage des Consumers für den Nutzer. Dieser wählt OpenID zur Anmeldung aus und gibt seine OpenID an. Der Consumer überprüft daraufhin das Format der angegebenen OpenID und startet den Discovery Prozess. Dabei wird mittels XRDS oder Link-Verweisen die OpenID nach dem OpenID-Provider Endpunkt angefragt. Der Consumer kann mit dieser Information den Anmeldevorgang starten. Dazu werden, wie im Protokoll beschrieben, Geheimnisse und Zeitstempel zwischen dem Consumer und dem

Provider ausgetauscht, um die Authentizität und Vertrauenswürdigkeit des Informationsaustausches sicherzustellen. Wird der OpenID-Provider zum ersten Mal innerhalb einer gültigen Browsersession vom Consumer zur Anmeldung verwendet, werden vom Provider die Anmelde-Informationen abgefragt. Erst wenn diese erfolgreich validiert sind, kann diese Anmeldung dem Consumer bestätigt werden. Um den Zugriff unberechtigter Web-Anwendungen auf eine aktuelle OpenID-Session zu unterbinden, kann optional eine Autorisierung des Zugriffs der anfragenden Web-Anwendung erfolgen. Die Übermittlung einer gültigen OpenID-Provider-Anmeldung erfolgt sicher signiert an den Consumer. Dieser vertraut den Rückgabewerten, führt eine Consumer-seitige Anmeldung durch und übernimmt gegebenenfalls übermittelte Profilinformationen. Der Zugriff auf geschützte Inhalte ist anschließend freigegeben (siehe Abbildung 2).

Directed Identity

Der ursprüngliche Ansatz von OpenID ging davon aus, dass jeder Nutzer seine OpenID in Form einer URL kennt. So sind beispielsweise die URLs des eigenen Blogs (http://<myname>.livejournal.com), einer Homepage (<www.myopenid-homepage.com> oder eines speziellen OpenID-Providers (<myname>.myopenid.com) als OpenID möglich. Da es sich bei einer URL um eine mehr oder weniger komplexe Information handelt, welche für den Nutzer zusätzlich und zudem nur schwer zu merken ist, gibt es die Möglichkeit der Directed Identity. Dabei muss man beim Login nicht eine vollständige OpenID in Form einer URL eingeben, sondern kann einen speziellen OpenID-Provider (Google, Yahoo etc.) oder die Domain seines Accounts (google.com, yahoo.com etc.) eingeben. Der OpenID-Consumer löst abhängig von dieser Information den anzufragenden OpenID-Provider auf. Die Funktionalität Directed Identity ist Bestandteil der OpenID-2.0-Spezifikation.

Der eigene OpenID-Provider

Der OpenID-Consumer und der OpenID-Provider stellen in einer produktiven Umgebung zwei voneinander getrennte Dienste dar und sind somit in verschiedenen Anwendungen implementiert. Für die im Folgenden aufgezeigten OpenID-Beispiele wird im ersten Schritt ein einfacher OpenID-Provider, basierend auf dem simple-openid Beispiel der openid4java Bibliothek, entwickelt. Alternativ kann an dieser Stelle auch ein gültiger OpenID-Provider, wie Google (https://www.google.com/accounts/o8/id), MyOpenID, CloudID oder OpenID.org, verwendet und die Implementierung eines eigenen Providers übersprungen werden.

Grundlage für den OpenID-Provider ist das „simple-openid"-Beispiel von openid-4java 0.9.6. Für den Build und Start des

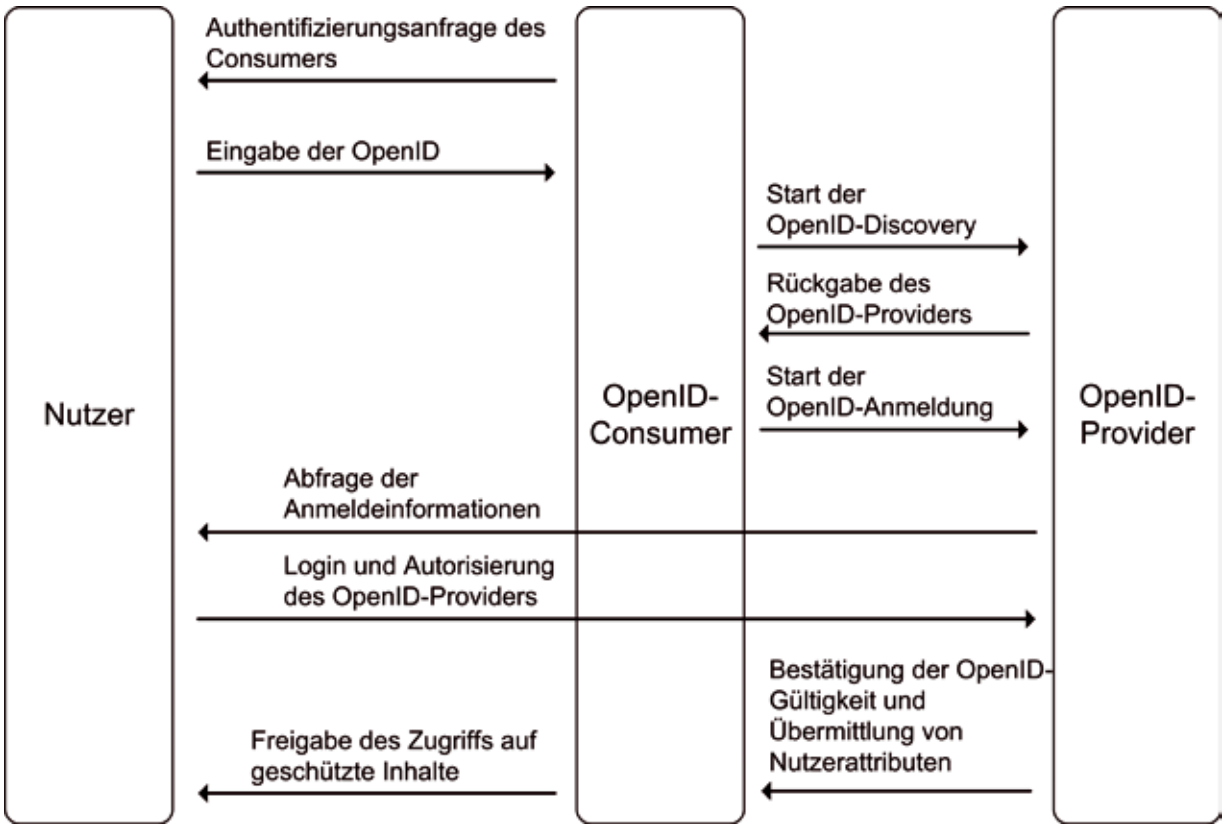


Abbildung 2: Ablauf einer OpenID-Anmeldung

Beispiels sind mindestens Java 1.4 und Maven2 erforderlich. Zum Starten der Anwendung muss ein Patch der POM erfolgen. Dazu ist die Datei „openid4java-0.9.6.662\samples\simple-openid\pom.xml“ wie in Listing1 anzupassen.

```
<dependencies>
  …
  </dependency>
  <dependency>
    <groupId>org.openid4java</groupId>
-   <artifactId>openid4java</artifactId>
+    <artifactId>openid4java-consumer</
     artifactId>
    <version>${version}</version>
+    <type>pom</type>
  </dependency>
</dependencies>
```

Listing 1: POM-Patch simple-openid

Der Start der Anwendung erfolgt mit „mvn jetty:run“. Die bereitgestellte OpenID des OpenID-Providers ist über „http://localhost:8080/simple-openid/user.jsp“ aufrufbar. Als Ergebnis kommt eine XRDS-Datei zurück, welche die Url zum OpenID-Provider enthält (siehe Listing 2).

```
<?xml version="1.0" encoding="UTF-8"?>
<xrds:XRDS xmlns:xrds="xri://$xrds"
  xmlns:openid=http://openid.net/xmlns/1.0
  xmlns="xri://$xrd*($v*2.0)">
  <XRD>
    <Service priority="0">
      <Type>http://openid.net/signon/1.0</
      Type>
      <URI>http://localhost:8080/
simple-openid/provider.jsp</URI>
    </Service>
  </XRD>
</xrds:XRDS>
```

Listing 2: XRDS-Response einer OpenID

Der OpenID-Provider des simple-openid Beispiels prüft keine Anmelde-Informationen, sodass wir diesen dahingehend erweitern müssen. Der Link „Click To become logged in and authorize“ wird durch eine Login-Box ersetzt. Dafür ändert man die „provider_authorization.jsp“ (siehe Listing 3, Seite 49).

Damit werden beim Zugriff auf den OpenID-Provider eine Nutzername/Passwort-Kombination abgefragt und geprüft sowie eine vorhergehende Anmeldung wiedererkannt. Die Überprüfung einer erfolgreichen Anmeldung am OpenID-Provider erfolgt mittels Cookie. Wurden die Anmeldeinformationen einmal erfolgreich validiert, wird das Sessioncookie „openIDLoggedIn“ mit dem Wert „true“ gespeichert. Bei einer wiederholten Verwendung des OpenID-Providers wird dieses Cookie ausgewertet, sodass keine erneute Anmeldung notwendig ist (Single-sign-on). Ist kein openIDLoggedIn-Cookie mit dem Wert „true“ gespeichert, erscheint die Login-Maske. Mit dem Nutzernamen „user“ und dem Passwort „user123“ kann man sich am OpenID-Provider anmelden.
Der implementierte OpenID-Provider kann in der aufgezeigten Beispielimplementierung zur Anmeldung verwendet werden. Die relevanten Url-Endpunkte sind:

- *OpenID*
- *http://localhost:8080/simple-openid/user.jsp*
- *OpenID-Provider*
- *http://localhost:8080/simple-openid/provider.jsp*

Sicher anmelden mit OpenID

Um in einer eigenen Web-Anwendung OpenID zur Anmeldung verwenden zu können, ist ein sogenannter „OpenID-Consumer“ zu implementieren. Ausgangsbasis dafür ist das „consumer-servlet“-Beispiel der „openid4java“-Bibliothek. Wie bereits für den OpenID-Provider beschrieben, ist der Patch (siehe Listing 4) auf die Datei „openid4java-0.9.6.662\samples\consumer-servlet\pom.xml“ anzuwenden.

```
<dependencies>
  …
  <dependency>
    <groupId>org.openid4java</groupId>
    <artifactId>openid4java-consumer</
artifactId>
    <version>${version}</version>
+    <type>pom</type>
  </dependency>
</dependencies>
```

Listing 4: POM-Patch consumer-servlet

Die Anwendung wird mit dem Kommando „mvn -Djetty.port=7080 jetty:run“ gestartet und kann über die URL „http://localhost:7080/consumer-servlet/“ aufgerufen werden. Es stehen drei Beispiele zur Demonstration der OpenID-Anmeldung, Simple-Registration-Extension und Attribute-Exchange-Extension bereit.

