Spring Data und Querydsl: Weniger Code, mehr Spaß ▸ 20

S&S

JAVA Mag

Deutschland €9,80 Österreich €10,80 Schweiz sFr 19,50 Luxemburg €11,15 9.2013

Javamagazin™

Java • Architekturen • Web • Agile

www.javamagazin.de

W-JAX 13 Alle Infos hier im Heft! ▸ 51

HaikuVM
Java Virtual Machine für Mikrocontroller ▸ 92

Reactor
Neues asynchrones Framework ▸ 73

ANDROID

Android ist anders – Rückblick und Ausblick ▸ 48

Android-Entwicklung macht Spaß – mit IntelliJ IDEA ▸ 65

Tutorial: Die 5-Minuten-Android-App ▸ 55

Cloud Save: Neues Feature für Google Play Services ▸ 70

MEHR THEMEN: Spring XD: Springs Lösung für Extreme Data ▸ 41 | Red5-Flash-Medienserver: Streaming leicht gemacht ▸ 76 | javahidapi: Per Java auf USB zugreifen ▸ 89

4 194586 709801 09

Android First?

Endlich ist es wieder so weit: nach drei Jahren Abstinenz hat es Android wieder auf das Cover des Java Magazins geschafft. Die damals gestellte Frage „Wie hebt sich Android von Java ME ab?“ kann mittlerweile leicht beantwortet werden: Android hatte Erfolg.

So viel Erfolg, dass es inzwischen zum Vorbild für andere mobile Betriebssysteme werden konnte. Ja, Sie haben schon richtig gehört: Im Internet unkt man bereits, das neue Apple iOS 7 habe Features eingeführt, die Android teils schon seit langer Zeit bereitstellt (http://bit.ly/15RvMks). Nehmen wir beispielsweise das neue Command Center in iOS 7, das mit einer Swipe-Geste aufgerufen wird, genau wie das Panel, das seit Android 4.2 aus der Notification Bar heraus aufrufbar ist. Oder nehmen wir das Multi-Tasking-Feature, die Möglichkeit, durch eine Preview aller geöffneten Apps zu swipen. Apple hatte bislang die Multi-Tasking-Bar immerhin in Form von zuletzt geöffneten Apps mit Icons dargestellt, also gilt dieser Vergleich nur teilweise. Aber haben Sie schon den neuen Lock-Screen von iOS 7 gesehen? Zumindest dieser beweist, dass die Zeit des hässlichen Entleins Android vorbei ist. Seit dem Meilenstein 4.0 ist Android so sexy geworden, dass es Maßstäbe setzen kann. Das hätte man sich vor drei Jahren noch nicht träumen lassen, oder?

Android Studio

Letzten Monat haben wir Ihnen bereits einen Einblick in das neue Android Studio gegeben, Googles neues Tool für Android-Entwickler. Es ist noch lang kein erwachsenes Projekt, doch zeigt die enorme Resonanz in der Community, dass Android Studio einen Nerv getroffen hat: immer mehr Entwickler suchen immer besseres Tooling für ihre Entwicklung. Und immer mehr Entwickler scheinen mit dem Status quo der Android-Entwicklung mit Eclipse unzufrieden. Google kennt die Probleme und hat sich mit JetBrains zusammengetan, um Android Studio zu entwickeln. Alle Probleme wurden aber anscheinend nicht gelöst, so sollen die alten Emulatoren im Release enthalten sein, was zu denselben Performanceproblemen führt wie in Eclipse (http://www.techwell.com/2013/06/good-and-bad-android-studio). Bis zum ersten echten Release kann das aber schon wieder ganz anders aussehen. In unserem heutigen Heftschwerpunkt zeigen wir, wo Android Studio seine Wurzeln hat: in der IntelliJ IDEA.

So oder so kann Android Studio ein wichtiger Meilenstein in Richtung Android First sein – der Punkt, an dem die Android-Entwicklung für Unternehmen und Entwickler zur Priorität wird und nicht mehr zum „zweiten Weg neben iOS“.

Java Magazin Lambda Tour

Im September gehen wir auf Road Tour! Java 8 steht vor der Tür und mit diesem Release auch die wichtigen Lambda-Ausdrücke. Gemeinsam mit unseren Topreferenten der JAX, Angelika Langer, Paul Sandoz, Klaus Kreft und Dalibor Topic reisen wir durch die Republik und machen Sie fit für Lambda und Java 8. Die Hacking-Events sind kostenlos, mehr Infos finden Sie hier im Heft auf Seite 14/15 und natürlich im Internet unter www.jaxenter.de/lambdatour. Um Updates zur Tour auf schnellstem Weg zu erfahren, empfehle ich Ihnen, dem Twitter- oder Google+-Account des Java Magazins zu folgen – hier werden wir als Erstes informieren, wenn eine neue Stadt, eine neue Location etc. feststeht!

Den Höhepunkt der Lambda Tour bildet der Java Core Day auf der W-JAX in München. Angelika Langer führt als Moderatorin durch den Tag, und auch hier gibt es wieder einiges zu Lambdas und Java 8 zu erfahren. Die Anmeldung zur W-JAX ist über http://jax.de/tickets möglich.

In diesem Sinne wünsche ich Ihnen viel Spaß bei der Lektüre dieser Ausgabe und hoffe, wir sehen uns „on the road“!

Claudia Fröhling, Redakteurin

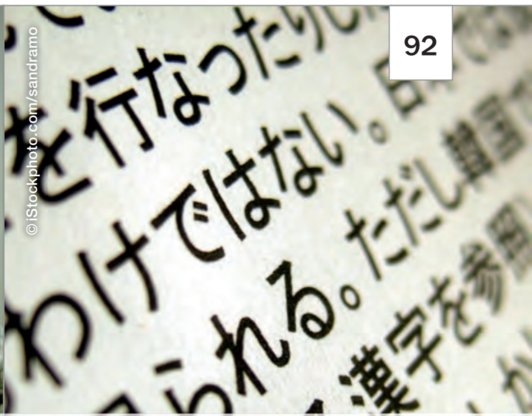
@JavaMagazin

gplus.to/JavaMagazin

Leserbriefe und Feedback zu den Artikeln des Java Magazins bitte an redaktion@javamagazin.de.

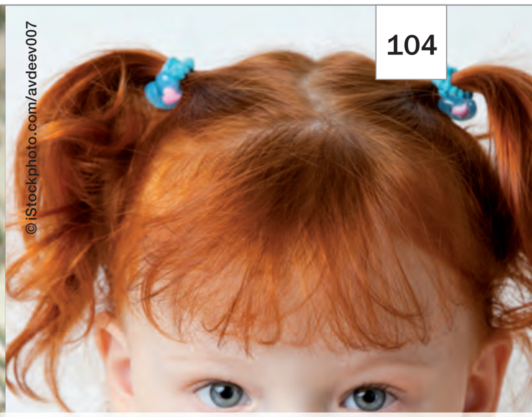
20

© iStockphoto.com/redhumv



92

© iStockphoto.com/sandramo



104

© iStockphoto.com/avdeev007

Web

Tools

Embedded

Agile

Retrospektive

Standards

Continuous Integration on a Dollar a Day

frei nach James Shore (http://www.jamesshore.com/Blog/Continuous-Integration-on-a-Dollar-a-Day.html)

Add-on

Continuous Integration kann sehr einfach sein. Man benötigt dafür keine Software wie CruiseControl oder Jenkins. Es ist so einfach, dass du sofort damit starten kannst. Nimm einen alten ausgedienten Entwicklungsrechner, schließ ihn an einen Monitor an und finde eine Ecke in eurem Teamraum. Richte dort eine Arbeitsumgebung ein und checke den letzten Stand aus der Quellcodeverwaltung aus.

Suche dir ein Build-Maskottchen. Ein mittelgroßes Plüschtier reicht vollkommen aus. Automatisiere deinen Build. Jedes Teammitglied muss in der Lage sein, alle für den Build notwendigen Schritte durch einen einzigen Tastendruck (oder einen einzigen Befehl auf der Kommandozeile) in der richtigen Reihenfolge auszuführen. Ein Tool dafür ist zum Beispiel Gradle. Folge den sieben Schritten.

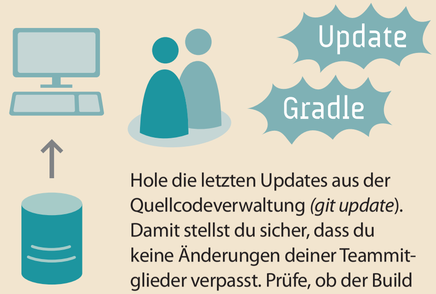
1 Lokal bauen bis grün

Prüfe deine Änderungen am Code lokal. Ist der Build erfolgreich? Laufen alle Tests noch? Committe deine Änderungen *(git commit)*.

2 Integrationstoken reservieren

Nimm dir das Build-Maskottchen. Integrieren darf nur, wer dieses Token besitzt. Damit verhinderst du, dass sich Integrationen gegenseitig behindern und du unnötig mergen musst.

3 Lokal integrieren bis grün



Hole die letzten Updates aus der Quellcodeverwaltung *(git update)*. Damit stellst du sicher, dass du keine Änderungen deiner Teammitglieder verpasst. Prüfe, ob der Build lokal durchläuft. Sind deine Änderungen mit den Änderungen der Teammitglieder kompatibel? Laufen noch immer alle Tests?

4 Einchecken

Bringe deine Änderungen in die Quellcodeverwaltung *(git push)*.

5 Auf Integrationsserver integrieren bis grün

Aktualisiere die Workarea auf dem Integrationsrechner. Starte dort den Build. Ist alles noch grün, laufen alle Tests? Hast du alles eingecheckt?

6 Bescheid geben

Gib das Integrationstoken frei und informiere deine Teammitglieder über deine Änderungen. Diese können jetzt ihre Arbeitsumgebungen aktualisieren, um auf dem neuesten Stand zu arbeiten und Merges zu verringern.

7 Ausliefern (optional)

Alle Änderungen sind integriert, der Build ist erfolgreich und alle Tests laufen. Warum nicht gleich ausliefern? Frühes Feedback ist wichtig und zeigt uns, ob wir auf dem richtigen Weg sind.

Sven Günther arbeitet bei der it-agile GmbH als Entwickler und Trainer für agile Entwicklungspraktiken. Er studierte an der Fachhochschule Brandenburg Informatik mit den Schwerpunkten Softwareentwicklung und Mikroprozessortechnik. Er interessiert sich besonders für agile Prozesse und für testgetriebene Softwareentwicklung. Zur Zeit entwickelt er mit Kollegen die nächste Top-100-iOS-App.

Frisch von JAXenter

News, Kommentare und Interviews von www.jaxenter.de

NIO-Framework Netty 4.0.0 erschienen

Das NIO-Projekt Netty erfreut sich wachsender Beliebtheit. Überraschend viele Projekte wie z. B. das Play-Framework oder AeroGear verwenden Netty zur Abwicklung der asynchronen Netzwerkkommunikation. Diese wird es freuen, dass das Netty-Team bei Red Hat, Twitter und Co. nun das vierte Major-Release gestemmt hat.

Nach mehr als einem Jahr wurden die Arbeiten an Netty 4.0.0 abgeschlossen. Zahlreiche API-Verfeinerungen wurden vorgenommen, viele davon brechen die Abwärtskompatibilität. Das Thread-Modell wurde überarbeitet. Allgemein unterstützen die meisten Netty-Operationen nun das Aneinanderhängen von Methoden. Nicht konfigurierende Getter haben das *get*-Präfix verloren – aus *Channel.getRemoteAddress()* wird damit *Channel.remoteAddress()*.

In letzter Minute wurde noch die *MessageList* entfernt und durch einen intelligenteren Mechanismus ersetzt. Die Releaseankündigung beschreibt diese letzten Fixes im Detail.

Netty 4.0.0 kann ab sofort unter der Apache-2.0-Lizenz genutzt werden. Für einen Einstieg empfiehlt sich der Blick in die Dokumentation.

▶ http://bit.ly/1apnjr9

Ein kleiner Schritt zur 2.0: Apache Log4j 2.0-beta8 erschienen

Von der Version 2.0 versprechen sich die Macher des Logging-Frameworks Log4j so einiges. Sie soll verschiedene Probleme in der Logback-Architektur beheben und das Logback allgemein verbessern. Um das zu erreichen, ist nun schon die achte Betaversion erschienen.

Diese beseitigt zunächst einmal fünfzehn Bugs. Beispielsweise werden Filter Calls von Avro oder Flume jetzt vom Flume Appender ignoriert. Async Logger Threads sind ab sofort Daemon Threads und halten die JVM nicht mehr vom Schließen ab. Durch ein Upgrade des Javadoc-Plug-ins auf Version 2.9.1 wurde ein Javadoc-Sicherheitsproblem behoben.

Aber bei der Beta 8 handelt es sich nicht um ein reines Bug-Fix-Release, denn es wurden auch einige kleinere Änderungen vorgenommen. Im Logger-API wurden *Printf*-Methoden hinzugefügt, der Default-Status-Level kann jetzt als System-Property spezifiziert werden. JUnit wurde von 4.7 auf 4.11 aktualisiert, Jansi Jar von 1.9 auf 1.11. Eine Abhängigkeit vom Apache ORO Jar wurde entfernt. Um zu funktionieren, benötigt Log4j 2.0-beta8 mindestens Java 6.

▶ http://bit.ly/1bdGWoZ

Spring Tool Suite 3.3.0 unterstützt Eclipse Kepler

Die beiden Spring-Entwicklerwerkzeuge STS (Spring Tool Suite) und GGTS (Groovy/Grails Tool Suite) bringen in der neuen Version 3.3.0 den Anschluss an Eclipse Kepler. Neben der Umstellung auf die Eclipse-4.3-Plattform wurden Aktualisierungen zu zahlreichen, mit dem Kepler-Release-Train ausgelieferten Projekten vorgenommen, darunter Mylyn, EGit, m2e und m2e-wtp.

Ein neuer Spring Project Wizard, eine schnellere Workspace-Suche und Updates zum tc Server 2.9.2, Spring Roo 1.2.4 und Grails 2.2.3 gehören zu den weiteren vollbrachten Taten. Zudem ist auf der Highlightsliste „Ready for Spring 4“ zu lesen, was bedeutet, dass die Weichen für die bevorstehende vierte Major-Version des Spring-Frameworks gestellt wurden (geplantes Releasedatum noch in diesem Jahr). In STS 3.3.0 wurden die aktuellen Spring 4.0.0.M1 Libraries integriert, mit denen sich Spring-4-Projekte parsen und analysieren lassen. Spring-4-Projekte sollten sich somit problemlos mit der Spring Tool Suite bearbeiten lassen.

In Spring 4 ist der Support für Java 8 mit den neuen Features wie Lambda-Ausdrücke und das neue Date and Time API (JSR 310) geplant. Außerdem stehen WebSocket-Architekturen (Support für JSR 356) und feingranulares Eventing und Messaging innerhalb von Spring-Anwendungen auf dem Programm. Explizit soll Spring 4 auch Java EE 7 mit seinen Komponenten wie JMS 2.0, JPA 2.1, Bean Validation 1.1 und Servlet 3.1 unterstützen.

Die nächste Version des Spring Tooling ist für Oktober 2013 geplant, kurz nach dem ersten Eclipse-Kepler-Service-Release (4.3.1). Bis dorthin wird es aber wie gehabt Milestone-Builds geben, sodass Ungeduldige schon früher die neuen Spring-4-Features nutzen können.

▶ http://bit.ly/12yaANF

Akka 2.2.0 unterstützt Cluster à la Amazon Dynamo

Mit dem Framework Akka lassen sich nebenläufige Anwendungen in Java und Scala bauen. Dazu kommen so genannte Aktoren zum Einsatz, die Messages über eine ereignisgetriebene Empfängerschleife asynchron abarbeiten. Unter dem Codenamen „Coltrane" ist nun die Version 2.2.0 von Akka erschienen.

Akka 2.2.0 bringt die volle Unterstützung für Cluster, die bisher nur in einer Previewversion enthalten war. Nach dem Vorbild der Amazon-Hashtabelle Dynamo kommen Gossip-Protokolle, eine automatische Fehlererkenung, Cluster Rebalancing und automatische Partitionierung zum Einsatz, um elastische, dezentrale Peer-to-Peer-Cluster zu verwalten. Die bisherige Cluster-Umsetzung wurde in „Coltrane" finalisiert und um Features wie Node-Rollen und einen Cluster-Client ergänzt. Zu den weiteren Neuerungen gehören eine Aktoren-basierte Remoting-Implementierung, eine skalierbare Netzwerk-IO-Schicht mit SSL-Support und eine experimentelle Umsetzung typisierter Kanäle.

Dabei wurden die für User sichtbaren API-Veränderungen minimal gehalten, die Codebasis unter der Haube hingegen an vielen Stellen überarbeitet. So hat sich die Testabdeckung der OSGi-Bundles verbessert, ebenso die Art und Weise, wie Aktoren-Failures geloggt werden. Akka 2.2.0 ist ab sofort auf der Projektseite zu beziehen.

▶ http://akka.io

JCache-Spezifikation steht zur Einsicht bereit

Deutlich vor Java EE 8 soll er fertig werden, der lang ersehnte JSR 107, besser bekannt unter dem Namen JCache – Java Temporary Caching API. Java-EE-Evangelist Reza Rahman ist angesichts des Public-Review-Dokuments der Spezifikation zuversichtlich, bald die finale Version des Standard-Cache-Systems für Java freigeben zu können.

Nur knapp hatte die JCache-Spezifikation den Einzug in Java EE 7 verpasst, für das es schon fest eingeplant war. Erst im Januar 2013 wurde klar, dass man den angesetzten Termin für Java EE 7 nicht halten konnte, sehr zur Enttäuschung zahlreicher Java-Entwickler. Trost können diese jetzt aus Rahmans Aussage schöpfen, dass JCache problemlos auch vor Java EE 8 als Drop-in *jar* zu nutzen sein wird – nun, eigentlich sagt er: „zu nutzen sein sollte". Jedenfalls sei JCache ein exzellenter Kandidat für Java EE 8.

Die JCache-Spezifikation ist im Public Review bis zum 5. August für Ihre Kommentare offen, entweder durch Mails an jsr107@googlegroups.com oder über das Erstellen von Issues auf GitHub.

▶ http://bit.ly/12ywY9v

Oracle stellt Cloud Application Foundation vor

Unter dem Namen Cloud Application Foundation (CAF) hat Oracle die Server- und DataGrid-Technologien WebLogic bzw. Coherence zusammengefasst. Das CAF-Release 12c integriert den Applikationsserver Oracle WebLogic Server 12.1.2 und die In-Memory-DataGrid-Lösung Oracle Coherence 12.1.2 zu einer Grundlagenplattform für Cloud-Systeme. Zur Verwaltung der beiden Komponenten wurde der Oracle Enterprise Manager 12c Release 3 um Cloud-Management-Features ausgestattet, mit denen sich private Cloud-Umgebungen einrichten lassen.

Die Oracle Cloud Application Foundation darf als weiterer Baustein in Oracles erklärter Strategie gesehen werden, im umkämpften Enterprise-Cloud-Segment Fuß zu fassen. In jüngster Vergangenheit hatte Oracle mit prominenten Cloud-Partnerschaften, etwa mit Microsoft und salesforce.com, Aufsehen erregt. Auf der CloudWorld in München hatte Oracle Mitte Mai davon gesprochen, zukünftig jegliche Oracle-Software sowohl „On Premise" als auch in der Cloud anbieten zu wollen.

▶ http://bit.ly/14f8bL7

CI-Server Bamboo 5.0 integriert Deployments

Bamboo, der Continuous-Integration-Server von Atlassian, ist in Version 5.0 erschienen. Wichtigste Neuerung ist der Einbezug von Projekt-Deployments. Im Sinne der DevOps-Bewegung soll die Lücke zwischen Entwicklern und Betreibern einer Software geschlossen werden. Der hohe Automatisierungsgrad, der zwischen Entwicklung und Quality Assurance mittlerweile zum Standard gehört, soll auch auf den Betrieb und dessen Deployment-Arbeit ausgedehnt werden können.

In dem Tool werden Issues, Commits, Testergebnisse und Deployments zusammengefasst und den verschiedenen Stakeholdern in ihren jeweiligen Umgebungen zugänglich gemacht. Bamboo verbindet sich mit dem Atlassian Issue Tracker JIRA und anderen Atlassian-Tools, beispielsweise einem Kanban-Board, über das sich agile Workflows abbilden lassen. Insgesamt soll der gesamte Releaseprozess damit transparent und vorhersehbar werden.

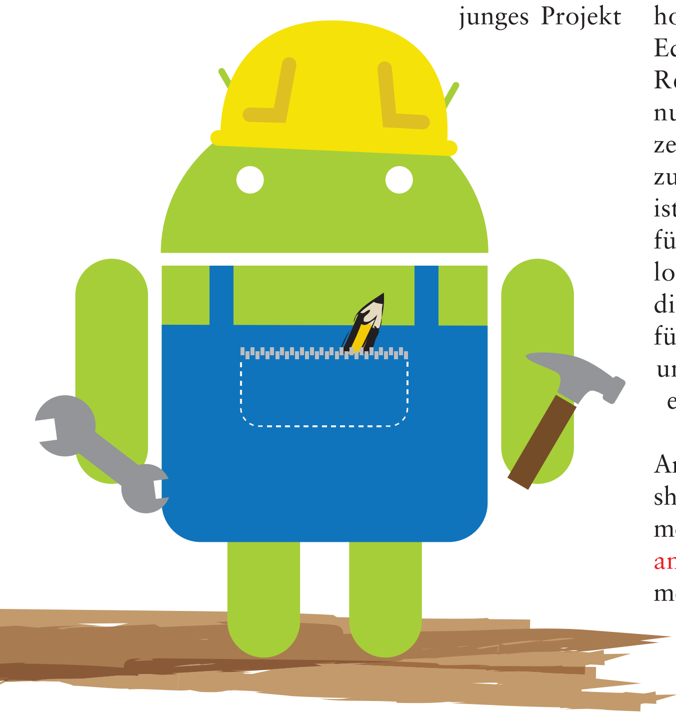
Bamboo steht als On-Demand-Service und als Download zur Verfügung. Preislich beginnt die On-Demand-Variante bei 10 monatlichen Euro für 10 Jobs und einem Agent. Die obere Grenze ist bei 100 Agents erreicht, für die man 1 000 Euro pro Monat hinlegen muss. Bei der Desktopvariante liegt die Preisspanne bei 10 bis 16 000 Euro.

▶ http://bit.ly/13IEBU8

Android Studio 0.2.0 verfeinert Ressourcen-Repository

Auf der Google I/O Ende Mai hatte der Suchmaschinenriese mit dem neuen Android Studio zu überraschen gewusst – eine komplett neue Entwicklungsumgebung für Android auf der Basis von IntelliJ IDEA. Um das Android Studio weiter als Standard-IDE zu etablieren, wurde nun die Version 0.2.0 freigegeben.

Da Android Studio noch ein recht junges Projekt ist, ging es in dem Release zunächst einmal darum, das User Feedback ernst zu nehmen und gemeldete Bugs auszumerzen. Beispielsweise funktionieren Linux-Font-Größen und das Font-Rendering nun besser. Aktualisiert wurde das Android-Gradle-Plug-in auf die 0.5.0, eine Version, die allerdings nicht abwärtskompatibel ist.



Neue Features sind auch mit dabei, beispielsweise können Screenshots in der Layoutpreview und im Editor nun gespeichert werden. Im Ressourcen-Repository werden nun zyklische Ressourcenreferenzen erkannt. Eine erste Unterstützung für .aar Library Dependencies ist auch mit an Bord (beispielsweise für die Nutzung einer Library ohne lokale Kopie der Sourcen, allerdings funktioniert dies noch nicht für Resource-XML-Validierungen und die Navigation in Quellcode-editoren).

Für die folgenden Arbeiten hat das Android-Studio-Team alle Troubleshooting-Informationen im Dokument „Known Issues“ (http://tools.android.com/knownissues) zusammengefasst. Wer auf ein Problem stößt, findet evtl. in diesem regelmäßig aktualisierten Doc erste Hilfe.

▶ http://bit.ly/12qji4b

Maven 3.1.0 nicht mehr abwärtskompatibel

Maven 3.1.0 ist da. Das populäre Build-Tool hat nach etlichen Entwicklerreleases und Alphas den Startschuss für die allgemeine Verfügbarkeit gegeben, was bedeutet, dass Maven 3.1.0 ab sofort für produktive Einsatzszenarien empfohlen ist.

Neuheiten findet man vor allem dann, wenn man unter die Haube blickt. So wurde im Maven-Kern erstmals der JSR 330 (Dependency Injection for Java) für Erweiterungen und Maven-Plug-ins verwendet. Maven hatte lange Zeit einen nicht standardisierten Dependency-Injection-Mechanismus auf der Basis von Plexus genutzt. Maven 3.1.0 ändert dies durch eine Guice-basierte Lösung, die direkt den JSR 330 unterstützt.

Weiter kommt in Maven 3.1.0 nun das Logging-Projekt SLF4J zum Einsatz. Wie schon bei der Dependency Injection hat man sich hier von Plexus verabschiedet und nutzt mit SLF4J ein weit verbreitetes System, das man als De-facto-Standard bezeichnen könnte.

Neben zahlreichen Bug Fixes ist noch der Umschwung von Sonatype Aether auf Eclipse Aether erwähnenswert. Die bei Sonatype entwickelte Library für die Arbeit mit Artefakt-Repositorys war bereits 2011 als offizielles Eclipse-Projekt vorgeschlagen worden und erhält jetzt den Vorzug vor der Sonatype-Variante, die als solche nicht mehr weiterentwickelt wird.

Maven 3.1.0 steht als Apache-Projekt kostenlos und quelloffen zum Download bereit. Wer aktualisieren möchte, sollte allerdings die Warnung von Sonatype zur Kenntnis nehmen, dass der Wechsel zu Eclipse Aether Inkompatibilitäten zu früheren Maven-Versionen nach sich zieht. Insbesondere sind Maven Tycho, das Android-Maven-Plug-in und das NetBeans-Modul-Plug-in betroffen. Die Entwickler dieser Plug-ins arbeiten derzeit an aktuellen Versionen, sodass sich an dieser Stelle noch etwas Geduld lohnen könnte.

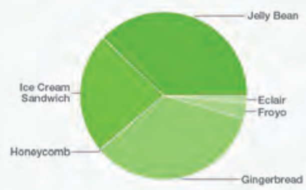
▶ http://bit.ly/10Wbxlc

Android-Versionen im Juni: Jelly Bean überholt erstmals Gingerbread

Gute Neuigkeiten für Google: Nur rund einen Monat, nachdem Apples CEO Tim Cook gegen das Android-Betriebssystem wetterte, da insgesamt gesehen nur wenige Nutzer auf die neueste Version updaten, löst das aktuelle Android Jelly Bean das inzwischen zweieinhalb Jahre alte Android 2.3 Gingerbread als beliebteste Android-Version ab.

Die in einem zweiwöchigen Zeitraum bis zum 8. Juli erhobenen Zahlen messen die Anzahl der Nutzer anhand der Zugriffe auf den Google Play Store und geben so ein akkurates Bild über die tatsächliche Nutzung der Betriebssysteme. Auf Platz eins liegt jetzt erstmals die aktuelle Version des Google OS, Jelly Bean, zu der die Versionen 4.1 (32,3 Prozent) und 4.2 (5,6 Prozent) gehören. Gegenüber dem Vormonat bedeutet dies ein Plus von fast 5 Prozentpunkten. Verloren haben hingegen Gingerbread (2.3.3 – 2.3.7; minus 2,4 Prozent zum Vormonat) und Ice Cream Sandwich (4.0.3 – 4.0.4; minus 2,3 Prozent zum Vormonat). Somit liegt Gingerbread mit einem Anteil von 34,1 Prozent auf Platz 2, gefolgt von Ice Cream Sandwich mit 23,3 Prozent.

Version	Codename	API	Distribution
1.6	Donut	4	0.1%
2.1	Eclair	7	1.4%
2.2	Froyo	8	3.1%
2.3.3 - 2.3.7	Gingerbread	10	34.1%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	23.3%
4.1.x	Jelly Bean	16	32.3%
4.2.x		17	5.6%



Data collected during a 14-day period ending on July 8, 2013.
Any versions with less than 0.1% distribution are not shown.

Quelle: developer.android.com

Auf die übrigen Versionen entfallen insgesamt knapp 5 Prozent. Selbst Android-Urgestein 1.6 Donut und das für Tablets entwickelte 3.0 Honeycomb fallen mit 0,1 noch in die Statistik. Der Grund für das stetige Wachstum von Jelly Bean dürfte neben der geänderten statistischen Erfassung, bei der nur noch Zugriffe auf den Play Store, nicht aber auf andere Google-Seiten gezählt werden, auch der Absatz neuer Smartphones sein, wie etwa dem Galaxy S3 und S4.

Insgesamt sind diese Zahlen gute Neuigkeiten für Google und Entwickler, da die Fragmentierung das Entwickeln von Apps erschwert. Genau dies hatte Tim Cook auf der World Wide Developer Conference angesprochen und aufgezeigt, dass 93 Prozent der Apple-Nutzer auf iOS 6 umgestiegen seien, während Android weitestgehend dreigeteilt bleibt. Für App-Entwickler von Android-Geräten bedeuten die Ergebnisse zudem, dass sie ihre Anwendungen für Android 4.x optimieren sollten, um so über 60 Prozent der Smartphone- und Tablet-Nutzer erreichen zu können.

▶ http://bit.ly/LkOB9H

Flyway 2.2 mit Gradle-Plug-in

Flyway ist ein Framework zur agilen Datenbankenmigration, das bislang als Java-API, Kommandozeilentool, Maven-Plug-in und Ant Tasks zur Verfügung stand. Ab Version 2.2 gibt es nun auch ein Gradle-Plug-in. Außerdem wurde der SQL-Azure-Support eingeführt.

Abgesehen davon hat die neuste Version einige weitere Änderungen zu verzeichnen. So wird die separate Spring-Distribution nicht mehr unterstützt, zum Spring-Support müssen die Spring-Jars jetzt direkt im Flyway-Klassenpfad bzw. im */jars*-Verzeichnis hinzugefügt werden.

Das Framework wirft künftig eine Ausnahme, wenn in einem Platzhalter ein Ersatzwert fehlt, und es wurde mit dem Microsoft JDBC Driver 4.0.2206.100 getestet. Dazu kommen über zwanzig gefixte Bugs.

Mit Flyway 2.2 ist das Projekt von Google Code auf GitHub umgezogen. Dort kann es kostenlos unter der Apache-2.0-Lizenz bezogen werden.

▶ http://bit.ly/Yh8hfv

Links und Downloadempfehlungen zu den Artikeln im Heft

- Android-ADT-Plug-in für Eclipse: http://developer.android.com/tools/sdk/eclipse-adt.html
- Android-ADT-Bundle: http://developer.android.com/sdk
- IntelliJ-IDEA-Android-IDE: http://www.jetbrains.com/idea/features/android.html
- Google-Cloud-Backup-API: http://developer.android.com/training/cloudsync/backupapi.html
- Android Developer Documentation: Resolving Cloud Save Conflicts: http://developer.android.com/training/cloudsave/conflict-res.html
- Google I/O 2013: Android Graphics Performance: https://developers.google.com/events/io/sessions/325418001
- Mockito: http://code.google.com/p/mockito
- Codeanalysetools Checkstyle und PMD: http://checkstyle.sourceforge.net, http://pmd.sourceforge.net
- HaikuVM: http://haiku-vm.sourceforge.net
- Eclipse Orion: http://www.eclipse.org/orion
- Code zum Artikel „Weniger Code, mehr Spaß – Spring Data, JPA 2 und Querydsl": http://github.com/codepitbull/javamagazin-spring-data-jpa-querydsl
- Red5-Medienserver: https://code.google.com/p/red5
- Reactor: https://github.com/reactor/reactor
- SpringXD: https://github.com/springsource/spring-xd
- Aktuelle Studie zu BPM Suites: Forrester Wave, Business Process Management Suites, Q1 2013: http://ibm.co/ZWp086

Autor des Monats

Wolfgang Pleus arbeitet als Technologieberater, Autor und Trainer im Bereich moderner Softwarearchitekturen. Seit zwanzig Jahren unterstützt er internationale Unternehmen bei der Realisierung komplexer Geschäftslösungen auf der Basis von Java EE und .NET. Seine Schwerpunkte liegen in den Bereichen SOA, BPM und Agile.

Wie bist du zur Softwareentwicklung gekommen?

Ich bin mit Rechnern wie C64, Atari und CBM PET aufgewachsen. Spiele wurden irgendwann langweilig, also habe ich mit der Programmierung begonnen. Da war ich zwölf. Seitdem hat mich die Softwareentwicklung nicht mehr losgelassen und so habe ich sie 1993 zu meinem Beruf gemacht.

Was ist für dich der schönste Aspekt in der Softwareentwicklung?

Zusammen mit anderen Menschen Lösungen zu entwickeln, die Nutzen bringen und den Anwendern gefallen. Insbesondere die interdisziplinäre Arbeit im Team ist reizvoll, wenn durch die Mischung von Charakteren und Kompetenzen tolle Software entsteht. Die agile Bewegung hat die Softwareentwicklung interessanter gemacht. Statt die Projektbeteiligten immer weiter voneinander zu entfremden, führt die enge Zusammenarbeit, wenn man es gut macht, zu mehr Effizienz und mehr Freude an der Arbeit.

Was ist für dich ein weniger schöner Aspekt?

Wenn sich das Potenzial eines Teams nicht optimal entfalten kann. Technologien sind heute größtenteils gut verstanden und beherrschbar. Die Herausforderung der Projektarbeit liegt häufig im politischen oder organisatorischen Bereich. Im Team hat man darauf manchmal wenig Einfluss.

Wie und wann bist du auf Java gestoßen?

Das war 1999. Als ich mit Java anfing, habe ich noch primär mit C/C++ und Assembler programmiert. In meinem ersten Java-Projekt ging es um die Entwicklung eines Applets, um Teppichentwürfe in einem dreidimensionalen Raum zu visualisieren. Das war spannend. Mit der Zeit wurde Java in den Unternehmen immer präsenter, also bin ich dabei geblieben. Mittlerweile ist Java für mich eine der wichtigsten Programmiersprachen.

Wenn du für einen Tag König der Java-Welt wärst, was würdest du verändern?

Ich finde, die Vielfalt in der Java-Welt ist Fluch und Segen zugleich. Ich habe den Eindruck, die Java-Plattform zerfasert immer mehr. Daher würde ich Strukturen etablieren, die die Übersichtlichkeit fördern. Vielleicht ein unabhängiges Qualitätssiegel für Bibliotheken und Produkte? Kürzere Innovationszyklen würde ich auch begrüßen. Es dauert einfach zu lange, bis beispielsweise Closures oder das Java-Stream-API Einzug in die Sprache finden.

Was ist zurzeit dein Lieblingsbuch?

Da ich zurzeit in einem Bankenprojekt arbeite, lese ich vor allem Bücher über Derivatehandel. Sehr spannendes Thema, mit einer ähnlich hohen Akronymdichte wie in der IT. Außerdem habe ich „JavaScript – The Definitive Guide“ von David Flanagan unter meinem Kopfkissen liegen.

Was machst du in deinem anderen Leben?

Ich verbringe die Zeit mit meiner Familie. Als Ausgleich zur kopflastigen IT-Arbeit gehe ich Laufen und Windsurfen.

JUG-Kalender* Neues aus den User Groups

WER?	WAS?	WO?
JUG **Darmstadt**	14.08.2013 – Native Android-Apps für Smartphone und Tablet entwickeln	http://jugda.wordpress.com
JUG **Ostfalen**	15.08.2013 – Spring MVC Integration Testing	http://jug-ostfalen.de
JUG **Rostock**	20.08.2013 – IDEs: Eclipse, IntelliJ, Netbeans & Co.	http://sites.google.com/site/jughro
JUG **Frankfurt**	28.08.2013 – Continuous Updating	http://jugf.de
JUG **Ostfalen**	29.08.2013 – Gute Zeilen, schlechte Zeilen – Regeln für wartbare Programme	http://jug-ostfalen.de
JUG **Schweiz**	05.09.2013 – Fahrenheit 451: Agile schadet der Dokumentation	http://jug.ch
JUG **Schweiz**	05.09.2013 – Java Cryptography for Beginners	http://jug.ch
JUG **Augsburg**	06.09.2013 – Struts Hackathon: Essential Apache Struts 2	http://jug-augsburg.de
JUG **Stuttgart**	16.09.2013 – ObjektForum Agilität und Softwarearchitektur	http://jugs.de
JUG **Saxony**	19.09.2013 – Java Web Application Security	http://jugsaxony.org
JUG **Karlsruhe**	25.09.2013 – Open Source BPM mit BPMN 2.0 und Java	http:// jug-ka.de

*Alle Angaben ohne Gewähr. Da Termine sich kurzfristig ändern können, überprüfen Sie diese bitte auf der jeweiligen JUG-Website.

Scrum in der Praxis

von Sven Röpstorff und Robert Wiechmann

Eines gleich vorab: Wer ein Buch sucht, um Scrum zu lernen, wird mit dem vorliegenden Werk nicht fündig. Denn die Autoren haben ein Buch geschrieben, um demjenigen, der erste Erfahrungen mit Scrum hat, in die Praxis zu begleiten. Frisches Wissen über Scrum sei das eine, die alltägliche Praxis dagegen das andere. Und so sprechen die Autoren eher Scrum-Einsteiger und insbesondere frischgebackene Scrum Master mit ihrem Buch an. Es geht darum, dieses Wissen nun gewinnbringend in die Praxis umzusetzen sowie die tägliche Praxis zu unterstützen. So führen die Autoren auch nicht in die Scrum-Prinzipien ein, sondern setzen sie weitgehend als bekannt voraus.

Wie setzt sich das Team zusammen? Wie geht man mit unterschiedlichen Persönlichkeiten um? Welche praktischen Aufgaben müssen die einzelnen Mitglieder, insbesondere Scrum Master und Product Owner, wahrnehmen? Die grundsätzliche Bedeutung dieser Rollen sollte dem Leser bekannt sein; das Buch zeigt, wie diese praktisch auszufüllen sind. Worum geht es beim Kick-off? Wie wird der Product Backlog gefüllt? Was ist bei Schätzungen zu beachten? Dies sowie die anderen, aus Scrum bekannten Phasen und Aktionen (Sprint, Daily Scrum, Review und mehr) werden ausführlich behandelt und mit Praxistipps versehen. Um das Ganze anschaulich zu gestalten, haben die Autoren eine Firma, ein Projekt und diverse Charaktere erfunden, die sich als Team zusammenfinden, das Projekt starten und zu Ende bringen. Diese Geschichte begleitet den Leser durch das gesamte Buch und zeigt so anschaulich, wie die Scrum-Praktiken sinnvoll umgesetzt werden können. Daran schließen sich Erläuterungen, Praxistipps, Checklisten und mehr an. Zeichnungen im Buch unterstützen den Leser nicht nur beim Verständnis, sondern dienen zum Teil auch als Vorlage für den täglichen Gebrauch.

Doch manchmal können Dinge schiefgehen. So kommt das Team in der begleitenden Geschichte auch des Öfteren an kritische Punkte, die dann aber doch irgendwie gelöst oder umgangen werden können. Zu zeigen, was auch so alles schief gehen kann, macht dieses Praxisbuch sehr anschaulich und soll zeigen, wie mit kritischen Situationen und Fallstricken umgegangen werden kann, ohne hier gleich belehrend den Zeigefinger zu heben.

Das Team entwickelt, der Scrum Master muss den Prozess zum Laufen bringen und Hindernisse vom Team fernhalten. Der Master ist der Kenner des Prozesses. Und so ist nicht verwunderlich, dass die Autoren den Leser meist als Scrum Master ansprechen. Sie wollen mit dem Buch aber nicht nur diese, sondern auch agile Coaches, Projekt- und Entwicklungsleiter, Produktmanager sowie ganz normale Softwareentwickler ansprechen. Für sie alle dürfte dieses Buch interessant sein, soweit sie nach den Scrum-Prinzipien agieren. Die Erläuterungen sind informativ, aber keine schwere Kost. Auch wenn die begleitende Story insgesamt den kleineren Teil des Buchs einnimmt, hilft sie, diese Erläuterungen auch gleich auf ein praktisches Beispiel zu projizieren. So macht es Vergnügen, das Buch zu lesen.

