NEU
FRÜHJAHR 2011

Java aktuell

Das iJUG Magazin

Java aktuell

Magazin der Java-Community

Rapid Java Development

Tipps zur Java-Zertifizierung

AJAX vs. Java & Co.

Außerdem:
Interview mit Ed Burns

Ungewisse Zukunft: Die Gerüchteküche um Java

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 02/2011

Editorial

Wolfgang Taschner
Chefredakteur Java aktuell

Ein Java für alle

Die Sache bleibt spannend. Die Zukunft von Java ist ungewiss, die Gerüchteküche brodelt. Seit der Übernahme von Sun durch Oracle sind viele Java-Anwender verunsichert, Oracle hält sich mit klaren Aussage weitgehend zurück. Die ersten Java-Entwickler sehen sich bereits nach Alternativen um. Auch die Community ist aktiv, auf Blogs werden Möglichkeiten diskutiert, wie Harmony und einen Fork vom OpenJDK zu einer eigenständige Sprache zusammenzulegen – unabhängig von Oracle. Die Vision klingt verlockend, denn damit könnte man gleich etliche Altlasten entsorgen und hätte Apache, Google etc. wieder mit im Boot.

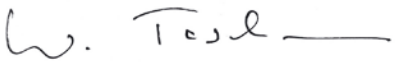
Viele Entwickler würden sich sicher für dieses offenere „Java reloaded" entscheiden, die anderen blieben wahrscheinlich eher beim kommerziell ausgerichteten Original. Beiden Welten wären jedoch nicht vollständig kompatibel zueinander. Letztlich fahren dann zwei Communities mit halber Kraft, und am Ende verlieren vielleicht alle. Zudem würde Oracle den Bestrebungen zu einem neuen offenen Java sicher nicht tatenlos zuschauen. Massive Behinderungen und Patentklagen könnten die Bestrebungen über Jahre blockieren.

So gesehen kann eine Spaltung nicht im Sinne der Community sein – höchstens dann, wenn Java unter Oracle wirklich einen völlig falschen Weg einschlägt. Momentan kann das Ziel jedoch nur lauten: „Ein Java für alle".

Ein anderes Thema: Ende letzten Jahres hat der Interessenverbund der Java User Groups e.V. (iJUG), Herausgeber der Java aktuell, Zuwachs bekommen: Zu den neun Gründungsmitgliedern hat sich die Swiss Oracle User Group (SOUG) gesellt (siehe Seite 37). Zumindest hier stehen die Zeichen eindeutig auf „Eine Community für alle".

Ich wünsche Ihnen viel Spaß beim Lesen dieser Zeitschrift und freue mich auf Ihr Feedback an redaktion@ijug.eu.

Ihr



Inhalt

Ungewisse Zukunft: Die Gerüchteküche um Java

„Das Java-Tagebuch“, *Seite 7*

Interview mit Ed Burns, Spec lead für JSF, *Seite 5*

Ein Erfahrungsbericht zur Java-Zertifizierung „Sun Certified Java Programmer“, *Seite 13*

„Für mich ist der Übergang von Sun zu Oracle ein guter Wechsel …“

Oliver Szymanski (auf den Fotos rechts), Vorstand des iJUG, traf sich am Rande der DOAG 2010 Konferenz + Ausstellung mit Ed Burns, Spec Lead für JSF bei Oracle.

Wie hat sich das Leben durch den neuen „Besitzer“ von Java verändert?

Ed Burns: Es gibt zwei Antworten. Zunächst hat sich nur der Name auf der Gehaltsabrechnung geändert. Es ist sehr gut, dass es noch Gehalt gibt. Auch die Kontinuität hält an. Auf der anderen Seite hat sich die gesamte Philosophie verändert. Sun war ein von Technikern getriebenes Unternehmen. Wir erfanden die Technologie, hatten sie und suchten Märkte. Oracle ist anders, mehr produktbezogen und auf die Werte der Anteilseigner fokussiert. Das ist ein guter Weg, um sich auf etwas zu konzentrieren und die Software mehr in den Mittelpunkt zu stellen. Für mich ist es ein guter Wechsel.

Was ist der größte Unterschied zwischen Oracle und Sun?

Ed Burns: Oracle ist viel größer und bietet mehr Tools. Es ist einfach schön, in einem größeren Team zu arbeiten und mehr Ressourcen zur Verfügung zu haben.

Kommen wir zu der Technologie, die du repräsentierst. Wie ist die Zukunftsvision von JSF?

Ed Burns: Wir haben im Juli 2009 die Version 2.0 herausgebracht. Für Ende 2010 ist ein größeres Maintenance-Release 2.1 geplant. Darin sind einige Fehlerbehebungen enthalten, die Konvertierung von JSP nach Facelets wird einfacher und es gibt neue Möglichkeiten, um Zustände zu speichern, zum Beispiel Zustände in Komponenten, die nur im Request Scope gültig sind. So lässt sich in einer Tabelle eine einzige Zeile oder Zelle gezielt rendern. Anfang 2011 kommt dann ein neuer JCP für JSF, es wird JSF 2.2 für Java EE 7 sein.

Wird es darin viele neue Features für Entwickler geben?

Ed Burns: JSF 2.2 wird einige neue Features haben wie HTML5-Unterstützung oder die Datei-Upload-Komponente. JSF 2.0 benötigte AJAX, das war Pflicht, sonst wäre es mit JSF aus gewesen. Ich denke, der nächste JSR für JSF braucht HTML5. Wir sind in einer guten Position dafür. Für HTML5 haben wir bereits die Separierung von Komponenten-Semantik vom Renderer. Es gab immer die XML-Puristen, die Presentation von Rendering trennen wollten – SGML hatte das bereits zum Ziel. Dann kam HTML und packte alles wieder zusammen. Jetzt wird endlich einiges an Semantik wieder ausgelagert, beispielsweise kann der Browser die neuen HTML5-Range-Tags als Slider oder etwas anderes rendern. Somit bietet HTML5 etwas, das wir in JSF schon lange unterstützen.

DOAG @talk:
Interviews und Gespräche

Das Interview mit Ed Burns ist in englischer Sprache auf Youtube unter http://www.youtube.com/watch?v=SdNMRkR_vM8 zu sehen.

Wenn du selbst frei entscheiden könntest, was würdest du in JSF ändern?

Ed Burns: JSF ist ein sehr stabiler Standard. Ich denke, es ist an der Zeit, etwas auf der Basis von JSF zu machen, möglicherweise einen neuen JSR zur Standardisierung der nützlichen Komponenten. Es gibt eine breite Palette an reichhaltigen Komponenten wie RichFaces, ICEFaces, PrimeFaces oder ADF. Ich würde gerne die wichtigsten identifizieren und in den Standard aufnehmen. Die Herausforderung ist, dabei den Markt für Komponenten nicht zu enttäuschen, denn dieser ist der Schlüssel für den Erfolg von JSF. Der ideale Weg wäre, wenn sich Komponenten-Bibliotheken unterscheiden, aber dennoch die Messlatte für Standard-Komponenten höher gelegt werden kann.

Wie arbeitet ein so großes Team zusammen, das in JSF involviert ist?

Ed Burns: Das Team ist weit verteilt. Es ist aber nicht so groß, wenn man es mit den Oracle-Development-Teams vergleicht. Wir bekommen Unterstützung von vielen, die Code einreichen. In meinen Präsentationen zeige ich immer ein Slide mit allen Gesichtern. JSF ist ein typisches Open-Source-Projekt – es gibt Personen mit unterschiedlich verfügbarer Zeit und verschiedenen Arbeitsplänen. Wir haben einen IRC-Channel (Internet Relay Chat) und benutzen IM (Instant Messaging). Ich bevorzuge IRC, damit andere sehen, was gerade vor sich geht. Das ist gut für das Teamgefühl. Und wir haben eine offene Mailingliste.

Bei Java gibt es den Java Community Process (JCP). Wie lebt ihr den Community-Gedanken in einem realen Projekt wie JSF?

Ed Burns: Wir nutzen viel von dem Spielraum, den uns der JCP für den Spec Lead gibt, und versuchen, so offen wie möglich zu sein. Wir sind die erste EE Spec, die eine öffentliche Open-Source-Referenz-Implementierung hatte, sowie bereits seit 2004 eine transparente Entwicklung. Wir machen unseren JCP-Prozess sehr durchsichtig und haben eine Mailingliste, die lesbar allen zur Verfügung steht. Jeder kann sich anmelden. Die JSF-Experten, die im JSPA eingetragen sind, sind Teil der JSP-Expertengruppe. Darüber hinaus gibt es die erweiterte Expertengruppe für Personen, die wir kennen und die leitend in der Community sind. Diese haben Schreibrechte.

Du hast ein Buch über JSF2 geschrieben. Deckt es alle Inhalte ab, die Entwickler benötigen?

Ed Burns: Ich habe das Buch und die JSF-Spec im selben Zeitraum geschrieben und denke, es deckt die wichtigsten und meisten Dinge ab. Es gibt einige Themen, die ich gerne detaillierter beschrieben hätte, aber leider fehlte mir dazu die Zeit. Vielleicht hole ich das in der Neuauflage von „Java Server Faces 2.0, The Complete Reference“ nach, das ich mit Chris Schalk und Neil Griffin im McGraw-Hill-Verlag veröffentlicht habe.

Vielen Dank für das Gespräch!

Zur Person: Ed Burns

Ed Burns ist derzeit Berater in der Technical Group von Oracle America, Inc. Er leitet die Spezifizierung von JavaServer Faces, dem Standard Web-Applications-Framework fur Java EE. Ed Burns führt in dieser Position seine Arbeit von Sun fort. Er leitet ein Team von Web-Experten aus verschiedenen Industriebereichen, um die JavaServer-Faces-Technologie innerhalb des Java Community Process als Open Source weiterzuentwickeln.
Sein Interesse gilt Web-Applications-Frameworks, AJAX, der Reduzierung von Komplexität, Test-driven Development, dem Sammeln von Anforderungen und der computerunterstützten Zusammenarbeit. Bevor er sich mit JavaServer Faces beschäftigte, arbeitete Ed Burns seit 1994 an verschiedenen Client-/Server-Web-Technologien wie NCSA Mosaic, Mozilla, das Sun Java Plugin, Jakarta Tomcat, das Cosmo Create HTML Authoring-Tool, und dem Web-Transport-Layer im Irix Betriebssystem von Silicon Graphics.
Ed Burns besitzt den Bachelor of Computer Science von der Universität Illinois in Urbana-Champaign. Während des Studiums belegte er Deutsch als Nebenfach und arbeitet für IBM, wo er seine Vorliebe für Computer-Geschichte entdeckte, als er mit der Office-Software des Systems /370 arbeitete.
Ed Burns hält häufig Vorträge auf internationalen Konferenzen, darunter etliche Male auf der JavaOneSM Konferenz.

Das Java-Tagebuch

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

Seit Oracle Sun und damit Java übernommen hatte, gab es viele Fragen und noch mehr Gerüchte, welche Strategie der neue Eigentümer mit Java verfolgen wird. Die Gerüchteküche brodelte, und Oracle selbst hielt sich ziemlich bedeckt.

Es gab eine Reihe von Erklärungen, die auch in der letzten Ausgabe der Java aktuell standen, aber viel Substanzielles war in den offiziellen Statements nicht enthalten – letztlich wurde immer wieder darauf verwiesen, dass Oracle erst einmal Zeit benötigt, um eine publizierbare Strategie zu entwickeln. Dies sollte, so hieß es immer wieder aus dem Hause Oracle, Ende September 2010 geschehen, auf den nun gemeinsam in San Francisco stattfindenden Konferenzen OpenWorld und JavaOne.

Tatsächlich wurde dort auch einiges an Neuigkeiten verkündet. Mindestens ebenso spannend waren aber die Geschehnisse in den Tagen vor und den Wochen nach diesen Konferenzen. Mit dem „Java-Tagebuch" möchte ich in komprimierter Form einen Blick auf die Ereignisse rund um das Thema „Java" werfen: auf Ankündigungen von Oracle und anderen wichtigen Java-Kontributoren ebenso wie auf Kommentare aus der „Community", auf die entscheidenden Presse-Erklärungen ebenso wie auf die kleineren Themen am Rande. Alles aus meiner subjektiven Sicht, aber dennoch mit dem Ziel, einen guten Überblick über die Entwicklungen zu geben. Beginnen möchte ich mit dem Tag vor dem offiziellen Start der „Open World" – an dem erstmals seit längerer Zeit wieder Konkretes zu Java verkündet wurde.

20. September 2010

„It's time for … Plan B"

... so steht es im Blog von Mark Reinhold, dem „Chef-Architekten" der Java-Plattform (http://blogs.sun.com/mr/entry/plan_b). Anfang September hat er in seinem Blog zwei alternative Pläne für das lang ersehnte JDK 7 beziehungsweise eine Aufteilung der Features in ein JDK 7 und ein JDK 8 vorgestellt und um Feedback gebeten – ein Schritt, die Java-Community wieder einzubinden. Jetzt wird die (intern getroffene, aber auch in den Blog-Kommentaren favorisierte) Entscheidung verkündet. Plan B verspricht ein JDK 7 immerhin noch Mitte 2011, aber um wichtige Features reduziert, gefolgt von einem JDK 8 Ende 2012 mit eben diesen wichtigen Features wie Modularisierung („Jigsaw") und Lambda-Ausdrücken. Die Alternative wäre gewesen, bis Mitte 2012 auf das JDK 7 zu warten. In den Kommentaren ist vielfach vom „kleineren zweier Übel" die Rede, die etwas wütenderen Kommentatoren („Wake up Oracle!") haben aber wohl vergessen, dass das Projekt bereits vor der Übernahme von Sun durch Oracle deutlich im Verzug war.

21. September 2010

Offizieller Start der JavaOne – Thomas Kurian stellt Java-Pläne vor

Auf seiner Präsentation zu Beginn der JavaOne bestätigt Thomas Kurian, Chef der Software-Entwicklung bei Oracle, den Plan für das JDK. Darüber hinaus stellt er vor, wie Oracle sich die Zukunft von Java im Desktop- und Mobile-Bereich vorstellt (Interoperabilität von Java und JavaScript, JavaFX, Java Mobile.Next). Die Pläne klingen gut, sind aber in vielen Fällen noch zu unkonkret und insbesondere noch kein Beweis dafür, dass Oracle „ernst macht". Zwei konkrete Informationen gibt es aber doch. Erstens: Netbeans bleibt für Oracle als IDE gesetzt. Damit tritt Kurian den Spekulationen entgegen, dass es als strategisch unwichtigste der drei IDEs neben JDeveloper und Eclipse bald verschwinden wird. Zweitens: Die GlassFish-Version 3.1 mit High-Availability-Features soll wie versprochen bis Ende 2010 fertig werden – ein überschaubarer Zeitraum, an dem Oracle direkt gemessen werden kann.

JavaFX und Swing

Ebenfalls um die Desktop-Entwicklung geht es in einem Blog-Eintrag von Amy Fowler aus dem Java-Entwicklungsteam (http://amyfowlersblog.wordpress.com/2010/09/21/a-heartfelt-ramble-on-swing-javafx/). Interessanter als der reine Inhalt – es geht um die Sorgen von Swing-Entwicklern und was JavaFX für diese Entwickler bedeuten könnte – ist, dass hier mit relativ offenen Worten die „Community" angesprochen wird. Es entwickelt sich auch eine lebhafte Diskussion in den Kommentaren, in der der oberste Produktmanager für die Entwicklungsplattformen (Duncan Mills) sich genötigt fühlt, die IDE-Strategie noch einmal zu präzisieren: Das Ja zu NetBeans heißt nicht, dass jetzt JDeveloper oder das OEPE (für Eclipse) geopfert werden. Es mag ein subjektiver Eindruck sein, aber Oracle scheint langsam seine Kommunikationsstrategie besser an die Java-Community anzupassen.

22. September 2010

„General Technical Session" auf der JavaOne

Auftritt der „Chief Architects": Mark Reinhold (Java Platform Group), Roberto Chinnici (Java EE) und Greg Bollellaa (Embedded Java). Mark Reinhold detailliert im Wesentlichen die Pläne für JDK 7 und 8 und redet über eine Zusammenführung der HotSpot und JRockit JVMs. Roberto Chinnici spricht über die – schon nicht mehr taufrische – Java Enterprise Edition 6 und wirft dann einen Blick auf kommende Releases: Unterstützung für Cloud Computing – etwas, das für die meisten Entwickler, die froh

sind, wenn sie schon Java EE 5 einsetzen können, aber wohl noch weit weg ist. Greg Bollellaa geht ausführlicher auf die Embedded- und Mobile-Pläne aus Thomas Kurians Keynote ein (verbesserte Kompatibilität zwischen CLDC, CDC und SE für Entwicklung und Betrieb für Java ME.next) sowie auf die Integration von Java ME und Web-Technologien (HTML, JavaScript). Klingt gut, aber angesichts des iPhones und der Diskussion um Android bleibt die Zukunft für Java im Mobile-Bereich spannend. Vielleicht gibt die vielzitierte „ability to execute" nach Gartner ja den Ausschlag zugunsten von Oracles Plänen.

27. September 2010

Die Zukunft von JavaFX Script: Project Visage

Für alle Freunde von JavaFX Script gibt es Hoffnung: JavaFX-„Guru" Stephen Chin hat ein Projekt ins Leben gerufen, das JavaFX Script erhalten und weiterentwickeln will (http://steveonjava.com/accouncing-visage/).

30. September 2010

The Register meldet „JCP-Aufstand"

Die Internetseite „The Register" berichtet über eine nicht-öffentliche Abstimmung im Executive Committee (EC) des Java-Community-Process kurz vor der OpenWorld (http://www.theregister.co.uk/2010/09/30/oracle_alone_on_java/). Dessen Mitglieder (unter anderem IBM, Google, VMware und Vodafone) hätten gegen Oracle eine Resolution verabschiedet, dass der JCP in eine unabhängige und neutrale Organisation ausgelagert werden soll. Oracle selbst hat diese Forderung noch 2007 gegenüber Sun vertreten, aber nach der Übernahme von Sun – wenig überraschend – das Interesse daran verloren. Die Resolution an sich dürfte ohne Folgen bleiben, aber die Auswirkungen des Kräftemessens im JCP sind noch nicht abzusehen. Wenn man den Rechtsstreit mit Google sowie den seit Sun-Zeiten andauernden Konflikt mit der Apache Software Foundations hinzunimmt, ist für weiteres politisches Spektakel gesorgt – sicher nicht zum Besten von Java.

10. Oktober 2010

Java SE: Weitere Details zu „Plan B"

Wie Mark Reinhold in seinem Blog berichtet, wurde Oracles „Plan B" auf einem Meeting des JCP Executive Committee in Bonn von den anderen Mitgliedern abgesegnet (http://blogs.sun.com/mr/entry/plan_b_details). Er kann nun in Form von JSRs in den JCP einfließen. Befürchtungen, dass der JCP aufgrund der politischen und juristischen Auseinandersetzungen zwischen einzelnen Parteien in seiner Arbeit behindert wird, haben sich damit erst mal nicht bestätigt.

11. Oktober 2010

IBM und Oracle wollen zusammen am OpenJDK arbeiten

IBM und Oracle verkünden in Presseerklärungen, dass sie in Zukunft gemeinsam am OpenJDK arbeiten wollen (http://emea-pressoffice.oracle.com/Press-Releases/Oracle-and-IBM-Collaborate-to-Accelerate-Java-Innovation-Through-OpenJDK-173d.aspx). Ein gelungener Schachzug von Oracle. Was immer hinter den Kulissen ausgehandelt wurde, es dürfte für Oracle nicht einfach gewesen sein, IBM mit ins Boot zu holen. Gleichzeitig ist dies ein herber Verlust für die Apache Software Foundation beziehungsweise ihr Open-Source-JDK „Project Harmony" (das auch als Grundlage für Android gedient hat), da IBM sich hier sehr stark engagiert hat und dies nun einstellen wird. Letztlich dürfte die Zusammenarbeit am OpenJDK aber der Java-Community zugute kommen.

20. Oktober 2010

Apple will Java für OS X nicht weiter pflegen

Ganz unscheinbar in den Release Notes für „Java for Mac OS X 10.6 Update 3 and 10.5 Update 8" steht ein Satz, der für Aufruhr in der Java-Community sorgt: „... the Java runtime ported by Apple and that ships with Mac OS X is deprecated" – die von Apple gepflegte Runtime könnte in zukünftigen OS X Releases gar nicht mehr vorhanden sein. Das wäre ein schwerer Rückschlag für „write once, run anywhere". Wer übernimmt?

08. November 2010

Eine kostenpflichtige Virtual Machine?

Verschiedene Online-Magazine und Blogs berichten, dass Oracle in Zukunft auch eine kostenpflichtige Version der JVM auf Grundlage der zusammengeführten Hotspot und JRockit VMs anbieten will – dies habe Vice President Adam Messinger auf einer Konferenz gesagt (http://www.heise.de/open/meldung/Oracle-will-auch-kostenpflichtige-Java-Virtual-Machine-anbieten-1131860.html). Die Nachricht sorgt natürlich auf der Stelle für Unruhe in der „Blogosphäre" – die Verfechter freier Software (wie in „freies Bier") sehen ihre Albträume nach der Sun-Übernahme bestätigt. Die Nachricht ist aber viel zu unkonkret, um daraus wirkliche Schlüsse zu ziehen. Letztlich war auch JRockit (ursprünglich von BEA entwickelt) immer eine kostenpflichtige JVM, wenn auch nicht einzeln lizensierbar, sondern nur mit dem Weblogic Server.

09. November 2010

Die Apache Software Foundation droht aus dem JCP auszusteigen

Die ASF wurde gerade für weitere drei Jahre in das Executive Committee des JCP gewählt. In einer Erklärung nach der Wahl droht sie aber gleich, die Zusammenarbeit im JCP zu beenden, wenn der langwährende Streit um das TCK nicht endlich zu ihren Gunsten beendet wird. Oracle sei ebenso wie Sun durch das Java Specification Participation Agreement (JSPA) vertraglich verpflichtet und solle endlich das tun, was Sun jahrelang verweigert hat: Der ASF ein Test Compatibility Kit (TCK) ohne weitere Lizenzeinschränkungen zu gewähren, damit die Open Source JVM „Harmony" als Spezifikations-konform unter einer Apache-Lizenz veröffentlicht werden kann. Auch ohne Rechtsexperte für Lizenzen zu sein, denke ich, dass dies so ziemlich das Letzte ist, was Oracle während eines laufenden Rechtsstreits um Android tun wird, das auf einem Teil von „Harmony" basiert. Wir dürfen auf weitere Schritte gespannt sein.

(https://blogs.apache.org/foundation/entry/statement_by_the_asf_board1)

10. November 2010

Oracle präzisiert Aussagen zur kostenpflichtigen JVM

In seinem Blog stellt Henrik Ståhl, bei Oracle verantwortlich für die Java-Produktstrategie, weitere Details zu den JVM-Plänen vor. Dabei geht er insbesondere auf die kommerziellen Versionen ein. Oracle wird weiterhin die kostenlose Implementierung unterstützen beziehungsweise selber herausbringen (OpenJDK beziehungsweise die konvergierenden VMs JRockit und Hotspot). Nur einige „value-adds" sollen proprietär bleiben, sie sollen im Rahmen kostenpflichtiger Versionen erhalten sein (namentlich erwähnt werden „Java for Business" und „JRockit Mission Control", „JRockit Real Time" und „JRockit Virtual Edition"). Diese „value-adds" sollen Verbesserungen in Management und Deployment für Enterprise-Umgebungen bringen. Andere wichtige Features („all JRockit performance features") sollen dagegen ins OpenJDK einfließen. Nicht explizit erwähnt wird die Implementierung neuer „value-adds" – ansonsten würden die kostenpflichtigen Versionen irgendwann obsolet. Aber das ist vermutlich nur eine sprachliche Ungenauigkeit (http://blogs.oracle.com/henrik/2010/11/oracles_jvm_strategy.html).

12. November 2010

Java für Mac OS – die Wiederauferstehung

Oracle und Apple verkünden, dass nun auch Apple im OpenJDK-Projekt mitarbeiten wird. Nur wenige Tage nach Apples Ankündigung, dass Java für OS X nicht mehr weiter gepflegt werde, kehrt auf dieser Seite wieder Ruhe ein, und das OpenJDK-Projekt wird weiter aufgewertet. Natürlich werden sofort Hoffnungen in der Java-Community geweckt, dass demnächst Java-Applikationen über den AppStore vertrieben werden können. Aber bis dahin ist es sicher noch ein weiter Weg. Aktuell wird Java namentlich als eine für den AppStore nicht zulässige Technologie erwähnt.

16. November 2010

Oracle und die Apache Software Foundation, reloaded

Das Spiel geht weiter. Oracle hat auf die Drohung der ASF geantwortet, die Zusammenarbeit mit dem JCP einzustellen: Oracle stelle das TCK unter fairen Bedingungen bereit und es sei Zeit, gemeinsam an der Zukunft von Java zu arbeiten, statt die Stagnation der vergangenen Jahre fortzusetzen (http://blogs.oracle.com/henrik/2010/11/moving_java_forward_open_response_from_oracle_to_apache.html). Die ASF antwortet prompt: „Oracle, the ball is in your court. Honor the agreement" (https://blogs.apache.org/foundation/entry/statement_by_the_asf_board2).

Die JSRs für Java SE 7 und 8 sind da

An anderer Stelle gibt es positive Nachrichten: Die JSRs für Java SE 7 und 8 sind beim Java-Community-Process eingereicht worden. Insgesamt sind es vier: jeweils einer für den „Release Content", einer für „Project Coin", das viele kleinere Sprachverbesserungen zusammenfasst, und einer für die „Lambda Expressions". Bis zum 6. Dezember 2010 können nun Kommentare abgegeben werden beziehungsweise Bewerbungen zur Teilnahme an der Expert Group für die Spezifikationen. Dann wird abgestimmt, ob die JSRs akzeptiert werden und damit der eigentliche Spezifikationsprozess starten kann. Ob die Apache Software Foundation sich vom Nikolaus milde stimmen lässt?

18. November 2010

Java auf der DOAG-Konferenz

Im Rahmen der DOAG 2010 Konferenz hat das Thema „Java" dieses Jahr logischerweise einen höheren Stellenwert als zuvor. Trotz der zeitlichen Kollision mit der Devoxx sind auch viele prominente Oracle-Vertreter aus Entwicklung und Produktmanagement vor Ort. Eines der Top-Themen ist natürlich immer noch Java EE 6 mit den Einzelspezifikationen wie JSF 2.0, das tatsächlich die Augen vieler Entwickler wieder zum Leuchten bringt, die ja in der Vergangenheit mit der sperrigen Java-EE-Spezifikation nicht verwöhnt waren. Noch ist nicht alles perfekt, aber Java EE ist wieder eine echte Alternative zu den Frameworks (wie Spring) geworden, die ja überhaupt erst durch die – drastisch ausgedrückt – Unbenutzbarkeit vergangener Spezifikationen entstanden sind.

Weitere wichtige Themen sind auch hier die Zukunft von JavaFX sowie von Java für Embedded und Mobile. Eine Reihe von Fragen bleiben offen, etwa wie Oracle sich die Zukunft von Java für Mobile jenseits von Projekten wie Java ME.next und LWUIT vorstellt: Wie sieht die Strategie für Android aus; wäre eine Zusammenarbeit mit Apple irgendwann auch für das iPhone denkbar? Aber es ist wohl utopisch, darüber jetzt etwas zu erfahren. Generell ist das Feedback aus der Community: Es passiert in den letzten Monaten mehr als in den Jahren zuvor; die Leute warten jetzt einfach ab, ob Oracle seine Versprechen einlöst – bis Mitte 2011 wissen wir mehr. Diese – verglichen mit den teilweise hitzigen Diskussionen in der Blogosphäre – doch recht entspannte Haltung mag teilweise dadurch begründet sein, dass die klare Mehrheit der Java-Fans hier deutlich näher an Oracle „dran" ist und die Übernahme nicht ganz so argwöhnisch beäugt wie der Rest der Community. Vielleicht ist es auch ein Zeichen dafür, dass diese Diskussionen – so sehr sie natürlich sinnvoll sind, um die Entwicklung von Java voranzutreiben – auch nicht den Querschnitt der Community widerspiegeln.

Devoxx 2010

Die Devoxx ist als etablierte Java-Konferenz natürlich ein Anziehungspunkt für alle Java-Fans, von denen die meisten bislang wohl weniger mit Oracle zu tun hatten. Aber auch hier ist die Stimmung schon entspannter, als manche Blog-Einträge im Vorfeld vermuten lassen. Einem Kommentar zufolge ist „der erste Schock verdaut". Eine nette Idee des Veranstalters ist es, Whiteboards mit bestimmten Fragen aufzustellen, auf denen die Teilnehmer dann per Strichliste abstimmen oder Kommentare hinterlassen können. Auf die Weise wird einiges an Feedback aus der Community gesammelt, auch wenn es nicht unbedingt repräsentativ ist. So ist beispielsweise eine deutliche Mehrheit überzeugt, dass die Klage gegen Google eine schlech-

te Idee war. Bei der Frage, ob Oracle an die Java-Community glaubt, entscheidet sich die Mehrheit für „Vielleicht, sie sehen es nur etwas anders als Sun." Eine weniger politisch aufgeladene, aber mindestens ebenso spannende Abstimmung: Bei der Frage, ob Java 8 oder 9 mit vorhergehenden Versionen inkompatibel sein soll, setzt die überwältigende Mehrheit ihren Strich unter „Ja, es wird Zeit, altes Zeug zu entfernen, um Platz für Neues zu schaffen". Die Fotos davon sind unter (http://picasaweb.google.com/JavaPolis.com/Devoxx2010) zu sehen, und im Weblog von Stephen Colebourne findet sich auch eine Abschrift (http://www.jroller.com/scolebourne/entry/devoxx_whiteboard_votes_2010).

26. November 2010

Kairos und Chronos

Eine sehr interessante Interpretation von Geschwindigkeit und Zeit in Bezug auf Java findet sich in Markus Eiseles Blog. Er bezieht sich auf eine Umfrage auf java.net, ob die aktuelle Geschwindigkeit für Java SE 7 und 8 angemessen sei. Dies setzt er in Zusammenhang mit den altgriechischen Zeit-Begriffen Kairos (qualitativ, der rechte Zeitpunkt) und Chronos (quantitativ, die messbare Zeit). Eine Analyse vergangener Release-Zyklen zeigt beide am Werk: Releases etwa alle zwei Jahre, aber die „geraden" Releases beinhalten große, revolutionäre, die „ungeraden" weniger tiefgreifende Veränderungen. Dies hat sich seit Release 1.5 verändert. Jetzt warten wir seit vier Jahren auf Release 7. Aber die Zeit muss nicht verloren sein, sie hat vielleicht einfach Gelegenheit gegeben, „Luft zu holen". Den alten Zweijahres-Turnus mit der Unterscheidung zwischen revolutionären und evolutionären Releases sieht er dennoch für die Zukunft als sinnvoll an (http://blog.eisele.net/2010/11/thoughts-on-javas-pace-will-kairons.html).

06. Dezember 2010

Die Wahlbüros haben soeben geschlossen

Die „Review Ballots" für die vier JSRs zu Java SE 7 und 8 sind abgeschlossen. Die Apache Software Foundation ist bei ihrer Linie geblieben und hat alle vier JSRs abgelehnt – mit der Begründung, dass sie zwar inhaltlich zustimmen, aber nicht mit „ja" stimmen könnten, solange der Specification Lead „die vertraglichen Pflichten aus dem JSPA verletze". Auch andere Mitglieder des Exceutive Committees unterstützen diese Position: Google, Tim Peierls, RedHat, die Eclipse Foundation und sogar IBM gehen in ihren Kommentaren zur Abstimmung darauf ein. Die meisten stimmen aber trotzdem mit „ja" – um die dringend nötige Weiterentwicklung von Java nicht aufzuhalten. So werden zumindest die beiden ersten JSRs „Small Enhancements to the Java Programming Language" und „Lambda Expressions for the Java Programming Language" mit nur einer Gegenstimme und einer Enthaltung (Tim Peierls) akzeptiert. Weniger einheitlich sieht das Bild bei den wichtigen „Umbrella JSRs" für SE 7 und 8 aus, die den Gesamtumfang der Releases definieren. Neben der ASF votieren auch Google und Tim Peierls mit „nein". Aber auch diese werden mit immer noch großer Mehrheit (12 ja-Stimmen) akzeptiert. Sorge und Erleichterung halten sich also weiter die Waage: Der Streit um Lizenzen und die Zusammenarbeit im JCP wird weitergehen, aber es geht trotzdem inhaltlich voran – zügiger als in den vergangenen Jahren.

07. Dezember 2010

Der nächste Akt im JCP-Streit: Tim Peierls steigt aus

Es hat nur einen Tag gedauert: Nach dem Ende der Review Ballots für Java SE 7 und 8 verlässt Tim Peierls, unabhängiges Mitglied im Executive Committee für Java SE, das Gremium. In seinem Blog (http://tembrel.blogspot.com/2010/12/resigned-from-ec.html) berichtet er, dass er nicht mehr daran glaubt, mit seiner Stimme in diesem Gremium etwas auszurichten. Ausschlaggebend sei insbesondere die Tatsache gewesen, dass Oracle zu Fragen um die hier bereits mehrfach erwähnten Lizenzbedingungen für das TCK geschwiegen habe; außerdem habe Oracle in einer Telefonkonferenz des Executive Committee verkündet, sie wollen mit den JSRs weitermachen, egal wie das Votum ausfalle.

Inhaltlich unterstützt er die Arbeit an den JSRs weiterhin, fügt aber hinzu, dass das Java-Ökosystem seiner Meinung nach so groß und reich sei, dass es auch ohne die Weiterentwicklung in den JSRs eine vielversprechende Zukunft haben könne. Im Prinzip hat er damit wohl Recht. Schwächen in den Spezifikationen – oder „steckengebliebene" Specification Requests wurden immer durch Weiterentwicklungen durch Dritte kompensiert (beispielsweise das alte J2EE durch Spring). Aber hier geht es um die gemeinsame Basis für den Sprachstandard, von der nicht nur Java selbst, sondern sogar andere Sprachen wie Scala und Clojure betroffen sind, die auf der JVM laufen. Da wäre es ein schlechtes Zeichen, wenn jetzt, wo die Weiterentwicklung endlich wieder Fahrt aufnehmen soll, stattdessen eine weitere Zersplitterung eintritt.

09. Dezember 2010

Und noch ein Paukenschlag: Die ASF steigt ebenfalls aus

Man hätte es nach dem Ausstieg von Tim Peierls erwarten können. Auch die Apache Software Foundation steigt aus dem JCP aus und legt ihren gerade erst wiedererlangten Sitz im Executive Committee für Java SE/EE nieder. Die Gründe sind letztlich dieselben (größtenteils altbekannten) wie bei Tim Peierls. Während man den Austritt von Tim Peierls vielleicht hätte ignorieren können, ist dies hier ein wirklich ernstes Signal. Die ASF hat viel Gewicht in der Java-Welt und es wäre schlecht für die Zukunft des Ökosystems, wenn sie und Oracle gegeneinander arbeiten würden.

Oracles Antwort auf den Ausstieg

Oracle antwortet postwendend: Java-„Produktstratege" Henrik Ståhl veröffentlicht in seinem Blog ein Statement von Adam Messinger (Vice President of Development bei Oracle). Darin steht aber auch nur, dass Oracle die Verantwortung dafür trägt, dass Java weiterentwickelt wird, und zwar unter Einhaltung von Standards. Die ASF möge daher bitte ihre Position noch einmal überdenken, da sie einen wichtigen Beitrag zum Java-Ökosystem leiste.

Die Fronten bleiben also weiterhin verhärtet. Aus der Community gibt es (bis-

lang einzelne) Stimmen, unabhängig vom Java-Standard, aber aufbauend etwa auf Harmony / dem OpenJDK, eine eigenständige Sprache mit eigener VM zu etablieren. Bei allen Schwierigkeiten, die ein solches Projekt vermutlich hätte (siehe „Android", Oracle würde auch hier sicher nicht tatenlos zusehen), wird es so weit hoffentlich nicht kommen. Was kostet es Oracle, das TCK für Java SE uneingeschränkt freizugeben? Es geht um den Markt für Mobilfunkgeräte und entsprechende, nicht unbeträchtliche Lizenzeinnahmen (für Java ME). Schon Sun hat jahrelang verweigert, Apache Harmony eine TCK-Lizenz zu erteilen, die die volle Nutzung für mobile Geräte erlaubt (inklusive der Patentrechte) – um eben diese Lizenzeinnahmen vollständig für sich zu haben. Nun bedroht Android die Lizenzeinnahmen, und Oracle sieht in der TCK-Lizenzfrage wohl einen wichtigen Baustein seiner Argumentation im Rechtsstreit: Google hat Android auf einem Apache Harmony aufgebaut, das eben keine vollen Lizenzrechte (einschließlich der Patente) für den Mobilfunkbereich hat. Und solange dieser Rechtsstreit nicht beigelegt ist, besteht wenig Hoffnung auf eine Einigung mit der ASF (http://blogs.oracle.com/henrik/2010/12/oracle_response_to_apache_departure_from_jcp.html).

14. Dezember 2010

Myriad verklagt Oracle wegen Lizenzgebühren

Das muntere Klagen geht in die nächste Runde. Myriad, ein schweizerischer Hersteller von Software für den Mobilfunkmarkt (unter anderem für Java ME und Android), hat Klage gegen Oracle wegen „unfairer Lizenzbedingungen" eingereicht. Sie werden dabei, wie im „fosspatents"-Blog nachzulesen ist, von der gleichen Firma vertreten wie Google im Android-Rechtsstreit (http://fosspatents.blogspot.com/2010/12/google-ally-myriad-group-sues-oracle.html). Ein Schelm, wer Böses dabei denkt. Na, so ein Zufall ... Oracle hat noch am selben Tag mit einer Gegenklage geantwortet. Wir werden wohl noch mehr Kapitel dieser Geschichte erleben. Wann kommt die nächste positive Nachricht? Müssen wir bis Mitte 2011 warten?

28. Dezember 2010

JBoss 6 freigegeben – mit Java EE 6 „Web Profile"-Unterstützung

Wir wollen das Jahr nicht mit Negativ-Schlagzeilen enden lassen. Heute hat JBoss das Release 6.0.0 des gleichnamigen Application Servers als „final" freigegeben. Es unterstützt im Gegensatz zu GlassFish und JEUS noch nicht die komplette Java EE 6-Spezifikation – aber mit dem „Web Profile" immerhin eine in der Spezifikation definierte Teilmenge, die für viele oder die meisten Projekte ausreichen dürfte. Ein wichtiger Meilenstein für die Verbreitung der „Enterprise Edition" (http://community.jboss.org/wiki/AS600FinalReleaseNotes).

Andreas Badelt ist Berater und Softwareentwickler / -Architekt bei der CGI Information Systems and Management Consultants (Deutschland) GmbH. Sein Schwerpunkt ist die Realisierung von großen eCommerce-Systemen, insbesondere für die Telekommunikationsbranche. Daneben organisiert er seit 2001 die Special Interest Group (SIG) Development bzw. SIG Java der DOAG.

Stuttgarter Test-Tage

JUGS java user group stuttgart

Nach den erfolgreichen Stuttgarter Test-Tagen im Jahr 2009 veranstaltet die Java User Group Stuttgart am 4. und 5. Mai 2011 den Workshop erneut. Sie laden dazu noch Referenten ein. Gesucht sind Erfahrungen im Softwaretest sowie Methoden und Werkzeuge für den effizienten Test.

Der Hintergrund: Softwaretest spielt in der täglichen Projektpraxis eine große Rolle. Ein erheblicher Teil des Gesamtaufwands wird für das Testen verwendet. Dennoch ist der Test oft lückenhaft und unvollständig, und bekannte Methoden und verfügbare Testwerkzeuge werden unzureichend eingesetzt. Zum Schluss stellt man immer wieder fest, dass nicht alle Fehler gefunden wurden.

Für die Stuttgarter Test-Tage 2011 plant die Java User Group Stuttgart die Umsetzung eines neuen Konzepts. Morgens tragen die Referenten vor, Nachmittags werden die Themen unter Anleitung in Arbeitsgruppen in der Praxis vertieft. Die Vorträge dauern jeweils dreißig Minuten mit anschließender Diskussion. Für die Arbeitsgruppen stehen vier Stunden am Nachmittag zur Verfügung. Der Vorteil: Jeder Teilnehmer vertieft das Thema, das ihn am meisten interessiert.

Referenten können sich mit einem Vortrag und einer Übung beteiligen. Dazu schicken sie bis zum 11. Februar 2011 einen Abstract des geplanten Vortrags und der geplanten Übung (jeweils maximal eine halbe DIN-A4-Seite) per E-Mail an aop@jugs.org. Für den Vortrag sind dreißig Minuten einzuplanen, für die Übungsaufgabe sollte ein Teilnehmer nicht länger als eine Stunde benötigen, da die Teilnehmer die Arbeitsgruppen wechseln können, um sich verschiedene Methoden und Werkzeuge anzuschauen.

Weitere Informationen unter http://jugs.org

JavaFX Update

Oliver Szymanski, Freiberufler, Source-Knights.com

JavaFX begann als GUI-Lösung für unterschiedliche Client-Arten als Kombination aus neuer Skriptsprache, Programmier-Schnittstelle und einem zugrunde liegenden Framework. Dabei wurde besonders auf die allgemeine Java-Devise „Write once – run everywhere" Wert gelegt und daher Zielgeräte wie Browser mit Java-1.5-Unterstützung, Mobile Client und TV-Geräte zusätzlich zu Rich-Client-Anwendungen unterstützt.

Ob ein neues GUI-Framework nötig ist oder war, liegt sicherlich in der Sichtweise des Betrachters. Fakt ist aber, dass Swing immer wieder dazu führt, dass sich einige Entwickler über alternative Frameworks Gedanken machen. Und wenn man Alternativen braucht, scheint etwas in bestehenden Lösungen zu fehlen oder falsch zu sein.

JavaFX

JavaFX ist eine GUI-Technologie. Darin enthalten sind bisher die JavaFX-Skriptsprache, die JavaFX-APIs (etwa für Transformationen auf Elementen der Oberfläche) und die Laufzeit-Umgebung. Diese liegt für die verschiedenen Client-Arten vor und ist auch für die Ereignisbehandlung und den Lebenszyklus der Applikation verantwortlich.

Das für die Entwicklung benötigte JavaFX-SDK hat Compiler für die Skriptsprache, die Laufzeitumgebung und einige Tools (wie den JavaFX Mobile Emulator) im Lieferumfang. Kompilieren und Ausführen kann man in JavaFX geschriebene Anwendungen dann zum Beispiel über die Kommandozeile mit den Tools „javafxc" und „javafx". Die Integration in die Entwicklungsumgebung ist in NetBeans gegeben oder in Eclipse mit dem JavaFX-Plugin. Die erforderlichen Tools kann man über die offiziellen Webseiten (http://javafx.com) von JavaFX finden.