Um die einfache OpenID-Anmeldung testen zu können, gibt man im Eingabefeld des Sample 1 die eigene OpenID (beispielsweise http://localhost:8080/simple-openid/user.jsp) ein. Daraufhin wird die Anmeldung vom ConsumerServlet.java initiiert (siehe Listing 5). Im ersten Schritt wird dabei der OpenID-Provider anhand der eingegebenen OpenID aufgelöst (OpenID Discovery). Mit der korrekten URL des OpenID-Providers entsteht daraufhin ein sogenannter „AuthRequest“, der per Redirect aufgerufen wird. Die relevanten OpenID-Werte werden an die URL des Providers als Parameter angehängt (siehe Listing 6).

```
List discoveries = manager.
discover(userSuppliedString);
DiscoveryInformation discovered = manager.
associate(discoveries);
…
AuthRequest authReq = manager.
authenticate(discovered, returnToUrl);
…
httpResp.sendRedirect(authReq.
getDestinationUrl(true));
```

Listing 5: Initiierung der OpenID-Anmeldung

Nach Anmeldung auf Seitenn des OpenID-Providers, wird die übermittelte openid.return_to-url ergänzt um die Parameter des Providers aufgerufen, sodass die Consumer-seitige Auswertung erfolgen kann.

```
http://local.acandolocal.de:8080/simple-ope-
nid/provider.jsp?

openid.identity=http://local.acandolocal.
de:8080/simple-openid/user.jsp&
openid.return_to=http://localhost:7080/
consumer-servlet/consumer?is_
return=true&openid.rpnonce=2011-07-
08T13%3A36%3A30Z0&openid.rpsig=evH0
BD001OT0dXAhy1r5yOXg1sytD7Hu6Lqhv
TLLvk0%3D&
openid.trust_root=http://localhost:7080/
consumer-servlet/consumer?is_return=true&
openid.mode=checkid_setup&
openid.ns.ext1=http://openid.net/srv/ax/1.0&
openid.ext1.mode=fetch_request
```

Listing 6: Initialer Aufruf des OpenID-Providers mit URL-Parametern

```
<h1>Login to OpenID Provider</h1>
<%
Cookie openIDCookie = null;
Cookie[] cookieList = request.getCookies();
for (Cookie cookie : cookieList) {
  if („openIDLoggedIn“.equals(cookie.getName()) && „true“.equals(cookie.getValue())) {
    openIDCookie = cookie;
  }
}

if (request.getParameter(„action“) == null && openIDCookie == null) {
  String site=(String) (openidrealm == null ? openidreturnto : openidrealm);
  if (request.getParameter(„error“) != null && !““.equals(request.getParameter(„error“))) {
    out.println(„<span style=\“color:red\“>“ + request.getParameter(„error“) + „</span><p>“);
  }
%>
<form action=“?action=authorize“ method=“POST“>
Username: <input type=“text“ name=“username“ size=“12“ maxlength=“12“><p>
Password: <input type=“password“ name=“password“ size=“12“ maxlength=“12“> <input type=“submit“ value=“Login“>
</form>
<p>
<strong>ClaimedID:</strong> <pre><%= openidclaimedid%></pre>
<strong>Identity:</strong> <pre><%= openididentity %> </pre>
<strong>Site:</strong> <pre> <%= site %></pre>
</p>

<!-- Click <a href=“?action=authorize“ id=“login“>To become logged in and authorize</a> -->
<%
} else {
  if (openIDCookie != null) {
    session.setAttribute(„authenticatedAndApproved“, Boolean.TRUE); // No need to change openid.* session vars
    response.sendRedirect(„provider.jsp?_action=complete“);
  } else if (request.getParameter(„username“) != null && request.getParameter(„password“) != null) {
    if („user“.equals((String) request.getParameter(„username“)) && „user123“.equals((String) request.getParameter(„password“))) {
      Cookie ssoCookie = new Cookie(„openIDLoggedIn“, „true“);
      response.addCookie(ssoCookie);
      session.setAttribute(„authenticatedAndApproved“, Boolean.TRUE); // No need to change openid.* session vars
      response.sendRedirect(„provider.jsp?_action=complete“);
    } else {
     response.sendRedirect(„provider_authorization.jsp?error=Wrong username/password given!“);
    }
  } else {
    response.sendRedirect(„provider_authorization.jsp?error=No username/password given!“);
  }
}
%>
```

Listing 3: Einbindung einer Login-Maske in die „provider_authorization.jsp“

```
ParameterList response = new ParameterList(httpReq.getParameterMap());

DiscoveryInformation discovered = (DiscoveryInformation) httpReq.getSession().
getAttribute(„openid-disc“);
…
VerificationResult verification = manager.verify(receivingURL.toString(), response, discovered);

Identifier verified = verification.getVerifiedId();
if (verified != null) {
  …
  return verified; // success
}
```

Listing 7: Verarbeitung des OpenID-Response, Auszug aus „ConsumerServlet.verifyResponse(„)

```
openid.mode=checkid_setup
…
openid.ns.ax=http://openid.net/srv/ax/1.0
openid.ax.mode=fetch_request
openid.ax.type.name=http://axschema.org/namePerson
openid.ax.type.email=http://axschema.org/contact/email
openid.ax.type.web=http://axschema.org/contact/web/default
openid.ax.type.country= http://axschema.org/contact/country
```

Listing 8: OpenID-Attribute-Exchange URL-Parameter

```
FetchRequest fetch = FetchRequest.createFetchRequest();

fetch.addAttribute(„FirstName“, „http://schema.openid.net/namePerson/first“, true);
fetch.addAttribute(„LastName“, „http://schema.openid.net/namePerson/last“, true);
fetch.addAttribute(„Email“, „http://schema.openid.net/contact/email“, true);

fetch.setCount(„Email“, 1);

AuthRequest req = _consumerManager.authenticate(discovered, return_to);
req.addExtension(fetch);
```

Listing 9: Abfrage von OpenID-Nutzerattributen mittels „openid4java“

Der „org.openid4java.consumer.ConsumerManager“ verifiziert die Überprüfung der aufgerufenen OpenID-Consumer-URL (openid.return_to-Parameter). Als Parameter wird die vollständige Consumer-URL mit Querystring, die ParameterList des HTTPRequests und die DiscoveryInformation, welche zu Beginn der Initialisierung über die OpenID aufgelöst wurde, übergeben (siehe Listing 7).

Als Verified-Identifier kommt die OpenID „http://localhost:8080/simple-openid/user.jsp“ zurück.

OpenID Attribute Exchange

Das Ergebnis einer erfolgreichen OpenID-Anmeldung ist die verifizierte OpenID. Eine zum Login angegebene OpenID wird durch den Provider bestätigt. Für viele Anwendungsfälle ist es jedoch praktisch, zusammen mit einer Identitätskennung gleichzeitig Nutzerattribute übermittelt zu bekommen. Die Anfrage vom Consumer und die Übermittlung durch den Provider dieser Daten erfolgt mittels „OpenID Attribute Exchange“. Der „checkid_setup-Request“, der prüfen soll, ob ein Nutzer Besitzer einer bestimmten OpenID ist, wird dabei um die zusätzlichen Felder ergänzt. Für einen Attribute Exchange Request werden dafür die folgenden Felder an den AuthRequest angehängt. Die angeforderten Attribute werden über die Parameter openid.ax.type.[alias] gesteuert (siehe Listing 8).

Um den FetchRequest an den AuthRequest anzuhängen, sind die Codezeilen aus dem Listing 9 zu verwenden.

Sample 3 der Beispielanwendung bietet die Möglichkeit, den Anwendungsfall „OpenID Attribute Exchange“ nachzuvollziehen. Als OpenID sollte dafür eine Identitäts-Url eines OpenID-Providers in der Version 2.0, wie die Google-OpenID https://www.google.com/accounts/o8/id zum Einsatz kommen.

Fazit

OpenID ist als Standard zum Plattformübergreifenden Single-sign-on in Web-Anwendungen kaum noch wegzudenken. Insbesondere seine Einfachheit hinsichtlich des Protokolls als auch dessen Implementierung zeichnet OpenID gegenüber SAML aus. Der Anwendungsentwickler erhält durch eine Vielzahl an Bibliotheken die Fähigkeit, schnell und unkompliziert den Protokoll-Stack zu implementieren und seine Anwendung OpenID-fähig zu machen.

Als Web-Anwendung mit kleiner Nutzerbasis ermöglicht es OpenID ohne viel Aufwand, neue Anwender für ein Online-Produkt zu gewinnen. Nutzer können sich mit einer bestehenden Kennung (Google, Facebook etc.) an der eigenen Anwendung anmelden, ohne einen umfangreichen Registrierungsprozess zu durchlaufen oder sich eine neue Nutzernamen/Passwort-Kombination ausdenken zu müssen. Zusätzlich werden Profildaten nur an einer Stelle hinterlegt und können gegebenenfalls aktualisiert beziehungsweise gesperrt werden.

Sebastian Glandien
sebastian.glandien@acando.de

Sebastian Glandien ist IT-Consultant bei der Acando GmbH mit dem Schwerpunkt Java-Enterprise-Anwendungen. Das besondere persönliche Interesse von ihm gilt den Themen „Identity- und Access-Management“, „Web-Service Security“ und „mobile Plattformen“.

Java-Problem-Determination mit der IBM Support Assistant Workbench

Marc Bauer, IBM Deutschland GmbH

Java- und JEE-Anwendungen werden immer komplexer. Die große Flexibilität, die die Verwendung verschiedener Frameworks mit sich bringt, hat als Preis den fremden und unbekannten Code. Gerade bei Problem-Analysen ist es daher oft mühsam, den Verursacher zuverlässig ausfindig zu machen. Schnelle Reaktionszeiten und die Auswertung von Problemsituationen sind allerdings kritisch für die Entwicklung und den Betrieb von Applikationen. Grund genug, sich mit der umfangreichen Toolsammlung der IBM Support Assistant Workbench auseinanderzusetzen.

Die schnelle und präzise Analyse von Fehlerszenarien ist einer der kritischsten Aspekte der Anwendungsentwicklung und auch im späteren Betrieb unerlässlich.

Ziel einer solchen Analyse ist es, den Verursacher-Code oder die genauen Umstände, die zu einem Problem geführt haben, zu ermitteln, um diese nachhaltig zu beseitigen. Da im Fehlerfall die Wiederaufnahme des Services im Vordergrund steht, rückt die Sammlung von Daten oft in den Hintergrund, die für eine Nachbereitung notwendig wären. Um dieses Problem zu beherrschen, sollte man sich im Vorfeld Gedanken machen, welche Fehlerbilder auftreten können, wie und mit welchen Werkzeugen sich diese in der Nachbereitung analysieren lassen und welche Daten dafür notwendig sind. Im Folgenden werden verschiedene Problemszenarien vorgestellt und aufgezeigt, mit welchen Analysedaten diese untersucht werden können und welche Werkzeuge dafür in der IBM Support Assistant Workbench bereitstehen.

Abbildung 1: Die IBM Support Assistant Workbench kann kostenfrei unter http://www.ibm.com/software/support/isa heruntergeladen werden

Ein häufiges Fehlerszenario ist der Out-of-Memory-Error, der auftritt, wenn der JVM kein Speicher mehr zur Verfügung steht. Dies kann unterschiedliche Ursachen haben, beispielsweise ist die Heap-Konfiguration zu klein oder für die erwartete Workload unpassend gewählt oder Memory-Leaks beziehungsweise andere programmatische Fehler verhindern die Freigabe von nicht mehr benutzten Objekten. Ebenso sind Szenarien denkbar, in denen als Folge einer Benutzer-Interaktion eine zu große Anzahl an Objekten erzeugt wird, für die der zur Verfügung stehende Speicher schlicht zu gering ist. Besonders Anwendungen mit benutzerdefinierten Suchanfragen an Datenbanken sind häufig anfällig für diese Problematik. Daher sollte bei der Auswertung einer Problemsituation stets mit der Analyse der Garbage-Collection begonnen werden, da sich hier viele Informationen verstecken, die die Problemanalyse erheblich erleichtern können.

Damit im Problemfall die richtigen Daten zur Verfügung stehen, sollten die Ausgaben der Garbage-Collection aktiviert sein, bei denen die JVM bei jedem Lauf der Garbage-Collection Informationen, wie Dauer und Heap-Layout vor und nach der Collection, protokolliert. Dies ist bei der IBM JVM mit dem Parameter „-Xverbose:gc" möglich. Um diese Daten einfach und schnell grafisch analysieren zu können, stehen in der IBM Support Assistant Workbench unter anderem das „IBM Pattern Modeling and Analysis Tool for Java Garbage Collector (PMAT)" und der „Garbage Collection and Memory Visualizer" zur Verfügung. Es sollte zunächst geprüft werden, ob die Größe des Heap für die Workload angemessen ist. In der Regel sollten nicht mehr als 70 Prozent des Heap nach einer Collection in den Spitzenlastzeiten belegt sein. Andernfalls muss hier entweder der Heap vergrößert oder die Anzahl lebender Objekte im Heap verringert werden.

Des Weiteren sollte der Heap keinen stetigen Anstieg vorweisen, da dies auf ein Memory-Leak schließen lässt, für dessen Analyse, ebenso wie zur Analyse eines sprunghaft ansteigenden Heap, ein Heap- oder besser noch ein System-Dump benötigt wird. Die entsprechenden Dumps können zum Fehlerzeitpunkt beispielsweise unter Verwendung der Dump-Agents der IBM JVM erzeugt werden. Dump-Agents können als Listener innerhalb der JVM verstanden werden, die in vorgegebenen Situationen beliebige Dumps zur Analyse erzeugen können. So kann beispielsweise bei einem Out-of-Memory-Error ein System-Dump erzeugt werden oder können beim Auftreten einer bestimmten Exception weitere Analysedaten, wie Speicher- und Variablen-Inhalte, gesammelt werden.