Michael Müller

Sven Röpstorff, Robert Wiechmann

Scrum in der Praxis

Erfahrungen, Problemfelder und Erfolgsfaktoren

348 Seiten, 36,90 Euro
dpunkt, 2012
978-3-89864-792-2

Java Magazin LAMBDA TOUR

On the Road to Java 8

Wir machen Sie fit für Java 8! Das Release ist für Anfang 2014 geplant, und damit wird es höchste Zeit, die Lambda-Ausdrücke als wichtigstes Feature von Java 8 kennen zu lernen. Der Java Core Day auf der W-JAX in München ist da genau das Richtige!

Um die Wartezeit bis zur Konferenz für Sie zu verkürzen, veranstalten wir die **Java Magazin Lambda Tour** gemeinsam mit Top-Speakern der JAX in mehreren deutschen Städten. Diese Events sind kostenlos und starten am späten Nachmittag/Abend. Nach einem einleitenden Vortrag zu Lambdas beginnt der Hackathon unter Anleitung namhafter Experten. Großes Finale der Lambda-Tour ist dann die W-JAX in München, auf der Sie Ihr Java-Wissen vertiefen können. Kommen Sie mit auf unseren Roadtrip zu Java 8!

Anmeldung

Für alle Informationen zu den Terminen, Locations und Anmeldungen besuchen Sie jetzt **www.jaxenter.de/lambdatour**.
Die Teilnehmerzahl pro Event ist begrenzt – wer zuerst kommt, hackt zuerst!


Angelika Langer


Klaus Kreft


Paul Sandoz


Dalibor Topic

Termine*

02.09.2013 Duisburg mit Angelika Langer und Klaus Kreft **For Free!**
in Zusammenarbeit mit der Abteilung INKO der Universität Duisburg-Essen

09.09.2013 Hamburg mit Paul Sandoz und Dalibor Topic in den Räumen der oose GmbH **For Free!**

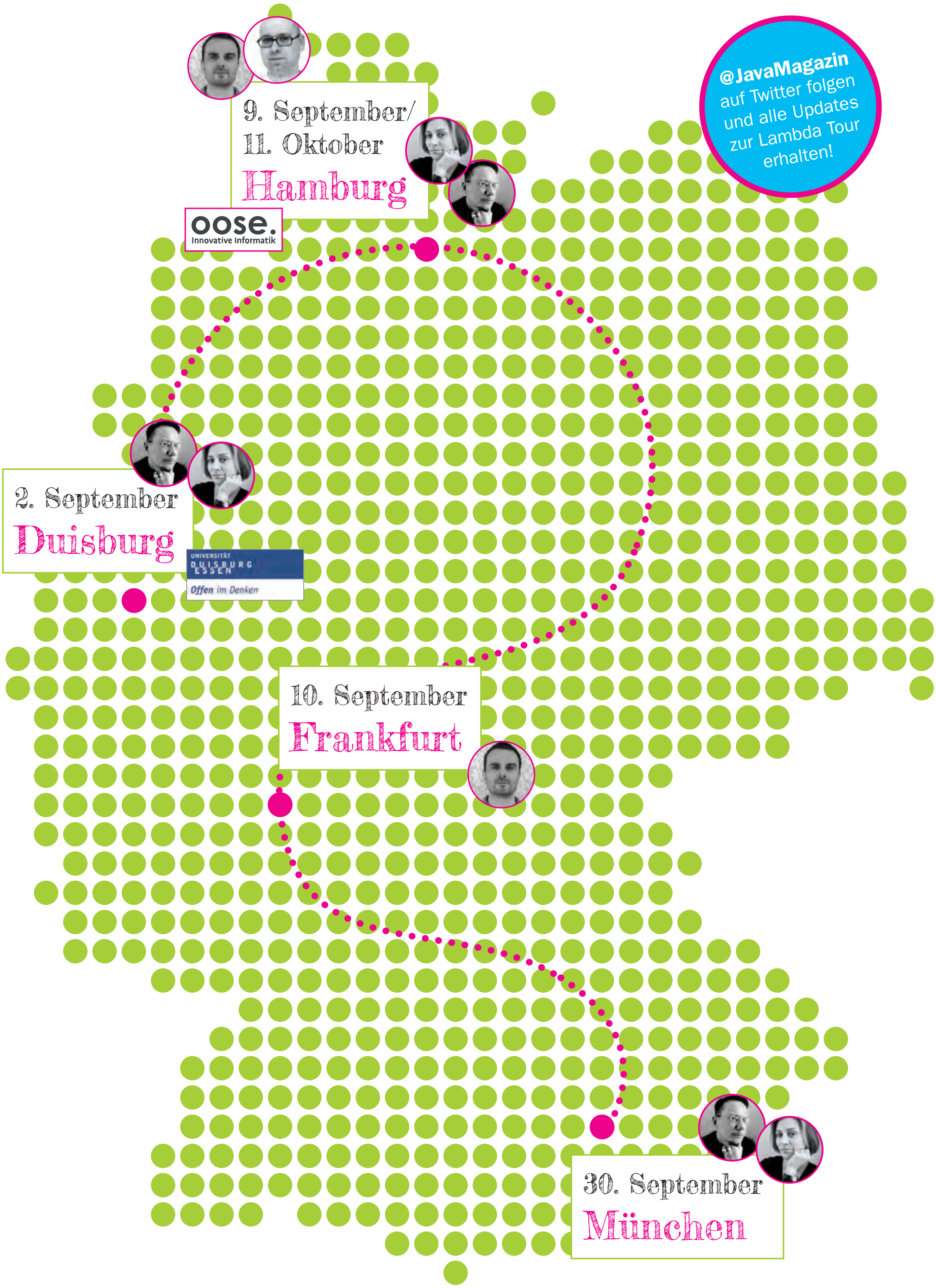
10.09.2013 Frankfurt am Main mit Paul Sandoz **For Free!**

30.09.2013 München mit Angelika Langer und Klaus Kreft **For Free!**

11.10.2013 Hamburg mit Angelika Langer und Klaus Kreft in den Räumen der oose GmbH **For Free!**

06.11.2013 München: W-JAX Java Core Day
mit Angelika Langer und vielen mehr

* Änderungen vorbehalten


9. September/
11. Oktober
Hamburg
oose.
Innovative Informatik
@JavaMagazin
auf Twitter folgen
und alle Updates
zur Lambda Tour
erhalten!
2. September
Duisburg
Universität Duisburg Essen
Offen im Denken
10. September
Frankfurt
30. September
München

Die browserbasierte Tooling-Plattform macht große Fortschritte

Orion 3.0

Zum Eclipse-Juno-Release vor einem Jahr war das Eclipse-Orion-Projekt noch nicht einmal in Version 1.0 erschienen und ein junges Mitglied der Eclipse-Familie. Seitdem hat das Projekt enorme Fortschritte gemacht. Zum Kepler-Release erscheint bereits Eclipse Orion 3.0 und zeigt, wie Softwareentwicklung im Browser zukünftig funktionieren kann. Schauen wir uns die Neuerungen der letzten Versionen einmal genauer an.

von Martin Lippert

Das Eclipse-Orion-Projekt [1] ist in vielerlei Hinsicht anders als viele andere Eclipse-Projekte. Es basiert technisch nicht auf der etablierten Eclipse-Plattform, die bis dato als Basis für die vielen Eclipse-IDEs, Rich-Client-Anwendungen und anderen Werkzeuge diente. Orion geht neue Wege und zeigt, wie die Zukunft der IDEs aussehen könnte: webbasiert, den Browser als Laufzeitumgebung nutzend und größtenteils in JavaScript implementiert.

Was vielen Eclipse-Nutzern heute vielleicht noch merkwürdig vorkommt oder von anderen als Spielzeug abgetan wird, hat sich innerhalb des letzten Jahres zu einem der interessantesten Eclipse-Projekte entwickelt. Doch schauen wir uns die Neuerungen im Detail an.

Aufgeräumtes UI

Um das Wichtigste gleich einmal vorweg zu nehmen: Die Oberfläche von Eclipse Orion ist kein auf Hochglanz poliertes Web-UI, wie man es von vielen anderen modernen Webanwendungen kennt. Das liegt zum einen sicherlich daran, dass sich das Orion-Projekt noch in einer relativ frühen Phase befindet. Zum anderen ist es auch nicht das Ziel des Orion-Projekts, eine fertige und polierte IDE zu werden. Es handelt sich in erster Linie um eine Plattform, auf deren Basis umfangreiche Entwicklungswerkzeuge implementiert werden sollen.

Dennoch hat sich am UI im Orion-Projekt einiges getan. Das grundsätzliche UI sieht deutlich aufgeräumter und einfacher strukturiert aus. Der Editor hat eine optionale Dateinavigationsansicht spendiert bekommen, die man an der linken Seite des Editors einblenden kann. Das erleichtert den Wechsel zwischen Dateien erheblich und gibt einem gleichzeitig einen guten Überblick über das Projekt, ohne den Editor verlassen zu müssen.

Im Header der Editor-Page findet man jetzt neben der aufgeräumten Quick-Link-Navigation zu den Hauptseiten von Orion eine zusätzliche Breadcrumb-Navigation – ein aus vielen anderen Werkzeugen bekanntes und liebgewonnenes Feature.

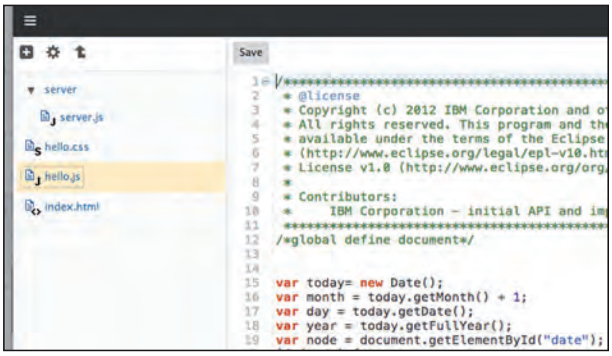


Abb. 1: Endlich auch in Orion: der ausklappbare Navigator direkt neben dem Codeeditor

Abb. 2: Die allseits beliebten Breadcrumbs, jetzt auch in Orion

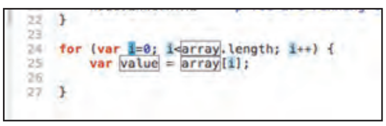


Abb. 3: Wie man das „Linked Editing" auch von Eclipse gewohnt ist: mehrere Positionen im Code gleichzeitig editieren

Der Editor

Das Herzstück von Orion ist der Codeeditor. Galt es vor einigen Jahren noch als unmöglich, wirklich gute Text- und Codeeditoren im Browser zu implementieren, existiert heute eine ganze Reihe von eindrucksvollen Gegenbeispielen. Der Codeeditor von Orion gehört mit Sicherheit dazu. Er bietet inzwischen alle wichtigen

Abb. 4: Editieren in der Cloud: Änderungen werden automatisch gespeichert

Funktionen, die ein guter Codeeditor haben sollte (inklusive Syntax-Highlighting, Code Completion, Error- und Warning-Marker, diverse Keyboard-Shortcuts, sowie ausführliche Codetemplates).

Neu ist das von Eclipse bekannte Linked Editing. Wird beispielsweise per Codetemplate ein Schleifenkonstrukt erzeugt, kann man die variablen Anteile direkt editieren (beispielsweise die Bezeichnung der Laufvariable), ohne sie an den verschiedenen Stellen im Codeanschnitt manuell verändern zu müssen.

Zusätzlich wurde das aus vielen Cloud-Umgebungen bekannte und beliebte „Auto Save" implementiert. Es kann über das neue Editor-Einstellungsmenü aktiviert werden.

Git-Integration

Die Integration für Git ist inzwischen recht umfangreich und vor allem einfach zu benutzen. Einen typischen Git-Workflow (*clone*, *edit*, *stage*, *commit*, *pull*, *push*) kann man damit problemlos durchführen. Besonders die direkte Side-by-Side-Visualisierung der Unterschiede, wie man sie auch von anderen Versionsverwaltungswerkzeugen kennt, ist in Orion gut in die Git-Unterstützung integriert.

Globalization

Neben der Accessibility spielt auch die Globalization eine wichtige Rolle, gerade für Cloud-basierte Anwendungen, die von einer Vielzahl von Anwendern über die ganze Welt verstreut benutzt werden. Unterschiedliche Character-Sets spielen dabei eine entscheidende Rolle. Orion ist nun unter anderem getestet und kompatibel mit koreanischen, chinesischen und japanischen Zeichensätzen.

Node.js

Zusätzlich zu der altbewährten und Java-/OSGi-basierten Serverimplementierung gibt es inzwischen auch eine Node.js-/JavaScript-basierte Implementierung des Orion-Servers. Mittels Node Package Manager (npm) kann man Orion einfach und schnell auf einer Maschine installieren und auf Basis von Node.js laufen lassen. Eine JVM ist in diesem Kontext nicht mehr nötig. Das mag vielleicht für die Java-Gemeinde nach einem leeren Argument klingen (schließlich hat doch jeder sowieso ein JDK oder ein JRE installiert). Schaut man sich in der JavaScript-Gemeinde um, sieht das jedoch schon ganz anders aus. Eine reine Node.js-basierte Umgebung, die leichtgewichtig ist und deren UI im Browser läuft, findet dort viel eher Gehör als eine Entwicklungsumgebung, die auf Basis von OSGi und Java läuft.

Gerade vor diesem Hintergrund ist die Node.js-basierte Orion-Variante eine interessante Alternative –

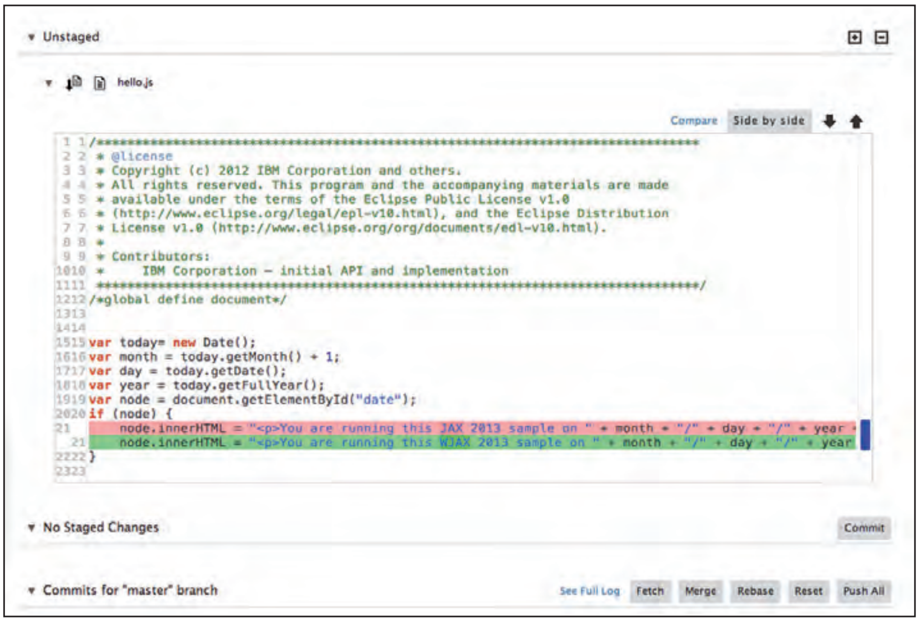


Abb. 5: Die Git-Integration zeigt die Dateiänderungen auf Wunsch direkt an

Abb. 6: Unterschiedliche Schriftsätze in Orion [3]

vor allem auch dann, wenn man selbst Node.js-basierte JavaScript-Anwendungen implementieren möchte. Der Trick dabei: Orion (auf Basis von Node.js) versteht die npm- und Node-Kommandos in der Shell. JavaScript-basierte Node.js-Anwendungen lassen sich damit komplett in Orion implementieren, testen und sogar debuggen. Mehr Details dazu finden sich unter [2].

Unter der Haube

Für Cloud-basierte Anwendungen spielt die Skalierbarkeit der serverseitigen Komponenten in der Regel eine große Rolle. Während in den ersten Tagen von Orion dieses Thema fast gar nicht betrachtet wurde (schließlich ging es darum, zunächst einmal einen Proof of Concept für die grundsätzlichen Ideen hinter Orion zu implementieren), wendet sich das Orion-Team nun verstärkt dem Thema Skalierbarkeit zu. Im Rahmen des 3.0-Releases wurden zunächst die grundsätzlichen Probleme und Ansätze analysiert und erste Teile der Implementierung verändert.

Vielleicht ist die Skalierbarkeit von Orion selbst kein besonders spannendes Thema für den Entwickler, der Orion einfach nur als IDE oder Editor verwenden möchte. Wenn man sich jedoch vergegenwärtigt, dass Orion als Tooling-Plattform gedacht und gebaut ist, auf deren Basis Unternehmen umfangreiche Entwicklungswerkzeuge implementieren und betreiben können sollen, spielt die Skalierbarkeit eine viel entscheidendere Rolle. Was wäre schon eine Cloud-basierte Entwicklungsumgebung, wenn sie nur für wenige gleichzeitige Benutzer vernünftig funktionieren würde?

Fazit

Eclipse Orion entwickelt sich zunehmend zu einer ernstzunehmenden Plattform zur Werkzeugentwicklung. Auf der EclipseCon North America waren dieses Jahr einige Anwendungsbeispiele zu sehen, die zeigen, welch ein großes Potenzial in Eclipse Orion steckt. HP zeigte beispielsweise mit dem „HP Code Anywhere"-Projekt eine experimentelle Entwicklungsumgebung basierend auf Orion.

Allerdings wird in absehbarer Zeit wahrscheinlich kein vollständiges Java-Tooling à la Eclipse JDT mit Orion implementiert werden. Man darf also nicht erwarten, dass Eclipse Orion die komplette Eclipse-Java-IDE in naher Zukunft einfach ablösen wird und wir alle nur noch im Browser arbeiten. Aber die Richtung scheint klar zu sein.

Martin Lippert ist Lead der Spring-Tool-Suite-Entwicklung und arbeitet bei Pivotal (im SpringSource-Team) mit einem internationalen Team an Entwicklungswerkzeugen rund um Spring- und Cloud-Techniken. Als Mitgründer der it-agile GmbH gilt er zudem als Spezialist für agile Softwareentwicklung.

Links & Literatur

[1] http://www.eclipse.org/orion/

[2] http://wiki.eclipse.org/Orion/Getting_Started_with_Orion_node

[3] http://planetorion.org/news/2013/05/orion-3-0-m2-new-and-noteworthy

[4] https://orionhub.org/

[5] http://planetorion.org/news/

Spring Data, JPA 2 und Querydsl

Weniger Code, mehr Spaß

Die Kombination von Querydsl und Spring Data ist derzeit konkurrenzlos. Kein anderes Framework schafft es, die Menge an Boilerplate-Code so konsequent zu reduzieren und dabei ein so elegantes API bereitzustellen. Getreu dem Prinzip: „Die beste Zeile Code ist die, die man erst gar nicht schreibt" enthalten Repositories am Ende nur das, was sie auch wirklich brauchen. Für den Entwickler bedeutet das: mehr Spaß bei der Arbeit mit relationalen Datenbanken.

von Jochen Mader

Ziel dieses Artikels ist, zu zeigen, wie die Verwendung von Spring Data in Verbindung mit JPA 2 und Querydsl das Leben extrem erleichtern kann. Vorausgesetzt wird, dass man weiß, wie man einen ORM (**O**bject **R**elational **M**apper) aufsetzt und die Konzepte hinter JPA 2 (Annotationen, EntityManager usw.) beherrscht. Den vollständigen Code zu diesem Artikel samt der zugehörigen ORM-Konfiguration findet man auf GitHub [1].

ORMs sind nicht aus der Softwareentwicklung wegzudenken. In Sachen Performance und Verwendung von speziellen DB-Features hat sich hier einiges getan. Auf der Strecke blieben dabei notwendige Weiterentwicklungen des API. Es mag ein subjektiver Eindruck sein, aber seit einer gefühlten Ewigkeit waren wir gezwungen, Mapping-Informationen in XML-Dateien abzulegen und Querys mühsam per Hand zu schreiben und zu pflegen.

Typsicherheit war hier schon immer ein Fremdwort, und die Unmengen an Boilerplate-Code waren nur schwer zu verstecken. Ständig hat man aus älteren Projekten DAO-Implementierungen kopiert oder sich zum x-ten Mal über einen Tippfehler in der Query geärgert.

Zugegeben, die Sache mit den XML-Mapping-Informationen sind wir mittlerweile losgeworden, auch wenn

Ziel von Spring Data war es, die Verwendung verschiedenster Data Stores so einfach wie möglich zu gestalten. Man legte besonderen Wert darauf, die Unterschiede der Stores nicht zu verwischen.

es immer noch genügend Leute gibt, die sich dagegen wehren. Die fehlende Typsicherheit ist das größte Problem. Egal, wie intelligent eine IDE mit der Erzeugung einer Query umgeht, er bleibt ein String, und Probleme werden im schlimmsten Fall erst erkannt, wenn es schon zu spät ist. Wer jetzt das Criteria-API anführen möchte, sei auf später vertröstet.

Spring Data

Bei Spring Data handelt es sich um ein „Umbrella"-Projekt ähnlich Spring Security oder Spring Social. Ziel war es, die Verwendung verschiedenster Data Stores so einfach wie möglich zu gestalten, ohne dabei Funktionalitäten zu beschränken. Dabei legte man besonderen Wert darauf, die Unterschiede der Stores nicht zu verwischen. Schließlich möchte man keine relationale Algebra auf Neo4j abbilden, sondern mit Graphen arbeiten.

Das so entstandene Projekt bietet Anbindungen für verschiedenste dokumentenorientierte Datenbanken, Key-Value Stores und andere alternative Konzepte. Daneben hat man auch daran gedacht, sich um den immer noch dominantesten Anteil unter den Data Stores zu kümmern: die relationale Datenbank.

Spring Data JPA

ORM ist für viele Datenbankanwendungen das Mittel der Wahl, wenn es darum geht, von Java aus mit einer DB zu interagieren. Über **den** richtigen ORM lässt sich vortrefflich streiten. Glücklicherweise will sich Spring Data in diese Diskussion nicht einmischen. Man setzte auf die Verwendung von JPA 2 als Basis. Durch diesen Standard war es möglich, fast vollkommen unabhängig von einer konkreten Implementierung zu bleiben. JPA 2 ist aber bei Weitem nicht perfekt, und man ist immer noch an einigen Ecken dazu gezwungen, Vendor-Erweiterungen zu verwenden. Für die Aufgaben von Spring Data benötigt man aber keine davon.

Bootstrap

Details zur Konfiguration von Hibernate findet man, wie eingangs erwähnt, im Beispielprojekt. Um nun Spring Data verwenden zu können, muss man die entsprechende Abhängigkeit im Build eintragen:

```
org.springframework.data:spring-data-jpa:1.3.1.RELEASE
```

Nachdem diese angezogen wurde, muss noch der Spring-Applikationskontext erweitert werden. Hier hat man sich sehr stark am *Context Component Scan*-Mechanismus orientiert. Spring Data benötigt nur die Information, in welchen Packages es nach *Repositories* zu suchen hat. Folgendermaßen wird dies mit Java Config erreicht:

```
...
@EnableJpaRepositories("com.senacor.repository")
public class MyConfiguration {
```

Für diejenigen, die kein Java Config einsetzen, zeigt Listing 1 das äquivalente Beispiel in XML.

In beiden Fällen weise ich Spring Data an, im Package *com.senacor.repository* nach Repository-Definitionen zu suchen. Jetzt wird es Zeit, das erste Repository zu definieren.

Repositories

Repositories sind der Dreh- und Angelpunkt aller Spring-Data-Projekte. Sie sind die Abstraktion zum Zugriff auf den jeweiligen Data Store. Im Fall von JPA 2 kapselt ein Repository den *EntityManager* und stellt verschiedene

Listing 1

```
...
<beans
...
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xsi:schemaLocation="
...
    http://www.springframework.org/schema/data/jpa http://www.springframework.org/
                                               schema/data/jpa/spring-jpa.xsd
...>
  <jpa:repositories base-package=" com.senacor.repository " />
</beans>
```

Listing 2

```
@Entity
public class UserEntity implements Serializable {
  @Id
  private Long id;
  private String firstname;
  private String lastname;
  @Temporal(TemporalType.DATE)
  private Date birthday;

  ...
```

Kombinationen von Methoden zum Umgang mit selbigem bereit. Am besten lässt sich das mit einem Beispiel erklären. Anhand der *UserEntity* aus Listing 2 werde ich den Repository-Mechanismus näher erklären.

Um ein *CrudRepository* (Create, Retrieve, Update, Delete) zu erzeugen, muss ich nur das entsprechende Interface (Listing 3) ableiten.

Das Listing wird dann in einem der von mir festgelegten Packages (s. o.) abgelegt:

```
public interface UserEntityRepository extends CrudRepository<UserEntity,
                                                              Long>{}
```

Listing 3

```
public interface CrudRepository<T, ID extends Serializable> extends Repository<T, ID> {
  <S extends T> S save(S entity);
  <S extends T> Iterable<S> save(Iterable<S> entities);
  T findOne(ID id);
  boolean exists(ID id);
  Iterable<T> findAll();
  Iterable<T> findAll(Iterable<ID> ids);
  long count();
  void delete(ID id);
  void delete(T entity);
  void delete(Iterable<? extends T> entities);
  void deleteAll();
}
```

Listing 4

```
@NoRepositoryBean
public interface RORepository<T, ID extends Serializable> extends Repository<T, ID> {
  T findOne(ID id);
  boolean exists(ID id);
  Iterable<T> findAll();
  Iterable<T> findAll(Iterable<ID> ids);
  long count();
}
```

Listing 5

```
public class UserEntityRepositoryImpl implements UserEntityRepositoy{
  @PersistenceContext
  EntityManager em;

  List<UserEntity> findByFirstnameLikeAndLastnameEquals(String firstname, String
                                                                  lastname) {
    Query q = em.createQuery("select user from UserEntity user where user.firstname like
                                      :firstname% and user.lastname = :lastname");
    q.setParameter("firstname", firstname);
    q.setParameter("lastname", lastname);
    return q.getResultList();
  }
}
```

Von jetzt an lässt sich mein *UserEntityRepository* durch Einsatz von *@Autowired* an beliebiger Stelle in der Anwendung verwenden:

```
@Autowired
UserEntityRepository userEntityrepository;
...
Iterable<UserEntity> allEntities = userEntityrepository.findAll();
...
```

Das war's auch schon. Von nun an sind so alle essenziellen CRUD-Operationen für die *UserEntity* verfügbar.

Wie geht das?

Eine Zeile Code (wenn man die Imports unterschlägt), um ein Repository zu erzeugen – da wird so mancher böse Magie vermuten. Entzaubern wir das Ganze: Spring Data setzt, genau wie Spring, sehr stark auf die Verwendung von Proxies. Für jedes Interface, das von *org.springframework.data.repository.Repository* ableitet, wird somit ein entsprechender Proxy erzeugt. Dessen Aufgabe ist es, Aufrufe am Interface an ihre Entsprechung am *SimpleJpaRepository* durchzuleiten. Das *SimpleJpaRepository* stellt alles an benötigten Methoden und Funktionalitäten bereit. Die Repository-Interfaces werden genutzt, um immer nur das bereitzustellen, was auch tatsächlich benötigt wird. Das Ganze wird mit einem Beispiel etwas klarer.

Read-only Repository

Wie wäre es z. B. mit einem *ReadOnly*-Repository für meine *UserEntity*? Dazu werfen wir einen Blick in die Definition des *CrudRepository* (Listing 3, eine schamlose Kopie aus den Spring-Quellen).

Mit diesem Wissen kann ich mein eigenes *ROCrudRepository* (Listing 4) erzeugen. Wichtig ist hierbei die Verwendung von *@NoRepositoryBean*. Diese weist Spring Data das Interface beim Scan zu ignorieren, obwohl es von *Repository* ableitet.

Der Vollständigkeit halber ist hier die Verwendung des neuen Repository-Typen:

```
public interface ReadOnlyUserEntityRepository extends
                                    RORepository<UserEntity, Long>{}
```

Repositories sind also extrem flexibel an den jeweiligen Anwendungsfall anpassbar. Sie sind typsicher und bieten bis auf eine „Kleinigkeit" alles, was man braucht.

Eigene Querys hinzufügen

Was noch fehlt, ist die Möglichkeit, eigene Abfragen zu formulieren. Hier bietet Spring Data mehrere Wege. Für die folgenden Beispiele soll eine Query erzeugt werden, die alle User mit einem bestimmten Nachnamen und einem teilweise angegebenen Vornamen liefert:

```
select user from User user where user.firstname like :firstname and
                                              user.lastname = :lastname
```

Die einfachste Variante, sie zu erzeugen, ist, sie gar nicht erst selbst zu schreiben, sondern die Arbeit Spring Data zu überlassen. Bereits erklärt wurde, wie die Methoden aus den Repository-Interfaces auf das *SimpleJpaRepository* abgebildet werden. Für alle Methoden, die keine Entsprechung an selbigem haben oder anderweitig implementiert wurden, erzeugt Spring Data eine Query. Der Mechanismus funktioniert dabei wie folgt:

- Eine Methode, die mit *findBy* beginnt, wird automatisch als Kandidat erkannt.
- Bei allen anderen Methoden überprüft man, ob sie mit dem Namen einer Property der Entität beginnen.
- Die Parameter der Methode werden entsprechend der Reihenfolge als Queryparameter interpretiert.

Eine solche Methode sieht wie folgt aus:

```
public interface UserEntityRepository extends CrudRepository<UserEntity,
                                                                    Long>{
  List<User> findByFirstnameLikeAndLastnameEquals(String firstname, String
                                                                lastname);
}
```

Alternativ hätte ich auch das *findBy* weglassen können, ich bevorzuge aber explizite Methodennamen. Ob man diese Variante insgesamt mag oder nicht, ist Geschmackssache. Aber es gibt noch mehr Möglichkeiten, mein Ziel zu erreichen.

Manchmal benötigt man nur einen Tick mehr Kontrolle über die erzeugte Query. Diese liefert Variante zwei. Hierbei werden die Query und ihre Parameter durch die Verwendung von *@Query*- und *@Param*-Annotationen erzeugt:

```
public interface UserEntityRepository extends CrudRepository<UserEntity,
                                                                    Long>{
  @Query("select user from UserEntity user where user.firstname like
                                   :firstname% and user.lastname = :lastname")
  List<UserEntity> findWhereFirstnameLikeAndLastnameEquals(
 @Param("firstname") String firstname, @Param("lastname") String lastname);
}
```

So können deutlich komplexere Querys formuliert werden, ohne dass man signifikant mehr Code benötigen wurde. Für die ganz harten Fälle gibt es dann noch eine dritte Variante. Bei dieser übernehme ich die Queryerzeugung komplett selbst. Allerdings muss ich hierfür die bisher vorgestellten Mechanismen umgehen. Dies gelingt durch die Auslagerung der entsprechenden Methoden in ein separates Interface:

```
public interface UserEntityRepositoryCustom {
  List<UserEntity> findByFirstnameLikeAndLastnameEquals(String firstname,
                                                          String lastname);
}
```

Daraufhin wird eine Implementierung bereitgestellt (Listing 5). Am Ende muss dieses neue Interface natürlich noch meinem Ausgangsinterface hinzugefügt werden:

```
public interface UserEntityRepository extends CrudRepository<UserEntity,
                                        Long>, UserEntityRepositoryCustom {}
```

Querydsl

Eine Sache stört allerdings an allen bisher vorgestellten Varianten: Sie sind weder Typ- noch Refactoring-sicher. Der informierte ORM-Entwickler wird jetzt natürlich den Finger erheben und anmerken, dass ich Variante drei durch die Verwendung des JPA-2-Criteria-API typsicher machen könnte. Die schreckgeweiteten Augen seiner Kollegen werden ihn dann schnell seinen Finger wieder sinken lassen. Wer schon mal mit diesem API-Verbrechen zu tun hatte, wird verstehen, weshalb ich sie für ORM meide wie Struts für Webanwendungen. So gut die Ideen auch sein mögen, die in ihr stecken, so sehr verdarben mir die Unmengen an Boilerplate und das akademische API-Design den Spaß an der Arbeit damit (dieser Satz sollte auf keinen Fall als Aufwertung von Struts gewertet werden). Glücklicherweise gibt es seit einiger Zeit eine großartige Alternative: Mit Querydsl wird es zum Kinderspiel, alle meine Anforderungen zu erfüllen.

Bootsrapping Querydsl

Zwei Abhängigkeiten sind notwendig, um Querydsl nutzen zu können:

```
com.mysema.querydsl:querydsl-jpa:2.9.0
com.mysema.querydsl:querydsl-apt:2.9.0
```

Die Erste ist das eigentliche API zum Umgang mit JPA 2 (es gibt noch eine ganze Reihe anderer Anbindungen, die allerdings nicht in den Rahmen dieses Artikels passen). Bei der Zweiten handelt es sich um den JPA Annotation Processor. Dieser wird benötigt, um die statischen Metainformationen zur Build-Zeit zu erzeugen. Wie, das sieht man in Listing 6 und 7. Sie zeigen die Verwendung mit Gradle, respektive Maven.

Bei den statischen Metainformationen handelt es sich um die so genannten Queryobjekte.

Nach dem Lauf des *JPAAnnotationProcessor* findet man für jede Entität eine weitere Klasse mit dem gleichen Namen und einem vorangestellten „Q" (in meinem Beispiel wird zur existierenden *UserEntity* die *QUserEntity* erzeugt). Sie enthält statt der Entity Properties eine Pfadbeschreibung in Form von *public static* Members. So gibt es statt *(get/set)Firstname* eine *firstname*-Methode. Diese werden wiederum in Querys eingesetzt und liefern uns alles, was wir brauchen, um mithilfe des Builder Patterns eine Query aufzubauen (Listing 8).

Diese Lösung ist typsicher, Refactoring-sicher und gut lesbar. Ganz zufrieden bin ich aber immer noch nicht. Auch für einfache Querys ist immer noch vergleichsweise viel Code nötig. Das geht noch besser.

QueryDslPredicateExecutor

Fangen wir mit dem *UserEntityRepository* noch mal von vorne an und fügen das Interface *QueryDslPredicateExecutor* hinzu:

```
public interface UserEntityRepository extends CrudRepository<UserEntity,
                                  Long>, QueryDslPredicateExecutor<UserEntity>{
}
```

Listing 6

```
dependencies {
  compile libs.jpa20
  compile libs.querydsl_apt
  compile libs.commons_lang
}

def generatedDir = "$projectDir/src/main/generated"

sourceSets {
  generated.java.srcDirs = [generatedDir]
}

configurations {
  querydslapt
}

task generateQueryDSL(type: JavaCompile, group: 'build', description: 'Generates the
                                                          QueryDSL query types') {
  source = sourceSets.main.java
  classpath = configurations.compile + configurations.querydslapt
  options.compilerArgs = [
         "-proc:only",
         "-processor", "com.mysema.query.apt.jpa.JPAAnnotationProcessor"
         ]
  destinationDir = new File(generatedDir)
  dependencyCacheDir = new File("$buildDir/dependencyCacheDir")
}
```

Listing 7

```
<plugin>
  <groupId>com.mysema.maven</groupId>
  <artifactId>maven-apt-plugin</artifactId>
  <executions>
    <execution>
      <goals>
        <goal>process</goal>
        <goal>test-process</goal>
      </goals>
      <configuration>
        <outputDirectory>target/generated-sources/java</outputDirectory>
        <processor>com.mysema.query.apt.jpa.JPAAnnotationProcessor</
                                                          processor>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Listing 8

```
public class UserEntityRepositoryImpl implements UserEntityRepositoy{
  @PersistenceContext
  EntityManager em;

  List<User> findWhereFirstnameLikeAndLastnameEquals(String firstname,
                                                    String lastname) {
    JPAQuery query = new JPAQuery(entityManager);
    QUserEntity qUserEntity = QUserEntity.userEntity;
    return query.from(qUserEntity).where(qUserEntity.firstname.like(
      firstname).and(qUserEntity.lastname.eq(lastname))).list(qUserEntity);
  }
}
```

Durch diese kleine Änderung bekommen wir einen ganzen Satz neuer Methoden:

```
public interface QueryDslPredicateExecutor<T> {
  T findOne(Predicate predicate);
  Iterable<T> findAll(Predicate predicate);
  Iterable<T> findAll(Predicate predicate, OrderSpecifier<?>... orders);
  Page<T> findAll(Predicate predicate, Pageable pageable);
  long count(Predicate predicate);
}
```

Diese enthalten in ihrer Signatur das *Predicate*-Objekt, bei dem es sich um eine Repräsentation der *where* Clause handelt. Mithilfe der Q-Objekte ist der Aufbau der Prädikate sehr einfach, und wir können die folgende Beispielquery erzeugen:

```
@Autowired
UserEntityRepository userEntityrepository;
...
QUserEntity qUserEntity = QUserEntity.userEntity;
userEntityrepository.findAll(qUserEntity.firstname.like(firstname).and(qUserEntity.lastname.eq(lastname)));
...
```

Und mit diesem Beispiel bin ich da angekommen, wo ich hin wollte. Querys lassen sich durch einfaches Betätigen der Auto-Complete-Funktion erstellen. Änderungen an den Entitäten führen zu Build-Fehlern, und ihre Auswirkungen werden frühzeitig erkannt. Der komplette Zugriff auf die Datenbank wurde gegenüber reinem JPA 2 mit dem Criteria-API deutlich entschlackt.

Ergebnis

Ich gebe es zu: Ich bin schon immer ein Spring-Fan und hatte noch nie ein Problem damit, statt des gesetzten den gelebten Standard zu verwenden. Nach der Enttäuschung, die JPA 1 darstellte, war ich dann in vielen Punkten doch angenehm von JPA 2 überrascht. Das änderte sich schnell, als ich das Criteria-API das erste Mal in einem Projekt einsetzen durfte.

Die zugrunde liegenden Ideen waren ja sehr gut – aber was nützen alle guten Ideen, wenn sie sich hinter einem derart hässlichen API verstecken? Der Wechsel zu Querydsl war die logische Konsequenz, und ich habe es bisher in keinem Projekt bereut. Im Gegenteil: Neue Entwickler kommen mit diesen Konzepten deutlich besser zurecht und finden einen schnelleren Einstieg in das Arbeiten mit ORMs.

Die Kombination von Querydsl und Spring Data ist in meinen Augen derzeit konkurrenzlos. Kein anderes Framework schafft es, die Menge an Boilerplate-Code so konsequent zu reduzieren und dabei ein so elegantes API bereitzustellen. Getreu dem Prinzip: „Die beste Zeile Code ist die, die man erst gar nicht schreibt" enthalten Repositories am Ende nur das, was sie auch wirklich brauchen.

Wie bereits erwähnt, gibt es eine Vielzahl verschiedener Integrationen im Spring-Data-Projekt, und ich kann jedem Entwickler nur empfehlen, sich mit diesen zu beschäftigen. Ich selbst verwende derzeit die Integration von Neo4j und MongoDB und bin mehr als zufrieden. Die aktive Community beschert uns hier eine wachsende Anzahl von Optionen. Kompliment an die Teams von Spring Data und Querydsl!

Jochen Mader ist Chief Developer bei der Senacor Technologies AG, wo er sich mit der Umsetzung komplexer Mehrschichtenanwendungen beschäftigt. Angetrieben durch seine Erfahrungen aus Codereviews ist er immer auf der Suche nach Techniken, um Copy and Paste und Boilerplate-Code vollständig in Projekten zu eliminieren.

@codepitbull

Links & Literatur

[1] http://github.com/codepitbull/javamagazin-spring-data-jpa-querydsl

Kolumne: EnterpriseTales

von Lars Röwekamp und Arne Limburg

Im Zeichen der 75: Java EE 7 trifft HTML5

Nachdem die beiden marketinggetriebenen Blockbuster „Cloud“ und „PaaS“ keinen Platz mehr in der aktuellen Java-EE-Spezifikation gefunden haben – sie wurden erst einmal in die Folgeversion 8 verbannt –, rückt plötzlich das Thema HTML5 in den Fokus des Java-Enterprise-Standards. Fragt sich nur, wie sich dies in den verschiedenen APIs widerspiegelt.

„The main focus of this release is on enhanced simplification, productivity, and support for HTML5“, heißt es in der neuen Version 7 der Java-Enterprise-Spezifikation. Ein HTML5-API sucht man allerdings vergebens. Was also genau ist mit HTML5-Support gemeint und wo findet sich dieser in den ca. dreißig Einzelspezifikationen wieder? Ein Blick hinter die Kulissen verrät mehr.

Keine Lösung ohne Problem

Um die oben gestellte Frage beantworten zu können, ist zunächst einmal ein kurzer Blick zurück in die *reale* Welt der Java-EE-Programmierung notwendig. Zwar wurde mit den beiden Versionen 5 und 6 eine deutliche Vereinfachung des Programmiermodells und damit einhergehend auch eine verbesserte Produktivität erreicht. Parallel dazu hat sich aber insbesondere im Webumfeld die Realität deutlich schneller entwickelt als die Spezifikation. HTML5 und Co. sind, nicht zuletzt auch dank „Mobile Web“ (es soll an dieser Stelle nicht die eher philosophische Frage untersucht werden, ob es so etwas überhaupt gibt) nicht mehr wegzudenken. RESTful Webanwendungen lassen JSON zum De-facto-Standard für die Webkommunikation werden, und mit WebSocket hält ein mehr oder minder neues Kommunikationsparadigma Einzug in die Wunderwelt des Webs. In allen eben genannten Feldern war die Java-EE-Spezifikation bisher – also vor Java EE 7 – eher schwach bis gar nicht vertreten, sodass in der Regel die Nutzung von proprietären Libraries notwendig wurde. Dies wird sich nun grundlegend ändern – so die Hoffnung. Dass dies auch die Community so sieht, zeigte eine auf java.net durchgeführte Umfrage nach den wichtigsten neuen Features in Java EE. Wenig überraschend steht hier HTML5 – inklusive JSON-API und WebSocket – mit 90 Prozent unangefochten auf Platz eins.

Porträt

W-JAX Speaker

Lars Röwekamp ist Geschäftsführer der open knowledge GmbH und berät seit mehr als zehn Jahren Kunden in internationalen Projekten rund um das Thema Enterprise Computing.

@mobileLarson

Arne Limburg ist Softwarearchitekt bei der open knowledge GmbH in Oldenburg. Er verfügt über langjährige Erfahrung als Entwickler, Architekt und Consultant im Java-Umfeld und ist auch seit der ersten Stunde im Android-Umfeld aktiv.

@ArneLimburg

JSON

Das auf einem Subset von JavaScript basierende JSON-Format (auch bekannt als JavaScript Object Notation) wird nicht zuletzt aufgrund seiner – im Vergleich zu XML – sehr kompakten Schreibweise auch als Datenaustauschformat immer beliebter. Da wundert es kaum, dass sich in vielen Enterprise-Java-Anwendungen Third Party Libraries wie org.json, JSON Tools, Jettison oder Jackson finden, die das Parsen und Schreiben von JSON-Daten ermöglichen.

Mit dem neuen Java-API for JSON Processing (JSR-353) soll dem bisherigen Wildwuchs ein Ende bereitet werden. Mithilfe eines Fluent API und eines *JsonObjectBuilder*s lassen sich „on the fly" JSON-Objekte erstellen und anschließend via *JsonWriter* in einen Character- oder Byte-Stream schreiben. Alternativ kann auch ein *JsonGenerator* für diese Aufgabe genutzt und so der Umweg über das JSON-Objektmodell umgangen werden.

Natürlich lassen sich mit dem neuen API die JSON-Objekte bzw. Streams nicht nur schreiben, sondern auch parsen. Hierbei ist zu unterscheiden, ob die Daten beim Parsen in ein JSON-Objekt überführt werden sollen oder man gezielt einzelne Daten aus dem Stream auslesen möchte. Im ersten Fall nutzt man einen *JsonReader*, der, je nach aufgerufener *read*-Methode, als Ausgabe ein *JsonObject*, eine *JsonStructure* oder ein *JsonArray* erzeugt. Im zweiten Fall dagegen würde man auf einen *JsonParser* zurückgreifen, der nach dem Pull Parsing Programming Model agiert und so den gezielten Eingriff in den Parsing-Prozess ermöglicht. Listing 1 zeigt exemplarisch das neue JSON-API in Aktion beim Erstellen eines JSON-Objekts mittels *JsonObjectBuilder*.

Etwas ärgerlich ist, dass es – anders als zum Beispiel bei XML – bisher kein eigenes Binding-API für JSON-Objekte gibt. Aber was nicht ist, kann ja noch werden und ist tatsächlich bereits für die kommende Java-EE-Version geplant.

Standardisiertes Realtime-Web

Mit dem JSR-356 – Java-API for WebSocket – und der zugehörigen Referenzimplementierung Tyrus – hält das „Realtime-Web" Einzug in die Java-EE-Spezifikation. Ziel des JSRs ist es, eine echte bidirektionale und somit gleichberechtigte Kommunikation zwischen Client und Server zu ermöglichen, die vom klassischen Request-/Response-Modell abstrahiert und so auch dem Server erlaubt, Clients gezielt Messages zu senden. Bisher war dies nur über Tricks bzw. Workarounds, wie zum Beispiel Client Side Polling oder Long Polling (Stichwort: Comet) möglich. Während die erste Variante unnötig viele Requests und somit auch unnötig viel Traffic verursacht, wird bei der zweiten Variante ein Request künstlich so lange am Leben gehalten, bis der Server etwas für den Client hat, was serverseitig unnötig Ressourcen bindet. Beide Varianten skalieren also nur bedingt. Anders bei WebSocket: Hier wird eine Session zwischen Client und Server aufgebaut und von diesem Moment an die Verbindung nur dann „belastet", wenn es zum realen Datenaustausch kommt – in welche Richtung auch immer. Der damit verbundene Vorteil macht sich besonders bei einer großen Anzahl von Verbindungen bemerkbar [1].

Listing 1: JsonObjectBuilder in Action

```
// JSON structure to create
// {
//   "title":"Epic",
//   "year":2013,
//   "cast":[
//       "Amanda Seyfield",
//       "Josh Hutcherson"
//   ]
// }
new new JsonObjectBuilder()
  .add("title", "Epic")
  .add("year", 2013)
  .add("cast", new JsonArrayBuilder()
  .add("Amanda Seyfield")
  .add("Josh Hutcherson"))
.build();
```

Das neue API bietet alles Wesentliche, um WebSocket auch im Java-EE-Umfeld nutzen zu können. Dabei wurde bewusst auch ein Client-API zur Verfügung gestellt, um so die gleichen Konzepte auf beiden Seiten des Kommunikationskanals realisieren zu können. Neben Verbindungsaufbau und -abbau sowie dem eigentlichen Datenaustausch sind insbesondere die Encoder und Decoder erwähnenswert. Mit ihrer Hilfe kann – neben Text- und Binärdaten – auch das eigene Objektmodell problemlos verarbeitet werden. Listing 2 zeigt einen einfachen WebSocket-Endpoint, der mittels Encoder/Decoder ein eigenes Message-Format verarbeiten kann.

JSF als HTML-5-Schleuse

So weit, so gut – oder auch nicht. Denn auch wenn durch die beiden bisher genannten JSRs 353 (JSON) und 356 (WebSocket) wichtige Neuerungen in den Java-EE-Standard eingeflossen sind, bleibt nach wie vor eine Frage offen: Wie erzeuge ich mit dem Enterprise-Java-Standard HTML5-konforme Webseiten? Oder anders gefragt: Wie nutze ich die neuen HTML5-Features – zum Beispiel den vereinfachten DOCTYPE, die zusätzlichen Form- und Input-Attribute oder aber die Custom-Data-Attribute? Bisher, d. h. mit JSF-2.1-Bordmitteln, war dies nicht so einfach möglich – und wenn, dann nur unter Verwendung eines Third-Party-Render-Kits, wie zum Beispiel dem von OmniFaces [2].

Abhilfe schafft hier der JSR-344, also die neue JSF-Spezifikation 2.2. Dank der neuen JSF-Variante baut der Webentwickler keine JSF-Seiten mehr, die zufällig ein wenig HTML beherbergen, sondern geht – wenn er möchte – genau den umgekehrten Weg. Dank eines neuen Namespaces lässt sich JSF nahtlos in bestehendes HTML (5) integrieren, indem die HTML-Tags einfach um *jsf*-Attribute, wie zum Beispiel *jsf:id* oder *jsf:value* angereichert werden (Listing 3). Der Vorteil liegt klar auf der Hand: Der Webauftritt kann vorab in HTML designt und nachträglich über *jsf*-Attribute dynamisiert werden. Dabei muss keine Rücksicht mehr auf eventuell durch JSF-Tags „verunreinigten" HTML-Code genommen werden. Wahrscheinlich haben Sie es schon gemerkt: Durch diesen Ansatz bekommt man die oben genannten HTML-Features geschenkt, da die HTML-Tags unverändert in der Webseite erhalten bleiben (Listing 3 – Attribut *placeholder*).

Was ist nun aber, wenn man nicht mit dem neuen Namespace arbeiten möchte oder kann? Oder anders gefragt: Wie stellt man bestehende JSF-Anwendungen auf HTML5 um? Auch hier bietet JSF 2.2 eine Lösung namens „Passthrough". Mithilfe der beiden neuen Tags

- *<f:passThroughAttribute ... />*
- *<f:passThroughAttributes ... />*

können die „unbekannten" HTML5-Attribute 1:1 bei den klassischen JSF-Eingabekomponenten einfach mit angegeben und so durchgeschleust werden, ohne dass das Eingabe-Tag diese tatsächlich kennen muss. Alternativ kann auch ein neuer Namespace http://java.sun.com/jsf/passthrough eingebunden und dann innerhalb des JSF-Eingabe-Tags das entsprechende Attribut direkt referenziert werden.

Listing 2: WebSocket Endpoint mit eigenem Decoder/Encoder

```
// Encoder/Decoder class reference in ServerEndpoint
@ServerEndpoint (
  value="/myEndpoint",
  encoders=MyEncoder.class,
  decoders=MyDecoder.class)
...

// Encoder/Decoder allows to work with object model
@OnMessage
public void messageReceived(MyObject msg, Session s) {
  for (Session connected : connectedSessions) {
    try {
      connected.getBasicRemote().sendObject(msg);
    } catch (IOException | EncodeException e) {...}
  }
}
```

Listing 3: HTML 5 trifft JSF 2.2

```
<!DOCTYPE html>
<html xmlns="http://ww.w3.org/1999/xhtml" ...
      xmlns:jsf="http//java.sun.com/jsf/>
<head jsf:id="head"><title>...</title></head>
<body jsf:id="body">
  <form jsf:id="form">
    <label>Vorname</label>
    <input type="text"
        jsf:id="firstName"
        jsf:value="#{createCustBean.firstName}"
        placeholder="Vorname">
      <f:validateRequired />
    </input>
...
  </form>
</body>
</html>
```

Fazit

Alles in allem sind die Neuerungen innerhalb der Java-EE-7-Spezifikation in Richtung HTML5 und Co. ein wichtiger und richtiger Schritt. Zum einen spiegeln sie Features wider, die bisher nur über proprietäre Third-Party-Libraries zugänglich waren (z. B. JSON Parsing), zum anderen spezifizieren und vereinheitlichen sie Details, die ohnehin schon bei so gut wie jeder Implementierung als Zusatz vorhanden waren.

Insbesondere im JSF-Umfeld fällt auf, dass man zum Glück nicht versucht hat, dem stetigen Fortschritt hinterherzurennen – dieses Rennen verliert ein Standard per Definition – sondern via Passthrough und neuem Namespace bewusst einen Weg geht, der offen für Erweiterungen zu sein scheint. Auf den ersten Blick ein vielversprechender Ansatz. Ob dieser letztendlich greift, werden wir in ein paar Monaten sehen. In diesem Sinne: Stay tuned.

Links & Literatur

[1] Introducing to WebSocket: http://www.codeproject.com/Articles/531698/Introduction-to-HTML5-WebSocket

[2] OmniFaces HTML 5 RenderKit: http://omnifaces.org

Integrierte Prozessplattformen mit Activiti und Camel

Zu Ende gedacht

Open-Source-Frameworks und Produkte zur Implementierungen von Prozessen gibt es viele. Integrierte Plattformen eher nicht. Erfahren Sie, wie sich Activiti und Camel zu einer integrierten Prozessplattform zusammenführen lassen.

von Wolfgang Pleus

In der Softwareentwicklung ist seit langer Zeit ein Trend zu beobachten, bei dem sich die Repräsentation von Quellcode an den Konzepten der Wirklichkeit orientiert. Objekte orientieren sich an realen Dingen wie beispielsweise Versicherungsverträgen, Services an realen Dienstleistungen wie Tarifberechnungen und Prozesse an realen Abläufen wie beispielsweise einer versicherungsfachlichen Policierung. Die damit verbundene erhöhte Abstraktion erleichtert die Kommunikation zwischen Technikern und Nichttechnikern und kann zu verständlichem und intuitivem Quellcode führen. Besonders gut lässt sich das im Bereich der Automatisierung von Geschäftsprozessen und Workflows beobachten. Hierbei werden Prozesse beispielsweise mit BPMN modelliert und ausgeführt. So kann im Idealfall eine gemeinsame Sprache zwischen Fachabteilung und IT etabliert werden.

Eine prozessorientierte Betrachtungsweise findet sich ebenfalls im Bereich der Systemintegration. So bieten gängige Enterprise-Application-Integration-Werkzeuge (EAI) oder Enterprise-Service-Bus-Produkte (ESB) ebenfalls die Möglichkeit, Prozesse entweder grafisch oder mithilfe von domänenspezifischen Sprachen (DSL) oder Fluent APIs zu beschreiben beziehungsweise zu implementieren. Diese Prozesse werden Integrations- oder auch Mediationsprozesse genannt. Die Grenzen zwischen deklarativer und imperativer Entwicklung sind dabei fließend. Laut einer aktuellen Forrester-Studie zum Thema BPM Suites [1] gibt es einen Trend in Richtung integrierter End-to-End BMP Suites, die sowohl benutzerzentrierte (human-centric), als auch integrationszentrierte (integration-centric) Prozesse unterstützen. Und das ist sinnvoll, denn wenn es in der Praxis um Prozessautomatisierung geht, kommen fast immer Aspekte aus den beiden Bereichen Workflow (Geschäftsprozesse) und Mediation (Integration) zum Tragen. Das liegt daran, dass ausführbare Prozesse in jeder nicht trivialen Umgebung mit bestehenden oder neuen Backend-Systemen interagieren müssen. Dazu gehören beispielsweise Datenbanken, CRM-Systeme oder auch Mainframes. Nicht selten werden die genannten Aspekte von unterschiedlichen Produkten abgebildet. Dies führt in der Regel zu Systembrüchen und unnötiger Komplexität.

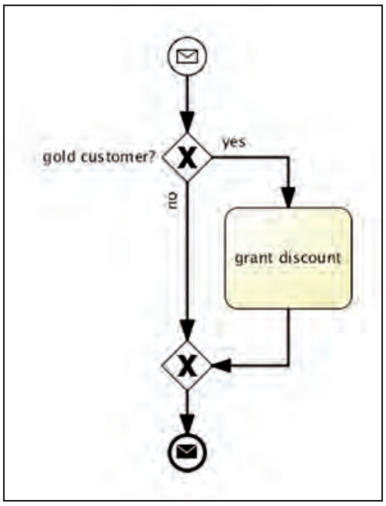


Abb. 1: BPMN-Prozess

Vermieden werden kann das durch den Einsatz einer integrierten Prozessplattform, die die Aspekte Mediation und Workflow möglichst ohne Systembrüche unterstützt. Eine solche Plattform kann durch die Kombination von Apache Camel und Activiti realisiert werden. Camel punktet dabei im Bereich der Integration und Activiti im Bereich langlaufender Workflows und Benutzerinteraktion.

Technik soll natürlich kein Selbstzweck sein. Daher ist es wichtig, die Rolle der eingesetzten Technologien klar abzugrenzen. **Abbildung 1** zeigt einen einfachen Prozess in BPMN-Notation. Listing 1 zeigt den gleichen Prozess in einer Camel-DSL-Notation.

Es ist erkennbar, dass beide Technologien grundsätzlich in der Lage sind, die Prozesslogik abzubilden. Wofür soll man sich also entscheiden? BPMN-Verfechter werden jetzt die grafische Darstellung von BPMN ins Feld führen. Dieser vermeintliche Vorteil kann bei komplexen Prozessmodellen jedoch auch zum Nachteil werden, da grafische Modelle bei hoher Komplexität eher schwer zu lesen sind. Tabelle 1 zeigt bewährte Faktoren zur Abgrenzung der Einsatzgebiete.

Da automatisierte Prozesse meist mehrere der Merkmale aus Tabelle 1 umfassen, erscheint die Kombination von Activiti und Camel sinnvoll. Dabei geht es darum, die beiden Technologien möglichst nahtlos miteinander zu verbinden, sodass eine Prozessplattform entsteht, die eine ganzheitliche Sicht auf die ausgeführten Prozesse erlaubt. Eine solche Plattform ist in der Lage, einen Prozess, wie in **Abbildung 2** gezeigt, zu automatisieren.

Bei diesem Prozessbeispiel geht es um eine einfache Taxireservierung. Ein Kunde kann den Prozess entweder direkt durch Senden einer E-Mail (1) oder indirekt durch den Anruf in einem Callcenter (2) starten. Die orangen Pools zeigen den operativen Teil der Disponenten und Kunden. Sie dienen dazu, die rein fachliche Perspektive der handelnden Akteure im Prozessmodell zu erhalten. Diese Teile werden technisch nicht abgebildet. Der blaue Workflow-Pool zeigt den durch Activiti automatisierten Workflow. Die grünen Pools zeigen die durch Camel automatisierten Integrationsteile. Das Modell hat analytischen Charakter und zeigt den Prozess im Ganzen. Bei der Automatisierung wird der blaue Pool mit Activiti ausgeführt, während die grünen Pools in Camel-Routen implementiert werden.

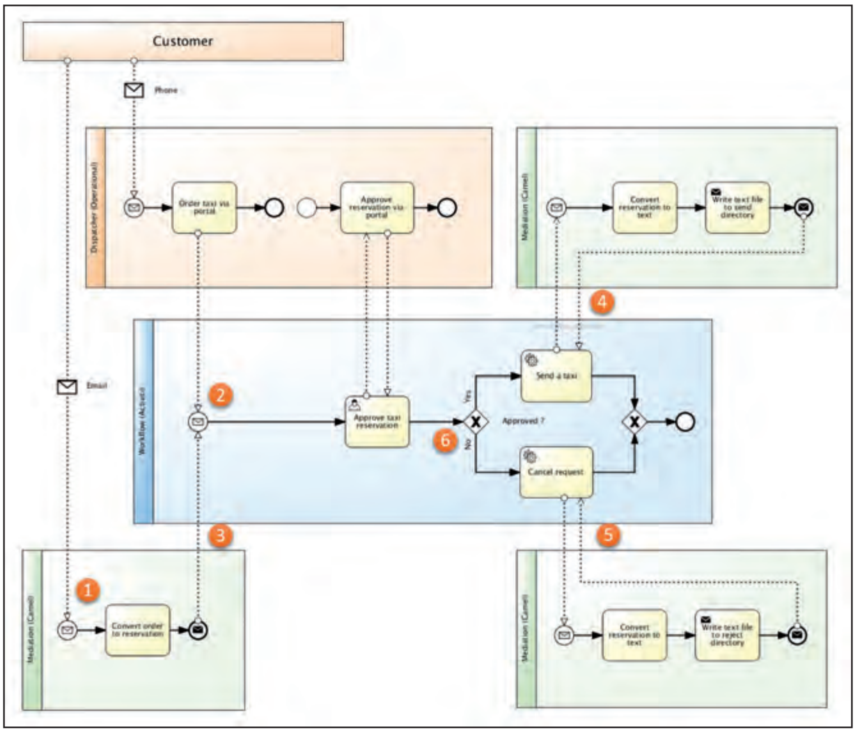


Abb. 2: End-to-End-Prozess

Von Camel zu Activiti

Um den Prozessentwickler von wiederkehrenden Aufgaben zu entlasten, empfiehlt es sich, die Technologieübergänge einmalig als Teil der Prozessplattform zu implementieren. Bei dem ersten Übergang (3) soll ein Activiti-Prozess aus einer Camel-Route gestartet werden. Im Fall einer Korrelation kann es ebenfalls vorkommen, dass eine existierende Prozessinstanz nach Eintreffen einer Nachricht fortgesetzt werden muss. Diese Anwendungsfälle lassen sich über eine spezifische Camel-Komponente realisieren. Die Komponente wird mit dem Präfix *workflow://* registriert. So kann der Prozess wie in Listing 2 gezeigt gestartet werden.

Es fällt auf, dass das *to*-Statement keinen Prozessnamen, sondern nur das reservierte Wort *start* enthält. Dies reicht aus, wenn der Activiti-Workflow zusammen mit der Camel-Route in einem Deployment-Artefakt installiert wird und der Prozessname dadurch implizit bekannt ist. Innerhalb des Producers der Camel-Komponente wird der Activiti-Prozess über die Activiti-Methode *RuntimeService.startProcessInstanceByKey* gestartet. Das folgende Beispiel zeigt die Fortsetzung einer gestarteten Prozessinstanz nach einem Nachrichteneingang, wobei die Korrelations-ID einem fachlichen oder technischen Attribut entspricht, das die Prozessinstanz eindeutig identifiziert.

```
from("file://start")
 to("workflow://continue?reservation=1234");
```

Innerhalb der Camel-Komponente wird der Activiti-Prozess über die Methode *RuntimeService.signal* fortgesetzt. Die erforderliche Prozessinstanz wird zuvor über *RuntimeService.createExecutionQuery* ermittelt. Die Entwicklung von Camel-Komponenten war schon Thema im Java Magazin (bspw. in den Ausgaben 8.2012 und 9.2012) und wird daher hier nicht weiter betrachtet. Weiterführende Informationen zu diesem Thema finden sich unter [2].

Listing 1

```
from("file://start")
  .choice()
    .when(header("goldCustomer").isEqualTo("true"))
      .to("bean:processor?method=grantDiscount")
      .to("direct:out")
    .otherwise()
      .to("direct:out");
from("direct:out")
  .to("file://end");
```

Von Activiti zu Camel

Weitere Technologieübergänge (4 und 5) sind in **Abbildung 2** zwischen Activiti und Camel zu erkennen. Dies-

Merkmal	Apache Camel	Activiti
Langlaufende, statusbehaftete Prozesse		X
Human Workflow/Benutzerinteraktion		X
Adaptoren/Konnektivität	X	
Datentransformation	X	
Fertige EIP Pattern	X	

Tabelle 1: Abgrenzung Camel und Activiti

Listing 2: Starten eines Workflows via Camel

```
import org.apache.camel.builder.RouteBuilder;

public class FileToWorkflow extends RouteBuilder{
  public void configure(){
    from("file://start")
    .to("workflow://start");
  }
}
```

Listing 3: Aufruf von Camel via Activiti

```
// workflow.bpmn
<?xml version="1.0" encoding="UTF-8"?>
  <definitions ...>
    <process id="taxi">
      <servicetask id="send" activity:delegateExpression="${mediation}"/>
    </process>
  </definitions>

  // Java code
  package net.pleus.demo;
  import org.activiti.engine.impl.pvm.delegate.ActivityBehaviour;
  import org.activiti.engine.impl.pvm.delegate.ActivityExecution;
  import org.activiti.engine.imple.javax.el.ELResolver;
  import org.apache.camel.Exchange;

  public class MediationActivityBehaviour implements ActivityBehaviour{
    public void execute(ActivityExecution execution) throws Exception{
      // Erzeugen Endpunktname passend zur ServiceTask-ID
      String key = "workflow://" + execution.getActivity().getId();
      // Erzeugen Camel Exchange
      Exchange exchange = new DefaultExchange(getCamelContext());
      // Konvertieren Activiti-Nachricht zu Camel-Nachricht
      convertMessage(execution,exchange);
      // Nachricht an den Camel-Endpunkt senden
      getCamelContext().createProducerTemplate().send(key,exchange);
    }
  }
  public class MediationELResolver extends ELResolver{
  public Object getValue(ELContext ctx, Object base, Object property){
    // Behaviour für delegateExpression = "mediation" ermitteln
    if (property.equals("mediation")){
      return new MediationActivitiBehaviour();
    }
    ...
  }
}
// spring.xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans">
  <bean id="processEngineConfiguration" class="org.activiti.engine.impl.cfg.
                                StandaloneInMemProcessEngineConfiguration">
    <property name="expressionManager">
      <!— ELResolver registrieren -->
      <bean class="net.pleus.demo.MediationELResolver"/>
    </property>
  </bean>
</beans>
```

mal werden Daten von Activiti an eine Camel-Route übergeben. Eigene Camel-Routen können an Activiti-, bzw. BPMN-ServiceTasks wie folgt angehängt werden:

```
from("workflow://send")
.to("file://send");
```

Sobald eine Nachricht den *ServiceTask* mit der ID *send* erreicht, wird automatisch die Camel-Route aufgerufen. Zu erkennen ist dies an Punkt 4 in **Abbildung 2**. Um das zu erreichen, wird der *ServiceTask* in der BPMN-Datei mit einer *delegateExpression* versehen:

```
<servicetask id="send" activity:delegateExpression="${mediation}"/>
```

Die *delegateExpression* verwendet die statische Expression *${mediation}*. Diese wird von Activiti zur Laufzeit über einen *ELResolver* aufgelöst, der eine *ActivitiBehaviour*-Implementierung ermittelt. *ActivitiBehaviours* dienen in Activiti neben *JavaDelegates* dazu, Tasks mit Java zu implementieren. In der *execute*-Methode der Klasse *ActivitiBehaviour* wird der zum *ServiceTask* passende Camel-Endpunkt ermittelt. Die Activiti-Nachricht wird in eine Camel-Nachricht umgewandelt. Mit dieser Nachricht wird dann der Camel-Endpunkt aufgerufen und so die Kontrolle an Camel übergeben. Damit Activiti den *ELResolver* zur Laufzeit verwenden kann, muss dieser über die *ProcessEngineFactory* konfiguriert werden. Listing 3 verdeutlicht den Zusammenhang.

Durch diesen einmaligen Integrationsaufwand innerhalb der Prozessplattform gestaltet sich die Verbindung von Camel-Routen und Activiti für den Prozessentwickler vergleichsweise einfach.

Einheitliche Datenrepräsentation

Sowohl Camel als auch Activiti verwalten einen Zustand während der Prozessausführung. In unserem Beispiel können das Informationen über die Taxibestellung und den Kunden sein. Dieser Zustand wird im Prozess mitgeführt und kann programmatisch gelesen und verändert werden. Insbesondere bei langlaufenden Prozessen wird der Zustand zwischen den Verarbeitungsschritten persistiert. Camel transportiert den Zustand in einer Message-Struktur. Diese Struktur umfasst Header (*Map<String,Object>*), einen Body (*Object*) sowie Attachments (*Map<String, javax.activation.DataHandler>*). Die fachliche Nutzlast ist im Body enthalten. Im Falle einer E-Mail ist das der Nachrichtentext. Die Header enthalten zusätzliche Informationen, wie beispielsweise den Absender der Nachricht. Falls die E-Mail Anhänge enthält, werden diese als Attachments transportiert. Activiti erlaubt den Transport einer Liste benannter Eigenschaften beliebigen Typs (*Map<String,Object>*). Um eine möglichst nahtlose Integration zu erreichen, müssen alle Eigenschaften der Camel-Nachricht über die *ActivitiExecution* transportiert werden. Da eine Camel Message nicht serialisierbar ist, muss sie in eine serialisierbare Struktur überführt werden, bevor sie als Variable der *ActivitiExecution* übergeben wird. Listing 4 verdeutlicht den Ansatz.

Durch die standardisierte Struktur kann die Typumwandlung zwischen Camel und Activiti automatisch erfolgen, ohne dass Informationen verloren gehen. Der Zugriff auf die Nutzlast erfolgt immer nach dem gleichen Prinzip. Das Gateway (6) aus **Abbildung 2** kann beispielsweise mit Ausdrücken wie *${pm.headers["rom"]=="max@pleus.net"} oder ${pm.body.approved}* konfiguriert werden. Alternative 1 greift auf die Header der Camel-Mail-Komponente zu, um den Sender zu ermitteln. Alternative 2 verwendet den Body der Nachricht. In diesem Fall muss der Body einen Typ enthalten, der über die Methode *isApproved* verfügt. Ein solcher Ansatz ermöglicht den standardisierten Zugriff auf die Inhalte. Dies vereinfacht die Prozessentwicklung und ist eine wichtige Voraussetzung, um eine deklarative Prozessüberwachung zu realisieren.

Prozessüberwachung

Die Ausführung komplexer Prozesse erfolgt meist in verteilten Umgebungen. Langlaufende Prozesse können Tage oder Wochen benötigen, bis sie abgeschlossen sind. Dabei erfolgen viele Kontextwechsel. Beispielsweise wird der Prozess auf einer Maschine initiiert, manuell von einem Sachbearbeiter bearbeitet und später auf einer weiteren Maschine abgeschlossen. Einen Call-Stack, der den gesamten Ablauf nachvollziehbar macht, gibt es nicht. Um den Prozessfluss sichtbar zu machen, ist so genanntes *Flow Tracking* erforderlich. Dabei wird die Ausführung einzelner Prozessschritte protokolliert. Diese Protokolle werden den jeweiligen Anwendern in aufbereiteter Form zur Verfügung gestellt. Da sich die Prozesse in der Regel an fachlichen Abläufen orientieren, ist das Flow Tracking neben Entwicklern und Betriebsmitarbeitern ebenfalls für Personen aus den Fachbereichen interessant. Zusätzlich zu technischen Informationen lassen sich aus den Protokollen interessante Erkenntnisse in Bezug auf fachliche Kenngrößen wie beispielsweise Umsatzvolumina oder regionale Verteilungen ableiten. Ein grafisch aufbereiteter Flussgraph für den Taxiprozess wird in **Abbildung 3** gezeigt.

In diesem Beispiel wird das Taxi per E-Mail bestellt und vom Disponenten freigegeben. Während das Prozessmodell alle möglichen Ausführungspfade zeigt, ist im Flussgraph der tatsächlich ausgeführte Pfad enthalten. Es ist sozusagen der Call-Stack des Prozesses. Durch die Überlagerung von Prozessmodell und Flussgraph wird es möglich, Anwendern den Zustand laufender Prozesse im Kontext zu visualisieren. Bei dem Flussgraphen handelt es sich um eine End-to-End-Betrachtung, die den gesamten Prozessablauf über Camel und Activiti zeigt. Diese Betrachtungsweise erfüllt das Informationsbedürfnis der verschiedenen Anwender in optimaler Weise, da es für maximale Transparenz sorgt.

Um einen Flussgraphen zu erzeugen, müssen bei jeder Ausführung eines BPMN-Tasks sowie eines Camel-Schritts Informationen über den jeweiligen Schritt gespeichert werden. Um alle Ausführungsschritte bei der Visualisierung zu einem Flussgraphen zu verbinden, muss im Header der *ProcessMessage* eine eindeutige GUID zur Identifikation der Prozessinstanz (*piid*) transportiert werden. Um eine robuste Vorgängerbeziehung aufzubauen, ist ebenfalls eine Instanz-ID für jeden ausgeführten Prozessschritt zu transportieren.

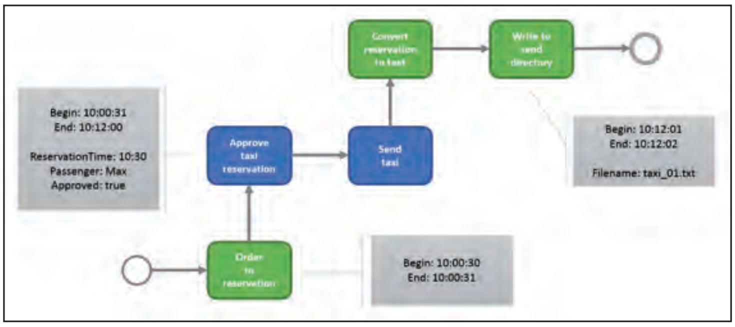


Abb. 3: Flussgraph

Zur Aufzeichnung der BPMN-Tasks wird der Activiti-Konfiguration, wie bereits am Beispiel des *ELResolvers* in Listing 3 gezeigt, eine Implementierung des Interface *BpmnParseListener* als *PostParseListener* hinzugefügt. Während des BPMN-Parsings fügt der *ParseListener* allen Aktivitäten einen *ExecutionListener* hinzu, der bei jeder Ausführung eines BPMN-Tasks aufgerufen wird und so die Tracking-Daten protokollieren kann. Die Implementierung der Listener wird in Listing 5 exemplarisch gezeigt.

Listing 4: Aufruf von Camel via Activiti

```
import org.apache.camel.Exchange;
import org.apache.camel.Message;
import javax.activation.DataHandler
import org.activiti.engine.impl.pvm.delegate.ActivityExecution;

// Serialisierbarer Nachbau der Camel Message
public class ProcessMessage implements Serializable{
  private Map<String,String> headers;
   private Object body;
  private Map<String,byte[]> attachments;
}

// Konvertierung Camel-Nachricht in Activiti-Nachricht
public void convertMessage(ActivityExecution execution, Exchange
                                                          exchange){
  // Lesen der Camel-Nachricht
  Message message = exchange.getIn();
  Map<String,Object> headers = message.getHeaders();
  Map<String, DataHandler> attachments = message.getAttachments();
  Object body =  message.getBody();
  // Konvertieren der Camel message in eine ProcessMessage
  ProcessMessage pm = new ProcessMessage(headers, body, attachments);
  // Setzen ProcessMessage für Activiti
  execution.setVariable("pm",pm);
}
```

Listing 5: BPMN-Tracking

```
import org.activiti.enginge.delegate.ExecutionListener;
import org.activiti.enginge.impl.bpmn.parser.BpmnParseListener;

// Execution Listener
public class TrackStart implements ExecutionListener{
  public void notify(DelegateExecution execution){
    ProcessMessage pm = (ProcessMessage) execution.getVariable("pm");
    String piid = pm.getHeaders().get("piid");
    // Track message
    trackMessage(piid,pm);
  }
}

// Parse Listener
public class TrackingParseListener implements BpmnParseListener{
  private trackStart = new TrackStart();
  @Override
  public void parseTask(..., ActivityImpl activity){
    activity.addExecutionListener(PvmEvent.EVENTNAME_START,trackStart,0);
  }
  ...
}
```

Listing 6: Camel-Tracking

```
import org.apache.camel.spi.InterceptStrategy;
import org.apache.camel.processor.DelegateAsyncProcessor;
import org.apache.camel.model.ProcessorDefinition;
import org.apache.camel.Processor;
import org.apache.camel.Exchange;

public class TrackingProcessor implements Processor{
  @Override
  public void process(Exchange exchange) throws Exception{
    Message message = exchange.getIn();
    String piid = Message.getHeader("piid");
    // Track message
    trackMessage(piid,message);
  }
}

public class TrackingInterceptor implements InterceptStrategy{
  @Override
  public Processor wrapProcessorInInterceptors(...,
                final ProcessorDefinition<?> definition,
                final Processor target){
    return new DelegateAsyncProcessor(new TrackingProcessor(definition,
                                                                target));
  }
}

private void setup(){
  CamelContext ctx = getCamelContext();
  ctx.addInterceptStrategy(new TrackingInterceptor());
}
```

Die Aufzeichnung der Camel-Ausführung erfolgt über Interzeptoren. Diese werden den Camel-Routen hinzugefügt. Die Interzeptoren erzeugen Prozessoren, die bei jeder Ausführung eines Camel-Prozessschritts aufgerufen werden und dabei die Tracking-Daten protokollieren. Listing 6 zeigt den Ansatz in vereinfachter Form.

Genau wie beim Tracking des BPMN-Prozesses wird die Nachricht aus dem Ausführungskontext extrahiert. Die einheitliche Repräsentation der Daten und automatische Konvertierung an den Technologiegrenzen erlaubt eine generische Verarbeitung. Um das Aufkommen der Tracking-Daten zu begrenzen, ist es sinnvoll, einen Mechanismus zur selektiven Verarbeitung zu realisieren. Beispielsweise können Teile der Prozessnachrichten durch MVEL Expressions [3] extrahiert werden. Eine Entkopplung des Trackings innerhalb der Methode *trackMessage* beispielsweise über JMS-Queues ist ebenfalls zu empfehlen, um die Performance der ausgeführten Prozesse nicht zu beeinträchtigen. Da durch das Tracking sehr umfangreiche Daten anfallen können, empfiehlt sich eine performante Art der Speicherung und Suche, wie beispielsweise über einen Lucene-Index [4].

Fazit

Die Bedeutung integrierter Prozessplattformen wird in Zukunft vermutlich zunehmen. Dieser Artikel beschreibt ausgewählte Aspekte für die Entwicklung einer solchen Plattform auf Basis von Open-Source-Produkten. Weiterführende Themen, wie beispielsweise Transaktionsmanagement, Prozessdeployment oder Prozessversionierung, sind von einer solchen Plattform ebenfalls zu leisten. Eine Plattform, die die beschriebenen Ansätze bereits implementiert, ist oparo [5]. Sie steht allen Interessierten unter der liberalen Apache-2-Lizenz zur Verfügung.

Wolfgang Pleus arbeitet als Technologieberater, Autor und Trainer im Bereich moderner Softwarearchitekturen. Seit zwanzig Jahren unterstützt er internationale Unternehmen bei der Realisierung komplexer Geschäftslösungen auf der Basis von Java EE und .NET. Seine Schwerpunkte liegen in den Bereichen SOA, BPM und Agile.

Links & Literatur

[1] Forrester Wave, Business Process Management Suites, Q1 2013: http://ibm.co/ZWp086

[2] Writing Components: http://camel.apache.org/writing-components.html

[3] MVEL: http://mvel.codehaus.org

[4] Lucene: http://lucene.apache.org

[5] oparo: http://www.oparo.org

Springs Lösung für Extreme Data

Spring XD

Das Thema „Big Data“ ist in aller Munde – und natürlich darf eine passende Lösung im Spring-Portfolio nicht fehlen. Im Mittelpunkt steht aber nicht das Speichern der Daten, sondern die Verarbeitung. Spring XD [1] steht für „Spring Extreme Data“ und basiert auf Lösungen wie Spring Integration, Spring Data und Spring Batch. Spring XD ergänzt sie um Support für einige Big-Data-Technologien, eine DSL für Konfigurationen – und eine Container-Architektur.

von Eberhard Wolff

Meistens denkt man bei „Big Data“ an die Speicherung großer Datenmengen. Aber die Daten müssen auch verarbeitet werden. Schließlich gehen Daten mit Datenströmen einher – für den Import oder den Export in oder aus Big-Data-Speichern. Dabei können auch schon die ersten Verarbeitungsschritte stattfinden – beispielsweise das Zählen, Aufsummieren oder Verdichten der Werte. Eine Verarbeitung der Daten als Batch ist genauso denkbar wie die Onlineverarbeitung. Für solche Szenarien existieren schon die Frameworks Spring Batch und Spring Integration. Sie sind jedoch nicht miteinander integriert. Spring XD stellt eine solche Integration bereit und ergänzt sie um eine Ablaufumgebung ähnlich einem Application Server. Grundsätzlich verwendet Spring XD dazu folgende Elemente:

- Jobs sind Batch-Verarbeitungen. Sie können mit Spring Batch als klassische Batches umgesetzt werden, wie sie von Großrechnern her bekannt sind. Eine Alternative ist die Implementierung als MapReduce-Job mithilfe von Spring for Apache Hadoop. Beiden Ansätzen ist gemeinsam, dass sie eine große Datenmenge von irgendwoher einlesen und dann verarbeiten. Jobs sind im aktuellen Stand des Codes noch nicht enthalten. Sie werden aber definitiv ein Teil von Spring XD werden.
- Bei Streams werden im Gegensatz zu Jobs nicht alle Daten auf einmal übertragen. Stattdessen wird jeder einzelne Datensatz übertragen und verarbeitet. Streams transportieren Daten von einer Source zu einem Sink. Mögliche Sources sind HTTP, ein Tail auf eine Datei, eine Suche bei Twitter, eine Continuous Query im Gemfire Cache von Pivotal, Syslog Events oder Werte, die über TCP eingehen. Liegen in den Sources neue Werte vor, werden sie direkt an die Sinks geschickt. Sinks können ein Log über ein Logging-Framework, eine Datei im Dateisystem des Computers, eine Datei im Hadoop-Dateisystem HDFS, ein TCP-Port oder der GemFire-Server sein.
- Außerdem gibt es Taps, mit denen die Verbindung zwischen einer Source und einem Sink „abgehört“ werden können. Neben der eigentlichen Verarbeitung können so noch weitere Verarbeitungen angestoßen werden.

Die Streams werden bereits im ersten Meilenstein [2], [3] unterstützt. Der Umgang mit dem Server gestaltet sich recht einfach. Streams können mit einer DSL beschrie-

In Zukunft sollen auch andere Infrastrukturen unterstützt werden. Dazu zählt das CloudFoundry-Angebot von Pivotal.

ben werden, die an UNIX-Pipes erinnert. So verbindet *http | file* die Source HTTP mit dem Sink File. Es können auch Optionen angegeben werden, beispielsweise *http --port=9020 | file --name=eineDatei.txt --dir=/tmp*.

Als Schnittstelle bietet Spring XD ein HTTP-REST-Interface an. Die DSL-Ausdrücke müssen also lediglich per HTTP an den Server übertragen werden. Nach dem Start des Single-Node-Servers mit *xd-singlenode* kann also mit dem REST-Interface ein Stream angelegt werden – z. B. mit *curl*:

```
curl -d "http --port=9020 | file --name=eineDatei --dir=/tmp"
                              http://localhost:8080/streams/meinStream
```

Wenn nun an den Port 9020 ein HTTP-Request ankommt, wird er in die Datei geschrieben:

```
curl -d "Landet in der Datei" http://localhost:9020/
```

Entsprechend dem REST-Gedanken kann der Stream mit einem *HTTP-DELETE* wieder gelöscht werden: *curl -X DELETE http://localhost:8080 streams/meinStream*.

Dieses Interface soll in Zukunft um HATEOAS-Ansätze erweitert werden. Diese Abkürzung steht für „Hypermedia as the Engine of Application State". Dank dieses Konzepts muss ein Client nur einige wenige URLs kennen– idealerweise nur einen einzigen. Weitere Ressourcen können über Links erschlossen werden. Außerdem soll in Zukunft auch eine Shell zur Verfügung stehen, mit der Nutzer direkt mit Spring XD interagieren können, ohne das REST-Interface zu nutzen.

Weitere Features der Spring-XD-DSL

Neben der direkten Verbindung kann Spring XD auch Prozessoren in den Streams einfügen, um die Nachrichten zu verarbeiten. Spring XD hat dazu Filter oder Transformer mit der Spring Expression Language (SpEL) oder Groovy. Eine weitere Möglichkeit sind der JSON-Field-Value-Filter zum Herausfiltern von JSON-Dokumenten, die bestimmte Werte enthalten, und der JSON Field Extractor, bei dem nur ein bestimmtes Feld aus dem JSON-Dokument weitergeleitet wird. Außerdem können Groovy-Skripte als Prozessoren genutzt werden. Diese Skripte können die Nachrichten beliebig bearbeiten. In Zukunft werden in diesem Bereich sicher noch weitere Technologien unterstützt.

Ein Beispiel für einen Stream mit einem Prozessor ist *http | filter --expression=payload.contains('good') | file* . Wegen des Prozessors werden nur die Nachrichten in der Datei landen, die den String *good* enthalten. Für den Ausdruck wurde übrigens SpEL genutzt. Es ist natürlich auch möglich, mehrere Prozessoren zu kombinieren und so auch komplexere Pipelines aufzubauen, die mehrere Verarbeitungsschritte umfassen.

Sollen Nachrichten auf verschiedene Arten verarbeitet werden, so helfen Taps. Wie erwähnt, können Streams durch Taps zusätzlich an weitere Sinks und auch Prozessoren gebunden werden. Beispielsweise wird mit *curl -d "tap@meinStream | log" http://localhost:8080/streams/* die Ausgabe zusätzlich mit dem Logger im Server ausgegeben. Taps können genutzt werden, um die Daten in Echtzeit zu analysieren. Beispielsweise können Zähler für bestimmte Arten von Nachrichten auf diese Weise implementiert werden. Für Metriken dieser Art bietet Spring XD auch eine Unterstützung. Solche Daten kann Spring XD In-Memory oder in der NoSQL-Datenbank Redis ablegen. Dadurch können also aus einem Datenstrom schon bestimmte Daten herausgefischt und gezählt werden. Der URL [3] zeigt beispielsweise, wie mit Spring XD Twitter-Nachrichten nach bestimmten Schlüsselwörtern durchsucht werden können und die Anzahl der Treffer gespeichert werden kann.

Außerdem unterstützt Spring XD Tuples, bei denen neben den eigentlichen Daten zusätzliche Informationen übertragen werden können. Dadurch können also Daten im Laufe der Verarbeitung um zusätzliche Informationen angereichert werden.

Module

Wie arbeitet Spring XD intern? Sources, Sinks und Processors sind spezielle Module, die also jeweils eine bestimmte Funktionalität anbieten. Letztendlich ist ein Modul aber nichts anderes als ein Spring-*ApplicationContext*. Aus einer XML-Vorlage wird eine Spring-Konfiguration erzeugt, wenn ein Modul installiert wird. In der Vorlage sind einige Platzhalter enthalten. Sie werden durch die Variablen, die bei der Erzeugung des Moduls übergeben werden, ersetzt.

Aufgrund dieser Architektur ist es sehr einfach, eigene Module zu schreiben. Im Wesentlichen müssen nur eine XML-Vorlage und die darin referenzierten Klassen geschrieben werden. Die Dokumentation von Spring XD erläutert auch detailliert, wie ein Entwickler eigene Sources, Processors oder Sinks implementieren kann. Neben der reinen Entwicklung und Nutzung geht die Dokumentation außerdem auch darauf ein, wie solche Module sinnvollerweise getestet werden können. Dabei ist die Dokumentation auch sonst auf dem gewohnten hohen Standard der Spring-Projekte.

Skalierung

Wenn wirklich sehr viele Daten in einem Spring-XD-System verarbeitet werden sollen, muss ein Cluster von

Servern genutzt werden. Die REST-Schnittstelle wird aber von einem Single-Node-Server angeboten, der für solche Anforderungen sicher nicht ausreichend ist. Die Alternative ist DIRT. Diese durchaus gewöhnungsbedürftige Abkürzung steht für „Distributed Runtime“. Dahinter steckt eine Aufteilung des Servers in einen XD-Admin-Server, der die Streams auf mehrere XD-Container verteilen kann. So stellt DIRT eine hohe Skalierbarkeit sicher. Die beiden Server nutzen die NoSQL-Datenbank Redis zur Kommunikation. Redis bietet sich an, da es als In-Memory-Datenbank sehr schnell ist und durch Redis-Queues auch Messaging unterstützt. Wird also in einem solchen Set-up ein neues Modul erzeugt, so erstellt der XD-Admin-Server eine entsprechende Nachricht in einer Redis-Queue. Diese Nachricht wird dann von einem der XD-Container ausgelesen und das entsprechende Modul auf Basis der übertragenden Konfiguration erzeugt. Dieser Ansatz ist recht einfach und elegant. Gleichzeitig ist er sehr flexibel, da Spring-Konfigurationen genutzt werden, die bekanntermaßen mächtig sind. Und nicht zuletzt können so viele Spring-Entwickler ohne größere Anstrengungen Erweiterungen für Spring XD implementieren.

In Zukunft sollen auch andere Infrastrukturen unterstützt werden. Zu diesen Infrastrukturen zählt beispielsweise das CloudFoundry-Cloud-Angebot von Pivotal. Eine andere Technologie wird Hadoop YARN sein. Diese Technologie stellt die neue Basis für Apache Hadoop dar und dient zur Verwaltung der verteilten Jobs.

Fazit

Spring XD ist im Moment noch in einer sehr frühen Phase – aber die grundlegenden Features sind schon erkennbar. Es macht den Aufbau von Verarbeitungs-Pipelines für Datenströme sehr einfach. Spring XD greift auf die etablierten Ansätze von Spring Integration und Spring Batch zurück. Gleichzeitig werden typische Big-Data-Technologien unterstützt. Diese Kombination aus Bewährtem mit neuen Ansätzen bedeutet zum einen, dass dieses System für viele Entwickler vertraut ist. Sie können es leicht nutzen und auch erweitern. Außerdem hat sich die Basis auch schon in vielen Projekten technologisch bewährt, sodass das Risiko für die Nutzung der Technologie überschaubar ist.

Die angekündigte Nutzung von Reactor ([4], siehe auch Artikel auf S. 73) als Basis von Spring XD lässt erwarten, dass Spring XD auch sehr anspruchsvolle Szenarien bezüglich Performance und Skalierbarkeit unterstützen wird.

Eberhard Wolff arbeitet als Architecture and Technology Manager für die adesso AG in Berlin. Er ist Java-Champion, Autor einiger Fachbücher und regelmäßiger Sprecher auf verschiedenen Konferenzen. Sein Fokus liegt auf Java, Spring und Cloud-Technologien.

@ewolff

Links & Literatur

[1] https://github.com/springsource/spring-xd

[2] http://repo.springsource.org/libs-milestone-local/org/springframework/xd/spring-xd/1.0.0.M1/spring-xd-1.0.0.M1-dist.zip

[3] http://blog.springsource.org/2013/06/12/spring-xd-1-0-milestone-1-released/

[4] http://jaxenter.de/node/164377

Wie Data Science unsere Sicht auf die Welt verändert

Game Changer

Manchmal verändert ein einziges Element Dinge von Grund auf. Der Wechsel vom Straddle zum Flop im Hochsprung war so ein Element. Danach brauchte man sich nicht mehr ernsthaft mit dem Straddle um Medaillen zu bemühen. Data Science ist ein solcher Game Changer, wenn es um unseren Umgang mit Daten und Informationen geht. Es wird uns alle verändern.

von Dr. Matthias Nagel und Thomas Leitner

Die deutsche Sprache hat kein Äquivalent für den Begriff Game Changer. „Die Spielregeln verändernd" ist inhaltlich falsch, denn das Spiel bleibt unverändert. Die Art, wie es gespielt wird, ist hinterher allerdings eine ganz andere. Game Changer ist ein Begriff mit starken Konnotationen von Unvermeidbarkeit und Unwiderstehlichkeit.

Im Hochsprung war der Flop ein solcher Game Changer. Für den Umgang mit Daten und Informationen ist es Data Science. Dabei ist Data Science wie der Flop kein Element, das unabhängig von äußeren Ereignissen sein Potenzial entwickelt. Beim Hochsprung war es die Einführung weicher Hochsprungmatten an Stelle der zuvor verwendeten Sandgruben. Für Data Science sind es zwei Elemente. Zum einen der Siegeszug von Big Data, Cloud Computing und Breitbandinternet, zum anderen die Möglichkeiten der modernen Datenvisualisierung.

Eine weitere Parallele zwischen diesen beiden Game Changern ist, dass es den richtigen Moment brauchte, damit sie ihre volle Wirkung entfalten konnten. So wie Fritz Pingle zehn Jahre vor Dick Fosbury die Technik nutzte, die schlussendlich unter dessen Namen in die Sportgeschichte einging, gab es Datenwissenschaftler, bevor es Data Scientists gab.

Der Hype um Data Science

Die Google-Suche nach „Data Science" findet aktuell knapp zwei Millionen Referenzen im Web und die Harvard Business Review bezeichnete Data Science als „sexiest Job des 21. Jahrhunderts". Erstmals von Peter Naur um 1960 [1] geprägt, wurde der Begriff erst in den letzten zwölf Jahren, dank des Goldrauschs rund um Daten, wirklich populär.

Wer immer auch der Erste war, der die Aussage: „data is the new gold" formulierte, sie oder er hat damit den Zeitgeist perfekt auf den Punkt gebracht [2]. Im Dezember 2011 griff Neelie Kroes, Vizepräsidentin der EU-Kommission und EU-Kommissarin für die Digitale Agenda, sie auf und formulierte: „So wie Öl als schwarzes Gold bezeichnet wird, gewinnen Daten eine neue Relevanz und eine neue Werthaltigkeit im digitalen Zeitalter. ... Um es kurz zu sagen, meine Damen und Herren, meine Botschaft heute ist, dass Daten Gold wert sind."

Das deckt sich mit dem Versprechen der IT-Industrie, das da lautet, mittels Big Data und Daten aus sozialen Netzen und dem Internet der Dinge Umsatzwachstum und Wettbewerbsvorteile zu generieren. Analytics soll insbesondere im Geschäft mit privaten Endkunden maßgeschneiderte, individuelle Angebote wirtschaftlich machen. Die Zielgruppe ist tot, es lebe die Zielperson. Wenn einem bei dieser Wortwahl ein leichter Schauer des Unwohlseins über den Rücken läuft, kommt das nicht von ungefähr. Der Einstieg in Big Data entbindet Unternehmen nicht von einer sorgfältigen Abwägung der Implikationen bezüglich Persönlichkeitsrechten und Datenschutz.

Gleichzeitig greifen diese plakativ kommunizierten Use Cases aber eigentlich zu kurz. Data Science erschließt Unternehmen und Organisationen Daten, die bislang unzugänglich waren, gerade im Bereich der unstrukturierten Daten. Data Science verändert aber auch radikal den Umgang mit allen Daten, indem es neue Fragestellungen unterstützt und Zusammenhänge aufdeckt, die nicht offensichtlich, teilweise sogar bisher unbekannt, aber signifikant sind. Dazu ist es in der Regel hilfreich, bereits von Anbeginn an Visualisierungstools einzusetzen, damit Erkenntnisse im einfachsten Sinne des Wortes erkannt werden.

Sicherlich eines der unterhaltsamsten Beispiele für einen solchen Erkenntnisgewinn lieferte Hans Rosling, Professor für Internationale Gesundheit an den schwedischen Institutionen för Folkhälsovetenskap des Karolinska Institutet als Referent bei einer TED-Konferenz [3]. In dem Beitrag, der als Video inzwischen mehr als 5,2 Millionen Mal heruntergeladen wurde, zeigt er anhand grafisch aufbereiteter und animierter Daten aus Gesundheitsstatistiken unerwartete Korrelationen, wie die unterschiedliche Abhängigkeit von Säuglingssterblichkeit und Lebenserwartung von Bildung beziehungsweise Durchschnittseinkommen, auf. Und in einem engagierten Vortrag räumt er dann gleich auch noch mit der westlichen, durch Vorurteile und Generalisierungen geprägten Sicht auf die so genannte Dritte Welt auf. Am Ende fällt es schwer, Afrika weiterhin als einen homogenen Kontinent zu betrachten und Sinn in einem „One size fits all"-Ansatz der Entwicklungshilfe zu sehen.

Dabei entsprechen die von Rosling für seinen Vortrag genutzten Daten bei Weitem nicht den Kriterien von Big Data. Weder im Umfang noch in der Komplexität der Datenformate sind die verwendeten statistischen Daten ungewöhnlich. Doch schon aus diesen strukturierten Quellen ist er in der Lage, durch gut gewählte Visualisierung signifikante Erkenntnisse abzuleiten. Wie viel mehr Informationen und Erkenntnisse lassen sich dann erst generieren, wenn alle relevanten Daten, strukturierte wie unstrukturierte, zur Analyse zur Verfügung stehen.

Bereitschaft zur Veränderung

An diesem simplen Beispiel lässt sich zudem etwas viel Grundlegenderes aufzeigen, das mit der Arbeit eines Data Scientist einhergeht. Es geht, setzt man Data Science richtig ein, eigentlich nur noch als Nebenprodukt um bessere Daten zur Entscheidungsfindung. Sie können ohne Frage anfallen oder bewusst generiert werden, aber Data Science ist viel mehr als nur Business Intelligence in Echtzeit und auf alle Daten. Es geht weit über Business Intelligence hinaus, denn diese befasst sich lediglich mit der Darstellung von bekanntem Wissen, also Fragen erster Ordnung. Richtig eingesetzt, beschäftigt sich Data Science mit Fragen zweiter Ordnung.

Während Fragen erster Ordnung sich mit dem Gegenstandsbereich beschäftigen und in der Regel mit „Was" anfangen, beispielsweise: „Was muss mein Mitarbeiter wissen, um bessere Entscheidungen treffen zu können?", beschäftigen sich Fragen zweiter Ordnung mit Prozessen, Methoden und Prinzipien und fragen eher: „Wie oder unter welchen Bedingungen können Mitarbeiter bessere Entscheidungen treffen?"

Während die Antwort auf die erste Kategorie von Fragen recht einfach IT-technisch umzusetzen ist, indem Dashboards, Alarme oder der Echtzeitzugriff auf zusätzliche Datenquellen in BI-Lösungen bereitgestellt werden, stellen die Antworten auf Fragen zweiter Ordnung per se die bestehenden Strukturen in Frage.

Heinrich Serwas formuliert es in seinem Buch „Gap Management durch Beirat und Aufsichtsrat" [4] wie folgt:

„Sie sind oft, wegen ihres fundamentalen Charakters, mit Einstellungen verbunden (Identität des Unternehmens, Notwendigkeit einer bestimmten Unternehmensstrategie, Gerechtigkeitsansprüche, Wert und Unwert von Traditionen ...)."

> Data Science ist viel mehr als nur Business Intelligence in Echtzeit und auf alle Daten.

Und damit sind sie nicht einfach technologisch abzuarbeiten. Die Antworten, die Data Science liefert, sind häufig derart, dass sie Entscheidungen der Geschäftsführung oder der Organisationsleitung forcieren. Das kann sowohl interne Strukturen und Prozesse, aber auch grundsätzliches Vorgehen betreffen. Spiegel online brachte kürzlich einen Beitrag unter dem Titel „Marode Gleise: Bahn kämpft mit Mehdorns Billigbauten" [5]. Es ist davon auszugehen, dass sich die Ingenieure der Bahn durchaus mit Fragen der Haltbarkeit der verwendeten Materialien und des idealen Verhältnisses von Kosten zu Lebensdauer beschäftigt haben. Technisch richtige Antworten stehen aber immer auch in einem Konkurrenzverhältnis zu Organisationszielen und der Firmenstrategie. Um wirken zu können, um die Erkenntnisse auch in wirksame Maßnahmen umsetzen zu können, bedarf es bei der Einführung von Data Science als strategisches Tool der Unternehmensplanung daher einer Top-down-Strategie. Dann sind die Mitarbeiter in der Lage, fundierte Kennzahlen zu liefern, die auch schwierige Entscheidungen rechtfertigen. Die Frage nach Kosten-Nutzen-Verhältnis und ROI für unterschiedliche Verfahren der Fahrbahndeckensanierung dürfte Kämmerer wie Verkehrsminister gleichermaßen interessieren und erlaubt belastbare Argumente langfristiger Investitionsplanung.

Data Science eignet sich aber nicht nur für strategische Kennzahlen. Je kleinteiliger und zeitnäher Fragestellungen seitens der Führungsebene formuliert werden, desto leichter ist es, einen kontinuierlichen Prozess der Unternehmensevaluation in Gang zu setzen, quasi ein IT- und datengestütztes Kaizen [6] – einen kontinuierlichen Verbesserungsprozess der kleinen Schritte, wie er insbesondere durch Toyota bekannt wurde. Die Aufgabe des Data Scientist besteht dann darin, ständig neue Sichten auf die Relationen unterschiedlicher Parameter zu liefern. Das kann von einfachen Fragen, wie prozentualem Ausschuss in der Produktion in Abhängigkeit von Tages- oder Jahreszeit, bis hin zu komplexen Relationen von Umsatz nach Produktgruppe in Bezug auf regionale Sozialstatistiken reichen. Das exponentielle Wachstum interner wie extern verfügbarer Daten bietet Unternehmen hier gänzlich neue Möglichkeiten, Flaschenhälse und Schwachpunkte zu erkennen und zu beheben.

Data Scientist – ein Anforderungsprofil

Wie aber sieht das Anforderungsprofil eines Data Scientist aus? Hilary Mason, Chief Data Scientist bei bitly, beschrieb Data Science einmal als Schnittstellenkompe-

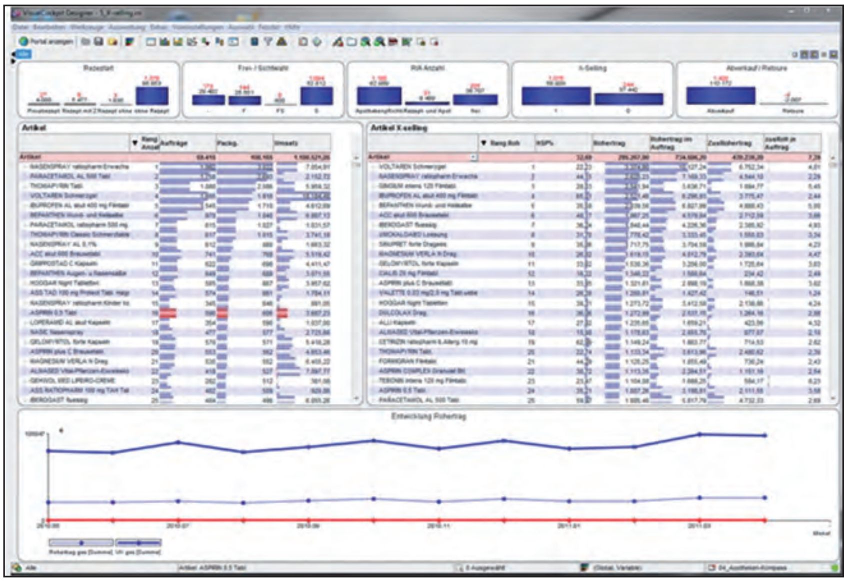


Abb. 1: Cross Selling-Analyse am Beispiel von Arzneimitteln © n³ GmbH 2012

tenz [7]: „Data Science liegt da, wo Computerwissenschaften, Statistik und Mathematik, Ingenieurskunst und ‚Hacking' (hier: der neugierige, kreative Umgang mit Computern), Design und Algorithmen zusammentreffen. Data Science ist ein multidisziplinärer Raum, in dem neue Ideen und Lösungen entstehen."

James Kobielus, IBMs Big-Data-Evangelist, ergänzte diese generische Beschreibung in einem Blogbeitrag um ein eindrucksvolles Curriculum und eine Liste wünschenswerter Kenntnisse und Technologien. Algorithmen und Modellierungstechniken, die ein Data Scientist beherrschen sollte: linear algebra, basic statistics, linear and logistic regression, data mining, predictive modeling, cluster analysis, association rules, market basket analysis, decision trees, time-series analysis, forecasting, machine learning, Bayesian and Monte Carlo Statistics, matrix operations, sampling, text analytics, summarization, classification, primary components analysis, experimental design, unsupervised learning, constrained optimization [8]. Dieses Curriculum enthält die heutigen Anforderungen, zu denen aber ständig neue Methoden und Algorithmen hinzukommen. Zu diesen eher technischen Beschreibungen kommen dann noch zwei Fähigkeiten, die eher in den Bereich der Soft Skills fallen.

Data Scientists brauchen ein ausgeprägtes Talent, komplexe Sachverhalte verständlich zu kommunizieren. Visualisierung, als Tool des Erkenntnisgewinns, eröffnet neue Möglichkeiten, um unbekannte Sachverhalte herzuleiten.

Der Screenshot einer Cross-Selling-Analyse von Arzneimitteln mittels VisualCockpit macht deutlich, wie hilfreich Visualisierungstools beim „Erkennen" von Zusammenhängen sein können. Cross-Selling-Analysen können dabei von relativ einfachen Formen, wie der Frage zur Nachfrage bei den eigenen Produkten, über das On- und Offlinekaufverhalten von Kunden bis hin zu Fragen nach Markensynergien gehen. Statt einfach nur die eigene kleine Kollektion von Accessoires anzubieten, könnte ein Automobilunternehmen herausfinden, welche Mode, Uhren oder Getränkemarken ihre Kunden bevorzugen, das in Werbung und Verkauf integrieren und die Umsätze mit entsprechenden Tools monitoren. Informationen aus dem Social Media, Kundenforen und dem Kundenservice können hierbei wichtige Frühindikatoren für Trends und Probleme liefern (**Abb. 1**).

Aber nicht immer lohnt sich der Aufwand, Ergebnisse aufwändig für das Publikum aufzubereiten. Der Einsatz von Grafiken wandelt sich in Visual Analytics damit von einer eher schmückenden und ergänzenden Darstellung der finalen Ergebnisse zum Arbeits- und Erkenntnisinstrument. Auch bei der Umsetzung von Maßnahmen ist es wichtig, die den Entscheidungen zugrunde liegenden Fakten allen betroffenen Stakeholdern, von Investoren bis Mitarbeitern, verständlich darzulegen. Nichts ist in diesem Stadium so kontraproduktiv, wie sich hinter wissenschaftlicher Terminologie zu verschanzen.

Zudem sollten Data Scientists ein gesundes Maß an unabhängigem Denken und eine gehörige Portion Neugierde mitbringen. Sie sollten wissen, wann sie an die Grenzen ihrer Tools stoßen und nach anderen Methoden und Algorithmen suchen müssen. Die Kenntnis von Problemen und Use Cases aus anderen Fachgebieten erweist sich dabei, in einer Zeit oftmals hochspezialisiertem Fachwissens, als extrem wichtig. Viele Ansätze, die sich in einem Themenbereich erfolgreich bewährt haben, lassen sich auch auf andere Gebiete anwenden.

Gerade bei statistischen Verfahren gilt, dass Methoden, die dafür gemacht sind, um von Stichproben auf die Gesamtheit zu schließen, von Big Data womöglich ad absurdum geführt werden. Umgekehrt gilt es, kritisch auf die Datenqualität, deren Herkunft und die Relevanz der Daten für die Fragestellung zu schauen. Arbeitet man besser mit Rohdaten, weil sich bei den aggregierten Werten Fehler eingeschlichen haben? Sind die Daten in sich und in Verbindung mit Daten anderer Datenquellen plausibel? Wie einfach das geschehen kann, zeigt die

Wert von Daten

- Werden die richtigen Daten gesammelt?
- Stehen sie auch zeitnah zur Verfügung?
- Wie leicht kann darauf zugegriffen werden?
- Wie relevant sind die Daten für die Fragestellung?

kürzlich aufgeflammte Diskussion um Fehler bei der Berechnung von Auswirkungen der Verschuldungsgrenze eines Staats auf das Wirtschaftswachstum [9].

Schließlich besteht das Ziel ja oftmals darin, Dinge zu finden, von denen man noch nicht einmal weiß, dass es sie gibt und dass man sie sucht. Von Jörn Kohlhammer, Head of Competence Center, Fraunhofer Institute for Computer Graphics Research, stammt das Zitat [10]: „But in many data sets, I don't know what I'm looking for.“

First Steps in Data Science

Um mehr als nur Advanced Business Intelligence im eigenen Unternehmen zu nutzen, muss die Bereitschaft zur Veränderung bestehen, die von der Unternehmensleitung initiiert und getragen wird. Es muss ja nicht gleich ein „Alles oder nichts“-Ansatz verfolgt werden. Die Anwendung von Data Science ähnelt in diesem Punkt den Erfahrungen aus anderen IT-Projekten, zum Beispiel der Implementierung von SOA. Hier haben sich überschaubare Projekte, ausgesuchte Quick Wins, als Referenzen für die weitere Umsetzung des Ansatzes bewährt.

Data Science ist nicht nur ein Thema für Großkonzerne, sondern auch klassische Mittelständler können von einer professionellen Analyse aller unternehmensrelevanten Daten profitieren. Daher ist die Frage nach dem personellen Aufwand nicht allgemeingültig zu beantworten. Wenn die Personalstruktur und der zeitliche Rahmen es erlauben, bietet eine In-House-Lösung natürlich Vorteile bei der Flexibilität und dem über die Zeit akkumulierten Wissen. Andererseits können externe Data Scientists ähnliche Vorzüge aufweisen, wie sie im Bereich klassischer Agenturleistungen üblich sind. Es muss nicht alles Data-Science-Wissen im Unternehmen vorgehalten werden. Erfahrungen aus unternehmensfremden Projekten können genutzt werden und die Personalkosten fallen nur dann an, wenn es auch wirklich ein Projekt gibt. Gerade zu Beginn ist es schwierig, alle benötigten Kompetenzen und Ressourcen schon mit eigenen Mitarbeitern abzudecken. Bei Fachkompetenz in der Kombination Statistik, Modellierung und Programmierung ist die Nachfrage am Arbeitsmarkt höher als das Angebot. Die schrittweise Einführung von Data Science erlaubt es dem verantwortlichen Manager aber auch, zunächst mit externen Data Scientists auf Projektebene zu beginnen, dann den Nutzen dieser Projekte zu bewerten, um den Personalbedarf für die Projekte zu ermitteln und um sukzessive interne Ressourcen aufzubauen.

Eine Voraussetzung, die davon unabhängig vorangetrieben werden sollte, ist die Vernetzung aller Applikationen in Unternehmen und der Aufbau einer Struktur für notwendige Repositories und Auswertungsstrukturen für Big Data. Ein Ansatz besteht darin, einen echten Enterprise Service Bus zu implementieren, der nicht nur in der Lage ist, Daten selbst aus „unwilligen“ Applikationen auszulesen, sondern den anfallenden Nachrichtenstrom sowohl zu speichern als auch gleich regelbasiert auszuwerten. Integrationsplattformen wie beispielsweise InterSystems Ensemble bieten eine entsprechende Lösung. Dies hat den Vorteil, dass Auswertungen auf Basis aktueller transaktionaler Daten gemacht werden können und der Umweg über ggf. kosten- und betreuungsintensive Data Warehouses entfällt. Die Daten liegen dann zudem in einer für Visualisierungstools, wie das Visual Cockpit von n³ geeigneten Form vor, um über Visual Analytics Datenanomalien und Flaschenhälse einfacher und schneller auszumachen, als das über eine Suche in schlichten Tabellen oder Spreadsheets möglich ist.

Aufeinander abgestimmte Technologien sind für erfolgreiche Data-Science-Projekte eine wichtige Voraussetzung. In Kombination mit dem erforderlichen mathematisch-statistischen Rüstzeug und einer ausgeprägten Portion Neugierde auf Daten können dann Prozesse optimiert, Schwachstellen aufgespürt und neue Möglichkeiten entdeckt werden. Datenvolumen und Verarbeitungsgeschwindigkeit werden auch in Zukunft stetig zunehmen. Statt ein „weiter wie bisher mit besseren Entscheidungshilfen“ ist der Data Scientist so in der Lage, daraus ein „besser als bisher“ zu machen, einen echten Game Changer.

Links & Literatur

[1] En.wikipedia.org/wiki/Data_science

[2] Europa.eu/rapid/press-release_SPEECH-11-872_en.htm

[3] www.ted.com/talks/hans_rosling_shows_the_best_stats_you_ve_ever_seen.html

[4] Serwas, Heinrich: „Gap Management durch Beirat und Aufsichtsrat“, Books on Demand, 2008, ISBN 978-38370-5967-0

[5] http://www.spiegel.de/wirtschaft/unternehmen/berliner-hauptbahnhof-marode-neubauten-machen-der-bahn-zu-schaffen-a-895453.html

[6] en.wikipedia.org/wiki/Kaizen

[7] Bollhoefer, Klaas; Plaum, Alexander (oreillyblog): „Aus dem Leben eines Datenforschers“, Ein Gastbeitrag in Karriere(n) in der IT-Technologie

[8] www.ibmbigdatahub.com/blog/data-scientist-consider-curriculum

[9] www.spiegel.de/wirtschaft/panne-mit-excel-tabelle-rogoff-und-reinhart-haben-sich-verrechnet-a-894893.html

[10] Dambeck, Holger: „So schön können Daten sein“, Technology Review, 28.01.2013

Dr. Matthias Nagel ist Diplom-Mathematiker mit einem Fokus auf Statistik und Datenanalyse und Mitgründer sowie CEO der n³ data analysis, software development, consulting GmbH. Als externer Data Scientist und Spezialist für Datenvisualisierung unterstützt n³ Unternehmen unter anderem rund um die Themen Datenqualität, Auswertung unstrukturierter Daten und Big Data.

Thomas Leitner ist Regional Managing Director Europe Central & North bei InterSystems und in dieser Position in vielen Projekte involviert, die sich mit neuen Methoden der Datenauswertung beschäftigen. Die Bandbreite reicht dabei von Möglichkeiten, die sich durch nationale Patientenakten in Skandinavien ergeben, bis hin zu neuen Business Cases durch Informationen, die in Smart Grids entstehen.

Über das Ende von Android, wie wir es zu kennen glaubten

Android ist anders

Die Google I/O 2013 kam und ging ohne die Vorstellung neuer, spektakulärer Android-Geräte oder eines neuen Android-Plattformreleases. Ein guter Zeitpunkt daher, um die Mythen und Besonderheiten des Android-Ökosystems Revue passieren zu lassen, zu diskutieren und etwas über die Veränderungen und die Zukunft zu spekulieren.

von Christian Meder

In Zeiten, in denen die Statistiken vermelden, dass die PC-Verkäufe Quartal für Quartal einbrechen, Smartphones das traditionelle Marktsegment von Desktop und Notebook komplett in den Schatten stellen und Tablets auf dem besten Weg sind, in absehbarer Zukunft zahlenmäßig ebenfalls an den „Personal Computern“ vorbeizuziehen, lohnt es sich, sein etabliertes Weltbild in Bezug auf die aktuell meistverkaufte Android-Plattform zu überprüfen [1]. Ist der sympathische und äußerst beliebte kleine grüne Roboter auch offiziell noch keine fünf Jahre alt, so hat er doch schon eine Menge erlebt und schnelle Entwicklungsphasen stoisch (oder blechköpfig?) durchgestanden. Einige, oft heiß diskutierte Themen möchte ich im Folgenden einzeln aufgreifen und aus aktueller Perspektive diskutieren.

User-Interface-Design

„Android UIs sind inkonsistent und hässlich, auf jedem Gerät sieht Android unterschiedlich aus.“

Aktuell ist eine große Konvergenz der UI-Design-Philosophie über die mobilen Plattformgrenzen hinweg zu beobachten: das „Flat-Design“. Die erste Welle der Smartphones, allen voran Apples iPhone, stellte den so genannten *Skeuomorphismus* in den Vordergrund. Dabei wird versucht, in der Nutzeroberfläche das Aussehen natürlicher Dinge realitätsgetreu nachzuahmen: das Notizbuch mit Spiralbindung, Lederumschlag und Papiertextur, Animationen, die das Umblättern von Buchseiten darstellen, ein Taschenrechner mit Tasten, die durch 3-D-Schatteneffekte plastisch werden etc. Das UI arbeitet hierbei stark mit Schattierungen, Farbgradienten, Texturen, Verzierungen, Animationen usw.

Im Flat-Design dagegen wird das Aussehen einfach und funktional gestaltet – ohne 3-D-Effekte, also buchstäblich „flach“. Texturen, Gradienten und Verzierungen werden weggelassen, und es werden einfache, kräftige Farben verwendet. Icons und Anordnungen sind einfach und klar strukturiert, die Funktionalität steht bewusst im Vordergrund (**Abb. 1**).

Seit Android 4.0 Ice Cream Sandwich ist der Einzug des Flat-Designs in das Android UI klar erkennbar [2]. Die mit Android 4.1 Jelly Bean eingeführte kontextbezogene Erweiterung der Suche, Google Now, hat eine kartenbasierte Variante des Flat-Designs gewählt (**Abb. 2**, [3]).

Sukzessiv hat Google auch weitere seiner Apps, wie etwa in diesem Jahr den Google Play Store, auf das kartenbasierte Flat-Design umgestellt (**Abb. 3**).

Die Windows-Phone-Plattform von Microsoft hat das Flat-Design radikal umgesetzt (**Abb. 4**), die iOS 7 Beta von Apple bewegt sich ebenfalls in die Richtung, weg vom Skeuomorphismus hin zu einem flacheren Design (**Abb. 5**).

Gibt es auch diese Konvergenz der grundsätzlichen Designphilosophie, so haben alle drei Plattformen unabhängig voneinander starke Designsprachen entwickelt. Niemand zweifelt heute mehr ernsthaft daran, dass Android User Experience Director Matias Duarte mit seinem Team in den letzten Jahren eine eigene Android-Identität mit dazugehörigen Regeln, Empfehlungen und Patterns entwickelt hat, die Android-Apps als solche erkennbar machen [4].

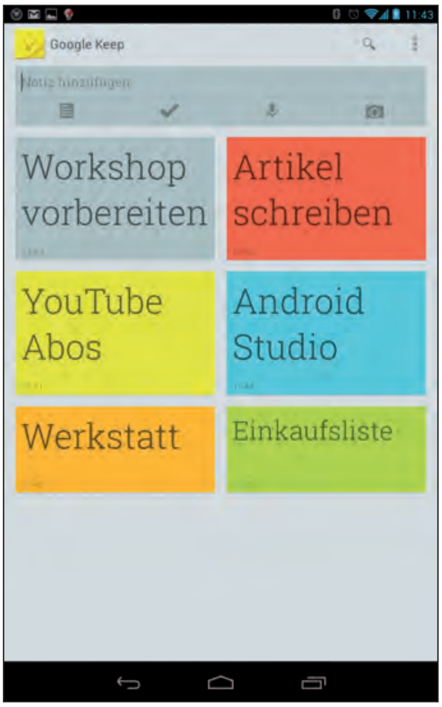


Abb. 1: Flat-Design (Screenshot von Google Keep)

Abb. 2: Google Now mit kartenbasiertem Flat-Design

Abb. 3: Google Play Store im aktuellen Flat-Design

Abb. 4: Windows Phone Homescreens (Quelle: Microsoft)

Die Android-Plattform hat also ihre Identität ausgeprägt, aber zweifellos kommen Android-Geräte in einer Vielzahl unterschiedlich angepasster Oberflächen auf den Markt. Allerdings ist auch hier mehr Konvergenz zwischen den verschiedenen Android-Geräten zu beobachten: Es gibt die Nexus-Reihe von Google [3], [5] mit der puren Android-Version. Die Herstelleranpassungen werden bei einigen Herstellern dezenter. Ganz aktuell sind sowohl das Samsung Galaxy S4 als auch das HTC One zumindest in den USA alternativ in einer reinen Android-Version ohne Herstelleranpassungen zu bekommen [6].

Bei den Interaktionskonzepten wird weiterhin diskutiert und verbessert, etwa in der Diskussion des Verhaltens der Navigation [7] oder bei der Einführung des Patterns des *Navigation Drawer*, bei dem ein Navigationsmenü über eine Wischbewegung von der linken Seite zur Verfügung gestellt wird [8]. Selbst die Systemschrift *Roboto* wird nach aktuellen Erkenntnissen im nächsten Release weiter verfeinert werden [9]. Das kartenbasierte Layout wird vermehrt in Google-Apps umgesetzt, und Google Glass setzt ebenfalls auf ein Kartenlayout für die Anzeige der Daten. Aus meiner Sicht lassen sich einige positive Aspekte an der geschilderten Entwicklung ablesen:

- Das unmodifizierte, reine Android-Plattform-UI besitzt heute eine eigene starke und hochwertige Designsprache inklusive der dazugehörigen Regeln und Vorgaben.
- Es gibt aber noch Freiraum für Experimente, Änderungen und Ergänzungen (siehe Navigation Drawer), eine zu große Starrheit macht größere Änderungsprozesse ungleich schwieriger (siehe die Diskussionen um iOS 7, [10]).
- Es gibt eine immer größere Auswahl an Geräten mit einer unmodifizierten oder wenig angepassten Android-Version (Nexus-Geräte).
- Herstelleranpassungen sind inzwischen oft weniger tiefgreifend und eher auf bestimmte Bereiche beschränkt (Kamera und Bildergalerie, zusätzliche Apps oder zusätzliche Funktionen etc.).

Es ist eine beachtliche Entwicklung, die das ehemals „hässliche Entlein“ seit den ersten Android-Versionen zurückgelegt hat.

Performance

„Android fühlt sich langsam an und ruckelt oft.“

Spätestens seit Android 4.1 Jelly Bean besonderes Augenmerk auf die Optimierung der Grafik-Pipeline gelegt hat (Project Butter, [11]), sind auch die stärksten Kritiker der Android-UI-Performance verstummt. Geräte aller Preiskategorien laufen mit aktuellen Jelly-Bean-Versionen sehr flüssig, vorausgesetzt sie sind nicht komplett untermotorisiert.

Es ist erstaunlich, wie flüssig selbst ein Original-7-Zoll-Samsung-Galaxy-Tab aus dem Jahr 2010 mit einem aktuellen Cyanogenmod 10.1 Nightly Build läuft (allerdings nur Bastlern empfohlen).

Obwohl die Plattformprobleme im Bereich der UI-Performance also als gelöst gelten können, wird es in der nächsten Android-Version noch interessante Optimierungen in diesem Bereich geben: die Reduktion der notwendigen Grafikoperationen durch Umsortierung und Zusammenfassung von Operationen (in einem Beispiel von 88 auf 39 Zeichenoperationen) und die parallele Verarbeitung mehrerer Grafikoperationen (Multi-Threading) sind hier zu nennen [12], [13].

Hersteller- und Gerätevielfalt

Android-Geräte gibt es in vielen unterschiedlichen Variationen von vielen unterschiedlichen Herstellern auf