Meine größte Kritik an JavaFX war seit Beginn dieser Technologie der Faktor, dass hier eine neue Skriptsprache künstlich geschaffen wurde. Ohne expliziten Grund ist keine neue Sprache erforderlich, und Java-Entwickler sollten nicht gezwungen werden, für Aufgaben, die sie ohne Schwierigkeiten in Java selbst entwickeln können, eine neue Sprache zu lernen. Die Begründung war, dass Designer die Skriptsprache leichter lernen und damit GUIs erstellen können. Gut, das lassen wir mal kurz wirken, und dann kann sich jeder selbst Gedanken darüber machen, ob er glaubt, jemals einen reinen Designer zu sehen, der JavaFX lernt und darin GUIs definiert. Designer haben ihre eigenen Werkzeuge und teils eigene Sprachen. Wir sollten uns nicht damit beschäftigen, wie wir die Arbeit der Designer ändern, sondern wie wir unsere darauf abstimmen. Realistisch betrachtet hätte diese Skriptsprache doch, wie auch in anderen Technologien, dazu geführt, dass Java-Entwickler sie zusätzlich lernen und damit Design (re-) implementieren. Das kennen wir bereits von JSPs, JSF und mehr. Doch zum Glück hat Oracle gelernt. Vielleicht wurden sie von Technologien wie dem Webframework JSXP (http://jsxp.org) inspiriert. Denn die neue Skriptsprache hat ihre aktive Phase bald abgeschlossen, die Zukunft von JavaFX beinhaltet die Skriptsprache nicht mehr. Oracle wird die Weiterentwicklung der Skriptsprache einstellen, diese wird dann wahrscheinlich der Open-Source-Community zur Verfügung gestellt.

Die Zukunft?

Doch ohne die Skriptsprache wird JavaFX als Technology keineswegs sterben. Im Gegenteil, ich denke der Wegfall der Skriptsprache wird JavaFX stärken. Da die Skriptsprache bereits vorher zur Definition von Klassen und zur Laufzeit zu Objektinstanzen führte und in Java ByteCode kompiliert wurde, war das Ergebnis kompatibel zu Java. Wir konnten innerhalb der Skriptsprache auf die gesamte Java-Welt zugreifen. Dies ermöglicht die kommende komplette Integration der JavaFX-Technologie in Java, sodass GUIs nicht mehr in einer Skriptsprache, sondern in Java-Code definiert werden. JavaFX wird stärker mit Swing verbunden und somit ist der Weg frei in Richtung einer einheitlichen und starken GUI-Technologie in Java. Wer gern sehen möchte, wie so etwas aussehen kann, sollte sich Amino von Josh Marinacci ansehen. Der Entwickler, der aus dem JavaFX-Team stammte, hat mit Amino (http://leonardosketch.org/amino) ein ähnlich geartetes, reines Java-GUI-Framework geschaffen.

Nach meiner Meinung zur Abstimmung von Entwicklung und Design, scheint Oracle diese bei JavaFX nicht nur beim Wegfall der Skriptsprache zu teilen. Mit der kommenden Version von JavaFX wird es auch Möglichkeiten geben, JavaFX GUIs mit Adobe Photoshop CS4 and Adobe Illustrator CS4 zu definieren. Das wird die Designer sicher mehr freuen als eine neue Sprache. Ob sich in der OpenSource-Community genügend Unterstützung findet, die JavaFX Skriptsprache weiterzupflegen, wird sich zeigen.

Auf der DOAG 2010 Konferenz gab es von Dr. Stefan Schneider (Oracle Deutschland) auch die Aussage, die JavaFX-Technologie werde jetzt wieder stärker auf Desktop-Anwendungen fokussiert. Was dies konkret für die anderen Client-Typen bedeuten wird, muss abgewartet werden. Ich hoffe nicht, dass der „Write once – run everywhere"-Gedanke bei JavaFX verloren geht. Es ist (war?) eine der Stärken dieser Technologie. Auf eine andere können wir uns jetzt im reinen Java freuen: die Binding-API, um auf Wertänderungen von Variablen automatisch zu reagieren und abhängige Werte direkt neu zu ermitteln.

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

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

Fotos: DOAG

Die Java-Zertifizierung „Sun Certified Java Programmer“ – ein Erfahrungsbericht

Kai Wähner, MaibornWolff et al GmbH

Der Sun Certified Java Programmer (SCJP) – zukünftig auch Oracle Certified Professional Java Programmer genannt – ist ein weltweit anerkanntes Zertifikat, das dem Besitzer die Kenntnis der wichtigsten Konzepte und Application Programming Interfaces (API) der Programmiersprache Java in der Standard Edition (Java SE) bescheinigt. Allerdings existieren auch Meinungen: „Zertifizierungen bringen eh nichts, und die gestellten, unlesbaren Code-Schnipsel sind fern von jeder Realität“. Dieser Artikel ist ein Erfahrungsbericht über die Durchführung der SCJP-Zertifizierung – Java SE 6 und zeigt dabei einige Tipps und Fallstricke auf.

Viele Entwickler stehen vor der Frage, ob sie an der Prüfung zur SCJP-Zertifizierung teilnehmen oder darauf verzichten sollen. Oftmals schreckt auch das fehlende Wissen über den damit verbundenen Aufwand sowie die eigentlichen Prüfungsmodalitäten ab. Der Autor beschreibt daher kurz den fachlichen Inhalt der SCJP-Zertifizierung und diskutiert das „Für“ und „Wider“ einer Teilnahme sowie den Prüfungsablauf. Danach wird erläutert, warum die Durchführung überhaupt sinnvoll erscheint, und detailliert auf den Prüfungsablauf eingegangen. Es werden wichtige Hilfsmittel zur Unterstützung genannt und Tipps zur Vorbereitung auf die Zertifizierung gegeben. Darüber hinaus skizziert der Autor seine Verbesserungsvorschläge, zieht ein persönliches Fazit und geht auf weitere Zertifizierungen ein.

Warum ist diese Zertifizierung überhaupt sinnvoll?

Die SCJP-Zertifizierung ist den meisten Java-Entwicklern bekannt. Dennoch sind viele nicht bereit, den Aufwand für die Prüfung in Kauf zu nehmen. Es stellt sich die berechtigte Frage, warum diese Zertifizierung überhaupt gemacht werden sollte. Die naheliegendste Antwort ist natürlich die Zertifizierung an sich, die dem Java-Entwickler nach erfolgreichem Abschluss der Prüfung die grundlegenden Kenntnisse dieser Technologie bescheinigt und sowohl den Kunden als auch potenziellen Arbeitgebern vorgelegt werden kann.

Durch die Teilnahme an diesem Zertifizierungsprogramm werden die technischen Grundkenntnisse von Java erworben und die wichtigsten Konzepte hierzu verinnerlicht. Dies resultiert in der Regel in besserem und sichererem Quellcode, der einfacher zu warten ist, sowie in einer Reduzierung der Duplikate. Beispiele dafür sind:

- Verwendung von StringBuilder statt StringBuffer oder gar statt der „+“-Konkatenation zur Performance-Verbesserung
- Einsatz von „static initializer“ für die Erstellung einer Thread-sicheren Singleton-Klasse
- Anonyme innere Klassen nicht nur einsetzen (beispielsweise in der Swing-

API), sondern auch verstehen und in eigenen Klassen sinnvoll nutzen

Die Zertifizierung beinhaltet die wichtigsten Designkonzepte von Java sowie wichtiger APIs, die jedem Entwickler bekannt sein sollten. Dies betrifft beispielsweise die gängigsten Konzepte der Objektorientierung wie Vererbung, Polymorphie und innere Klassen. Aber auch die Themen Sprachsyntax, API, Generics, Multithreading und die Anwendungserstellung werden behandelt.

Der Prüfungsablauf

Die Prüfung wird in einem von vielen „Prometric Test Centern" [1] durchgeführt, die weltweit und auch in Deutschland in jeder größeren Stadt zur Verfügung stehen. Die Prüfung kostet 235 Euro und dauert 180 Minuten. Dabei müssen 58,33 Prozent (35 von 60) der „Multiple-Choice"- und „Drag-and-Drop"-Fragen richtig beantwortet sein, auf dem Zertifikat ist diese Zahl allerdings nicht sichtbar. Das Prüfungsergebnis wird unmittelbar im Anschluss an die Prüfung mitgeteilt. Auch Upgrades sind möglich, diese fragen nur die Änderungen beziehungsweise Neuigkeiten zur letzten durchgeführten Zertifizierung ab, beispielsweise bei einem Upgrade von SCJP5 auf SCJP6.

Vorsicht ist bei der Auswahl des Test-Centers geboten, da seitens des Anbieters Prometric keine Qualitätssicherung durchgeführt wird. Aus eigener Erfahrung ist zu berichten, dass Monitore mit guter Bildqualität, akzeptable Licht- und Temperaturverhältnisse sowie geringe Lautstärke nicht immer selbstverständlich sind.

Außerdem ist zu beachten, dass die Firma Prometric, die in Deutschland nicht ansässig ist, keine Rechnung auf den Firmennamen ausstellt, sondern nur auf den Test-Kandidaten – eine nicht akzeptable Situation für die Buchführung der eigenen Firma. Als Begründung gibt Prometric an, dass das Zertifikat für eine konkrete Person gilt und daher die Rechnung auch immer nur auf diese Person ausgestellt werden kann. Als Lösung bleibt nur die Möglichkeit, dass die Firma den Test-Voucher direkt bestellt oder aber die Anmeldung direkt beim ausgewählten Test-Center durchgeführt wird anstatt auf der Prometric-Webseite.

Hilfsmittel zur Vorbereitung

Als Hilfsmittel sollten Bücher, Foren sowie Prüfungssimulationen (sogenannte „Mock-Examen") verwendet werden. Zu empfehlen ist das Buch „SCJP Sun Certified Programmer for Java 6 Study Guide" von Katherine Sierra und Bert Bates (ISBN 978-0071591065). Die Autoren sind auch an der Erstellung der SCJP-Prüfungsfragen beteiligt. Neben ausführlichen Erläuterungen des Prüfungsinhalts sind viele „Question & Answer"-Blöcke inklusive Erläuterungen nach jedem Kapitel sowie zwei vollständige Mock-Examen enthalten.

Das „JavaRanch"-Forum bietet zudem einen eigenen, sehr gut besuchten Bereich, der ausschließlich die SCJP-Zertifizierung thematisiert [2]. Fragen werden in der Regel innerhalb weniger Stunden beantwortet, auch die Autoren des obigen Buches sind dort tätig.

Mock-Examen ermöglichen eine realistische Prüfungsvorbereitung zu Fragen, die alle Themen der Zertifizierung enthalten. Die oben erwähnten zwei Examen als Beilage des Buches sind jedoch nicht ausreichend, da nicht alle in der Prüfung vorkommenden Fallstricke behandelt werden. Zahlreiche weitere Mock-Examen sind allerdings im Internet verfügbar. Der Kauf von kostenpflichtigen Examen ist nicht notwendig, da genug kostenloses Material kursiert. Für den Einstieg ist die Webseite „Java Black Belt" [3] gut geeignet, die unabhängig vom SCJP-Inhalt grundlegende Java-Kenntnisse kostenlos abfragt. „Exam Labs" [4] bietet danach ausreichend zusätzliche Fragen zur Vorbereitung auf die SCJP-Prüfung an. Die grafische Oberfläche ist dabei nahezu identisch zur wirklichen Prüfung. Zu beachten ist, dass diese Fragen sehr schwierig sind. Wer hierbei etwa 50 Prozent der Fragen korrekt beantworten kann, sollte keine Probleme beim Bestehen der eigentlichen Prüfung haben.

Tipps zur Vorbereitung

Im Folgenden werden einige Tipps gegeben, welche bei der Vorbereitung auf die Prüfung beachtet werden sollten. Der gesamte Inhalt – wirklich jeder Absatz des oben genannten Buches – ist prüfungsrelevant, sofern dies nicht ausdrücklich anders erwähnt wird. Wirklich jede Zeile kann geprüft werden und ist insofern relevant zur Vorbereitung. Die Konzepte und APIs sollten dabei nicht nur auswendig gelernt, sondern wirklich praktisch durch Programmierung nachvollzogen werden, und zwar ohne Entwicklungsumgebung, da diese schließlich auch während der Prüfung nicht verfügbar sein wird. Dies gilt insbesondere für Konstrukte, die zwar oft verwendet werden, die jedoch nicht jeder Entwickler in voller Tiefe versteht. Zudem dürfen Mock-Examen nicht einfach nur durchgeführt werden. Einen großen Teil der Zeit sollte man für die Analyse falsch beantworteter Fragen aufwenden, um sich Konzepte einzuprägen und diese zu verstehen. Auch das Prinzip der „Test-taking Methodology" sollte zum Einsatz kommen. Konkret bedeutet das, Schlüsselwörter wie beispielsweise „may" oder „must" zu beachten, das Schema der Fragestellungen zu verstehen (etwa alle korrekten oder alle falschen Antworten auswählen) und immer erst die Syntax zu prüfen, dann die Logik (außer es gibt gar keine „does not compile"-Antwort).

Für den Teilnehmer bietet es sich zudem an, sich erst dann zur Prüfung anzumelden, wenn die Vorbereitung nahezu abgeschlossen ist. Anderenfalls wird nur unnötiger Druck aufgebaut. Prüfungstermin und Uhrzeit sind frei wählbar und auch wenige Tage vor der Prüfung kurzfristig möglich. Die Prüfungsdauer ist bei guter Vorbereitung völlig ausreichend. Es bietet sich an, zuerst alle einfachen beziehungsweise schnell lösbaren Aufgaben abzuschließen und erst am Ende die komplizierten zu beantworten. Obwohl die Prüfung auch auf Deutsch durchführbar ist, empfiehlt es sich, die englische Variante auszuwählen. Dies liegt darin begründet, dass die deutsche Übersetzung von eher schlechter Güte ist. Außerdem sind sowohl in Qualität als auch in Quantität keine gleichwertigen deutschen Hilfsmittel zur Prüfungsvorbereitung verfügbar.

Verbesserungsvorschläge

Generell ist anzumerken, dass hinsichtlich der SCJP-Prüfung einige Verbesserungen angemessen erscheinen. Sicherlich sollte ein zertifizierter Java-Entwickler die wichtigsten APIs kennen – das Auswendiglernen vieler Methoden, wie etwa der Collection-Klassen, ist allerdings nur wenig hilfreich. Aus diesem Grund gibt es APIs, um die Funktionalität nachzuschauen. Stattdessen sollten lieber wichtige und essenzielle Bereiche zur Zertifizierung hinzugefügt werden, die bisher fehlen. Beispiele sind hier Annotations, Reflection und das Classloader-Konzept.

Des Weiteren erscheinen manche Fragen zweideutig, weshalb eine konkrete Antwort oft nicht möglich ist. Dies gilt insbesondere für das Thema „Coupling, Cohesion and Encapsulation". Verbesserungswürdig ist auch die Situation der schlechten Prüfungsbedingungen in manchen Test-Centern. Eine Rechnung auf Firmenadresse sollte, wie bereits erwähnt, ebenfalls ermöglicht sein.

Fazit

Die Vorbereitung auf die SCJP-Zertifizierung ist sehr aufwändig. Sofern diese hauptsächlich nach Feierabend oder am Wochenende durchgeführt werden soll, sind auch für Java-erfahrene Entwickler mindestens zwei Monate Vorbereitungszeit einzuplanen. Dennoch lohnt sich die Zertifizierung für jeden Java-Entwickler, da nicht nur die Zertifizierung, sondern auch ein verbessertes Java-Wissen als Ergebnis herauskommen. Auch wenn noch Zertifizierungen für ältere Java-Versionen möglich sind, sollte man unbedingt den SCJP für die aktuelle Version Java SE 6 machen.

Ausblick auf weitere Java-Zertifizierungen

Im Java-Umfeld sind noch zahlreiche weitere Zertifizierungen verfügbar. Der Sun Certified Java Associate (SCJA) prüft lediglich die grundlegenden Konzepte und richtet sich daher eher an Einsteiger. Für professionelle Entwickler ist diese Zertifizierung nur wenig hilfreich. Auf dem SCJP aufbauend sind einige weitere Zertifizierungen verfügbar, welche insbesondere einzelne Bereiche der Java Enterprise Edition (JEE) prüfen, wie beispielsweise Web-Entwicklung (Sun Certified Web Component Developer), Komponentenentwicklung (Sun Certified Business Component Developer) oder Web Services (Sun Certified Developer for Java Web Services). Praxiserfahrung im jeweiligen Gebiet sollte vorhanden sein, ansonsten ist die Zertifizierung nur schwer zu schaffen – auch deshalb, weil die Hilfsmittel bei weitem nicht die Anzahl, Aktualität und Qualität des SCJP erreichen.

Des Weiteren sind im Java-Umfeld auch Zertifizierungen für konkrete Produkte, wie beispielsweise der IBM Certified Solution Developer – Web Services Development for WAS 7.0, verfügbar. Hierbei ist allerdings oftmals ein anderes Vorgehen zu empfehlen: Diese Zertifizierungen sind insbesondere dann sinnvoll, wenn das konkrete Produkt in einem Projekt eingesetzt wird. Für die meisten Produkte werden mehrtägige Vorbereitungskurse angeboten, wodurch die Zertifizierung oft innerhalb einer Woche erledigt werden kann.

Kontakt:
Kai Wähner
kai.waehner@mwea.de

Kai Wähner ist als IT-Consultant bei „MaibornWolff et al" tätig. Seine Schwerpunkte liegen in den Bereichen JEE, EAI und SOA. Außerdem berichtet er auf „www.kai-waehner.de/blog" über seine Erfahrungen mit neuen Technologien, IT-Konferenzen und Zertifizierungen.

Quellen:

1. http://www.prometric.com
2. http://www.coderanch.com/forums/f-24/java-programmer-SCJP
3. http://www.blackbeltfactory.com
4. http://examlab.tk/

Single Sourcing mit JVx

René Jahn, SIB Visions GmbH

Hinter dem Begriff „Single Sourcing" versteckt sich die Möglichkeit, eine Anwendung einmalig zu codieren und mit unterschiedlichsten Technologien – ohne Source-Code-Änderung – zu verwenden. Dieser Artikel zeigt, wie man Single Sourcing in JVx umsetzt, und eine Beispiel-Anwendung demonstriert, wie eine Anwendung als Desktop- sowie als Web-Anwendung läuft.

Abhängig von den Projekt- oder Kundenanforderungen stehen bereits frühzeitig geeignete Technologien und Frameworks für die Umsetzung fest. Um Desktop-Anwendungen zu entwickeln, sind hingegen andere Frameworks und Mechanismen erforderlich. Im Bereich der RIAs sieht es nicht anders aus.

Ein häufiges Problem bei der Entwicklung ist die Komplexität und die Menge eingesetzter Frameworks, denn nicht immer harmonieren diese problemlos miteinander. Außerdem sind nicht in jedem Unternehmen ausreichend Spezialisten dafür vorhanden. Wenn die Unterstützung von mobilen Geräten, die Nutzung mit einem Browser, der Komfort einer Desktop-Anwendung, kurze Antwortzeiten und die Bewältigung von Millionen Datensätzen gefordert werden, dann verursachen die Komplexität und der zu erwartende Aufwand erhebliches Kopfzerbrechen. Spätestens dann wird nach neuen Lösungen gesucht, um die Entwicklung vielleicht doch etwas einfacher und auch effizienter durchführen zu können. Das Enterprise Application Framework JVx ist in diesem Zusammenhang mehr als eine Alternative. Durch den Single-Sourcing-Mechanismus benötigt man nur ein einzelnes Framework für die Entwicklung von RIAs, Web- oder Desktop-Anwendungen.

Die Architektur

Das Framework ist ein sogenanntes „Full-Stack Application-Framework" für die Entwicklung von Multi-Tier-Anwendungen und beinhaltet Lösungen für alle Tiers (siehe Abbildung 1). Wir beschränken uns hier jedoch auf die Client-Tier, da nur diese für Single Sourcing relevant ist.

Die Basis bildet das Interface-basierte User Interface (Generic UI). Mit simplen Java Interfaces wird ein User Interface abstrahiert und es ist lediglich zu definieren, welche Eigenschaften die einzelnen GUI-Komponenten unbedingt benötigen. Die Label-Definition „ILabel" demonstriert das:

```
public interface ILabel extends IComponent,
                                 IAlignmentConstants
{
      public String getText();
      public void setText(String pText);
}
```

Das Interface legt fest, dass ein Label ausschließlich einen Text darstellt – auch wenn in manchen Technologien ein Label zusätzlich mit einem Icon ausgestattet sein kann. Das wirkt zwar auf den ersten Blick sehr eingeschränkt, ist aber mit den flexiblen Layout-Managern in der Verwendung kein Nachteil. Für die üblichen GUI-Komponenten wie Button, Checkbox, Menü, Tabelle und Tree existieren ebenfalls passende Interfaces. Das Generic UI ist durch

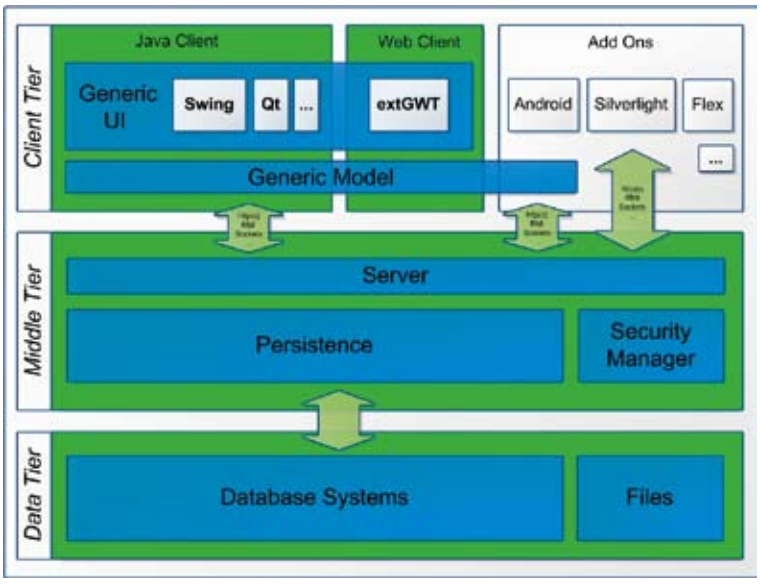


Abbildung 1: JVx-Architektur

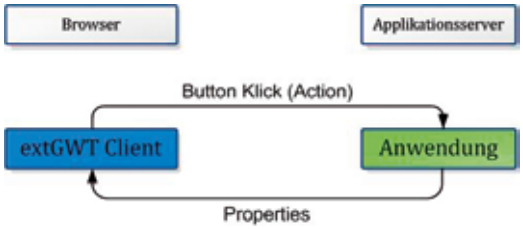


Abbildung 2: WebUI

```
public class SwingLabel extends SwingComponent<JLabel>
                        implements ILabel
{
       public SwingLabel()
       {
              super(new JLabel());
       }

       public String getText()
       {
              return resource.getText();
       }

       public void setText(String pText)
       {
              resource.setText(pText);
       }
}
```

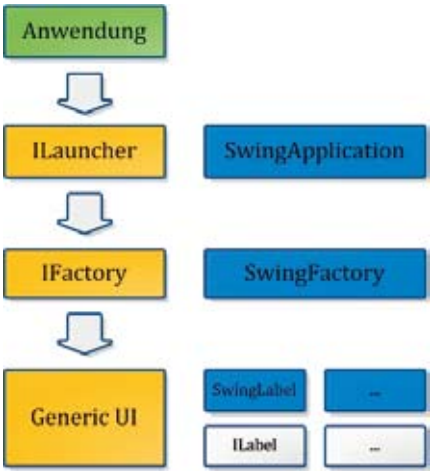


Abbildung 3: Anwendung, Launcher und Factory

die reine UI-Definition vollkommen unabhängig von jeglichen UI-Technologien.

Eine Technologie-abhängige Implementierung muss in weiterer Folge die Interfaces einbauen, um mit JVx kompatibel zu sein – denkbar wären eine SWT- oder JavaFX-Implementierung. Für Swing und Qt Jambi sind die Interfaces bereits eingesetzt. Die Swing-Implementierung für ein Label sieht wie folgt aus:

Die Implementierung greift auf Standard-Swing-Komponenten zurück und stellt diese als Ressource zur Verfügung (siehe Konstruktor). Eine weitere Implementierung existiert für Web-Anwendungen. Diese unterscheidet sich allerdings von den anderen Umsetzungen, da die Anwendung am Server ausgeführt wird. Dafür wurde eine Interface-Implementierung auf Property-Basis durchgeführt. Der Client ist in diesem Fall eine separate Applikation, die mit dem Server bei speziellen Events, wie Button-Klick, kommuniziert. Bei jeder Kommunikation werden die aktuellen Properties zum Client übermittelt und dieser aktualisiert zur Laufzeit die geänderten Komponenten. Die Client-Applikation selbst basiert auf GWT und wurde mit extGWT umgesetzt (siehe Abbildung 2). Die Server-Implementierung ist vollkommen unabhängig von der Client-Technologie und könnte auch mit Javascript-Clients zum Einsatz kommen.

Bisher wurde sehr viel über Interfaces gesprochen, jedoch sind diese ohne Hilfsmechanismen wie das Factory-Pattern nicht initialisierbar und würden die GUI-Entwicklung nicht sehr intuitiv gestalten:

```
ILabel label = getFactory().
createLabel();
label.setText(„Vorname“);
```

Um die gewohnte Arbeitsweise beibehalten zu können, wurde eine Technologie-unabhängige Standard-Implementierung durchgeführt, um Klassen-orientiert zu entwickeln:

```
UILabel label = new UILabel();
label.setText(„Vorname“);
```

Der Anwendungs-Entwickler verwendet ausschließlich das Generic UI und entscheidet sich erst später für eine bestimmte Technologie. Für den Start der Anwendung in einer bestimmten Technologie sind sogenannte „Launcher“ notwendig. Diese sind immer abhängig von der eingesetzten Technologie und initialisieren lediglich die passende UI-Factory für die Anwendung (siehe vorletztes Listing). Für Swing existieren die Launcher „SwingApplet“ und „SwingApplication“. Beide Implementierungen greifen auf die „SwingFactory“ zurück. Abbildung 3 zeigt die Abhängigkeiten zwischen Anwendung, Launcher, Factory und Generic UI.

Eine Beispiel-Anwendung

Das Beispiel ist eine Datenbank-Anwendung mit den Features „Anmeldung mit Benutzername und Passwort“, „Menü und Toolbar“ sowie einer Bearbeitungsmaske (CRUD) für eine Tabelle aus der Datenbank. Wir verwenden hierfür die bereits lauffähige Anwendung (die erste Applikation) des Frameworks. Das Archiv kann von [1] geladen werden. Es enthält ein vorkonfiguriertes Eclipse-Projekt, alle notwendigen Bibliotheken, den Source Code und die zugrundeliegende HSQL-Datenbank. Das Archiv wird einfach an beliebiger Stelle entpackt und das Projekt in Eclipse importiert:

1. Menü: File/Import…
2. Auswahl: General, Existing Projects into Workspace
3. <Projektverzeichnis>/rad/apps/firstapp wählen
4. Projekt JVxFirstApp importieren

Das Projekt wird problemlos kompiliert und sollte sich dann, wie in Abbildung 4 gezeigt, in Eclipse darstellen.

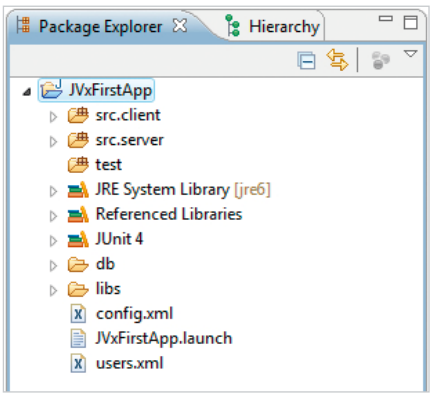


Abbildung 4: Eclipse-Projekt

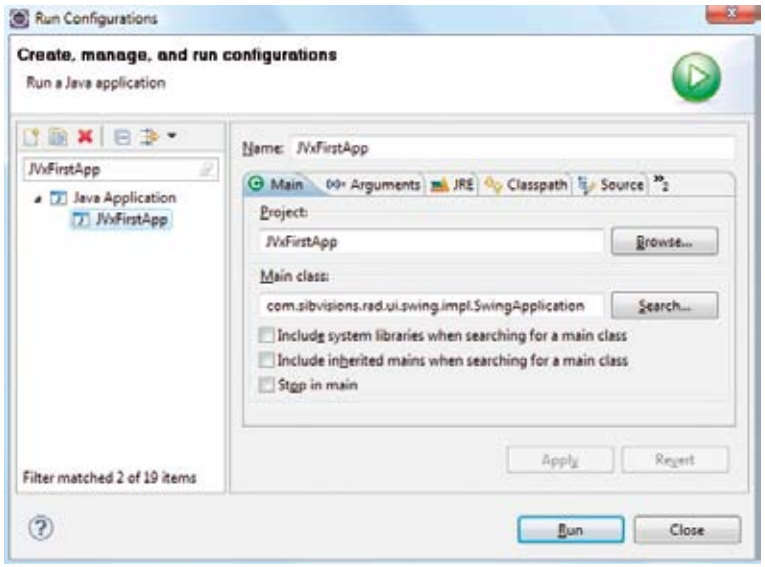


Abbildung 5: Run Configuration Main

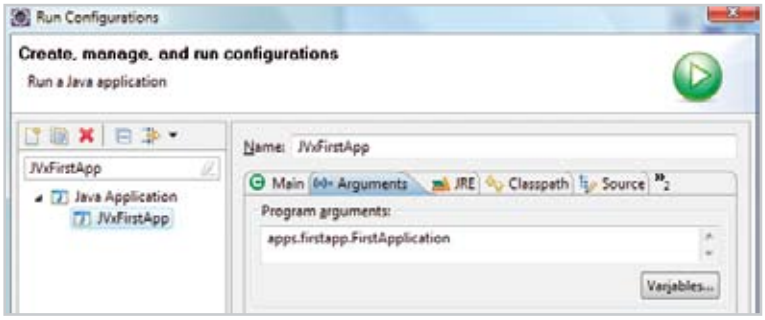


Abbildung 6: Run Configuration Arguments

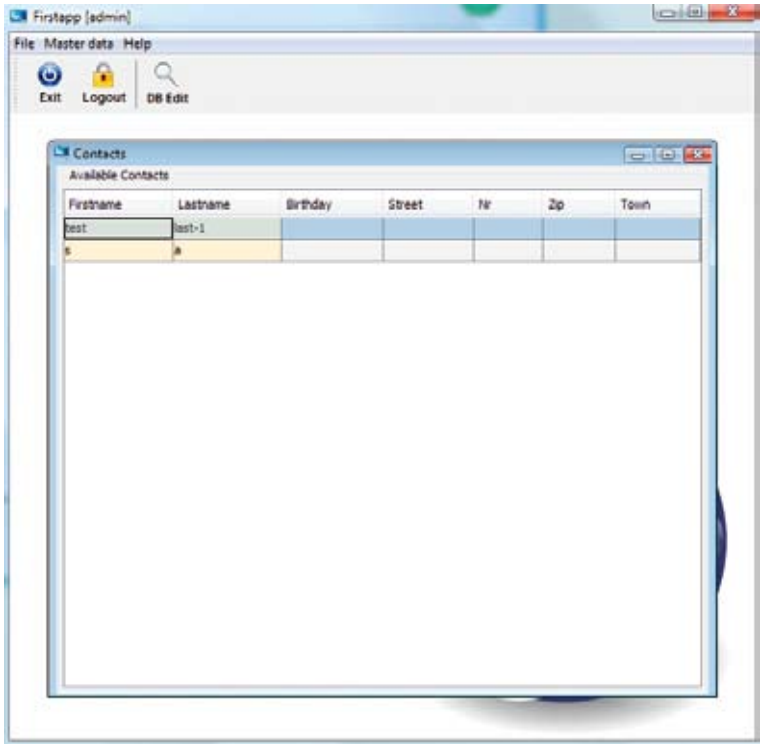


Abbildung 7: Applikation, Swing

Nun wird die Anwendung zum ersten Mal als Swing Application gestartet. Dafür öffnen wir die Run Configurations (Menü: Run) und verwenden entweder die vorhandene Konfiguration JVxFirstApp oder erstellen eine neue mit folgenden Einstellungen (siehe Abbildung 5):

Als Main Class verwenden wir die Klasse „com.sibvisions.rad.ui.swing.impl.SwingApplication" als Launcher. Nun fehlt nur noch die Anwendung selbst. Diese wird als Argument „apps.firstapp.FirstApplication" an den Launcher übergeben (siehe Abbildung 6).

Bevor wir die Anwendung ausführen, ist noch unsere Datenbank zu starten. Dazu reicht es aus, die Datei „<Projektverzeichnis>/rad/apps/firstapp/db/startHSqlDB.bat" auszuführen. Nach dem Start der Anwendung können wir uns mit dem Benutzernamen „admin" und dem Passwort „admin" an der Anwendung anmelden. In der Toolbar befindet sich ein Icon zum Öffnen unserer Bearbeitungsmaske (siehe Abbildung 7).

Nun wird die gleiche Anwendung als Web-Anwendung gestartet. Dazu benötigen wir die Web-Implementierung des Generic UI, die wir von [2] laden. Das Archiv enthält die Client- und Server-Komponenten für die Installation in einem Applikationsserver. Wir verwenden hierfür einen Apache Tomcat 6. Der Server muss bereits installiert und lauffähig sein. Nun richten wir unsere Web-Anwendung ein:

1. Die Verzeichnisse <Tomcat>/webapps/firstapp und <Tomcat>/webapps/firstapp/WEB-INF erstellen
2. Den Inhalt des client-Verzeichnisses aus dem Archiv in den firstapp-Ordner kopieren
3. Den Inhalt des server-Verzeichnisses aus dem Archiv in den firstapp/WEB-INF-Ordner kopieren
4. Die Datei firstapp/WEB-INF/web.xml öffnen, den Parameter „main" auf apps.firstapp.FirstApplication und den Parameter „config" auf einen Leerstring setzen
5. In der Datei firstapp/webui.html die kompletten META-Tags mit den Namen gwt:property löschen

Zum Abschluss benötigen wir noch die kompilierte Anwendung. Dazu müssen wir folgende Schritte durchführen:

1. Den Inhalt des Verzeichnisses <Projektverzeichnis>/rad/apps/firstapp/classes in ein zip-Archiv mit dem Namen „firstapp.jar" packen
2. Die Datei in das Verzeichnis <Tomcat>/webapps/firstapp/WEB-INF/lib kopieren
3. Die Datei <Projektverzeichnis>/rad/apps/firstapp/libs/server/hsqldb.jar in das Verzeichnis <Tomcat>/webapps/firstapp/WEB-INF/lib kopieren
4. Die Verzeichnisstruktur für <Tomcat>/webapps/firstapp/WEB-INF/rad/apps/firstapp und <Tomcat>/webapps/firstapp/WEB-INF/rad/server erstellen
5. Die Dateien <Projektverzeichnis>/rad/apps/firstapp/config.xml und users.xml anschließend nach <Tomcat>/webapps/firstapp/WEB-INF/rad/apps/firstapp kopieren
6. Die Datei <Projektverzeichnis>/rad/server/config.xml nach <Tomcat>/webapps/firstapp/WEB-INF/rad/server kopieren

Nun starten wir unseren Tomcat, und nach der Eingabe von „http://localhost/firstapp/webui.html" erhalten wir unsere Applikation als Web-Anwendung mit den gleichen Funktionen wie in der Swing-Anwendung.

Fazit

Die Beispielanwendung könnte man auch als Applet im Browser oder mit Webstart anbieten. Die Anwendung selbst muss in keinem Fall geändert werden. Es reicht aus zu konfigurieren, mit welcher Technologie die Anwendung zu starten ist. Der Aufwand für die Erstellung der Konfiguration hält sich auch in Grenzen, da die Konfiguration für weitere Anwendungen wiederverwendet werden kann. Üblicherweise würde das Build-System diese Aufgaben übernehmen.

Links

[1] http://www.sibvisions.com/files/download/releases/jvxfirstapp-1.1.zip

[2] http://sourceforge.net/projects/jvx/files/WebUI/0.7%20beta/jvxwebui-0.7%20beta.zip/download

Weitere Informationen

Details zu JVx und die Dokumentation: http://www.sibvisions.com/jvx

Kontakt:
René Jahn
rene.jahn@sibvisions.com

René Jahn ist Mitbegründer der SIB Visions GmbH und Head of Research & Development. Er verfügt über langjährige Erfahrung im Bereich der Framework-Entwicklung und der Qualitätssicherung von Software-Projekten.

Direct-Call-Pattern

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

Wie jeder weiß, schließt die beste Art und Weise, Objekte miteinander kollaborieren zu lassen, Annotationen (siehe Spring und EJB für Beispiele), einiges an XML (wieder mit einem Blick auf Spring und EJB), noch mehr XML (siehe Oracle ADF) und dazu Drag & Drop in einem obskuren visuellen Editor (etwa wieder ADF) ein. Das ist Industrie-Standard und es ist wunderbar, denn Consultants können so viel Geld machen und Consulting-Unternehmen können noch mehr Consultants in Projekte bringen. Wie auch immer, es mag Situationen geben, in denen wir etwas anderes benötigen. Daher haben wir das Direct-Call-Pattern entwickelt (in einer Cocktailbar).

Direct-Call-Pattern [1]

(auch bekannt als **Collaborateur-With-No-Intermediator** Entwurfsmuster)

Einführung

Zwei Objekte wollen miteinander kommunizieren. Einer ist der Aufrufer (Caller), der andere der Aufgerufene (Callee). Für den Aufgerufenen ist das völlig in Ordnung und der Aufrufer macht alles Erforderliche vor und nach dem Aufruf gerne selbst.
Anforderungen: Zwei Objekte, manchmal ein Objekt in zwei Rollen (Aufrufer, Aufgerufener), sonst nichts.
Lösung: Der Aufrufer ruft den Aufgerufenen direkt auf. Kein Proxy, Interceptor oder sonst ein Vermittlerobjekt sind im Aufruf involviert. Wirklich, bloß ein Aufruf von einem vertrauenswürdigen Freund zum anderen. Vielleicht loggt eine dritte Partei den Aufruf, aber das lässt sich heutzutage schwer vermeiden.

Pros

- Schneller Aufruf
- Saubere und kurze Stack-Traces
- Weniger Verwirrung
- WYSIWYG

Contras

- Viele „WTF ist hier die Dependency Injection"- und „Wie finde ich den Interceptor"-Kommentare
- Verärgert das nicht benötigte Annotationen- und Aspektorientierungs-Zeug
- Andere können den Code verstehen, den man geschrieben hat
- Weniger Geld mit Software-Support

Nutzen, wenn

- Kein Geld für Application Server vorhanden ist
- Keine Zeit ist, um Frameworks zu debuggen
- Man sich nicht nach langen Zeiten voll exklusivem Consulting sehnt
- Projekt-Laufzeit länger als einige Wochen/Monate beträgt (und man sich selbst ohne Aufpreis um den Support kümmern soll)

[1] http://source-knights.com/2010/11/direct-call-pattern.html

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

Rapid Java mit XDEV 3

Gerald Kammerer, freier Redakteur

4GL-Tools wie Access, FoxPro, Oracle Forms, PowerBuilder, Filemaker oder Visual Basic haben die Anwendungsentwicklung so einfach und komfortabel gemacht, dass heutzutage nahezu jedes Unternehmen Geschäftsanwendungen mit einer 4GL-Lösung erstellt.

Die Ziele von Rapid Application Development sind dieselben wie bei 4GL (4th-Generation-Language), unter anderem die Verringerung des Codieraufwands durch eine visuelle Entwicklungsumgebung und automatisches Generieren von Code, kürzere und dadurch besser lesbare und leichter wartbare Programme sowie die Reduzierung der Entwicklungskosten. Die meisten 4GL-Lösungen wie beispielsweise Oracle Forms sind mittlerweile in die Jahre gekommen und gelten bei vielen Anwendern als nicht mehr zeitgemäß. Verstaubte Oberflächen mit starrem Bildschirm-Layout, begrenzte Erweiterbarkeit, mangelhafte Interoperabilität, oft fehlende Web-Fähigkeit, Runtime-Lizenzgebühren, sinkende Akzeptanz bei Endanwendern und nicht zuletzt Unsicherheit in Bezug auf Support-Garantien durch den Hersteller zwingen Lösungsanbieter zur Suche nach zukunfts- und investitionssicheren Alternativen wie Java oder .NET.

Kritiker bemängeln ohnehin, dass mit „Drag&Drop"-Tools prototypisch, überstürzt und unstrukturiert vorgegangen und zudem überwiegend prozedural programmiert wird, was zu unsauberem Code und dadurch schwer wartbaren Anwendungen führt. Hinzu kommt, dass die Möglichkeiten entsprechender Tools meist sehr begrenzt sind. Notwendige Erweiterungen sind wegen der proprietären, in sich geschlossenen Systeme dagegen kaum möglich, sodass erfahrene Entwickler oft prophezeien, langjährig gewachsene Anwendungen müssten früher oder später mit großem Aufwand letztendlich doch auf eine „vernünftige" Technologie portiert werden, und daher raten, von Anfang an auf Java oder .NET zu setzen.

Java-Programmierung ist jedoch auch nach zwanzig Jahren immer noch genauso kompliziert, und der Einstieg vor allem für Anwender mit 4GL-Erfahrung oft ein

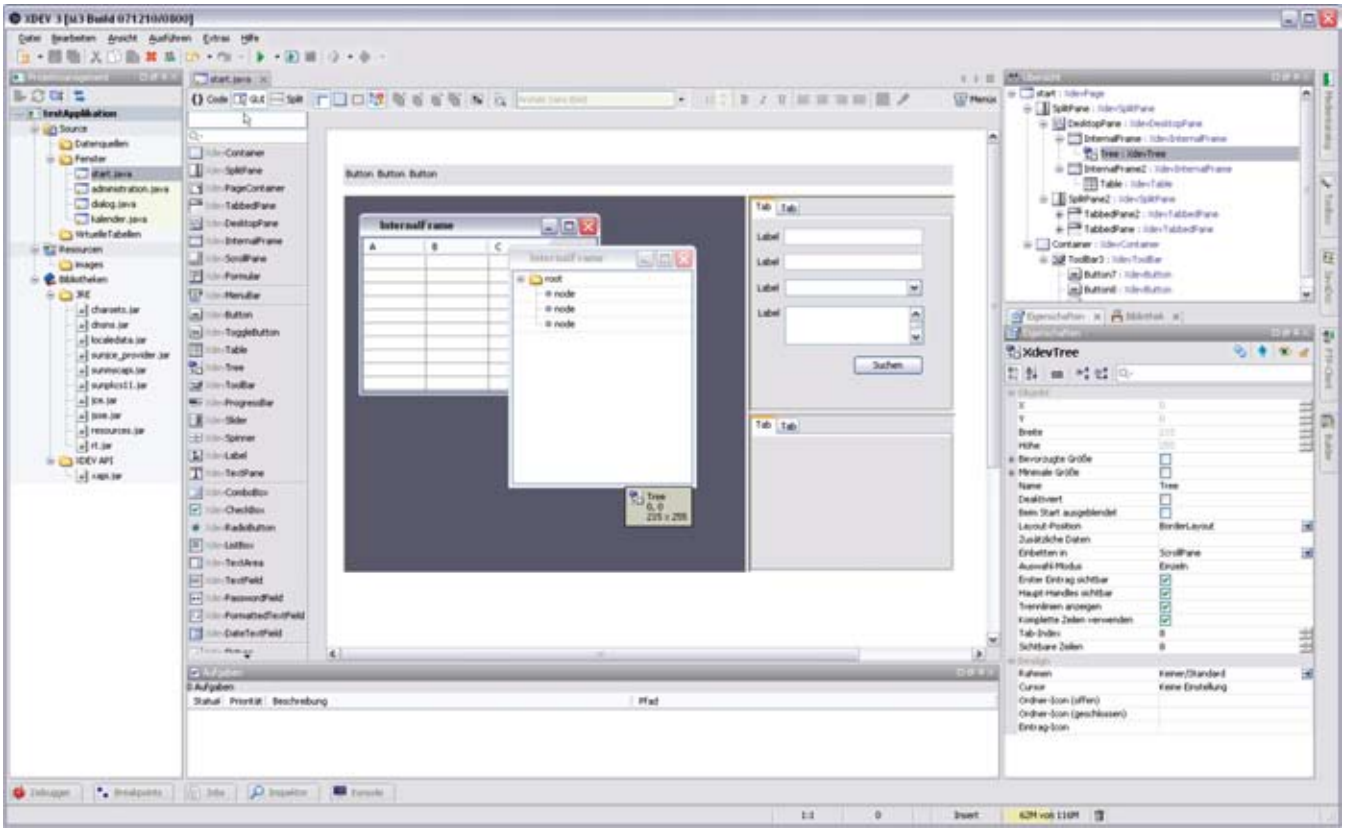


Abbildung 1: XDEV 3 bietet den einfachsten GUI-Builder im gesamten Java Umfeld. Jede denkbare Oberfläche sowie aufwändige Formulare lassen sich schnell und einfach wie mit einem Grafikprogramm umsetzen. Mit Hilfe von GUI-Beans lässt sich der Standardumfang von XDEV 3 erstaunlich leicht mit eigener Funktionalität erweitern.

qualvolles Erlebnis. Schon die Syntax von Java mit seinen zahlreichen Klammern und unbekannten Kontroll- und Datenstrukturen wie Try-Catch, Hashtables etc. ist für viele Umsteiger ungewohnt. Richtig zu schaffen macht auch das zeitaufwändige Suchen nach benötigten Methoden in der riesigen Java-API, die zudem nur in englischer Sprache dokumentiert ist. Die wohl größte Problematik besteht jedoch bei der Umstellung von prozeduraler auf objektorientierte Programmierung (OOP). Dies verlangt vom Entwickler ein vollständiges Umdenken und Umstellen der oft langjährig gewohnten und aus Entwicklersicht bewährten Programmierweise.