Zur Analyse von Heap- und System-Dumps steht das Memory-Analyzer-Tool bereit. Heap-Dumps sind Speicherabzüge, die Informationen über alle Objekte im Heap und deren Referenzen umfassen. System-Dumps beinhalten darüber hinaus auch Variablen-Inhalte und detaillierte Informationen über die Threads der JVM. Die Daten aus den Dumps können Ungereimtheiten aufdecken, beispielsweise vergessene Referenzen, die die Garbage-Collection von Objekten verhindert und somit einen Out-of-Memory-Error produzieren. Außerdem lassen sich sämtliche Umstände, die zur Problemsituation geführt haben, aufklären, wie beispielsweise die Parameter einer Anfrage oder SQL-Statements, die aus der Anwendung heraus abgesetzt wurden. Das Memory Analyzer Tool kann auf Grund einer umfangreichen Report-Sammlung, wie Leak-Suspection und Top-Consumers sowie der Möglichkeit, den Heap mittels Suchanfragen zu durchsuchen, die Analyse optimal unterstützen.

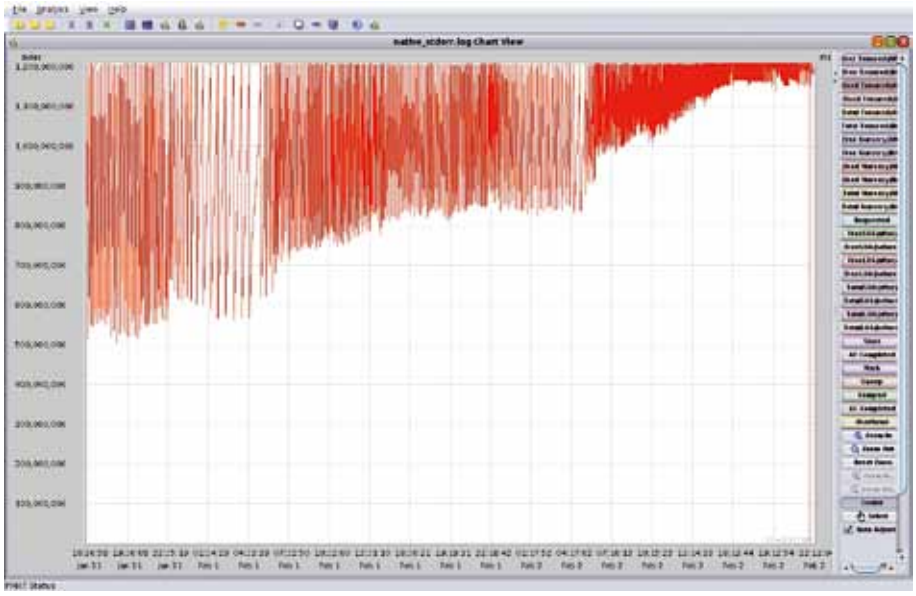


Abbildung 2: Das Pattern Modeling and Analysis Tool for Java Garbage Collection zeigt deutlich ein Memory Leak auf

Ein weiteres bekanntes Fehlerbild ist ein nicht reagierender Java-Thread. Dieses Problem äußert sich besonders bei JEE-Applikationen in der Weise, dass auf eine bestimmte Anfrage keine Antwort geliefert wird und das System nicht mehr zu reagieren scheint. In diesem Fall kann mithilfe eines Thread-Dump, auch Javacore-Dump genannt, untersucht werden, in welchem Programmteil das System hängen geblieben ist und ob es beispielsweise auf die Antwort eines Back-End-Systems wartet oder ob es sich in einem Ressourcen-Lock verfangen hat.

Der Thread and Monitor Dump Analyzer for Java aus der IBM Support Assistant Workbench kann Thread-Dumps aufbereiten und aufzeigen, in welcher Programmzeile sich jeder Thread befindet und ob Threads wegen synchronisierter Statements blockiert werden. Auch können mehrere Thread-Dumps miteinander verglichen werden, um beispielsweise Endlosschleifen über einen längeren Zeitraum zu erkennen.

In den allermeisten Fällen klärt sich das Problem schon bei der Analyse der Thread-Dumps. Jedoch reicht in einigen Fällen die einfache Information über die Thread-Zustände nicht aus, um das Problem zu lösen, besonders, wenn es nur in bestimmten Konstellationen auftritt, bei denen die Zustände bestimmter Objekte entscheidend sind. In diesem Fall kann wiederum ein System-Dump die notwendigen Informationen über Objekte und Variableninhalte sowie Thread-Stacks liefern.

Eine beispielhafte Fehlersituation könnte so aussehen: Nach einem Out-of-Memory-Error eines Application Servers analysiert der Experte zunächst die Garbage-Collection Ausgaben und stellt ein Memory-Leak fest (Abbildung 2).

Bei einem Memory-Leak erhöht sich der Speicherverbrauch einer Anwendung stetig. Eine Anpassung der Heap-Parameter könnte das Problem des Out-of-Memory-Errors an dieser Stelle nur hinauszögern. Um herauszufinden, welche Objekte den Heap stetig füllen, ist ein Heap- oder System-Dump notwendig. Die Standardkonfiguration der IBM JVM erzeugt einen Heap-Dump automatisch, sobald ein Out-of-Memory-Error auftritt.

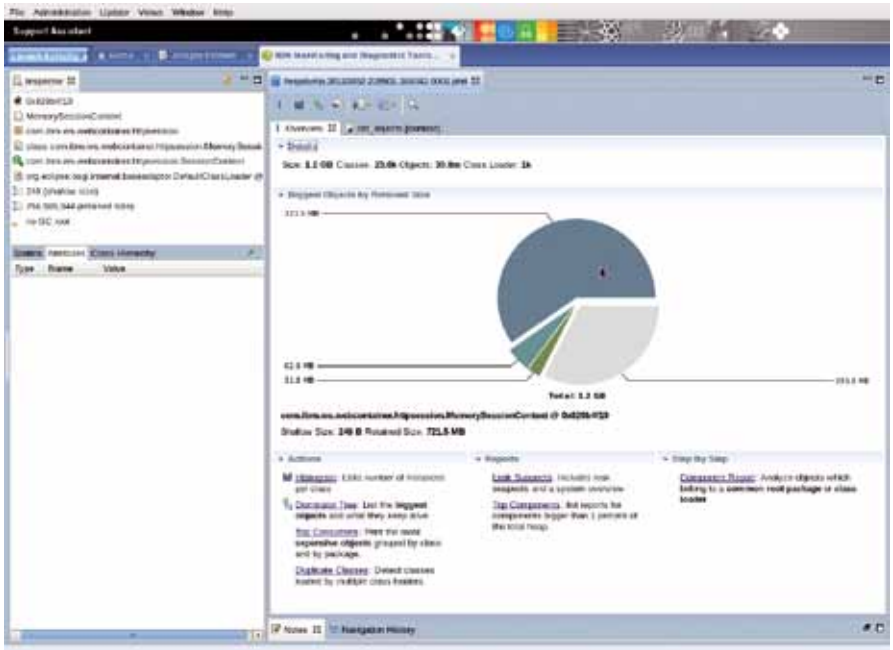


Abbildung 3: Das Memory Analyzer Tool zeigt in Diagrammform auf, welche Objekte für den größten Speicherverbrauch verantwortlich sind

Aus der Analyse des Beispiel-Dump geht hervor, dass der Server etwa 720 MB Session-Daten im Heap hält (Abbildung 3). Dabei stellt sich ebenfalls heraus, dass vereinzelte Sessions mehr als 20 MB groß sind und laut Konfiguration bis zu 1.000 Sessions im Speicher gehalten werden dürfen. In Kombination mit einem langen Session-Timeout ginge dem Server früher oder später der Speicher aus. Um das Problem dauerhaft zu lösen, sollten die Session-Größen innerhalb der Anwendung überdacht und verringert werden. Kurzfristig kann man durch das Anpassen der Session-Timeouts und der maximalen Anzahl der im Heap gehaltenen Sessions Erfolge erzielen.

Wie das Beispiel verdeutlicht kann auf Probleme mit den richtigen Daten und Werkzeugen schnell und nachhaltig reagiert werden. Langfristig werden sich so die Produktqualität sukzessive verbessern und die Ausfallzeiten von Anwendungen reduzieren – ein Vorteil für Anwendungsentwicklung und Betrieb.

Marc Bauer
marc.bauer@de.ibm.com

Marc Bauer arbeitet im Software Service der IBM Deutschland GmbH und berät Kunden bei Problem-Analysen und Performance-Optimierungen von WebSphere und Java-Applikationen im Betrieb und während der Entwicklung.

Vorschau

Java aktuell – Winter 2011

Die nächste Ausgabe erscheint am 1. Dezember 2011.

Falls Sie einen Artikel in der nächsten Java aktuell veröffentlichen möchten, schicken Sie bitte vorab Ihren Themenvorschlag an redaktion@ijug.eu, Redaktionsschluss ist am 12. Oktober 2011.

Impressum

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

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

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

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

Titel, Gestaltung und Satz:
Claudia Wagner, Katja Borgis
DOAG Dienstleistungen GmbH
Titelbild: Fotolia

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

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

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

Java aktuell
Magazin der Java-Community

Java EE 7 – eine Reise in die Wolken

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

Am 15. März 2011 wurde der Java Specification Request JSR-342, in dem die Inhalte der Java EE 7 bestimmt werden, einstimmig von der Java SE/EE Expert Group angenommen. Das war auch der Startschuss für die Arbeiten an der nächsten Version der Java-EE-Plattform.

Während Umfang und genaue Inhalte der Spezifikation noch diskutiert und geklärt werden, steht bereits die anvisierte Roadmap fest: eine Vorabversion noch in diesem Jahr und die Fertigstellung im nächsten. Der vorliegende Artikel beleuchtet deren Schwerpunkte.

Ein Platz unter den Wolken

Hauptthema von Java EE 7 ist die verbesserte Unterstützung für Cloud-Umgebungen mit dem Ziel, die Java-EE-Plattform nach dem PaaS-Modell (Platform-as-a-Service) auszurichten und für die Clouds salonfähig zu machen. Das würde dazu führen, dass Anwendungen, die für einen Kunden auf Java EE 7 PaaS installiert sind, neben solchen für andere Kunden laufen würden und von den typischen Vorteilen des Cloud-Computing wie wirtschaftlichere Ressourcen-Nutzung, Elastizität, garantierte Quality of Service (QoS) etc. profitieren können. Cloud-Anbieter wären in der Lage, eine standardisierte Plattform effizienter und kostengünstiger zu betreiben.

Die Java EE eignet sich schon heute für den Einsatz in öffentlichen und privaten Clouds und baut auf dieselben Abstraktionen wie Services und Ressourcen. Das zugrunde liegende Container-Modell ermöglicht schon die Bereitstellung, Injektion und Nutzung von Services und Ressourcen sowie das Deployment von Anwendungen in Clustern, die – abhängig vom Application Server Hersteller – mit der Last wachsen und schrumpfen können. Deshalb werden sich das Java-EE-Programmiermodell und das API bei der Umgestaltung der Plattform in Richtung PaaS nicht radikal ändern. Was beachtet werden muss, sind die neuen Herausforderungen, die sich generell für den Betrieb von Anwendungen auf einer PaaS ergeben. Das heißt, dass die Java EE als PaaS mandantenfähige Anwendungen unterstützen muss, die besser von einander getrennt sind, damit sie keine Auswirkungen auf andere parallel laufende Applikationen haben und nur die Daten ihrer Mandanten sehen und manipulieren können. Für alle Ressource-Manager-APIs wie JPA, JDBC, JMS oder JCA bedeutet dies, dass sie aufzuarbeiten sind, um gleichzeitig und sicher von mehreren Anwendungen und Mandanten benutzt werden zu können. Es wäre dann die Aufgabe vom Container, einzelne Anwendungs-Requests auf Mandanten abzubilden und ihre Identität über verschiedene Ressource-Manager und Tiers zu propagieren. Abbildung 1 stellt die angestrebte High-Level-Architektur von Java EE 7 als kontrollierte Cloud-Plattform PaaS dar.

Basis bildet die neue Virtualisierungsschicht, die beim Deployment, Betrieb, Management und Monitoring der Laufzeitumgebung zum Einsatz kommt. Sie abstrahiert die Schnittstellen zu Virtualisierungsprovidern wie Xen, KVM, VirtualBox und öffentlichen Cloud-Providern wie Amazon EC2 oder Microsoft Azure. Außerdem standardisiert sie den Umgang mit Application Server Clustern und passt ihn an die Cloud-Umgebung an.