> „Android-Geräte gibt es wie Sand am Meer, und alle sind langweilig."

dem Markt: von günstigen Einsteigergeräten für Smartphoneneulinge von unbekannteren Herstellern über eine große Zahl von Mittelklassesmartphones bis hin zu hochpreisigen Flaggschiffgeräten von bekannten Premiumherstellern. Daneben gibt es eine Reihe von speziellen Outdoorgeräten, die staubdicht und wasserdicht sind, oder besonders robuste Handys für Baustellen. Es gibt eine Vielzahl von Standardausstattungen im Bereich Bildschirmgrößen und -qualität, Gerätegrößen, Prozessorausstattung, Kameras, Lautsprecherqualität etc. Eine große Anzahl zusätzlicher Möglichkeiten ist ebenfalls gegeben: NFC-Unterstützung, LED-Farbbänder, Micro-HDMI-Anschluss, Infrarotsender, staubdicht, wasserdicht, mit Dockingstation, mit zwei SIM-Karten etc. Das macht die Auswahl eines Android-Smartphones natürlich im ersten Schritt unübersichtlich, erlaubt es dem Nutzer aber, genau das Smartphone zu kaufen, das seinen persönlichen Anforderungen entspricht.

Interessant ist hierbei auch, dass quasi jeder Premiumhersteller aktuell ein hochgelobtes Flaggschiffmodell im Angebot hat: das HTC One, das Samsung Galaxy S4, das Sony Xperia Z und das LG P880 Optimus 4X. Samsung ist derzeit unangefochtener Marktführer, aber offensichtlich scheinen alle Hersteller ihre Nische zu finden.

Nicht vergessen werden soll natürlich auch das Nexus 4, das direkt über den Google Play Store verkauft wird und ein sehr attraktives Preis-Leistungs-Verhältnis aufweist [5].

Apps vs. Plattform

> „Android ist fragmentiert, es gibt keine Herstellerupdates, und nur mit einem neuen Gerät bekomme ich die aktuelle Funktionalität."

Schaut man in die aktuelle Statistik der Android-Plattformversionen [14], zeigt sich, dass die bestehende Fragmentierung selbstverständlich nach wie vor vorhanden ist. Allerdings erkennt man auch, dass sich die Android-Geräte ganz grob in zwei Lager aufteilen lassen: Etwa 60 Prozent der Geräte arbeiten mit Android 4.0 Ice Cream Sandwich oder Android 4.1 und 4.2 Jelly Bean und ca. 35 Prozent der Geräte laufen mit Android 2.3 Gingerbread. Entwickler, die für ihre App eine möglichst große Reichweite erzielen wollen, müssen also auch weiterhin auf Kompatibilität mit Gingerbread oder sogar Froyo achten. Man sollte im Hinterkopf behalten, dass die Android-Plattformstatistik je nach Land sehr unterschiedlich aussehen wird. Vermutlich sind in Deutschland beispielsweise keine 35 Prozent Gingerbread-Geräte mehr in Betrieb.

Die Situation bei den Herstellerupdates hat sich gebessert, bleibt aber problematisch. Android-Geräte bekommen in der Regel noch ein oder zwei Versionsupdates, mehr aber nicht. Es zeigt sich aber bereits seit letztem Jahr, dass Google zunehmend versucht, die Abhängigkeiten von Android-Plattformreleases zu reduzieren. Zum einen kann man immer weitere Apps, die früher Teil des Android-Plattformreleases waren, auch direkt aus dem Play Store nachinstallieren. Jüngstes Beispiel ist hierbei die Google-Keyboard-App. Zum anderen werden zentrale Infrastrukturkomponenten in den Google Play Services zusammengefasst, die sich, wie der Play Store, automatisch updaten. Der App-Entwickler kann hierbei davon ausgehen, dass die Play Services auf jedem Android-Gerät ab Version 2.2 in der jeweils aktuellsten Version zur Verfügung stehen. Neue Möglichkeiten für App-Entwickler, die auf der Google I/O vorgestellt wurden, wie zum Beispiel die neuen *Location Services* oder die *Google Play Game Services*, stehen damit sofort auf allen Android-Geräten ab Android 2.2 zur Verfügung – ohne Abhängigkeiten zu neuen Android-Firmware-Updates [15].

Beide Entwicklungen führen dazu, dass die Android-Plattform nur noch seltener Updates benötigt: Änderungen bei Linux-Kernel, Grafikverarbeitung, hardwarenahen Operationen (Kamera, Bluetooth, Netzwerk, Miracast etc.) und systemweiter Funktionalität (Multiuser) brauchen weiterhin Android-Versionupdates, neuere Versionen von Apps (Keyboard, Browser, Musik etc.) oder Services dann aber nicht mehr. Prinzipiell ist es sogar denkbar, dass ich mir auf diesem Wege die pure Android-Version (Nexus) zum großen Teil per Apps nachinstallieren kann.

Der Ansatz über einen leichtgewichtigen Servicekern, der sich automatisch updatet und auf allen Android-

Abb. 5: Apple iOS 7 Beta (Quelle: Apple)

Geräten ab 2.2 verfügbar ist, in Kombination mit der Zerlegung in einzeln aktualisierbare Apps sieht daher nach einem sehr cleveren Schachzug von Google aus, um die Problematik der Firmware-Updates zu verbessern.

Responsive Design

„Es gibt nur wenige Android-Tablet-Apps."

Zu Zeiten des Web 1.0 kannte jeder die Meldung: „Diese Webseite ist optimiert für eine Auflösung von 800 x 600 Pixeln". Dramatisch wurde es nur, wenn die Webseite in anderen Auflösungen gar nicht funktionierte. Es wurde lange und viel darüber diskutiert, für welche Auflösungen Webseiten optimiert werden und wie flexibel das Layout dann sein sollte (*fluid layout*, [16]). Mit dem Aufkommen von relativ kleinen, aber hochauflösenden Bildschirmen auf Smartphones wurde das Problem verschärft, und es wurde der Begriff des „Responsive Webdesigns" geprägt. Dahinter steht das Konzept, dass Webseiten dynamisch auf Größe, Auflösung und Orientierung des Zielbildschirms reagieren und eine passende Ansicht wählen.

Wie oben beschrieben, gibt es ein großes Spektrum an Android-Gerätegrößen (3 bis 13 Zoll) mit unterschiedlichen Auflösungen. Android-Apps sollten also von Haus aus eine möglichst passende Darstellung für das verwendete Gerät nutzen und nicht in die Versuchung geraten, pixelgenaues Layout für eine fixe Displayauflösung realisieren zu wollen. So wie die Ära des Web 1.0 sollten diese Zeiten auch im Android-Bereich vorüber sein.

Glücklicherweise werden zunehmend Apps für oft sogar drei unterschiedliche Darstellungen angepasst, meist in einer Kombination aus einer einfachen Listendarstellung, einer mit Bildern angereicherten Listendarstellung und einer Magazinansicht (Beispiele sind die Feedly-App oder die Google-Plus-App). Google stellt dem Entwickler auch eine Checkliste zur Überprüfung der wichtigsten Aspekte einer für Tablets optimierten App zur Verfügung [17].

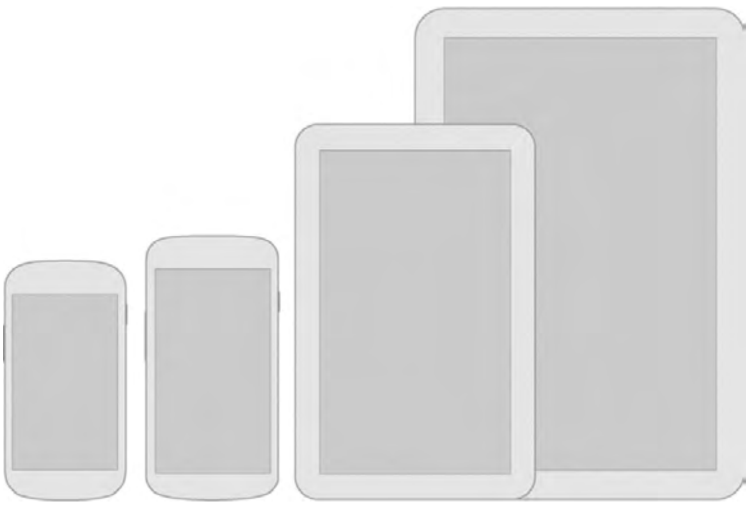


Abb. 6: Unterschiedliche Android-Gerätegrößen

Native Apps vs. Web

„Der Android-Browser ist langsam und schlecht."

Die Webbrowsersituation ist etwas kompliziert bei Android: Es gibt den alten Android-Browser, der Teil der Android-Plattform ist und damit der Standardwebbrowser unter Android 2.x. Dieser Browser erfährt keine Updates mehr. Der aktuelle und moderne Standardbrowser im Android-Umfeld ist Chrome, aber Chrome ist erst ab Android 4.0 Ice Cream Sandwich installierbar, da dieser die erst mit 4.0 vorhandene Hardwarebeschleunigung voraussetzt. Daraus resultiert, dass anspruchsvollere Webapplikationen auf Android 2.3 ein schwieriges Umfeld vorfinden – ohne Aussicht auf einfache Lösungen. Wer im Entscheidungsprozess „native App versus hybride App versus Web" steht, sollte diese Situation im Hinterkopf haben.

Hybride Apps (Webapplikationen als installierbare Apps verpackt) sind für den Endnutzer eigentlich nur ein technologisches Detail, eine technische Toolkit-Entscheidung. Er wird die hybride App mit den anderen installierten nativen Apps vergleichen, da er die App auf dem gleichen Weg installiert hat. In der Regel wird die hybride App dann im direkten Vergleich mit nativen Apps in den Bereichen Nutzerführung, Performance und der Umsetzung der Android-UI-Richtlinien Einschränkungen aufweisen (Diskussion unter [18]). Denn native Apps zeigen sich derzeit noch flüssiger in der Bedienung, passen sich sauber in das Android-UI ein und bieten mehr Möglichkeiten im direkten Zugriff auf die Hardware (Kamera, Mikrofon etc.).

Aber es ist nur noch eine Frage der Zeit, bis eine Reihe von Anwendungsfällen auch genauso performant und flüssig direkt im Webbrowser abgebildet werden kann. Da Chrome für Android ein Teil des Chromium-Open-Source-Projekts ist, profitiert es damit unmittelbar von allen Erweiterungen, die zu Chromium hinzugefügt werden. Unter [19] findet man den aktuellen Status aller Chrome-HTML-Erweiterungen in Bezug auf Standardisierung und Implementierung in den Chrome-Versionen für die verschiedenen Plattformen (WebGL, WebRTC, Web Audio API etc.), darunter auch Android. Ein weiterer positiver Nebeneffekt ist, dass man die Chrome DevTools direkt zum Debugging und der Analyse auf Android verwenden kann [20]. Gerade die Flüssigkeit der Bedienung beim Scrollen von Webseiten, repräsentiert durch die Framerate der Darstellung, lässt sich auf diese Weise sehr gut analysieren [21].

Mehr als Smartphone

Das Nexus 7 stand sicher mit am Anfang des Erfolgs des Taschenbuchformats bei Tablets [3]. Vermutlich wird in nächster Zeit auch ein Nachfolger des Nexus 7 mit aktualisierter Hardware angekündigt werden. Machen Smartphones auch in Zukunft wohl etwa 75 Prozent der verkauften Computer aus, so gehen die meisten Analysten davon aus, dass Tablets, und vor allem Android-Tablets, 2014 Computer und Laptops im Verkauf überholen [1].

„Billig-Android-Smartphones haben den Handymarkt übernommen, aber das iPad ist das einzige ernst zu nehmende Tablet."

Auf der Softwareseite gibt es aber noch eine ganze Reihe von Herausforderungen: Die Einbindung unterschiedlicher Eingabegeräte von Tastatur über Controller bis hin zu Stiften sollte in der Plattform und in Apps besser unterstützt werden. Die Gestensteuerung in Apps kann oft noch für unterschiedliche Gerätegrößen optimiert werden, Gestenkombinationen, die auf einem 5-Zoll-Gerät leicht von der Hand gehen, leiden auf einem 10-Zoll-Gerät durch die weiten Wege auf dem Touchscreen. Die Kopplung an unterschiedliche *wearables* (Smartwatch, Google Glass, Fitnesstracker etc.) sollte einfach möglich sein [12]. Gerade im Bereich des Zugriffs auf Benachrichtigungen für *wearables* scheint sich im nächsten Android-Release aber etwas anzukündigen [22].

Embedded

Durch seine offene Lizenzierung bietet Android viele Möglichkeiten für den Einsatz als Grundlage für eigene Projekte. Ein populäres Beispiel ist die Kindle-Fire-HD-Familie von Amazon. Diese Geräte basieren auf Android, besitzen ein vollkommen modifiziertes UI und verwenden den Amazon App Shop statt des Google Play Stores. Das deutsche Gegenstück zum Kindle, das E-Book-Lesegerät Tolino Shine, basiert ebenfalls auf Android. Die Ouya, eine Spielekonsole für den TV, verwendet ebenfalls Android mit einer modifizierten Oberfläche und einem eigenen Spiele-Shop.

„Android? Ja, klar, Smartphones, oder?"

Abb. 7: E-Book-Lesegerät Tolino Shine

All diese Beispiele werden durch die offenen Lizenzbedingungen der Android-Plattform ermöglicht: Der Linux-Kernel ist unter der GPLv2 lizenziert, und die darauf aufsetzende Android-Plattform ist großteils Apache-2.0-lizenziert [23]. Damit ist es erlaubt, eigene modifizierte Android-Versionen zu bauen, zu verwenden und auch kommerziell zu vertreiben.

Wir stehen aktuell noch am Anfang des Einsatzes von Android für industrielle Zwecke, aber viele Einsatzfelder sind denkbar. Vor allem Geräte, die von einer ausgereiften, hochauflösenden und ansprechenden Bedienung per Touchscreen profitieren können, sind prädestiniert für den Einsatz von Android. Von unmodifizierten Android-Geräten, die mit einer speziell angepassten Android-Version inklusive vorinstallierter Apps bestückt werden, über selbst gebaute Geräte, die mit einem angepassten Android für die Unterstützung von spezieller Hardware installiert werden, ist vieles denkbar: mobile Mess- und Analysegeräte mit Touchscreen, mobile Datenerfassung mit großen Touchscreens in Lager, Handel und Fertigungsanlagen, mobile Dokumentenbibliothek mit hochauflösenden Plänen für Reparaturteams etc.

Die Voraussetzungen für den Android-Einsatz im Embedded-Bereich sind hervorragend. Der

Abb. 8: Ouya-Spielekonsole

Linux-Kernel und speziell der Android-Kernel sind den Entwicklern und Hardwareherstellern im Embedded-Bereich bereits vertraut. Die UI-Entwicklung unterstützt skalierbare Bedienkonzepte per Multitouch für unterschiedliche Bildschirmgrößen. Damit ist es auch möglich, für ganze Familien von Geräten nur ein responsives UI zu entwickeln. Die UI-Entwicklung (App) kann von Android-App-Entwicklern durchgeführt werden und braucht keine Spezialkenntnisse in einer Embedded-UI-Entwicklung.

Community

„Android ist nur ein Pseudo-Open-Source-Projekt."

Die Android-Plattform wird primär von Google weiterentwickelt und über source.android.com in Form des Android Open Source Project (AOSP) der Öffentlichkeit zur Verfügung gestellt. Zwischen den Releases geschieht die Entwicklung hinter verschlossenen Türen und ist nicht öffentlich einsehbar. Allerdings nehmen die Cyanogenmod-Community und diverse kleinere Communitys im Umfeld von xda-developers die AOSP-Releases und bauen daraus eigene modifizierte Versionen für viele unterschiedliche Geräte, oft auch ältere, von den Herstellern nicht weiter unterstützte Geräte [24]. Cyanogenmod hat erst vor Kurzem das auf dem aktuellen Android Jelly Bean 4.2.2 basierende Cyanogenmod 10.1 freigegeben. Die Communitys sind lebhaft und Experimentierfeld für Erweiterungen, beispielsweise einen Inkognitomodus für Apps [25].

Fazit

Google hat die Geschwindigkeit bei den Android-Plattformversionen herausgenommen. Betrachtet man die verschiedenen Aspekte, ist das ein guter Schritt. UI/UX, Design und Performance sind derzeit in ruhigem Fahrwasser. Den Herstellern wird damit Zeit gegeben, auf die aktuelle Android-Version upzugraden bzw. diese bereitzustellen. Die App-Entwickler können endlich die Tablet-optimierte Version ihrer App angehen und die neuen Möglichkeiten von Jelly Bean 4.2 sinnvoll einsetzen. Funktionalitäten werden von Google aus der Plattform in Apps separiert und über den Play Store einzeln bereitgestellt. Services werden automatisch mithilfe von Google Play Services aktualisiert. Chrome für Android hat annähernd zu seinem Desktoppendant aufgeschlossen, und auch die Community arbeitet mit einem stabilen Release der aktuellen Android-Version.

Für das nächste Android-Release wird spekuliert über Verfeinerungen im Design, die Unterstützung von Bluetooth LE (Low Energy), Optimierungen in der Grafik-Pipeline und verbesserte Zugriffsmöglichkeiten auf Benachrichtigungen – alles inkrementelle Verbesserungen. Das gibt alles ein erstaunlich rundes und ruhiges Bild.

Wer nun das Gefühl hat, dass alles zu langweilig wird, kann ja schon mal über eigene Hardwareprojekte mit Android nachdenken oder sich sein eigenes Android für sein Gerät bauen. Die Maker-Szene und die Android-Community freuen sich immer über weitere Enthusiasten.

Christian Meder ist CTO bei der inovex GmbH in Pforzheim. Dort beschäftigt er sich vor allem mit leichtgewichtigen Java- und Open-Source-Technologien sowie skalierbaren Linux-basierten Architekturen. Seit mehr als einer Dekade ist er in der Open-Source-Community aktiv.

Links & Literatur

[1] http://www.independent.co.uk/life-style/gadgets-and-tech/news/state-of-play-android-devices-double-tablet-sales-soar-and-pcs-tumble-8671640.html

[2] Helleberg, Dominik; Meder, Christian: „Android 4.0", in Android360 Magazin 4.2011

[3] Meder, Christian: „Nexus 7: Das Taschenbuch unter den Tablets", in Mobile Technology 4.2012

[4] http://developer.android.com/design/index.html

[5] Meder, Christian: „We are Family: Nexus 4 und Nexus 10 genauer unter der Lupe", in Java Magazin 4.2013

[6] http://blogs.computerworld.com/android/22397/galaxy-s4-htc-one-google-play-editions

[7] https://plus.google.com/102272971619910906878/posts/GqB5SSVDQ26

[8] http://developer.android.com/design/patterns/navigation-drawer.html

[9] http://www.androidpolice.com/2013/06/29/typeface-teardown-roboto-gets-a-facelift-in-android-4-3/

[10] http://www.theverge.com/apple/2013/6/10/4416726/the-design-of-ios-7-simply-confusing

[11] Meder, Christian: „Geleebohnen in Butter", in Java Magazin 12.2012

[12] http://www.anandtech.com/show/6965/the-next-version-of-android-some-of-whats-coming

[13] https://developers.google.com/events/io/sessions/325418001

[14] http://developer.android.com/about/dashboards/index.html

[15] http://developer.android.com/google/index.html

[16] http://www.nngroup.com/articles/computer-screens-getting-bigger/

[17] http://developer.android.com/distribute/googleplay/quality/tablet.html

[18] http://www.androiduipatterns.com/2012/03/multi-platform-frameworks-destroy.html

[19] http://www.chromestatus.com/features

[20] https://developers.google.com/chrome-developer-tools/docs/remote-debugging?hl=de

[21] http://ariya.ofilabs.com/2013/03/frame-rate-hud-on-chrome-for-android.html

[22] http://www.androidpolice.com/2013/07/01/android-4-3s-new-notification-service-read-dismiss-and-press-action-buttons-from-other-apps-imagine-the-wearable-computing-possibilities/

[23] http://source.android.com/source/licenses.html

[24] Meder, Christian: „To infinity ... and beyond. Die wunderbare Welt der Custom ROMs", in Android360 Magazin 2.2012

[25] https://plus.google.com/100275307499530023476/posts/6jzWcRR6hyu

In fünf Minuten zur eigenen App – kann das klappen?

Die 5-Minuten-App

Wir stellen Ihnen die Rezeptur und die Zutaten zur Verfügung, Sie kümmern sich um die Entwicklungsumgebung, das Android-SDK, steuern noch Ihre eigenen Ideen dazu bei, und dann müssen Sie eigentlich nur noch etwas umrühren.

von Stephan Elter W-JAX Speaker

Nun, wahrscheinlich werden wir es nicht ganz schaffen, in fünf Minuten eine App zu programmieren, aber wir wollen doch etwas zügiger vorgehen und Ihnen zumindest einen schnellen Einblick in die App-Entwicklung geben. Und damit Sie nicht alles von vorne machen müssen, kürzen wir etwas ab, stellen Ihnen schon ein paar Komponenten zur Verfügung und sagen Ihnen, was zur Zubereitung unserer „5-Minuten-App" noch benötigt wird. In diesem und den beiden folgenden Artikeln zeigen wir Ihnen, wie Sie eine erste App programmieren und auf Ihrem Smartphone oder Tablet zum Laufen bekommen. In drei doch eher kurzen Artikeln ist das natürlich ein recht ambitioniertes Vorhaben. Wir steigen deshalb schon etwas höher ein und setzen auch einiges an Wissen voraus:

- Sie können natürlich in Java programmieren.
- Sie haben bereits mit Eclipse gearbeitet und können damit auch etwas umgehen.
- Sie wissen, wie Sie mit den Android-SDK-Tools, bzw. dem ADT-Plug-in [1] die notwendigen Einrichtungen vornehmen, um Eclipse fit für Android zu machen.

Und die Sache mit den AVDs, den virtuellen Smartphones, bekommen Sie sicherlich auch ohne große Probleme hin.

Alternativ können Sie übrigens auch das ADT-Bundle von [2] herunterladen und zum Laufen bringen. Das ADT-Bundle ist tatsächlich sogar die etwas einfachere und (von den längeren Downloadzeiten einmal abgesehen) auch die schnellere Möglichkeit, um mit der Entwicklung von Android zu beginnen. Bei dem ADT-Bundle handelt es sich um ein fertig geschnürtes Paket, in dem sich eine vorkonfigurierte Version von Eclipse befindet, in die das Android-SDK mit allen wichtigen Komponenten bereits integriert ist.

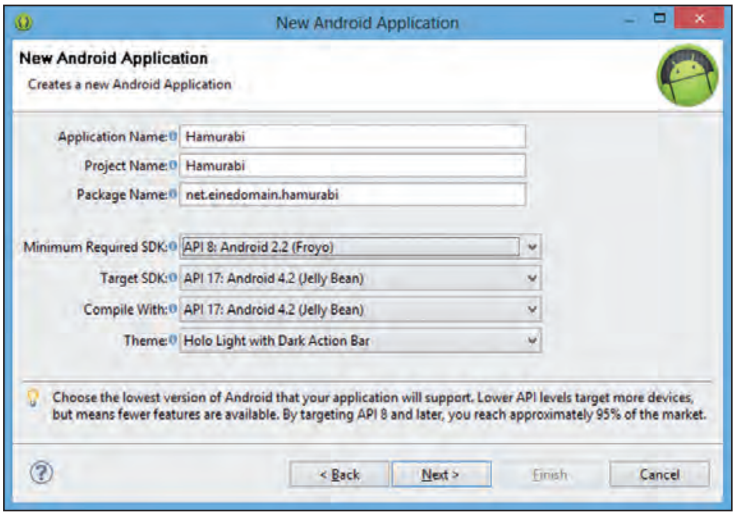


Abb. 1: Neben eher bekannten Festlegungen können wir hier Einstellungen zur Kompatibilität zu bestimmten Android-Versionen vornehmen; immer daran denken, nicht jeder hat das neueste Handy oder das letzte Update

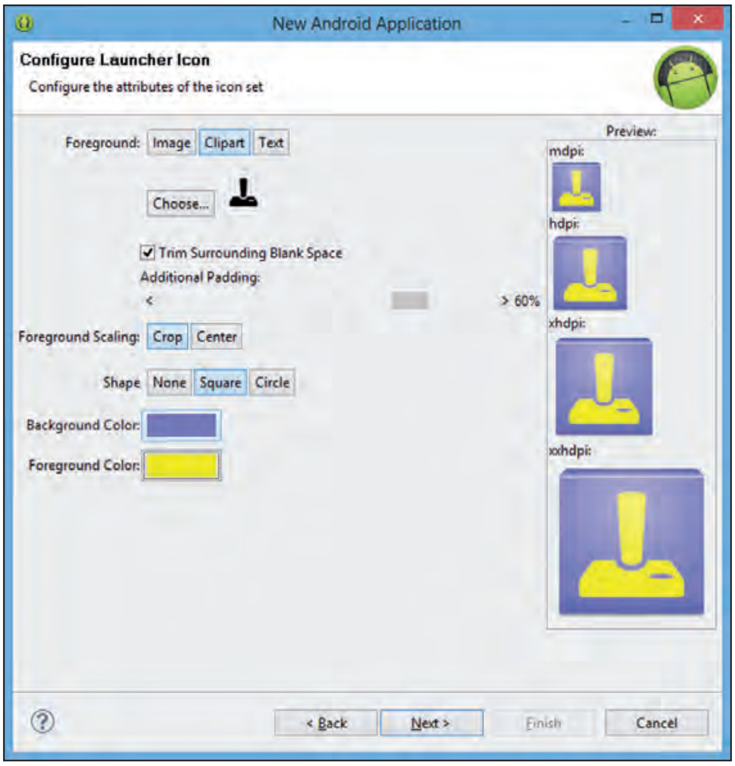


Abb. 2: Schön sieht anders aus, der Assistent lädt aber zum Spielen ein; interessant ist in jedem Fall die Darstellung des Icons in verschiedenen Größen und deren Bezeichnungen

An dieser Stelle ist der Hinweis vielleicht ganz angebracht, dass Google das bestehende ADT-Bundle auf Basis von Eclipse in absehbarer Zeit durch das „Android Studio“ (s. Artikel von Dominik Helleberg im Java Magazin 8.13 auf S. 108 und Artikel zu IntelliJ von Dino Esposito auf S. 65 in dieser Ausgabe) ersetzen wird. Es handelt sich dabei im Kern um die Entwicklungsumgebung IntelliJ IDEA der Firma JetBrains. Sicherlich betritt damit kein Unbekannter das Gebiet der Android-Entwicklung, aber noch ist das Android Studio eine „early access preview“, also noch nicht in einem finalen oder unbedingt stabilen Stadium. Auch stehen noch nicht alle Features zur Verfügung. Da wir davon ausgehen können, dass Eclipse in der Java-Welt nicht ganz unbekannt ist und auch eine gewisse Verbreitung hat, kann man momentan nicht allzu viel falsch machen, wenn man sich nach wie vor auf das recht stabile ADT-Bundle stützt. Und schließlich muss man selbst bei einem späteren Wechsel zu Android Studio auch nicht vollkommen umlernen. Wegen der weiten Verbreitung von Eclipse ist aber auch nicht damit zu rechnen, dass die Android-Entwicklung hier in den kommenden Jahren am Ende sein wird. Wir werden Sie hier auf dem Laufenden halten.

Was bereiten wir in den drei Artikeln zu?

Wir werden nicht sofort programmieren. Nicht, dass sich die Android-Entwicklung schon so weit von der schnöden Programmierung entfernt hätte, nein – es gibt erst einmal viel anderes zu verstehen und einzurichten. Um uns aber nicht in abstrakten Vorhaben zu verlieren, wollen wir Sie im Laufe der drei Artikel dahin bringen, dass Sie eine einfache Version des alten Klassikers Hammurabi als App erstellen. Sie kennen Hammurabi? Hammurabi war ein Herrscher im alten Babylon, bekannt für seine Gesetze, die passenderweise „Code of Hammurabi“ genannt werden. Nach diesem Herrscher wurde auch eine der ersten, sehr einfachen Wirtschaftssimulationen benannt, in der Sie über eine festgelegte Anzahl von Runden (oder Jahren) mit der Eingabe einfacher Parameter über Wohl und Wehe eines Reichs bestimmen. Sie legen dazu in jedem Jahr fest, wie viel Korn Sie aussäen, an Ihr Volk verteilen und ob Sie Ackerland kaufen oder verkaufen. So einfach diese Simulation auch war, so sehr hat sie fast alle darauffolgenden, wesentlich komplexeren Wirtschaftssimulationen beeinflusst. Wegen des recht einfachen Codes ist Hammurabi auch heute noch ein durchaus interessantes Programm, um damit ein einfaches Spiel, in unserem Fall eine erste App zu programmieren. Die für das Spiel verbreitete Schreibweise „Hamurabi“, mit nur einem „m“, ist übrigens nur der alten Begrenzung auf acht Zeichen für Dateinamen unter DOS geschuldet.

Damit der Programmieraufwand (und damit auch der Umfang unserer Artikelreihe) im gesteckten engen Rahmen bleibt, stellen wir Ihnen rechtzeitig zum zweiten Artikel eine etwas einfachere Version dieses Klassikers zum Download zur Verfügung. Diese Klasse wurde nach dem Vorbild des Originals erstellt, für unseren Zweck zum Lernen aber noch ein klein wenig angepasst und sogar noch weiter vereinfacht. Sie können die Klasse direkt in Ihr Projekt einbinden, eben wie eine Art Fünf-Minuten-Terrine, zu der Sie nur etwas heißes Wasser bzw. in unserem Fall ein paar Android-Activities dazugeben müssen. Einfach noch einmal umrühren, und die App ist fertig. Da in Java bis zum heutigen Tage das Schlüsselwort „GOTO“ zwar reserviert, aber noch immer ohne Funktion ist, handelt es sich natürlich nur um eine ungefähre, aber nicht so elegante Umsetzung, die im Original mit einem guten Dutzend GOTOs arbeitet. Sobald mit einer kommenden Version von Java „GOTO“ endlich wie das BASIC-Vorbild arbeitet, werden wir natürlich eine besser angepasste Version der Klasse nachliefern.

Steigen wir direkt ein: Bringen wir das Wasser zum Kochen

Was machen wir, um das Wasser zum Kochen zu bringen? Wir werden in Eclipse ein neues Projekt anlegen, ein eigenes Icon für unsere App erstellen und mit einer so genannten Activity einen mehr oder wahrscheinlich eher weniger imposanten Startbildschirm erstellen.

Ein neues Projekt in Eclipse anzulegen ist wenig aufregend. Hier kocht noch nicht besonders viel. Wenn alles richtig eingerichtet ist, sollten Sie mit STRG+N in einem neuen Bereich ANDROID einen Assistenten bzw. Wizard für ein ANDROID APPLICATION PROJECT auswählen können (**Abb. 1**).

Der *Application Name*, *Project Name* und der *Package Name* sollten verständlich sein, wir legen damit die Namen für unsere App und unser Projekt fest. Auch die restlichen Einstellungen im ersten Fenster übernehmen wir unverändert, ohne uns hier größere Gedanken (oder Sorgen) zu machen. Die hier möglichen Einstellungen sind auch davon abhängig, welche API-Versionen auf dem Entwicklungssystem installiert wurden. Standardmäßig ist zumindest die jeweils aktuellste Version dem SDK beigefügt, was für einen ersten Anfang ausreicht. Wir gehen weiter und lassen im nächsten Fenster angewählt, dass ein Icon und besonders eine Activity erstellt werden.

Bei dem Icon können wir uns gestalterisch austoben, spielen macht eben doch Spaß. Tatsächlich interessant ist aber die Darstellung des Icons in verschiedenen Größen auf der rechten Seite. In diesen unterschiedlichen Größen werden Grafiken später tatsächlich in der Projektstruktur in jeweils eigenen Ordnern abgelegt und das jeweilige System, Smartphone oder Tablet kann sich später nach einer Installation die passende Größe selbst heraussuchen. Dies bitte als wichtigen Punkt schon einmal im Hinterkopf behalten, denn an diesem einfachen Beispiel des Icons können wir sehen, wie Inhalte von Apps dynamisch vorgehalten werden (**Abb. 2**).

Auch im folgenden, vorerst letzten Fenster verändern wir nichts, sondern werden uns eine „Blank Activity“ erzeugen lassen, die immerhin sogar schon ein „Hello world“ enthalten wird.

Activity? Was ist denn jetzt eine Activity?

Die Bezeichnung Activity ist zumindest bei der ersten Begegnung weder selbsterklärend noch schön. Dabei ist es aber ganz einfach: Gerade auf einem Smartphone mit begrenzten Ressourcen ist das Wort „dynamisch“ nicht

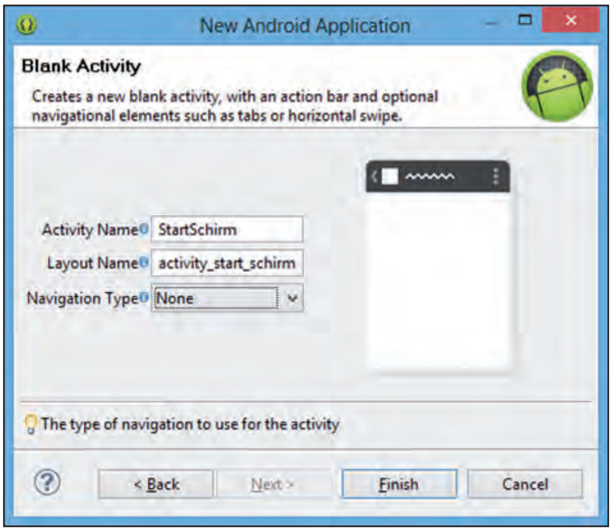


Abb. 3: Der Name der Activity; der „Activity Name“ wird als Name unserer Java-Klasse übernommen, und der „Layout Name“ wird namensgebend für die XML-Datei, die das Layout unserer Activity enthält, sinnvolle Namen sind natürlich zu bevorzugen

nur eines von vielen Buzzwords, sondern tatsächlich ein wichtiges Konzept. Egal, wie viel Speicher zur Verfügung steht oder wie viele Kerne der Prozessor hat, es kann doch immer mal wieder eng werden. Würden eine oder mehrere umfangreiche Apps immer vollständig im Speicher gehalten werden müssen, würde man schnell Probleme mit den Ressourcen bekommen. Eben deshalb gibt es Activities als kleinere, relativ autarke „Organisationsformen“ innerhalb von Apps. Eine Activity ist, vereinfacht gesagt, der kleinste mögliche (und in gewissem Sinne autarke) Teil, der eine eigene Oberfläche und Logik besitzt. Das Rezept dafür ist recht einfach:

- Eine Oberfläche in Form einer XML
- Eine Logik in Form einer zugehörigen, fast normalen Java-Klasse
- Inhalte und Wertangaben, die auch in Form verschiedener XML vorliegen

Fertig ist unsere Activity als kleinste mögliche Form eines Programms. Glücklicherweise unterstützen uns bei der Erstellung die Android-SDK-Tools bzw. unsere entsprechend eingerichtete Entwicklungsumgebung, sodass das Ganze einfacher wird, als man es vermuten könnte. Eine einfache App kann aus mehreren oder natürlich auch aus einer einzigen Activity bestehen, und es ist auch jedem selbst überlassen, wie viel Funktionalität er in eine Activity quetscht und wann er dabei wenigstens ein schlechtes Gewissen bekommt.

Und was bringt das jetzt für den ganzen Aufwand?

Activities und der damit verbundene Aufwand sind kein Selbstzweck. Wird innerhalb einer App von einer zur nächsten Activity gewechselt, beispielsweise von einem Startschirm zum eigentlichen Spiel oder am Ende des Spiels zu einer Ergebnisdarstellung mit abschließender Siegesfanfare, dann wird die vorherige Activity inaktiv und geht in einen anderen Zustand über. Sie wird – vereinfacht gesagt – zu Freiwild und kann von dem System jetzt jederzeit terminiert werden, falls die Ressourcen knapp werden. Dabei ist aber nicht alles verloren – wichtige Informationen über den Zustand der Activity können recht einfach aufgehoben werden. Diese Änderungen des Zustands kann man sehr einfach innerhalb einer Activity abfragen und auch gezielt darauf reagieren – das aber nur als Hintergrundinformation, auf die wir vorerst nicht tiefer eingehen. Damit sollte es auch etwas verständlicher werden, warum man zwar unendlich viele Funktionalitäten in eine einzige Activity einprogrammieren könnte, dies aber eben doch nicht tun soll. Übrigens: Herzlichen Glückwunsch, Sie haben gerade einen der wichtigsten Punkte bei der Android-Entwicklung verstanden.

Wir gehen aber weiter, lassen im Fenster *Create Activity* die bestehende Auswahl Blank Activity bestehen und geben im nächsten Fenster unserer ersten Activity einen aussagekräftigen Namen (**Abb. 3**).

Nach einer kurzen Wartezeit haben wir ein neues Projekt und damit bereits unsere erste sehr einfache App, die damit vorerst aus einer einzigen Activity besteht. Sehen wir uns einmal an, wie sich das in Eclipse darstellt (**Abb. 4**). Auf der linken Seite finden wir den Package Explorer mit unserem Projekt. Nicht alles davon ist interessant, unter *src* finden wir unsere Klasse *Startschirm.java*, die momentan nicht mehr als ein Alibi ohne echte Funktion ist, unter *res* schlummern die Ressourcen, die zu unserem Projekt gehören. Wichtig ist hier der Unterordner *layout*, denn hier findet sich als *activity_start_schirm.xml* die Oberfläche unserer Activity. Wer aufgepasst hat, weiß auch, warum die Dateien so heißen – oder gegebenenfalls anders, wenn man sich nicht sklavisch an die Screenshots in diesem Artikel gehalten hat.

Vormerken sollte man sich in der direkten Nachbarschaft auch schon einmal den Unterordner *values* und dort besonders die Datei *strings.xml*. Die Vermutung, dass dort Strings bzw. Texte abgelegt werden, ist tatsächlich richtig. Und genauso vormerken sollte man sich auch schon einmal die Datei *AndroidManifest.xml* im Stamm des Projekts. Bei dieser Datei handelt es sich quasi um den Ausweis, die Papiere unserer App. Diese XML enthält Informationen wie den Namen der App, die Versionsnummer, eine Angabe der zugehörigen Activities und sogar die Information, welche denn den Startpunkt darstellt. Auch mögliche Berechtigungen, die später zur Ausführung der App benötigt werden, wie ein Zugriff auf das Internet, müssen hier angegeben werden. Gerade beim Einstieg ist das eine beliebte Fehlerquelle, denn ohne explizit angeforderte Berechtigungen geht gar nichts.

Wer sich den Spaß gegönnt und ein eigenes Icon erstellt hat, findet unter dem Ordner *res* mehrere ähnlich benannte Ordner *drawable-XYZdpi*. Es handelt sich um Ordner, in denen grafische Elemente (in unserem Fall vorerst nur unser Icon) in unterschiedlichen Größen

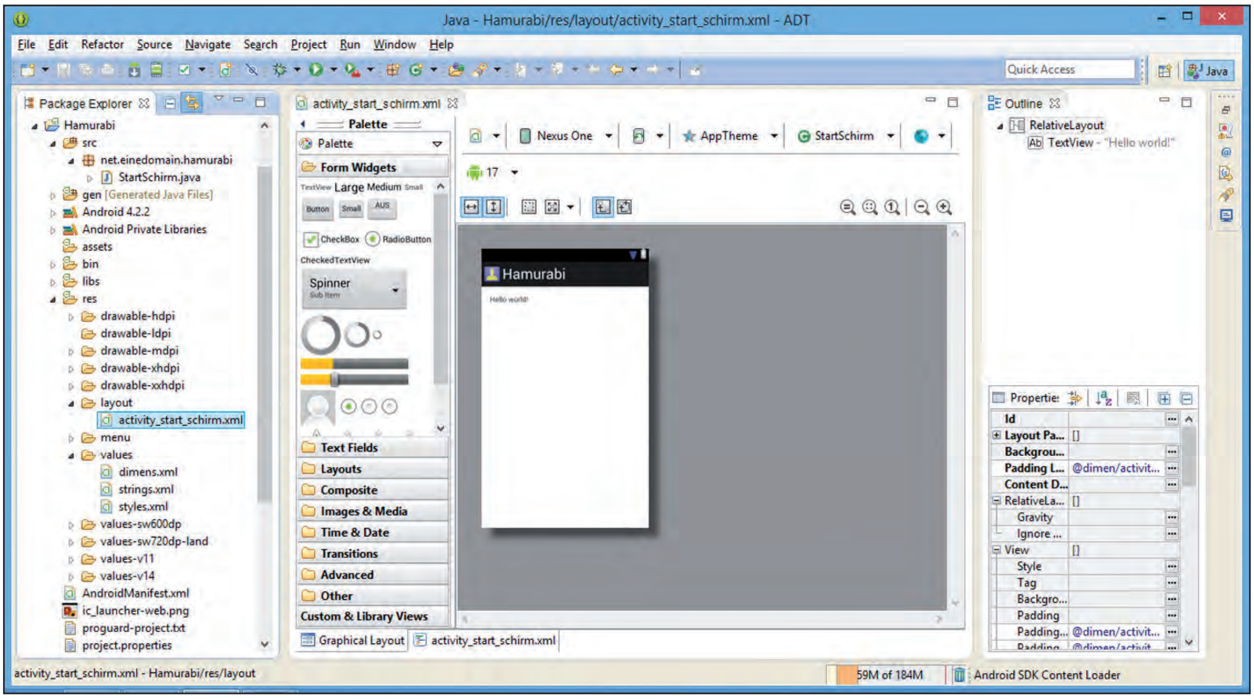


Abb. 4: Das sieht nicht unbedingt alles unbekannt aus; einiges gibt es aber doch noch zu entdecken

abgelegt sind. Unsere *activity_start_schirm.xml* finden wir in visueller Form auch rechts in unserer Ansicht, in der wir unsere neue Oberfläche auch gleich bearbeiten können. Links in der Palette haben wir alle Elemente zur Verfügung, die wir einfach per Drag and Drop in unsere Oberfläche ziehen können. Rechts finden wir unter Structure das Fenster *Outline* und darunter die Properties. Letztere sollten klar sein – hier finden wir die Eigenschaften eines jeweils angewählten Objekts, die wir dort auch bearbeiten können. Das Fenster *Outline* zeigt uns alle Elemente, die Teil unserer Oberfläche sind, geordnet und fein säuberlich dargestellt. Soweit zur sichtbaren Oberfläche.

Von Views, ViewGroups und LayoutManagern

So wie Activities ein ganz zentrales Element (auch für das Verständnis) von Android sind, so wichtig sind Views – *ViewGroups* und *LayoutManager*. Und genauso wie die Bezeichnung „Activity" nicht gerade schön oder wirklich selbsterklärend ist, sind auch diese Bezeichnungen erklärungsbedürftig. Views sind (vereinfacht ausgedrückt) die einzelnen Elemente, die wir aus der Palette in die Vorschau, also in das Layout unserer Activity, ziehen können. ViewGroups sind so etwas wie Sammlungen oder Gruppen von Elementen. Google selbst definiert das so wundervoll: „A ViewGroup is a special view that can contain other views". Das hat schon fast etwas von fernöstlicher Zen-artiger Einfachheit. LayoutManager sind letztendlich eine Sonderform der ViewGroups, und zwar sind sie die Container, die für eine bestimmte Anordnung der Elemente zueinander sorgen. Der einfachste LayoutManager ist das *LinearLayout*. Alle darin enthaltenen Elemente, egal ob ein Button, ein Bild oder ein anderer LayoutManager (das ist erlaubt und ganz wichtig), folgen einfach direkt untereinander. Ein *RelativeLayout* hingegen ist in der Lage, die Elemente fast beliebig in Beziehung zueinander zu positionieren.

Soweit zur notwendigen Theorie, wir gehen nun zur Praxis über: Wir ändern den bereits vorhandenen Text, ziehen ein (eigentlich beliebiges) Bild in unser Layout und fügen noch einen zweiten Text, eine zusätzliche View hinzu. Fertig ist unser (naja) schicker Starbildschirm.

Sag mir, wo die Texte liegen ...

Wir wollen es gar nicht so spannend machen: Unsere bereits vorhandene TextView mit „Hello world" hält den angezeigten Text gar nicht selbst, sondern hat nur einen Verweis auf den Inhalt, der sich an anderer Stelle befindet. In den Eigenschaften dieser TextView finden wir unter dem Punkt Text einen Verweis *@string/hello_world*. Natürlich könnten wir hier einfach unseren Text direkt hineinschreiben, das ist eine kurzfristige, einfache Lösung, aber für eine App dauerhaft nicht flexibel genug. Wir nehmen uns so die Möglichkeit, die App international für unterschiedliche Sprachen „fit" zu machen. Texte werden deshalb nach Möglichkeit in XML-Dateien abgelegt. In unserem Fall liegt der Inhalt in der *strings.xml*, im Projekt zu finden unter *res/values/*. Ein Blick in die Datei enthüllt uns unter anderem folgenden verräterischen Schnipsel:

```
<string name="app_name">Hamurabi</string>
<string name="hello_world">Hello world!</string>
```

Hier befindet sich also unser gesuchter Text, den wir hier auch direkt ändern können. Ich hatte ja geschrie-

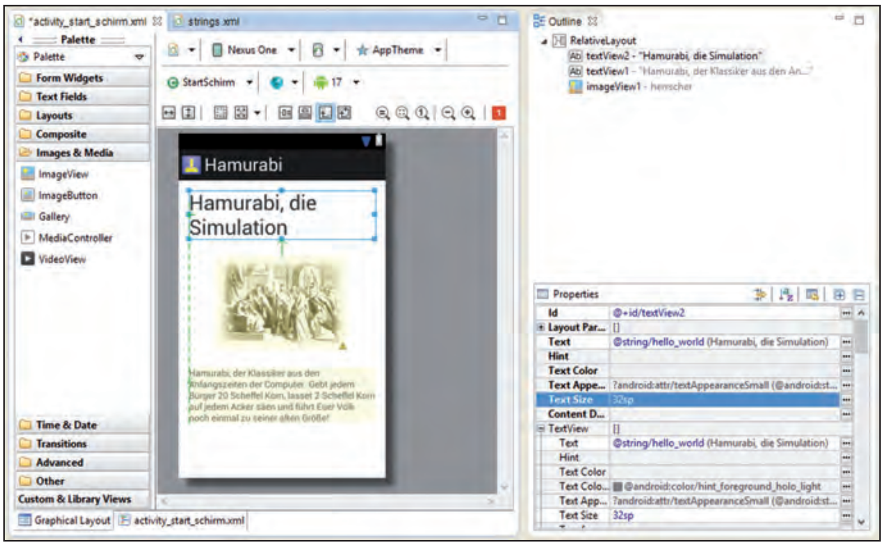


Abb. 5: Beim Schummeln erwischt! Hier wurde die Überschrift unter der Eigenschaft „TextSize“ mit der relativen Größenangabe „32sp“ noch einmal richtig groß gemacht. Und ein Bild wurde auch noch auf die Schnelle eingeschmuggelt

ben, dass in der zentralen Datei *AndroidManifest.xml* unter anderem der Name der App festgelegt wird, aber auch dort befindet sich in Wirklichkeit nur ein Verweis auf den Text, *android:label="@string/app_name"*, den wir hier finden. Es ist alles ganz einfach, man muss nur einmal wissen, wo sich alles versteckt und welches grundsätzliche Prinzip sich dahinter befindet. Und wenn wir schon einmal hier sind, ändern wir den Text und legen uns sofort einen zweiten Text an, den wir gleich noch brauchen. Das sieht fertig dann so aus:

```
<string name="hello_world">Hamurabi, die Simulation</string>
<string name="start_hinweis">Hamurabi, der Klassiker aus den
          Anfangszeiten der Computer. Gebt jedem Bürger 20 Scheffel Korn,
        lasset 2 Scheffel Korn auf jedem Acker säen und führt Euer Volk noch
                                      einmal zu seiner alten Größe!</string>
...
```

Das Speichern nicht vergessen, und wenn wir jetzt wieder zu unserem Layout unserer Activity wechseln, sehen wir bereits den neuen Inhalt aus dem Element *hello_world*. Jetzt noch ganz schnell eine neue TextView erstellen und dieser unseren neuen Text zuweisen. Dazu ziehen wir links aus der Palette unter Form Widgets (wer würde Textelemente dort nicht vermuten?) einfach mit der Maus eine TextView in das Vorschaufenster oder (was genauso geht) rechts an die gewünschte Position in das Fenster *Outline*. Das kann sinnvoller sein, wenn man Elemente schnell in eine bestimmte Reihenfolge bringen will oder sich schon sehr viele Elemente angesammelt haben. Unserer TextView weisen wir jedenfalls noch unter der Eigenschaft *Text* den passenden Eintrag *@string/start_hinweis* zu. Fertig!

Bleibt nur noch das eingeschmuggelte Bild, aber auch das geht ganz einfach. Wir könnten (wie bei unserem Icon) das Bild in verschiedenen Größen in die entsprechenden Ordner legen, aber wir wollen in der Kürze nur eine Version für alle Displaygrößen haben. Deshalb legen wir in unserem Projekt unter *res* einen neuen, allgemeinen Bildordner *drawable* an und ziehen eine bestehende Grafik per Drag and Drop in diesen Ordner. Leider können wir nicht direkt mit dem Bild arbeiten. Dafür ziehen wir aus der Palette, aus dem Bereich Images & Media, eine ImageView in unsere Activity. Sofort öffnet sich ein neues Fenster, und alle gültigen Grafiken aus unserem Projekt werden uns zur Auswahl angeboten. Den Rest sehen Sie beispielhaft in **Abbildung 5**. Alles doch recht einfach.

Bereits jetzt haben wir schon eine App, die wir uns in einem AVD, einem Android Virtual Device, ansehen könnten. Dazu müssten Sie sich nur noch ein passendes virtuelles Smartphone anlegen. Aus Platzgründen vertrauen wir darauf, dass Sie das allein hinbekommen.

Ein kleiner Hinweis nur noch: Der erste Start eines virtuellen Geräts kann recht lange dauern, und damit Sie die App auch wirklich starten können, sollten Sie im Projekt auf die Java-Datei *StartSchirm.java* wechseln, zu finden unter *src/net.einedomain,hamurabi/StartSchirm.java*.

Das soll es für einen ersten kurzen Teil sein. Der zweite Teil wird natürlich noch viel spannender, lehrreicher und noch bunter. Versprochen! Außerdem verraten wir Ihnen dann, wo Sie die Klasse für Hamurabi finden.

Sicherlich können Sie nicht erwarten, nach drei Artikeln (und besonders nicht nach dem ersten Teil) ein fertiger Android-Entwickler zu sein, aber Sie haben einen Einblick gewonnen, was Sie wie machen können und was Sie erwartet.

Stephan Elter arbeitet als Entwickler für den Bereich Internet bei der NORDSEE-ZEITUNG GmbH in Bremerhaven. Neben PHP und Java sind seit dem Palm III mobile Geräte seine Leidenschaft. Er ist auch Autor beim PHP Magazin. Neben seiner eigenen Webseite schreibt er häufig als Gastautor für andere Blogs.

Links & Literatur

[1] http://developer.android.com/tools/sdk/eclipse-adt.html

[2] http://developer.android.com/sdk

Android-Entwicklung mit IntelliJ IDEA, Grundlage der neuen IDE Android Studio

Android-Entwicklung mit Vergnügen

Wenn es darum geht, Anwendungen für mobile Plattformen zu schreiben, ist es oft nicht einfach, dafür eine komfortable und angenehme Umgebung zu finden. Anfangs haben Anbieter – vornehmlich Google für Android und Apple für iOS – eine spezielle IDE angeboten oder empfohlen. Wie man sich vorstellen kann, sind keine überragenden Ergebnisse zu erwarten, wenn die IDE nicht Kernbestandteil der Bestrebungen ist. Deshalb verwendet man heute immer noch Xcode für die iOS-Entwicklung. In Android sieht die Lage etwas besser aus, da Eclipse schon seit Längerem ein Android-spezifisches Plug-in bereitstellt. Bei Eclipse gehen die Meinungen auseinander. Doch auch wenn Sie fast jede IDE an Ihre gewohnte Arbeitsweise anpassen können, sollten Sie sich nicht einfach mit etwas Vorhandenem abfinden, sondern immer nach besseren Möglichkeiten Ausschau halten. Dieser Artikel setzt sich damit auseinander, was es bedeutet, Android-Anwendungen mithilfe von IntelliJ IDEA als IDE zu schreiben.

von Dino Esposito

Ursprünglich als Java-IDE entwickelt, bietet IntelliJ IDEA heute Plug-ins für einige andere Sprachen, unter anderem eines für die Android-Entwicklung. Vor allem aber ist das Android-Plug-in in der kostenfreien IntelliJ-Community-Edition enthalten. Dies bietet einen ausgezeichneten Ausgangspunkt für alle, die die Erfahrung einer Android-Kodierung ohne zusätzliche Kosten machen möchten.

Schritt 1: Vorbereitungen

Es empfiehlt sich, alles für die Android-Entwicklung einzurichten, bevor Sie IntelliJ IDEA installieren. Für Android heißt das, das Java-SDK – die Standard Edition 1.6 ist gut geeignet – und das Android-SDK zu installieren. Das Set-up des Android-SDK läuft normalerweise in zwei getrennten Schritten ab. Zuerst installieren Sie die Kern-SDK-Tools und Binärdateien. Anschließend laden Sie die Binärdateien für diejenigen Android-Plattformen (2.2., 3.0, 4.0 etc.) herunter, für die Sie kodieren möchten, und installieren sie. Den ersten Schritt erledigen Sie mit dem Set-up-Assistenten, der zweite Schritt verlangt ein Ad-hoc-Tool: den SDK-Manager. Im SDK-Manager wählen Sie die für Sie relevanten Android-Plattformen aus. Das Tool lädt dann alle Binärdateien und den Beispielcode für die ausgewählten Plattformen unmittelbar auf Ihren Computer herunter.

Den SDK-Manager können Sie jederzeit von der Befehlszeile aus ausführen oder besser noch über das Menü TOOLS | ANDROID von IntelliJ. Es sei darauf hingewiesen, dass Sie keinerlei Android-Anwendung erstellen können, wenn Sie nicht wenigstens ein Plattform-SDK konfiguriert haben. Um Ihr Android-Abenteuer zu starten, klicken Sie im Willkommenbildschirm auf CREATE NEW PROJECT und wählen den Projekttyp, den Sie erstellen möchten.

Schritt 2: Ein Android-Modulprojekt erstellen

Die häufigste Wahl ist ein Anwendungsmodul. Diese Option veranlasst IntelliJ IDEA, ein Projekt mit einer bestimmten vordefinierten Struktur einzurichten. Das resultierende Projekt können Sie unverändert kompilieren und erhalten eine noch fast leere, doch voll funktionsfähige Android-Anwendung (**Abb. 1**).

Die in **Abbildung 1** gezeigte Drop-down-Liste PROJECT SDK listet alle Plattformen auf, die Sie über den SDK-Manager ausgewählt haben. Die Schaltfläche NEXT bringt Sie zu einem Dialogfeld, das die Namen der Anwendung und des Pakets anzeigt und in dem Sie das Zielgerät und die Startaktivität festlegen können.

Um eine Android-Anwendung zu testen, brauchen Sie entweder einen Softwareemulator oder ein reales Gerät,

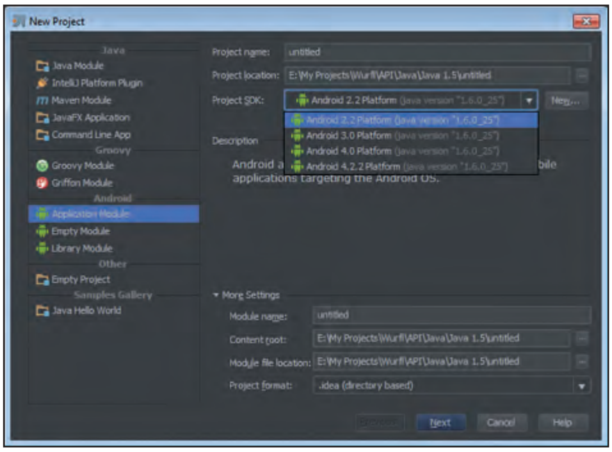


Abb. 1: Eine noch leere Android-Anwendung

das über USB an den PC angeschlossen ist. Es gibt drei grundlegende Run-Konfigurationen:

- **Show Device Chooser Dialog** zeigt an, dass IntelliJ Sie auffordert, den Emulator oder ein per USB angeschlossenes Gerät auszuwählen – jedes Mal, wenn Sie die Anwendung von der IDE aus starten.
- **USB Device** zeigt an, dass IntelliJ automatisch versucht, die Anwendung auf einem kompatiblen Gerät bereitzustellen, das über einen USB-Port ansprechbar ist. Verwenden Sie diese Option, wenn Sie auf einem physischen Gerät testen möchten.
- **Emulator** zeigt an, dass IntelliJ die Anwendung auf dem konfigurierten Emulator bereitstellt. Um auf dem Emulator zu testen, müssen Sie zudem ein AVD (Android Virtual Device) einrichten. Ein AVD ist nichts weiter als eine Konfigurationsdatei, über die Sie Hardware- und Softwareoptionen für den nachzubildenden Emulator definieren. Ein AVD erstellen Sie mit dem AVD-Manager – einem der Android-SDK-Tools, die IntelliJ und andere IDEs ihren Menüs zuordnen.

Das Ergebnis ist ein reines Java-Projekt mit vielen der klassischen Ordner, die in einem Java-Projekt üblich sind, wie zum Beispiel *src*, *libs* und *res*. Android unterstützt jeden Bildschirm, den der Benutzer zu Gesicht bekommt, durch eine Aktivität. Eine Aktivität ist eine reine Java-Klasse, die von einer vom System bereitgestellten Klasse erbt. Als absolutes Minimum ist die Startaktivität auf die Inhaltsansicht zu setzen. Die Inhaltsansicht wird durch eine XML-Datei *main.xml* definiert und befindet sich im Ordner *res/layout*.

Schritt 3: Die Benutzeroberfläche entwerfen

Das Layout jeder Android-Ansicht wird durch eine XML-Datei beschrieben. Die Layoutdatei können Sie über einen grafischen Designer oder einen XML-Texteditor bearbeiten. Listing 1 zeigt das Markup für eine grundlegende und nahezu leere Benutzeroberfläche.

IntelliJ IDEA bietet eine umfangreiche Palette von Widgets, die Sie per Drag and Drop auf die Zeichnungsoberfläche der Ansicht bringen können. Die Eigenschaften der einzelnen Widgets konfigurieren Sie im Fenster PROPERTIES am linken Rand des Bildschirms. Das Fenster COMPONENT TREE hilft Ihnen auch, ein gegebenes Widget nach seiner aktuellen Position in der Hierarchie der Benutzeroberflächenelemente aufzufinden (**Abb. 2**).

Ein effektiver UI-Editor ist für eine Android-IDE im Besonderen und ganz allgemein in jeder IDE für mobile Plattformen ein großes Plus.

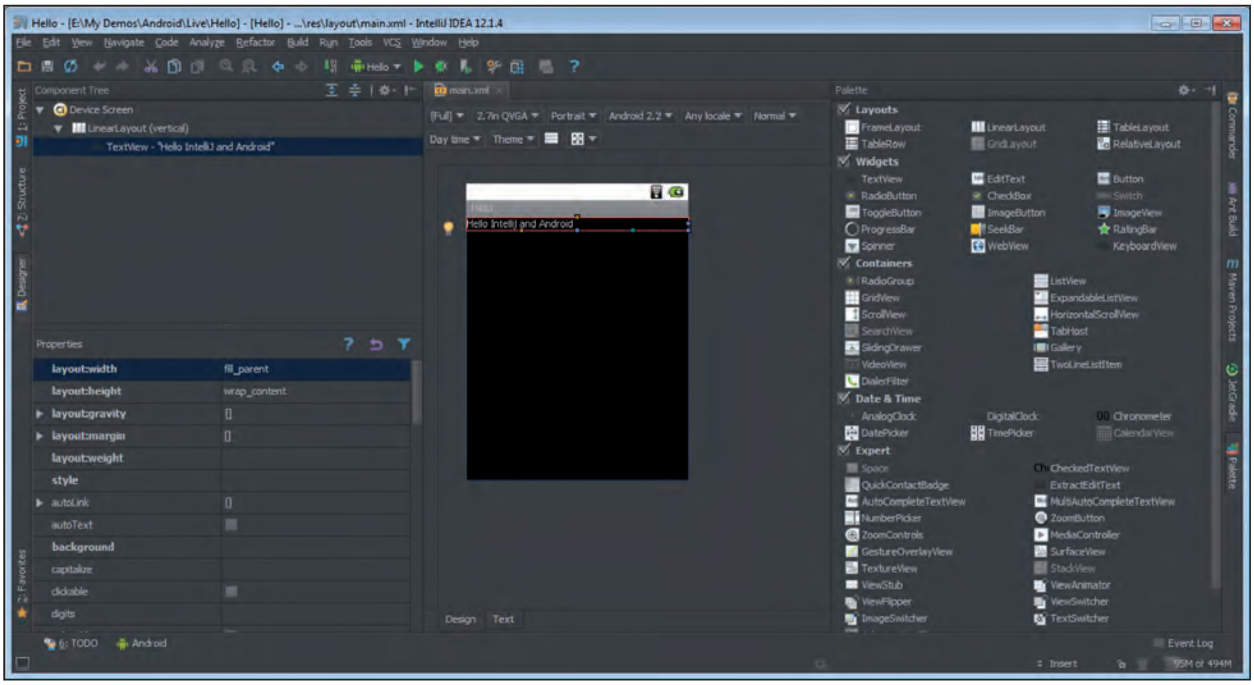


Abb. 2: Auffinden eines Widgets nach seiner aktuellen Position mit dem Fenster Component Tree

Schritt 4: Die App starten

Um die Beispielanwendung zu erstellen und auf dem Zielgerät (egal ob im Emulator oder auf einem realen Gerät) bereitzustellen, wählen Sie eine Run-Konfiguration aus – und los gehts. Der Assistent richtet eine standardmäßige Run-Konfiguration ein, wenn er ein neues Projekt vorbereitet. Über den Menübefehl Edit Configurations des Run-Menüs können Sie maßgeschneiderte Konfigurationen erstellen und sie an ein konkretes virtuelles Android-Gerät (Android Virtual Device, AVD) binden. Um ein Ad-hoc-AVD zu erstellen, verwenden Sie einfach den nativen AVD-Manager, der über das Menü Tools | Android in die IntelliJ IDEA eingebunden ist.

Persönlich bin ich kein großer Freund von Emulatoren mobiler Plattformen. Nach Möglichkeit teste ich meinen Code direkt auf einem realen Gerät und versuche, vollkommen ohne Emulator auszukommen. Der Android-Emulator hat den Ruf, langsam zu sein, insbesondere im Vergleich zum iOS-Simulator. Doch das ist der springende Punkt: In Android handelt es sich um einen Emulator, in iOS um einen Simulator. Wenn Sie eine iOS-App „im Simulator" ausführen, wird der Code in eine Desktopanwendung kompiliert, die nativ auf Ihrem Mac läuft. Deshalb erscheint sie so schnell. In der OS-Entwicklung unterscheidet sich der Code für den Simulator in der Tat von dem Code, der auf dem realen Gerät bereitgestellt wird. Der Simulatorcode wird für x86/64 statt für ARM kompiliert und ist deshalb schneller.

Dagegen ist der Android-Emulator ein virtuelles Gerät, das auf Ihrem Computer für die verschiedensten Plattformen läuft. Er hostet den virtuellen Android-Computer und übernimmt genau den gleichen Binärcode, den Sie auf einem realen Gerät installieren würden. Aufgrund dieser Architektur können Sie im Emulator die Version des Android-SDK auswählen, die

Listing 1

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent">
  <TextView
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:text="Hello IntelliJ and Android" />
</LinearLayout>
```

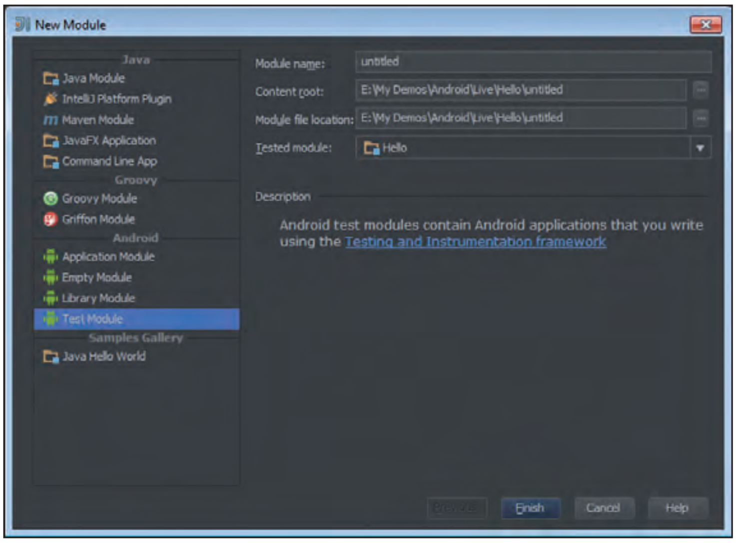


Abb. 3: Android-Testprojekt erstellen

Sie emulieren möchten. Dazu gehören auch Bildschirmabmessungen, Speicher, SD-Karten, Tastatur und sogar Konfigurationen, die realen Geräten nahe kommen. Um diese Parameter festzulegen, richten Sie ein AVD ein.

Schritt 5: Komponententests

In Android basieren Komponententests (Unit Tests) auf JUnit – und allein JUnit genügt, um Features zu testen, die ausschließlich auf Java-Code beruhen. Um Teile der Anwendung zu testen, die Android-spezifisch sind, brauchen Sie ein Bündel von Wrapper-Klassen, die auf JUnit aufsetzen. IntelliJ vereinfacht viele der erforderlichen Aufgaben, um ein Android-Testprojekt einzurichten. Ein Android-Testprojekt erstellen Sie über den Befehl New Module des File-Menüs (**Abb. 3**).

Das Testmodul ist ein eigenständiges Modul mit eigenem Paket, Manifest-Datei und Quellcode. Standardmäßig enthält das von IntelliJ erzeugte Testmodul einen Verweis auf die zu testende Anwendung. Jede Testklasse ist eine Klasse, die von *ActivityInstrumentationTestCase2<T>* abgeleitet ist, wobei *T* den Typ der zu testenden Aktivität angibt. Dadurch ist es ein Leichtes, einen Verweis auf die zu testende Aktivität zu erhalten, wie der folgende Code zeigt:

```
public void testStringForDisplay() throws Exception {
  int count = 1;
  MyActivity activity = getActivity();
  String result = activity.getStringForDisplay(count);
  Assert.assertEquals(result, "once");
}
```

Assertions werden über die Dienste des JUnit-Frameworks implementiert, worauf die Quelldatei geeignet verweisen muss.

Um Tests auszuführen, brauchen Sie eine Build-Konfiguration, die lediglich Tests startet. IntelliJ IDEA erstellt automatisch eine Standardkonfiguration, wenn Sie ein Testmodul einrichten. Dann müssen Sie nur noch die Build-Konfiguration für den Test auswählen und den Test starten.

Schritt 6: Die Anwendung packen

Nach dem Kompilieren wird eine Android-Anwendung als APK-Datei gepackt. Das Paket enthält die Binärdateien und Ressourcen. Während der Entwicklung werden Anwendungen mit einem speziellen Debug-Schlüssel signiert, den die Android-SDK-Build-Tools erzeugen. Der gesamte Vorgang ist für Sie transparent. Allerdings können Sie eine Anwendung nicht in irgendwelchen App Stores veröffentlichen, wenn das Paket nicht mit einem privaten Zertifikat signiert ist.

Um die Anwendung für eine Veröffentlichung zu signieren, müssen Sie zuerst ein Zertifikat beschaffen. Das Zertifikat dient dann dazu, Sie als Autor der Anwendung zu identifizieren. Es ist nicht notwendig, ein Zertifikat von einer öffentlichen Autorität zu kaufen; ein selbst ausgestelltes Zertifikat genügt, um die Anwendung zu veröffentlichen.

Das Java-SDK enthält ein Tool, mit dem sich Zertifikate generieren und verwalten lassen – die Anwendung *keytool*, die im Java-SDK-Verzeichnis verfügbar ist – sowie ein Tool, um Binärdateien zu signieren. Diese Tools sind in einen IntelliJ-IDEA-Assistenten eingehüllt, den Sie über das *Build*-Menü aufrufen. Wenn der Assistent fertiggestellt ist, haben Sie eine APK-Datei, die für die Veröffentlichung kompiliert und digital signiert ist. Das ist genau die Datei, die Sie auf Google Play oder den App Store Ihrer Wahl hochladen. Zudem können Sie diese Datei per E-Mail weitergeben, sodass Benutzer sie direkt auf dafür geeigneten Geräten installieren können.

Abschließende Überlegungen

IntelliJ IDEA versucht, die erweiterten Features einer modernen integrierten Entwicklungsumgebung mit den spezifischen Anforderungen der Android-Entwicklung in Einklang zu bringen. So finden Sie einen dringend benötigten grafischen Editor sowie Codetemplates und Refactoring-Features. Des Weiteren sind Einrichtungen für Komponententests vorhanden, und der Emulator von Google ist nahtlos integriert. Schließlich sind viele der bekannten Android-SDK-Tools in die IDE integriert worden und lassen sich einfach als Erweiterungen der Kernbenutzeroberfläche verwenden. Wenn Sie mit Eclipse arbeiten, sollten Sie IntelliJ IDEA eine Chance geben und dann entscheiden, welche IDE für Sie besser geeignet ist.

Aus dem Englischen von Frank Langenau.

Dino Esposito ist CTO bei e-tennis.net, einer Firma, die sich auf webbasierte und mobile Lösungen für Sportereignisse in ganz Europa spezialisiert hat (http://www.e-tennis.net). Außerdem ist Dino der Autor von „Programming Microsoft ASP.NET MVC3“, Microsoft Press, 2011 und „Architecting Mobile Solutions for the Enterprise“, Microsoft Press, 2012.

Ein Blick auf Googles neues Feature „Cloud Save“

Verteilter Zustand in der Cloud

GOOD TO KNOW

Wer kennt das Problem nicht: Man spielt ein Spiel zuhause auf seinem Tablet und möchte es später auf dem Handy fortsetzen. Aber das ist nicht so einfach. Der aktuelle Spielstand wird in der Regel nicht automatisch zwischen Tablet und Handy synchronisiert. Man muss auf dem Handy also komplett von vorne beginnen. Bisher war es für einen Spielehersteller in Android auch nicht einfach, den Status zwischen verschiedenen Geräten eines Nutzers automatisch abzugleichen. Vor allem für nicht serverbasierte Spiele verzichteten die meisten Hersteller daher komplett auf ein solches Feature. Das aktuelle Update der Google Play Services schafft hier mit „Cloud Save“ Abhilfe. Dieses kleine, aber feine Feature werden wir in dieser Kolumne einmal näher beleuchten.

von Lars Röwekamp und Arne Limburg

W-JAX Speaker

Für die meisten Android-Anhänger war die Google I/O, die jährliche Google-Entwickler-Konferenz, eine Enttäuschung, wurde doch die sehnlichst erwartete neue Version von Android, nämlich Android 5, nicht einmal erwähnt. Und selbst der Termin für die kleine Version des Betriebssystemupdates, Android 4.3, wurde nicht veröffentlicht.

Eines wurde aber auf dieser Google I/O deutlicher denn je: Google hat einen Plan, wie man der Fragmentierung in Android begegnen will, nämlich jener Tatsache, dass es für ältere Android-Geräte keine aktuellen Betriebssystemversionen mehr gibt, weil diese von den Telefonherstellern zur Verfügung gestellt werden müssten. Diese fokussieren sich aber lieber auf ihre aktuellen Geräte.

Der Plan von Google sieht vor, alle Neuerungen nach Möglichkeit vom Betriebssystem zu entkoppeln und als App über den Play Store zu verteilen. Das hat den großen Vorteil, dass sie auch auf älteren Geräten verfügbar sind. Wer braucht Android 4.3, wenn er alle coolen Features auch auf seinem aktuellen Gerät mit z. B. Android 2.3.3 bekommen kann – sofern dieses von der Leistungsfähigkeit her in der Lage ist, das Feature zu verwenden? Einige Neuerungen, die besonders nah an der Hardware entwickelt werden, müssen allerdings weiterhin mit einem neuen Betriebssystem ausgeliefert werden. Ein Beispiel dafür ist „Bluetooth Low Energy“, das voraussichtlich mit Android 4.3 kommt.

Anstatt also ein neues Android-Betriebssystem vorzustellen, präsentierte Google auf der Google I/O eine ganze Menge neuer APIs, die über verschiedene Apps im Play Store verfügbar sind. Einige davon sind Teil der Play Services selbst, so auch das neue „Cloud Save“-API, das wir hier näher vorstellen wollen.

Synchronisation über die Cloud

Neben dem Cloud-Backup [1], das zum Übertragen der Daten von einem alten Device auf ein neues gedacht ist, bietet Google ja schon länger den „Cloud Messaging“ Service an [2], bei dem man vom eigenen Server aus mit den angeschlossenen Endgeräten kommunizieren kann. Dazu braucht man allerdings, wie geschrieben, einen eigenen Server. Für das einfache Abgleichen von Spielständen zwischen zwei Geräten desselben Benutzers ist das etwas viel Overhead, vor allem, wenn das Spiel ansonsten keinen Server benötigt.

Um diese Lücke zu schließen, gibt es in der aktuellen Version der Google Play Services – genauer gesagt: der Google Play Game Services – ein neues API namens „Cloud Save“. Damit ist es möglich, kleine Pakete in der Google Cloud zu speichern und von einem anderen Gerät aus abzurufen. Die Google Cloud ersetzt dabei einen eigenen Server. Dieser Dienst ist kostenlos, bietet aber auch nur ein geringes Datenvolumen von 128 KB pro Slot. Aktuell stellt Google pro App vier Slots zur Verfügung. Sowohl das Volumen pro Slot als auch die Anzahl der Slots könnte sich laut Google allerdings in Zukunft ändern. Die aktuellen Werte können über das API abgefragt werden. Weniger als die 128 KB sollen es aber nicht werden. Für einen einfachen Statusausgleich reicht das auch vollkommen aus. Für echten Datentransfer ist es ungeeignet.

„Cloud Save“ verwenden

Um „Cloud Save“ verwenden zu können, muss man seine App zunächst bei Google registrieren. Das geschieht über die Generierung einer Client-ID [3], die in der Manifest-Datei eingetragen werden muss.

Die Schnittstelle zwischen der App und der Google Cloud stellt der so genannte *AppStateClient* dar. Dieser kann über einen *Builder* erzeugt werden, der im Konstruktor drei Parameter übergeben bekommt (Listing 1). Der erste Parameter ist der *Context*. Hier kann direkt die aktuelle *Activity* übergeben werden. Der zweite Parameter ist eine Implementierung des Interface *GooglePlayServicesClient.ConnectionCallbacks*. Die Methoden dieses Callbacks werden immer dann aufgerufen, wenn eine Verbindung mit der Cloud hergestellt oder die Verbindung unterbrochen wurde. Der dritte Parameter ist eine Implementierung des Interface *GooglePlayServicesClient.OnConnectionFailedListener*, der immer dann informiert wird, wenn eine Verbindung fehlgeschlagen ist. Das Interessante an diesem Listener ist, dass es durchaus Connection-Fehler gibt, die „recoverable“ sind. Näheres dazu, wie sich ein *OnConnectionFailedListener* korrekt implementieren lässt, kann man der API-Dokumentation entnehmen [4].

Listing 1

```
appStateClient = new AppStateClient
    .Builder(activity, connectionCallbacks, connectionFailedListener)
    .setScopes(Scopes.APP_STATE)
    .create();
```

Dem Builder muss dann noch der Scope *APP_STATE* (eine Konstante der Klasse *com.google.android.gms.common.Scopes*) übergeben werden (Listing 1), bevor der Client über die Methode *create()* erzeugt werden kann. Der erzeugte Client enthält dann Methoden, um sich mit der Cloud zu verbinden und die Verbindung später wieder zu trennen.

Daten laden und speichern

Besagter *AppStateClient* kann verwendet werden, um den Status zwischen Cloud und App abzugleichen. Dazu stellt er die Methoden *updateState* und *loadState* zur Verfügung.

Mit der Methode *updateState* kann unter Angabe eines Slots (siehe oben) ein Datenpaket zur Cloud übertragen werden. Es handelt sich hierbei um ein Fire-and-Forget. Man erhält also nicht direkt Feedback, ob die Übertragung erfolgreich war. Sollte die Cloud z. B. aktuell nicht verfügbar sein, werden die Daten zunächst lokal gecachet und später, bei Verfügbarkeit der Cloud, übertragen.

Die Methode *loadState* kann dazu verwendet werden, den App-Status aus der Cloud zu laden. Das Laden erfolgt asynchron. Der Methode wird ein Listener übergeben, der nach dem Laden über Erfolg oder Misserfolg informiert wird. Bei Erfolg wird die Listener-Methode *onStateLoaded* aufgerufen, die in einem Parameter die geladenen Daten erhält. Misserfolg tritt dann ein, wenn der in der Cloud gespeicherte Zustand aktueller ist als der zuletzt lokal geladene Zustand. Festgestellt wird ein solcher Konflikt eigentlich schon beim Aufruf der Methode *updateState*. Dank ihres Fire-and-Forget-Charakters wird hier der Client aber nicht informiert. Vielmehr wird der Konflikt lokal gecachet und beim Aufruf von *loadState* durch den Aufruf der Listener-Methode *onStateConflict* signalisiert. Möchte man bereits beim Übertragen der Daten über einen Konflikt informiert werden, so kann die Methode *updateStateImmediate* verwendet werden, der als Parameter derselbe Listener übergeben werden kann wie der Methode *loadState*.

Umgang mit Konflikten

Ein Konflikt tritt immer dann auf, wenn zwei Clients gleichzeitig versuchen, ihren Status in die Cloud zu übertragen, ohne vorher den Status des jeweils anderen Clients geladen zu haben. Wie mit einem konkreten Konflikt umgegangen werden kann, hängt natürlich stark von den übertragenen Daten ab. Daher kann hier kein generelles Vorgehen vorgeschlagen werden. Die Android-Developer-Dokumentation bietet aber einen guten Überblick über mögliche Konfliktszenarien und deren Lösungsmöglichkeiten [5].

Beim Auftreten eines Konflikts wird, wie bereits geschrieben, die Methode *onStateConflict* des *OnStateLoadedListeners* aufgerufen. Dieser wird als Parameter sowohl der State aus der Cloud als auch der lokale State, der dazu im Konflikt steht, übergeben. Die Aufgabe des Entwicklers ist es nun, die beiden States zu einem State zusammenzuführen. Dieser kann dann über die Methode *resolveState* des *AppStateClients* erneut an die Cloud übertragen werden (Listing 2). Dabei ist es für die Zuordnung des Konflikts zu seiner Lösung wichtig, dass der Methode *resolveState* die Version des Konflikts mitgegeben wird. Diese wurde zuvor in Form eines String-Parameters an die Methode *onStateConflict* übergeben.

Listing 2

```
public void onStateConflict(int slot,
                            String version,
                            byte[] localState,
                            byte[] cloudState) {
  byte[] mergedState = ...;
  appStateClient.resolveState(this, slot, version, mergedState);
}
```

Außerdem ist beim Aufruf der Methode *resolveState* zu beachten, dass natürlich ein neuer Konflikt auftreten kann, falls die Daten in der Cloud parallel erneut geändert wurden. Der Algorithmus zum Auflösen der Konflikte kann dann beliebig oft wiederholt werden.

Fazit

Generell hat Google einen guten Weg mit der Strategie eingeschlagen, immer mehr Funktionalität aus dem Betriebssystemkern zu entfernen und über Apps und Libraries zur Verfügung zu stellen, die über den Play Store verfügbar sind.

Das hier konkret vorgestellte „Cloud Save"-API ist ein Beispiel für eine solche Funktionalität. Konkret bietet dieses die Möglichkeit, den Status einer App über mehrere Endgeräte desselben Benutzers hinweg zu synchronisieren, ohne dass der App-Hersteller dafür einen eigenen Server benötigt.

Auch wenn dieses Feature eigentlich nur für Spiele gedacht ist, lässt es sich hervorragend für jede beliebige Android-App verwenden, um dem Benutzer eine konsistente Nutzung über Gerätegrenzen hinweg zu ermöglichen.

Lars Röwekamp ist Geschäftsführer der open knowledge GmbH und berät seit mehr als zehn Jahren Kunden in internationalen Projekten rund um das Thema Enterprise Computing.

@mobileLarson

Arne Limburg ist Softwarearchitekt bei der open knowledge GmbH in Oldenburg. Er verfügt über langjährige Erfahrung als Entwickler, Architekt und Consultant im Java-Umfeld und ist auch seit der ersten Stunde im Android-Umfeld aktiv.