Nach mehreren Monaten Kampf mit Java müssen viele Umsteiger feststellen, dass sie gerade einmal die Grundlagen erlernt haben und dies noch lange nicht ausreicht, um Anwendungen für den Unternehmenseinsatz entwickeln zu können, die man mit der alten Lösung noch aus dem Ärmel geschüttelt hätte. Denn für die Entwicklung unternehmenskritischer Geschäftsanwendungen sollte man nicht nur objektorientierte Programmierung, GUI-, Datenbank- und Input/Output-Programmierung beherrschen, sondern möglichst auch eine Java-Enterprise-Technologie wie Servlets, Spring oder Hibernate. Voraussetzung dafür ist wiederum, dass der Entwickler alle wichtigen Java-Konzepte wie Interfaces, Reflections, Generics, Annotations, sämtliche Patterns etc. beherrscht und anwenden kann. Falls die benötigte Lösung zudem als Web-Anwendung umgesetzt werden soll, ist auch noch umfassendes Wissen über Web-Technologien wie JSP, JSF oder andere Java-Frameworks notwendig, wobei HTML-, CSS- und JavaScript-Know-how stets vorausgesetzt wird. Kein Wunder, dass manche Java-Einsteiger sich schon nach kurzer Zeit ihre alte Lösung zurückwünschen und aufgeben.

Für alle, die einen Ausweg aus diesem Dilemma suchen, gibt es mit XDEV 3 eine professionelle Entwicklungsumgebung, welche die Vorteile von Rapid Application Development und konventioneller Java-Programmierung vereint und auch während der Entwicklung einen fließenden Übergang ermöglicht, sodass zum einen Java-Einsteiger auf Anhieb zurechtkommen und zum anderen auch erfahrene Java-Entwickler eine gewohnte Umgebung vorfinden und sich damit wohl fühlen.

Dafür bietet die IDE ein umfassendes RAD-Konzept mit allen wichtigen Komponenten, die man sich für die Entwicklung von Datenbank-Anwendungen wünscht, darunter einen GUI-Builder auf Basis von Java Swing, ER-Diagramm-Tool, Data Bindings, einen komfortablen Query-Assistenten, ein Application-Framework mit Konzepten für Datenbank-Zugriffe und Datenverarbeitung inklusive einer für Datenbank-Programmierung optimierten Funktionsbibliothek, eine vorinstallierte Testumgebung sowie ein vollautomatisiertes Deployment für die Erstellung von Builds. Damit entlastet die Entwicklungsumgebung den Entwickler von aufwändiger Infrastruktur-, GUI-, Input/Output-, Datenbank- und Web-Programmierung, bei Lokalisierungen, bei Anwendungstests sowie beim Deployment und reduziert den Entwicklungsaufwand insgesamt - um bis zu 90 Prozent - enorm. Zudem ist für die aufgeführten Arbeiten kein Java-Know-how mehr nötig, was die Einstiegshürde deutlich senkt. Erfreulich ist, dass XDEV 3 den Entwickler nicht mehr zum Einsatz seiner RAD-Features zwingt. Denn unter der RAD-Haube steckt ein konventioneller Java-Code-Editor, der nicht nur von der Funktionalität her, sondern auch durch sein Aussehen etwas an Eclipse erinnert. Das Austauschen von Code, APIs und Beans mit anderen Java-IDEs wie Eclipse geht damit reibungslos von der Hand. Nicht nur Projekte, sondern auch die Entwicklungsumgebung selbst lässt sich dadurch bei

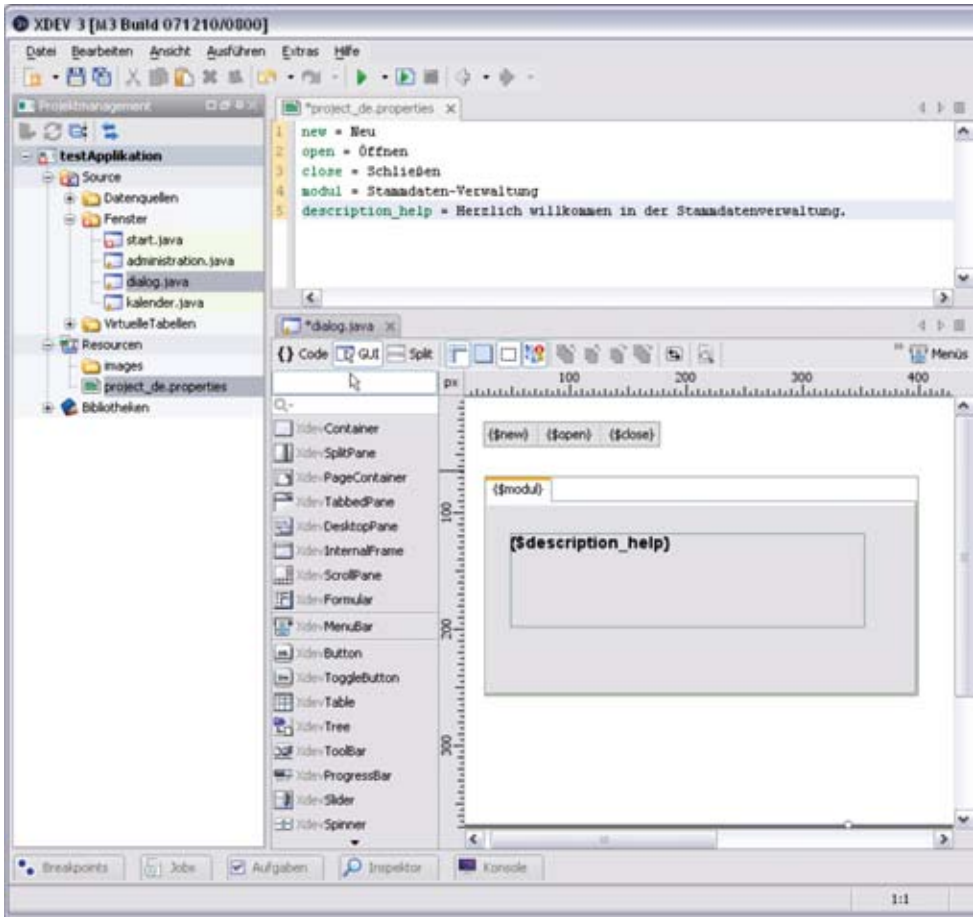


Abbildung 2: Mehrsprachige Oberflächen lassen sich mit Hilfe von Text-Variablen umsetzen, die sich an allen Stellen der GUI verwenden lassen, auch für Spaltennamen von Tabellen sowie für Hinweistexte und andere Strings. Die Variablenwerte werden automatisch aus einer Sprachdatei ausgelesen. Dabei handelt es sich um eine ganz normale Textdatei, die als Ressource an die Anwendung angebunden wird. Über die Systemeinstellungen wird immer die richtige Sprache ermittelt. Manuelles Umschalten ist ebenfalls problemlos möglich.

Bedarf flexibel erweitern, sodass sich hier grundsätzlich alles umsetzen lässt, was Java kann.

Aus Sicht des Herstellers stellt die Entwicklungsumgebung das längst überfällige Zwischenstück zwischen 4GL-Tools, wie Access und Oracle Forms, und Profiwerkzeugen wie Eclipse dar. Die XDEV-3-IDE ist völlig lizenzkostenfrei und für Windows, Linux und Mac zum freien Download unter www.xdev-software.de verfügbar. Das XDEV-Application-Framework ist Open Source.

GUI-Builder

Das auffälligste Feature der Entwicklungsumgebung ist der WYSIWYG- GUI-Builder zur Erstellung grafischer Oberflächen. Die Basis ist Java Swing, die neben AWT, SWT und JavaFX am häufigsten verwendete Grafikbibliothek in Java. Damit stehen dem Designer alle wichtigen Controls zur Verfügung, die man für den Entwurf moderner Oberflächen braucht, wie Buttons, Formular-Eingabe- und Auswahlfelder, Registerreiter, Tabellen, Trees sowie Drag&Drop, Fenstertechnik, Menüs und Kontextmenüs. XDEV 3 verwendet allerdings nicht die Standardkomponenten von Java Swing (wie JButton), sondern davon abgeleitete GUI-Beans (wie XdevButton), die mit zusätzlichen Eigenschaften angereichert wurden, etwa für die einfache Umsetzung mehrsprachiger Oberflächen. Die Vorgehensweise beim GUI-Design ist fast genauso wie bei Visual Basic. Alle Controls lassen sich mit Drag&Drop in die Arbeitsfläche einfügen, dort frei skalieren und pixelgenau positionieren. Ein Raster sowie Hilfslinien ermöglichen ein schnelles und exaktes Ausrichten. Auch ein Verschachteln der Elemente ist möglich. Auf diese Weise lässt sich jede erdenkliche Oberfläche oder Eingabemaske verblüffend schnell und kompromisslos umsetzen. Aufzeichnen auf Papier geht kaum schneller.

Damit sind auch Software-Architekten, Designer, Projektleiter oder Mitarbeiter von Fachabteilungen in der Lage, die Umsetzung der kompletten Oberfläche zu übernehmen.

Bereits während der Arbeit am GUI-Builder entsteht der entsprechende Java-Code in Echtzeit im Code-Editor. Auch Event-Handler, die für die Verarbeitung von Ereignissen notwendig sind, werden automatisch erzeugt. Der generierte Code ist sauber und strukturiert und lässt sich vom von Hand geschriebenen GUI-Code praktisch nicht unterscheiden. Sehr positiv fällt auf, dass nachträgliche Änderungen am generierten Code vom GUI-Builder automatisch geparst und entsprechend umgesetzt werden, sodass die GUI-Entwicklung bidirektional möglich ist. Sogar mit Eclipse geschriebener GUI-Code lässt sich übernehmen und kann vom Parser verarbeitet werden.

Anders als mit älteren 4GL-Tools ist mit XDEV 3 auch die Erstellung skalierfähiger Fenster und Dialoge möglich. Dafür setzt die Entwicklungsumgebung auf die von Swing zur Verfügung gestellten Layout-Manager. Mithilfe eines Grid-Layout-Assistenten las-

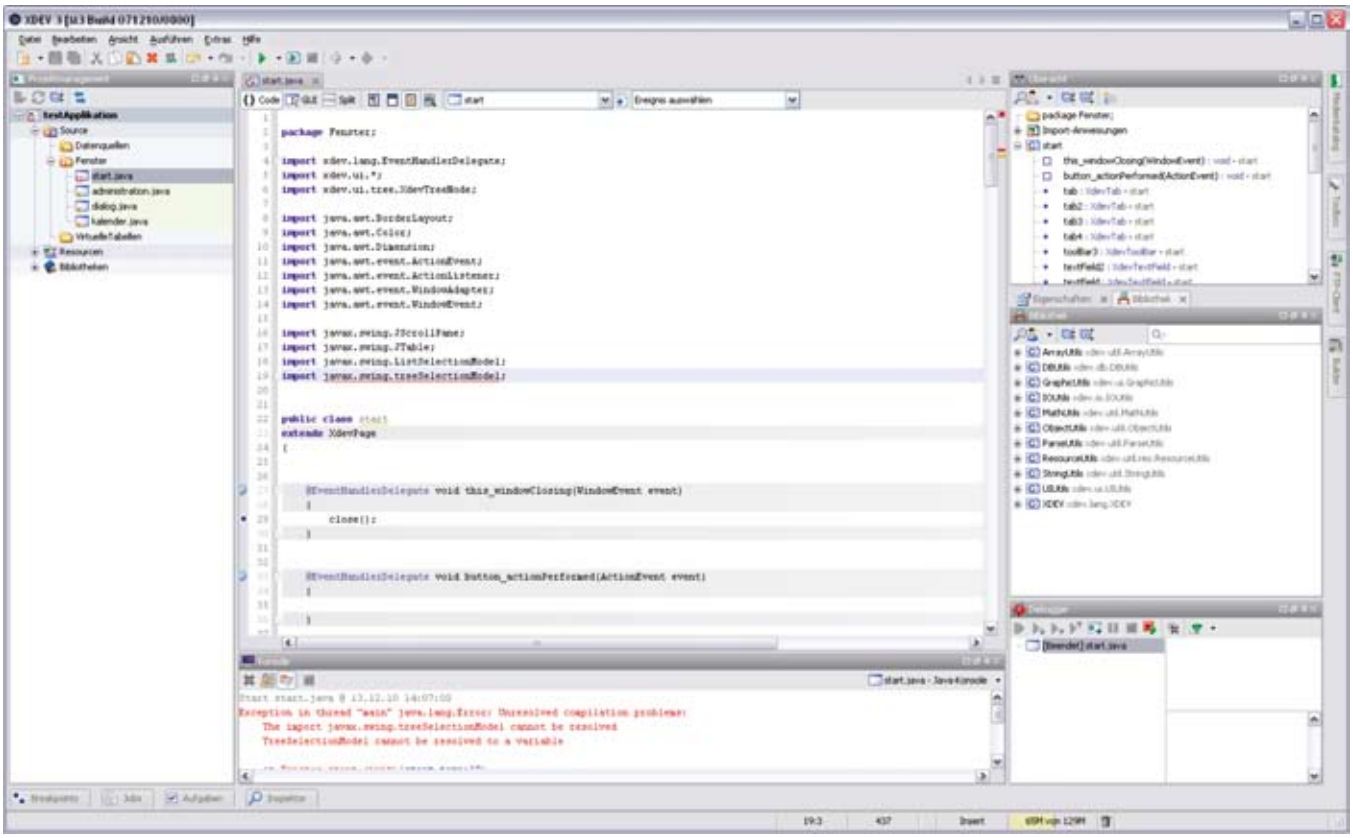


Abbildung 3: XDEV 3 bietet jetzt einen konventionellen Java Code-Editor, der den Vergleich mit Eclipse nicht scheuen muss. Autovervollständigung, Method-Finder, Drag&Drop Funktionalität und Partial-View machen das Codieren für Java Einsteiger viel einfacher als andere Java Editoren. Erfreulicherweise lässt sich generierter Code händisch ändern und erweitern. Der Austausch von Code von und nach Eclipse oder Netbeans klappt reibungslos, sodass praktisch keine Abhängigkeit von XDEV 3 mehr besteht.

sen sich selbst aufwändigste Formulare einfach und schnell layouten, ohne dass man sich mit der komplizierten und vor allem für Einsteiger komplizierten Gridbag-Layout-Programmierung herumplagen muss.

Mit der Verwendung von Textvariablen bei der Beschriftung der Oberfläche lassen sich auch mehrsprachige Oberflächen sehr leicht umsetzen. Die Texte selbst werden in externe Sprachressourcen-Dateien ausgelagert. Selbst wer kein Anhänger von generiertem Code ist, sollte - angesichts der enormen Zeitersparnis und der möglichen Arbeitsteilung - die Entwicklungsumgebung zumindest als GUI-Builder für Swing in seine Überlegungen mit einbeziehen. Denn ein damit generiertes Swing-User-Interface kann problemlos mit jeder beliebigen Zwischenschicht zu einer n-Tier-Architektur verbunden werden, etwa mit Hibernate, EJB, Spring oder mit Businesslogik direkt in einer Datenbank.

Mehr Unabhängigkeit von der Datenbank

Trotz vorhandener JDBC-Treiber stellt XDEV 3 eigene Datenbank-Schnittstellen für alle wichtigen Datenbanken zur Verfügung, die auf JDBC aufsetzen. Damit werden zum Teil erhebliche Unterschiede bei der SQL-Syntax zwischen den unterstützten Datenbanken über den JDBC-Standard hinaus ausgeglichen, sodass man von einer JDBC-Erweiterung sprechen kann. Mit dem Einsatz der mitgelieferten Schnittstellen lässt sich demnach eine gewisse Datenbank-Unabhängigkeit erreichen. Im Falle eines Datenbank-Wechsels müssen lediglich die Schnittstelle ausgewechselt und das Programm neu kompiliert werden, während der Code für sämtliche Abfragen unangetastet bleibt. Das ist nicht nur sehr praktisch, sondern schließt gleichzeitig eine erhebliche Fehlerquelle aus.

Zur Entwicklung und Verwaltung von Datenmodellen steht ein ER-Diagramm-Editor zur Verfügung. Auch der Import und Export von Relationen ist damit möglich. Aufgrund der hinterlegten Relationen lassen sich sogar komplexe Datenbankabfragen auf einfache Weise generieren.

Eines der Highlights ist der Query-Assistent. Selbst aufwändigste Datenbank-Abfragen über mehrere Tabellen sowie Filterbedingungen lassen sich damit bequem mit der Maus zusammenklicken, ohne dass man dafür eine Zeile SQL-Code schreiben muss. Die Vorgehensweise ist simpel. Die im ER-Diagramm verknüpften Tabellen werden mithilfe einer TreeTable wie eine Checkliste abgebildet, sodass man nur noch systematisch alle Datenfelder anklicken muss, die man selektieren möchte. Notwendige Join-Bedingungen werden automatisch erzeugt. Auch Filter-Bedingungen (Where-Conditions) lassen sich in Sekundenschnelle zusammenklicken. Damit auch bei verknüpften Filter-Bedingungen die Übersicht erhalten bleibt, wird jede Bedingung in einer eigenen Zeile angegeben. An Stelle von Klammern werden die entsprechenden Bedingungen eingerückt. Eine innovative Lösung, mit der selbst komplexeste Abfrage-Bedingungen übersichtlich bleiben. Auch das Suchen nach einem Datum oder in Zeiträumen sowie die Angabe von Limits zur Begrenzung der Ergebnismenge sind sehr einfach umsetzbar.

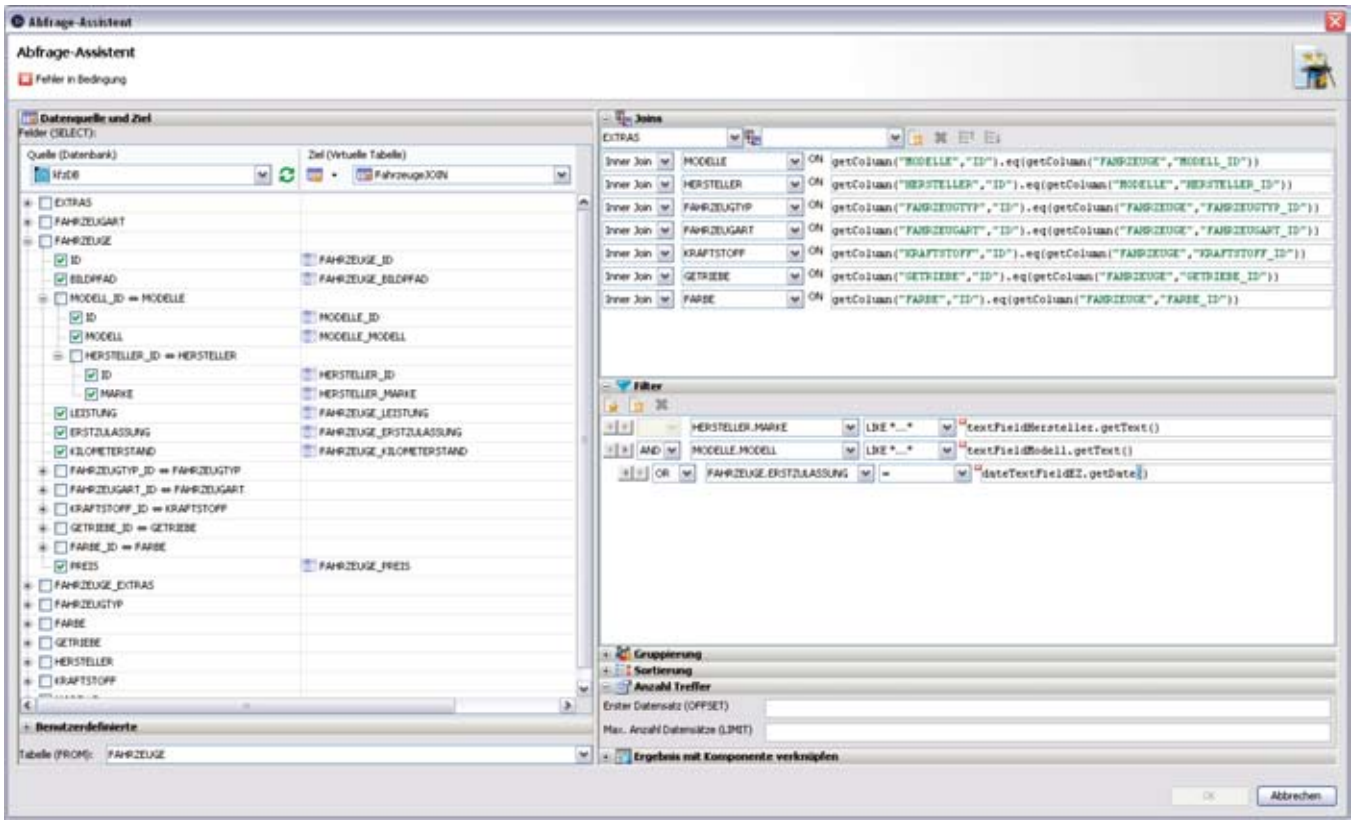


Abbildung 4: Mit Hilfe des Query-Assistenten lassen sich selbst aufwändigste Abfragen zusammen klicken. Dabei werden Joins und Filterbedingungen werden automatisch generiert. SQL-Kenntnisse braucht man dazu nicht. Der generierte Abfragecode lässt sich auch im Code-Editor ändern und erweitern.

Mit dem Schließen des Query-Assistenten generiert die Entwicklungsumgebung zuerst einmal Datenbank-unabhängigen Java-Code. Erst zur Laufzeit erzeugt eine spezielle SQL-Engine daraus den SQL-Code für die jeweils angeschlossene Datenbank. Änderungen und Erweiterungen, die am generierten Abfrage-Code vorgenommen werden, sind automatisch geparst vom Query-Assistenten umgesetzt, sodass die Erstellung und Bearbeitung von Queries grundsätzlich bidirektional möglich ist. Wer seine Abfragen lieber selber formulieren möchte, kann mithilfe einer Methode natürlich auch „plain" SQL-Strings absetzen oder dynamisch erzeugen lassen.

Bei XDEV 3 setzt sich das RAD-Konzept auch beim Code-Editor fort. So lassen sich sämtliche Anweisungen nicht nur über die Tastatur eingeben, sondern auch über eine Command-Toolbar bequem per Drag&Drop einfügen. Beim Einfügen einer Anweisung wird automatisch die komplette Kontrollstruktur inklusive Klammern und Semikolon erzeugt, sodass man kaum mehr Syntaxfehler machen kann. Markierungen zeigen zudem an, wo Parameter anzugeben sind. Alle Kontrollstrukturen sind als Code-Template hinterlegt, das sich zudem individuell anpassen lässt. Vor allem für Einsteiger, die mit der Java-Syntax noch nicht so vertraut sind, ist das eine wertvolle Hilfe. Auch das Anlegen eigener Code-Templates wird unterstützt.

Weniger Code für bessere Übersicht

Der Code-Editor befindet sich standardmäßig in einem sogenannten „Partial-Code-View-Modus". Damit sieht der Entwickler nur den Aktionscode, den er für ein bestimmtes Ereignis selber schreibt, etwa den Code für ein Mausklick-Ereignis eines Formular-Buttons. 95 Prozent des restlichen, automatisch generierten Codes, für den sich RAD-Entwickler ohnehin kaum interessieren werden, insbesondere Imports, Konstruktoren etc., bleibt unsichtbar. Der Programmcode wird dadurch sehr viel leichter lesbar, was vor allem für Java-Einsteiger eine erhebliche Erleichterung darstellt. Erfahrene Entwickler können den Partial-Code-View über eine Einstellung abschalten, wodurch der generierte Source-Code vollständig sichtbar wird.

Um die Suche nach Methoden möglichst kurz zu gestalten, bietet der Code-Editor eine spezielle Split-View. Ein kurzer Klick auf eine GUI-Komponente genügt und die Bibliothek zeigt alle Methoden zum angewählten Objekt inklusive Dokumentation dazu an. Die benötigte Methode lässt sich anschließend per Drag&Drop in den Code-Editor einfügen. Ein Assistent hilft dabei sogar bei der Übergabe der richtigen Parameter.

Test-Umgebung und Deployment

Bereits bei der Installation richtet XDEV 3 automatisch eine Testumgebung ein, welche auch das Testen von Java-Applets ermöglicht. Für das Erstellen von Builds gibt es ein vollautomatisiertes Deployment auf Basis von Apache Ant. Damit lassen sich Projekte wahlweise als Java-Applikation, Java-Applet oder als Webstart-Applikation deployen. Die so installierten Applikationen sind lauffähig unter Windows und Linux sowie auf dem Mac und lassen sich wahlweise als Fat-Client- (Applikation mit Embedded-Datenbank wie HSQLDB) oder auch als Client-Server-Applikation betreiben. Auch die Java-Runtime lässt sich beim Deployen gleich mit anbinden. Für Windows wird zudem eine EXE-Datei erzeugt, welche jedoch lediglich die ausführbare JAR-Datei anstößt. Wer seine Anwendung mit einem Installations-Setup ausliefern möchte, benötigt eine spezielle Java-Installations-Software. Um die Kosten zu sparen, kann man seine Applikation mithilfe von Java-Webstart verfügbar machen, womit gleichzeitig interessante Features wie automatische Updates und User-Autorisierung möglich sind.

Entgegen aller Vorurteile, die Java-Applets zu Unrecht anhaften, sind die von XDEV 3 deployten Applets weder langsam noch unsicher, denn sie sind in viele Einzeldateien aufgesplittet, sodass beim Aufruf der Anwendung immer nur das Startfenster geladen wird. Alle weiteren Fenster und Dialoge werden „on Demand" nachgeladen, sodass das Java-Applet wie eine Flash-Animation funktioniert. Dadurch bleibt das Applet, unabhängig von seiner Gesamtgröße, stets performant. Da Java-Applets immer in einer Sandbox laufen, ist Sicherheit ohnehin stets gewährleistet. Für das Betreiben von Java-Applets wird ein Application-Server wie Jetty oder Tomcat benötigt, während auf dem Client in jedem Fall eine Java-Runtime vorhanden sein muss. Der Hersteller möchte das Deployment in Zukunft sogar noch einfacher machen und das Deployen fertiger Anwendungen direkt in eine Cloud-Infrastruktur ermöglichen, sodass sich Entwickler nicht mehr mit der Einrichtung der Zielumgebung auseinandersetzen müssen. Konkrete Infos sollen in Kürze auf der Webseite bekannt gegeben werden.

Fazit

Wer möglichst schnell, einfach und kostengünstig auf Java umsteigen und dabei schnell produktiv in Java entwickeln möchte, erhält mit XDEV 3 das perfekte Tool. Es ist die erste Entwicklungsumgebung überhaupt, die den Spagat zwischen Rapid Application Development und konventioneller Java-Programmierung schafft und somit sowohl RAD- als auch erfahrene Java-Entwickler zufriedenstellt. Durch die Ähnlichkeit mit Access, Oracle Forms und Visual Basic gelingt der Einstieg auf Anhieb. Die schnellen Ergebnisse motivieren jeden Einsteiger und das Arbeiten macht richtig Spaß. Für den Einsatz in Teams muss der Hersteller mit einer Versionsverwaltung noch nachbessern. Eine Integration für Apache Subversion wurde jedoch bereits angekündigt. Unter der Haube sitzt ein konventioneller Java-Editor, der das Editieren des generierten Codes und somit bidirektionales Arbeiten ermöglicht. Wer an die Grenzen des RAD-Konzepts stößt oder entsprechende Features bewusst umgehen möchte, kann seine Arbeit nahtlos in Java fortsetzen und somit alle Freiheiten und Möglichkeiten genießen, die Java grundsätzlich bietet. XDEV 3 ist für jeden Einsatzzweck völlig lizenzkostenfrei.

Kontakt:
Gerald Kammerer
info@redaktion-kammerer.de

Gerald Kammerer ist als freier Redakteur seit 2006 für verschiedene Computerfachzeitschriften mit Schwerpunkt Java- und AJAX-Entwicklung tätig (siehe auch Seite 34).

Fragile Agile

gelesen von Andreas Badelt

Im September 2010 erschien im Carl Hanser Verlag das Buch „Fragile Agile" von Pavlo Baron und Michael Hüttermann. Die beiden Autoren sind bereits durch diverse Bücher und Zeitschriftenartikel insbesondere zu Java, IT-Architektur und (agilen) Entwicklungsmethoden einschlägig bekannt. In diesem Werk widmen sie sich den Wurzeln der agilen Software-Entwicklung, wie sie im „Agilen Manifest" (http://agilemanifesto.org) definiert wurden.

Dieses „Agile Manifesto", wie es im Original heisst, wurde 2001 von 17 renommierten Verfechtern agiler Software-Entwicklung gemeinsam verfasst, und hat großen Einfluss auf die Entwicklung und Verbreitung entsprechender Methoden und Vorgehensmodelle genommen. Wie üblich hat aber auch dieses Manifest viel Spielraum für Interpretationen gelassen, und selbst wo keiner mehr war, wurde der Begriff „agil" immer noch benutzt, wissentlich oder unwissentlich - wie sehr, zeigt eindrucksvoll das „Half-Arsed Agile Manifesto" eines frustrierten Entwicklers (http://www.halfarsedagilemanifesto.org). Diesen falschen Interpretationen entgegen zu treten, haben sich die beiden Autoren auf die Fahne geschrieben.

Das Buch ist konsequenterweise anhand der vier Wertepaare des Agilen Manifests gegliedert, darunter „Individuen und Interaktionen sind wichtiger als Prozesse und Werkzeuge", beziehungsweise anhand der daran ausgerichteten zwölf Prinzipien (zum Beispiel „Die besten Architekturen, Anforderungen und Designs ergeben sich aus selbst-organisierenden Teams"). Jedem dieser Prinzipien ist ein eigenes Kapitel gewidmet. Darin wird zunächst das Prinzip und seine Bedeutung erklärt. Daran schließt sich eine Beschreibung von Vorgehensweisen oder Situationen an, in denen dieses Prinzip verletzt wird. Diese „Fehlinterpretationen" werden dabei von den Autoren anhand des aristotelischen Gedankens vom „rechten Maß" aufgebaut: Eine „Fehlinterpretation" ist immer diejenige, in der das Prinzip gar nicht befolgt wird, in der anderen Interpretation wird es so überzogen angewendet, dass es ad absurdum geführt wird. Dabei wird immer wieder mit kleinen Beispielen ein Bezug zu realen oder gut ausgedachten Projekten hergestellt und es wird aufgezeigt, welche teilweise skurrilen Fehler in sogenannten „agilen" Projekten gemacht werden.

Neben den Projekt-Beispielen arbeiten Pavlo Baron und Michael Hüttermann auch viel mit kleinen Anekdoten und Bildanalogien, sowohl um ihre Beschreibungen zu unterstützen, als auch um den Text ein wenig aufzulockern. Hinzu kommt, dass sie eine eher lockere und lebendige „Schreibe" haben und viele flapsige Formulierungen verwenden. Das Buch ist dadurch sehr gut zu lesen.

Einige Kritikpunkte sind: Die gerade erwähnten Stilmittel werden vielleicht etwas übertrieben genutzt (nicht im „rechten Maß", um im oben bemühten Bild zu bleiben). Insgesamt hätte das Buch sicher um einige Seiten kürzer ausfallen können, ohne an Informationsgehalt zu verlieren. Konkret auf die Beispiele „aus dem wahren Leben" bezogen: Diese sind manchmal zu schwarz-weiss, auch wenn den Autoren in ihrer langjährigen Berufserfahrung sicher schon einiges widerfahren ist. Und gerade für „Agile-Anfänger" wären ein paar Analogien sinnvoll, die weniger die Wertepaare und Prinzipien des „Agile Manifesto" erklären, sondern den Wert agiler Projekte an sich: Warum kann es Sinn machen, mich jetzt noch nicht damit zu beschäftigen, was in drei Monaten implementiert werden muss? Aus dem gleichen Grund, und weil das Buch sich mit den Grundwerten agiler Softwareentwicklung beschäftigt, wäre auch eine gründlichere Einbettung in den Kontext sicher sinnvoll gewesen: Wo kommt „Agile" eigentlich her, in welchem historischen Kontext ist es entstanden?

Fazit

Das Buch richtet sich grundsätzlich an alle Menschen, die sich für agile Software-Entwicklung interessieren, egal ob technisch oder fachlich orientiert, oder aus dem (Projekt-) Management. Das können „Agile-Anfänger" oder schon Fortgeschrittene sein, die mehr über die grundlegenden Prinzipien ihrer täglichen Arbeit erfahren wollen. Diese Gruppe ist sicher größer als man auf den ersten Blick vermuten würde. Auch ein erfahrener Scrum-Master, um ein Beispiel zu nennen, ist nicht unbedingt mit allen Aspekten von „agile" vertraut – aber es wird ihm sicherlich in seinen Projekten helfen. Aber auch für „Agile-Kenner", die sich ein paar neue Denkanstöße holen wollen, kann das Buch interessant sein.

Das Konzept ist gut und einfach: Die Autoren wollen den Leser zum Nachdenken über den Kern agiler Software-Rntwicklung anregen und mit vielen Missverständnissen rund um das Wort „Agile" aufräumen – dabei arbeiten sie entlang der erwähnten Prinzipien. Und das machen sie auf jeden Fall auf sehr unterhaltsame und anregende Weise.

Kontakt:
Andreas Badelt
andreas.badelt@cgi.com

Andreas Badelt ist Berater und Softwareentwickler / -Architekt bei der CGI Information Systems and Management Consultants (Deutschland) GmbH. Sein Schwerpunkt ist die Realisierung von großen eCommerce-Systemen, insbesondere für die Telekommunikationsbranche. Daneben organisiert er seit 2001 die Special Interest Group (SIG) Development bzw. SIG Java der DOAG.

Code-Generierung

David Tanzer, Freiberufler

Code-Generierung, ob aus Domain Specific Languages (DSLs) oder grafischen Modellen (UML, BPMN), ist vor allem im Enterprise-Bereich seit Jahren ein Thema. Begriffe wie „Model Driven Software Development" (MDSD) oder „Model Driven Architecture" (MDA) sind bei Beratern und großen Unternehmen sehr beliebt. Dieses Essay stellt einen kritischen Standpunkt zu diesen Themen dar.

Ich werde manchmal gefragt, ob ich nun für oder gegen Code-Generierung bin. Manche denken, ich sei uneingeschränkt dafür, da wir in dem Web-Framework „JSXP" (http://jsxp.org) Code-Generierung einsetzen. Andere schätzen mich als totalen Gegner ein, da ich oft gegen Code-Generatoren argumentiere. Was ist also wirklich meine Meinung zu diesem Thema? Dazu muss ich leider vorerst die Standardantwort eines Beraters geben: „Es kommt darauf an."

Code-Generierung in JSXP

Das Web-Framework JSXP setzt sehr stark auf Code-Generierung. Jedes Mal, wenn eine View (typischerweise eine XHTML-Datei) verändert wird, ist eine Basisklasse für den Controller neu zu generieren. Man kann moderne IDEs so konfigurieren, dass beim Speichern eines X(HT)ML-Dokuments in einem bestimmten Ordner automatisch der Generator aufgerufen wird. Der Aufruf des Generators stört also den Arbeitsablauf nicht allzu sehr. Trotzdem besteht immer noch das Risiko, dass die XML-Datei und der generierte Code nicht zusammenpassen. Aus unterschiedlichen Gründen – beispielsweise eine falsche Konfiguration bei einem Mitarbeiter – ist es möglich, dass der Generator nach einer Änderung nicht ausgeführt wird.

Warum verwenden wir bei JSXP also einen Code-Generator? Einerseits wird dadurch ein technisches Problem gelöst, das anders äußerst schwer zu lösen wäre. Die Schwierigkeit, die viele Web-Frameworks haben, besteht darin, dass man über String-Konstanten Elemente des Designs mit dem Code verbindet. Bei Wicket muss man zum Beispiel die IDs (wicket:id) aus dem Design im Code referenzieren, bei JSF schreibt man Expression Language (EL). Manche Frameworks zwingen einen sogar dazu, Code und Design zu mischen (EL ist Code und hat eigentlich im Design nichts zu suchen). JSXP löst das Problem, Design mit Code zu verbinden, durch einen Generator. Für jedes benannte Element aus dem Design wird ein typsicherer Getter erzeugt, für jede Variable ein typsicherer Setter (siehe dazu http://jsxp.org/Element-Template.xhtml und http://jsxp.org/Hello-World_1_2.xhtml).

Andererseits werden durch den Generator zwei völlig unterschiedliche Aspekte einer Web-Anwendung miteinander verbunden: das User-Interface-Design in Form von XHTML und der Programmcode der Anwendung in Java. Dadurch kann man getrennt voneinander entweder am Design oder am Code arbeiten, und der Generator bildet eine Brücke. Der generierte Code hilft auch dabei, inkompatible Änderungen an einem der beiden Teile im anderen zu entdecken: Es ist sichergestellt, dass jedes im Code referenzierte Element im Design auch existiert. Bei Wicket oder JSF müsste man dafür Unit-Tests schreiben, da solche Fehler erst zur Laufzeit erkannt werden.

Dadurch war für uns der Einsatz eines Generators gerechtfertigt: Wir lösen ein technisches Problem und der erzeugte Code bildet eine Brücke zwischen zwei unterschiedlichen Aspekten der Anwendung.

Code-Generierung in MDSD

In MDSD verfolgt das Generieren von Code einen anderen Zweck: Hier wird aus einem Modell – typischerweise ein grafisches (UML) oder textuelles (DSL) – der Programmcode generiert. Da dieser auch ein abstraktes Modell der Realität ist, handelt es sich hierbei in Wirklichkeit um eine „Model-to-Model"-Transformation. Das heißt, eine Repräsentation eines Aspekts des Systems wird in eine andere Repräsentation desselben Aspekts überführt. Damit lösen wir kein technisches Problem, sondern ein organisatorisches: Es wird sichergestellt, dass die unterschiedlichen Darstellungen eines Aspekts konsistent zueinander sind. Das ist so ähnlich wie das, was der Compiler macht: Dieser übersetzt Code aus einer höheren Programmiersprache in Maschinencode. Daran ist doch nichts verkehrt, oder? Also, warum bin ich für Compiler, aber gegen MDSD?

Der Grund, aus dem wir Compiler verwenden, ist, dass niemand ein Projekt in Maschinencode entwickeln würde. Das wäre äußerst schwierig, fast schon unmöglich, und deshalb wird es nicht einmal versucht. Wenn wir also „Blub"-Programmcode (die hypothetische Programmiersprache aus „Beating the averages" von Paul Graham, siehe http://www.paulgraham.com/avg.html) generieren wollen, würde das doch bedeuten, dass wir nicht direkt Blub-Code entwickeln wollen. Vielleicht, weil Blub nicht mächtig genug ist (siehe http://paulgraham.com/power.html). Warum sollten wir dann aber Blub-Code generieren, anstatt gleich eine mächtigere Programmiersprache zu verwenden, deren Compiler direkt Maschinencode erzeugt?

Ich denke jedoch, dass Java (und andere „Mainstream"-Programmiersprachen) sehr gut für viele Projekte geeignet sind. Selbst bei MDSD oder MDA will man ja den Programmcode, der die Geschäftslogik abbildet, in Java oder C# schreiben. Deswegen muss der Grund für Code-Generatoren ein anderer sein als der, warum wir Compiler verwenden.

Die Begründung, die oft angegeben wird – ich habe das bereits erwähnt – ist, dass man das Modell und den Code konsistent halten will. Aber – auch das habe ich bereits bemerkt – der Code ist ein Modell. Und ich bin überzeugt davon, dass

der Code „das Modell“ sein sollte, also dass nicht mehrere Modelle existieren. Die Frage ist also: Warum brauchen wir unterschiedliche Modelle derselben Realität?

Ein Grund, der manchmal genannt wird: Grafische Darstellungen sind einfacher zu verstehen. Aber: Das stimmt nur für sehr kleine Systeme oder Subsysteme, da grafische Darstellungen sehr schlecht skalieren. Deswegen ist es nicht sinnvoll, ein großes System grafisch zu entwerfen und dann den Code zu generieren. Andererseits eignen sich grafische Darstellungen sehr gut, um kleine Teilbereiche des Systems zu dokumentieren. Dafür gibt es aber eine bessere Lösung, als den Programmcode aus der grafischen Darstellung zu generieren: Die grafische Darstellung wird aus dem Programmcode generiert! Doxygen (http://doxygen.org) funktioniert hier sehr gut, und es gibt auch Systeme, mit denen man solche Diagramme in Javadoc einbinden kann.

Eine andere, oft genannte Begründung lautet, dass man das System in der Sprache des Fachbereichs entwickeln will. Da grafische Darstellungen so schlecht skalieren, entwickelt man „Domain Specific Languages“, also eine textuelle Darstellung in Form einer formalen Sprache. Wäre das nicht schön, wenn man direkt in der Sprache der Domäne entwickeln könnte? Nein, wäre es nicht. Erstens ist die DSL noch immer eine formale Sprache, es wird also nicht möglich sein, dass die Endanwender das System direkt in der DSL beschreiben. Man braucht also weiterhin Programmierer, um das System zu entwickeln. Es werden sogar noch mehr Software-Entwickler benötigt, da man die DSL ja entwickeln und warten muss. DSLs sind auch per Definition Domänen-spezifisch, können also nicht so einfach für mehrere Projekte verwendet werden. Man wird auch weiterhin Code in Blub entwickeln müssen, also muss man auch die Anbindung an das in Blub entwickelte System berücksichtigen.

Aber es kommt noch schlimmer: Man kann DSLs oder grafische Darstellungen weder Unit-testen noch debuggen! Wenn also ein Problem auftaucht, muss man generierten Code debuggen. Wenn man das Problem dann endlich gefunden hat, kann man es nicht einfach beheben. Man muss zuerst überprüfen, ob das Problem im Modell oder im Generator begründet ist. Falls es im Modell begründet ist, ändert man dieses, generiert neuen Code, ändert gegebenenfalls die Unit-Tests, falls sie nicht mehr zum generierten Code passen, und kann dann endlich den Fehler neu testen. Falls das Problem im Generator begründet ist, wird es erst richtig kompliziert.