In der darüberliegenden Schicht werden Zustand und Auslastung der Anwendungen und der Application-Server-Instanzen überwacht und verwaltet. Hier sind bei Bedarf Virtual Machines (VM) instanziiert, provisioniert und dem Cluster hinzugefügt sowie Anwendungszustände repliziert.

Die Funktionalität der Java EE Plattform manifestiert sich in der Services-Schicht. Zu ihr gehören Dienste wie JavaEE Container (für EJB, Servlets, CDI), Message-Queue- und Persistence-Services. Sie werden mehr abgekoppelt und können gegebenenfalls sogar in einer eigenen VM laufen.

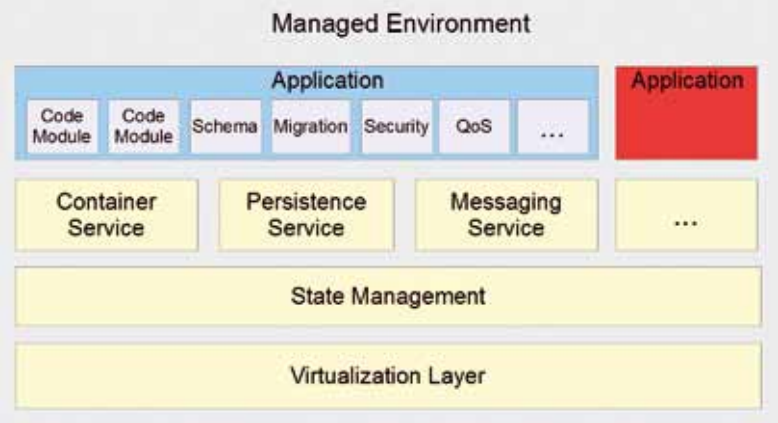


Abbildung 1: Java EE 7: PaaS-Architektur

Die oberste Schicht bilden die Anwendungen. Der JSR-342 sieht vor, dass diese Metadaten einen erweiterbaren Metadata-Deskriptor mit sich führen. Darin bestimmen sie explizit, ob sie für eine PaaS-Umgebung geeignet sind, und wenn ja, welche Anforderungen sie an die PaaS haben; ob sie zum Beispiel mandantenfähig sind, Elastizitätsanforderungen wie minimale/maximale Anzahl von Instanzen, Sicherheits- und QoS-Informationen, Abhängigkeiten von anderen Anwendungen, Ressourcen und Services erfüllen und ob beziehungsweise welche Dienste sie selbst anbieten. Die Plattform wird die Versionierung von Anwendungen und die Koexistenz mehrerer Versionen der gleichen Anwendung unterstützen. Die rot gefärbte Anwendung verdeutlicht, dass es durchaus Applikationen mit besonders hohen Sicherheitsansprüchen geben kann, die keine Dienste und Ressourcen mit anderen teilen und komplett abgeschottet sein wollen.

Während sich die Entwicklersicht auf APIs und Plattform nur wenig ändert, muss im Umfeld von Packaging und Deployment deutlich mehr getan werden. Ein schon diskutiertes Szenario sieht vor, dass der Java-EE-Plattform-Provider zum Zeitpunkt der Installation einer Anwendung zunächst ihre Metadaten ausliest. Entsprechend den Elastizitätsanforderungen wird die minimale Anzahl von Virtual Machines von parametrisierten Templates instanziiert (und somit schon provisioniert) und darauf ein Application-Server-Cluster aufgesetzt, der die geforderten Dienste und Ressourcen bereitstellt. Die Anwendung ist dann im Cluster installiert.

Abhängig von der Auslastung werden dynamisch weitere VM gestartet und dem Cluster hinzugefügt oder von ihm herausgenommen und entsorgt. Die Konfiguration von serverseitigen Ressourcen wie Connection Pools, Queues, Topics etc., die zur Zeit sehr stark vom jeweiligen Application Server abhängt, wird entfallen. Sie sind automatisch beim Deployment entsprechend den Anwendungsanforderungen im Cluster provisioniert. Der Java-EE-Plattform-Provider wird eine standardisierte Management-Schnittstelle anbieten, die einem aufgebohrten Cluster-Administrationsserver ähnelt. Neue Rollen wie PaaS-Administrator, PaaS-Provider und PaaS-Kunde (Mandant) werden etabliert.

Modularität

Der Bedarf an Modularität für Java-EE-Anwendungen war schon immer groß, zumal sie oft mit ihrem Footprint denjenigen vom zugrundeliegenden Application Server übertreffen. Das Fehlen eines nativen Modulsystems in Java SE hat dazu geführt, dass viele Application-Server Hersteller eigene Modulsysteme aufgebaut haben, mehrere davon auf OSGi-Basis und mit meistens inkompatiblen Classloader-Hierarchien. Da Java EE 7 auf Java SE 7 aufsetzt und die Modularisierung der Java Plattform auf Java SE 8 verschoben wurde, wird es leider keine echte standardisierte Modularisierung für Java EE vor Java EE 8 geben. Plattform-Anbieter, die in der Architektur von ihrem Application Server eine Abstraktionsschicht über dem eigenen Modulsystem vorgesehen haben (wie das HK2-Modulsystem über OSGi in GlassFish) werden künftig vermutlich leichter ihre Produkte auf die Modularität von Java SE 8 umstellen können. Auch wenn Java EE 7 noch keine Modularisierung anbieten wird, so ist zumindest angestrebt, sich darauf vorzubereiten, etwa durch die Verwendung von Metadaten, die Abhängigkeiten und Versionen explizit beschreiben, sowie durch Modifikationen der Classloader und der Art ihrer Verwendung. Ziel ist, schon mit Java EE 7 typische Anwendungsfälle der Nutzung von Modularität zu unterstützen. So ein Fall wäre der Support für Anwendungen, die aus versionierten Modulen bestehen. Ein anderer wäre folgender: Falls eine Anwendung eine Bibliothek gebündelt mit sich führt, weil sie von ihr abhängt und nicht weiß, ob sie auf dem Application Server vorliegt, und falls der Application Server genau dieselbe Version dieser Bibliothek implementiert, so soll die Modulauflösung die Implementierung vom Application Server vorziehen. Man geht dabei davon aus, dass die Bibliothek-Implementierung vom Application Server durch seinen Hersteller besser optimiert ist und besser in seiner Infrastruktur integriert wurde.

Unterstützung der neuesten Web-Standards

Um mit den neuen technologischen Entwicklungen Schritt zu halten und die Wettbewerbsfähigkeit der Plattform zu erhöhen, wird die Unterstützung für eine Reihe von neu entstehenden Web-Standards wie WebSockets und HTML5 angestrebt. WebSockets sind bereits in Grizzly – die Netzwerk-Protokoll-Engine von GlassFish – implementiert. Diese Implementierung ist zwar nicht portabel, könnte aber als Ausgangspunkt für eine Standardisierung in Java EE 7 zum Einsatz kommen.

Offensichtlich ist JSF die natürliche Heimat für den HTML5-Support. Im März 2011 wurde der JSR-344 für JSF 2.2 gestartet, in dem HTML5-Funktionalitäten wie HTML5-Forms und einige der Inhaltsmodelle der HTML5-Elemente unterstützt werden sollen. Mit wachsender Verbreitung von NoSQL-Architekturen im Web soll auch das Thema der Abbildung von Java-Modellen auf nicht-relationalen Datenbanken untersucht werden. Eine Erweiterung des JPA-API erscheint aufgrund seiner starken Abhängigkeit von relationalen Datenbanken nicht als sinnvoll, sodass vermutlich ein neues API zur Anbindung von NoSQL Backend-Systemen notwendig sein wird.

Vorläufiger Inhalt

Wenn man den beabsichtigten Umfang von Java EE 7 nach aktuellem JSR-Status kategorisiert, ergibt sich Tabelle 1.

Manche der APIs haben im Fokus, die Entwicklung zu vereinfachen. Dazu zählt das JMS-2.0-API, das komplett modernisiert wird. Geplant ist die Einführung von Annotations, gegebenenfalls den Übergang zu „Connectionless API“ (um das übliche Bootstrapping ConnectionFactory → Connection → Session → … zu eliminieren) und die Standardisierung der Integration mit Application Server und verbreiteter JMS-Hersteller-Erweiterungen. JAX-RS, das serverseitige API für REST-basierte Architekturen, wird in seiner neuen Version 2.0 zwei Client-APIs definieren, ein Low-Level-API unter Verwendung vom Factory-Pattern und ein High-Level-API, das auf das erste aufsetzt. Weiterhin wird JAX-RS-2.0 eine MVC-Architektur spezifizieren, die mit dem JAX-RS-Programmiermodel kompatibel ist und mit verschiedenen Viewing-Technologien wie JSP, FreeMaker oder StringTemplate zusammenarbeiten kann. Eine gute Ergänzung für JAX-RS wird das neue in Betracht gezogene JSON-API. Weiterhin in Richtung Vereinfachung der

Beantragt und abgestimmt	Noch zu beantragen	Andere (wieder aufgenommen, kleine Änderungen)
JPA 2.1 – JSR 338 JAX-RS 2.0 – JSR 339 Servlets 3.1 – JSR 340 EL 3.0 – JSR 341 Java EE 7 – JSR 342 JMS 2.0 – JSR 343 JSF 2.2 – JSR 344 EJB 3.2 – JSR 345 CDI 1.1 – JSR 346 Data Grids – JSR 347	DI 1.1 JSON 1.0 Bean Validation 1.1	JCache – JSR 107 Concurrency Utilities – JSR 236 Common Annotations 1.2 JAX-WS 2.3 JTA 1.2 JSP 2.3 Connector 1.7

Tabelle 1: Vorläufiger Inhalt von Java EE 7

Entwicklung ist die Überlegung, wie das Managed-Bean-Modell verfeinert und erweitert werden kann, um die Überlappungen und Inkonsistenzen unter Managed Beans, EJB, Servlets, JSF, CDI und JAX-RS zu eliminieren.

Andere APIs werden die Funktionalität der Plattform deutlich erweitern. JCache zum Beispiel standardisiert das Caching von Java-Objekten, auf die von mehreren Threads im laufenden Prozess zugegriffen wird. Das Caching kann erheblich die Performance und Skalierbarkeit von Web -Anwendungen erhöhen. Es wird gerade diskutiert, wie die Funktionalität des JCache-API durch das neu angenommene Data-Grids-API auf verteilte Caches mit Unterstützung für Transaktionen und asynchrone Zugriffe ausgedehnt werden kann. Das würde Application-Server-Cluster und ihre Elastizitätsfähigkeit mit einem separaten hochverfügbaren Cache-Tier ergänzen. Das Concurrency Utilities for Java EE API wird auf dem „java.util.concurrent"-API von Java SE basieren und Nebenläufigkeit auf der Anwendungsebene in der kontrollierten Umgebung von Java EE ermöglichen.

Transparenz

In den letzten Monaten arbeitet Oracle mit Hochdruck daran, die Transparenz bei der Erstellung der einzelnen JSRs und vom Java Community Process (JCP) selbst zu erhöhen. Das zeigt sich zuletzt in der Beantragung und einstimmigen Annahme vom JSR-348. Er hat zum Ziel, noch in diesem Herbst die Regeln aufzustellen, nach denen der JCP in Richtung erhöhte Transparenz und effizienteres Management modifiziert wird. Vor diesem Hintergrund werden im Punkt 2.15 vom JSR-346 für die Java EE 7 die Maßnahmen aufgezählt, die zur Verbesserung der Kommunikation und der öffentlichen Verfolgung bei der Entwicklung der Spezifikation beitragen. Für die gesamte und jede Teilspezifikation soll es ein separates Projekt auf java.net geben, mit Links auf die öffentlich einsehbaren Mailing-Lists, das JIRA Problem-Verfolgungswerkzeug, Source-Code Repository und gegebenenfalls Wiki. Für Java EE 7 ist die Projektseite unter http://java.net/projects/javaee-spec, für JAX-RS 2.0 API, als Beispiel einer Teilspezifikation, unter http://java.net/projects/jax-rs-spec zu erreichen. Für den Zugriff benötigt man einen „java.net"-Benutzer. Die Registrierung ist kostenlos.