@ArneLimburg

Links & Literatur

[1] Backup-API: http://developer.android.com/training/cloudsync/backupapi.html

[2] „We're trying to fix Android Fragmentation": http://news.cnet.com/8301-1023_3-57584973-93/google-engineers-were-trying-to-fix-android-fragmentation/

[4] OnConnectionFailedListener: https://developer.android.com/reference/com/google/android/gms/common/GooglePlayServicesClient.OnConnectionFailedListener.html

[5] Resolving Cloud Save Conflicts: http://developer.android.com/training/cloudsave/conflict-res.html

Ein neues asynchrones Framework von SpringSource

Reactor

Node.js hat es vorgemacht: Anwendungen können asynchron entwickelt werden. Durch Frameworks wie Akka oder vert.x sind diese Ansätze auch auf der JVM vertreten. SpringSource betritt nun mit Reactor diesen Markt – da lohnt sich sicher ein erster Blick.

von Eberhard Wolff

Asynchrone Programmierung ist vor allem eine Lösung für den Umgang mit Wartezeiten auf I/O – sei es beim Zugriff auf externe Systeme oder Datenbanken. Herkömmliche Systeme nutzen für jeden Request einen Thread. Wenn die Verarbeitung auf Ressourcen warten muss, blockiert der Thread, und das Betriebssystem sorgt dafür, dass ein anderer Thread an die Reihe kommt. Dieser Ansatz führt dazu, dass für jede Netzwerkverbindung und jeden Client ein Thread genutzt werden muss. Bei sehr vielen offenen Netzwerkverbindungen kann das zu Problemen führen – beispielsweise, wenn bei WebSocket jeder Browser auf jedem Client ständig eine Verbindung mit dem Server hat. Dann müssen schnell tausende von Threads genutzt werden. Das verbraucht viele Ressourcen und bietet keine besonders gute Performance.

Das asynchrone Programmiermodell funktioniert anders: Der Entwickler schreibt Callbacks. Statt den Thread darauf warten zu lassen, dass die Ressourcen zur Verfügung stehen, wird der Callback aufgerufen, wenn die erforderlichen Ressourcen wieder da sind. Ein Thread kann mehrere Callbacks hintereinander ausführen, wenn die jeweils notwendigen Ressourcen vorhanden sind. Dadurch können sehr viele Netzwerkverbindungen mit sehr wenigen Threads unterstützt werden. Dieses Vorgehen stammt aus dem Reactor Pattern [2], das dem Framework auch den Namen gegeben hat.

Vorab eine Warnung: Reactor [1] ist noch in einer sehr frühen Entwicklungsphase – es kann also ohne Weiteres sein, dass sich noch vieles ändert. Aktuell ist noch nicht einmal der erste Meilenstein erreicht. Für einen ersten Blick reicht der aktuelle Stand aber auf jeden Fall aus.

Die Grundlagen

Das Reactor-Framework hat drei wesentliche Elemente:

- *Events* werden verschickt und behandelt.
- *Consumer* erhalten die Events und können sinnvoll auf sie reagieren.
- Der *Selector* entscheidet, welche Events ein Consumer bekommt.

Listing 1 zeigt ein sehr einfaches Beispiel: Zunächst wird ein *Reactor* erzeugt. Dort wird dann ein *Consumer* registriert. Der *Selector* wird durch ein *$* erzeugt. Das ist eine durch jQuery inspirierte abkürzende Schreibweise – es wäre auch möglich, einen *Selector* mit einem Konstruktor zu erzeugen. Der *Selector* wählt die Events aus, die „hello" als Inhalt haben. Am Ende wird dann an den *Reactor* ein Event übergeben, das asynchron an den *Consumer* weitergegeben wird, da es dem *Selector*-Ausdruck entspricht. Die Methode *accept()* wird also zu einem späteren Zeitpunkt aufgerufen.

Selectors gibt es aktuell für reguläre Ausdrücke, Klassen der Objekte und URI-Templates. Für die Performance des Systems ist es natürlich entscheidend, wie schnell Callbacks verarbeitet werden können. Der Reactor kann intern verschiedene *Dispatcher* nutzen, um die Events asynchron an Consumer zu verteilen. Beispielsweise kann eine *BlockingQueue* oder ein *ThreadPool* genutzt werden. Ebenso gibt es eine Implementierung auf Basis des *RingBuffers* aus dem Disruptor-Projekt [5]. Gerade diese Implementierung verspricht eine sehr hohe Performance. Das Disruptor-Projekt kommt aus dem Bereich des High-Speed-Tradings, bei dem niedrige Latenzzeiten und hohe Performance bares Geld wert sind. Der *RingBuffer* erlaubt eine effiziente Speicherung der Events, da er mit einem Array auskommt und so kaum dynamische Speicherallozierung notwendig ist [4]. Das Disruptor-Projekt erreicht seine hohe Performance unter anderem dadurch, dass auf Caches der CPUs und andere Einflussfaktoren geachtet wird. SpringSource selbst gibt an, dass mit schnellen Dispatcher-Implementierungen auch auf wenig leistungsfähiger Hardware bis zu 15 Millionen Events pro Sekunde ausgeführt werden können.

Listing 1

```
Reactor reactor = R.reactor().get();
reactor.on($("hello"), new Consumer<Event<String>>() {
  public void accept(Event<String> t) {
    // Logik
  }
reactor.notify("hello");
```

Listing 2

```
Composable<Integer> c = S.defer("42").get()
  .map(new Function<String, Integer>() {
    public Integer apply(String s) {
      return Integer.decode(s);
}
  }).filter(new Function<Integer, Boolean>() {
    public Boolean apply(Integer i) {
      return (i % 2) == 0;
    }
  }).consume(new Consumer<Integer>() {
    public void accept(Integer event) {
       // TODO: was Sinnvolles tun
   });
   c.await();
```

Listing 3

```
TcpServer<String, String> echoServer =
  new TcpServer.Spec<String,String>(NettyTcpServer.class)
    .codec(StandardCodecs.STRING_CODEC)
    .using(env).listen(4242)
    .consume(
      new Consumer<TcpConnection<String, String>>() {
        public void accept(TcpConnection<String, String> conn) {
          conn.receive(
            new Function<String, String>() {
              public String apply(String s) {
                return "Hallo " + s;
              }
          });
        }
    }).get().start();
```

Listing 4

```
TcpClient<String, String> client =
  new TcpClient.Spec<String, String>(NettyTcpClient.class)
    .using(env)
    .codec(StandardCodecs.STRING_CODEC).connect("localhost", 4242)
.get();
client.open().consume(
  new Consumer<TcpConnection<String, String>>() {
    public void accept(TcpConnection<String, String> conn) {
      conn.in().consume(new Consumer<String>() {
        public void accept(String s) {
          System.out.println(s);
        }
      });
      conn.out().accept("Welt!");
    }
});
```

Bei komplexerer Logik wird der Code schnell schwer lesbar, weil eine Vielzahl von Callbacks implementiert werden muss. Das ist in Java jeweils eine eigene anonyme Inner Class. Das Reactor-Framework bietet die Möglichkeit, Funktionen zu komponieren. Dazu dienen die Klasse *Composable* und das Interface *Function*.

Ein Beispiel zeigt Listing 2: Zunächst wird mit der statischen Methode *defer* der Klasse *S* ein *Stream.Spec* erzeugt. Sie vereinfacht das Erzeugen von *Streams*. In diesem Fall soll *Streams* nur den String „42“ als Eingabe bekommen, sodass sofort auf den *Stream.Spec get* aufgerufen wird. Das Ergebnis ist ein *Stream*. Ihm wird mit *map* eine Funktion zugewiesen, die aus dem *String* ein *Integer* macht. Dieser Wert kann dann von dem *Consumer* verarbeitet werden.

Das Interessante ist nun, dass jeder dieser Schritte – also die Umwandlung wie auch das Verarbeiten – durch eine der schon bekannten Dispatcher-Implementierungen asynchron verarbeitet werden kann. Dadurch müssen weniger Callbacks implementiert werden, und es ist viel leichter, das asynchrone Paradigma zu nutzen.

Das Programmiermodell ist natürlich mit den Lambda-Ausdrücken aus Java 8 oder den Closures in Groovy wesentlich einfacher. Dann können Funktionen direkt hingeschrieben werden, statt sie in einer Klasse zu kapseln. Reactor hat auch schon eine Unterstützung von Groovy, mit der die Implementierung von Reactor-Anwendungen in Groovy ohne größeren Aufwand möglich ist.

Server

Wie schon erwähnt, kann mit einem asynchronen Modell vor allem bei der Entwicklung von Servern sehr effizient gearbeitet werden, da der Umgang mit den Wartezeiten bei I/O wesentlich einfacher ist. Im Moment bietet das Reactor-Framework allerdings nur Möglichkeiten für reine Socket-Kommunikation an. Listing 3 zeigt, wie ein solcher Server mit der TCP-Unterstützung in Reactor implementiert werden kann. Der Server soll Strings über den TCP-Port auslesen und auch zurückgeben. Zunächst wird mit der Klasse *TcpServer.Spec* eine passende Spezifikation erstellt. Sie definiert den Port, auf dem der Server erreichbar ist, und verschiedene andere Optionen.

Dann wird mit *consume()* ein *Consumer* registriert. Das ist dieselbe Art von Callback, wie sie schon in Listing 1 verwendet worden ist. Der hier verwendete Consumer akzeptiert eine TCP-Verbindung. Genau genommen wird also durch den Code aus dem Listing zunächst der Port geöffnet. Erst, wenn er wirklich bereit steht, wird dieser *Consumer* aufgerufen. In der Zwischenzeit kann der Thread für andere Tätigkeiten im System genutzt werden. Wenn die Verbindung zur Verfügung steht, wird an ihr eine *Function* registriert. Sie wird dann jeweils aufgerufen, wenn Daten in der Verbindung eintreffen, und verarbeitet die Daten.

Wie man sieht, wird also jedes Mal, wenn eine Operation auf I/O warten müsste, ein Callback genutzt, der dann aufgerufen wird, wenn die I/O-Operation abgeschlossen ist. Auch das Programmiermodell ist recht elegant: Der Server ist im Kern nur noch eine Funktion – was ja auch den grundlegenden Reactor-Mechanismen entspricht.

Listing 4 zeigt den zu dem Szenario passenden Client. Auch hier wird zunächst mit einer Spezifikation der passende Client erzeugt. Dann wird die Verbindung zum Server eröffnet. Ist die Verbindung offen, wird der Callback vom Typ *Consumer<TcpConnection<String,String>>* aufgerufen. Dieser registriert an der offenen Verbindung einen Callback vom Typ *Consumer<String>*. Dieser Consumer wird jedes Mal aufgerufen, wenn über den Socket eine Antwort vom Server eintrifft – weil er an *in()* registriert wird. Über *out()* wird eine Nachricht an den Server geschickt – und zwar der String "*Welt!*". Der Server schickt "*Hallo Welt!*" zurück – und das wird durch den *Consumer<String>* auf der Konsole ausgegeben.

Wenn man die Listings aufmerksam liest, kann man schon erkennen, dass intern Netty zur Kommunikation genutzt wird. Auf Basis der TCP-Kommunikation kann in Zukunft auch eine HTTP-Kommunikation implementiert werden – auch wenn dazu noch keine Sourcen im Reactor-Projekt vorliegen.

Eine andere konkrete und jetzt schon implementierte Anwendung des Reactor-Frameworks ist Logging. Mit dem *reactor-logback*-Modul kann das Logback-Framework um asynchrones Logging ergänzt werden. Es muss nur der entsprechende Appender konfiguriert werden. Dabei kommt der *RingBuffer* aus dem Disruptor-Projekt zum Einsatz. So kann auch in asynchronen Anwendungen geloggt werden, ohne dass die Verarbeitung durch I/O aufgehalten wird.

Da das Projekt von SpringSource kommt, gibt es natürlich auch eine Unterstützung für das Spring Framework. Damit kann ein Reactor in Spring konfiguriert werden. Außerdem gibt es eine Implementierung des Spring-Interface *TaskExecutor*. Dieses Interface wird immer genutzt, wenn bei Spring Aufgaben verarbeitet werden müssen. Es ist also daher jetzt schon möglich, zur Parallelisierung von Spring-Anwendungen statt Thread Pools Reactor zu nutzen. Und natürlich gibt es eine Unterstützung für die Konfiguration von Reactor in Spring. Schließlich können Spring Beans sich gegenseitig über Reactor Events schicken.

Fazit

Reactor zum jetzigen Zeitpunkt zu bewerten, fällt schwer. Das Framework ist in einem sehr frühen Stadium. Die grundlegenden Ideen bezüglich Streams, Consumer und Events sind nach einigen Iterationen nun stabil. Hingegen sind weitere Features wie die Netzwerkkommunikation noch in einem sehr frühen Stadium. Es ist also noch gar nicht wirklich abschätzbar, was mit dem Framework alles machbar ist.

Außerdem ist Reactor eine Basistechnologie. Überall dort, wo viel I/O und hoher Durchsatz notwendig ist, kann Reactor als Kern eingesetzt werden. Beispielsweise soll in Spring Integration, Spring Batch und Spring XD Reactor genutzt werden. Diese Frameworks werden genutzt, um Massendatenverarbeitung zu implementieren. Dabei fällt viel I/O an, und genau da kann Reactor helfen. Aber auch ein Eventsystem für Grails soll mit Reactor implementiert werden.

Eberhard Wolff arbeitet als Architecture and Technology Manager für die adesso AG in Berlin. Er ist Java-Champion, Autor einiger Fachbücher und regelmäßiger Sprecher auf verschiedenen Konferenzen. Sein Fokus liegt auf Java, Spring und Cloud-Technologien.

@ewolff

Links & Literatur

[1] https://github.com/reactor/reactor

[2] http://en.wikipedia.org/wiki/Reactor_pattern

[3] http://blog.springsource.org/2013/05/13/reactor-a-foundation-for-asynchronous-applications-on-the-jvm/

[4] http://mechanitis.blogspot.de/2011/06/dissecting-disruptor-whats-so-special.html

[5] http://lmax-exchange.github.io/disruptor/

Red5-Flash-Medienserver

NEUE SERIE

Streaminganwendungen leicht gemacht

Video- und Audiostreaming-Anwendungen sind heute sowohl im Consumer- als auch im Businessbereich weit verbreitet. Die Entwicklung solcher Anwendungen ist oftmals komplex und erfordert die Verwendung eines Streamingservers mit entsprechenden Entwicklungsbibliotheken. Red5 bietet durch die Bereitstellung von Basisfunktionalitäten für verteilte Streaminganwendungen die Möglichkeit, sich auf die eigentliche Anwendungsfunktionalität zu konzentrieren, ohne sich mit der notwendigen Low-Level-Kommunikation auseinandersetzen zu müssen. Die Entwicklung von Red5-Anwendungen erfolgt mit bekannten Technologien aus dem JEE-Umfeld (z. B. Apache Tomcat, Spring etc.).

von Sebastian Weber und Cornelius Moucha

Red5 [1] ist ein Open-Source-Medienserver, der kostenlos erhältlich ist und unter der Apache License 2.0 [2] steht. Red5 kann für die Entwicklung verteilter Multimediaanwendungen verwendet werden, z. B. im Bereich Videokonferenzsysteme, Multi-Player-Spiele oder auch Mehrbenutzer-Geschäftsanwendungen. Die erste Version wurde im Jahr 2005 veröffentlicht, im Dezember 2012 dann die Version 1.0. Aktuell wird die Version 1.0.1 zum Download angeboten.

Red5 wurde auf Basis einer Rekonstruktion von Adobes proprietärem „Real Time Messaging Protocol" (RTMP) und des „Action Message Format" (AMF) entwickelt. Intern wird auf bekannte Java-Technologien wie u. a. Apache Tomcat zurückgegriffen.

Was kann Red5?

Red5 unterstützt das Streaming und die Aufzeichnung von Audio- und Videoinhalten, Veröffentlichung von Live-Streams und das Konzept des Flash Remotings via AMF. Mit diesem können Server- und Clientanwendungen auf Basis der Flash-Technologie, wie z. B. Adobe/Apache Flex oder Adobe AIR, über Objekte und Methodenaufrufe miteinander kommunizieren. Audio- und Videoinhalte können in verschiedenen Formaten mit dem Medienserver ausgetauscht werden. Folgende Liste zeigt, welche Protokolle und Services momentan von Red5 unterstützt werden [1]:

- Streaming Video (FLV, F4V, MP4, 3GP)
- Streaming Audio (MP3, F4A, M4A, AAC)
- Recording Client Streams (FLV und AVC+AAC in FLV Container)
- Shared Objects
- Live Stream Publishing
- Remoting
- Protokolle (RTMP, RTMPT, RTMPS, RTMPE)

Red5-Aufbau

Red5 ist ein Medienserver, der intern aus verschiedenen Teilen besteht. So kommen bekannte Open-Source-Anwendungen aus der Java-Welt für verschiedene Anwendungsaspekte zum Einsatz:

- Apache MINA ist ein Framework zur Entwicklung von performanten und skalierbaren Netzwerkanwendungen. An Netzwerkprotokollen werden durch den Einsatz von Java NIO sowohl TCP/IP als auch UDP/IP unterstützt.
- Java NIO (New-I/O) ist eine API-Sammlung für Java, die Funktionen für Low-Level-I/O-Operationen auf der jeweils eingesetzten Plattform anbietet. Es wurde mit J2SE 1.4 eingeführt und sollte das bis dahin existierende I/O-API ablösen.
- Das Spring Framework bietet ein Programmier- und Konfigurationsmodell für Java-basierte Geschäftsanwendungen. Ziel des Frameworks ist die infrastrukturelle Unterstützung im Anwendungsumfeld.

Artikelserie

Teil 1: Einführung in die Entwicklung mit Red5
Teil 2: Audio-/Videostreaming und weitere Red5-Aspekte (z. B. Logging, Monitoring, Sicherheit)

Unabhängig von der eingesetzten Laufzeitumgebung soll so der Entwicklungsfokus auf der Anwendungs- bzw. Geschäftslogik liegen.

- Als Anwendungs-Container bietet Red5 die Möglichkeit, zwischen Jetty und Apache Tomcat zu wählen:
  - Jetty ist ein Servlet-/JSP-Container und Webserver und bietet Integrationsmöglichkeiten für viele verschiedene Java-Technologien wie z. B. OSGi, WebSocket, JMX und JAAS.
  - Apache Tomcat ist ein Web-Container und Webserver, der die Java-Servlet- und Java-Server-Pages-(JSP-)Spezifikationen implementiert. Zum Einsatz kommen intern der Servlet-Container Catalina, die JSP Engine Jasper sowie das Connector-Framework Coyote.

Im Rahmen dieses Artikels wurde in der Beispielanwendung als Anwendungs-Container der in der Standardkonfiguration verwendete Apache Tomcat eingesetzt.

Streamingtechnologien

Zur Umsetzung von Livestreaming existieren bereits verschiedene Verfahren und Protokolle. Von Adobe wurde hierzu das „Real Time Messaging Protocol" (RTMP) entwickelt, dessen Spezifikation 2009 veröffentlicht wurde. Dieses Protokoll erlaubt den Transfer von Audio- und Videodaten bzw. beliebiger Daten über das Internet in Echtzeit zwischen dem Server und ggf. mehreren Clients. Ein weitverbreitetes Streamingverfahren stellt heute die RTMP-Kommunikation in Verbindung mit Flash-basierter Clientsoftware dar. Inzwischen gibt es auch mehrere auf RTMP aufsetzende Protokolle (z. B. RTMPT, RTMPE oder RTMPS), die zusätzliche Eigenschaften wie z. B. verschlüsselten Datentransfer ermöglichen.

Mit dem „Real Time Streaming Protocol" (RTSP) existiert eine mögliche Alternative zu RTMP. Bezüglich des Streamings bieten beide Protokolle ähnliche Funktionalitäten an. RTSP wurde ursprünglich entwickelt, um Steuerungsfunktionen wie *Play*, *Pause* und *Stop* für Medieninhalte anzubieten. Eingesetzt wurde es beispielsweise von RealNetworks im RealPlayer in den 90er Jahren [5].

Um Medieninhalte von einer Clientsoftware über eines dieser Protokolle abzurufen, ist ein entsprechender Server als Gegenstelle notwendig. Für RTMP-basierte Kommunikation in Kombination mit der Flash-Technologie können dies der Flash Media Server (FMS) oder der hier vorgestellte Red5-Server ermöglichen. Im Verlauf dieses Artikels wird dazu eine Beispielanwendung für den Red5-Medienserver erstellt.

Installation des Red5-Servers

Die benötigten Installationspakete sowie Quelldateien können über die Projektseite bei Google Code bezogen werden [1]. Die aktuelle Red5-Version 1.0.1 wird als Installer für Windows sowohl für Java 6 als auch Java 7 angeboten. Zusätzlich wird ein Zip-Archiv bzw. Tarball für Linux und Mac bereitgestellt. Unter Linux wird lediglich eine installierte Java Runtime der Version 1.6 oder 1.7 benötigt. Im Archiv werden Administrationsskripte zum Starten und Stoppen des Red5-Servers bereitgestellt. Je nach Entwicklungsstand sollte man hier den Server mit *red5-debug.sh* (bzw. *red5-debug.bat* in Windows) im Debug-Modus oder mit *red5-highperf.sh* für den Produktivbetrieb starten. In beiden Fällen kann mit *red5-shutdown.sh* eine laufende Red5-Instanz beendet werden. Die Konfiguration des Red5-Servers erfolgt über mehrere Konfigurationsdateien im XML-Format im Verzeichnis *[RED5_HOME]/conf*. Die meisten Dateien referenzieren hierbei die Datei *red5.properties*, über die verschiedene Konfigurationswerte wie z. B. Portzuweisungen oder Puffergrößen gesetzt werden können. Für den Anfang empfiehlt es sich, die Standardwerte beizubehalten. Je nach Situation kann es allerdings notwendig sein, die Ports anzupassen, z. B. wenn ein Port bereits durch eine andere Anwendung blockiert sein sollte.

Vorbereitung für die Entwicklung einer Red5-Anwendung

Für die Entwicklung von Red5-Anwendungen empfiehlt sich die Verwendung der JEE-Variante der Eclipse-IDE [6]. Die Firma Infrared5 stellt ein entsprechendes Eclipse-Plug-in zur Verfügung [7]. Nach der Installation lässt sich über Window | Preferences | Server | Runtime Environments eine Red5-Server-Runtime (Auswahl Infrared5 | Red5 Server Runtime) definieren. Hier muss lediglich der Pfad der Red5-Installation angegeben werden. Anschließend ist analog ein Server über Server | New: Infrared5 | Red5 Server Runtime zu erstellen. Hier ist der Ordner für das anschließende Deployment der Red5-Anwendungen (*<Red5-Installationspfad>/webapps*) sowie der Pfad des Start- und Stoppskripts anzugeben. Andere Angaben können auf den definierten Standardwerten belassen werden. Abschließend können bereits existierende Webanwendungen mit diesem Server verknüpft werden, sofern bereits vorhanden.

Anwendungsbeispiel Meeting-Place „Red5Chat"

Als Beispielanwendung zur Einführung in die Entwicklung von Red5-basierten Clientserveranwendungen wird eine Meeting-Place-Anwendung vorgestellt. Um ein vollständiges Bild von der Red5-Entwicklung zu bekommen, ist neben der serverseitigen Entwicklung (d. h. dem eigentlichen Red5-Bestandteil) eine entsprechende Beschreibung des Clients auf Basis von Adobe Flash notwendig. In diesem Bereich des Artikels wird der Fokus auf die Netzwerkkommunikation mit dem Red5-

Java-basierte Alternativen zu Red5

Red5 kann als Alternative zu kommerziellen Lösungen, wie beispielsweise zum Adobe Flash Media Server (FMS) [3], angesehen werden. Während Red5-Anwendungen in Java umgesetzt werden, nutzt der FMS „Serverside ActionScript". Link [4] bietet einen Überblick über einige weitere Java-basierte Alternativen, die wie Red5 u. a. auf RTMP basieren.

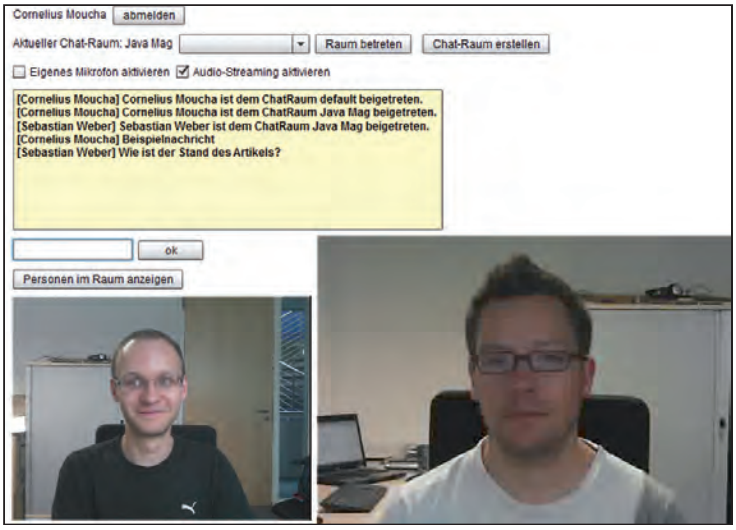


Abb. 1: Beispieldarstellung der Red5-Chatanwendung (Clientsicht)

Server gelegt. Ein Screenshot einer exemplarischen Clientumsetzung ist in **Abbildung 1** dargestellt. Für die Umsetzung der Meeting-Place-Anwendung werden die folgenden Szenarien beschrieben:

- In Chaträumen können Benutzer mit einem oder mehreren anderen Benutzern Textnachrichten austauschen. Benutzer haben die Möglichkeit, neue separate Chaträume zusätzlich zum Standardraum anzulegen bzw. in andere Räume zu wechseln.
- Weiterhin besteht die Möglichkeit der Audiokommunikation zwischen allen Benutzern eines Chatraums.
- Zusätzliche Chaträume können als „offen“ oder „geschlossen“ deklariert werden. Letzteres verlangt das Versenden von Einladungen, um dem Raum beitreten zu dürfen.
- Abschließend können Videochaträume explizit für zwei Benutzer erstellt werden, damit beide Benutzer per Videostreaming miteinander kommunizieren können.

Entwicklung der Red5-Serveranwendung „Red5Chat“

Aufbauend auf der Installation des Red5-Servers und der Konfiguration der Entwicklungswerkzeuge wird nun die Umsetzung einer Red5-Anwendung beschrieben. Zu Beginn wird in Eclipse ein neues Dynamic Web Project erstellt. Im Dialog ist darauf zu achten, dass die oben erstellte Red5 Server Runtime als Target Runtime ausgewählt ist. Der Auswahlbereich Configuration | Modify bietet zusätzlich die Möglichkeit, eine zusätzliche Project Facet (Red5 Application Generation) auszuwählen. In diesem Fall wird bereits ein minimales Beispieltemplate einer Red5-Anwendung vorbereitet. Zusätzlich wird ein zweites Projekt *<Projektname>Client* generiert, das Beispielcode für den Flexclient zur Verfügung stellt. Auf die clientseitige Umsetzung wird im Anschluss an die serverseitige Umsetzung eingegangen.

Nach der Erstellung des Red5-Anwendungsprojekts sind noch einige Änderungen notwendig, da das zur Verfügung stehende Eclipse-Plug-in nicht komplett kompatibel zur aktuellen Red5-Version ist:

1. In der Datei *WebContent/WEB-INF/red5-web.xml* ist in der Definition des Beans *web.scope* die Klasse *org.red5.server.scope.WebScope* anzugeben. Hier wurde die Paketstruktur verändert.
2. Zusätzlich ist im Java Build Path des erstellten Projekts eine zusätzliche Datei aus dem *lib*-Verzeichnis der Red5-Installation anzugeben: *spring-core-3.1.1.RELEASE.jar* (für die Red5-Version 1.0.1). Alternativ kann der Build Path im Template des Eclipse-Plug-ins entsprechend angepasst werden.
3. In der generierten Anwendungsdatei *Application.java* ist die *import*-Direktive auf *org.red5.server.api.scope.IScope* anzupassen.

Anschließend ist das Projekt initial mit dem vorher erstellten Server zu verknüpfen: Durch die Auswahl von Run | Run as | run on server lässt sich die Anwendung auf dem Red5-Server installieren. Spätere Änderungen am Anwendungscode führen zu einem automatischen Redeployment der Anwendung auf dem Server.

Durch Überladen bereits existierender Methoden in der Red5-Basisklasse *ApplicationAdapter* bzw. besser *MultiThreadedApplicationAdapter* lässt sich die eigene Anwendungslogik in der Anwendungsdatei *Application.java* implementieren. In Tabelle 1 werden einige der im Anwendungsbeispiel verwendeten Funktionen exemplarisch benannt. Die konkreten Methodensignaturen können der Red5-Dokumentation bzw. der Basisklasse entnommen werden. Zu beachten ist, dass Red5 mehrere konzeptuelle Ebenen unterscheidet. Primär existiert die Red5-Anwendung in Form eines *(MultiThreaded)ApplicationAdapters*. Zusätzlich gibt es für jede Anwendung einen oder mehrere Räume (so genannte „Scopes“). Es besteht außerdem die Möglichkeit, Räume ineinander zu verschachteln. Das prinzipielle Schema eines Verbin-

Tabelle 1: Basisfunktionen einer Red5-Anwendung [8]

Methode	Grundfunktion
start	Start der Red5-Anwendung bei Serverstart
appConnect/appDisconnect	Client-Connect/-Disconnect zur Anwendung
appJoin/appLeave	Client-Join/-Leave zur Anwendung
roomStart/roomStop	Start/Stopp eines Raums
roomConnect/roomDisconnect	Client-Connect/-Disconnect zu/von einem Raum
roomJoin/roomLeave	Client-Join/-Leave eines Raums

dungs-URLs zu einer Red5-Anwendung sieht folgendermaßen aus:

```
rtmp://<Servername bzw. IP>/<appName>/<scope1>/<scope2>
```

Im Kontext der Beispielanwendung entspricht dies:

```
rtmp://10.0.0.1/Red5Chat/chatRaum1/chatRaum2
```

Des Weiteren unterscheidet Red5 zwischen dem initialen Verbindungsaufbau (*connect*) und anschließendem Beitritt (*join*) jeweils auf Anwendungsebene als auch auf Raum-/Scope-Ebene. Analog gilt diese Differenzierung für den Verbindungsabbau: *disconnect* und *leave*. Bei verschachtelten Räumen sieht die Abarbeitung dementsprechend folgendermaßen aus: *appConnect*, *roomConnect* raum1, *roomConnect* raum2, *appJoin*, *roomJoin* raum1, *roomJoin* raum2. In **Abbildung 2** ist der sequenzielle Ablauf exemplarisch veranschaulicht.

Zur Umsetzung der Red5-Chatanwendung wird zunächst beim Starten der Red5-Anwendung initial der primäre Standardchatraum „default" definiert. Hierfür kommt entsprechend dem gleichnamigen Entwurfsmuster ein Value Object (VO) zum Einsatz, das später an Clients geschickt wird, wenn diese eine Liste verfügbarer Chaträume anfordern. Hierfür muss die entsprechende Klasse sowohl im Java- als auch im Actionscript-Code definiert und die entsprechende Paketstruktur muss in beiden Fällen identisch sein. Der entsprechende Code ist in Listing 1 dargestellt. Hierbei ist ein Standardkonstruktor ohne Parameter zwingend erforderlich, damit ein entsprechendes Objekt automatisch von der AMF-Verarbeitung instanziiert werden kann, wenn ein VO vom Client empfangen wird. Beim Verbindungsaufbau eines Clients zur Anwendung (*appConnect*) wird der übermittelte Benutzername an das lokale Clientobjekt als Attribut gebunden, damit der Name später nicht bei jedem Funktionsaufruf übermittelt werden muss.

Wenn ein Client einen neuen Chatraum erstellt, baut dieser unter Angabe des Raumnamens eine neue Serververbindung auf, beispielsweise zum URL *rtmp://10.0.0.1/Red5Chat/NeuerChatRaum*. Hierbei wird in der Red5-Anwendung die Methode *roomStart* aufgerufen. Damit andere Clients über diesen neuen Chatraum informiert werden, ist es notwendig, diese Information an alle bereits verbundenen Clients weiterzuleiten. Dazu wird über eine Liste aller Verbindungen vom globalen Scope (globale Variable *scope*) iteriert und mittels eines RPC-Aufrufs der Methode *clientGetChatRooms* die Liste aller Chaträume als Array von *ChatRoomVOs* übermittelt. Hierbei ist zu beachten, dass sämtliche an die Clientmethode zu übermittelnde Parameter in einem *Object* Array gebündelt werden müssen, damit der Client sie verarbeiten kann. Um einen RPC-Aufruf auf eine Clientverbindung zu initiieren, ist es notwendig, vorher die Verbindung zum Typ *IServiceCapableConnection* zu casten (Listing 2). Analog dazu ist die Red5-Methode *roomStop* umzusetzen.

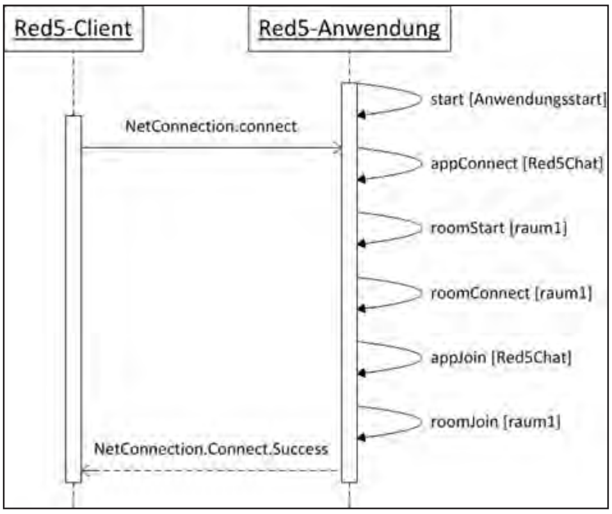


Abb. 2: Vereinfachter Sequenzablauf Verbindungsaufbau; URL: rtmp://10.0.0.1/Red5Chat/raum1

Ein ähnliches Verhalten ist beim Verbindungsaufbau eines neuen Clients umzusetzen. Hier muss die Präsenz eines neuen Benutzers an bereits verbundene Clients weitergeleitet werden. Allerdings betrifft dies nur Cli-

Listing 1

```
//// Red5Chat.java
package de.fhg.iese;

public class Red5Chat extends MultiThreadedApplicationAdapter {
  private HashMap<String, ChatRoomVO> chatRooms=
        new HashMap<String, ChatRoomVO>();

  @Override
  public synchronized boolean start(IScope scope) {
    chatRooms.put("default",new ChatRoomVO("default",false,false,null));
    return super.start(scope);
  }

  @Override
  public boolean appConnect(IConnection conn, Object[] params) {
    String userID=(String)params[0];
    System.out.println("new Client has connected to Red5Chat:
                                                      userID="+userID);
    conn.getClient().setAttribute("userID", userID);
    return super.appConnect(conn, params);
  }
}

//// ChatRoomVO.java
package de.fhg.iese.vo;

public class ChatRoomVO {
  public String  roomName;
  public Boolean isClosed;
  public Boolean isVideoEnabled;
  public String  password;
  public ChatRoomVO() {
    roomName="notSet"; isClosed=false;
                                  isVideoEnabled=false;password=null;
  }
}
```

ents, die sich im selben Raum befinden, zu dem sich der neue Benutzer verbindet. Daher wird hierfür nicht der globale Scope verwendet, sondern stattdessen der Scope des Benutzers, der die Verbindung initiiert hat. Entsprechend der Red5-Abarbeitung beim Verbindungsaufbau bietet sich hier die Methode *roomJoin* an. Zusätzlich ist zu beachten, dass dem neuen Benutzer alle bereits existierenden Benutzer mitgeteilt werden und analog jedem bereits existierenden Chatclient der neue Benutzer vorgestellt wird (Listing 3).

Um analog eine Mitteilung an verbundene Chatbenutzer zu gewährleisten, wenn ein Benutzer sich abmeldet, wird in der Red5-Methode *roomLeave* ein RPC-Aufruf *clientUserLeave* an alle Benutzer des betreffenden Raums (Scopes) gesendet.

Bisher wurden Methoden vorgestellt, die über das Red5-API zur Verfügung gestellt werden und vom Server selbstständig in den betreffenden Situationen aufgerufen werden. Wenn ein Client über RPC eine Methode aufruft, muss diese der Red5-Anwendung ebenfalls bekannt sein. Andernfalls erhält er das Statusevent *NetConnection.Call.Failed*. Am Beispiel der Erstellung eines neuen Chatraums ist eine entsprechende Methode exemplarisch verdeutlicht (Listing 4). Hier wird das bereits erwähnte *ChatroomVO* des zu erstellenden Chatraums an den Server übermittelt. In der Methode *serverCreate-*

Listing 2

```
@Override
public boolean roomStart(IScope room) {
  boolean isSuccessful = super.roomStart(room);
  if(isSuccessful && !room.getName().equals("default")) {
    for(Set<IConnection> connections : scope.getConnections()) {
      for(IConnection conn: connections) {
        if(conn instanceof IServiceCapableConnection) {
          IServiceCapableConnection curConnection=(IServiceCapableConnection) conn;
          curConnection.invoke("clientGetChatRooms", new
                                              Object[]{chatRooms.values().toArray()});
  }}}}
  return isSuccessful;
}
```

Listing 3

```
@Override
public boolean roomJoin(IClient client, IScope room) {
  String newClientID = (String)client.getAttribute("userID");
  boolean isSuccessful =super.roomJoin(client, room);
  if(isSuccessful) {
    // Scope des neu verbindenden Benutzers!
    IScope curScope=Red5.getConnectionLocal().getScope();
    Set<IConnection> clientConnections=client.getConnections();
    IServiceCapableConnection con=
        (IServiceCapableConnection)clientConnections.toArray()[0];
    // send 'own' clientNewUser
    con.invoke("clientNewUser", new Object[]{newClientID});
    // iterating over already connected users
    for(Set<IConnection> connections : curScope.getConnections()) {
      for(IConnection conn: connections) {
        IServiceCapableConnection userConnection=(IServiceCapableConnection)conn;
        String userID=(String)userConnection.getClient().getAttribute("userID");
        //hier sendet man dem neu verbundenen User die bereits existierenden User zu
        con.invoke("clientNewUser", new Object[]{userID});
        // hier sendet man einem bereits existierendem User die neue userID zu
        userConnection.invoke("clientNewUser", new Object[]{newClientID});
      }
    }
  }
  return isSuccessful;
}
```

Listing 4

```
@Override
public void serverCreateChatRoom(ChatRoomVO newChatRoom) {
  chatRooms.put(newChatRoom.roomName, newChatRoom);
  IConnection current=Red5.getConnectionLocal();
  IServiceCapableConnection con=(IServiceCapableConnection)current;
  con.invoke("clientChatRoomCreated", new Object[]{});
}

public void serverSendChatMessage(String msg) {
  IConnection current=Red5.getConnectionLocal();
  IScope curScope=current.getScope();
  String username=(String)current.getClient().getAttribute("userID");
  for(Set<IConnection> connections : curScope.getConnections()) {
    for(IConnection conn: connections) {
      IServiceCapableConnection con=(IServiceCapableConnection)conn;
      con.invoke("clientReceiveChatMessage", new Object[]{username, msg});
    }
  }
}
```

Listing 5

```
public class Red5Manager {
  private static const URL_RED5:String="rtmp://10.0.0.1/Red5Chat/";
  private var nc:NetConnection=null;
  public var model:ChatPresentationModel;

  public function red5Connect(userName:String):void {
    nc=new NetConnection();
    nc.client=this; // Antwortobjekt für Anfragen vom Red5-Server setzen
    nc.addEventListener(NetStatusEvent.NET_STATUS, handleNetStatus);
    // auf weitere Statusevents hören.
    nc.connect(URL_RED5 + model.currentRoomName);
  }

  public function red5Logout():void {
    nc.close();
    nc=null;
  }
}
```

ChatRoom wird dieses Objekt lokal gespeichert, bevor als Antwort die Methode *clientChatRoomCreated* auf dem Client aufgerufen wird, der die Kommunikation initiiert hat. Wenn ein Client eine Textnachricht durch Aufruf der Red5-Methode *serverSendChatMessage* sendet, ist diese ähnlich dem Verhalten in *roomJoin* an alle verbundenen Benutzer desselben Chatraums weiterzuleiten. Die entsprechende Umsetzung ist ebenfalls in Listing 4 dargestellt.

Im Folgenden wird auf die clientseitige Umsetzung der Beispielanwendung eingegangen.

Entwicklung der Clientanwendung von „Red5Chat"

Die Clientanwendung ist umfangreicher als die entsprechende Serveranwendung, denn neben der Kommunikationslogik mit dem Red5-Server ist auch Anwendungs- und Präsentationslogik nötig. Im vorliegenden Beispiel wird der Fokus auf die ActionScript-Klasse *Red5Manager* gelegt. Diese steuert die Kommunikation mit dem Red5-Server. Auf Präsentationscode in MXML wird an dieser Stelle verzichtet.