Fazit

Code-Generierung in MDSD ist Zeit- und Ressourcen-Verschwendung. Es bremst die Entwicklung, ohne dass es irgendjemandem einen wirklichen Vorteil bringen würde. MDSD ist zwar Management-tauglich, da es den falschen Eindruck von Kontrolle erweckt („bei uns ist alles genauestens durch Modelle dokumentiert“), aber davon sollte man sich nicht täuschen lassen. Auch DSLs lösen die Probleme von MDSD nicht zufriedenstellend. In gewissen Fällen halte ich Code-Generierung für das geringere Übel, aber auch in JSXP bin ich nicht zu 100 Prozent zufrieden damit. Es wäre viel schöner, wenn wir eine noch engere Verbindung zwischen Design und Code hätten, zum Beispiel: public class IndexXhtmlController extends „index.xhtml“.

Der Java-Compiler selbst (oder ein Plug-in) müsste die Verbindung zwischen „index.xhtml“ und der Java-Klasse herstellen. Das haben wir, als wir das Framework konzipierten, gar nicht in Betracht gezogen, da es technisch viel zu aufwändig (falls überhaupt möglich) wäre.

Um den Programm-Code mit den Anforderungen und der Sprache der Domäne synchron zu halten, ist „Domain Driven Design“ sehr gut geeignet. Hier ist das Buch von Eric Evans sehr empfehlenswert. UML-Diagramme halte ich durchaus für sinnvoll: Man sollte sie auf ein Whiteboard zeichnen und wieder löschen, sobald der Code das erfüllt, was das Diagramm aussagt.

Kontakt:
David Tanzer
david@davidtanzer.net

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

BPM-Lösungen mit Java EE selbst gebaut

Ralph Soika, Imixs

Das Imixs-Workflow-Projekt ist Open Source und erleichtert den Bau von Workflow-Management- (WfMS) sowie Business-Process-Management-Systemen (BPM). Wesentlicher Bestandteil sind die Imixs-JEE-Workflow-Komponenten, um BPM-Lösungen auf Basis der Java-EE-Plattform zu bauen.

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
apache.org/xsd/settings-1.0.0.xsd">
    <profiles>
        <profile>
            <id>default</id>
            <repositories>
                <!-- JBoss RichFaces Repository -->
                <repository>
                    <id>repository.jboss.com</id>
                    <name>Jboss Repository for
                    Maven</name>
                    <url>http://repository.jboss.com/maven2/</url>
                </repository>
            </repositories>
        </profile>
    </profiles>
    <activeProfiles>
        <activeProfile>default</activeProfile>
    </activeProfiles>
</settings>
```

Listing 1

Das folgende Tutorial zeigt, wie mithilfe von Eclipse, GlassFish und Maven eine Imixs-BPM-Lösung in wenigen Minuten erstellt werden kann. Für die Ausführung sind ein Applikationsserver, eine Datenbank und die Eclipse IDE notwendig. Als Applikationsserver kommt der GlassFish Server V3 zu Einsatz, auf dem die fertige Anwendung verteilt und ausgeführt werden kann. Die Daten sind in einer Datenbank gespeichert. Java EE stellt keine besonderen Anforderungen an eine Datenbank; im Open-Source-Umfeld fällt die Wahl meist auf MySQL.

Für die Ausführung der BPM-Lösung stellt der Applikationsserver zwei wesentliche Funktionen bereit. Zum einen speichert er die Prozessdaten in der Datenbank über einen sogenannten „Database-Pool", zum anderen stellt er einen Security-Realm bereit. Dieser sorgt dafür, dass User sich gegenüber der Anwendung authentifizieren können und unberechtigte Zugriffe unterbunden werden.

Die Imixs-Workflow-Lösung stellt umfangreiche Funktionen bereit, um einzelne Datensätze in Abhängigkeit des Prozessmodells vor unerlaubtem Zugriff zu schützen. Für den späteren Zugriff auf das System wird mindestens ein User als Mitglied der Gruppe „IMIXS-WORKFLOW-Manager" benötigt. Mitglieder dieser Gruppe sind auch berechtigt, Workflow-Modelle zu synchronisieren. Die Konfiguration des Database-Pools sowie des Security-Realms ist schnell erledigt. Eine vollständige Installationsanleitung für GlassFish und JBoss steht unter http://doc.imixs.org/jee/install_glassfish.html.

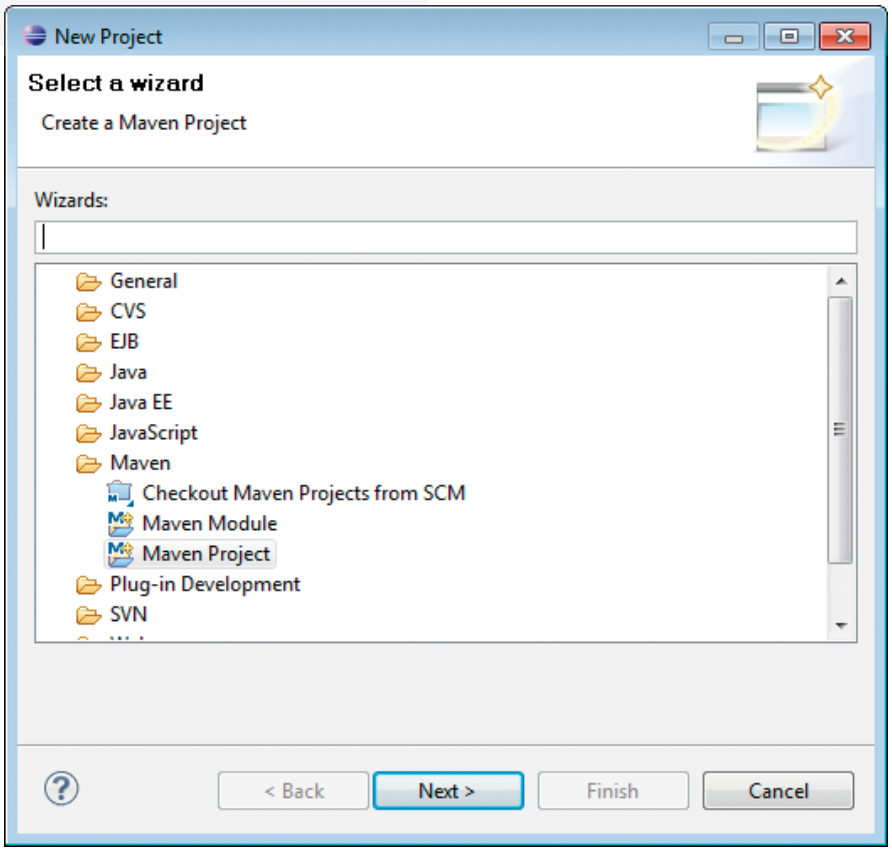


Abbildung 1: Ein neues Maven-Projekt erzeugen

Eclipse und Maven konfigurieren

Für die Arbeit mit Eclipse werden das Modeler-Plugin zur Modellierung des Prozessmodells sowie das Maven-Plugin von Sonatype für den Bau des JEE-Projekts benötigt (http://www.sonatype.com/products/m2eclipse). Beide lassen sich innerhalb von Eclipse bequem über den Menübefehl „Help –> install new Software“ aus folgenden URLs installieren:

- http://www.imixs.org/org.imixs.eclipse.updatesite
- http://m2eclipse.sonatype.org/sites/m2e

Die Web-Oberfläche der BPM-Lösung wird mithilfe von RichFaces realisiert. Dies ist eine Komponenten-Bibliothek für JavaServer Faces (JSF), mit der Ajax-basierte Web-Anwendungen erstellt werden können. Für die Verwendung von RichFaces in Maven ist das Jboss-Repository in die eigene Maven-Konfiguration aufzunehmen. Die einfachste Art, das RichFaces-Repository für Maven sichtbar zu machen, besteht darin, in der Datei „%USERPROFILE%\.m2\settings.xml“ ein entsprechendes Profil einzurichten. Die settings.xml-Datei findet sich in Windows-Systemen unter c:\Benutzer\USERNAME\.m2\. Existiert noch keine settings.xml-Datei, kann man diese einfach erstellen (siehe Listing 1).

Nachdem die Entwicklungsumgebung Eclipse und die Server-Infrastruktur stehen, kann der Bau der BPM-Lösung beginnen. Da das Workflow-Projekt vollständig auf Maven basiert, ist das Erstellen einer neuen Anwendung sehr einfach. Über die Auswahl eines sogenannten „Maven Archetypes“ kann ein Template verwendet werden, das bereits alle notwendigen Komponenten einer BPM-Lösung enthält. Das neue Maven-Projekt lässt sich in Eclipse über „File –> new Project....“ erzeugen (siehe Abbildung 1).

Über den Projekt-Typ „Maven –> Maven Project“ gelangt man in den Maven-Projekt-Wizard (siehe Abbildung 2). Dieser Wizard erlaubt die Erstellung eines neuen Projekts auf Basis eines Maven-Templates. Auf der zweiten Wizard-Seite „Select an Archetype“ sind alle bekannten Templates angezeigt.

Da das Imixs-JEE-Template bisher noch nicht verwendet wurde, muss es einmalig über den Button „Add Archetype“ hinzugefügt werden (siehe Abbildung 3). Für den Dialog „Add Archetype“ sind folgende Angaben notwendig:

- Archetype Group Id: org.imixs.application
- Archetype Artivact Id: imixs-workflow-jee-archetype
- Archetype Version: 0.0.4

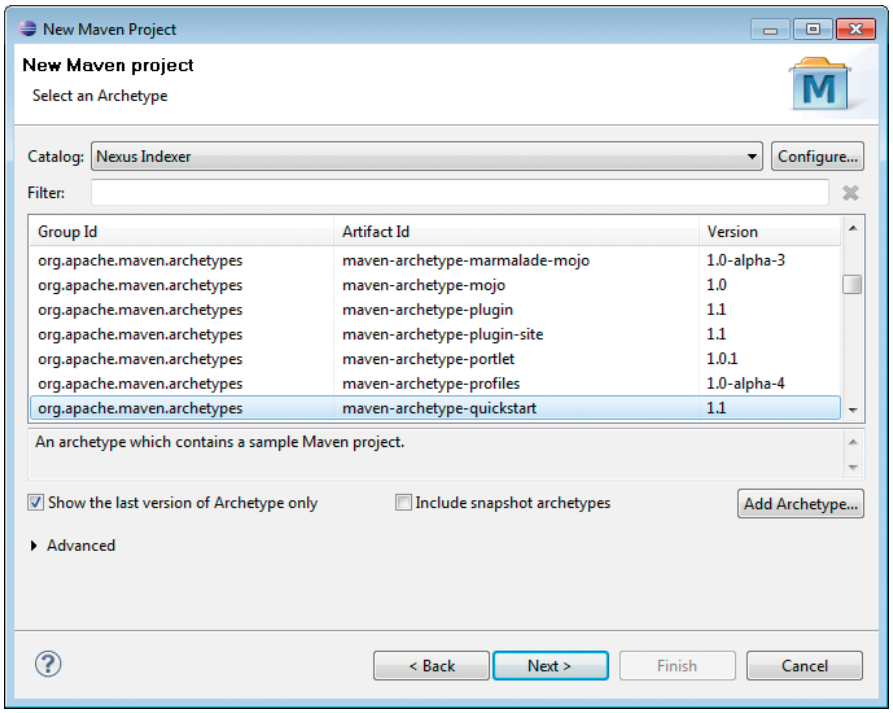


Abbildung 2: Auswahl eines Maven-Archetypes

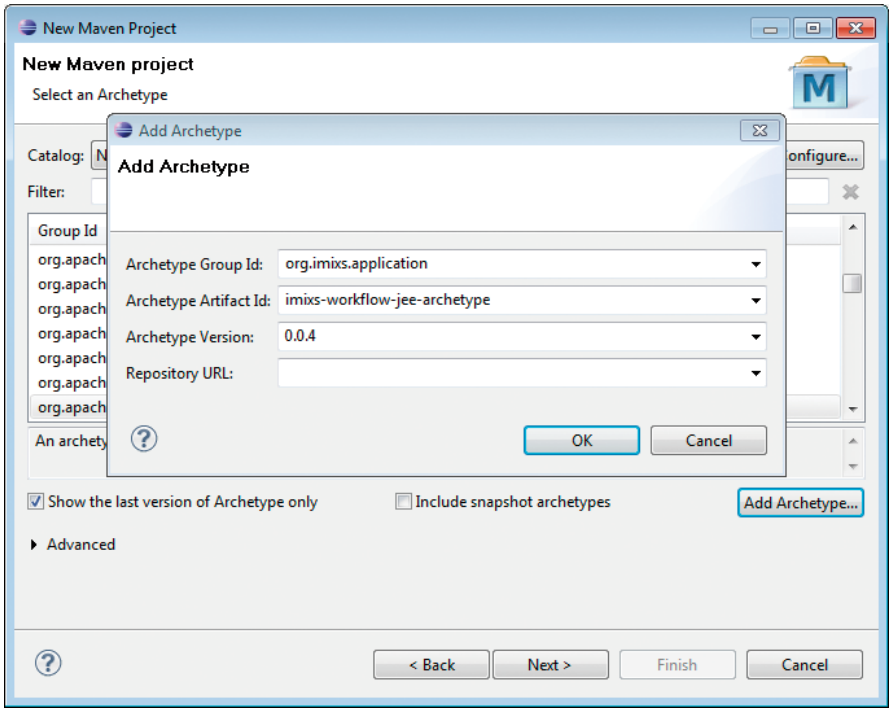


Abbildung 3: Einen Maven-Archetype hinzufügen

Maven führt daraufhin den Download des Templates durch, das anschließend ausgewählt werden kann. Auf der letzten Wizard-Seite gibt man nun noch verschiedene Parameter für das neue Projekt an (siehe Abbildung 4). Group Id und Artifact Id sind beliebig wählbar. Über die Archetype-Eigenschaften können die zuvor am

```
......
[INFO] Unnamed - com.foo:my-bpm-app-ejb:ejb:0.0.1-SNAPSHOT  SUCCESS [2.756s]
[INFO] Unnamed - com.foo:my-bpm-app-web:war:0.0.1-SNAPSHOT  SUCCESS [3.899s]
[INFO] Unnamed - com.foo:my-bpm-app-ear:ear:0.0.1-SNAPSHOT  SUCCESS [3.658s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 11.135s
[INFO] Finished at: Sat Jul 17 17:00:55 CEST 2010
[INFO] Final Memory: 3M/15M
[INFO] ------------------------------------------------------------------------
```

Listing 2

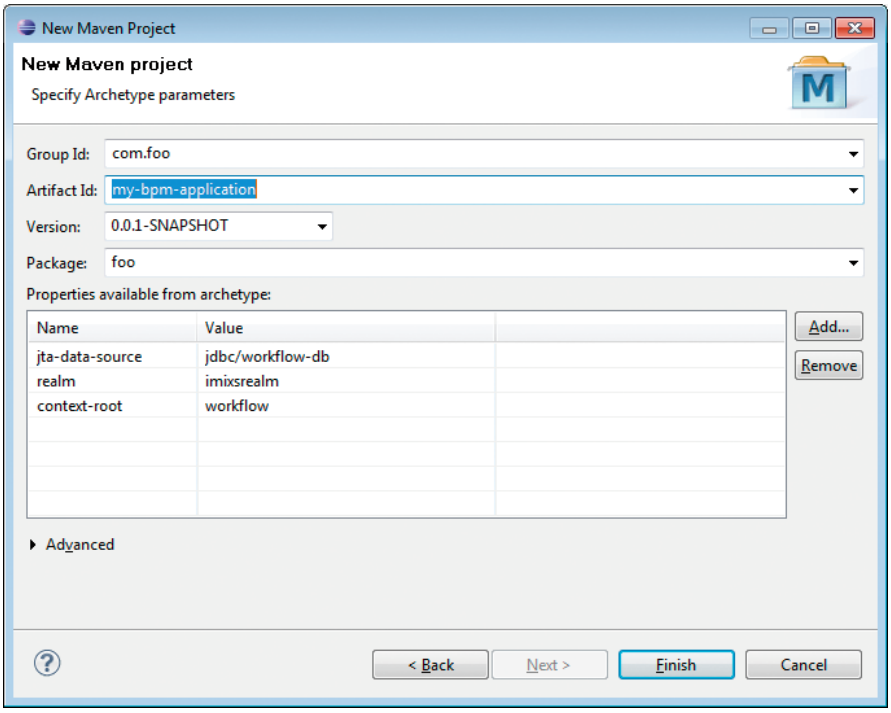


Abbildung 4: Maven-Archetype konfigurieren

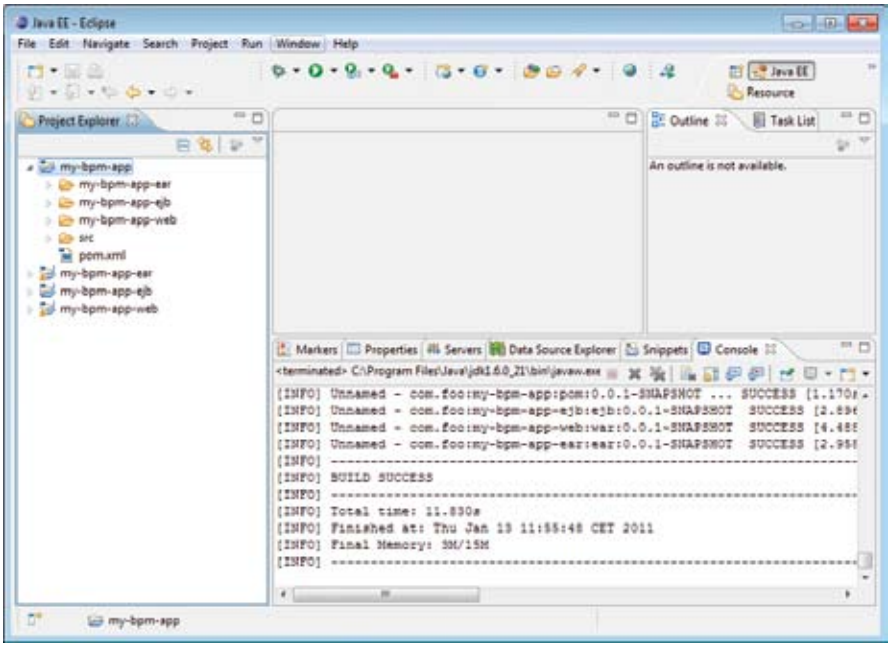


Abbildung 5: Die Projektstruktur

Server eingestellten Informationen über den Database-Pool und den Security-Realm vorgenommen werden. Insgesamt lassen sich für das BPM-Projekt drei Parameter einstellen:

- jta-data-source: Dieser Wert bestimmt den vom Server zu verwendenden Database Pool, in dem später die Prozessdaten gespeichert werden sollen
- realm: Der Realm ist der Server Security Realm, der zur Authentifizierung verwendet werden soll
- context-root: Der Context-Root bestimmt die Webadresse, unter der die Anwendung aufgerufen werden kann (etwa http://localhost:8080/workflow)

Die Schaltfläche „Finish" erzeugt nun das Projekt. Maven beginnt automatisch damit, sämtliche benötigten Bibliotheken aus dem Internet zu laden und prüft dabei auch alle notwendigen Abhängigkeiten der einzelnen Bibliotheken. Dies erleichtert die Arbeit sehr, da man keinerlei Java-Bibliotheken manuell herunterladen und installieren muss. So werden auch die RichFaces-Komponenten automatisch in das Projekt mit aufgenommen.

Anschließend erscheinen im Package Explorer insgesamt vier Projekte. Das erste (hier my-bpm-app) kann für den vollständigen Build und die Installation verwendet werden. Die Unterprojekte -ear, -ejb und -web enthalten jeweils das entsprechende Java-EE-Modul.

Die Anwendung builden, deployen und testen

Um nun die fertige Anwendung zu bauen, genügt es, im Hauptprojekt (my-bpm-app) über das Kontext-Menü den Befehl „Run as –> Maven Install" aufzurufen (siehe Abbildung 5). Daraufhin startet Maven automatisch den kompletten Build-Prozess und erstellt eine ausführbare EAR-Anwendung, die im GlassFish-Server installiert werden kann. Die Eclipse-Ansicht „Console" zeigt den Build-Status an. Der Build ist erfolgreich, sobald die Ausgabe „BUILD SUCCESS" erscheint (siehe Listing 2). Nun kann die Anwendung bereits auf dem Applikationsserver installiert werden. Das fertige EAR-Modul steht im Maven-Verzeichnis unter %USERPROFILE%\.m2\com\foo\my-

bp-app-ear\0.0.1-SNAPSHOT. Um es unter GlassFish zu deployen, wählt man über die GlassFish-Web-Konsole den Menüpunkt „Applications –> Deploy“ aus (siehe Abbildung 6). Danach wird das EAR-File aus dem .m2-Verzeichnis ausgewählt. Während des Deployments legt das Java-EE-Framework automatisch die notwendigen Tabellen-Strukturen in der angegebenen Datenbank an. Die manuelle Erzeugung eines speziellen Schemas ist hier also nicht notwendig.

Ein Modell erstellen

Bevor die Anwendung über den Web-Browser geöffnet wird, ist noch ein Workflow-Modell bereitzustellen. Die BPM-Anwendung in Eclipse besitzt bereits im Hauptprojekt unter /src/workflow/ticket.ixm ein Beispiel-Workflow-Modell. Die Datei ticket.ixm lässt sich mit dem Imixs-Modeler öffnen und bearbeiten. Über den Tab „Synchronize“ gelangt man zu den Web-Service-Settings, über die die Modelldaten per REST-Service mit der Workflow-Anwendung synchronisiert werden. Die URL für den Web-Service lautet http://localhost:8080/workflow-rest/model. Wurde der Context-Root angepasst, ändert sich diese Adresse entsprechend in /[ROOT_CONTEXT]-rest/model. Über den Link„synchronize model“ wird das Modell in die Workflow-Anwendung übertragen (siehe Abbildung 7). Hierzu ist eine Anmeldung mit dem zuvor eingerichteten User-Account aus der Zugriffsgruppe „IMIXS-WORKFLOW-Manager“ erforderlich. Anschließend kann man die Workflow-Anwendung auf http://localhost:8080/workflow über den Web-Browser starten. Über den Menübefehl „Meine Aufgaben –> neue Aufgabe –> Ticket“ wird dann ein erster Ticket-Prozess gestartet. Natürlich lässt sich anschließend das Modell über den Imixs-Modeler beliebig ändern beziehungsweise erweitern. Die laufende Anwendung muss dazu nicht mehr neu gestartet oder deployed werden.

Die Web-Anwendung weiterentwickeln

Natürlich lässt sich die erstellte Workflow-Anwendung an die eigenen Anforderungen anpassen. Wer sich das in Eclipse erzeugte JEE-Projekt genauer ansieht, wird feststellen, dass es aus nur wenigen Java-Klassen besteht. Die meiste Funktionalität kommt direkt aus den Imixs-JEE-Komponenten. Um die Anwendung anzupassen, genügt es meist, schon die im Web-Modul hinterlegten JSF-Seiten (.xhtml) zu ändern und die Java-Klasse „WorkflowMB“ zu erweitern. Wer mehr über die hier verwendeten JSF-Klassen erfahren möchte, kann unter http://doc.imixs.org/jsf/ beziehungsweise http://doc.imixs.org/modeler/ Beispiele über die Imixs-JSF-Tools-Projekt-Homepage aufrufen oder den Source Code einsehen.

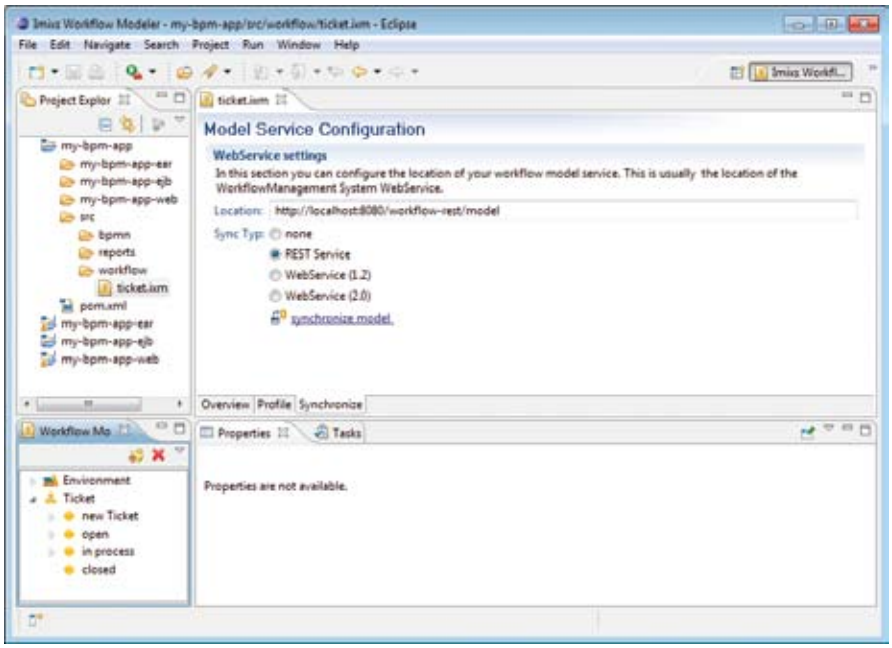


Abbildung 6: Imixs-Modeller

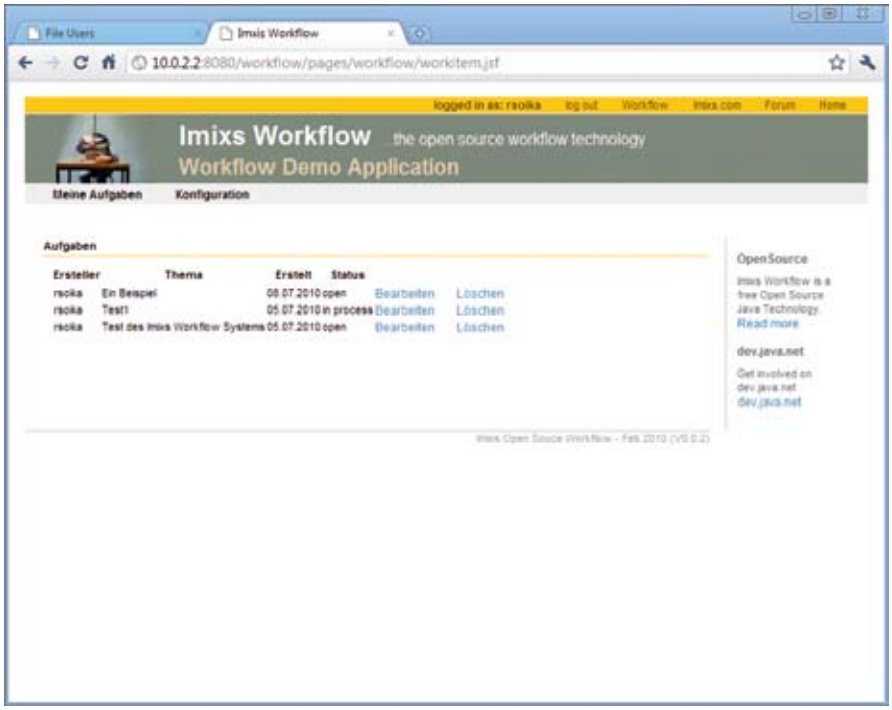


Abbildung 7: Die fertige Workflow-Anwendung

Kontakt:
Ralph Soika
ralph.soika@imixs.com

Ralph Soika begann mit 16 Jahren Homecomputer in Basic, Assembler und Turbo Pascal zu programmieren. Danach hat er sich mit Windows-Programmierung in Delphi beschäftigt. Seit 1998 konzentriert er sich auf die Entwicklung von Software mit Java. Seine Schwerpunkte liegen im Bereich JEE. Er betreibt seit 2005 das Open-Source-Projekt Imixs.

Cloud in a Box

Jens Dollenbacher, Oracle Software (Schweiz) AG

Cloud Computing ist zu Zeit der beherrschende IT-Trend als Werkzeug der Konsolidierung, optimaler Ausnutzung bestehender Ressourcen und flexibler Lastverteilung in Rechenzentren.

Technologien wie Virtualisierung und schnelle Netzwerke ermöglichen eine neue Dynamik im unternehmensweiten Computing. Sie bieten neue Möglichkeiten zur Kosteneinsparung und schnellem Deployment – der Sprung von „Software as a Service" zu „Platform as a Service". Längst sind große Anbieter wie Amazon auf diesen Zug aufgesprungen und haben auf dieser Basis erfolgreich Geschäftsmodelle etabliert: die „Public Cloud", die es jedermann ermöglicht, Computer-Ressourcen kurzfristig zu mieten, zu nutzen und danach wieder in der Wolke verschwinden zu lassen. Für Unternehmen außerhalb dieses Dienstleitungsbereichs bietet sich die „Private Cloud" an, die dynamische Nutzung dieser Ressourcen innerhalb einer Unternehmens-IT.

Viele Jäger sind des Hasen Tod

Die Einführung einer solchen „Private Cloud" stellt allerdings Herausforderungen unterschiedlicher Art. Zum einen entstehen organisatorische Konsequenzen, denn auch eine virtualisierte Umgebung will verwaltet werden, zum anderen stellt sich auch die Frage, auf welcher Hard- und Software-Basis eine solche Umgebung entstehen soll. In klassischen Umgebungen treffen wir in den Rechenzentren eher kleinere Maschinen im Bereich von vier bis sechzehn Cores an, auf denen zumeist nur eine Applikation dediziert betrieben wird, mit allen daraus resultierenden Konsequenzen, wie mangelhafte Auslastung der Maschine, vermehrter Standplatz und großer Administrationsaufwand. Ein immer größer werdender Maschinenpark, teils mit unterschiedlicher Hardware, teils mit unterschiedlichen Betriebssystemen, jagen den Administrator vor sich her wie die Jäger den Hasen, und viele Jäger sind bekanntlich des Hasen Tod; der Aufwand im Operations wird durch Provisioning, Deployment, Patch Management und Monitoring immer größer. Dieses Problem wird auch durch die klassische Virtualisierung nicht kleiner. Wir brauchen also neue Konzepte und Technologien, um die Herausforderungen einer Cloud zu meistern.

Die „Cloud in a Box"

Mit der Exalogic-Maschine hat Oracle nun eine Lösung präsentiert, die genau diese Punkte adressiert. Statt mit vielen, relativ kleinen Maschinen unterschiedlicher Bauart und mit unterschiedlichen Betriebssystemen geht Oracle nun den Weg, seinen Kunden eine Komplettlösung für Private Clouds anzubieten, eine „Cloud in a Box". Die Exalogic-Maschine ist modernes Computing vom Feinsten, von Hardware-Experten von Sun und Software-Experten von Oracle gemeinsam entwickelt und für die Aufgabenstellung optimal auf maximale Performance und Hochverfügbarkeit getunt. Dieses Expertenwissen und über hundert Personenjahre an Arbeit stecken in dieser Lösung, die es ermöglicht, Applikationen in perfekt virtualisierbarer Umgebung hochperformant zu betreiben, Ressourcen on Demand zuzuordnen, und dies auf einer absolut standardisierten Plattform, die nicht beim Kunden, sondern beim Hersteller gereift ist. Das bedeutet, dass nicht nur Tuning und Engineering dieser Maschine nicht mehr vom Kunden zu leisten sind, sondern bereits im Auslieferungszustand ein perfekter Setup steht. Und hier ist erst der Anfang.

Patch-Management in IT-Operations stellt immer eine große Aufgabe dar. Welcher Patch kann in welcher HW/SW-Kombination eingespielt werden? Dies erfordert nicht unbeträchtlichen Aufwand an Testing und kann damit auch zum wahren Showstopper im Upgrade-Prozess werden. Dies alles entfällt in einer durch den Hersteller komplett standardisierten Um-

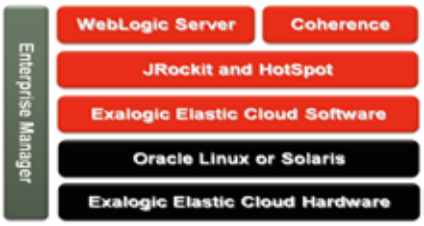


Abbildung 1: Das System im Überblick

	Quarter-Rack	Half-Rack	Full-Rack	2-8 Racks
2.93 GHz Xeon Cores	96	192	360	720-2880
1333 MHz RAM	768 GB	1.5 TB	2.8 TB	5.6-22.4 TB
FlashFire SSD	256 GB	512 GB	960 GB	1.9-7.7 TB
SAS Disk Storage	40 TB	40 TB	40 TB	80-320 TB

Tabelle 1

gebung. Patches aller Art, ob Treiber, Betriebssystem oder Applikationsserver, sind bei Oracle/Sun bereits auf absolut identischer Hardware und Software getestet und freigegeben; der Aufwand verschiebt sich vom Kunden zum Hersteller. Erfahrene und talentierte Mitarbeiter können sich auf wichtigere Aufgaben konzentrieren.

Ein Systemüberblick

Die Exalogic-Maschine (siehe Abbildung 1) ist die erste Maschine, die komplett dafür designed worden ist, Unternehmen eine sichere und hochverfügbare Private Cloud zu ermöglichen, mit einer unerreichten Performance, extrem einfachem Management und fast unbeschränkter Skalierbarkeit. Sie ist eine ideale Plattform für Applikationen aller Art. Obwohl die Exalogic-Maschine für Enterprise Java, Fusion Middleware und Fusion Applications optimiert ist, stellt sie dennoch eine optimale Umgebung für tausende Applikationen aller Art für Linux und Solaris. Sie ist die Maschine des Datacenters des 21. Jahrhunderts. Durch den optimalen und ausbalancierten Einsatz standardisierter Hardware-Technologien wie Infiniband, Flash Disks und Intel-Xeon-Prozessoren, optimiert durch Oracle Elastic Cloud Software und maximal getunte Oracle-Java-Technologie, entsteht ein völlig neuer Typ von Maschine, die in Bezug auf Leistung und Managebarkeit die erste ihrer Art ist.

Die Exalogic-Maschine wird in drei Basiskonfigurationen angeboten: Das Quarter-, das Half- und das Full-Rack, drei verschiedene Ausbaustufen, welche komplett Upgrade-fähig sind, das heißt, ein Quarter-Rack lässt sich z.B. problemlos nachträglich zu Half- oder Full-Rack ausbauen. Und auch ein Full-Rack lässt sich noch bis zu acht Maschinen skalieren. Tabelle 1 zeigt die verschieden Typen.

Extreme Java Performance

Die Kombination aus Oracle Exalogic Software und Hardware führt zu signifikanten Performancezugewinnen bei Java-Applikationen, die auf Weblogic Server und anderen Fusion-Middleware-Technologien laufen. Benchmark-Tests auf vergleichbarer Hardware (gleicher Prozessor, gleiches RAM) lassen hier einen Vergleich zu und zeigen in verschiedenen Szenarien deutliche Verbesserungen:

- Zwei- bis dreifache Verbesserung bei OLTP-Applikationen gegenüber den DB
- Bis zu 60 Prozent mehr Java-Operationen pro Sekunde
- Bis zu zehnfach schnellere Antwortzeiten

Diese Ergebnisse zeigen die deutliche Überlegenheit der Exalogic-Maschine gegenüber üblicher Standardhardware. Was macht den Unterschied? Die Optimierung für genau diesen Zweck. So wie ein Fahrzeug für einen bestimmten Zweck entworfen wird – ein Sportwagen für schnelles Fahren, ein Lastwagen für große Lasten – ist die Exalogic-Maschine designt, um Java-Applikationen mit optimaler Performance auszuführen. Oder würde man ein Auto kaufen, das „für alles" gut sein soll?

Fazit

Die Herausforderungen an eine moderne IT sind zweifelsohne immer mehr Performance, dynamisches Ressourcen-Management, Verfügbarkeit und Skalierbarkeit, dies bei anhaltendem Kostendruck, der es erfordert, immer effizienter zu arbeiten. Um dies zu meistern, sind neue Wege und Lösungen notwendig. Systeme selbst zu bauen, die solchen Ansprüchen genügen, ist teuer, langwierig und mit Risiko behaftet. Die Exalogic-Maschine bietet hier eine Alternative, die es Unternehmen ermöglicht, von Beginn an ohne zusätzliches eigenes Engineering genau diese Punkte zu erreichen:

- Bessere Java-Performance
- Bessere Skalierbarkeit und Verfügbarkeit
- Bessere Managebarkeit

Weitere Informationen

http://www.oracle.com/us/products/middleware/exalogic/index.html

Kontakt:
Jens Dollenbacher
jens.dollenbacher@oracle.com

Jens Dollenbacher ist Principal Sales Consultant bei Oracle Schweiz und berät Kunden und Partner zu IT Architekturen. Seine Schwerpunkte sind Middleware, Java EE und IT Modernisierung.

AJAX-Entwicklung wie Swing

Gerald Kammerer, freier Redakteur

Vaadin ist eine Java-Grafikbibliothek, welche die Entwicklung und Pflege hochwertiger Web-Anwendungen auf Basis von AJAX (Asynchronous JavaScript and XML) erheblich vereinfacht.

Die Idee von Vaadin ist ein servergetriebenes Programmiermodell, mit dem Entwickler Web-Anwendungen vollständig in Java schreiben können – und zwar genauso wie mit Swing. Mit den Browser-Skriptsprachen HTML und JavaScript kommt der Vaadin-Entwickler nicht mehr in Kontakt, denn das Browser-Frontend wird komplett von Vaadin erzeugt. Beispielsweise wird ein Button mithilfe der Vaadin-Button-Klasse implementiert, während im Browser ein entsprechender JavaScript-Button angezeigt wird. Anders als andere AJAX-Frameworks übernimmt Vaadin nicht nur das Rendern der Oberfläche, sondern auch die komplette Kommunikation zwischen Browser-Frontend und Server - und somit den kompliziertesten Teil der AJAX-Programmierung.

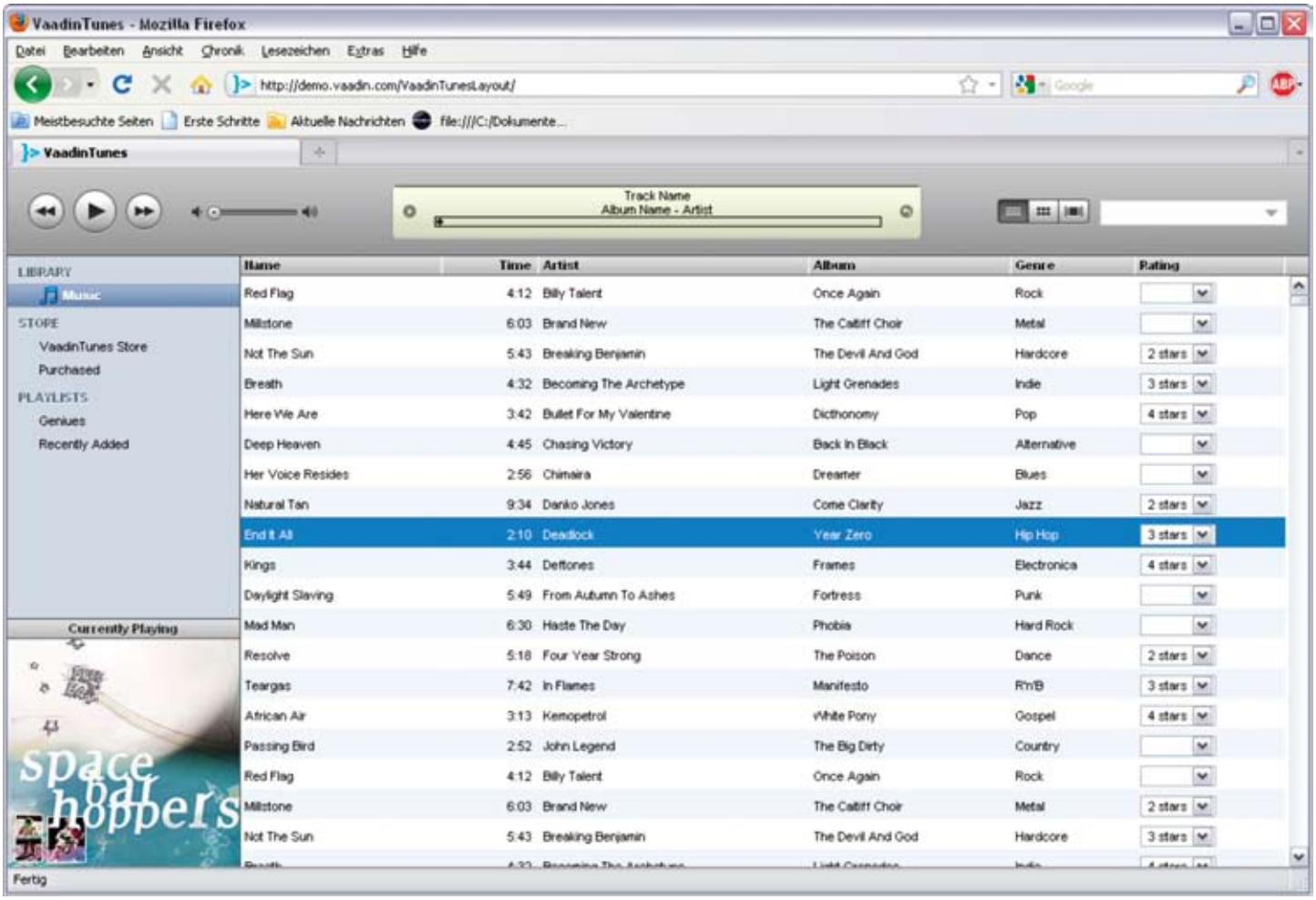


Abbildung 1: Mit Vaadin lassen sich Ajax-Anwendungen entwickeln, deren Aussehen und Verhalten sich kaum mehr von klassischen Desktop-Anwendungen unterscheidet. Was die Performance betrifft, sind echte Desktop-Programme jedoch immer noch klar schneller.

AJAX-Anwendungen vollständig in Java schreiben

Vaadin besteht aus einem serverseitigen Framework und einer clientseitigen JavaScript-Engine, die für das Rendern des Frontends im Browser sowie für Interaktionen mit dem Server zuständig ist. Beim Aufruf der Anwendung wird zunächst eine JavaScript-Engine geladen, welche mithilfe von AJAX die Oberfläche aufbaut. Die Businesslogik einer Vaadin-Applikation läuft dagegen als Servlet in der Umgebung eines Application-Servers auf dem Server. Da die Geschäftslogik auf dem Server getrennt und völlig unabhängig vom Frontend im Client läuft, kann man sich das User-Frontend wie einen Thin-Client vorstellen, der lediglich die Oberfläche anzeigt, Anwender-Aktionen entgegennimmt und an die eigentliche Applikation auf dem Server weiterleitet.

Da das komplette GUI dynamisch von einem JavaScript-Programm erzeugt wird, das jeder Web-Browser interpretieren kann, ist für die Ausführung einer Vaadin-Applikation - anders als bei Flash, Silverlight oder Java-Applets - kein proprietäres Browser-Plug-in erforderlich. Da die Oberfläche vom Framework selbstständig erzeugt wird, sind zudem keine eigenhändigen Browser-Anpassungen seitens des Entwicklers mehr notwendig. Aufwändiges und zeitraubendes Testen und Anpassen des Frontends an die verschiedenen Browser ist somit nicht mehr notwendig. Um das Frontend genauso reaktionsschnell und interaktiv zu machen wie konventionelle Desktop-Applikationen, setzt Vaadin auf AJAX (Asynchronous JavaScript and XML).

Frontend mit Google Web Toolkit

Was Vaadin aber so richtig spannend macht, ist die Tatsache, dass das Framework unter der Haube das Google Web Toolkit (GWT) für das Rendern der Oberfläche nutzt. GWT gilt als ausgereift, performant und ist für sehr große Nutzerzahlen ausgelegt, was die von Google selbst mit GWT entwickelten Applikationen wie Google Maps eindrucksvoll unter Beweis stellen. Auch GWT-Programme werden in Java geschrieben, sodass sich der Entwickler nicht mit HTML und JavaScript auseinandersetzen muss. Mithilfe des GWT-Compilers werden diese jedoch vollständig in JavaScript übersetzt.

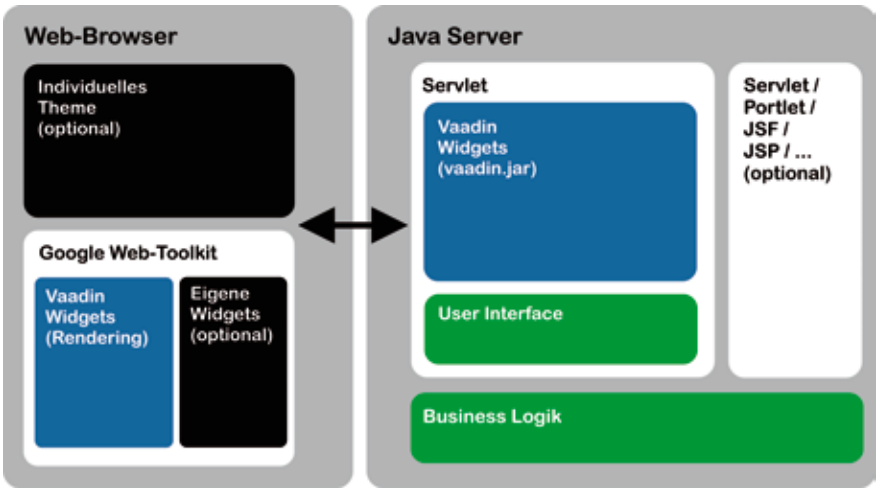


Abbildung 2: Die komplette Business-Logik einer Vaadin-Applikation läuft als Servlet auf dem Server. Das HTML/JavaScript Framework wird mit Hilfe von Google Web-Toolkit gerendert. Eigene Widgets und Themes lassen sich in Vaadin recht leicht einbinden. Die Datenübertragung managed das Framework, sodass sich der Entwickler darum nicht kümmern muss.