Ausblick

Für Java EE 7 hat die Reise in die Wolken begonnen. Sie wird sicherlich spannend, aber nicht einfach sein. Auf der einen Seite sind die Inhalte umfangreich und anspruchsvoll, die Wunschliste geforderter Funktionalitäten ist lang. Auf der anderen Seite gibt es viele Produkte auf dem Markt, die als Vorbild und Anregung bei der Standardisierung dienen können. So kann TopLink als Beispiel dienen, wie die Unterstützung von Mandantenfähigkeit bei relationalen Datenbank-Ressourcen (Erweiterungen für mandantenspezfische Entity-Customization, @Multitenant-Annotation) und die Anbindung an Data Grids und NoSQL-DBMS (Interceptoren zur Anbindung an Coherence oder NoSQL-Ressourcen) implementiert werden kann. Da die Fertigstellung der Spezifikation auf Ende nächsten Jahres terminiert wurde, ist es wahrscheinlich, dass sich einige angestrebte Features aus Zeitgründen auf die nächste Java-EE-8-Version verschieben.

Es ist noch möglich, sich an der Entwicklung der Java-EE-7-Plattform oder der einzelnen APIs federführend zu beteiligen. Voraussetzung ist die Registrierung als JCP-Member und die Unterzeichnung eines Java Specification Participation Agreement (JSPA). Unter http://jcp.org/en/participation/membership ist der Prozess genau beschrieben. Anregungen und Feedback sind auf den Mailing-Listen der Java-EE-7-Projektseite willkommen.

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 Architekt und Consultant bei Sun Microsystems gesammelt.

Varianten-Entwicklung in 3D mit Object Teams

Dr. Stephan Herrmann, GK Software AG

Das Entwickeln modularer Software ist nur oberflächlich betrachtet so einfach wie das Zusammenstecken von Lego-Bausteinen. Wer beispielsweise eine variantenreiche Produktlinie von Software-Systemen aus wiederverwendbaren Modulen erzeugen will, muss zwei Dinge beherrschen: Komplexität und Veränderungen über die Zeit. Längst sind die Konzepte von Klassen und Objekten nicht mehr als alleiniges Mittel ausreichend, um insbesondere die entstehenden Zielkonflikte zwischen enger Integration und loser Kopplung aufzulösen. Hier verspricht der Object-Teams-Ansatz mit nur wenigen neuen Konzepten eine radikale Verbesserung der modularen und nachhaltigen Entwicklung.

Als am 22. Juni 2011 das Indigo-Release von Eclipse (3.7) veröffentlicht wurde, war unter den 62 beteiligten Projekten erstmalig das Eclipse Object-Teams-Projekt mit von der Partie. Obwohl noch ganz neu im Simultaneous Release von Eclipse, blickt dieses Projekt doch auf eine fast zehnjährige Geschichte zurück. Ende 2001 begannen an der TU Berlin zwei Diplomanden, Compiler und Laufzeitumgebung für eine Erweiterung der Sprache Java zu konstruieren, die der Autor damals ObjectTeams/Java nannte (heute: OT/J). Hier flossen diverse aktuelle Forschungsergebnisse zusammen, die sich alle damit befassten, wie objektorientierte Programmiersprachen die modulare Entwicklung besser unterstützen können. Später wurde in einem dreijährigen, vom BMBF geförderten Forschungsprojekt unter anderem eine umfassende, Eclipse-basierte Entwicklungsumgebung entwickelt, das Object Teams Development Tooling (OTDT). Einige Zeit nach Ablauf der Förderung zog das Projekt aus dem universitären Kontext zu Eclipse um, aus „org.objectteams" wurde „org.eclipse.objectteams" [1]. Rechtzeitig vor dem Indigo-Release graduierte das Eclipse Object-Teams-Projekt, womit dem Projekt bescheinigt wurde, dass es nun ein reifes Mitglied der Eclipse-Community ist. Die aktuelle Versionsnummer 2.0.0 deutet an, dass hier ein Werkzeug vorliegt, dass schon lange den Kinderschuhen entwachsen ist, und mit viel Liebe zum Detail die alltägliche Arbeit mit OT/J unterstützt.

Formbare Software

Die Beschränkung der Legometapher und aller Techniken, die sich zu eng an diese Metapher klammern, wird überall dort deutlich, wo das „soft" in Software essenzielle Bedeutung hat. Der Object-Teams-Ansatz verspricht eine Verbesserung der modularen Entwicklung für sehr viele, sehr unterschiedliche Szenarien – darunter viele Situationen, in denen der normale Java-Entwickler überzeugt ist, dass seine Lösung, die viele Entwurfsmuster, Frameworks und was nicht alles auffährt, die bestmögliche Lösung sei. Wenn Verständlichkeit, Wartbarkeit etc. leiden, wenn die Komplexität durch die Lösung noch steigt, dann gibt sich manch einer damit zufrieden, „dass es eben nicht besser geht". Aber: Es geht besser, das Ende der Fahnenstange ist mit Java noch lange nicht erreicht, selbst wenn man zum Beispiel OSGi oder Dependency Injection hinzunimmt – Weiterklettern ist erlaubt.

Stellvertretend für die Fülle konfliktreicher Situationen, in denen Object Teams zu helfen verspricht, stelle man sich das folgende Szenario vor: Eine komplexe Applikation, die bereits vielfach im Einsatz ist, wird beständig weiterentwickelt. Eine mögliche Erweiterung stellt sich dabei als so umfangreich heraus, dass sie nicht innerhalb eines sechswöchigen Entwicklungszyklus erstellt werden kann. Genau genommen gibt es auch noch eine Reihe von offenen Fragen, sodass für dieses Teilprojekt noch gar kein langfristig verbindlicher Plan aufgestellt werden kann. Dennoch soll die Arbeit sofort beginnen und sobald wie möglich sollte das neue Feature an einige Testkunden ausgeliefert werden, um früh Feedback zu sammeln. Anfangs ist noch von einem Prototypen die Rede, sehr bald will man allerdings das neue Feature als Zusatzmodul installieren können. Sollte ein Kunde jedoch damit unzufrieden sein, soll er das Feature auf Knopfdruck auch wieder deaktivieren können. Ob beziehungsweise wann es in die Kernapplikation übernommen wird, ist noch ungewiss.

Das Projektteam konstatiert gleich zu Beginn seiner Arbeit, dass die Entwicklung der Erweiterung grundlegende Änderungen am Applikationskern erfordern würde, jedoch verweigert das Entwicklungsteam dieses Kerns die notwendigen Änderungen mit dem Verweis auf mögliche negative Auswirkungen auf stabile Teile der Applikation. Der Vorschlag, den gesamten Code der Applikation zu branchen, kommt bei den Entwicklern nicht gut an. Als sogar davon die Rede ist, dass dieser Branch auf unbestimmte Zeit bestehen bleibt, während im Hauptstrang munter weiterentwickelt wird, bekommen die Entwickler schlechte Laune. Als die schon überlasteten Build- und Deployment-Experten erfahren, dass sie ab sofort dafür verantwortlich sind, das Feature aus dem Branch zu deployen, und zwar so, dass der Kunde es jederzeit hinzufügen beziehungsweise auch wieder deaktivieren kann, springen sie im Dreieck. Keiner in

der Firma hat Lust, dauerhaft mit Branches und Patches zu arbeiten.

Tatsächlich entspricht diese Schilderung in weiten Zügen einer Situation aus dem Alltag des Autors als Eclipse-Committer [2]. Das konkrete Beispiel zeigt, dass durch die Verwendung von OT/J nicht nur die Probleme von Branching und Patches komplett vermieden werden können, sondern auch, dass der Code der Erweiterung in der OT/J-basierten Implementierung viel übersichtlicher, besser verständlich ist, wodurch die Weiterentwicklung viel einfacher wird.

Spezialisierung ohne Grenzen

Wenn von einer existierenden Applikation eine Variante erzeugt werden soll, so besteht dies aus zwei Teilaufgaben: neue Klassen und Objekte hinzufügen (einfach) sowie existierendes Verhalten spezialisieren. Objektorientierte Entwickler kennen sich mit Spezialisierungen aus und haben genau dafür das Thema „Vererbung" in ihrem Werkzeugkasten. Allerdings kennen sie auch die Grenzen der Vererbung nur zu genau und wissen, dass Vererbung für dieses Variantenproblem keine Lösung bietet. Trotzdem betreiben sie von der Sache her Spezialisierung, nur eben nicht mittels Vererbung sondern beispielsweise durch Branching. Hier führt Object Teams eine Spezialisierungsrelation ein, die bei aller Ähnlichkeit zu Vererbung deren Beschränkungen aufhebt. Ein Beispiel:

```
public class Employee playedBy Person { /*
details */ }
```

Dies deklariert, dass Employee eine „Rolle" ist, die von Personen gespielt wird. In drei Punkten ähnelt diese Rollenbindung der Vererbung:

1. Die Typen „Employee" und „Person" § sind nach bestimmten Regeln austauschbar
2. Die Methoden der Klasse „Person" stehen auch an Objekten der Klasse „Employee" zur Verfügung
3. Die Klasse „Employee" kann Methoden der Klasse „Person" überschreiben

Im Gegensatz zu Vererbung erfordern die Punkte 2 und 3 bei OT/J eine explizite Methodenbindung (siehe Listing 1).

```
public class Employee playedBy Person {
        String officePhone;
        String getName() -> String getName();
        getOfficePhone <- replace getPhoneNumber;
        callin String getOfficePhone() {
        return officePhone; }
}
```

Listing 1

Zeile 3 ist eine „callout"-Methodenbindung, die angibt, dass „getName()"-Aufrufe an die zugrundeliegende Person weitergeleitet werden; somit steht diese Methode der Klasse „Person" auch für „Employees" zur Verfügung. Zeile 4 ist eine „callin"-Methodenbindung, die angibt, dass „getPhoneNumber()"-Aufrufe statt der ursprünglichen Implementierung nun die Methode „getOfficePhone()" benutzen sollen; somit gilt im Kontext „Employee" eine andere Implementierung für diese Methode.

Ein erster Unterschied zur Vererbung liegt darin, dass Methodenbindungen explizit deklariert werden, wodurch beide Klassen deutlich loser gekoppelt sind, denn was nicht explizit gebunden wird bleibt auch unabhängig – keine Namenskonflikte, kein unabsichtliches Überschreiben etc.

Mit diesen drei Konstrukten, „playedBy", „callout" und „callin" sind bereits die wichtigsten Syntax-Elemente von OT/J vorgestellt, damit lässt sich also Spezialisierung nach Rollenart implementieren. Aber was ist damit gewonnen, inwiefern hebt dies die Beschränkungen von Vererbung auf?

Flexibel, flexibel und nochmals flexibel

Im Vergleich zur Vererbung ist eine Rollenbindung in dreifacher Weise flexibler:

- Rollen können dynamisch zu existierenden Laufzeitobjekten (Instanzen) hinzugefügt (und wieder entfernt) werden. Diese Flexibilität über die Zeit basiert darauf, dass Rollenbindung in der Tat eine Beziehung zwischen einem Basisobjekt und einem Rollenobjekt ist.
- Mehrere Rollenobjekte können dasselbe Basisobjekt spezialisieren. So kann eine Person neben ihrer Employee-Rolle auch eine Mutter-Rolle spielen, aber sogar auch eine zweite Employee-Rolle (in einem zweiten Job bei einer anderen Firma). Diese Mehrfachspezialisierungen sind zunächst unabhängig, aber für diejenigen Situationen, in denen es potenziell zu Konflikten käme, definiert OT/J einfach anwendbare Vorfahrtsregeln.
- Spezialisierung mit Rollen kann im Nachhinein auf eine existierende Applikation aufgeflanscht werden, ohne dass eine einzige Zeile der Applikation geändert wird oder diese in irgendeiner Weise dafür vorbereitet sein muss. Während bei Vererbung jeder Aufruf von „new" den Typ eines Objektes vollständig und endgültig festlegt, werden Rollenobjekte nach bestimmten Regeln implizit erzeugt. Dadurch braucht OT/J weder Factories noch Dependency Injection, denn derlei Flexibilität kann ohne Vorausplanung an jedem beliebigen Punkt des Programms eingesetzt werden.