```
package com.vaadin.demo;

import com.vaadin.ui.Label;
import com.vaadin.ui.Window;

@SuppressWarnings(„serial“)
public class HelloWorld extends com.vaadin.Application {

    /**
     * Init is invoked on application load (when a user accesses the appli
       cation
     * for the first time).
     */
    @Override
    public void init() {

        // Main window is the primary browser window
        final Window main = new Window(„Hello window“);
        setMainWindow(main);

        // „Hello world“ text is added to window as a Label component
        main.addComponent(new Label(„Hello World!“));
    }
}
```

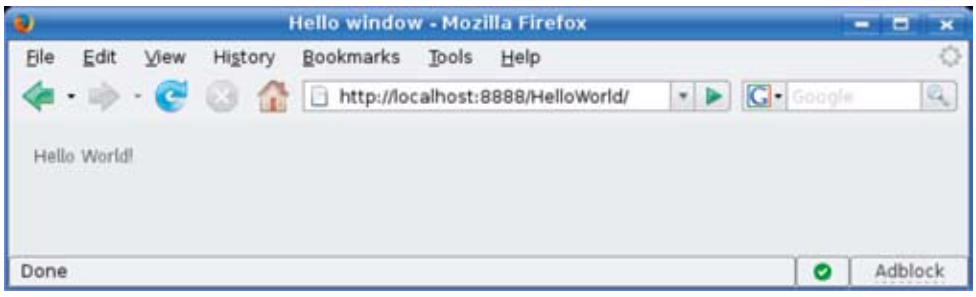


Abbildung 3: Das Ergebnis nach dem Aufruf (siehe Listing darüber)

GWT eignet sich somit sehr gut für die Implementierung von UI-Controls und Interaktionslogik auf dem Client. Um die Kommunikation zwischen Client und Server müssen sich GWT-Entwickler jedoch selber kümmern. GWT setzt dabei auf Remote Procedure Calls (RPC). Dabei versendet der Client eine Anfrage an eine Prozedur auf dem Server und wartet auf Antwort. Nach dem Empfang der Nachricht kann der Client mit anderen Prozessen fortfahren. Vaadin dagegen nimmt dem Entwickler die Client-Server-Kommunikation komplett ab, was die Architektur einer AJAX-basierenden Web-Anwendung und deren Programmierung erheblich einfacher macht. Vaadin komplettiert den Ansatz von GWT sozusagen. Für die Übertragung der Daten verwendet Vaadin HTTP(S), das JSON-Format sowie die User Interface Definition Language (UIDL), womit der Entwickler jedoch nicht konfrontiert wird.

Vaadin selbst stellt bereits alle wichtigen UI-Controls wie Fenster, Tabellen, Registerreiter sowie Formular-Controls zur Verfügung. Die Qualität der Controls ist dabei so gut, dass sich eine Vaadin-Oberfläche nicht von einer Desktop-Anwendung unterscheiden lässt. Ähnlich wie Swing bietet auch Vaadin verschiedene Look&Feels (Themes). Mithilfe von CSS lassen sich bei Bedarf auch individuelle Themes umsetzen. Zudem sind erstaunlich viele Extensions verfügbar. Da Vaadin-Oberflächen mit GWT erzeugt werden, lassen sich grundsätzlich alle GWT-Komponenten recht einfach einbinden. Wie Swing bietet auch Vaadin verschiedene Layout-Manager für die Realisierung skalierfähiger Oberflächen und Dialoge. Im Vergleich zu Swing erscheinen die Möglichkeiten zwar noch etwas rudimentär, die meisten Anwendungsfälle lassen sich damit jedoch sehr gut abdecken.

Geschäftslogik vollständig in Java schreiben

Der Einsatz von GWT bedeutet zudem, dass auch die Geschäftslogik einer Vaadin-Applikation in Java geschrieben wird. Für den Server-Teil setzt Vaadin auf die Java-Servlet-API auf. Aber auch mit dieser muss sich der Vaadin-Entwickler nicht direkt auseinandersetzen. Beim Aufruf einer Vaadin-Anwendung wird jeweils eine Instanz des Servlets mit Session erzeugt. Solange eine Session existiert, ordnet das Framework alle Client-Aktionen automatisch der richtigen Instanz zu.

Die Vaadin-Programmierbibliothek unterscheidet klar zwischen User-Interface und Programmlogik und erlaubt, diese auch unabhängig voneinander zu entwickeln. Vaadin wurde ganz klar für die Entwicklung von webfähigen Geschäftsanwendungen entwickelt, weniger für die Programmierung von herkömmlichen

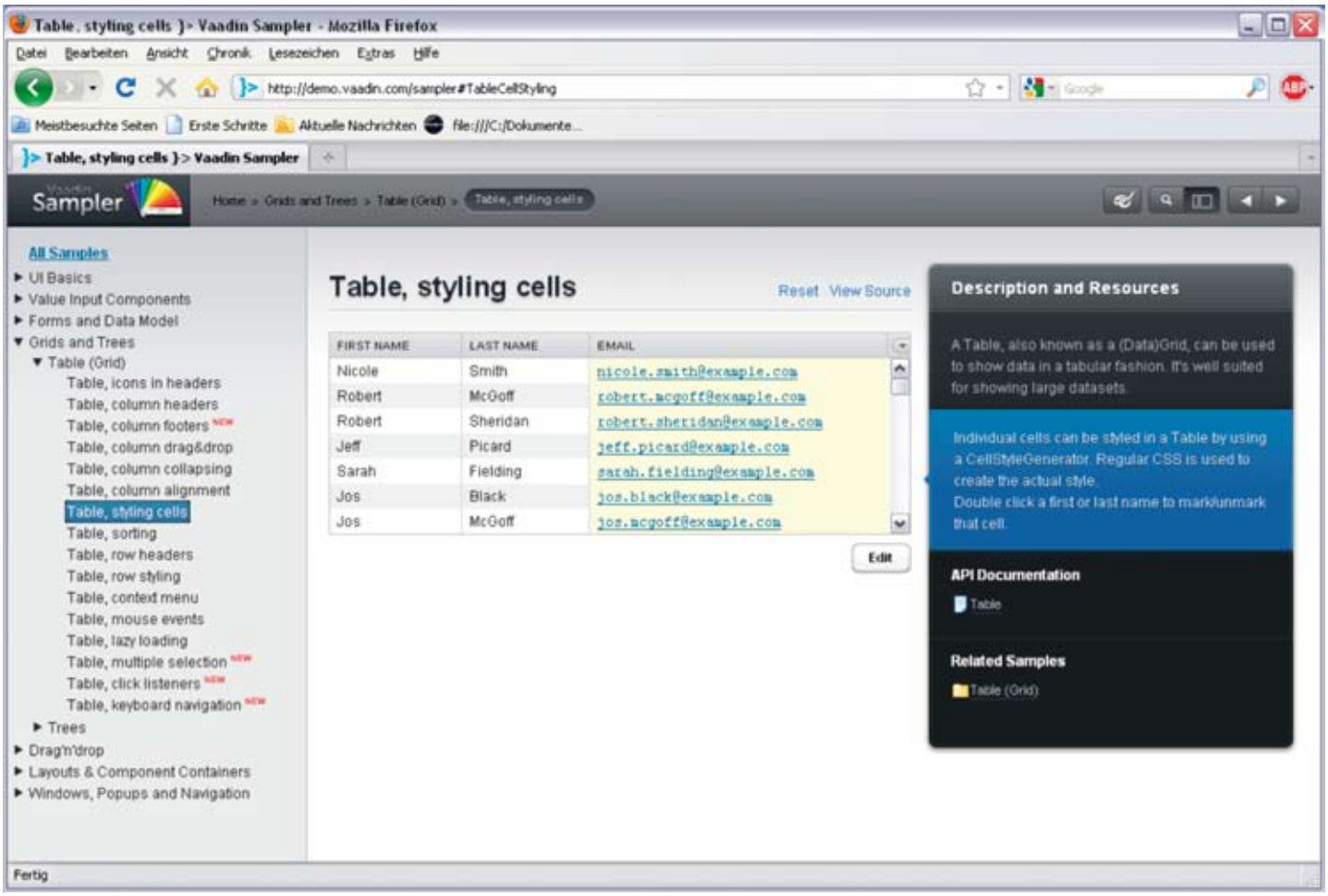


Abbildung 4: Auf der Vaadin Webseite findet man für alle verfügbaren UI-Controls entsprechende Demos.

Webseiten oder Portalen. Für Eclipse bietet Vaadin ein Plug-in, mit dem sich neue Vaadin-Projekte anlegen, Themes individuell anpassen und eigene UI-Controls entwickeln lassen.

Die XDEV Software Corp. hat für 2011 eine RAD-Entwicklungsumgebung für Vaadin angekündigt, mit der AJAX-Entwicklung sehr einfach und die AJAX-Technologie auch für 4GL-Entwickler zugänglich gemacht werden soll. Die IDE soll einen intuitiven Vaadin-GUI-Builder enthalten, der nicht nur den Vaadin-GUI-Code inklusive Event-Handler generiert, sondern auch Änderungen des generierten Codes und somit bidirektionale GUI-Entwicklung erlaubt. Damit sollen Software-Architekten und -Designer in der Lage sein, die komplette Oberfläche einer AJAX-Anwendung zu entwickeln, ohne dafür eine Zeile Code schreiben zu müssen. Datenbankzugriffe und Visualisierung sollen ähnlich einfach wie in Access und Oracle Forms funktionieren. Dazu kommen eine für Datenbank-Anwendungen optimierte Funktionsbibliothek und ein Java-Code-Editor für die Entwicklung der Anwendungslogik. Damit entwickelte Anwendungen sollen sich anschließend problemlos auf Eclipse portieren lassen. Informationen zu diesem Projekt finden Sie unter www.xdev-software.de. Vaadin ist als Open Source unter Apache-License verfügbar unter www.vaadin.com.

Kontakt:
Gerald Kammerer
info@redaktion-kammerer.de

Gerald Kammerer ist als freier Redakteur seit 2006 für verschiedene Computerfachzeitschriften mit Schwerpunkt Java- und AJAX-Entwicklung tätig. Seit über zehn Jahren programmiert er nun schon intensiv mit Java und kennt die Sprache seit ihren Anfängen. Seit einigen Jahren beschäftigt er sich nun auch intensiv mit der Entwicklung von Rich Internet Applications mit Java, JavaFX und verschiedenen AJAX-Frameworks wie jQuery, Wicket, Google Web-Toolkit und Vaadin (siehe auch Seite 20).

Mitglieder stellen sich vor

Neu im iJUG: Swiss Oracle User Group (SOUG)

Vor dem Hintergrund der Sun-Übernahme durch Oracle haben sieben Java-Usergroups aus Deutschland zusammen mit der DOAG Deutsche ORACLE-Anwendergruppe e. V. sowie der Sun User Group (SUG) Deutschland e.V. den Interessenverbund der Java User Groups e.V. (iJUG) gegründet. 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. Eine weitere Aktivität ist die Herausgabe dieser Zeitschrift. In der letzten Ausgabe wurden die Gründungsmitglieder ausführlich vorgestellt. Neu hinzugekommen ist die Swiss Oracle User Group (SOUG).

Die SOUG wurde 1987 als unabhängiger Verein gegründet und umfasst heute rund 600 Mitglieder. Als Community der Oracle-Spezialisten fördern sie den Informations- und Erfahrungsaustausch, vertriit die Interessen der Mitglieder auf nationaler sowie internationaler Ebene und bietet verschiedene Möglichkeiten zur Vernetzung mit anderen Personen. Die SOUG-R (SOUG Romandie) ist in der Westschweiz für die französisch sprechenden Mitglieder aktiv.

Das Angebot für die Mitglieder umfasst Veranstaltungen (Special Interest Group Meetings, Konferenzen), den vierteljährlich erscheinenden Newsletter (Print-Maganzin), Downloads (Newsletter-Artikel, Präsentationen etc.) und verschiedene Vergünstigungen wie Rabatt bei Schulungsanbietern und reduzierte Teilnahmegebühren an User Group Konferenzen.

Traditionell war die SOUG eher Datenbank- und Oracle-Tool-orientiert. Mit dem Aufkommen von Java gewann dieses Thema natürlich auch für Oracle-Spezialisten an Bedeutung. Heute hat Java einen festen Stammplatz an den Veranstaltungen und im Newsletter der SOUG.

Mit dem Beitritt zum iJUG bietet die SOUG ihren Mitgliedern ein erweitertes Angebot. Die Java aktuell ist wie der SOUG Newsletter im Mitgliederbeitrag enthalten und die Vernetzung mit Java-Spezialisten wird einfacher. Auf der anderen Seite möchte man auch den Mitgliedern des iJUG einen Mehrwert bieten. Als Oracle User Group kann die SOUG den Java-Spezialisten die „Oracle-Welt" unabhängig vom Hersteller etwas näher bringen. Die in diesem Bereich gemachten Erfahrungen dürften auch für die Java-Communities von Interesse sein. Weitere Informationen: www.soug.ch

JSXP-Templates

David Tanzer, Freiberufler

Just Simple eXtensible Pages (JSXP) ist ein Open-Source-Web-Anwendungs-Framework, das in der ersten Ausgabe von „Java aktuell" bereits kurz vorgestellt wurde. Eines der Hauptziele des Frameworks ist, eine enge Zusammenarbeit zwischen Designern und Entwicklern zu ermöglichen. Dafür muss man für bestimmte Bereiche einer Webseite einheitliche Designs definieren können – das macht man in JSXP mittels View Templates.

Ein View Template bestimmt eine Grundstruktur für beliebige andere Views. Solche Templates werden, wie jede andere View auch, als XML-Datei (etwa XHTML) definiert und mittels Java Code mit Funktionalität angereichert. Diese View (XHTML) und deren View Controller (Java) kann man nicht nur als Template, sondern auch als eigene Seite verwenden.

Das Template

Als Beispiel erstellen wir eine Seite „index.xhtml", die sowohl als Einstieg für die Anwendung dient, als auch die Grundstruktur für alle weiteren Seiten vorgibt. Auf dieser Seite kann man z.B. das Hauptmenü für die gesamte Anwendung definieren (siehe Listing 1).

Nach dem Speichern dieser Datei wird automatisch eine Basisklasse für den View Controller generiert. Von dieser generierten Klasse muss man ableiten und in der abgeleiteten Klasse bekanntgeben, dass diese View als Template verwendet werden kann. Dazu geben wir an, welche Teile des Templates durch andere Seiten ausgetauscht werden können (siehe Listing 2).

In der Anwendungsklasse muss das Template noch registriert werden. Das bewirkt, dass das Template auf alle Seiten der Web-Anwendung benutzt wird (siehe Listing 3).

Anwenden auf eine andere Seite

Das oben definierte Template wird ab jetzt auf alle Seiten angewendet. Jede Seite kann den Bereich mit der jsxp:id „content" austauschen, indem die Seite selbst einen Bereich mit dieser id definiert (siehe Listing 4).

Alle anderen Bereiche dieser Seite werden ignoriert. Diese Tatsache kann vom Designer genutzt werden, um in alle De-

```
index.xhtml:
<html>
	<body>
		<ul class="menu" jsxp:id="main_menu">
			<li jsxp:id="menu_item">$(MenuItemText)</li>
		</ul>
		<div jsxp:id="content">
			Willkommen in unserer Webanwendung
		</div>
	</body>
</html>
```

Listing 1

```
IndexXhtmlController.java:
public class IndexXhtmlController
		extends IndexXhtmlControllerGenerated<Object> {
	@Override
	public Element[] getTemplateElements() {
		return new Element[] { getElementContent() };
	}
}
```

Listing 2

```
Application.java:
public class Application extends WebApplication {
	private ViewController<?> template;
	public Application () {
		template = new IndexXhtmlController();
	}
	@Override
	protected ViewController<?> getTemplate() {
		return template;
	}
}
```

Listing 3

```
page2.xhtml:
<html>
	<body>
		<div jsxp:id="content">
			Das ist Seite 2 der Anwendung
		</div>
	</body>
</html>
```

Listing 4

signs Mockups der gesamten Seite einzufügen. Unsere „page2.xhtml“ könnte also ein Beispiel des Menüs enthalten, wie es sich der Designer für genau diese Seite vorstellt.

Funktionalität im Template

Um in Seiten Funktionalität zu implementieren, wird normalerweise die Methode „execute“ überschrieben. Diese wird aber nur aufgerufen, wenn die Seite selbst gerendert wird, nicht wenn sie ein Template für eine andere Seite ist. Deshalb muss man Funktionalität, die auf allen Seiten benötigt wird, in der Methode „init“ definieren. Funktionalität, die nur auf der Index-Seite benötigt wird, kommt weiterhin in „execute“(siehe Listing 5).

```
IndexXhtmlController.java:
public class IndexXhtmlController
              extends IndexXhtmlControllerGenerated<Object> {
       @Override
       protected void init(Importer importer) {
              menu = getElementMenu();
              //Menü dynamisch erzeugen...
       }
       @Override
       public void execute() {
       }
}
```

Listing 5

Fazit

Wir haben gezeigt, dass View Templates in JSXP verwendet werden können, um eine einheitliche Struktur für alle Seiten einer Webanwendung zu definieren. Es ist außerdem möglich, Templates nur für Teilbereiche der Seite zu definieren, und sogar, Templates zu schachteln. So kann man ein globales Template für einen Teilbereich der Seite weiter verfeinern. Einzelne View Controller können auch angeben, dass sie nicht am globalen Templating teilnehmen möchten, indem sie die Methode „isGlobalTemplateEnabled“ überschreiben. Insgesamt steht so ein mächtiges Werkzeug zum Design von Webanwendungen zur Verfügung.

Kontakt:
David Tanzer
david@davidtanzer.net

David Tanzer ist seit 2006 als selbstständiger Software-Entwickler und Berater tätig. Im Rahmen dieser Tätigkeit beschäftigt er sich mit Java-Enterprise-Anwendungen, Web-Anwendungen, mobilen Anwendungen und agiler Software-Entwicklung. Er ist Mit-Initiator des Java-Web-Frameworks JSXP (siehe auch Seite 27).

Java-Entwicklung wie Oracle Forms geplant

Markus Stiegler, XDEV Software GmbH

Anwender die sich schon lange eine richtig einfache Entwicklungsumgebung für Java wünschen, können jetzt die Entwicklung von XDEV 3 aktiv mitgestalten und maßgeblich beeinflussen.

Die XDEV Software Corp. plant, die freie Rapid Java Entwicklungsumgebung XDEV 3 so eng wie möglich an 4GL-Tools wie Oracle Forms und Microsoft Access anzulehnen. Dadurch sollen 4GL-Entwickler endlich schnell und einfach auf Java umsteigen können, eine vertraute Umgebung vorfinden und wie gewohnt entwickeln können. Dazu möchte XDEV Software die Anwender mit ins Boot holen und lädt alle Oracle Forms, Access und andere 4GL Entwickler am 23. März 2011 zu einem Brainstorming-Workshop in Nürnberg ein, bei dem zuerst die bereits vorhandenen RAD-Features von XDEV 3 vorgestellt werden und anschließend jeder Teilnehmer Feature benennen und vorführen kann, die XDEV 3 haben sollte. Ziel des Workshops ist die gemeinsame Erarbeitung einer Feature-Liste und Roadmap für die Umsetzung. Thomas Schmetzer, Geschäftsführer von XDEV Software Deutschland GmbH: „Wir wollen möglichst alle Featuers in XDEV 3 abbilden, die für 4GL-, allen voran für Forms-Entwickler wichtig sind, um den Umstieg und Softwaremigrationen auf Java so einfach wie möglich zu machen. Unsere Anwender können mit diesem Workshop die Weiterentwicklung von XDEV 3 entscheidend mitbestimmen“. Interessierte Anwender können sich unter www.xdev-software.de anmelden.

Kontakt:
Markus Stiegler
info@xdev-software.de

Markus Stiegler ist Technologie-Evangelist und Product Manager für die Entwickler-Toolsparte der XDEV Software Corp. in Deutschland. In dieser Funktion ist er verantwortlich für die Entwicklung von Produkt-, Marketing- und PR-Strategien und deren Umsetzung, für die Verbreitung von XDEV-Lösungen im deutschsprachigem Raum sowie für Kooperationen mit Technologie-Partner. Derzeit ist Markus Stiegler mit dem Aufbau eines Experten-Teams für Migrationsprojekte beschäftigt, das ab Anfang 2011 in der Lage sein soll, besonders zeitkritische Migrationen auf Java und AJAX sicher und kostengünstig durchzuführen. Von 2007 bis 2008 war Markus Stiegler für die Produktion der Fernsehsendung XDEV TV zuständig, die XDEV Software Corp. zusammen mit dem Gaming-Sender GIGA Digital bis zu dessen Auflösung produzierte. Neben der Co-Moderation leitete er das Redaktionsteam.

Integration mit Launch4j – der letzte Schliff

Dr. Stefan Koch, ORDIX AG

Anwender lieben das, was sie kennen. Also müssen sich Java-Programme nahtlos in die MS-Windows-Welt einfügen. Das Open-Source-Werkzeug Launch4j hilft dabei und lässt Java-Anwendungen genauso starten wie native Programme oder wandelt sie in ausführbare Windows-Dateien um. Obwohl Java mit Java Web Start selbst ein mächtiges Konzept zum Starten und Verteilen von Java-Software zur Verfügung stellt, wird dabei die Plattformunabhängigkeit angestrebt. Launch4j dagegen ist der Spezialist in Sachen „Integration".

Unter Windows-Systemen kommt es oft zu Schwierigkeiten, wenn Anwender versuchen, Java-Anwendungen auszuführen. Abhilfe schafft hier das Open-Source-Werkzeug Launch4j. Nach der Installation erstellt es ein Startprogramm für Java-Anwendungen. Dieses hat die typischen Eigenschaften einer nativen Anwendung. Es lässt sich als exe-Datei mit Doppelklick starten und wird im Explorer mit eigenem Icon dargestellt. Die Versionseigenschaften eines nativen Programms wie Dateiversion, Firma, Produktname und -version werden im Windows-Explorer angezeigt.

Die durch Launch4j erzeugte exe-Datei zeigt bei der Ausführung einen Startbildschirm; die installierte Java-Version wird überprüft. Alternativ ist eine spezielle Java-Installation für den Start möglich. Das Startprogramm kann die jar-Datei beinhalten. Der Nutzer sieht nur das ausführbare Programm. Launch4j ist über die Kommandozeilen-Schnittstelle oder über ant-Tasks in den automatischen Build-Prozess integrierbar. Für die Entwicklung der Konfiguration steht eine grafische Benutzeroberfläche zur Verfügung.

Los geht's: Download und Installation

Launch4j ist über die Launch4j-Webseite [1] kostenlos verfügbar. Nach dem Download wird launch4j.exe gestartet (siehe Abbildung 1). Zunächst ist der Name des Programms im Reiter „Basic" als Output-File einzutragen. Anschließend wird die jar-Datei angegeben, die gestartet werden soll. Standardmäßig wird sie in das Startprogramm eingebettet.

Als letzte erforderliche Information wird im Reiter „JRE" die Java-Version definiert, die für das Programm benötigt wird. Unter „Min JRE version" ist die Java-Version einzutragen, die durch „java –version" ausgegeben wird. Danach wird die Konfiguration abgespeichert. Anschließend muss der Anwender nur noch auf das Zahnrad klicken, um den Build-Prozess zu starten. Schon ist das Startprogramm fertig.

Neue Icons – darf es etwas bunter sein?

Das so erstellte Startprogramm wird im Windows-Explorer mit dem Standard-Icon für ausführbare Programme dargestellt (siehe Abbildung 2). Im Reiter „Basic" ist das Feld „Icon" für die Angabe eines Icons vorgesehen. Es ist sinnvoll, das Startprogramm mit einem Icon zu dekorieren – so kann der Benutzer das Programm leichter identifizieren. In der Regel wird das Icon des Anwendungsfensters verwendet.

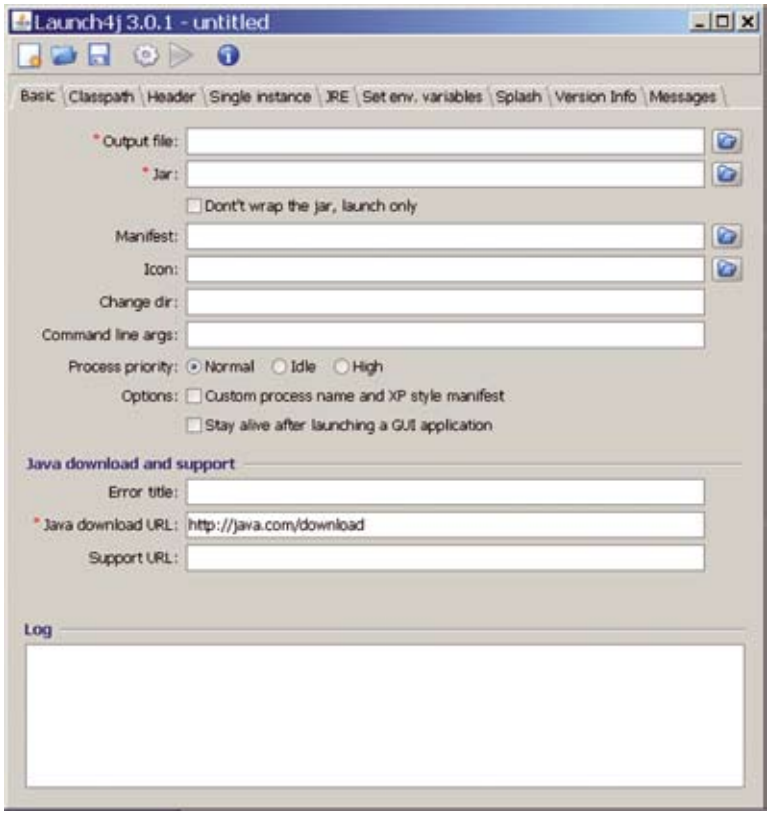


Abbildung 1: Benutzeroberfläche von Launch4

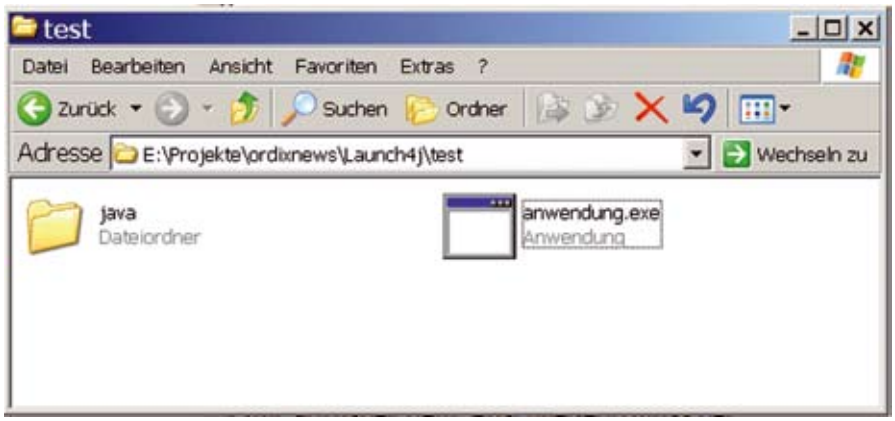


Abbildung 2: Explorer-Darstellung einer Anwendung ohne Icon

Bei Java-Programmen kann je nach Hardware einige Zeit vergehen, bis sich die Anwendung bemerkbar macht. Um dem Anwender unmittelbar zu zeigen, dass das Programm geladen wird, ist ein Bild für den Startbildschirm – auch „Splash" genannt – zu definieren. Das Startprogramm zeigt den Startbildschirm, noch bevor das Java-Programm gestartet wird. Dazu bietet Launch4j die Checkbox „Enable splash screen" im Reiter „Splash". Dort gibt der Entwickler eine Bitmap-Grafik als Splash-File an. Nach dem Erstellen des Startprogramms wird das definierte Icon im Windows-Explorer angezeigt (siehe Abbildung 3).

Java macht abhängig

Möglicherweise hat der Anwender nicht die richtige Java-Version installiert. In diesem Fall zeigt das Startprogramm die Nachricht, die im Reiter „Messages" im Feld „JRE" als „version error" angegeben ist. Darüber hinaus öffnet sich der HTML-Browser mit der Seite, die im Feld „Java download URL" des Reiters „Basic" eingegeben ist.

Gerade bei Java-Programmen mit Benutzeroberflächen liegt die Überlegung nahe, die Java Virtual Machine mit der Software auszuliefern, um Inkompatibilitäten zu neueren Java-Versionen auszuschließen. Damit diese Java-Version verwendet wird, muss im Reiter „JRE" das Feld „Bundled JRE path" mit dem Java-Installationspfad angegeben werden.

Weitere Startparameter

Typischerweise wird die Anwendung Abhängigkeiten zu weiteren Bibliotheken haben. Bei einer überschaubaren Anzahl können die Bibliotheken zusammen mit dem Programm in eine jar-Datei gepackt werden. Damit lassen sich externe Abhängigkeiten vermeiden. Alternativ werden die Abhängigkeiten zu externen Bibliotheken im Classpath angegeben. Launch4j spendiert dieser Einstellung eigens einen Reiter namens „Classpath".

Konfigurationsparameter für die virtuelle Maschine von Java, wie beispielsweise die Größe des Speicherraums, werden im Reiter „JRE" eingetragen. Bemerkenswert ist, dass der Speicherbedarf auch prozentual zum freien Speicherbereich angegeben werden kann. Weitere Startparameter für das Programm werden im Feld „Command line args" des Reiters „Basic" definiert. Umgebungsvariablen werden im Reiter „Set env. Variables" festgelegt.

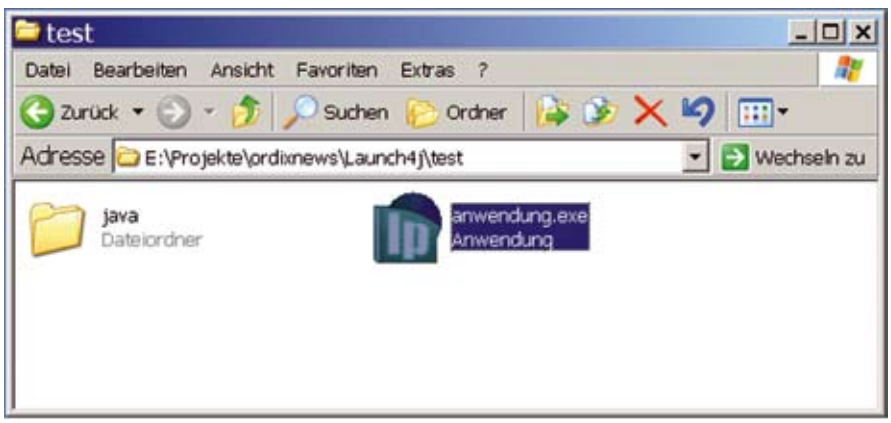


Abbildung 3: Darstellung des gewählten Icons im Explorer

Programmeigenschaften

Im Windows-Explorer lassen sich Informationen zu Programmen unter „Eigenschaften", Reiter „Version" anzeigen. Die wesentlichen Attribute sind „Beschreibung", „Copyright" und „Dateiversion". Darüber hinaus können auch Angaben zu Firma, Produktname, Produktversion und Sprache angezeigt werden. In Launch4j lassen sich diese Eigenschaften im Reiter „Version Info" definieren. Launch4j kann auch verhindern, dass eine zweite Instanz des Programms gestartet wird. Im Reiter „Single Instance" muss dazu ein sogenannter „Mutex name" angegeben sein.

Fazit

Launch4j ist bestens geeignet, damit der Anwender ein Java-Programm genauso wie eine native Anwendung nutzen kann. Der Umfang ist gut durchdacht, die Benutzerschnittstelle ist intuitiv bedienbar, ein automatischer Build-Prozess wird unterstützt. Sind Aspekte der Verteilung der Software und die Versions-Pflege zu berücksichtigen, ist der Einsatz von Java Web Start in Erwägung zu ziehen.

Kontakt:
Dr. Stefan Koch
info@ordix.de

Der Autor Dr. Stefan Koch, Java-Spezialist der ORDIX AG, schreibt seit Jahren Artikel über Neuigkeiten und Best-Practice-Möglichkeiten in der Java-Entwicklung in dem firmeneigenen IT-Magazin „ORDIX News".

Java-Coherence-Cache-Architektur

Wolfgang Weigend, ORACLE Deutschland B.V. & Co. KG

Steigende Transaktionsvolumen in der Verarbeitungslogik und zunehmende Anforderungen für Software-Echtzeitdaten bei der Prozessauswertung und bei SOA/ereignisgetriebenen Architekturen verursachen eine höhere Last auf den Backend-Systemen. Dabei soll die Infrastruktur die Anforderungen der Systeme und Anwender erfüllen und die geforderte Leistung liefern. Dies macht jedoch den Einsatz von neuen Architekturen notwendig.

Die Architektur-Lösung muss die Daten für eine höhere Leistung bereithalten, verteilte Datenmengen oftmals in Echtzeit verarbeiten und im Arbeitsspeicher befindliche Datenströme analysieren. Gleichermaßen soll sie sich in bestehende Hardware- und Software-Landschaften einfügen und Datenbanken, Anwendungsserver sowie Messaging-Systeme einbeziehen.

Mit einer Ressourcen-Bündelung durch die Anwendungsplattform werden die Skalierbarkeitsanforderungen optimal bedient, da sie die vorhandenen Ressourcen effizient koordiniert. Die Anforderungen an die Anwendungsplattform nach hohem Durchsatz, niedrigen Antwortzeiten und massiver Skalierbarkeit erfordern den Einsatz einer Middleware-Caching-Architektur und deren Implementierung. In einer gemeinsamen Speicherstruktur mit separaten Cache-Memory-Bereichen können die Speicher-Inhalte auf andere Software-Knoten beliebig verteilt werden; damit befinden sich ein Datum im Hauptspeicher und weitere Kopien im jeweiligen Cache-Memory. Cache-Coherence ist der Mechanismus, der dafür sorgt und sicherstellt, dass die Veränderungen der Daten sehr schnell innerhalb des Caching-Systems verteilt werden. Die daran teilnehmenden Anwendungen nutzen diese Daten und deren Änderungen direkt aus einer verteilten, linear skalierbaren und konzentrierten Caching-Struktur. Einfache und komplexe Abfragen können parallel innerhalb der Caching-Struktur verteilt „In-Memory“ ausgeführt werden. Damit wird auch die Anzahl der Zugriffe auf Backend-Systeme verringert oder stark eingeschränkt, um Datenbanken und Mainframe-Systeme zu entlasten.

Die Anforderungen an die IT-Infrastruktur steigen stetig und erfordern eine Optimierung in der Anwendungsplattform. Dabei sollen dynamische Änderungen und vorhersagbare Anwendungsskalierbarkeit die geschäftlichen Anforderungen unterstützen. Durch eine verbesserte Anwendungsleistung wird ein schnellerer Datenzugriff ermöglicht und die Antwortzeiten werden verkürzt. Durch die kontinuierliche Datenverfügbarkeit und Zuverlässigkeit mit definierten Service Level Agreements und deren Einhaltung werden die geschäftliche Kontinuität sichergestellt und die Kosten für Infrastruktur- und Entwicklungskosten gesenkt.

Neue Services und Events stellen auch neue Anforderungen an die Transaktionsverarbeitung. Mit zunehmender geschäftlicher Geschwindigkeit und Innovation nimmt auch die Komplexität von Services und damit auch deren Zustandsverwaltung zu. Dies erfordert die Bereitstellung von skalierbarer Leistung mit der Zielsetzung, dass ein Service innerhalb der erforderlichen, definierten Antwortzeit auch in extremen Situationen reagiert und bei Bedarf adhoc nahtlos skalierbar ist. Die dafür notwendige Transaktionsplattform verfügt über Merkmale einer verteilten Transaktionsverarbeitung (Distributed Transaction Processing for Open Systems) und extrem leistungsfähiger Transaktionsverarbeitung (XTP).

Die IT bestimmt den Datenbedarf und macht eine bessere Auslastung der Infrastruktur erforderlich, da die Verbreitung von Daten oftmals alle Systeme betrifft. Durch Virtualisierung wird die verbesserte Ausnutzung der Ressourcen erreicht, und eine Neuverteilung von Anwendungen erfolgt transparent und ohne Unterbrechnung des Datenzugriffs. Bei gleichzeitiger Datenversorgung und mehrfach erhöhter Last wird die Service-Infrastruktur einem starken Zugriff auf gemeinsame Ressourcen ausgesetzt und muss kontinuierliche Verfügbarkeit und absolute Zuverlässigkeit sicherstellen.

Geeignete Maßnahmen liefert Coherence in der aktuellen Version 3.6 über eine zuverlässige Datenschicht mit einer konsistenten Sichtweise auf die Daten und mit transaktionaler Integrität. Es wird die dynamische Erhöhung der Datenkapazität mit Lastverteilung, kontinuierlicher Verfügbarkeit und Fehlertoleranz ermöglicht. Auch die erforderliche Verarbeitung und Skalierbarkeit der benötigten Datenkapazität mit extrem niedrigen Antwortzeiten wird damit sichergestellt. Bei diesem Lösungsansatz wird ein Anwendungsserver mit Coherence als Datenvermittler eingesetzt, der die Datenversorgung mit dem Datenbedarf vermittelt und in der Lage ist, die Middleware-Schicht mit Standard-Hardware zu skalieren.

Die dabei auftretenden Herausforderungen betreffen mehrere Architektur-Schichten, von der Anwendung über Lastverteilung, Web-Server, Applikationsserver und Persistenzschicht bis zur Datenbank. Dabei wird die Transaktionsverarbeitung der Java-Anwendung besonders gefordert; Objekt-relationale Werkzeuge verzögern oft die Durchlaufzeiten und machen eine In-Memory-Speicherverarbeitung notwendig, um die gewünschten Antwortzeiten zu erzielen. Gleichermaßen darf der Transaktionsstatus nicht verloren gehen, und die Verwaltung von Session-Zuständen gebietet besondere Aufmerksamkeit in heterogenen Umgebungen. Den beschriebenen Herausforderungen begegnet Oracle Coherence durch Daten-Virtualisierung mit dem Caching von serialisierbaren Objekten. Die gemeinsame Nutzung von Daten wird organisiert und mit Read-Through und Read-Ahead Pat-

terns umgesetzt. Parallele Abfragen, Indizes und Transaktionen im Cluster werden abgefedert, und Coherence entlastet die unterliegende Persistenzschicht mit den Write-Through- und Write-Behind-Patterns. Dabei integriert sich Coherence mit Applikationsservern, Objekt-relationalen Werkzeugen und Datenbanken.

Coherence verteilt die Daten gleichmäßig auf alle Clusterknoten und sorgt dafür, dass auch ein Backup der Daten auf die entsprechenden Knoten verteilt wird. Dabei werden die Daten automatisch und synchron auf mindestens einen weiteren Server repliziert. Die logisch konsistente Sicht auf die Daten wird von allen Knoten hergestellt sowie mit einem Health-Check ständig gegenseitig überprüft. Der Status bei auftretenden Fehlern wird über alle Knoten Cluster-weit diagnostiziert. Wird ein Knoten aus dem Cluster entfernt, so werden die Daten wieder gleichmäßig verteilt. Durch die automatische und dynamische Partitionierung der Speicherdaten über mehrere Server gewährleistet Oracle Coherence die kontinuierliche Verfügbarkeit der Daten und Transaktionsintegrität auch beim Serverausfall. Neu hinzugefügte oder nach einem Ausfall erneut gestartete Server werden automatisch in das Cluster eingebunden und über Oracle Coherence mit Service-Anfragen versorgt, sodass die Cluster-Last transparent umverteilt wird. Oracle Coherence umfasst Funktionen zur Fehlertoleranz auf Netzwerkebene und zum transparenten Warmstart im Rahmen der Server-Selbstheilung. Coherence-Caching ermöglicht den Anwendungen, Daten direkt aus verteilten und linear skalierbaren, gebündelten Datenquellen zu nutzen. Die Anzahl der Zugriffe auf Backend-Systeme wird reduziert bzw. gezielt vermieden (Datenbanken, Mainframe). Das Write-Queue-Pattern reduziert dabei die Last auf Backend-Systeme. Die Analyse-Funktionalität von Coherence bietet Aggregationen, und einfache sowie komplexe Abfragen können parallel im Daten-Pool clusterweit verteilt im Speicher „In-Memory" ausgeführt werden. Als gemeinsam genutzte Infrastruktur kombiniert Coherence die lokale Datenverfügbarkeit mit lokaler Verarbeitungskapazität und ermöglicht damit Datenanalysen in Echtzeit, speicherbehaftetes Grid Computing sowie die parallele Verarbeitung von Transaktionen und Ereignissen mit automatischem Event-Handling und Änderungsbenachrichtigungen (siehe Abbildung 1).

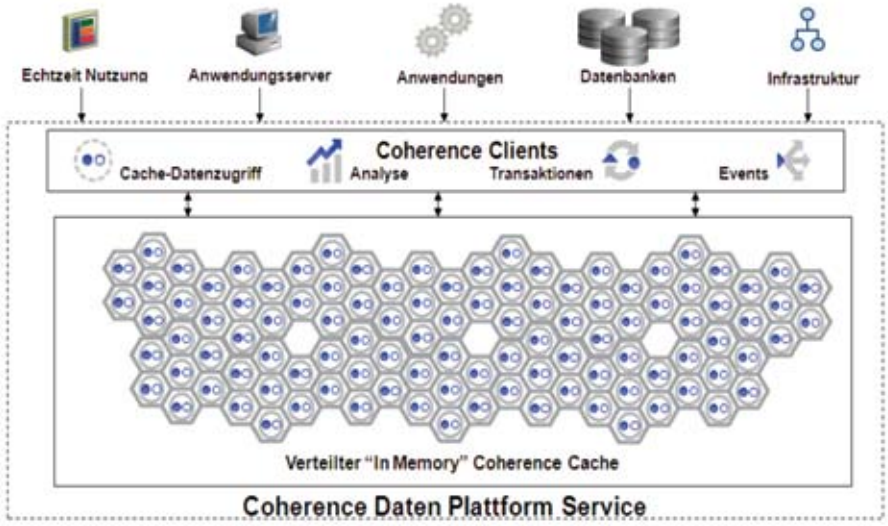


Abbildung 1: Verteilte In-Memory Caching-Struktur

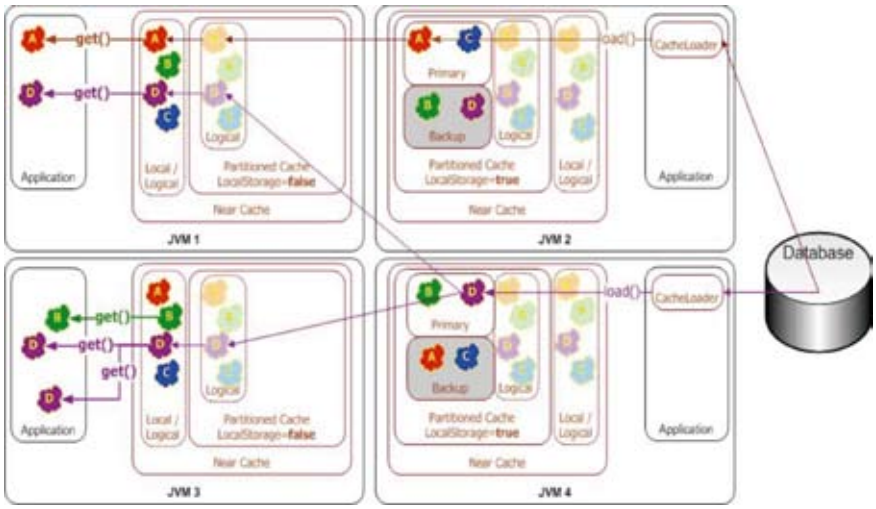


Abbildung 2: Coherence Read-Through Caching Pattern

Das Betriebsmodell von Coherence sieht sowohl die Integration mit einzelnen Applikationsservern vor, als auch den hybriden Einsatz mit dem Applikationsserver als Transaktions-Manager und dem Zusammenspiel mit mehreren Coherence-Knoten in einem Coherence Grid. Die Kombination von Coherence mit Applikationsservern bietet clusterweites Caching und Zustandsverteilung. Dabei werden verteilte Daten im Cache gehalten und eine unabhängige Zustandsverwaltung in heterogenen Java-Umgebungen sichergestellt. Der Coherence-Client kann sich dabei in einem Applikationsserver oder in einem Cluster von Applikationsservern befinden. Der zugeordnete Coherence-Cache-Server ist clusterweit organisiert und entlastet die Datenbank mittels Pufferfunktion, beispielsweise über „Read-Through Pattern" (siehe Abbildung 2) und „asychrone Write-Through Pattern" zur Datenbank. Mit dem Coherence-Datenmanagement wird eine Verbesserung der Applikationsleistung und deren Zuverlässigkeit erreicht.

Die Eigenschaften von Coherence umfassen die parallele Verarbeitung mit Daten und deren Verarbeitungsaffinität, parallele

Abfragen und Aggregation für Objekt-basierte Abfragen. Coherence unterstützt Replikation, Partitionierung, Read-through, Write-through, Write-behind und Refresh-ahead Caching-Mechanismen. Bei der Verwaltung von Daten über einen Cluster mit Coherence-Datenstrukturen wird ein sehr schnelles Objekt-Clustering-Verfahren verwendet. Die Kommunikation geschieht über das spezielle Tangosol Cluster Management Protocol (TCMP), welches über einen eigenen Serialisierungs-Mechanismus für den Netzwerk-Transport verfügt und UDP verwendet. Weiterhin benutzt Coherence das Portable Object Format (POF) für optimierte Objekt-Serialisierung in Java und kein generisches „Java Serializable Object Format". Die Coherence Daten- und Verarbeitungsdienste stehen vom lokalen Netzwerk bis zum Wide-Area-Netzwerk (WAN) zur Verfügung und beinhalten ein Failover- und Recovery-Management sowie das Transaktions-Management. Die Echtzeit-Ereignisüberwachung wird mit dem Listener Pattern (Cache-Event-Benachrichtigung) bereitgestellt. Coherence benutzt JAAS-basierte Authentifikation, Netzwerkdurchsatz-Encryption (NTE) sowie auf Java Management Extensions (JMX) basierendes Monitoring und Management über clustered JMX. Die Persistenzschicht von Coherence stellt über JPA die Integration mit Oracle TopLink und anderen Objekt-relationalen Mapping-Technologien her. Oracle Coherence ist mit Oracle WebLogic Server und anderen Java-EE-Applikationsservern kombinierbar. Weiterhin gibt es ein Coherence-API zur Unterstützung für C++ und .NET.

Zusammenfassend bestehen die Vorteile von Oracle Coherence aus der extremen und vorhersehbaren Reaktionsleistung und einer nahezu unbegrenzten Anwendungsskalierbarkeit. Zudem erlaubt das Ressourcen-Management den Einsatz von Standard-Hardware zur Leistungssteigerung der Anwendungsplattform und maximiert die Ausnutzung von Ressourcen im Rechenzentrum mit dynamischen Service Level Agreements zur Gewährleistung einer besseren Service-Qualität innerhalb einer flexiblen Anwendungsplattform.

In der Java-Entwicklungsumgebung wird mit dem Code-Beispiel (siehe Listing 1) der NamedCache erzeugt, ein Wert mit „put" eingefügt und anschließend überprüft, welche Werte im NamedCache vorhanden sind.