Rollen implementieren Varianten

Was hat nun Rollenbindung mit unserem Variantenproblem zu tun? Mit Rollen kann jede Erweiterung einer Applikation als eigenständiges Modul entwickelt werden. Überall, wo die Erweiterung normalerweise Änderungen am Kern erfordern würde, implementiert man stattdessen durch Rollen kontextspezifische Spezialisierungen. Damit wird jedes Branching überflüssig. Wenn Kern und Erweiterung parallel weiterentwickelt werden, entfällt jegliches Merging; stattdessen ist nur zu überprüfen, ob die expliziten Bindungen (playedBy, callout, callin) weiterhin gültig sind, und diese Überprüfung führt der OT/J Compiler unaufwendig und zuverlässig aus. Dies ist um Größenordnungen robuster und zuverlässiger als jeder textuelle Abgleich paralleler Entwicklungsstränge.

Aber es kommt noch besser: Branching/Merging sind nur auf Quelltext anwendbar, sprich diese Techniken bieten keinerlei Unterstützung für Build und Deployment. Mit den Rollen in Object Teams existiert aber ein Konzept, das durchgängig bis in die Laufzeit wirkt. Dadurch können Rollen ohne weitere Verrenkungen auch unabhängig gebaut und deployt werden. Der Deployment-Experte bekommt nur eine triviale

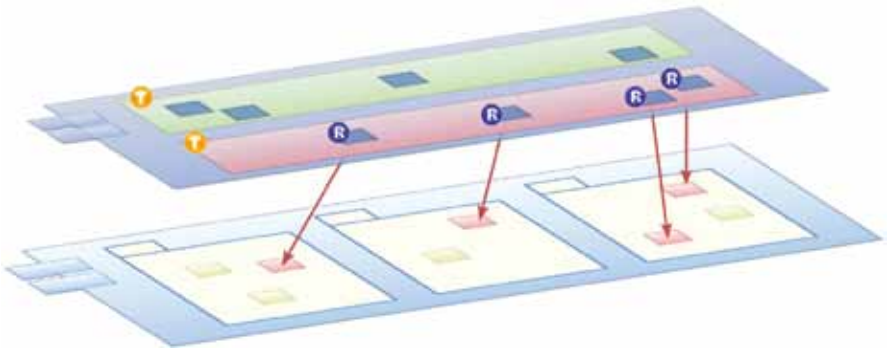


Abbildung 1: Teams als Ebenen im 3-dimensionalen Entwurfsraum

Zusatzaufgabe: beim Deployment der Variante mit der neuen Erweiterung zusätzlich die schlanke OT/J Runtime mit zu verteilen.

Von Rollen zu Team-Modulen

Nach der bisherigen Schilderung könnte man den Eindruck gewinnen, dass die Vorteile von Rollen dadurch zunichte werden, dass sich die Anzahl der Bausteine eher erhöht. Sind alle Rollen etwa explizit und einzeln zu deployen? Nein, in OT/J sind Rollen stets Mitglieder eines „Teams". Mit der gleichen Durchgängigkeit wie das Rollenkonzept fördern Teams die modulare Struktur in allen Stadien:

- Teamklassen bündeln eine Gruppe von Rollenklassen. Standardmäßig sind Rollen dabei noch stärker gegen Zugriffe von außen gekapselt als dies in normalem Java möglich wäre. Sogar eine konsistente Spezialisierung eines ganzen Teams mit all seinen Rollen wird ausdrücklich unterstützt.
- Je ein Teamobjekt verwaltet die On-demand-Erzeugung von Rollenobjekten und cacht diese für spätere Zugriffe. Diese Mechanismen werden durch die Sprache soweit gekapselt, dass ein Entwickler sich darum im Regelfall überhaupt nicht kümmern muss.
- Jedes Teamobjekt kann zur Laufzeit dynamisch aktiviert und deaktiviert werden. Damit wird gesteuert, welche Spezialisierungen in einem bestimmten Kontext wirksam sind. Ein Kontext kann hier sein: ein bestimmter Zeitabschnitt, ein Thread, oder auch die Menge aller Objekte, die ein gegebenes Prädikat erfüllen. Technisch wird Team-Aktivierung auf das Enablement von enthaltenen „callin"-Bindungen abgebildet, also: solange kein Teamobjekt aktiviert wurde, bleiben alle Überschreibungen mittels „callin" wirkungslos. Das Ausschalten eines Zusatzfeatures kann dadurch mit geringstem Aufwand zur Laufzeit angeboten werden, selbst ohne die Applikation neu zu starten.

Dreidimensionale Architektur

Mit normalem Java ist die Welt wie eine Scheibe: ohne den Einsatz zusätzlicher Infrastruktur (wie OSGi) besteht ein Programm aus einer unstrukturierten Menge von Klassen, wo jeder jeden sehen kann. Die Architektur kann Klassen in zwei Dimensionen anordnen: Assoziation und Vererbung. Mit nur einer Spezialisierungsdimension erfordern viele Erweiterungsaufgaben, dass Klassen in ganz unterschiedlichen Regionen der Architektur verändert werden müssen.

OT/J liefert eine dritte Dimension: die Rollenbindung bietet eine weitere Spezialisierungsdimension, Rollen stehen außerhalb der Kernarchitektur und können doch Einfluss auf deren Verhalten ausüben (siehe Abbildung 1). Teams bieten ein optimales Strukturierungsmittel, dass alle Rollen eines Features bündelt und kapselt, und zwar sowohl in der statischen Sicht des Compilers als auch in der dynamischen Laufzeitsicht. Alle Beziehungen zwischen den Rollen und dem Applikationskern sind explizit deklariert, auf Klassenebene durch „playedBy", sowie die beiden Aufruf-Richtungen „callout" (Rolle-zu-Basis) und „callin" (Basis-zu-Rolle).

In einer solchen dreidimensionalen Architektur bleibt jede einzelne Ebene deutlich schlanker als in herkömmlicher Architektur, da jede Ebene sich wirklich nur um ihre ureigenen Anliegen kümmern muss. Dadurch verbessert sich natürlich die Wartbarkeit jedes Moduls, aber auch die Komposition zu einem Gesamtsystem ist durch diese expliziten Bindungen besser wartbar. Und dies ist nicht nur eine akademische Aussage, sondern wird durch langjährige Erfahrung unterstützt: das OTDT selbst ist nämlich zu weiten Teilen in OT/J geschrieben, schließlich ist die IDE für OT/J eine Variante der Java IDE (JDT). Diese Architektur besteht seit rund fünf Jahren und hat bei der parallelen Weiterentwicklung von JDT und OTDT beste Dienste geleistet.

Fazit

Variantenentwicklung mit Branches und Patches ist nur ein Beispiel, wo OT/J neue Dimensionen der Modularität eröffnet und die Wartbarkeit ihrer Software verbessert.

[1] Object Teams Homepage http://www.eclipse.org/objectteams

[2] Stephan Herrmann: „Null annotations: prototyping without the pain" http://blog.objectteams.org/2011/03/null-annotations-prototyping-without-the-pain/

Dr. Stephan Herrmann
sherrmann@gk-software.com

Dr. Stephan Herrmann ist Softwarearchitekt bei der GK Software AG sowie Committer für die Eclipseprojekte JDT/Core und Object Teams. Vorher lehrte und forschte er von 1997 bis 2009 an der TU Berlin, wo er seinerzeit daran beteiligt war, objektorientierte Programmierung in der Lehre einzuführen. In seiner Dissertation hat er sich 2002 mit Prinzipien und Implementierung von modularen, repositorybasierten Entwicklungsumgebungen beschäftigt. Seitdem liegt sein Schwerpunkt auf der Entwicklung der Java-Erweiterung OT/J und der zugehörigen Entwicklungsumgebung. Er war Redner auf zahlreichen akademischen sowie praxisbezogenen Workshops und Konferenzen im In- und Ausland.

Unbekannte Kostbarkeiten des SDK Heute: Der Service-Loader

Bernd Müller, Ostfalia

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

Das Abstract Factory Pattern der Gang of Four wird vielfach verwendet. Allein im SDK (1.6.0_26) gibt es 551 Dateien, die als Namensbestandteil „Factory" enthalten. Die meisten von ihnen dürften das Abstract „Factory Pattern" realisieren. Bei der XML-Verarbeitung können beispielhaft SAX- und DOM-Parser sowie ein XSL-Transformator über das Pattern erzeugt werden (siehe Listing 1)

Da die Implementierungen das JAXP-API realisieren, ist ein Austausch der Implementierung einfach möglich und erfolgt durch die Verwendung eines Properties, beim SAX-Parser etwa:

```
java -Djavax.xml.parsers.
SAXParserFactory=<Fabrik> Class
```

```
SAXParser parser =
    SAXParserFactory.newInstance().newSAXParser();
DocumentBuilder builder =
    DocumentBuilderFactory.newInstance().
    newDocumentBuilder();
Transformer transformer =
    SAXTransformerFactory.newInstance().newTransformer();
```

Listing 1

Als Implementierung des Pattern muss die entsprechende abstrakte Fabrik realisiert werden; ein zwar geringer, aber doch zu leistender Aufwand. Seit Java 6 gibt es mit dem Service-Loader eine einfachere Möglichkeit, eine oder mehrere Implementierungen einer Schnittstelle zur Verfügung zu stellen.

Der Service-Loader

Die Klasse „java.util.ServiceLoader" stellt eine sehr einfache Schnittstelle zur Verwendung von Diensten dar. Ein Dienst ist hierbei als Implementierung eines (oder mehrerer) Interfaces zu verstehen. Eine Implementierung wird über eine textuelle Deklaration publiziert und durch den „ServiceLoader" zugegriffen. Als Beispiel wählen wir die Fakultätsberechnung, um den Code-Umfang klein halten zu können. Der Leser wird sich aus seinem Projektalltag leicht einen komplexeren Dienst vorstellen können. Der Dienst zur Fakultätsberechnung ist als Interface „Factorial" definiert:

```
package de.pdbm;
public interface Factorial {
    long factorial(long n);
}
```

Eine mögliche Implementierung erfolgt iterativ (siehe Listing 2).

Eine weitere Implementierung erfolgt rekursiv (siehe Listing 3)

Die Veröffentlichung des Dienstes erfolgt als textueller Eintrag in einer Datei, die sich im Verzeichnis „/META-INF/services" des JARs befindet und deren Namen dem Binärnamen des Interfaces entspricht. Für die iterative Version also die Datei „/META-INF/services/de.pdbm.Factorial" mit dem Inhalt „de.pdbm.IterativeFactorial". Es können auch mehrere Implementierungen (eine pro Zeile) angegeben werden. Die einzige Anforderung an eine Implementierung ist die Existenz des Default-Konstruktors.

Die Verwendung der Service-Provider erfolgt über ein sehr schlankes API. Über

```
package de.pdbm;
public class IterativeFactorial implements Factorial {

    public IterativeFactorial() { ... }

    @Override
    public long factorial(long n) {
        long fakultaet = 1;
        for (long i = 1; i <= n; i++) {
            fakultaet *= i;
        }
        return fakultaet;
    }
}
```

Listing 2

```
package de.pdbm;
public class RecursiveFactorial implements Factorial {

    public RecursiveFactorial() { ... }

    @Override
    public long factorial(long n) {
        if (n <= 0) {
            return 1;
        } else {
            return n * factorial(n -1);
        }
    }
}
```

Listing 3

die Methode „load(Class service)“ werden alle Klassen gesucht, die das Service-Interface implementieren und entsprechend der oben genannten Konvention veröffentlicht wurden:

```
ServiceLoader<Factorial> loader =
    ServiceLoader.load(Factorial.class);
```

Da „ServiceLoader“ das „Iterable“-Interface implementiert, kann einfach über alle Provider iteriert werden:

```
for (Factorial provider : loader) {
    // verwende provider
}
```

oder anderweitig mit einem Provider weitergearbeitet werden. Da Java sehr dynamisch arbeitet, ist auch das Nachladen von Services zur Laufzeit kein Problem.

Fazit

Mit dem „ServiceLoader“ hat Java 6 ein neues API erhalten, das es mit sehr einfachen Mitteln ermöglicht, neue Dienste zur Verfügung zu stellen. Über die einfache Verwendung in unserem Beispiel hinausgehend hat das SDK damit ein einfaches Service Provider Interface (SPI) bekommen, da das Interface frei definierbar ist und die Implementierungen keinen Einschränkungen unterliegen. Wir bewegen uns damit mit unseren Fachanwendungen auf derselben SPI-Ebene wie etwa JNDI (javax.naming.spi) in Java-SE oder CDI (javax.enterprise.context.spi) in Java-EE.