```
package com.oracle.coherence.handson;

import com.tangosol.net.CacheFactory;
import com.tangosol.net.NamedCache;

public class MyFirstSample {
public MyFirstSample() {
}

public static void main(String[] args) {

// create or get a named cache called mycache
NamedCache myCache = CacheFactory.getCache(„mycache");

// put key, value pair into the cache.
myCache.put(„Name","Claudia Meier");

System.out.println(„Value in cache is „ + myCache.get(„Name"));
}
}
```

Listing 1: NamedCache erzeugen; Werte einfügen und überprüfen

```
package com.oracle.coherence.handson;

import com.tangosol.net.CacheFactory;
import com.tangosol.net.NamedCache;

public class MyFirstSampleReader {

public MyFirstSampleReader() {
}

public static void main(String[] args) {
// ensure we are in a cluser

CacheFactory.ensureCluster();
// create or get a named cache called mycache

NamedCache myCache = CacheFactory.getCache(„mycache");

System.out.println(„Value in cache is „ + myCache.get(„Name"));
}
}
```

Listing 2: Werte aus dem Cache auslesen

Fügt man eine weitere Java-Klasse „MyFirstSampleReader" hinzu und verwendet „get" anstelle von „put", so kann man die eingefügten Werte vom Cache auslesen (Listing 2).

Kontakt:
Wolfgang Weigend
Wolfgang.Weigend@oracle.com

Wolfgang Weigend ist als Systemberater für die Oracle Fusion Middleware bei der Oracle Deutschland B.V. & Co. KG zuständig für Architektur und strategischen Produkteinsatz bei Großkunden. Er verfügt über langjährige Erfahrung in der Systemberatung und im Bereich objektorientierter Softwareentwicklung mit Java. Davor war er als Principal Systems Engineer 9 Jahre bei der BEA Systems GmbH.

OSGi-Scripting mit Groovy

Dirk Mahler, buschmais GbR

Laufzeit-Management besitzt für JEE-Anwendungen eine hohe Bedeutung. Klassische Application-Server bieten dafür neben grafischen Oberflächen mächtige Scripting-Schnittstellen an. Für OSGi als modulare JEE-Plattform ist es in diesem Punkt jedoch schlecht bestellt. Es existieren zwar neben Web- auch Telnet-basierte Konsolen, allerdings ist eine Automatisierung von Installationen und Konfigurationen so nur schwer zu erreichen. Der Artikel zeigt, wie MAEXO [1] und Groovy Abhilfe schaffen.

Folgendes Szenario soll verwirklicht werden: In einem leeren OSGi-Container ist eine neue Webanwendung zu installieren. Neben der eigentlichen Anwendung muss dafür ein Http-Service zur Verfügung gestellt werden. Dafür fallen folgende Management-Aktivitäten an:

- Abfrage des Container-Zustands
- Installation der notwendigen Bundles (Apache Felix HTTP Service [4], basierend auf Jetty)
- Konfiguration des zu verwendenden Ports (8085)
- Start des Bundles, welches den HttpService publiziert
- Überprüfung der Verfügbarkeit des Http-Service

Die Anwendung wird in einem Rechenzentrum gehostet, sodass kein direkter Zugang möglich ist und das Management per JMX abgewickelt werden muss. Glücklicherweise sind im OSGi-Container die entsprechenden MAEXO-Bundles bereits installiert und aktiviert.

Applikation und Werkzeugkasten

Zur Nachvollziehbarkeit des geschilderten Szenarios kann das mit MAEXO ausgelieferte Beispiel im Ordner „samples/basic" verwendet werden. Es stellt einen vorkonfigurierten Equinox- beziehungsweise Felix-Container zur Verfügung, der alle benötigten MBeans beinhaltet. Nach dem Start über das Shell-Skript kann man mittels JConsole eine Verbindung zum MBean-Server des Containers herstellen. Da die Anwendung im beschriebenen Szenario nicht lokal verfügbar ist, muss die Verbindung über eine JMX-Connection-URL erfolgen. Für das vorliegende Beispiel sieht diese URL folgendermaßen aus:

```
service:jmx:rmi:///jndi/rmi://
localhost:1099/jmxrmi.
```

Nach erfolgreichem Verbindungsaufbau erscheint im Reiter MBeans eine Domain „com.buschmais.maexo", unter der MAEXO verschiedene MBeans zur Verfügung stellt (siehe Abbildung 1).

Für das Erstellen und Ausführen der Admin-Skripte ist weiterhin eine Groovy-Installation notwendig. Das entsprechende ZIP-Archiv beziehungsweise ein Windows-Installer stehen auf der Homepage des Projekts [3]. Die ersten Management-Aktionen können direkt in der GroovyConsole ausgeführt werden. Diese Konsole ist in der Installation enthalten und wird über ein Shell-Skript im $GROOVY_HOME/bin-Verzeichnis gestartet.

Der erste Kontakt

Zunächst muss eine Verbindung zum entfernten MBean-Server aufgebaut werden. Dazu wird die zuvor genannte JMX-Connection-URL benötigt (siehe Listing 1).

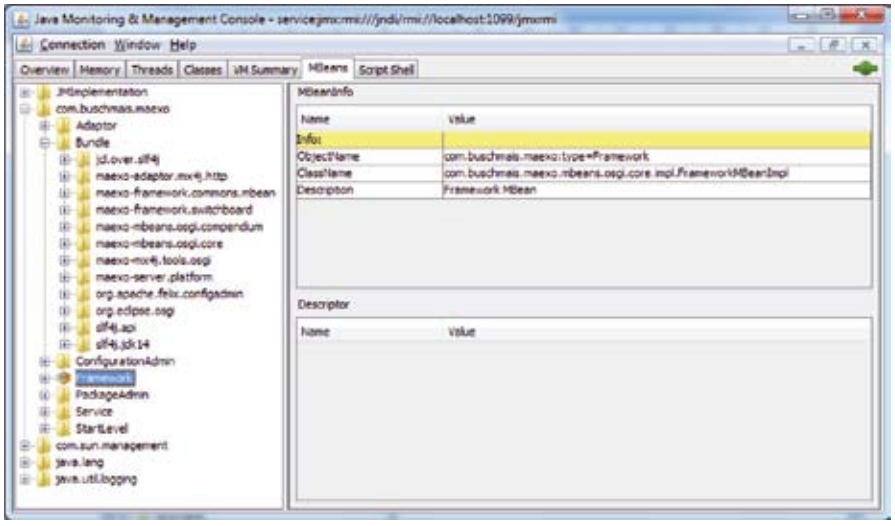


Abbildung 1: JConsole mit MAEXO-MBeans

```
url = new javax.management.remote.JMXServiceURL(
   ,service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi‘);

con = javax.management.remote.JMXConnectorFactory
   .connect(url).MBeanServerConnection;

con.getAttribute(new javax.management.ObjectName(
   ‘com.buschmais.maexo:type=Framework’),’vendor’);
```

Listing 1

Nach dem Ausführen dieser Anweisungen in der GroovyConsole ([Strg+R]) zeigt Groovy den letzten Rückgabewert an, in diesem Fall den Namen des Herstellers des OSGi-Containers. Dieser steht durch ein Attribut einer MBean zur Verfügung, welche das OSGi-Framework repräsentiert und über den Namen „buschmais.maexo:type=Framework" erreichbar ist.

Es besteht die Möglichkeit, eine GroovyConsole direkt innerhalb der JConsole auszuführen. Unter [5] ist eine entsprechende Anleitung zu finden. Daraus ergeben sich zwei Vorteile: Zum einen kann im Skript die bereits existierende Verbindung zum entfernten MBean-Server genutzt werden:

```
con = plugin.context.MBeanSer-
verConnection;
```

Andererseits kann man innerhalb der JConsole je nach Anwendungsfall beziehungsweise Nutzer-Präferenz durch einfaches Wechseln des Reiters mit der GUI oder mit der Script Shell arbeiten. Ein wichtiger Unterschied zur Groovy-eigenen Konsole besteht darin, dass die Eingaben unmittelbar nach jeder Zeile interpretiert werden – das kann situationsabhängig von Vor- oder Nachteil sein.

Kleines Werkzeug, große Wirkung

Als Nächstes soll die Menge installierter Bundles abgefragt werden. Unter Nutzung der verfügbaren Java-APIs und der durch MAEXO bereitgestellten Framework-MBean sieht dies wie folgt aus:

```
con.getAttribute(
   new javax.management.Object-
Name(
      ,com.buschmais.
maexo:type=Framework‘),
   ,bundles‘);
```

Leider liefert diese umständliche Anweisung einen für den Nutzer unbrauchbaren Rückgabewert: Es wird ein ObjectName[]-Array anderer MBeans zurückgeliefert, das die Menge installierter Bundles repräsentiert. Zur Vereinfachung ist daher eine Hilfsmethode erforderlich, welche aus einem übergebenen ObjectName einen Proxy erzeugt, der die entsprechende MBean im OSGi-Container repräsentiert:

```
def asMBean(objectName) {
   return new groovy.util.
GroovyMBean(con, objectName);
}
```

Damit können Attribute und Methoden in objektorientierter Art verwendet werden. Die Abfrage des Framework-Herstellers beziehungsweise der installierten Bundles gestaltet sich folgendermaßen:

```
framework =
asMBean(,com.buschmais.
maexo:type=Framework‘);
println framework.vendor;
framework.bundles.each {
   bundle = asMBean(it);
   println
it.getKeyProperty(,name‘) + , ,
+ bundle.stateAsName;
}
```

Die letzte Anweisung ermittelt das ObjectName-[]Array der MBeans, das die installierten Bundles repräsentiert. Mittels „each{…}" wird darüber iteriert und für jedes Element der Symbolic-Name und der jeweilige Zustand des Bundles ausgegeben.

Vier Bündel und ein Service

Als nächste Aufgabe steht die Installation und Konfiguration der Bundles des Apache Felix HttpService auf der Agenda. Dies ist mithilfe des erzeugten MBean-Proxys „framework" und der von ihm bereitgestellten Methode jetzt einfach zu realisieren. Es werden der Einfachheit halber Artefakte aus einem öffentlichen Maven2-Repository verwendet (siehe Listing 1). Die letzte Anweisung ermittelt den Zustand des Bundles, das den HttpService bereitstellt. Unmittelbar nach der Installation ist dies „INSTALLED" (siehe Listing 2).

Für die gewünschte Konfiguration des Http-Service ist eine Instanz des ConfigurationAdmin-Services erforderlich; entsprechende MBeans werden von MAEXO zur Verfügung gestellt. Um diese zu erhalten, muss man die MBeanServer-Verbindung abfragen:

```
con.queryNames(new javax.ma-
nagement.ObjectName(
   ,com.buschmais.maexo:type=Co
nfigurationAdmin,id=*‘),null);
```

Ergebnis ist ein ObjectName-Set, zum Beispiel [com.buschmais.maexo:type=ConfigurationAdmin,id=78]. Der erhaltene ObjectName – genaugenommen seine String-Repräsentation – wird verwendet, um einen MBean-Proxy für diesen Service („cfgAdmin") zu erzeugen. Mithilfe des ConfigurationAdmin-MBeans kann man nun auf die Konfiguration des Jetty-Bundles ("cfg") zugreifen.

```
cfgAdmin=asMBean(
   ,com.buschmais.maexo:type=Co
nfigurationAdmin,id=78‘);

cfg=asMBean(
   cfgAdmin.
getConfiguration(,org.apache.
felix.http‘,

jettyBundle.location));
```

```
framework.installBundle(„http://repo2.maven.org/maven2/“ +
   „org/apache/felix/org.apache.felix.log/1.0.0/“ +
   „org.apache.felix.log-1.0.0.jar“);

framework.installBundle(„http://repo2.maven.org/maven2/“ +
   „org/apache/felix/org.apache.felix.http.api/2.0.4/“ +
   „org.apache.felix.http.api-2.0.4.jar“);

framework.installBundle(„http://repo2.maven.org/maven2/“ +
   „org/apache/felix/org.apache.felix.http.base/2.0.4/“ +
   „org.apache.felix.http.base-2.0.4.jar“);

jettyObjectName = framework.installBundle(
   „http://repo2.maven.org/maven2/“ +
   „org/apache/felix/org.apache.felix.http.jetty/2.0.4/“ +
   „org.apache.felix.http.jetty-2.0.4.jar“);

jettyBundle = asMBean(jettyObjectName);
jettyBundle.stateAsName;
```

Listing 2

Kurzportrait MAEXO

MAEXO (Management Extensions for OSGi) ist ein Open-Source-Framework, welches das Laufzeitmanagement von OSGi-basierten Anwendungen ermöglicht. Es stellt sowohl die Infrastruktur zur Integration der Java Management Extensions (JMX) in den OSGi-Container als auch entsprechende Managed Beans (MBeans) zur Verwaltung häufig genutzter OSGi-Ressourcen zur Verfügung. Dazu zählen beispielsweise installierte Bundles und laufende Services. Diese können so unter Zuhilfenahme beliebiger JMX-Management-Konsolen von entfernten Rechnern überwacht bzw. gesteuert werden.

Darüber hinaus stellt MAEXO ein Programmiermodell zur Verfügung, das die Implementierung eigener MBeans samt Verwaltung ihres Lebenszyklus unterstützt. Der grundlegende Ansatz besteht in der Nutzung der Service-Infrastruktur des OSGi-Containers und der transparenten Registrierung aller MBeans und MBean-Server als OSGi-Services.

Das Setzen der Konfigurationsparameter erfolgt daraufhin am MBean-Proxy der Konfiguration; der Aufruf der Methode „update" publiziert diese Konfiguration:

```
cfg.setString(‚org.osgi.service.http.port', ‚8085');
cfg.update();
```

Als letzte Schritte stehen lediglich das Starten des Jetty-Bundles sowie die Überprüfung des Vorhandenseins des HttpService aus:

```
jettyBundle.start();
jettyBundle.registeredServices.each {
   println it.getKeyProperty(‚name');
}
```

Fazit

Der skizzierte Anwendungsfall hätte problemlos über eine grafische Oberfläche – wie sie JConsole & Co. anbieten – abgearbeitet werden können. Die Korrektheit und Nachvollziehbarkeit der solcherart vorgenommenen Management-Aktivitäten hängt dabei stark vom Konzentrationsvermögen des Anwenders ab: Es kann nur bedingt garantiert werden, dass auf verschiedenen Instanzen in unterschiedlichen Umgebungen wirklich die gleichen Aktionen ausgeführt werden. Abhilfe ist durch die Verwendung vordefinierter Skripte und den damit verbundenen Automatisierungsgrad zu erreichen. Begriffe wie „Reproduzierbarkeit" und „Testbarkeit", die in anderen Bereichen der Softwareentwicklung zum Standardvokabular zählen, halten damit Einzug im Bereich des OSGi-Laufzeitmanagements. Für komplexere als das geschilderte Szenario ist dies eine unverzichtbare Voraussetzung. Die Basis dafür stellt das Vorhandensein entsprechender Werkzeuge dar. Im vorliegenden Beispiel sind dies Groovy und die durch MAEXO bereitgestellten MBeans.

Quellen

[1] MAEXO, http://maexo.googlecode.com
[2] VisualVM, https://visualvm.dev.java.net/
[3] Groovy, http://groovy.codehaus.org/
[4] Apache Felix HTTP Service, http://felix.apache.org/site/apache-felix-http-service.html
[5] Groovier jconsole!, http://blogs.sun.com/sundararajan/entry/groovier_jconsole

Kontakt:
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 Softwarearchitektur auseinander, kann dabei eine Vorliebe für den Aspekt der Persistenz nicht verleugnen. Er ist Gesellschafter der buschmais GbR, einem Beratungshaus mit Sitz in Dresden.

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,
DOAG Dienstleistungen GmbH

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

Mediadaten und Preise finden Sie unter:
www.ijug.eu/images/vorlagen/2011-Publ-DOAG_Mediadaten_2011.pdf

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

Von Oracle Forms auf Java

Hans Niedermeier, Pro-Software GmbH

Nach über zwanzig Jahren Oracle Forms (und Designer-)Praxis war für den Autor eine Neuorientierung innerhalb der Anwendungsentwicklung notwendig. Der Weg führte über die visuelle Entwicklungsumgebung XDEV 3 zu Java.

Seit Anfang der 1990er Jahre bietet Oracle mit der Designer- und Developer-Schiene eine starke Plattform für 4GL-Entwicklung. Diese Kombination war deshalb bei Anwendern so beliebt, weil mit Designer generierte Forms wegen des einheitlichen Code-Aufbaus auch nach Jahren gut pflegbar sind. Die Nutzbarkeit von einmal erzeugten Forms-Anwendungen auf vielen gängigen Plattformen ist garantiert. Das Konzept, jeder Base-Table einen Forms-Block zuzuordnen und mehrere Blöcke über Foreign-Keys der Base-Tables sauber synchronisieren zu können, zieht sich durch alle Forms-Programme. Ist eine Base-Table einem Block zugeordnet, sorgt Forms auf generische Art selbst für die Definition und Ausführung von Select-, Insert-, Update- und Delete-Operationen. Die in allen Formularen und Table-Objekten einheitlich zur Verfügung stehenden Shortcut-Tasten (Query, Insert, Update Commit etc.) machen das Arbeiten für intensive Forms-Anwender sehr effektiv. Transaktionssicherheit, Daten-Locking und Multi-Row-Operationen gehören ebenso zu den Key-Features von Forms, auf die Entwickler setzen können ohne dafür Code schreiben zu müssen. Über Menü-Steuerung und Nutzung globaler Variablen ist eine Integration vieler Forms-Masken mit automatischer Datengewinnung beim Aufruf innerhalb einer Anwendung möglich. Eine sehr umfangreiche Property-Palette macht es möglich, alles Erdenkliche für das Look&Feel von GUI-Objekten definieren zu können. Das zentrale Sprachmittel für das Lösen betriebswirtschaftlicher Aufgaben ist PL/SQL, das direkt in Forms formuliert werden kann. Besser ist es jedoch, diese Schicht in Form von Packages direkt in die Datenbank zu verlagern.

Die Schwäche von Forms

Alles in allem bietet Forms eine sehr performante Plattform, speziell für Multi-Row-Bearbeitungen. Mit den neueren Versionen ist auch der Betrieb unter Webservern Standard. Zu den Schwächen von Forms aus Sicht des Autors zählt vor allem das GUI. Die in modernen grafischen Oberflächen geforderten Tree-Controls (Baumstruktur) sind mit Forms kaum oder nur mit erheblichem Aufwand und gleichzeitig geringer Leistungsfähigkeit erstellbar. Tree-Controls in Forms kommen daher praktisch nicht vor. Somit entfällt in Forms-Anwendungen

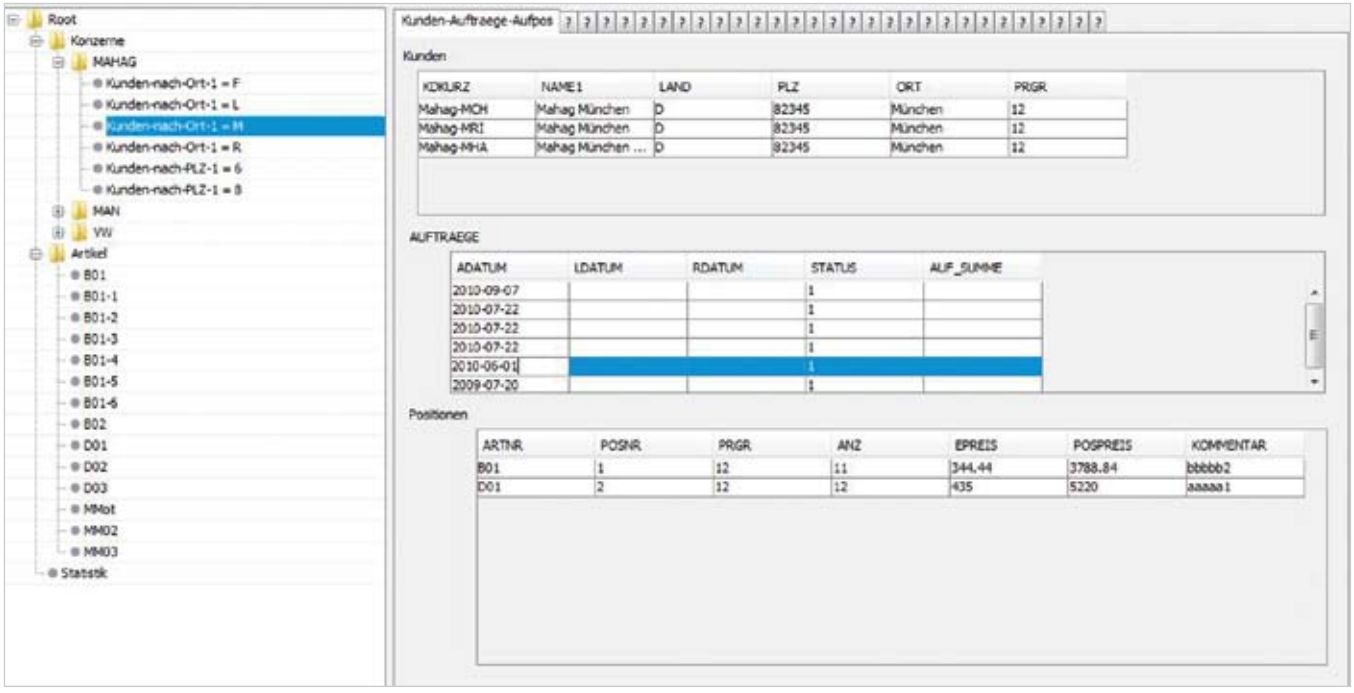


Abbildung 1: XDEV-Treecontrol und ein XDEV-Tab25 mit Pivot-Darstellung jener Daten, die mit den Anweisungen aus Anlage-1 zur Runtime gewonnen und dargestellt werden.

die Möglichkeit, sich mithilfe gut strukturierter Bäume die Daten einer Anwendung zu erschließen. Stattdessen muss man massiv mit Suchmasken und Suchvorgängen operieren.

Werden in größeren Projekten mit mehreren Entwicklern Anwendungen ohne Designer erstellt (leider ist die Anwendungs-Generierung mit Designer praktisch gestorben), kann man nur mit strengen Programmierregeln erreichen, dass der Forms-Code nicht zu sehr mit individuellen Stilen befrachtet wird. Es erfordert erheblichen Aufwand, über Templates und Libraries eine gewollte einheitliche Standardfunktionalität und Codebasis bereitzustellen. Forms kann seine Stärken jedoch nur in Verbindung mit Oracle-Datenbanken ausspielen.

Rapid Application Development ist mit Forms grundsätzlich möglich, jedoch auf die Gefahr hin, dass schwer wartbarer Code entsteht.

Für den Autor war der Umstieg in die Java-Welt bereits mit XDEV 2 sehr einfach zu schaffen. Die Umstellung der Version 3 auf pures Java macht die Java-Migration für Forms-Entwickler nochmals leichter. Die im GUI-Builder zur Verfügung stehenden Werkzeuge sind dem Forms-Designer in vielem verblüffend ähnlich. Analog zu den umfangreichen Property-Paletten in Forms stehen zu allen Objekten vielfältige Methoden innerhalb des XDEV Application Frameworks (XAPI) bereit, um alle gewünschten Features einer Datenbank-Anwendung erstellen zu können. Anders als in Forms müssen sich Java-Entwickler jedoch auch hier selbst um Transaktionssicherheit kümmern und Datensatzkonsistenz per Change-Numbers (SCN in Oracle-Tables oder ähnliches) selber implementieren. In Kürze soll jedoch eine spezielle Transaktions-Klasse eingeführt werden, die auch Fehlerbehandlungen innerhalb einer Transaktion ermöglichen soll.

Objektorientierung nicht zwingend notwendig

Wie mit jeder Java-IDE kann man natürlich auch in XDEV 3 objektorientiert entwickeln und eigene Klassen schreiben. Dies ist jedoch nicht zwingend notwendig. Auch rein prozedurale Programmierung ist möglich, was vor allem Forms-Entwickler, die in der Regel kaum über Erfahrung mit objektorientierter Programmierung verfügen, sehr entgegen kommt. Businesslogik lässt sich auch überwiegend in die Datenbank verlagern. Mit den auf JDBC aufsetzenden Datenbank-Schnittstellen (Nutzung von ODBC-Quellen möglich) kann man nahezu grenzenlos Daten aus verschiedenen Datenbanksystemen miteinander in Bezug setzen, innerhalb einer GUI repräsentieren und auch verarbeiten.

Unter Nutzung der Assistenten kann eine Anwendung mit überraschend wenigen Anweisungen erstellt werden. Für anspruchsvollere Applikationen und spezielle Features sollte man sich eine einheitliche Code- und Methodenbasis sowie eine eigene GUI-Bean-Palette schaffen. Anders als bei Forms generiert XDEV 3 keine fertigen Masken, sondern überlässt dem GUI-Designer die individuelle Gestaltung. Die Persistierung der Daten ist jedoch mit nur einem einzigen Methodenaufruf möglich. Für das Data Binding sind sogenannte „virtuelle Tabellen" vorgesehen. Sie sind Resultset-Objekt und gemäß Model-View-Control (MVC) -Pattern eine Art Universal-Model für sämtliche GUI-Komponenten zugleich. Damit ermöglichen diese virtuellen Tabellen eine effektive und einfache Art der Datenverarbeitung und Visualisierung, ohne dass der Entwickler dafür viel Code schreiben und sich mit MVC-Programmierung auseinandersetzen muss. Virtuelle Tabellen sind zudem auch als Steuerungselement möglich.

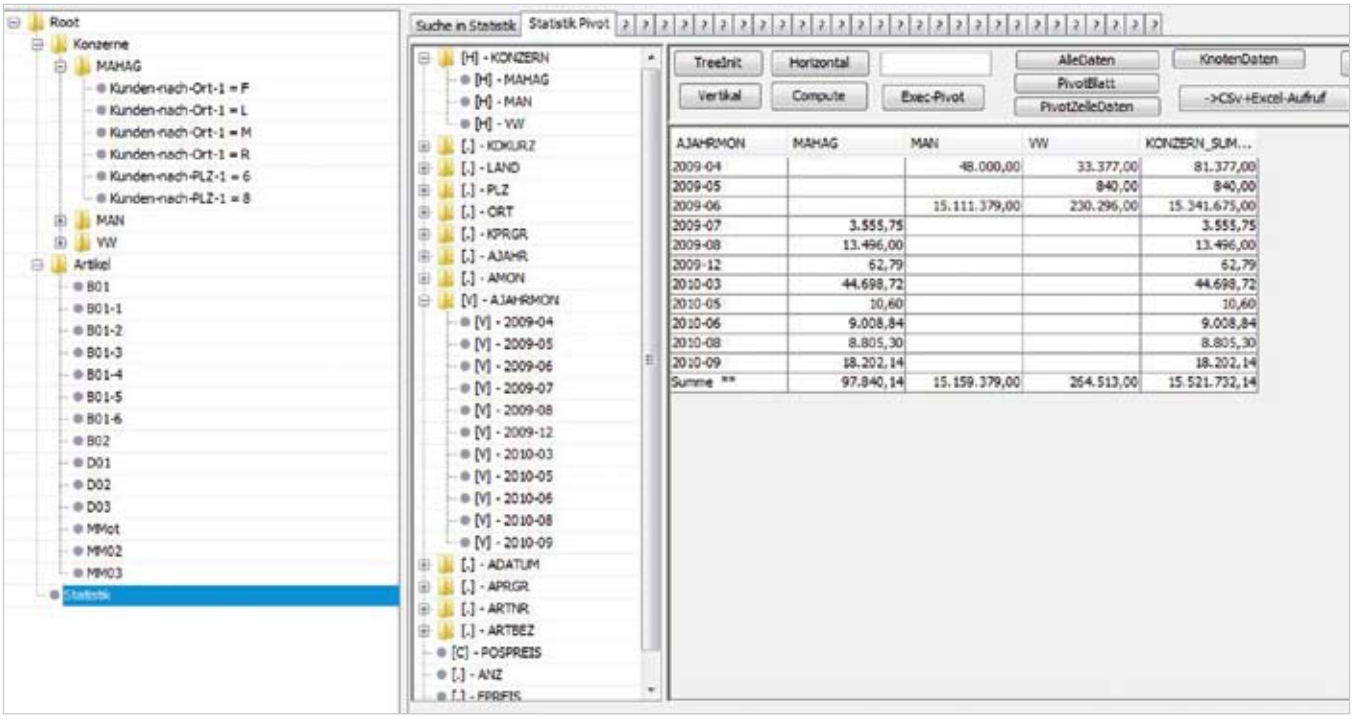


Abbildung 2: XDEV-Treecontrol und eine XDEV-Tab mit klassischer Master-Detail-Detail Ansicht, die mit dem Forms-Assistenten aufgrund von Foreign-Key-Constraints mit wenigen Eingriffen generierbar wäre.

Sehr angetan war der Autor von dem leistungsfähigen Tree-Control, das Java Swing standardmäßig zur Verfügung stellt, in Verbindung mit den angebotenen Füllmethoden dafür. Anstatt, wie in Swing üblich, komplizierte Tree-Models aufbauen zu müssen, ist es hier mithilfe virtueller Tabellen und etwas eigener Logik sehr leicht möglich, ein Tree-Control zur Laufzeit zu erzeugen und dabei statische Menüeinträge und variable Daten in Form dynamisch generierter Tree Nodes in beliebiger Tiefe und Variabilität zu mischen. Auf diese Weise lassen sich allein mit SQL-Mitteln Tree-Navigationen mit Daten aus beliebigen Tabellen und Views losgelöst von starren Bindungen in einem ER-Modell generieren, die sich während des Navigierens dynamisch erweitern. Alle Benutzeraktionen, die über die jeweiligen Tree Nodes aufgerufen werden, können ausgelagerten Code ausführen, wobei es sich um SQL, PL/SQL, Stored Procedures, System-Aufrufe sowie um beliebige Java-Klassen handeln kann.

Wiederverwendbare Komponenten

Für Forms-Entwickler erfreulich ist, dass sich entsprechende GUI-Objekte mit interner Funktionalität wie Synchronisierung, Refreshing und mehrsprachiger Beschriftung auch ohne Objektorientierung als GUI-Bean erstellen lassen. Diese lassen sich anschließend nicht nur global im gesamten Projekt, sondern auch in anderen Anwendungen mehrfach einsetzen.

Die Architektur von Swing und die gelungene GUI-Bean-Integration machen entsprechende Ableitungen und Abstraktionen auf frappierend einfache Weise möglich. So muss beispielsweise eine klassische Master-Detail-Darstellung zweier Tabellen, wie sie in Forms häufig verwendet wird, nur einmal als GUI-Bean realisiert werden und kann anschließend vielfach mit unterschiedlichen Daten (Views) eingesetzt werden.

Schon in XDEV 2 war es möglich, externe Java-APIs einzubinden. Mit der XDEV-3-Plattform werden diese Möglichkeiten noch wesentlich erweitert, sodass grundsätzlich alles möglich ist, was Java kann.

Fazit

Java mit XDEV 3 bietet deutlich mehr Möglichkeiten als Forms, moderne GUIs agil und schnell zu designen. Auch bei der Integration unterschiedlicher Datenquellen innerhalb einer Applikation ist die Entwicklungsumgebung Oracle Forms weit überlegen. Geht es um Transaktionsverhalten und -steuerung oder um Multi-Row-Verarbeitung, dann bietet Forms wesentlich mehr. In XDEV 3 muss man sich um diese Themen derzeit noch selber kümmern. Dass vieles in XDEV 3 nicht ganz so komfortabel ist wie in Forms, liegt zum einen an Java selbst und zum anderen daran, dass XDEV 3 nicht ausschließlich für die Oracle-Datenbank konzipiert, sondern als Java-IDE Datenbank-unabhängig ausgerichtet ist.

Alle Forms-Anwendungen, bei denen die Business-Logik konsequent als Stored Objects in die Datenbank verlagert wurde, können mit vertretbarem Aufwand im GUI-Builder von XDEV 3 überraschend schnell nachgebildet und verbessert werden. Datenbank und Geschäftslogik kann man 1:1 weiterverwenden. Mit der neuen Version 3 ist aus Forms-Anwender-Sicht der ideale Zeitpunkt gekommen, um den Umstieg auf Java anzugehen. Java-Migrationen erst dann anzugehen, wenn Oracle den Support für Forms eines Tages doch einstellt, ist fahrlässig und nicht zu empfehlen. Der von XDEV Software angekündigte AJAX-GUI-Builder auf Basis von Google Web Toolkit wird auch im Forms-Lager für noch mehr Interesse sorgen.

Kontakt:
Hans Niedermeier
hn@dap5.de

Hans Niedermeier arbeitet seit Ende der 1980-er Jahre mit Oracle-Datenbanken (beginnend mit Version 4.1.4) sowie von Anfang an mit der Oracle Designer-Schiene und war einer der ersten Mitglieder der DOAG Deutsche ORACLE-Anwendergruppe e.V.
Seit zwei Jahren beschäftigt sich Hans Niedermeier intensiv mit der Rapid Java Entwicklungsumgebung XDEV 2, mit der er den Umstieg von Oracle Forms auf Java erfolgreich vollziehen und seinen eigenen RAD-Konzepte umsetzen konnte. Schwerpunkt ist dabei die Leistungsmerkmale von Anwendungen in der Datenbank zu definieren, um möglichst kompakten Code erstellen zu können. Java und nun XDEV 3 mit dem XDEV Application Framework sorgen für die nötige Infrastruktur für diese Art der Anwendungsentwicklung.

Vorschau

Java aktuell – Sommer 2011

Die nächste Ausgabe der Java aktuell erscheint am 4. Mai 2011. Bitte schreiben Sie uns an redaktion@ijug.eu, was Ihnen an dieser Zeitschrift gefallen hat und welche Inhalte Sie sich für kommende Ausgaben wünschen.

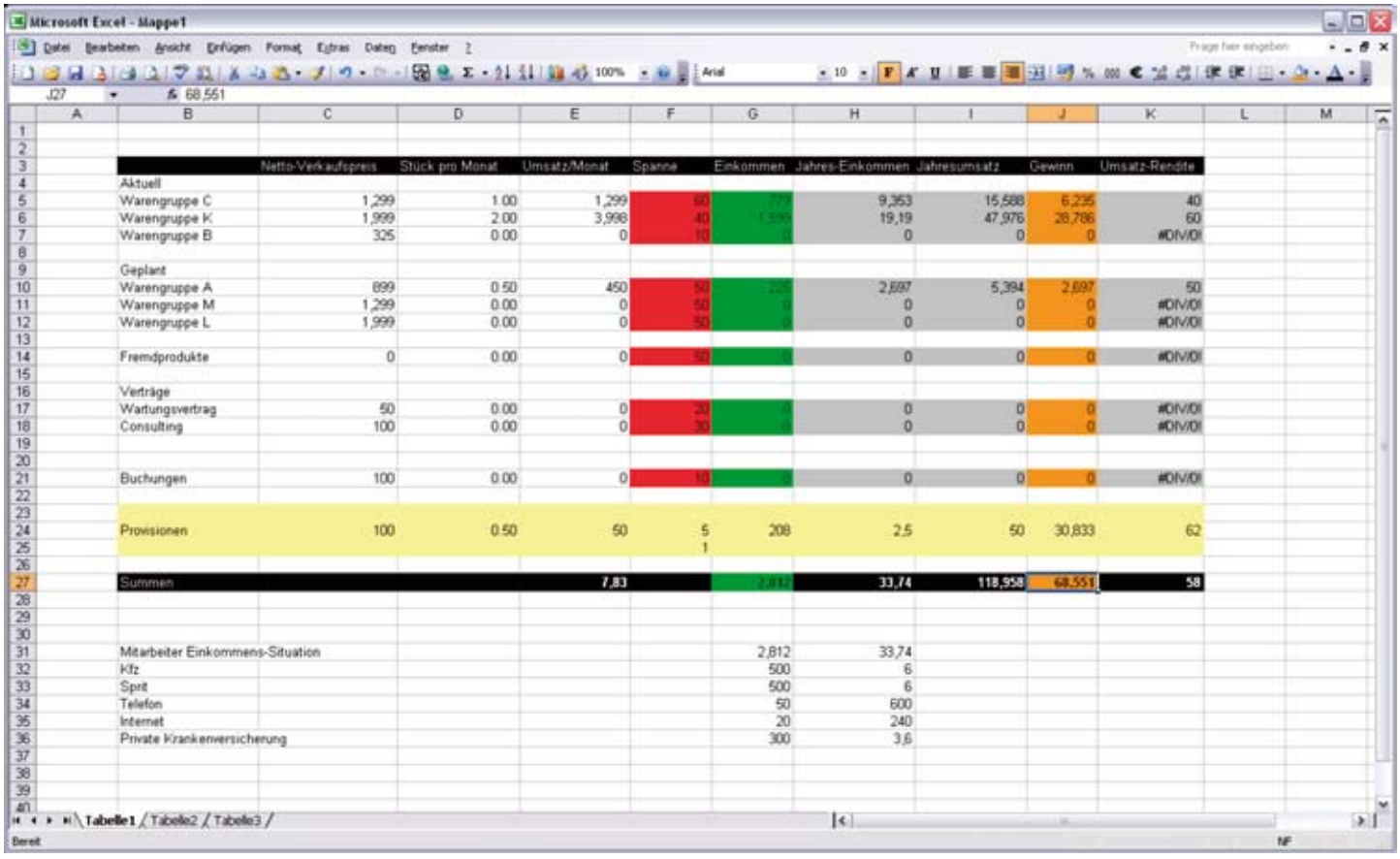
Falls Sie selbst einen Artikel in der nächsten Java aktuell veröffentlichen möchten, schicken Sie bitte vorab Ihren Themenvorschlag ebenfalls an redaktion@ijug.eu, Redaktionsschluss für den fertigen Text ist am 4. März 2011.

AJAX vs. Java & Co.

Markus Stiegler, XDEV Software GmbH

Der Markt für Rich Internet Applications (RIA) gilt längst als Softwaremarkt der Zukunft. RIAs sind Software-Anwendungen, die sich über das Internet aufrufen und in einem Web-Browser ausführen lassen. Dabei handelt es sich nicht um gewöhnliche Webseiten oder Portale, sondern um echte Anwendungsprogramme wie eine Kunden- oder Lagerverwaltung. Obwohl RIAs im Browser laufen, sehen sie aus und fühlen sich an wie konventionelle Desktop-Programme.

Mit RIAs ergeben sich vor allem für Unternehmenslösungen und Behörden-Software völlig neue Anwendungsmöglichkeiten, aber auch Einsparungspotenzial bei den Kosten. Bislang mussten Computerprogramme fest auf einem Rechner installiert sein. Das erfordert vom Hersteller die Produktion und Auslieferung eines Datenträgers, der normalerweise in einer Verpackung zusammen mit gedrucktem Handbuch ausgeliefert wird und hohe Kosten verursacht. Aber auch auf Anwenderseite ist der Aufwand für den Umgang mit konventioneller Anwendersoftware groß, denn die Software muss auf zahlreichen Unternehmensrechnern installiert, konfiguriert und manuell aktualisiert werden. Release-Wechsel verursachen erfahrungsgemäß großen Aufwand und Kosten; zudem stellen sie immer ein gewisses Risiko dar. Der Zugriff von außen ist mithilfe einer Terminal-Server-Lösung oder via VPN zwar grundsätzlich möglich, verursacht jedoch zusätzliche Kosten für Lizenzen oder für eine deutlich leistungsfähigere Infrastruktur. Die Anbindung mobiler Geräte ist praktisch nur mit einer zusätzlichen



	Netto-Verkaufspreis	Stück pro Monat	Umsatz/Monat	Spanne	Einkommen	Jahres-Einkommen	Jahresumsatz	Gewinn	Umsatz-Rendite
Aktuell									
Warengruppe C	1,299	1.00	1,299	60	779	9,353	15,588	6,235	40
Warengruppe K	1,999	2.00	3,998	40	1,599	19,19	47,976	28,786	60
Warengruppe B	325	0.00	0	10	1	0	0	0	#DIV/0!
Geplant									
Warengruppe A	899	0.50	450	50	225	2,697	5,394	2,697	50
Warengruppe M	1,299	0.00	0	50	0	0	0	0	#DIV/0!
Warengruppe L	1,999	0.00	0	50	0	0	0	0	#DIV/0!
Fremdprodukte	0	0.00	0	50	0	0	0	0	#DIV/0!
Verträge									
Wartungsvertrag	50	0.00	0	20	0	0	0	0	#DIV/0!
Consulting	100	0.00	0	30	0	0	0	0	#DIV/0!
Buchungen	100	0.00	0	10	0	0	0	0	#DIV/0!
Provisionen	100	0.50	50	5	208	2,5	50	30,833	62
				1					
Summen			7,83		2,812	33,74	118,958	68,551	58
Mitarbeiter Einkommens-Situation					2,812	33,74			
Kfz					500	6			
Sprit					500	6			
Telefon					50	600			
Internet					20	240			
Private Krankenversicherung					300	3,6			

Abbildung 1: Excel als klassisches Desktop-Programm wie man es seit vielen Jahren kennt.

Mobile-Version möglich, die im Normalfall jedoch nur auf bestimmten Devices lauffähig ist, für den Hersteller jedoch doppelten Entwicklungs- und Wartungsaufwand bedeutet, sodass viele Lösungsanbieter bis heute auf eine mobile Variante verzichten.

Revolutionäre Möglichkeiten mit RIAs

RIAs lasen sich dagegen über das Internet oder Firmennetz aufrufen und auf jedem beliebigen PC ausführen, auf dem ein Web-Browser installiert ist. Vorteile wie weltweite Verfügbarkeit, simple Verteilung, kein Installationsaufwand, ständige Aktualität, herkömmliche Internet-Bandbreiten sowie minimaler Administrationsaufwand – und somit deutlich niedrigere Administrationskosten – sind vor allem aus Unternehmenssicht geradezu revolutionär. Experten propagieren schon seit Jahren, dass Anwender in Zukunft nur noch einen Browser brauchen, um damit jede benötigte Anwendung über das Internet aufrufen und nutzen zu können. Lösungsanbieter würden die Software-Nutzung dann ähnlich wie Strom aus der Steckdose abrechnen. Mit RIAs, Software as a Service (SaaS) und Cloud Computing ist all das heute schon möglich und die Märkte dafür wachsen rasant.

Vor allem für Lösungsanbieter muss diese Entwicklung Signalwirkung haben. Denn durch RIAs steht der gesamte Software-Markt derzeit vor einem ähnlich dramatischen Umbruch wie einst nach der Ablösung von MS-DOS durch Windows. So gut wie jeder Software-Hersteller war durch Windows gegen Ende der 1980er Jahre gezwungen, seine praktisch über Nacht veralteten DOS-Lösungen mit Hochdruck auf moderne Windows-Anwendungen umzustellen. Entweder um dadurch schneller in den Markt zu kommen als die Konkurrenz, oder um mit Wettbewerbern Schritt halten zu können und ein massenhaftes Abwandern von Kunden zu verhindern. Doch selbst bis dahin marktbeherrschende Unternehmen haben die dringende Notwendigkeit einer Umstellung zu spät erkannt und sahen sich urplötzlich mit rasant einbrechenden Nutzerzahlen konfrontiert, verloren daraufhin schlagartig ihre Marktstellung, überlebenswichtigen Umsatz und Unternehmenswert und haben sich davon nie wieder erholt. Prominenteste Beispiele dafür sind die bis dahin populärste Datenbank dBASE von Ashton-Tate, die damals führende Textverarbeitung WordPerfect oder die Tabellenkalkulation Quattro Pro von Borland.

Wer clever ist, stellt jetzt um

Nachdem die Windows-Desktop-Anwendungen über zwanzig Jahre lang Status quo waren, beginnen Unternehmen jetzt damit, ihre Applikationen durch RIAs abzulösen oder setzen zumindest auf eine Doppel-Strategie. Vor allem die großen Anbieter wie SAP, Oracle und sogar der Desktop-Primus Microsoft arbeiten mit Hochdruck an RIA-Lösungen für ihr Business. Vorreiter Google bietet mit Google Docs sogar schon Office-Komponenten als RIA an, was OpenOffice und selbst Microsoft zu einer Web-Strategie zwingt, um

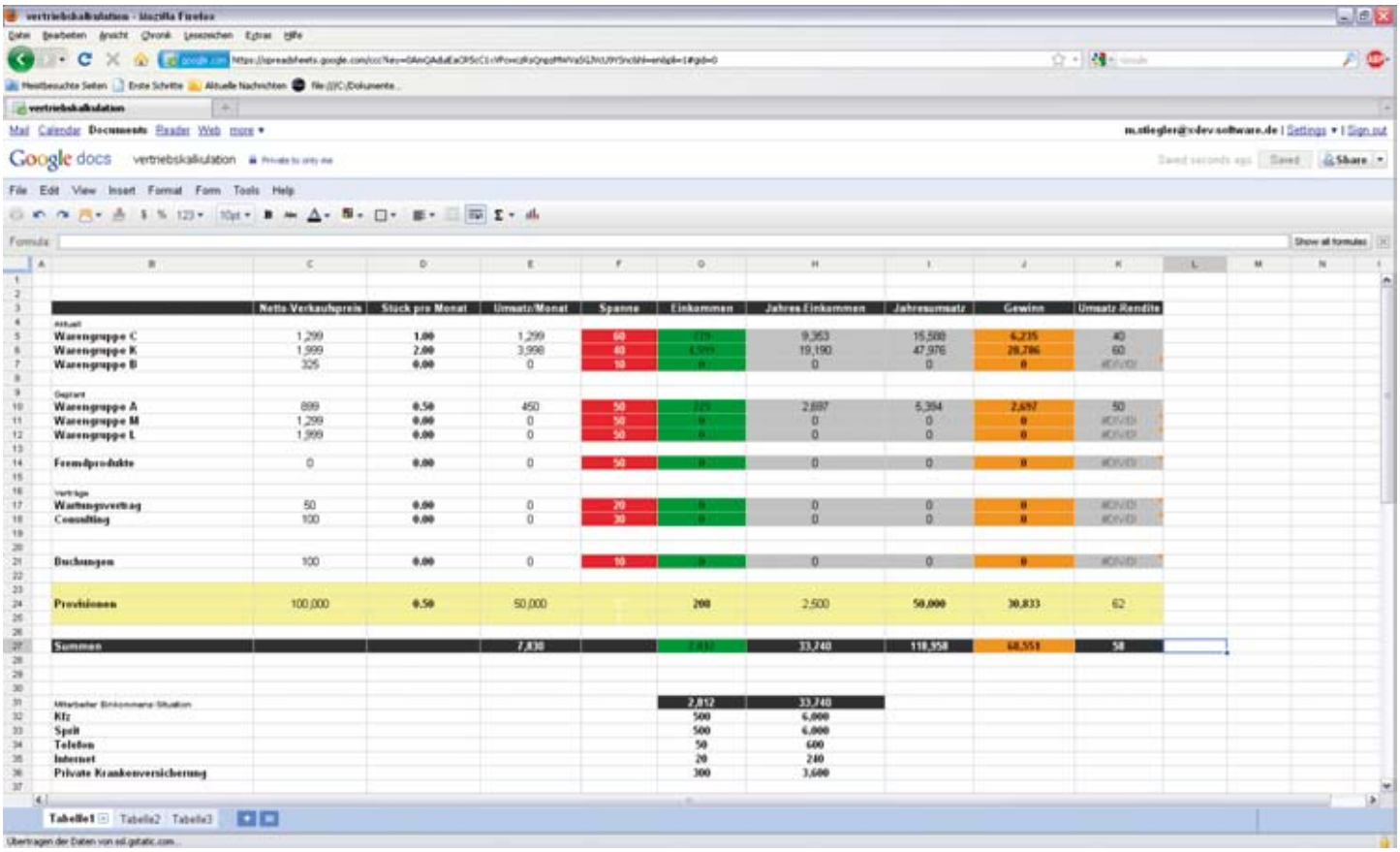


Abbildung 2: Google hat eine Tabellenkalkulation entwickelt, die nicht nur im Web-Browser läuft, sondern fast genauso aussieht und funktioniert wie Excel. Damit kann man mit jedem PC mit Internetzugang auf seine Dateien zugreifen und diese bearbeiten.

nicht die eigenen Anwender an Google zu verlieren. Das zeigt, wie wichtig es ist, den RIA-Trend nicht zu verpassen. Aber auch kleinere Anbieter sollten sich schnellstmöglich umstellen und können sich gerade jetzt einen wertvollen Vorsprung vor der noch schlafenden Konkurrenz verschaffen. Denn RIAs sind bereits so populär, dass mittlerweile so gut wie an jede neue Unternehmens- oder Behörden-Software der Anspruch gestellt wird, dass diese webfähig sein soll. Begünstigt wird dieser Trend durch immer leistungsfähigere mobile Geräte, mit denen das Betreiben anspruchsvoller Business-Applikationen längst möglich ist.

Die Entscheidung, ob man auf eine altbewährte Desktop-Lösung oder auf eine RIA-Strategie setzen soll, fällt vielen Herstellern schon nicht leicht. Wer sich dann für eine RIA entschieden hat, steht vor der schwierigen und strategisch entscheidenden Frage, auf welche RIA-Technologie man am besten setzen soll.

Im Markt für RIAs stehen sich grundsätzlich zwei völlig verschiedene Philosophien gegenüber, die miteinander konkurrieren. Auf der einen Seite steht AJAX (Asynchronous JavaScript and XML). Der Begriff umschreibt eigentlich nur die asynchrone Übertragung von Daten und Code zwischen Server und Browser mithilfe des XMLHttpRequests. Durch die revolutionären Möglichkeiten, die sich damit für das gesamte Internet und vor allem für die Anwendungsentwicklung ergeben, wurde der Begriff schnell zum Hype; er wird seither umgangssprachlich für alle Web-Anwendungen verwendet, welche AJAX als Übertragungstechnik benutzen. Die Basis von AJAX-Anwendungen ist jedoch ein Mix aus verschiedensten Web-Technologien.

Auf der anderen Seite stehen proprietäre, also herstellerspezifische und somit - anders als HTML - nicht offene RIA-Technologien. Dazu zählen Microsoft Silverlight, Java Applets und Java FX sowie Flash, das den meisten Anwendern als Animations-Tool

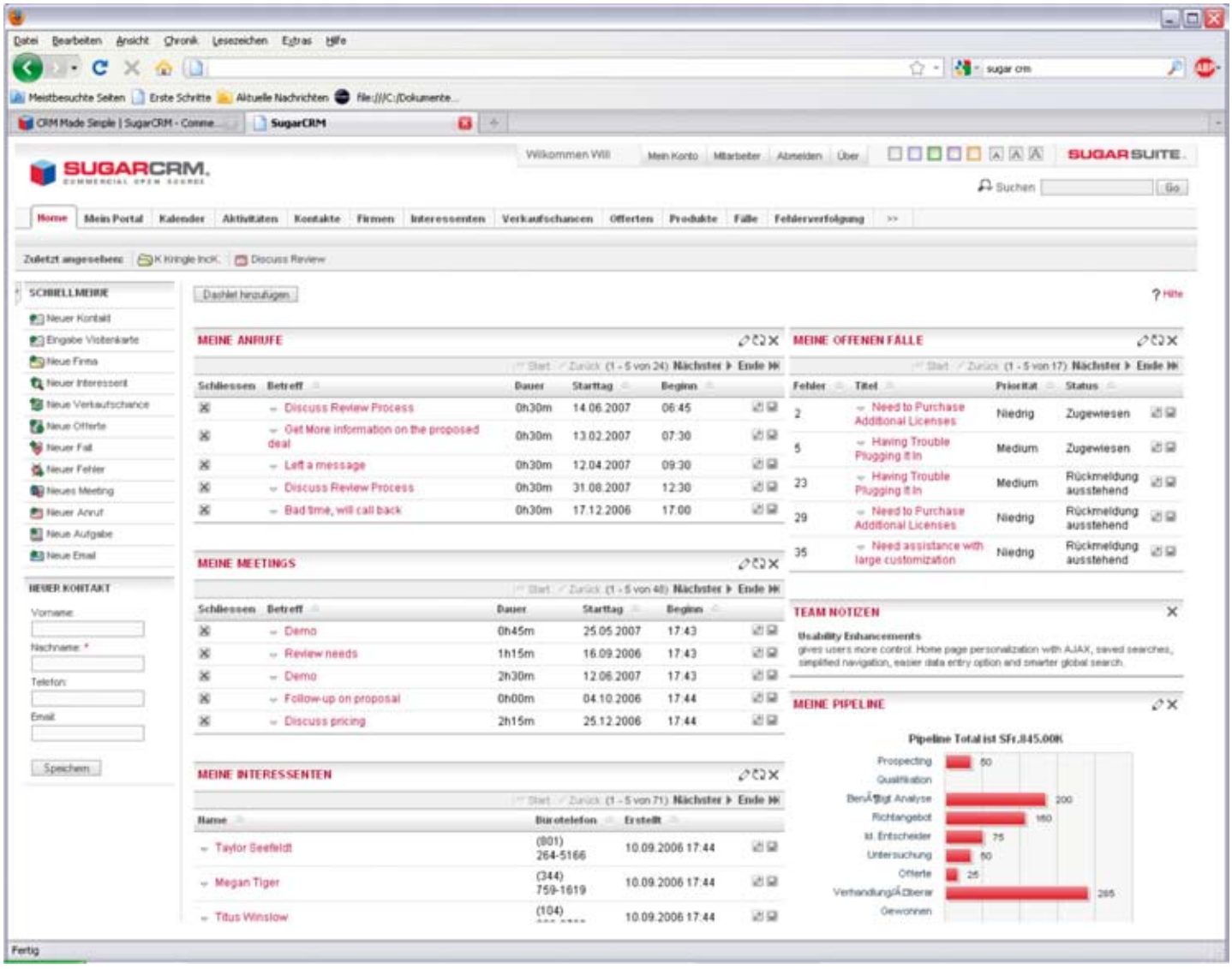


Abbildung 3: Unternehmen fordern immer stärker, dass ihre Mitarbeiter möglichst überall Zugriff auf wichtige Geschäftsanwendungen und Daten haben müssen. Dies gilt sowohl nur für Standardlösungen als auch für speziell entwickelte Individualsoftware. Lösungsanbieter sind dadurch gezwungen ihre Produkte möglichst schnell als webfähige RIA anzubieten. Auch immer mehr Inhouse-Software muss webfähig gemacht werden.

bekannt ist, zusammen mit Flex und Adobes neuer RIA-Plattform Air, die auch für die RIA-Entwicklung eingesetzt werden kann.

RIAs mit AJAX

Mit AJAX lassen sich Daten und Code nachladen, ohne dass der Browser die komplette Webseite neu aufbauen muss. Das führte schnell zu der Idee, dass man im Browser nicht nur gewöhnliche Webseiten anzeigen, sondern auch echte Software-Anwendungen ausführen könnte. Das große Problem dabei ist jedoch, dass ein Browser im Gegensatz zu Programmiersprachen wie Java in Sachen „grafische Oberfläche" außer Buttons, den wichtigsten Formular-Komponenten und Scroller rein gar nichts zu bieten hat. Tabellen lassen sich mithilfe von HTML-Tables gerade noch umsetzen, aber ein Tree-Control, Registerreiter und Menüs gibt es in HTML nicht, geschweige denn Drag&Drop, Kontextmenüs und Fenster-Technik. Das bedeutet, dass alles, was für eine moderne Benutzeroberfläche gebraucht wird und in jeder vernünftigen Programmiersprache zum Standardumfang zählt, im Sprachumfang von HTML nicht vorhanden ist und somit vom Browser auch nicht umgesetzt werden kann. Dies ist kein Wunder, da HTML ursprünglich für die Darstellung von formatiertem Text konzipiert wurde, niemals für die professionelle Anwendungsentwicklung.

Um dieses Defizit auszugleichen, müssen alle benötigten User Interface (UI) Controls in JavaScript nachprogrammiert werden. Diese können dann mithilfe der JavaScript-Engine des Browsers ausgeführt werden. Während sich Entwickler anfangs entsprechende Controls mit gigantischem Aufwand noch selber schreiben mussten, stehen mittlerweile zahlreiche JavaScript-UI-Libraries für die Oberflächen-Entwicklung wie jQuery, Rico, Dojo, qooxdoo oder die Yahoo!-UI-Library zur Verfügung. Bei allen handelt es sich um Open-Source-Bibliotheken, sodass auch der Einsatz in kommerziellen Anwendungen möglich ist. Die Qualität der Controls ist inzwischen so gut, dass man eine JavaScript-Oberfläche praktisch nicht mehr von einer Desktop-Applikation unterscheiden kann. Der Code einer kompletten JavaScript-Oberfläche ist jedoch sehr umfangreich und aus Optimierungsgründen oft kaum mehr lesbar. Für den Aufbau der Oberfläche wird normalerweise eine spezielle JavaScript-Engine beim Starten der Anwendung geladen, die unter anderem den Aufbau und Umbau der Oberfläche sowie das Nachladen von Daten und Code managt.

AJAX-Entwicklung ist aufwändig und teuer

Eines der größten Probleme bei HTML-/JavaScript-Frontends sind seit eh und je die nervtötenden Browser-Inkompatibilitäten, mit denen Entwickler trotz W3C-Standards wohl auch in Zukunft leben müssen, da sich die Browser-Hersteller nicht konsequent an die Standards halten und selbst die verschiedenen Versionen eines Browsers häufig nicht vollständig zueinander kompatibel sind. Dieser Umstand macht Web-Entwicklung unnötig aufwändig und teuer; er führt zu vergleichsweise kurzen Lebenszyklen einer Anwendung, wenn diese nicht permanent und zeitnah an aktuelle Browser-Updates angepasst wird. Mit dem Einsatz einer UI-Library wird die Wartung einer Anwendung deutlich leichter, da entsprechende Anpassungen nach Browser-Updates vom Framework-Hersteller erledigt werden müssen. Was dennoch bleibt, sind der Aufwand, das Library-Update einzubinden, manuelle Anpassungen bei erheblichen Änderungen und das erneute Deployen der Anwendung. Problematisch wird es jedoch, wenn das verwendete Framework nicht mehr zeitnah supportet oder schlimmstenfalls nicht mehr weiterentwickelt wird. Dies erlebt man bei Open-Source-Projekten leider häufig, insbesondere wenn führende Entwickler abgeworben werden, die Sponsoren des Projekts die finanzielle Unterstützung einstellen oder ihre Entwickler abziehen, wenn sich Interessengruppen vom Projekt abspalten oder zu einem anderen Projekt wechseln. Man sollte sich deshalb genau informieren, wer die treibenden Kräfte hinter dem Framework sind, wie groß die Community ist und wie die vergangenen Release-Zyklen aussehen.

Um Java-Entwickler vollständig vom Umgang mit HTML und JavaScript zu befreien, kommen immer mehr Frameworks auf den Markt, die das Schreiben der Oberfläche direkt in Java ermöglichen und sich anders als Java Server Pages (JSP) am Programmiermodell von Java Swing orientieren, während das Framework letztendlich HTML / JavaScript generiert, etwa Apache Wicket, GWT (Google Web-Toolkit), Vaadin oder wingS.

Während die Oberfläche einer AJAX-Anwendung im Browser dargestellt wird, läuft die komplette Programmlogik auf dem Server. Anders als das Frontend kann das Server-Programm mit jeder beliebigen Script- oder Programmiersprache wie PHP, Java oder mit einer .NET-Sprache geschrieben werden. Da die meisten Internet-Service-Provider LAMP-Systeme (Linux-Betriebssystem, Apache-Webserver, MySQL-Datenbank, PHP-Interpreter) zur Verfügung stellen und die Installation anderer Laufzeitumgebungen wie .NET oder Java nur auf dedizierten Servern erlauben, trifft man in der Praxis meist PHP-Anwendungen an.

Durch die Trennung von Client- und Serverteil handelt es sich bei einer AJAX-Anwendung faktisch um zwei autark voneinander laufende Programme, was erhebliche Probleme aufwirft. Unter anderem müssen Client und Server bei jedem Ereignis synchronisiert sein. Dies geschieht mithilfe des „Document Object Model" (DOM), das in beiden Anwendungsteilen alle Objekte auf einer Seite in Form einer Baumstruktur verwaltet. Das für die Registrierung von Events notwendige Polling darf nicht zur Überlastung des Servers führen, und die bei Anfrage und Antwort entstehenden Latenzzeiten müssen möglichst gering ausfallen. Darüber hinaus muss gelöst werden, wie die Anwendung mit Klicks auf die „Browser zurück"-Schaltfläche umgehen soll. Weitere Problematiken stellen Deep Links sowie Barrierefreiheit dar. Inzwischen gibt es Frameworks, die sich auch der Kommunikation zwischen Client und Server angenommen haben, sämtliche Probleme für den Entwickler lösen und die AJAX-Entwicklung damit nochmals erheblich vereinfachen, wie wingS und Vaadin.

AJAX-Kritiker führen an, dass Anwendungsentwicklung ohnehin schon kompliziert genug ist und große Business-Applikationen nur mithilfe durchgängiger Objektmodelle und vernünftiger Architekturen beherrschbar und wartbar bleiben. Aus Sicht erfahrener Entwickler ist es daher äußerst bedenklich, dass für die AJAX-Entwicklung nur Technologien zum Einsatz kommen, die wie HTML niemals für die Anwendungsentwicklung konzipiert wurden, während Java seit über zwanzig

Jahren genau dafür optimiert wird. Ebenso fatal sind die Browser-Inkompatibilitäten. Jeder Auftraggeber erwartet, dass eine Software über Jahre hinaus fehlerfrei und stabil läuft. Bei einer AJAX-Anwendung lässt sich dies jedoch nur bis zum nächsten Browser-Update garantieren. Wie viele Anpassungen man in Zukunft vornehmen muss, kann nicht vorhergesagt werden; die Kosten sind somit nur schwer kalkulierbar. Erschreckend ist, dass dies jedoch vielen Auftraggebern gar nicht bewusst ist. Für Software-Dienstleister ist das ein gutes Geschäft, für Auftraggeber eher ein Fass ohne Boden.

Sind Flash, Silverlight und Java die besseren RIA-Technologien?

Die Alternative zu AJAX sind Flash-, Silverlight- und Java-Frontends. Damit lässt sich die komplette Anwendung einschließlich des Browser-Frontends mit ein- und derselben Technologie entwickeln. Daraus ergeben sich jede Menge Vorteile. Dem Java-Entwickler beispielsweise stehen für die RIA-Entwicklung der komplette Sprachumfang von Java und somit alle Möglichkeiten einer ausgereiften Programmiersprache zur Verfügung. Für die Frontend-Entwicklung stehen leistungsfähige APIs wie Swing, SWT oder Java FX zur Verfügung, während HTML und JavaScript nicht mehr benötigt werden, sodass geübte Java Entwickler in der Lage sind, alle Vorgaben kompromisslos umsetzen. Darüber hinaus können die gewohnte Entwicklungs- und Testumgebung eingesetzt und Anwendungen vernünftig debugged werden. Ausgeführt werden Flash-, Silverlight- und Java-RIAs nicht vom Web-Browser, sondern von der jeweiligen Laufzeitumgebung, die als schlanke Browser-Plug-ins frei verfügbar sind. Flash-, Silverlight- und Java-RIAs laufen somit völlig Browser-unabhängig. Browser-Updates wirken sich damit anders als bei AJAX-Anwendungen nicht im Geringsten auf die Lauffähigkeit der Anwendung aus.

Mit dem Einsatz von Flash, Silverlight oder Java für die RIA-Entwicklung sind die Möglichkeiten zumindest derzeit noch größer als mit AJAX-Frameworks. Die Entwicklung ähnelt stark der Desktop-Entwicklung und ist im Vergleich zur AJAX-Entwicklung homogener und erheblich einfacher. Auch der Testaufwand ist deutlich geringer. Insgesamt wird man bei der RIA-Entwicklung mit Flash, Silverlight und Java kürzere Entwicklungszeiten erreichen und mit niedrigeren Entwicklungskosten kalkulieren können. Der durch Browser-Updates verursachte Wartungsaufwand ist gleich Null, sodass auch die Wartungskosten minimal sein werden. Durch die Browser-Unabhängigkeit ist eine lange Lebenszeit der Anwendung und somit Investitionssicherheit garantiert.

Aufgrund der zahlreichen Vorteile würden sich die meisten IT-Leiter wahrscheinlich schnell für eine RIA auf Basis von Flash, Silverlight oder Java entscheiden. Gegenüber AJAX haben alle proprietären Lösungen jedoch einen schwerwiegenden Nachteil: Auf dem Anwenderrechner muss die jeweilige Laufzeitumgebung installiert sein, oder der Anwender muss das entsprechende Plug-in downloaden. Dies ist jedoch nicht immer möglich. In vielen Firmen und Behörden ist Download und Installation von Anwendungen nicht möglich, etwa aus Sicherheitsgründen. Ebenso in Internet-Cafés, öffentlich zugänglichen Internet-Terminals, Hotels etc. Darüber hinaus sind die Laufzeitumgebungen für Flash, Silverlight und Java noch nicht für alle mobilen Geräte verfügbar oder werden von Geräteherstellern wie Apple nicht toleriert. Ohne Plug-in können Silverlight-, Java- und Flash-Anwendung jedoch nicht ausgeführt werden, sodass mit dem Einsatz einer proprietären RIA-Technologie immer bestimmte Benutzergruppen ausgeschlossen werden. Darüber muss sich der Betreiber der Anwendungen bewusst sein. Vor allem bei firmeninternen Anwendungen kann das kein Problem sein. Für B2C-Lösungsanbieter, die ihr Angebot auch oder sogar überwiegend an Konsumer richten, ist diese Einschränkung jedoch nicht akzeptabel, sodass dann nur AJAX als Basis-Technologie in Frage kommt.

Fazit

Wenn die benötigte Web-Anwendung nur von einem geschlossenen Anwenderkreis genutzt werden soll, wie z.B. ein Firmen-Intranet, die Anwendung nicht oder nur auf bestimmten mobilen Geräten lauffähig sein muss, die Anwender bereit sind, für die Nutzung ggf. auch ein Plug-in zu installieren und auf Entwicklerseite ausreichend Erfahrung mit Flash-, einer .NET-Sprache oder Java vorhanden ist, dann ist der Einsatz einer proprietären RIA-Technologie möglich und empfehlenswert. Der Entwicklungs- und Wartungsaufwand sowie die Entwicklungszeit sind damit leichter kalkulierbar, die Kosten und das Projektrisiko in den meisten Fällen geringer. Die Browser-Unabhängigkeit garantiert zudem minimalen Wartungsaufwand, einen langen Lebenszyklus der Anwendung und langfristige Investitionssicherheit.

Soll die Anwendung dagegen für möglichst viele, teilweise oder gänzlich unbekannte Nutzer verfügbar sein, wie beispielsweise ein Shop-System oder ein offenes Social Network, und muss man davon ausgehen, dass nicht alle Anwender in der Lage oder bereit sein werden, für die Nutzung extra ein Plug-in herunterzuladen, und die Anwendung zudem auf möglichst vielen Mobilgeräten lauffähig sein soll, dann führt kein Weg an AJAX vorbei. Die Entwickler sollten dann möglichst genau prüfen, auf welche AJAX-Frameworks man setzt. Die Projektverantwortlichen sollten dagegen neben dem größeren Entwicklungsaufwand auch den vielfach größeren Wartungs- und Änderungsaufwand aufgrund zukünftiger Browser-Updates ermitteln und unbedingt einkalkulieren.

Kontakt:
Markus Stiegler
info@xdev-software.de

Markus Stiegler ist Technologie-Evangelist und Product Manager für die Entwickler-Toolsparte der XDEV Software Corp. in Deutschland. In dieser Funktion ist er verantwortlich für die Entwicklung von Produkt-, Marketing- und PR-Strategien und deren Umsetzung, für die Verbreitung von XDEV-Lösungen im deutschsprachigem Raum sowie für Kooperationen mit Technologie-Partner. Derzeit ist Markus Stiegler mit dem Aufbau eines Experten-Teams für Migrationsprojekte beschäftigt, das ab Anfang 2011 in der Lage sein soll, besonders zeitkritische Migrationen auf Java und AJAX sicher und kostengünstig durchzuführen. Von 2007 bis 2008 war Markus Stiegler für die Produktion der Fernsehsendung XDEV TV zuständig, die XDEV Software Corp. zusammen mit dem Gaming-Sender GIGA Digital bis zu dessen Auflösung produzierte. Neben der Co-Moderation leitete er das Redaktionsteam.

Intelligente Migration?

Matthias Pfau, Zühlke Engineering GmbH und Berthold Maier, Oracle Deutschland

Oracles Smart Upgrade soll technologisch die Migration vom Oracle 10g Release zum 11g Release vereinfachen. Der folgende Artikel beschreibt, was des „intelligente" Upgrade Tool wirklich leistet und mit welchen zusätzlichen Migrationstätigkeiten kalkuliert werden muss.

Die Vision vom Upgrade bestehender OC4J und SOA Applikationen lässt die Herzen der Anwendungsentwickler höher schlagen. Schließlich müssen diese über kurz oder lang Bestandsanwendungen auf den WebLogic Server migrieren und benötigen dazu mehr als nur einen technischen Leitfaden.

Funktionsunfang des Tools

Um eins gleich Vorweg zu nehmen: Das Ziel von Smart Ugrade besteht nicht darin, die eigentlichen Migration durchzuführen. Vielmehr soll der Anwender darin unterstützt werden, während der Migration die richtigen Entscheidungen zu treffen. Die Smart Upgrade-Funktionalitäten setzen sich wie folgt zusammen:

- Analyse von gepackten Anwendungen und Bibliotheken wie EAR, WAR und JAR-Dateien
- Analyse der Server-Konfiguration von OC4J (Version 10.1.2 oder 10.1.3)
- Die Generierung von Webservice-Artefakten
- Die Migration von BPEL-Artefakten zu SCA

Die ersten beiden Funktionen erzeugen einen Report, wie er in den Abbildungen 1 und 3 dargestellt wird. Dieser Report listet alle während der Analyse mögliche Problemfelder auf und bietet die dazu passenden Lösungsmuster an.

Während der Generierung von Webservice-Artefakten werden WebLogic spezifischen Konfigurationsdateien und Java-Klassen erzeugt.

Im Rahmen der BPEL zu SCA Migration wird die durch BPEL definierte Geschäftslogik eines Services auf die Implementierung einer SCA-Komponente abgebildet.
Das Smart Upgrade [1] ist grundsätzlich in zwei Varianten verfügbar:

- Als Standalone Java-Anwendung
- Als JDeveloper Plugin

Bei der Verwendung des JDeveloper Plugins muss beachtet werden, dass Smart Upgrade zur Zeit ausschließlich mit der JDeveloper-Version 11.1.1.1 oder 11.1.1.2 genutzt werden kann. Außerdem integriert das JDeveloper Plugin nicht die vollständige Funktionalität des Upgrade Tools, so können z.B. keine WAR- und JAR-Archive analysiert werden. Daher beziehen sich die folgenden Ausführungen auf die Standalone-Anwendung.

Innerhalb der nächsten Abschnitte wird anhand von Migrationsszenarien die Unterstützung durch den Upgrade Wizard skizziert.

EJB3-Migrationsbeispiel

Eine Beispiel-Anwendung nutzt Entity-Beans zur Realisierung der Persistenz und Session- bzw. Message-Driven Beans zur Implementierung der Anwendungslogik.

Der Smart Upgrade Report bestätigt, dass OC4J spezifische APIs genutzt werden und die Anwendung frei von OC4J abhängigen Deployment-Deskriptoren ist. Der Report wird wie folgt erzeugt:

```
java -jar smartupgrade.jar
-html --ears ${EAR_FILE}
```

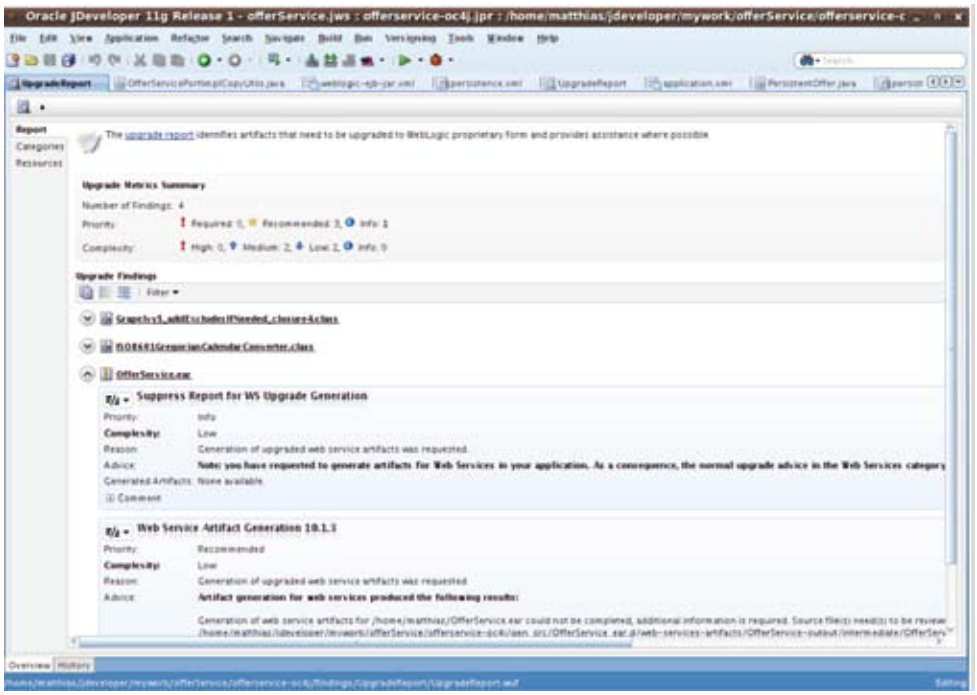


Abbildung 1: Der vom JDeveloper bereitgestellte Report nach Analyse eines Webservices

Webservice (JAX-RPC)

Die Migration eines JAX-RPC Webservices ist komplizierter, da dieser in jedem Fall OC4J abhängige Konfigurationsdateien enthält. Mit folgendem Kommando kann ein WAR-Archiv untersucht werden:

```
java -jar smartupgrade.jar
-html --wars ${WAR_FILE}
```

Der erzeugte Report weist z.B. darauf hin, dass die Inhalte bestimmter OC4J-Konfigurationsdatein in die Weblogic-Welt übertragen werden müssen. Aufgrund der unterschiedlichen JAX-RPC Implementierungen im OC4J bzw. WebLogic Server sind Code-Änderungen nötig. Doch auch auf dieses Problem wird man durch den ausführlichen Report hingewiesen.

Wie bereits angesprochen ist das Upgrade-Tool in der Lage Webservice-Artefakte zu generieren, um die Migration eines Webservices zu vereinfachen. Die gut gemeinte Funktionalität versteckt den ursprünglichen JAX-RPC Webservice hinter einer JAX-WS Implementierung.

Eine Migration durch Generierung des Codes ist erfahrungsgemäß problematisch. Diese zeigt sich z.B. in unserem Beispiel bei der unvollständigen und auch fehlerhaften Generierung von Java-Klassen. Zudem hat generierter Code den entscheidenden Nachteil, dass er äußerst schlecht wartbar ist, sodass von der Nutzung der Funktionalität abgeraten wird. Besser man behilft sich heute mit den JEE-Annotations, die eine sehr einfache und standardkonforme WS-Generierung erlauben.

Server-Konfiguration

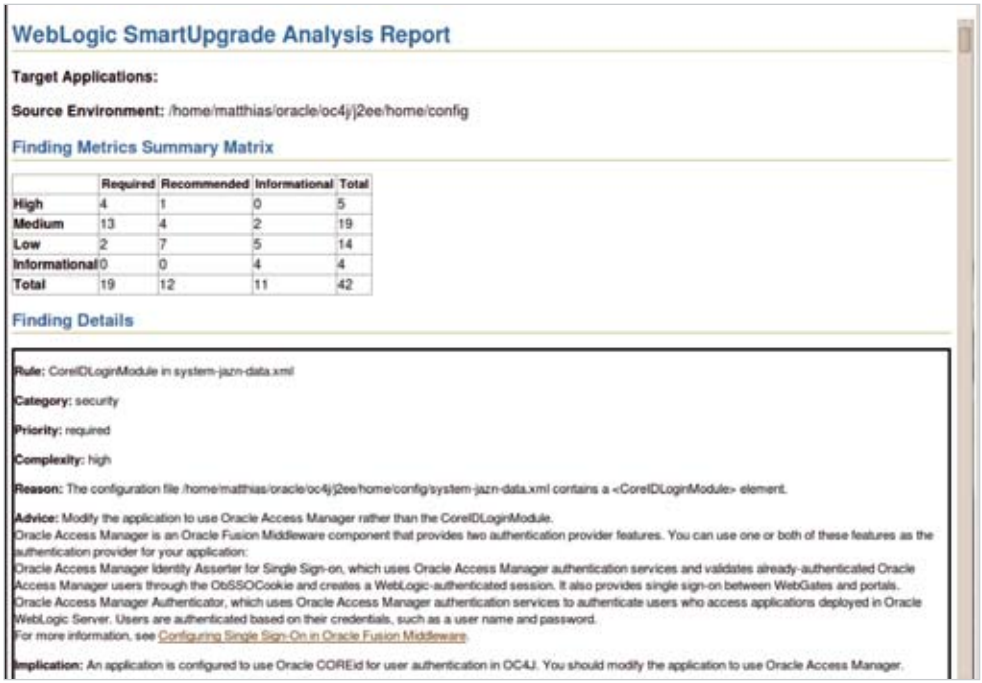
Die Analyse von Server-Konfigurationsdateien führt zu einer umfangreichen Auflistung von Hinweisen, um die OC4J-Konfiguration auf den WebLogic Server zu übertragen. Der in Abbildung 2 dargestellte Report kann wie folgt erzeugt werden:

```
java -jar smartupgrade.
jar -html --server-config
${OC4J_HOME}/j2ee/home/config >
oc4jReport.html
```

Was wird nicht unterstützt...

Das Upgrade Tool stellt keine Unterstützung für die Migration von EJB-Clients zur Verfügung, da diese ohnehin JEE-Standardkonform sind. Gewöhnlich werden diese durch Ändern der JEE Factory-Classen und Properties, sowie Anpassungen an den Projektabhängigkeiten ohne größere Aufwände migriert. Weitere Informationen zu diesem Thema sind in [4] zu finden.

Da neben der eigentlichen Anwendung hoffentlich ein automatisierter Build migriert werden muss, sollte hier mit zusätzlichen Aufwänden gerechnet werden. Häufig werden in Buildskripten, z.B. zum Deployment, Ant-Tasks des Application Servers verwendet. Diese müssen auf die vom WebLogic zur Verfügung gestellten Tasks migriert werden.



WebLogic SmartUpgrade Analysis Report

Target Applications:

Source Environment: /home/matthias/oracle/oc4j/j2ee/home/config

Finding Metrics Summary Matrix

	Required	Recommended	Informational	Total
High	4	1	0	5
Medium	13	4	2	19
Low	2	7	5	14
Informational	0	0	4	4
Total	19	12	11	42

Finding Details

Rule: CorelDLoginModule in system-jazn-data.xml

Category: security

Priority: required

Complexity: high

Reason: The configuration file /home/matthias/oracle/oc4j/j2ee/home/config/system-jazn-data.xml contains a <CorelDLoginModule> element.

Advice: Modify the application to use Oracle Access Manager rather than the CorelDLoginModule.
Oracle Access Manager is an Oracle Fusion Middleware component that provides two authentication provider features. You can use one or both of these features as the authentication provider for your application:
Oracle Access Manager Identity Asserter for Single Sign-on, which uses Oracle Access Manager authentication services and validates already-authenticated Oracle Access Manager users through the ObSSOCookie and creates a WebLogic-authenticated session. It also provides single sign-on between WebGates and portals.
Oracle Access Manager Authenticator, which uses Oracle Access Manager authentication services to authenticate users who access applications deployed in Oracle WebLogic Server. Users are authenticated based on their credentials, such as a user name and password.
For more information, see Configuring Single Sign-On in Oracle Fusion Middleware.

Implication: An application is configured to use Oracle COREid for user authentication in OC4J. You should modify the application to use Oracle Access Manager.

Abbildung 2: Der HTML-Report für eine OC4J Serverkonfiguration

Fazit

Der intelligente Upgrade Wizard ist nicht in der Lage den Entwickler von sämtlichen, der aufwändigen und fehlerträchtigen, Migrationstätigkeiten zu befreien. Allerdings bietet er eine sehr gute Unterstützung während der Planungsphase; durch die Analyse von Java-Artefakten wie WAR-Dateien ist ein schneller Überblick über die zu erwartenden Migrationstätigkeiten möglich (Vgl. [2]). So können die Migrationsaufwände besser abgeschätzt und später systematisch abgearbeitet werden. Für die tatsächlichen Migrationstätigkeiten ist des Smart Upgrade Tool hingegen nur eingeschränkt geeignet. Die wichtigste Voraussetzung für eine erfolgreiche Migration sind umfangreiche Testfälle und ein gutes Verständnis der Unterschiede zwischen den beiden Application Servern OC4J und Weblogic (Vgl. [3]).

Referenzen

[1] http://www.oracle.com/technology/software/products/middleware/smartupgrade.html

[2] Molter, Kücherer, Pfau: Migrationsstrategie für Kernbestandteile der Fusion Middleware. In: DOAG News 03/2009

[3] Upgrading Custom Java EE Applications from Oracle Application Server to WebLogic Server. Oracle Whitepaper, July 2009

[4] http://blogs.oracle.com/WebLogicServer/2009/09/application_client_upgrades_fr.html

Kontakt:

Matthias Pfau

matthias.pfau@zuehlke.com

Matthias Pfau ist als Software Ingenieur bei der Zühlke Engineering GmbH tätig. Seine Schwerpunktthemen sind agile Vorgehensweisen, testgetriebene Entwicklung und die Oracle Fusion Middleware.

GlassFish Open Source Server 3.0 – ein Vorgeschmack auf die Application-Server der nächsten Generation

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

GlassFish ist ein beliebter und weit verbreiteter Open-Source-Application-Server, der seit 2006 als Referenz-Implementierung der Java-EE-Plattform von der GlassFish-Community und damals Sun – heute Oracle – ausgeliefert wird. Seine Popularität ist vor allem darin begründet, dass er die Bedürfnisse von Entwicklern, Administratoren und Betreibern gleichermaßen adressiert. Er ist leichtgewichtig und trotzdem voll produktionsfähig mit ausgereifter Unterstützung für Hochverfügbarkeit und hervorragender Performance. Seine letzte Version ist GlassFish v3.0.1, die Referenz-Implementierung von Java EE 6.

Die Java-EE-6-Spezifikation setzt den Prozess der Vereinfachung und Flexibilisierung der Java-EE-Plattform weiter fort und führt mit dem Konzept von Profiles (im Allgemeinen) und dem Web-Profile (im Speziellen) zum ersten Mal die Möglichkeit ein, standardisierte Subsets als Plattform für die eigenen Anwendungen zu benutzen, auf unnötige Funktionalität zu verzichten und somit eine bessere Performance und Ausnutzung vorhandener Ressourcen zu erreichen. Die hohen Anforderungen an Flexibilität und Erweiterbarkeit, die Java EE 6 mit sich bringt, haben zu einer neuen Architektur in GlassFish v3 geführt, die auf ein neues Modulsystem und OSGi basiert und dadurch neue, ungeahnte Möglichkeiten für GlassFish-Anwender anbietet, die weit über Java EE 6 hinausgehen. Es ist sehr wahrscheinlich, dass auch andere Java-EE-Application-Server-Hersteller für ihre künftige Java-EE-6-Implementierung einen ähnlichen Weg wie GlassFish v3 einschlagen. Der Artikel zeigt einige der interessantesten Merkmale und Technologien von GlassFish v3, die ihn klar als ein Applications-Umfeld-Server der nächsten Generation positionieren.

Architektur, Modularität, Erweiterbarkeit

Die Hauptbestandteile in der Architektur von GlassFish v3 sind der HK2-Kernel, die Dienste, die er bereitstellt, und die Container, die von diesen Diensten Gebrauch machen (siehe Abbildung 1). HK2 steht dabei für „Hundred Kilobyte Kernel" und das Modul-System, das GlassFish v3 intern verwendet. Es hat eine Schnittstelle, über die andere Modul-Systeme angebunden werden können, etwa über OSGi, das als Standard-Implementierung mitgeführt wird. HK2 hat aber auch eine eigene, OSGi-unabhängige Implementierung, die vom GlassFish v3 im Embedded Mode verwendet wird. Typischerweise ist ein HK2-Modul ein OSGi-Bundle, das zusätzliche Meta-Informationen enthält.

Die Gesamtfunktionalität von GlassFish v3 ist in der Form von Diensten und entsprechend bewährten Methoden einer serviceorientierten Architektur bereitgestellt. Diese können unterschiedliche Schnittstellen-Arten anbieten: den META-INF/services-Mechanismus von Java SE, eine OSGi- oder eine HK2-Schnittstelle. Um System-Ressourcen zu schonen, kommt Lazy-Loading zum Einsatz, sodass Services nur dann geladen werden, wenn sie erforderlich sind. So wird zum Beispiel der EJB-Container erst beim Deployment der ersten EJB-Anwendung hochgefahren.

GlassFish v3 ist durch die Nutzung eigener APIs oder über OSGi erweiterbar. Man kann beispielsweise einen eigenen Custom-Container über die Extension-API von Grizzly hinzufügen, der hochperformanten, NIO-basierten Netzwerk-Protokoll-Engine von GlassFish. Dieser Mechanismus wurde für die Implementierung der nativen Jruby- und Jython-Container verwendet.

GlassFish v3 wird zusammen mit dem Felix-OSGi-Container ausgeliefert, könnte jedoch auch mit anderen OSGi-Containern wie Equinox oder Knopflerfish unmodifiziert laufen. Eine Besonderheit der

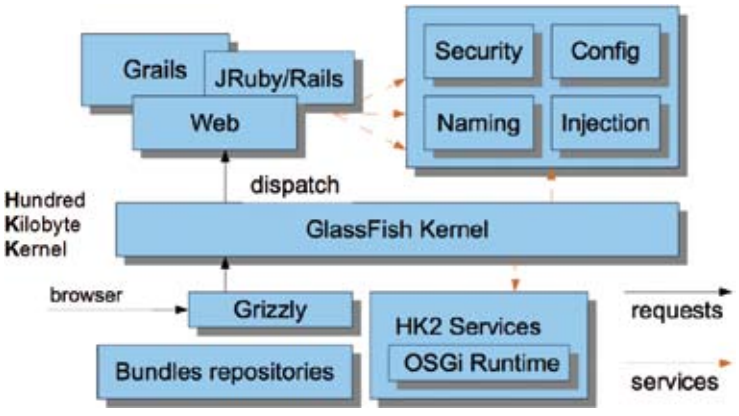


Abbildung 1: Die Architektur von GlassFish v3

Integration mit OSGi besteht darin, dass die Visibilität von OSGi von den GlassFish-Entwicklern auf die GlassFish-Benutzer ausgedehnt wird. Java-EE-Anwendungsentwickler können über Annotations und Resource-Injection transparent auf über OSGi definierte Services zugreifen, ohne jegliche Verwendung von GlassFish- oder OSGi-spezifischen APIs:

```
@Resource(mappedName="checkOsgi
Service")
CheckService checkService;
```

Dies ermöglicht die Erstellung von sogenannten „converged" Java-EE-Anwendungen, bei denen OSGi zur Auflösung von Abhängigkeiten herangezogen wird. Das GlassFish-Team erwägt auch die andere Richtung: Java-EE-Komponenten und -Dienste künftig als OSGi-Services zugänglich zu machen.

Einfache Entwicklung

Entwickler, die zum ersten Mal mit GlassFish v3 arbeiten, erleben eine angenehme Überraschung. Der Application-Server ist extrem leichtgewichtig und ressourcenschonend. Das Web-Profile ist etwa 50 MB, das Full-Profile etwa 80 MB groß. Die Start-up-Zeit beträgt nur wenige Sekunden – auf gängigen Laptops unter zehn. Das hängt damit zusammen, dass die Services erst bei Bedarf hochgefahren werden, und wenn noch nichts deployed ist, ist auch kein Container gestartet. Die Dokumentation ist ausführlich und übersichtlich. Es gibt zahlreiche Optionen für Management und Monitoring, wie nachfolgend beschrieben. Für Testzwecke besteht noch ein Maven-Plug-in für Embedded GlassFish mit Maven-Goals zum Starten/Stoppen und zur Installation/Deinstallation von Anwendungen.

GlassFish bietet diverse Deployment-Optionen. Zu den weniger bekannten gehören automatisches Deployment, Dynamic Reloading und Deployment von Java-EE-Application-Clients über Java Web Start. Eine Neuheit für Entwickler ist das sogenannte „Keep-Session-on-Redeploy"-Feature. Wenn konfiguriert, serialisiert GlassFish die User-Sessions zwischen Redeployments, sodass sie erhalten bleiben, selbst wenn die Anwendung inzwischen modifiziert wird.

Der Spaß mit GlassFish kommt aber erst dann richtig auf, wenn man dabei eine geeignete Entwicklungsumgebung verwendet. Für diejenigen, die mit den neuesten Java-EE-6-Features arbeiten wollen, ist NetBeans 6.9.1 die beste Wahl. In NetBeans kann man leicht GlassFish integrieren, verwalten und debuggen. Der hervorragende Support von Java EE 6 in Form von Code-Completion und Wizards erleichtert den Einstieg und steigert die Produktivität. Gute Unterstützung für GlassFish v3 und Java EE 6 bieten auch das Oracle Enterprise Pack für Eclipse, das Tools Bundle für Eclipse und IntelliJ IDEA.

Embeddable GlassFish

GlassFish v3 bietet die Möglichkeit, eingebettet in einer beliebigen Java-Anwendung zu laufen. Dafür gibt es APIs, über die man den Application Server verwalten kann, wie starten, stoppen, konfigurieren sowie Anwendungen installieren und deinstallieren. Er kann einfach als Bibliothek und Bestandteil einer Anwendung ausgeliefert werden, was ihn für OEM-Hersteller sehr interessant macht. Für solche Szenarien wird GlassFish in Form eines JAR-Files, jeweils für das Web- oder Full-Profile oder alternativ dazu als Maven-Plug-in bereitgestellt. Das Maven-Plug-in vereinfacht stark den Build- und Test-Prozess für Anwendungen, weil man dafür alles auf demselben Rechner abwickeln kann und nicht mal eine GlassFish-Installation braucht. Das folgende Code-Segment skizziert einen JUnit-Test, bei dem eine Anwendung getestet wird, nachdem sie auf GlassFish in Embedded-Mode deployed wurde:

```
@BeforeClass public static void initContainer() {
   Server.Builder builder = new Server.Builder();
   Server server = builder.build();
   ContainerBuilder b =              server.
createConfig(ContainerBuilder.Type.web);
   server.addContainer(b);
   ...
   File archive = new File(„hello.war");
   server.getDeployer().deploy(archive);
}

@Test public static void pingApplication() {
   ...
}
```

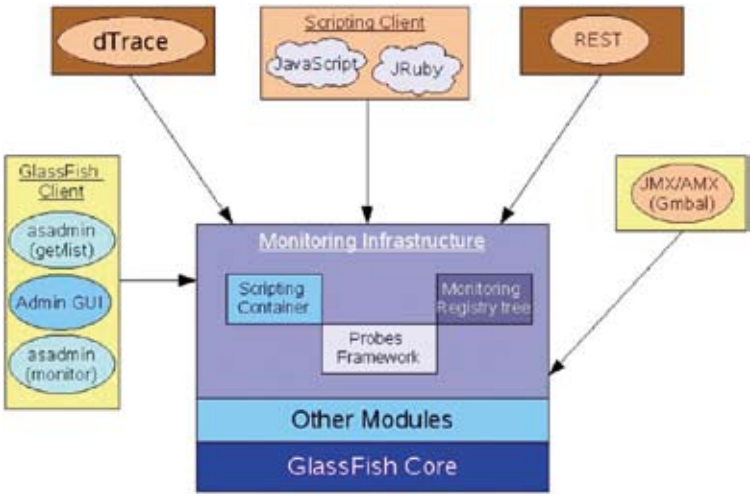


Abbildung 2: GlassFish v3 Monitoring

Management und Monitoring

Es gibt umfangreiche Möglichkeiten für das Management und Monitoring von GlassFish v3: über die Admin-Konsole, ein intuitives und gut durchdachtes Web-Interface, über das Command-Line-Interface, über Ant-Tasks, über JMX oder die AMX-API, eine GlassFish spezifische API, die den Umgang mit JMX und den Mbeans von GlassFish vereinfacht (siehe Abbildung 2).

Eine sehr interessante Möglichkeit für Management und Monitoring bietet das neue REST-Interface, das den Einsatz von GlassFish in Cloud-Umgebungen erleichtert. Jedes Konfigurationselement von GlassFish entspricht einem Knoten im Objekt-Graph einer GlassFish-Konfiguration, der als eine REST-Ressource über den URL http://host:port/management/domain/<path-in-object-tree> exponiert ist. Abhängig von dem Knoten-Typ ist die Zulässigkeit der http-Methoden GET, POST, DELETE, die zum Auslesen oder Manipulation des Konfigurationsgraphen dienen. Das REST-Interface macht es möglich, REST-basierte Management- und Monitoring-Clients für GlassFish v3 auch in anderen Sprachen zu schreiben. Das folgende Beispiel zeigt, wie man das bekannte cURL-Utility benutzen kann, um über das REST-Interface eine JDBC-Ressource in GlassFish v3 zu deaktivieren:

```
curl -X POST -d „enabled=false“
http://localhost:4848/manage-
ment/
domain/resources/jdbc-resource/
jdbc-sample
```

Das Monitoring in GlassFish v3 wurde komplett überarbeitet und erweitert. Es setzt auf ein BTrace-basiertes Framework auf, bei dem dynamisch und nicht-intrusiv von jeder GlassFish-Runtime-Klasse Events generiert werden können. Die Events werden dann von dynamisch registrierten Listenern gesammelt, statistisch ausgewertet und über diverse APIs für verschiedene Monitoring-Clients bereitgestellt. Dank BTrace sind die BTrace-Probes, die als Ereignis-Quelle dienen, dem DTrace-Framework unter Solaris zugänglich. DTrace ist ein umfangreiches Framework und Tool für dynamisches Tracing, das im Solaris-Kernel verankert ist und non-intrusive die gleichzeitige Untersuchung von Anwendungen und Betriebssystem im laufenden Produktionsbetrieb ermöglicht. Mit DTrace kann man sich durch dynamische Instrumentierung ein komplettes Bild vom gesamten Produktionssystem (Anwendung + Betriebssystem) machen und gegebenenfalls schnell Performance- und sonstige Problem-Quellen lokalisieren. Mit anderen Worten: Das neue Monitoring-Framework in GlassFish v3 ermöglicht eine feingranulare Analyse des Application-Servers durch das mächtigste Tracing-Tool unter Solaris und leistet dadurch einen großen Beitrag für höhere Verfügbarkeit und Qualität seiner Dienste in der Produktion.

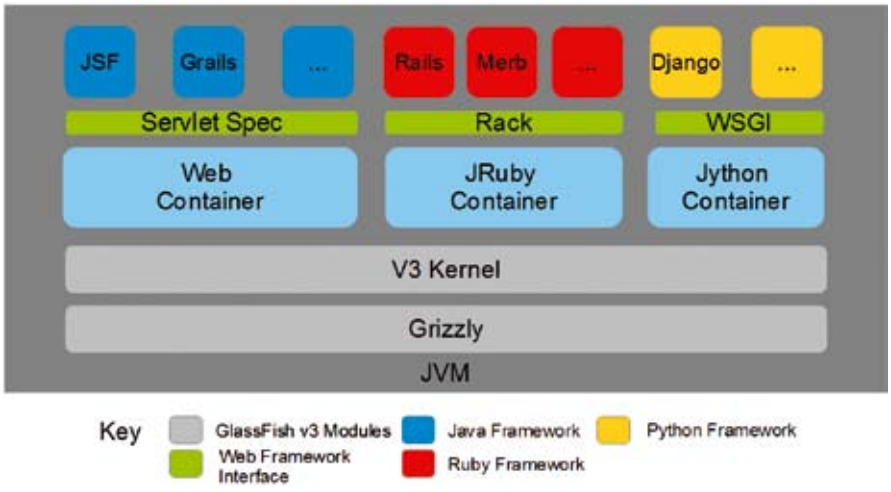


Abbildung 3: Scripting-Frameworks auf GlassFish v3

Unterstützung für Scripting

Dynamische Scripting-Sprachen wie Jruby, Groovy, Jython und Scala erfreuen sich zunehmend einer wachsenden Popularität und Verbreitung. Die Gründe dafür sind vielschichtig, aber zu den wichtigsten gehören die hohe Produktivität, der leichte Einstieg, die mächtigen und leicht verwendbaren Scaffolding-Frameworks, die sie begleiten, und nicht zuletzt die pragmatische Überlegung, dass sie sich für ein bestimmtes Projekt als schnell und gut genug erweisen.

GlassFish v3 hat deutlich die Unterstützung von dynamischen Scripting-Sprachen verbessert und bietet native Container für Jruby und Jython (siehe Abbildung 3). Weiterhin unterstützt GlassFish Groovy/Grails, Scala/Lift und PHP. Scripting-Anwendungen, die auf GlassFish deployed werden, profitieren zunächst von der Infrastruktur eines Java-EE-Application-Servers – bessere Skalierbarkeit durch die Verwendung von Thread- und Datenbank-Connection-Pools, bessere Sicherheit und Wiederverwendung seiner Management- und Monitoring-Features. Als großer Vorteil erweist sich aber auch die modulare und erweiterbare Architektur von GlassFish v3, da sie die Anbindung von anderen Containern und die Erweiterung der Admin-Console, etwa für Jruby-Spezifika, erleichtert. Über das GlassFish-Update-Center kann zu jedem Zeitpunkt eine bestehende GlassFish-Installation mit den notwendigen Modulen für die gewünschte Scripting-Unterstützung nachgerüstet werden.

Fazit

GlassFish v3 ist ein sehr leichtgewichtiger, erweiterbarer, embeddable Application-Server, dessen Unterstützung für Web-2.0 -Anwendungen sich über Java hinaus auch auf andere moderne Scripting-Sprachen ausdehnt. GlassFish v3 bietet seinen Benutzern den vollständigen Support für Java EE 6, die benutzerfreundlichste, flexibelste und vollständigste aller bisherigen Java-EE-Plattformen. Mit GlassFish v3 können Unternehmen zwischen dem leichtgewichtigen Web-Profile und dem mächtigeren Full-Profile als standardisierte Laufzeit-Umgebungen für ihre Anwendungen wählen.

Kontakt:
Peter Doschkinow
peter.doschkinow@oracle.com

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

Testen mit JUnit für Fortgeschrittene

Jens Schauder, LINEAS Informationstechnik GmbH

JUnit ist das bekannteste Test-Framework im Java-Umfeld. Ein Grund, warum es so beliebt ist, liegt darin, dass es recht klein und damit einfach zu erlernen ist. Dennoch bietet selbst diese kleine Bibliothek diverse Features, die nur wenig bekannt sind. Einige davon aus der aktuellen Version 4.8.2. werden nachfolgend vorgestellt.

Wer hatte nicht schon einmal eine Testklasse, deren Testmethoden in etwa so aussahen (siehe Listing 1).

Eine Gruppe von Tests ist praktisch gleich und unterscheidet sich nur in wenigen Parametern. Um zu vermeiden, immer wieder den gleichen Test schreiben zu müssen, gibt es in JUnit parametrisierte Tests. Um diese zu nutzen, benötigt die Testklasse einen öffentlichen Konstruktor, der die benötigten Parameter entgegennimmt.

Typischeweise speichert man die Parameter in Feldern der Klasse, sodass sie dann in den Tests verfügbar sind.

Geliefert werden die Parameter von einer öffentlichen statischen Methode, die eine „Collection" von Arrays bereitstellt. Die Methode wird mit der Annotation „@Parameters" versehen. Die gesamte Testklasse muss mit dem „Parameterized Runner" ausgeführt werden. Das Ganze sieht dann wie in Listing 2 aus.

Dieser Test wird bei der Ausführung durch JUnit nun dreimal instanziiert und jedes Mal mit einem anderen Satz Parameter. Er verhält sich also im Prinzip genau wie die ursprüngliche Variante. Das Maß der Code-Duplizierung ist jedoch erheblich reduziert. Dies macht sich besonders bezahlt, wenn die Tests nicht ganz so trivial sind wie in diesem Beispiel oder es noch mehr verschiedene Werte für die Parameter gibt.

Obwohl dies sehr praktisch ist, sollen gewisse Nachteile und Beschränkungen dieses Ansatzes nicht unerwähnt bleiben. Vielleicht ist dem einen oder anderen schon aufgefallen, dass im Listing 2 Objekte vom Typ „Character" vorkommen, während in den ursprünglichen Tests in Listing 1 der primitive Typ „char" verwendet wurde. Dies ist geschehen, da JUnit Arrays von Objekttypen benötigt.

```
public class SomethingTest{
    @Test
    public void something_returns_B_for_A(){
          assertEquals(‚B‘, something(‚A‘));
      }

      @Test
    public void something_returns_C_for_D(){
          assertEquals(‚C‘, something(‚B‘));
      }

      @Test
    public void something_returns_D_for_C(){
          assertEquals(‚D‘, something(‚C‘));
      }

      //...
}
```

Listing 1

```
@RunWith(Parameterized.class)
public class SomethingParameterizedTest {

      @Parameters
      public static Collection<Character[]> parameters() {
            return Arrays.asList(new Character[][] {
{ ‘a’, ‘b’ },
{ ‘b’, ‘c’ },
                  { ‘c’, ‘d’ }
});
      }

      private final char expectedResult;
      private final char parameterToTest;

      public SomethingParameterizedTest(char aParameterToTest,
                  char anExpectedResult) {
            expectedResult = anExpectedResult;
            parameterToTest = aParameterToTest;
      }

      @Test
      public void something_returns_the_correct_valuse() {
assertEquals(expectedResult,
something(parameterToTest));
      }
}
```

Listing 2

```
public class SomethingRuleTest {

	@Rule
	public MethodRule rule = new SomeRule();

	@Test
	public void testSomething() {
		assertEquals('y', something('x'));
	}
}
```

Listing 3

```
public class SomeRule implements MethodRule {

	@Override
	public Statement apply(Statement statement, FrameworkMethod method,
			Object test) {
		return statement;
	}

}
```

Listing 4

```
	@Override
	public Statement apply(final Statement statement, FrameworkMethod
method,
			Object test) {
		return new Statement() {

			@Override
			public void evaluate() throws Throwable {
				statement.evaluate();
			}
		};
	}
```

Listing 5

```
public interface Slow {}
public interface NeedsDB {}
public interface NeedsGUI {}

Einzelne Tests können mit der @Category Annotation mit einer oder mehreren
Kategorien versehen werden.

public class CategoryTest {
	@Test
	@Category(Slow.class)
	public void slowTest() {
		// ...
	}

	@Test
	@Category({ Slow.class, NeedsDB.class })
	public void slowDbTest() {
		// ...
	}
}
```

Listing 6

Die unterschiedlichen Testdurchläufe werden nur durch Indizes unterschieden, das heißt, wenn von zwanzig derartigen Tests einer fehlschlägt, ist nicht ohne Weiteres zu erkennen, welche Parameterkombination dafür verantwortlich ist. Es empfiehlt sich daher, die Meldungstexte von Assertions so zu gestalten, dass die verwendeten Parameter erkennbar sind.

Die vielleicht schwerwiegendste Kritik ist die, dass der Mechanismus sehr unflexibel ist und sich selbst nicht gut für eine Wiederverwendung eignet. Am Ende dieses Artikels wird ein anderes Feature vorgestellt, das diesen Nachteil nicht hat.

Rules

Rules sind eine Möglichkeit, die Art und Weise, wie eine Testmethode ausgeführt wird, zu beeinflussen. Alles, was man dazu im Test tun muss, ist ein öffentliches statisches Feld vom Typ „MethodRule“ oder von einem davon abgeleiteten Typ anzulegen und dies mit der Annotation „@Rule“ zu versehen (siehe Listing 3).

Dieser Teil ist extrem einfach. Spannend wird das Ganze in der Implementierung der Rule, d.h. in dem Beispiel in der Klasse „SomeRule“ (siehe Listing 4).

Diese muss das Interface „MethodRule“ implementieren. Dies wiederum besteht aus einer einzigen Methode „apply“, von der wir zunächst nur den ersten Parameter und den Rückgabewert betrachten. Beides ist vom Typ „Statement“. Ein Statement ist ein Test, der bereit ist, ausgeführt zu werden. Eine Rule wird für jeden Test mit eben diesem Test als Statement-Parameter aufgerufen und muss ein Statement zurückliefern, welches dann tatsächlich ausgeführt wird. Die oben vorgestellt Implementierung tut, wie man leicht erkennt, im Endeffekt nichts. Eine echte Rule wird typischerweise eine neue Statement-Instanz erzeugen, oft in Form eines anonymen Typs (siehe Listing 5).

„Statement“ selbst ist wiederum ein sehr einfaches Interface mit nur einer Methode „evaluate“, die zur Ausführung des Tests aufgerufen wird. Leider tut unsere Rule immer noch nichts Sinnvolles, aber wir haben den Aufruf des Tests nun in unserer Implementierung von „evaluate“ unter Kontrolle und können dort diverse Dinge tun, zum Beispiel einen eigenen

Thread ausführen, etwa im Eventhandling-Thread, wenn wir Swing-Komponenten verwenden. Wir können die Zeit messen, die unser Test benötigt, und mit einer JUnit-Assertion überprüfen, ob diese unterhalb eines bestimmten Schwellwertes liegt (siehe org.junit.rule.TimeOut). Wir können Ressourcen erzeugen, die vom Test benötigt werden, und diese anschließend wieder entfernen. Wir können die Ausführung des Tests protokollieren, indem wir vor und nach dem Aufruf von „statement.evaluate" Log-Ausgaben erzeugen.

Um alle Möglichkeiten von Rules nutzen zu können, schauen wir uns noch die verbleibenden Parameter der „apply"-Methode an. Die „FrameworkMethod" ermöglicht „reflection"-artigen Zugriff auf die eigentliche Testmethode, insbesondere auf den Namen und auf eventuell vorhandene Annotations. Dies kann sehr gut verwendet werden, um das Verhalten einer Rule für einzelne Tests durch Annotationen oder durch Namenskonventionen noch weiter zu beeinflussen. Der dritte Parameter ist die Test-Instanz selbst. Hierüber könnten Instanzvariablen manipuliert oder ausgelesen werden.

Eine weitere beliebte Methode ist es, in den Tests auf die Rule-Instanz zuzugreifen, was natürlich problemlos möglich ist, da es ja einfach ein statisches Feld ist. JUnit selbst bringt einige Rules mit, die neben ihrem direkten praktischen Wert auch exzellente Codebeispiele für Rules liefern. Die „TimeOut"-Rule betrachtet einen Test als fehlgeschlagen, wenn er länger als eine im Konstruktor der Rule angegebene Zeitspanne benötigt. Wenn man seine Assertions mit dem „ErrorCollector" statt mit dem normalen „Assert" schreibt, wird die Ausführung des Tests auch im Fehlerfall fortgeführt, und man bekommt am Ende alle fehlgeschlagenen Assertions anstatt nur der ersten. Diese Rule ist abgeleitet von der Rule „Verifier", die es davon abgeleiteten Rules ermöglicht, im Anschluss eines Tests noch weitere Bedingungen zu prüfen. Mit „ExpectedException" kann man erwartete Exceptions genauer prüfen als mit dem üblichen „@Test(expected=SomeException.class)", welches nur den Typ der Exception prüft. „ExternalResource" bildet den bekannten „before/after"-Mechanismus als Rule ab und dient als Basisklasse für weitere Rules, die dann nur jeweils „before()" und „after()" implementieren müssen. „TemporaryFolder" ist davon abgeleitet und erstellt einen temporären Ordner, in dem vom Test Dateien erzeugt werden können, die automatisch am Ende wieder weggeräumt werden.

„TestWatchman" ist eine weitere Basisklasse. Sie ist Basis für Rules, die die eigentliche Testausführung nicht weiter verändern wollen, aber vor oder nach einem Test Aktionen ausführen wollen. „TestName" ist eine davon abgeleitete Klasse, die sich einfach den Namen eines Tests merkt und diesen als Methode zur Verfügung stellt, sodass ein Test darauf zugreifen kann.

Categories

Die Empfehlungen für gute Tests sind sich darin einig, dass diese schnell sein und unabhängig von der Umgebung immer das gleiche Ergebnis liefern sollen. Leider sieht die Realität oft deutlich anders aus. Da gibt es langsame Tests, von denen ein paar hundert schon eine halbe Stunde benötigen. Andere benötigen eine Datenbank oder ein anderes Schnittstellensystem, und es gibt Tests, die eine GUI benötigen, womöglich ein bestimmtes Betriebssystem. Schnell ist man so weit, dass die gesamte Test-Suite nur noch selten ausführbar ist, weil die eine oder andere Bedingung nicht erfüllt ist. Mit Categories können Tests kategorisiert und dann Tests bestimmter Kategorien ausgeführt oder eben nicht ausgeführt werden. Als Marker für Kategorien dienen beliebige Klassen, typischerweise jedoch Marker-Interfaces, das heißt Interfaces, die keine Methoden enthalten (siehe Listing 6).

Es kann auch eine ganze Testklasse wird mit einer solchen Annotation versehen werden, was den gleichen Effekt hat, als würde man die Annotation an jede Testmethode der Klasse schreiben. Einzelne Tests können zusätzlich noch weitere Kategorien zugewiesen bekommen (siehe Listing 7).

Was nun noch fehlt, ist ein Runner, der alle Tests bestimmter Kategorien ausführt. Der Runner „Categories. Categories" ist vom „Suite"-Runner abgeleitet, kann also auf die gleiche Weise eingesetzt werden. Insbesondere werden die auszuführenden Testklassen mit der „@SuiteClasses"-Annotation angegeben. Es werden die auszuführenden Kategorien über die Annotation „@IncludeCategory" angegeben und über „@ExcludeCategory" auszuschließende Kategorien aufgelistet. Als Argument akzeptieren beide Annotationen ein Array von Klassen. Fehlt „@IncludeCategory", werden alle Kategorien inkludiert; fehlt „@ExcludeCategory", werden keine Kategorien exkludiert. So lässt sich zum Beispiel eine Testsuite definieren, die alle Tests ausführt, die eine GUI benötigen, solange sie nicht langsam sind (siehe Listing 8).

```
@Category(NeedsGUI.class)
public class CategoryGuiTest {
	@Test
	public void guiTestOne() {
		// ...
	}

	@Test
	public void guiTestTwo() {
		// ...
	}

	@Test
	@Category(Slow.class)
	public void slowGuiTest() {
		// ...
	}
}
```

Listing 7

```
@RunWith(Categories.class)
@IncludeCategory(NeedsGUI.class)
@ExcludeCategory(Slow.class)
@SuiteClasses({ CategoryGuiTest.class, CategoryTest.class })
public class GuiNotSlowTestSuite {}
```

Listing 8

Um die Auflistung aller Testklassen nicht immer wiederholen zu müssen, bietet es sich an, diese einmal in einer Test-Suite zu definieren, von der die anderen Tests dann abgeleitet werden können (siehe Listing 9).

```
@SuiteClasses({ CategoryGuiTest.class, CategoryTest.class })
public class AllTestSuite {}

@RunWith(Categories.class)
@IncludeCategory(Slow.class)
public class SlowTestSuite extends AllTestSuite {}
```

Listing 9

Theories

„Theories" ähneln in ihrer Verwendung den parametrisierten Tests vom Anfang des Artikels. Auch Theories benötigen ihren eigenen Runner „Theories". Jede Testmethode muss mit der Annotation „@Theory" gekennzeichnet sein, anstatt mit „@Test". Diese Testmethoden können Parameter erwarten. Zur Verfügung gestellt werden diese Parameter dann durch sogenannte „Datapoints", öffentliche statische Felder mit einer „@DataPoint"-Annotation. Jede Testmethode oder besser jede Theorie wird nun mit allen aus diesen DataPoints erzeugbaren Kombinationen von Datenpunkten aufgerufen, solange die Datentypen kompatibel sind. Der Test (siehe Listing 10) …

```
@RunWith(Theories.class)
public class TheorieTest {

      @DataPoint
      public static String a = "a";

      @DataPoint
      public static String b = "bb";

      @DataPoint
      public static String c = "ccc";

      @Theory
      public void stringTest(String x,
      String y) {
      System.out.println(x + " " + y);
    }
}
```

Listing 10

… liefert die Ausgabe:

```
a a
a bb
a ccc
bb a
bb bb
bb ccc
ccc a
ccc bb
ccc ccc
```

Interessieren bestimmte Parameterkombinationen nicht, da für diese die Theorie nicht gilt, kann man dies durch Methoden der Klasse „Assume" ausschließen. Diese ähneln stark den Methoden der Klasse „Assert", ein Nichterfüllen der Bedingung führt aber nicht zu einem Fehler, sondern zum Ignorieren der entsprechenden Parameterkombination (siehe Listing11).

Die Theorie „assumeTest" führt daher in der Klasse „TheorieTest" zu folgender Ausgabe und wird weiterhin als grüner Test betrachtet:

```
bb a
bb bb
bb ccc
ccc a
ccc bb
ccc ccc
```

Diese Art Tests zu schreiben ist sinnvoll, wenn man sich mit „Design by Contract" beschäftigt, wenn man sich also bestimmte Zusicherungen bewusst macht, die eine Klasse gibt. Eine Implementierung des „java.util.List"-Interface sichert zum Beispiel zu, dass man ein Objekt, das man erfolgreich mit „add" hinzufügt (der Rückgabewert ist „true"), sich anschließend in der Liste befindet („contains" mit dem gleichen Objekt als Argument liefert ebenfalls „true"). Dies lässt sich als Theory formulieren, die man dann mit beliebig vielen Datenpunkten überprüfen kann.

Die Definition der Datenpunkte – wie bisher geschehen – ist sehr wortreich. Dies stört, vor allem, da Theorien gerade dann sinnvoll sind, wenn es viele Datenpunkte gibt. Daher können alternativ auch Datenpunkte als Arrays angegeben werden. Wichtig ist dabei, dass man die Annotation „@Datapoints" (Plural) verwendet, ansonsten würde das gesamte Array als einzelner Array-wertiger Datenpunkt angegeben werden.

```
@DataPoints
public static Integer[] j = {
1, 2, 3 };
```

Das Vorgehen stößt mit den bisher vorgestellten Mitteln an seine Grenzen, wenn man Parameter hat, die zwar vom gleichen Typ sind, die sich aber in ihrer Interpretation unterscheiden. Wenn etwa zwei Argumente vom Typ „String" vorliegen, von denen der eine aber Personennamen enthält und der andere Kreditkartenanbieter. Für diese Fälle sind „ParameterSupplier" möglich (siehe Listing 12).

Die Klasse muss die Klasse „ParameterSupplier" erweitern und die einzige abstrakte Methode „getValueSource" implementieren. Diese liefert eine Liste von „PotentialAssignments". Sie bestehen aus dem eigentlichen Wert und einem Namen, der in Fehlerfällen verwendet werden kann. Als Parameter für „getValueSource" wird die Signatur des Parameters übergeben, für den der Parameter-Supplier eingesetzt wird. Dazu später mehr.

Um den Supplier an einen Parameter zu binden, muss zunächst noch eine Annotation definiert werden, die selbst mit der Annotation „@ParametersSuppliedBy" den Parameter-Supplier referenziert:

```
@Retention(RetentionPolicy.
RUNTIME)
@ParametersSuppliedBy(NameSuppl
ier.class)
public @interface AllNames {}
```

Wichtig ist dabei die Retention „RUNTIME", damit die Annotation auch zur Laufzeit wirklich zur Verfügung steht. In einem Test können dann Parameter von Theorien mit der neuen Annotation versehen werden. Diese Parameter werden dann vom Parameter Supplier mit Werten versorgt. In dem Beispiel ist die Annotation „@AllCre-

```
@Theory
public void assumeTest(String x, String y) {
	Assume.assumeTrue(x.length() > 1);
	System.out.println(x + " " + y);
}
```

Listing 11

```
public class NameSupplier extends ParameterSupplier {

	@Override
	public List<PotentialAssignment> getValueSources(ParameterSignature signature) {
		ArrayList<PotentialAssignment> result = new ArrayList<PotentialAssignment>();
		result.add(PotentialAssignment.forValue("Single Name", "Martin"));
		result.add(PotentialAssignment.forValue("Two Names", "Hans Martin"));
		result.add(PotentialAssignment.forValue("Weird Characters", "André"));
		// ...
		return result;
	}
}
```

Listing 12

```
@RunWith(Theories.class)
public class SuppliedByTest {

	@Theory
	public void supplierTest(@AllCreditCards String cc, @AllNames String name) {
		System.out.println(cc + " " + name);
	}
}
```

Listing 13

ditCards" analog zu „@AllNames" implementiert (siehe Listing 13).

Nun wird auch deutlich, dass man mit der „ParameterSignature", die dem Parameter-Supplier übergeben wird, diesem noch zusätzliche Informationen zukommen lassen kann. So kann er zum Beispiel Werte für unterschiedliche Datentypen erzeugen oder aber die Annotation, über die der Parameter-Supplier an einen Parameter gebunden wird, kann noch weitere Informationen aufnehmen, die vom Parameter-Supplier mit ausgewertet werden.

Fazit

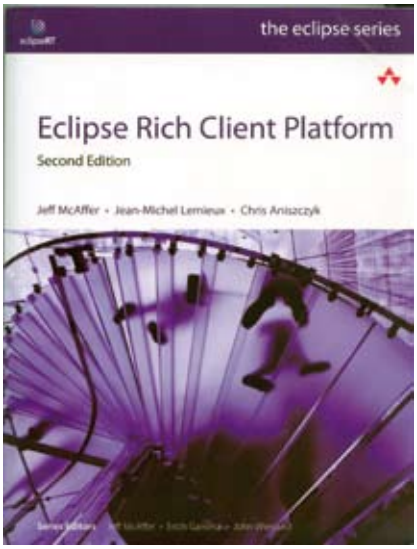
Es ist doch erstaunlich, was sich alles in einer so kleinen und weit verbreiteten Bibliothek wie JUnit noch finden lässt. Auch wenn die Handhabung von Annotationen für manchen noch immer fremd ist, empfehle ich dringend, diese Features im Kopf zu behalten, wenn es an das Schreiben von Tests geht, denn sie sind geeignet, Tests mit geringem Aufwand wesentlich gründlicher, einfacher und letztendlich besser wartbar zu gestalten.

Kontakt:
Jens Schauder
Jens.schauder@lineas.de

Eclipse Rich Client Platform

Jürgen Thierack

Als der Autor vor zwei Jahren erstmals an einem Eclipse RCP-Projekt mitmachte, wurde ihm auf seine Frage, was es zum Thema zu lesen gebe, „der McAffer" empfohlen. Dieser sei zwar schon etwas veraltet, aber für praktische Belange das beste Buch am Markt. Es sei für Eclipse das, was „Rich Client Programming: Plugging into the NetBeans Platform von Tim Boudreau, Jaroslav Tulach, Geertjan Wielenga, Prentice Hall International, 2007" für die Netbeans sei.



Eclipse Rich Client Platform, 2. Auflage
Jeff McAffer, Jean-Michel Lemieux,
Chris Aniszczyk
Addison-Wesley Professional, 2010
Paperback; 552 Seiten, $ 49,99
ISBN-10: 0321603788
Sprache: Englisch

Kurz vor dem Helios-Release kam 2010 die zweite Auflage des „Definitive Guide to Eclipse Rich Client Development" (Verlagstext) auf dem Stand von Eclipse 3.52 heraus. Die erste Auflage von 2005 war auf dem Level von Eclipse 3.1. Die Verfasser sind Projektleiter oder Committer im Eclipse-Projekt. Den Eclipse RCP-Anfänger erwartet eine verschachtelte API und eine ungewohnte Begrifflichkeit. Diverse „Berater" (Advisors) steuern das Erscheinungsbild von Fenstern, Menüs, Toolbars und Perspektiven (Beispiel: Java- und Debug-Perspektiven in der Java-Entwicklung mit Eclipse): WorkbenchAdvisor, WorkbenchWindowAdvisor, ActionBarAdvisor. Die Darstellung von Informationen kann in Form sogenannten „Views" erfolgen (Package Explorer, Type Hierarchy in der Java-Entwicklung). Eine View („MyView") wird erzeugt, indem ViewPart abgeleitet wird. Es ergibt sich eine nicht gerade selbstevidente Klassenhierarchie.

- public class MyView extends ViewPart
- public abstract class ViewPart extends WorkbenchPart implements IViewPart
- public abstract class WorkbenchPart extends EventManager implements IWorkbenchPart3, IExecutableExtension, IWorkbenchPartOrientation
- public abstract class EventManager (endlich von Object abgeleitet)

Mathias Fuchs, Leiter eines Projektes beim IT-Dienstleister Logica für den Kraftwerkbauer Alstrom, antwortete auf die Frage, welche Schlüsse sich aus heutiger Sicht aus dem Projekt ziehen lassen: „Die Entscheide für Java und Eclipse RCP waren richtig, aber die Lernkurve war steiler, als wir uns das vorgestellt hatten. Selbst Spitzenprogrammierer haben nach dem Einstieg bei solchen Projekten noch einiges vor sich. Das muss man bei der Release- und Kostenschätzung unbedingt berücksichtigen" (Quelle: http://www.de.logica.ch).

Anstatt sich theoretisch in die Thematik einzuarbeiten, sollte man den ersten Satz von Buchabschnitt II („RCP by Example") beherzigen: „The best way to learn about Eclipse as a rich client platform is to build a rich client application". Und genau das ist der Schwerpunkt des Buches. Es wird eine Anwendung für Instant Messaging (sofortige Nachrichtenübermittlung) erstellt, welche auf den Namen Hyperbola (Hyperbel) hört. Wenn Karl eine Nachricht in seinen Client tippt, erscheint diese im Client von Erna, sofern sich Erna ebenfalls am Server angemeldet hat. Die Applikation wird schrittweise erstellt, jedes Detail der Programmierung ausführlich besprochen, GUI, Events, Datenbindung, Erstellung als Stand-Alone Anwendung, Individualisierung mit Icons und Splash Screen, p2-Installer, Verwendung von Java Web Start (JNLP). Man lernt, indem man sich allmählich an die Eclipse-RCP-Welt gewöhnt.

Mit http://eclipsercp.org gibt es eine Webseite zum Buch, auf der es neben den Errata auch den übrigens seit Erscheinen des Buches stark überarbeiteten Quellcode von Hyperbola in seinen verschiedenen Entwicklungsstufen gibt. Ferner findet sich dort ein Forum, das zur Zeit 49 Topics hat. Eines der Topics lautet *Example-Code plugin does not install in Helios 3.6.1:*

> *Hi folks,*
> *just downloaded the latest helios version of eclipse. Now, when installing the example plug-in provided by http://www.ecipsercp.org, I get: ... (Liste von Fehlermeldungen)... With galileo (3.5.2), I do not have this problem, everything works fine.*
> *Any suggestions?*
> *Cheers, Marcel*

Fazit

Es besteht also die Hoffnung, dass dank Webseite und Forum auch die neueste Eclipse-Version Berücksichtigung findet und so das Schicksal aller Computer-Bücher, die schnelle Alterung, wenigstens verzögert wird. Zwar erfahren wir im Buch etwas zur Geschichte von Eclipse RCP, doch einen Ausblick auf Eclipse 4 sucht man vergebens, aber das ist kein Nachteil, denn es geht um das praktische Know-how, hier und jetzt erfolgreich Applikationen zu erstellen.

Kontakt:
Jürgen Thierack
thierack@gfm-muenchen.de

Jürgen Thierack. ist Diplomchemiker, wechselte aber in die Software-Branche, wo er freiberuflich unterwegs ist. Seit fast 10 Jahren liegt sein inhaltlicher Schwerpunkt bei Fragen der Finanzmathematik, darunter der Preisfindung für Optionen und Optionsscheine. Aktuelle Thematik: Trendfolgesysteme. Mit Java arbeitet er seit 2001.