Bernd Müller
bernd.mueller@ostfalia.de

Bernd Müller ist seit März 2005 Professor für Software-Technik an der Fakultät Informatik der Hochschule Braunschweig/Wolfenbüttel, nachdem er sieben Jahre Professor für Wirtschaftsinformatik an der Hochschule Harz war. Praktische Erfahrung hat er zuvor im Wissenschaftlichen Zentrum der IBM in Heidelberg sowie bei HDI Informationssysteme in Hannover gesammelt.

Rich Client Frontends für umfangreiche Unternehmensanwendungen

Björn Müller, CaptainCasa

Der Artikel zeigt, wie eine Community mittelständischer, vorwiegend deutscher Softwarehäuser die Frontends für ihre Anwendungen baut: über ein Framework, in dem vorne im Client eine Java-Swing-basierte Benutzeroberfläche läuft und die Anwendung selbst hinten im Server innerhalb einer JSF-Umgebung abgebunden wird. Das Community-Framework nennt sich „CaptainCasa Enterprise Client", wird seit 3 Jahren mannigfaltig verwendet und steht im vollen Umfang kostenfrei zur Verfügung.

Swing im Client? Ist das nicht komplett „out of Hype"? Im Server JSF? Ist das nicht „höllisch komplex"? Und ist JSF nicht ein Framework für HTML-basierte Frontends? Alles berechtigte Fragen. Es gehört schon einiges an Mut dazu, in der Hype-getriebenen Welt der UI-Technologien heute noch die Swing-Fahne zu hissen. Aber es gibt gute und vernünftige Gründe, dies auch und gerade heute zu tun! Und wenn man es schon macht, dann auch richtig, also mit einer vernünftigen, effizienten Architektur, die auf ihr Einsatzgebiet zugeschnitten ist.

Es geht nicht um „Apps"

In erster Linie reden wir hier über Benutzeroberflächen für umfangreiche Unternehmensanwendungen. Deren Bediener sind in der Regel Sachbearbeiter mit recht hohen Anforderungen an die Bedien-Performance und Bediener-Ergonomie (die berühmte Tastatursteuerung …). In der Regel gibt es Hunderte von unterschiedlichen Dialogen in einer solchen Anwendung – zur Erfassung von Steuer- und Stammdaten, zur operativen Nutzung des Systems, zur Analyse innerhalb des Reportings etc.
Ganz wichtig: Die Anwendungen haben einen Lebenszyklus, der gut und gerne 15 Jahre überschreitet: heute entwickelt, in zwei Jahren im Vertrieb, fünf Jahre vertrieben, acht Jahre Nutzungszeit beim Endkunden. Nutzungs- und damit Supportzeiten von weit über zehn Jahren kommen so ganz schnell zusammen. In einem solchen Umfeld fällt es schwer, zu schnell auf aktuelle Hype-Technologien zu springen. Bei mehr als zehn Jahren Lebenszyklus kommt und geht mancher Hype – gerade im Umfeld von UI-Technologien- Frameworks, die heute „in" und morgen wieder „out" sind - für ihre Anwendung muss es aber weitergehen.

Es fällt auch schwer, ganz ohne Hemmungen auf HTML/Javascript-basierte Ansätze zu setzen: Zu bescheiden ist immer noch zum Beispiel die Tastatur-Unterstützung, zu hoch ist immer noch der Aufwand, ein und dieselbe Oberfläche auf verschiedenen Browsern in gleicher Qualität zu Verfügung zu stellen. Wenn ein Dialog beim Endkunden nicht läuft, dann ist der Anwendungshersteller immer in der Pflicht, das Problem zu lösen – welche Framework-Schicht darunter auch immer das Cross-Browser-Management behandeln soll.

Warum nicht Java-Swing?

Vor dem Hintergrund der langen Lebenszyklen und der Ergonomie-Anforderungen liegt es nun wieder wesentlich näher, einen Blick auf Java-Swing zu werfen: Swing ist eine „gut abgehangene", mittlerweile sehr performante und stabile UI-Umgebung. Swing-Anwendungen gibt es zu Tausenden, gerade im industriellen Einsatzbereich. Dort sind sie meist die „Arbeitstiere" - oft nicht besonders attraktiv, aber sie laufen und laufen. Mit dem Java Update 1.6.10 hat sich in der Laufzeitumgebung für Swing-Oberflächen (Applet, Webstart) sehr viel getan – zum Positiven.

Natürlich gibt es auch Nachteile, die bei Java-Swing auf der Hand liegen: Zum einen ist dies die Notwendigkeit der Installation eines Java-Client-Plug-ins. Die Installation ist problemfrei, muss aber mit der System-Administration abgesprochen werden. Nicht vergessen: Das Nutzungsumfeld, von dem wir hier sprechen, ist das von Sachbearbeitern; es geht nicht um Anwendungen, die sofort bei jedem anonymen Web-User laufen müssen.

Zum anderen sind diese Nachteile bekannt: Swing ist für UI-Entwickler zwar „normal komplex". Swing ist aber definitiv nicht der richtige Level, den ein Anwendungsentwickler benötigt, um effizient einen Dialog nach dem anderen zu entwickeln. Der Aufwand in Swing, wirklich schöne Oberflächen zu bauen, ist recht groß. Man muss schon (und leider) einige

Kniffe beherrschen, um beim Benutzer einen Wow-Effekt hervorzurufen. Schade ist: Swing kann praktisch alles, um diese Wow-Effekte hervorzurufen. Aber man muss wissen, wie es geht, und es dann machen.
Schließlich: Swing ist eine reine Umgebung zur Gestaltung von Oberflächen. Ob die Anwendung gleich auf dem Client läuft, ob sie zentral läuft – all dies liegt nicht in der Verantwortung von Swing. Swing per se ist also keine „Rich-Client-Umgebung", weil die Art und Weise der Anbindung einer Server-seitigen Anwendung nicht festgelegt ist.

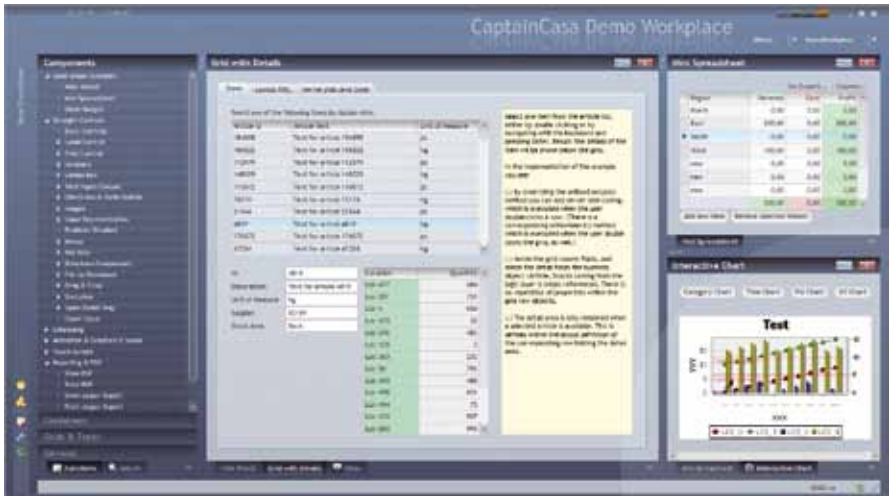


Abbildung 1: Beispieloberfläche CaptainCasa Enterprise Client

Frontend-getriebene Architektur?

Nun wurde in der Community aus recht pragmatischen Gründen ein Java-Swing-basierter Client-Ansatz gewählt. Und nun kommt die spannendste Frage: „Wie wird die Client-Oberfläche an die Server-seitige Anwendungslogik angebunden?"

Die typische, unbedachte Architektur ist eine, die an die Fat-Client-Implementierung der 1990er Jahre erinnert: Vorne im Client wird das GUI programmiert, das über Remote-Schnittstellen (wie Web-Services) auf die Server-seitige Anwendungslogik zugreift. Dieser Frontend-getriebene Ansatz erscheint „natürlich": Im Rahmen der GUI-Entwicklung hat man kompletten Durchgriff auf alle Features der GUI-Umgebung. Und das Backend muss sowieso Schnittstellen nach außen zur Verfügung stellen, nicht nur für GUI-Zwecke. Aber es gibt entscheidende architektonische Nachteile, die dann auftreten, wenn die Anzahl der Dialoge steigt: Die Anzahl der Schnittstellen zwischen GUI-Client und Anwendungsserver steigt beständig. Jedes Komfort-Feature im Frontend geht in der Regel einher mit dem Aufruf einer neuen Schnittstelle zum Server hin.

Dazu ein Beispiel: die Erfassung einer Bestellung erfordert zunächst die typische „CRUD-Schnittstelle" (Create, Retrieve, Update, Delete) für das Objekt „Bestellung". Nun soll der Benutzer aber bei der Eingabe auch die Verfügbarkeit der bestellten Ware sehen. Also wird entweder die CRUD-Schnittstelle erweitert oder eine Schnittstelle „Verfügbarkeit prüfen" implementiert und vom GUI aufgerufen. Nun soll dem Benutzer auch beim Eintippen von Positionen automatisch der Preis berechnet werden. Wiederum wird eine neue Schnittstelle eingerichtet. Und so weiter …

Die hohe Anzahl von Schnittstellen wird verschärft durch die Weiterentwicklung der Anwendung. Es entstehen neue Versionen von Schnittstellen; die alten müssen aber weiter unterstützt werden.

Neben dem immer größer werdenden Aufwand für die Implementierung und das Management dieser Schnittstellen gestaltet sich auch die Implementierung des UI-Clients immer schwieriger: Es muss vermieden werden, dass bei einer einzelnen Benutzeraktion („Benutzer drückt einen Button") eine Vielzahl von Schnittstellenaufrufen zum Server hin gemacht wird. Jeder Aufruf kostet eine gewisse Latenzzeit im Netzwerk – die Antwortzeit beim Benutzer wird bestimmt durch die Summe aller Latenzzeiten aller Aufrufe. Es tritt ein Phänomen auf, das man aus den 1990ern kannte: Im lokalen, schnellen Netzwerk (LAN) funktioniert die Anwendung gut, im langsameren Netzwerk (WAN) funktioniert sie nicht mehr, beziehungsweise äußerst träge.

Server-getriebenes UI

Der natürlich erscheinende, Frontend-getriebene Architekturansatz hat also seine Grenzen, die mit zunehmendem Umfang der Implementierung einer komplexen Anwendung zutage treten. Wie aber sieht eine Alternative aus? Ganz einfach: Im Frontend wird nicht mehr explizit codiert – es gibt also „vorne" keine GUI-Anwendungsentwicklung mehr. Stattdessen wird ein vom Server aus gesteuerter generischer GUI-Client geschrieben unter dem Motto: Der GUI-Client erhält vom Server eine XML-Layoutbeschreibung und zeigt diese an. Nun tätigt der Benutzer seine Eingaben und Aktionen. „Irgendwann" – nicht bei jedem Druck einer Taste, aber beispielsweise beim Drücken eines Buttons – gehen die im Layout geänderten Daten und die Aktion zum Server. Dort werden sie verarbeitet und der Server schickt eine aktualisierte XML-Layoutbeschreibung zum Client zurück.

Dieses Verarbeitungsprinzip kennen wir bereits von den grün-schwarzen Terminals aus den 1980er Jahren (3270-Terminals, da ohne XML) und natürlich auch vom normalen HTML-basierten Browser. Es ist das Prinzip eines „Thin Clients", manche nennen es auch „Formular-Prozessor".

Nun kommt aber ein entscheidendes Merkmal: Wie smart ist der XML-Austausch von Layoutbeschreibungen, wenn es um die Übertragung von Änderungen am Layout geht? Wenn sich in einem Dialog Teile des Layouts ändern, etwa weil durch eine Benutzeraktion weitere Daten sichtbar werden? HTML kennt hier prinzipiell nur den Ansatz, das gesamte Layout noch einmal zu senden – es findet also keinerlei „Delta-Verarbeitung" statt. Und es muss viel AJAX/JavaScript-Technologie zum Einsatz kommen, um dieses nachträglich einzuimpfen.

Es gilt also hier, beim Austausch von Layout-Beschreibungen von vornherein ein klares „Delta-Konzept" in der Kommunikation zu verankern. Wenn sich an einer Oberfläche auf dem Weg vom Client zum Server

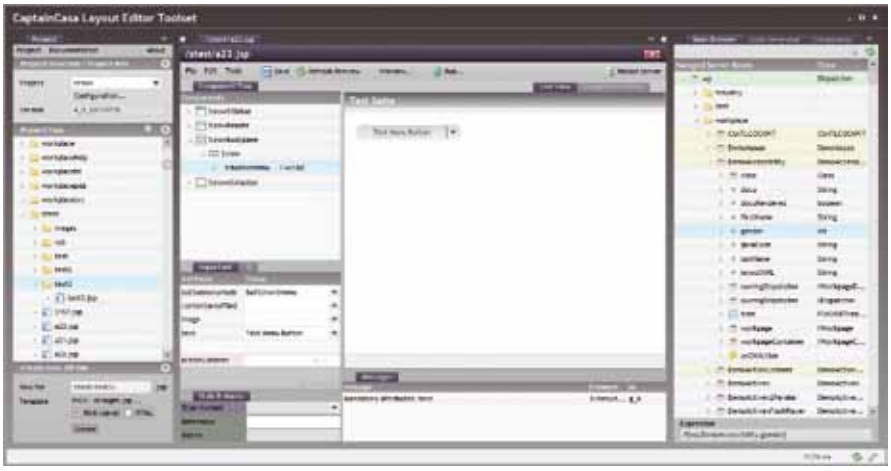


Abbildung 2: Layout-Editor zur WYSIWYG-Erstellung von Layouts

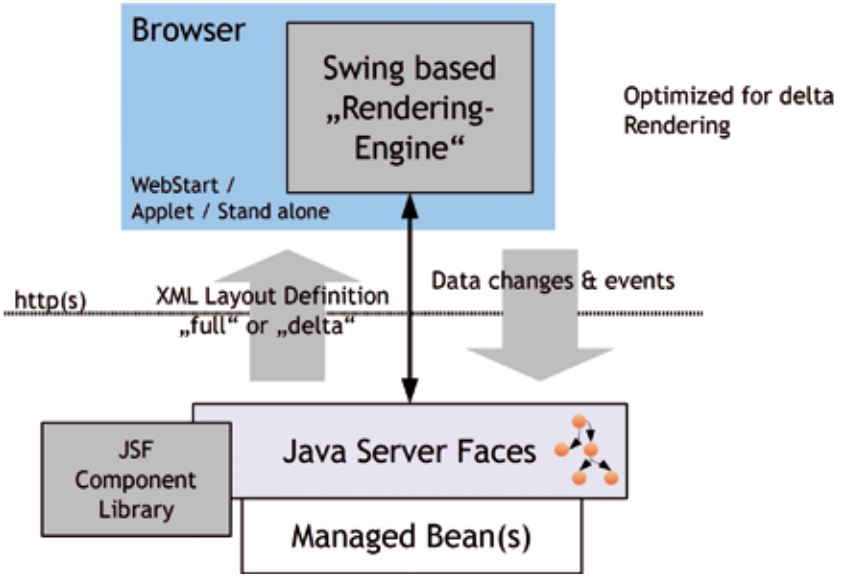


Abbildung 3: Swing-JSF-basierter Thin Client

und zurück nur wenig ändert, dann darf auch nur wenig kommuniziert werden. Dies bedeutet eine signifikante Reduktion des ausgetauschten Datenvolumens und auch eine signifikante Steigerung der Performance am Client. Schließlich muss nicht ein Re-Rendering einer kompletten Seite erfolgen, sondern es müssen nur die Änderungen eingearbeitet werden.

Bevor es mit Technologie weitergeht, wird noch einmal die Server-getriebene Denkweise dieser Frontend-Architektur betont: Im Client selbst erfolgt keine explizite Oberflächen-Entwicklung durch die Anwendung – diese findet komplett im Server statt. Der Client ist eine reine „Rendering-Engine“. Der Client weiß nicht, ob es sich bei einem dargestellten Formular um eine Bestellerfassung oder die Pflege eines Artikels handelt – er rendert das Layout aus und gibt zu definierten Zeitpunkten nach hinten kund, was der Benutzer gerade macht.

Aus Entwicklungssicht bedeutet dieser Server-getriebene Ansatz eine enorme Vereinfachung – es findet nur noch Server-seitige Anwendungsentwicklung statt. Es gibt keinen Zwang mehr, jede „Kleinigkeit“ als Schnittstelle nach außen zu legen, damit sie von der Oberfläche angezeigt werden kann. Die Frontend-Entwicklung läuft quasi im Server und setzt direkt auf lokalen Schnittstellen der Anwendung auf.

Diese Architektur löst viele Probleme der Frontend-getriebenen Architektur: Es gibt nur noch eine Schnittstelle zwischen Client und Server – nämlich die Übertragung der Layout-Daten. Das Prinzip, das eine Nutzeraktion auch nur zu einem Roundtrip führen darf, ist automatisch eingehalten. Und: Fehlerkorrekturen und Weiterentwicklungen in der Anwendung führen nicht zu einem Re-Deployment des Clients für alle Benutzer, da die Änderungen sich alle auf dem Server abspielen.

Bei einer „Frontend-getriebenen“ Architektur ist der Startaufwand klein. Für ein Frontend auf Basis von Java-Swing sind lediglich eine Entwicklungsumgebung zur Java-Swing-Programmierung sowie eine Schnittstellen-Technologie nach hinten erforderlich – schon kann es mit dem ersten Dialog losgehen.

Bei einer „Server-getriebenen“ Architektur ist der Startaufwand zunächst höher: Man braucht die „Rendering Engine“ im Client, diese muss – optimalerweise – über http/s an den Server angebunden sein. In diesem ist eine Server-seitige Dialogverarbeitung zu implementieren, die dann wiederum die eigentliche Anwendung anbindet – eine Menge von aufeinander abzustimmenden Komponenten, die erst einmal zur Verfügung stehen müssen.

CaptainCasa Enterprise Client

Genau diese Infrastruktur ist nun eine, die in der CaptainCasa-Community über eine Vielzahl von Softwarehäusern gemeinschaftlich erarbeitet, genutzt und aktiv gepflegt wird. Im Vorfeld wurde ja bereits erwähnt, dass die Client-Technologie der Wahl „Java-Swing“ heißt. Die beschriebene Rendering-Engine im Client ist also in Java-Swing implementiert.

Interessant ist nun die Frage, wie diese Rendering-Engine an eine Server-seitige Anwendung angeschlossen ist – irgendwo muss ja die XML-Layoutbeschreibung für den Client erstellt werden und irgendwo müssen ja Ereignisse vom Client im Server verarbeitet werden. Hier nun kommt Java Server Faces (JSF) ins Spiel. JSF ist eine Standardtechnologie, die nichts Anderes macht als eine Server-seitige Dialogverarbeitung. Dialoge werden im Server zur Laufzeit als Objektbaum gehalten. Im Falle von HTML entsteht aus dem Objektbaum eine HTML-Seite, indem der Baum

rekursive abgearbeitet wird und jede Komponente des Baums ihren Anteil an HTML erstellt.

JSF ist als Standard nicht auf HTML festgelegt – und das ist gut so. Denn bei CaptainCasa rendert sich der Server-seitige Komponentenbaum nicht in HTML aus, sondern er erstellt genau das XML, das wiederum der Swing-basierte Client benötigt. Beim rekursiven Erstellen des XMLs auf dem Server wird eine Delta-Ermittlung durchgeführt, die dafür sorgt, dass das zum Client gesendete XML nur die Änderungen an einem Layout enthält – und nicht jeweils das gesamte Layout übermittelt wird.

JSF hat den riesigen Vorteil, Standard zu sein. Fragen der Nutzer nach Skalierbarkeit, Failover-Konzepten, Deployment etc. werden über einen Standard beantwortet. Auf dem Server läuft nichts anderes als Standard.

JSF ist komplex, weil es viele Aufgaben flexibel löst und weil es einen ganzen Blumenstrauß von mehr oder minder wichtigen Seitenaspekten enthält, in dem man sich erst einmal zurecht finden muss. JSF darf deswegen nicht ungefiltert an eine Anwendungsentwicklung gegeben werden.

Im Rahmen des CaptainCasa Enterprise Clients wird JSF automatisch gefiltert: Die Anwendungsentwicklung erstellt Seiten über einen WYSIWYG-Editor. In diesem ist das XML-Layout definiert, über Expressions erfolgt die Bindung zu Server-seitigen Anwendungs-Dialogobjekten. Layoutbeschreibung und Dialogobjekt sind modular und multipel verwendbar, das heißt sie können als wiederverwendbare Einheiten in anderen Oberflächen verwendet werden.

Fazit

Wenn es um umfangreiche, langlebige Unternehmensanwendungen geht, sollte man sich gründlich fragen, ob der UI-Hype von heute Grundlage einer Architektur der nächsten zehn Jahre sein sollte. Java-Swing als bewährte UI-Umgebung ist aus diesem Betrachtungswinkel immer noch sehr aktuell. Bei der Anbindung des UI-Clients zum Server hin sollten Sie sich bewusst sein, ob Sie eine Frontend-getriebene (Fat Client) oder eine Server-getriebene (Thin Client) Architektur wählen. Pauschal gilt: Je umfangreicher eine Anwendung wird, desto mehr liegen die Vorteile auf Seiten des Server-getriebenen Ansatzes.

```
HTML-basierte Anwendungen, sondern auch für generische Swing Clients.

Björn Müller
bjoern.mueller@captaincasa.com

Layout Definition (JSF):

<t:rowbodypane>
        <t:row>
                <t:label text="Your Name" width="120" />
                <t:field text="#{DemoHelloWorld.name}"/>
        </t:row>
        <t:row>
                <t:coldistance width="120"/>
                <t:button actionListe-ner="#{DemoHelloWorld.onHello}"
         text="Hello!" />
        </t:row>
        <t:rowdistance height="50" />
        <t:row>
                <t:label text="Result" width="120" />
                <t:field background="#F0F0F0" enabled="false"
        text="#{DemoHelloWorld.output}" width="100%" />
        </t:row>
</t:rowbodypane>

Zugehöriges, Server-seitiges Java Programm (¿Maneged Beans¿):

public class DemoHelloWorld
{
   String m_name;
   String m_output;

   public void setName(String value) { m_name = value; }
   public String getName() { return m_name; }

   public String getOutput() { return m_output; }

   public void onHello(ActionEvent ae)
   {
      if (m_name == null)
         m_output = „No name set.";
      else
         m_output = „Hello World, „+m_name+"!";
   }
}
```

Listing 3

Hierbei trifft man natürlich keine Pauschal-Entscheidung für alle Oberflächen einer Anwendung, sondern hat seine Kern-Dialoge im Blick – diejenigen, die einen mitsamt der Anwendung über den gesamten Lebenszyklus begleiten werden. Und: Java Server Faces ist eine verlässliche, standardisierte Umgebung für die Server-seitige Dialogverarbeitung, nicht nur für HTML-basierte Anwendungen, sondern auch für generische Swing clients.

Björn Müller
bjoern.mueller.@captaincasa.com

Björn Müller arbeitete zunächst zehn Jahre in der Anwendungsentwicklung, Basisentwicklung und Architekturentwicklung für SAP. 2001 erfolgten die Gründung der Casabac Technologies GmbH als Pionier im Bereich von Rich Internet Applications auf Basis von Client-Scripting (AJAX). 2007 gründete er die CaptainCasa Community, eine Verbindung mittelständischer, deutschsprachiger Softwarehäuser mit dem Focus auf Rich Client Frameworks für umfangreiche, langlebige Anwendungssysteme.

Java aktuell – das Abo

4 Ausgaben für 18 Euro

Jetzt Abonnement sichern:

- o Java aktuell – das iJUG-Magazin Abo: vier Ausgaben zu 18 Euro im Jahr
- o Abonnement Newsletter: Java aktuell – der iJUG-Newsletter, kostenfrei

Für Oracle-Anwender und Interessierte gibt es das Java aktuell Abonnement auch mit zusätzlich sechs Ausgaben im Jahr der Fachzeitschrift DOAG News und zwei Ausgaben im Jahr Business News zusammen für 75 Euro. Weitere Informationen unter www.doag.org/shop/

Senden Sie das ausgefüllte Formular an:

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

oder faxen Sie es an:

0700 11 36 24 39

oder bestellen Sie online:

go.ijug.eu/go/abo

Anschrift:

Name, Vorname

Firma

Abteilung

Straße, Hausnummer

PLZ, Ort

ggf. Rechnungsanschrift:

Straße Hausnummer

PLZ, Ort

E-Mail

Telefonnummer

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.

www.ijug.eu