Abbildung 3 veranschaulicht den Grundaufbau des Flexclients in einer vereinfachten Architektursicht. Aus Red5-Sicht stellt die ActionScript-Klasse *Red5Manager* die zentrale Kommunikationsinstanz mit dem Red5-Server dar. Hier werden via RPC entfernte Methoden auf dem Red5-Server aufgerufen, um z. B. eine Liste aller verfügbaren Räume zu bekommen. Weiterhin wird hier die clientseitige Verarbeitung von Audio- und Videostreaming durchgeführt, die im zweiten Teil der Artikelserie beschrieben wird. *Red5Manager* verwendet die bereits beschriebenen VOs aus dem Modell für den Datenaustausch mit dem Red5-Server. In unserer Beispielanwendung wurde zur Umsetzung des Clients ein MVVM-Konzept (Model View ViewModel) [13] gewählt. Beispielsweise empfängt der *Red5Manager* in der Remote-Handler-Methode eine Liste aller Räume, die an das Presentation Model (PM) bzw. View Model (VM) *ChatPresentationModel* übergeben wird. Das PM ist für die Verarbeitung und Aufbereitung des Modells verantwortlich. Durch Data Bindings werden veränderte Werte bzw. Zustände automatisch an die Views zur Darstellung weitergegeben. Die zentrale Basisklasse für die Zusammenarbeit mit dem Red5-Server stellt *flash.net.NetConnection* dar. Hierüber erfolgt der Verbindungsaufbau, der anschließend Methodenaufrufe auf Red5-Seite über RPC erlaubt. In Listing 5 ist der notwendige Code für einen Verbindungsaufbau verdeutlicht.

Bei Operationen auf dem *NetConnection*-Objekt sollte immer

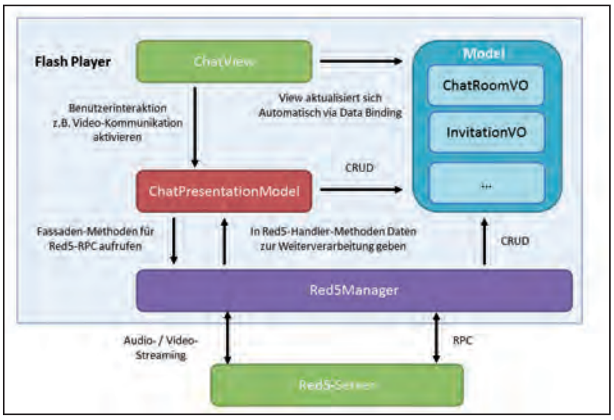


Abb. 3: Vereinfachte Darstellung des Red5-Clients und Red5-Servers

auf *null* geprüft werden, was in Listings in diesem Artikel der Übersicht halber nicht gemacht wurde.

Nachdem der Benutzer in der Anmeldemaske seinen Benutzernamen angegeben hat, ruft das PM die Methode *red5Connect* im *Red5Manager* auf. Über die *connect*-Methode wird die eigentliche Verbindung durch Angabe des URLs durchgeführt, um sich mit einem konkreten Raum in einer Red5-Anwendung zu verbinden. Die Methode *red5Logout* zeigt, wie man die Verbindung mit dem Red5-Server trennt. Das ist gleichbedeutend mit dem Austritt aus dem aktuellen Raum. Nach Aufruf der *close*-Methode auf dem *NetConnection*-Objekt sollte zunächst die Bestätigung durch das Statusevent *NetConnection.Connect.Closed* abgewartet werden, bevor das *NetConnection*-Objekt *nc* durch Setzen auf *null* für die Garbage Collection freigegeben wird. Zur Vereinfachung wurde in Listing 5 darauf verzichtet.

In *red5Connect* wurde exemplarisch ein Listener für ein *NetStatusEvent* registriert. Zusätzlich können hier zur besseren Fehlerbehandlung noch weitere *NetStatusEvents* abgefangen werden (Tabelle 2). Listing 6 zeigt die Implementierung der Handler-Methode für eintreffende *NetStatusEvents*. Details über das Ereignis erfährt man via *event.info.code*. Im Falle eines erfolgreichen Verbin-

NetConnection-Statusevents (Auswahl)

In Tabelle 2 ist eine Auswahl der im Beispiel verwendeten Statusevents für NetConnections aufgelistet.

Zusätzlich sind noch weitere Event Listener bei NetConnections für die folgenden Ereignisse empfehlenswert:

- *IOErrorEvent.IO_ERROR*
- *IOErrorEvent.NETWORK_ERROR*
- *AsyncErrorEvent.ASYNC_ERROR*

Statusevent	Grundfunktion
NetConnection.Connect.Success	Verbindungsaufbau erfolgreich
NetConnection.Connect.Failed	Verbindungsaufbau fehlgeschlagen
NetConnection.Connect.Closed	Verbindung geschlossen
NetConnection.Connect.Rejected	Verbindungsaufbau abgelehnt
NetConnection.Connect.InvalidApp	Ungültige Anwendung für Verbindungsaufbau
NetConnection.Call.Failed	RPC-Aufruf fehlgeschlagen

Tabelle 2: Übersicht von NetConnection-Statusevents

dungsaufbaus wird die Liste der verfügbaren Chaträume (*serverGetChatRooms*) vom Server abgerufen. Dies erfolgt über einen RPC-Aufruf auf dem Red5-Server, indem auf dem *NetConnection*-Objekt die Methode *call* aufgerufen wird. Als erstes Argument übergibt man den Namen der entfernten Red5-Methode. Das zweite Argument ist ein optionales Responder-Objekt, das hier zunächst nicht verwendet wird. Anschließend können weitere Argumente angegeben werden, die direkt an die Red5-Methode als Parameter übergeben werden.

Bisher wurde gezeigt, wie über das *NetConnection*-Objekt ein Red5-Raum betreten und verlassen werden kann. Zusätzlich können über RPC entfernte Methoden auf dem Red5-Server aufgerufen werden. Für Methoden zum Datenaustausch mit dem Red5-Server wird ein Namensschema verwendet, bei dem Methoden mit Präfix „red5" eine entfernte Red5-Methode aufrufen und Methoden mit Präfix „client" entsprechend Handler-Methoden darstellen, die vom Red5-Server aufgerufen werden. Listing 7 zeigt die Implementierungen der Methoden *red5SendChatMessage* und *clientReceiveChatMessage*. Diese werden verwendet, um Nachrichten im Textchat zwischen den Benutzern eines Raums auszutauschen. Die Handler-Methode übergibt die Daten an das PM, sodass die View die empfangene Nachricht anzeigen kann.

In diesem Artikel wurde bei der Entwicklung des Red5-Clients der Fokus auf den Verbindungsauf- und -abbau sowie den Aufruf von entfernten Red5-Methoden und Datenaustausch zwischen Client und Server gelegt. Der nächste Teil der Artikelserie geht im Detail auf die clientseitige Verarbeitung von Audio- und Videostreaming ein.

Listing 6

```
private function handleNetStatus (event:NetStatusEvent):void {
  if (event.info.code == "NetConnection.Connect.Success")
    nc.call("serverGetChatRooms", null);
  else if (event.info.code == "NetConnection.Connect.Failed") {
    // Verbindungsaufbau fehlgeschlagen, Meldung für Benutzer ausgeben
  } else if (event.info.code == "NetConnection.Connect.Closed") {
    // client- oder serverseitige Trennung der Verbindung
    // Verarbeitung notwendig ...
  }
}
```

Listing 7

```
public function red5SendChatMessage(msg:String):void {
  nc.call("serverSendChatMessage", null, msg);
}
public function clientReceiveChatMessage(user:String, msg:String):void {
  model.receiveChatMessage(user, msg);
}
```

Entwickeln von Red5-Clients mit ActionScript/Flex

Apache Flex [9] (bis 2011 Adobe Flex) liegt aktuell in Version 4.9 vor und stellt ein SDK für die Entwicklung von Rich Internet Applications (RIA) dar, das auf der Flash-Plattform aufsetzt. Damit können browserbasierte RIAs oder Desktop-RIAs (Adobe AIR) entwickelt werden. MXML stellt die Auszeichnungssprache für die Gestaltung der Views dar. Logik wird mit ActionScript geschrieben. Der kommerzielle Adobe Flash Builder [10] ist eine IDE auf Eclipse-Basis zur Entwicklung von Flex-Apps von Adobe. Alternativen stellen FDT [11] und FlashDeveloper [12] dar.

Fazit

Red5 bietet Entwicklern die Möglichkeit, verteilte Multimediaanwendungen umzusetzen, ohne die Komplexität einer Clientserverarchitektur für Streaminganwendungen selbst implementieren zu müssen. Stattdessen kann der Entwickler sich auf die serverseitige Anwendungslogik und Präsentationsschicht im Client konzentrieren. Die Infrastruktur basiert dabei auf bekannten und bewährten Technologien aus der Java- und Flash-Welt. Komplexe Aspekte wie beispielsweise Audio- und Videostreaming über den Red5-Server werden auf Basis des RTMP-Protokolls ohne eigenen Implementierungsaufwand bereitgestellt. Entwickler müssen lediglich die clientseitige Veröffentlichung und das Abrufen und Darstellen der Medien-Streams umsetzen. Speziell für Entwicklungsteams aus dem Java-Umfeld bietet der Red5-Medienserver eine komfortable und kostenlose Lösung, um Clientserver-basierte Streaminganwendungen effizient zu entwickeln.

Dipl.-Inf. Sebastian Weber arbeitet seit 2007 am Fraunhofer Institut für experimentelles Software Engineering (IESE) im Bereich UX und Softwareentwicklung für mobile Endgeräte. Er entwickelt seit einigen Jahren Anwendungen auf Apache-Flex- und Adobe-AIR-Basis.

M. Sc. Cornelius Moucha arbeitet seit 2011 am Fraunhofer Institut für experimentelles Software Engineering (IESE) im Bereich Sicherheit und Softwareentwicklung.

Links & Literatur

[1] https://code.google.com/p/red5

[2] http://www.apache.org/licenses/LICENSE-2.0

[3] http://www.adobe.com/products/adobe-media-server-family.html

[4] http://tomkrcha.com/?p=671

[5] http://www.ehow.com/info_12229684_rtmp-vs-rtsp.html

[6] http://www.eclipse.org/downloads/

[7] http://www.red5.org/red5-ide-plugin/

[8] http://www.red5.org/downloads/docs/red5-reference-1.0.pdf

[9] http://flex.apache.org/

[10] http://www.adobe.com/de/products/flash-builder.html

[11] http://fdt.powerflasher.com/

[12] http://www.flashdevelop.org/

[13] http://nirajrules.wordpress.com/2009/07/18/mvc-vs-mvp-vs-mvvm/

Multimedia mit Processing

Bilder und Töne

In den vorherigen Artikeln haben wir uns mit 2-D- und 3-D-Ausgaben im leeren Raum beschäftigt. Wir konnten uns zwar im Raum bewegen, aber das Ganze kann man auch interaktiver und lebhafter gestalten. Daher möchten wir in diesem Artikel einen Augmented Reality Equalizer bauen: Wir zeichnen auf den ausgedruckten Marker-Codes im Bild der Webcam die Amplitude – also die Lautstärke – der einzelnen Frequenzbänder des Mikrofoneingangs.

von Stefan Siprell

Mit der Bibliothek *video* ist es denkbar einfach, sich Bilder aus der Webcam zu besorgen. Der Einstiegspunkt ist die Klasse *processing.video.Capture*. Nachdem die Bibliothek über den Menüpunkt SKETCH | IMPORT LIBRARY | VIDEO eingebunden wurde, können wir *Capture* instanziieren. Es stehen verschiedene Konstruktoren zur Verfügung. In der Regel reicht schon *new Capture(this)* aus. Falls man von mehreren angeschlossenen Kameras eine bestimmte ansprechen oder die Auflösung selbst bestimmen möchte, stehen auch hierfür Konstruktoren bereit:

- *available()*: Gibt *true* zurück, falls mindestens eine Kamera gefunden wurde
- *start()*: Fängt mit der Aufnahme an
- *stop()*: Beendet die angefangene Aufnahme
- *read()*: Liest das aktuelle Bild in den Puffer, damit ist das Capture-Objekt auch als normales Bild zu verwenden – mehr später
- *list()*: Statische Methode, welche die vorhandenen Kameras und ihre Konfigurationen auflistet

Toneingang

Eine weitere spannende Bibliothek für Processing muss zunächst aus dem Internet geladen werden, was mit Processing 2.0.x sehr einfach ist. Analog zum letzten Artikel rufen wir die Bibliothek über SKETCH | IMPORT LIBRARY | ADD LIBRARY ... in der Suchmaske ab. In der Maske suchen wir nach *minim* und installieren die Bibliothek.

Artikelserie

Teil 1: Einführung ins Processing, Nutzen der 2-D-Rendering-Engine
Teil 2: Nutzen der 3-D-Rendering-Engine mit Kamerafahrten
Teil 3: Computer Vision und Augmented Reality mit Processing
Teil 4: Professionelle Datenvisualisierung mit Java

minim vereint drei Bibliotheken (*JavaSoundAPI*, *Tritonus*, *MP3SPI*) in sich und vereinfacht deren Umgang in typischer Processing-Manier: keine Callbacks, keine obskuren Arrays, keine Versionskonflikte. Mit *minim* lassen sich Sounddateien abspielen, Metadaten auswerten, Frequenzen analysieren und vieles mehr. In unserem Beispiel möchten wir einfach das Mikro abhören, um zum Beispiel auch eigene Töne zu analysieren.

Analog zum *Capture* gibt es die Einstiegsklasse *ddf.minim.Minim_*, die es zu instanziieren gilt, z. B. mit *new Minim(this)*. Zwei Methoden bieten einen sehr einfachen Zugriff auf unterschiedliche Tonquellen:

- *loadFile()*: Lädt eine Datei und stellt ein Player-Objekt für den Track zur Verfügung
- *getLineIn()*: Holt sich das Eingabesignal auf dem Eingabe-Port und stellt ein Input-Objekt zur Verfügung

Mit *getLineOut()* lassen sich Töne und Effekte ausgeben, was aber in dem Artikel nicht weiter von Belang ist. Die Fast-Fourier-Transformation wiederum ist wichtig, um das Signal in verschiedene Frequenzbänder aufzulösen. Die zeitliche Länge und die Abtastrate des Signals muss beim Konstruktor mitgegeben werden. In der Regel kann dies von der Audioquelle ausgelesen werden: *new FFT(source.bufferSize(), source.sampleRate())*.

Augmented Reality

Mikrofone, Töne, Kameras und Bilder sind bekannte Konzepte – Augmented Reality ist schon spannender: Wir werden in ein natürliches Bild vom Rechner generierte Grafiken einfügen.

Hierzu muss der Rechner im eingehenden Bild nach so genannten *Markern* suchen. Das sind einfache Grafiken mit einem hohen Kontrast, die nicht rotationssymmetrisch sind. Aus der perspektivischen Verzerrung des Markers kann der Rechner die Lage des Markers rekonstruieren und einen berechneten 3-D-Körper auf dem Marker ein-rechnen. In unserem Beispiel sind die Körper einfache Kuben, was der Einfachheit des Artikels und nicht den Einschränkungen der Anwendung geschuldet ist.

Auch für Augmented-Reality-Anwendungen gibt es für Processing ein Framework: nyAR4psg. Es ist ein Wrapper für das nyARToolkit, das komplett in Java verfügbar ist. Die Bibliothek ist so einfach zu nutzen, dass man anhand weniger Codebeispiele die Funktionalität ableiten kann. Leider kann man es nicht anders erlernen, da die Dokumentation in Japanisch gehalten ist.

Bevor es ans Coden geht, müssen die Marker erstellt werden. Ein Marker besteht hier aus einer binären computerlesbaren Beschreibung als Teil der PATT-Datei. Dazu gibt es eine sichtbare Repräsentation der PPM-Grafik. Diese Kombinationen können mit dem genannten Kommandozeilentool [1] generiert werden. Die Grafiken werden anschließend ausgedruckt und ausgeschnitten. In unserem Beispiel werden wir mit vier verschiedenen Markern arbeiten.

Mit den Markern zur Hand und auf der Festplatte binden wir die Bibliothek ein. Diesmal muss man die Bibliothek mit dem Browser herunterladen [2], das entpackte Verzeichnis in NyAR4psg umbenennen, da es Processing sonst nicht anerkennen wird, und in das Verzeichnis *~/Processing/libaries* verschieben. Jetzt kann man es unter SKETCH | IMPORT LIBRARY ... einbinden. Die Klasse *jp.nyatla.nyar4psg.MultiMarker* dient als zentrales AR-Objekt und kann instanziiert werden. Hierzu benötigt man die Sketch, Höhe und Breite des Bilds, eine Kamerabeschreibung und ein Konfigurationsobjekt.

Da die optischen Systeme in den Kameras nicht perfekt sind (siehe auch das Bild unter [3]), müssen diese kalibriert werden, um möglichst genaue Angaben zu liefern. Das ist zeitaufwändig und für einfache Anwendungen nicht vonnöten, sodass man sich die Standarddatei *camera_para.dat* einfach aus dem Beispielverzeichnis der Bibliothek kopieren kann. Der Multimarkerinstanz kann man mit *addMarker()* Referenzen zu den PATT-Dateien nennen, damit das Framework im Webcambild nach den ausgedruckten Markern suchen kann. Dazu mehr später.

Set-up-Methode und Instanzenvariablen

Die Konstrukte aus Listing 1 sollten mit der vorangegangenen Einführung einfach zu verstehen sein. *TextFont* ist an dieser Stelle neu und dient dazu, die Schriftart für folgende Schreiboperationen zu bestimmen. Das kann natürlich auch Courier, Arial oder ein beliebiger anderer Schriftsatz sein. Das *Capture*-Objekt wird mit dem Namen der Kamera parametrisiert, was aber nicht notwendig ist, wenn man nur eine Kamera angeschlossen hat. Das Mikrofon wird über dem Line-in-Port aufgemacht. Für die Konfiguration des ARToolkits ist ein Verweis auf die *camera_part.data* notwendig, die über SKETCH | ADD FILE ... zuerst eingebunden wird.

In der Schleife werden die Marker geladen, und zwei Arrays werden bestückt. Im ersten Array werden zufäl-

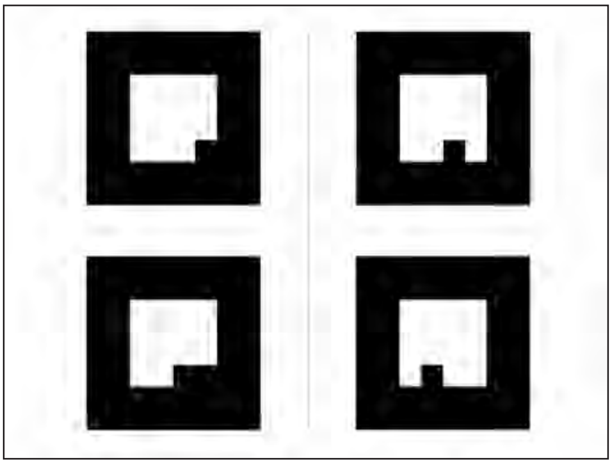


Abb. 1: Marker

Listing 1

```
String myPatterns = "/Users/stefansiprell/Documents/Processing/ARStuff";

Minim minim;
MultiMarker nya;
int numMarkers = 4;

FFT fft;
Capture iSight;
AudioInput mic;

color[] colors = new color[numMarkers];
float[] frequencies = new float[numMarkers];

void setup(){
  size(640, 480, OPENGL);
  frameRate(30);
  textFont(createFont("Gill Sans", 80));
  textAlign(CENTER, CENTER);
  textSize(10);

  println("Spreche Kamera an.");
  iSight = new Capture(this, width, height, "FaceTime Camera (Built-in)");
  iSight.start();

  println("Spreche Micro an.");
  minim = new Minim(this);
  mic = minim.getLineIn();
  fft = new FFT(mic.bufferSize(), mic.sampleRate());

  println("Konfiguriere AR Framework.");
  nya = new MultiMarker(this, width, height, "camera_para.dat",
                                              NyAR4PsgConfig.CONFIG_DEFAULT);
  nya.setLostDelay(1);
  for(int i = 0; i <numMarkers; i++){
    nya.addARMarker("4x4_"+(i+1)+".patt", 80);
    colors[i] = color(random(255), random(255), random(255), 160);
    frequencies[i] = ((float)20000/numMarkers)*(i+1);
  }
```

Abb. 2: Fertige Ausgabe

Listing 2

```
void draw(){
  iSight.read();
  fft.forward(mic.mix);
  background(iSight);
  nya.detect(iSight);
  drawBoxes();
}
```

lige halbtransparente Farben für die späteren Würfel definiert, und die Frequenzbänder werden im zweiten Array gespeichert. Je mehr Marker man verwendet, desto feingliedriger wird das Spektrum zwischen 0 und 20 000 KHz aufgeteilt. Normalerweise sind 20 KHz das Ende bei digitalen Aufnahmen.

Listing 3

```
void drawBoxes(){
  nya.setARPerspective();
  for(int i=0; i<numMarkers; i++){
    if((!nya.isExistMarker(i))){continue;}
    setMatrix(nya.getMarkerMatrix(i));
    drawBox(i);
  }
  perspective();
}

void drawBox(int i){
  //scale(1, -1);
  float scale = map(fft.getFreq(frequencies[i]), 0, 0.05, 0.7, 1.5);
  scale(scale);
  translate(0,0,20);
  lights();
  stroke(0);
  fill(colors[i]);
  box(40);
  noLights();
  translate(0,0,20.1);
  fill(0);
  text(""+frequencies[i], -20, -20, 40, 40);
}
```

Die *draw()*-Methode in Listing 2 ist diesmal nicht nur für das Zeichnen des Bilds verantwortlich, sondern aktualisiert die Eingaben. Zunächst muss das nächste Bild aus der Kamera geladen werden. Das Bild wird zum einen als Hintergrund gesetzt, und zum anderen wird der Multimarker mit dem aktuellen Bild gefüttert. Die Marker werden in der letzten Operation gesucht und stehen weiter unten noch zur Verfügung. Hat der Fast-Fourier-Transformer das aktuell aufgenommene Signal analysiert, sind alle Daten vorhanden, um die Boxen zu zeichnen.

Zum Zeichnen der Kuben in die Ausgabe ruft die *draw()*-Methode die *drawBoxes()*-Methode auf. Die Perspektive für alle Ausgaben wird an dem Bild ausgerichtet. Anschließend gehen wir durch die Definitionen durch und lassen das Framework nach dem Marker auf dem Bild suchen. Wurde keines gefunden, suchen wir einfach weiter. Sonst setzen wir die Matrix für die folgenden Zeichenoperationen – damit ist sichergestellt, dass die 3-D-Körper immer auf dem Marker sitzen. Nachdem die Boxen gezeichnet worden sind, stellen wir noch die normale – nicht ausgerichtete – Perspektive wieder her.

Das eigentliche Zeichnen der Box ist nicht neu nach dem letzten Artikel aus dieser Serie. Wir stellen die Koordinaten auf den Kopf, damit Processing ein natürlicheres Koordinatensystem nutzt. Weiterhin definieren wir noch eine weitere Skalierung, um die Größe der Objekte zu bestimmen. Die Größe leitet sich von der Amplitude des jeweiligen Frequenzbands ab, das sich bei den normalen Mikrofon- und deren Digitalisierungseinstellungen zwischen 0 und 0,05 bewegt. Den Eingangswertebereich des Mikrofons bilden wir auf den Ausgabewertebereich zwischen 0,75 und 1,5 mit der Funktion *map()* ab. Mit den entsprechenden Licht- und Farbeinstellungen zeichnen wir die Würfel. Schlussendlich muss noch der Text für das jeweilige Frequenzband auf den Würfel gezeichnet werden. Für das weitere Vertiefen gibt es schöne Beispiele auf [4]. Zu guter Letzt sollte man sich auch das eigentliche ARToolkit [5] anschauen, von dem sich alles ableitet.

Stefan Siprell (stefan.siprell@codecentric.de) ist Geschäftsführer der codecentric in Karlsruhe. Seine fachlichen und technischen Schwerpunkte liegen im Agile Software Enigneering. In der Freizeit beschäftigt er sich mit Arduino, Processing und Co.

Links & Literatur

[1] http://www.cs.utah.edu/gdc/projects/augmentedreality

[2] http://sourceforge.jp/projects/nyartoolkit/releases

[3] http://commons.wikimedia.org/wiki/File:Boesendorfer_003.jpg

[4] http://www.creativeapplications.net/processing/augmented-reality-with-processing-tutorial-processing

[5] http://www.hitl.washington.edu/artoolkit

Ansteuerung von USB-Schnittstellen via HDI mit Java

Java fährt Bus

Quellcode zum Download

Früher war alles einfach besser. Jeder PC hatte eine oder zwei serielle und eine parallele Schnittstelle(n), die auch relativ einfach per Java angesprochen werden konnten. Diese Schnittstellen wurden fast vollständig durch die USB-Schnittstellen verdrängt. Wie kann man nun auf die USB-Schnittstelle per Java zugreifen? Und wie kann die einmal programmierte Java-Anwendung dann auch noch auf Windows, Mac OS X, Linux und anderen Systemen ausgeführt werden, ohne dass Anpassungen am Quellcode nötig sind – nach dem alten Sun-Motto „Write once, run anywhere"?

von Thomas Wenzlaff

Der weit verbreitete Universal Serial Bus (USB) wird für die Ansteuerung externer Geräte, wie zum Beispiel Mäuse, Tastaturen, Drucker und andere Hardware (Bluetooth HID), verwendet. Die Geräte können dank der Hot-Plugging-Fähigkeit im laufenden Betrieb miteinander verbunden und sogar über USB noch mit Energie versorgt werden. Seit 2011 hat sich USB in der Version 3.0 auf dem Massenmarkt verbreitet, trotz der von Apple eingeführten Thunderbolt-Schnittstelle in direkter Konkurrenz dazu.

An der Haltestelle

Die USB-Schnittstellen benötigen für jedes Betriebssystem eigene Device-Treiber. Und da fängt das Elend für die Entwickler schon an: Device-Treiber in C oder C++ programmieren und das für verschiedene Plattformen mit verschiedenen 32- und 64-Bit-Systemen, nur um auf den USB-Port zuzugreifen, und das alles noch testen – das will man als Java-Entwickler eigentlich nicht. Es ist seit einiger Zeit auch nicht mehr nötig, denn seit 2010 gibt es das „javahidapi"-Projekt [1], das einen JNI-Wrapper über die C-/C++-Device-Treiber für Windows, Linux, Mac und auch andere Systeme zur Verfügung stellt. Das javahidapi besteht aus nur einem JAR, das unter [2] kostenlos geladen werden kann. Optional können von dort zusätzlich der Sourcecode und die Javadoc geladen werden. Wer Lust hat, kann sich auch alles aus dem Sourcecode selbst zusammenbauen. Dies ist aber normalerweise nicht nötig. Dieses eine JAR enthält auch alle nötigen nativen Treiber. Für Windows sind das die nötigen **.ddl*, für Linux die **.so* und für Mac die **.jnilib*. Auch wird das Projekt aktuell noch weiterentwickelt, und eventuell nötige Fixes werden bereitgestellt. So ist gerade in Issue 48 vom 01.02.2013 ein Thread eröffnet worden, der eine Version für den beliebten ARM-Prozessor des Raspberry Pi [3] bereitstellt.

Die aktuelle Version 1.1. [4] steht unter sehr liberalen Softwarelizenzen. Es kann zwischen der GNU-, BSD- und einer HIDAPI-Lizenz gewählt werden. Damit wird der weiten Verbreitung in vielen Projekten Vorschub geleistet.

Der Bus fährt pünktlich los

Um auf den USB-Port zugreifen zu können, muss nur das 199 KB große *hidapi-1.1.jar* geladen und dem Classpath hinzugefügt werden. Das ist für die Installation schon alles. Dann muss das API einmalig initialisiert werden, damit die nativen Treiber für die jeweiligen Betriebssysteme geladen werden. Dazu gibt es in der Klasse *ClassPathLibraryLoader* eine entsprechende Methode, die das alles automatisch erledigt. Es reicht also dieser Aufruf in der jeweiligen Klasse:

```
static {
  ClassPathLibraryLoader.
                        loadNativeHIDLibrary();
}
```

Beschäftigt man sich mit der HID-USB-Programmierung, will man meistens zuerst wissen, ob und welche Devices angeschlossen und verfügbar sind. Das ist auch eine der am häufigsten gestellten Fragen in den diversen Foren im Internet zu dem Thema. Also: Wie ermittle ich alle angeschlossenen USB-Geräte?

Im javahidapi gibt es die Klasse *HIDManager* mit der Methode *listDevices()*, die alle verfügbaren Geräte auflistet. Es werden von jedem Gerät die in Tabelle 1 aufgelisteten Attribute an Informati-

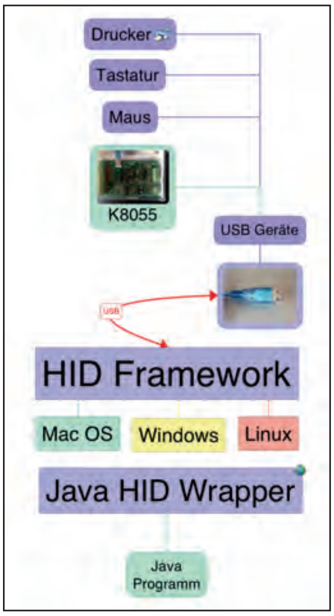


Abb. 1: Architektur

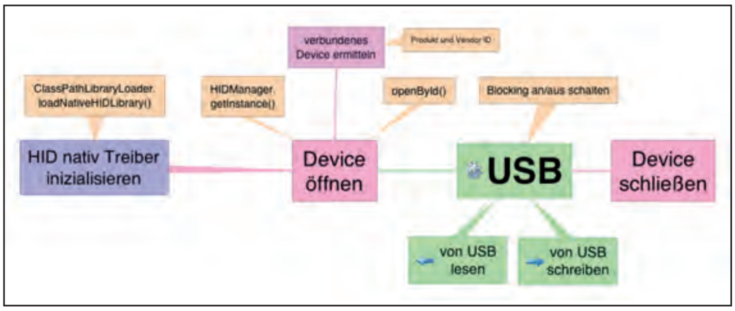


Abb. 2: Überblick zum Flow des javahidapi

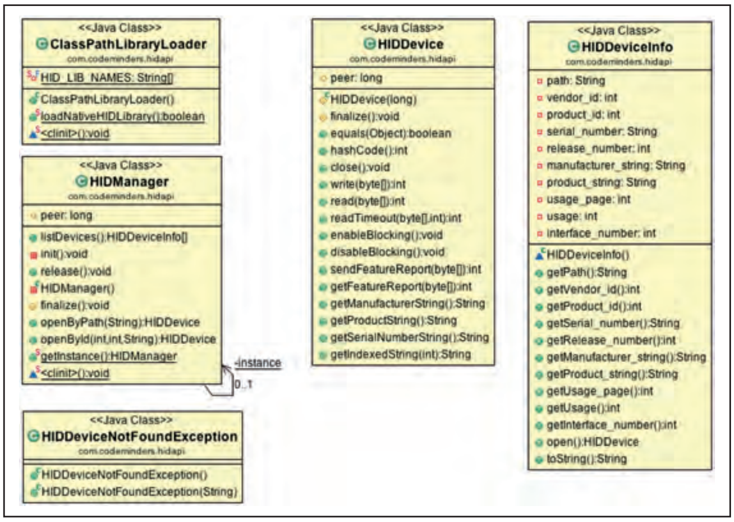


Abb. 3: javahidapi-UML-Klassendiagramm

onen im *HIDDeviceInfo*-Objekt geliefert. **Abbildung 1** zeigt die Architektur.

Wichtig sind dabei die *vendor_id* und die *product_id*. Diese beiden IDs werden für die Adressierung des Geräts benötigt. Die *serial_number* ist optional und dient zur Unterscheidung, wenn gleiche Geräte vorhanden sind. Alle anderen Attribute müssen nicht unbedingt mit sinnvollen Werten gefüllt sein, je nach Hersteller der Hardware sind dort Einträge zu finden oder auch nicht.

Wenn bekannt ist, welche USB-Geräte derzeit verbunden sind und die *vendor_id* und die *product_id* ermittelt

Attribut	Beispiel K8055	Beispiel Tastatur
vendor_id	4303	1452
product_id	21760	598
path	USB_10cf_5500_ 0x7fc869416aa0	Bluetooth_05ac_ 0256_0x7fdff2134150
serial_number		44-2a-61-ed-d7-f7
release_number	0	80
manufacturer_string	Velleman	Apple
product_string	USB K8055	Toms Tastatur
usage_page	65280	1
usage	1	6
interface_number	-1	-1

Tabelle 1: Attribute, die an das „HIDDeviceInfo"-Objekt geliefert werden

wurden, kann das Gerät über eine Instanz des *HIDManagers* mit *openById* wie folgt geöffnet werden:

```
HIDManager manager = HIDManager.getInstance();
manager.openById(vendor_id, produkt_id, null);
```

Es können beim Öffnen auch *HIDDeviceNotFoundException*soder *IOException*s geworfen werden, wenn das Gerät zwischenzeitlich getrennt wurde oder sonstige I/O-Probleme auftreten.

On tour

Nun kann das Gerät für Lese- und Schreiboperationen verwendet werden. Dafür stehen die zwei Funktionen *read(byte[])* und *write(byte[])* der *HIDDevice*-Klasse bereit. Über diese einfachen Funktionen können Daten an das USB-Gerät gesendet oder auch empfangen werden. Wenn es beim Lesen oder Schreiben Probleme gibt, wird eine I/O-Exception geworfen.

To block or not to block

Für die Verbindung zum USB-Device kann jederzeit für Lesefunktionen der Blocking-Modus gesetzt werden (**Abb. 2**). Ist das der Fall, wird so lange gewartet, bis eine Eingabe vom Gerät geliefert wird. Erst dann wird die Programmausführung fortgesetzt. Im Non-blocking-Modus wird sofort, ohne zu blocken, in der Programmausführung weitergemacht, ohne auf das Gerät zu warten.

Der Blocking-Modus kann leicht mit der Funktion *enableBlocking()* der Klasse *HIDDevice* eingeschaltet und mit *disableBlocking()* ausgeschaltet werden.

Endstation

Wenn kein Zugriff mehr benötigt wird, muss das Gerät mit

```
device.close();
manager.release();
```

geschlossen werden. Diese Aufrufe können ohne Probleme auch mehrfach geschehen. Das ist schon alles. Mit nur fünf einfachen Klassen (UML-Diagramm in **Abb. 3**) und einer Handvoll Methoden kann leicht auf die USB-Schnittstelle zugegriffen werden.

Und ab zum TÜV

Um das noch zu veranschaulichen und den Einstieg in die USB-Programmierung zu erleichtern, ist unter [7] ein ganzes Eclipse-Java-Projekt zu finden. In dem Beispielprojekt ist für den Zugriff auf das USB-Device K8055 (*TestK8055.java*) der Firma Vellman ein Beispiel enthalten. Wer dieses Board nicht hat, kann sich unter [8] ein Video vom Ablauf des Testprogramms ansehen.

Alternativ kann man lediglich das *TestTastatur.java*-Programm ansehen – dort wird beispielhaft gezeigt, wie eine USB-Tastatur, die wohl häufiger vorhanden ist als das K8055, über das javahidapi per USB und nicht per *System.in* abgefragt werden kann.

Obwohl man dies in einigen Projekten zu hören bekommt: In der Softwareentwicklung gilt keineswegs das Motto „Wer testet, ist feige". Das ist schlicht nicht professionell. Deshalb muss natürlich auch jede Zielplattform ausführlich, am besten mit automatischen JUnit-Tests, getestet werden. Die JUnit Testsuites können für jedes Betriebssystem separat erstellt und ausgeführt werden. Denn gerade mit verschiedenen Zielplattformen und auch in Bezug auf USB kann es Probleme geben, da für jede Zielplattform separate und unterschiedliche native Treiber nötig sind, die tief in die jeweiligen Betriebssysteme eingreifen und auch von der entsprechenden Hardware nicht immer hundertprozentig entkoppelt sind.

So wurde auch im Testprojekt eine lauffähige Version auf dem Mac OS X 10.8.3 erstellt. Beim anschließenden Test auf Windows 7 mit 64 Bit gab es das Problem, dass die Datenlänge beim Schreiben nicht gültig war. Nach einer kurzen Recherche in der Issues-Liste [9] des javahidapi-Projekts konnte der Fehler schnell gefixt werden. So viel zum Thema „Write once, run anywhere".

Fazit: Die Heimreise

Das javahidapi ist ein einfach einzusetzendes Framework: Es besteht nur aus fünf leicht zu verstehenden Klassen und ist für die häufigsten Betriebssysteme verwendbar. In nur 30 Minuten kann per Java leicht auf die USB-Schnittstelle [10] zugegriffen werden.

Thomas Wenzlaff (java@wenzlaff.de) ist IT-Berater der Frobese Informatikservices GmbH, Hannover. Seine Schwerpunkte liegen in der Beratung, Konzeption und Entwicklung von Programmen im Java-Umfeld – nicht nur für Embedded Devices und Tests, sondern hauptsächlich im J2EE-Umfeld von Banken und Versicherungen mit Maven und Jenkins.

Links & Literatur

[1] Homepage javahidapi: http://code.google.com/p/javahidapi/

[2] Download hidapi-1.1.jar: http://code.google.com/p/javahidapi/downloads/list

[3] Raspberry Pi: http://www.raspberrypi.org und http://www.finchrobot.com/compiling-javahidapi-raspberry-pi

[4] USB-HID-API-Download von (hidapi-0.7.0.zip): https://github.com/signal11/hidapi/downloads

[5] Infos und Kanäle mit guter Beschreibung über das K8055 (engl.): http://www.rhinoman.org/K8055/K8055_Upgrade.htm

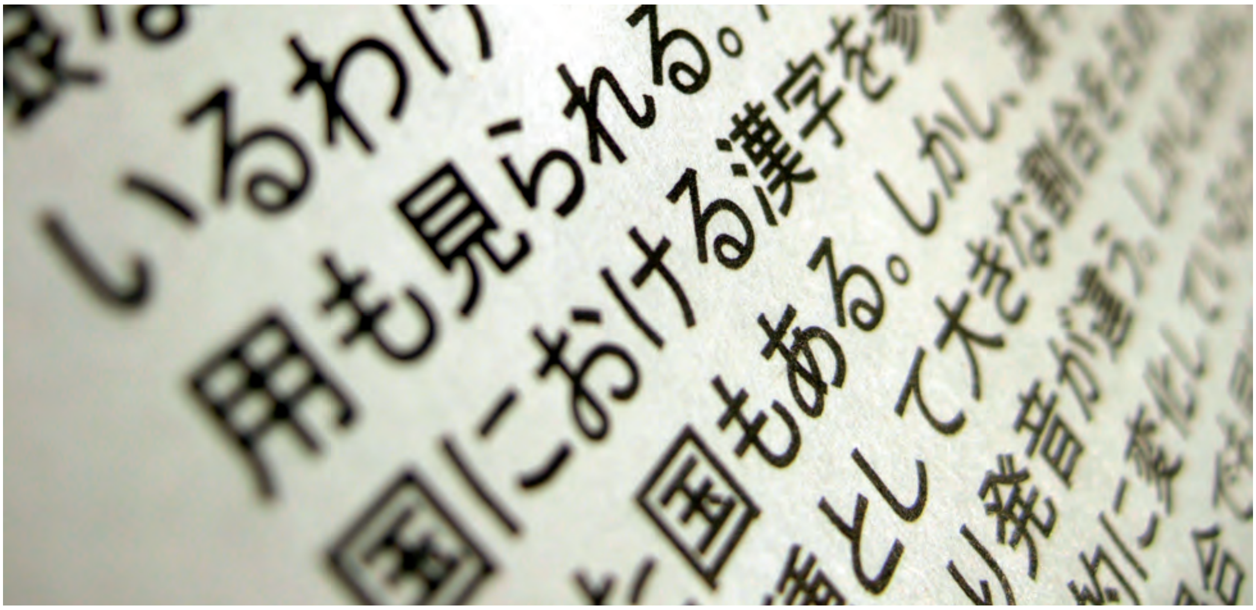
[6] Allgemeine USB-Beschreibung (engl.): http://www.kadtronix.com/usbhidapi_usr.htm

[7] Eclipse-Beispielprojekt TWUsbMulti: http://www.wenzlaff.de/download/twusbmulti/twusbmulti.zip

[8] Video vom K8055-Beispielprogramm: http://www.youtube.com/watch?v=UX_3bE33Xj8

[9] javahidapi-Issues-List: http://code.google.com/p/javahidapi/issues/list

[10] USB allgemein: https://de.wikipedia.org/wiki/USB



© iStockphoto.com/sandramo

Eine Java VM für Mikrocontroller

HaikuVM

Java auf dem Raspberry Pi, Java auf dem LEGO MINDSTORMS NXT Controller. Doch gibt es Java in noch kleinerer Variante? Ja, solch eine Lösung gibt es – und sie heißt HaikuVM.

von Bob Genom

HaikuVM ist eine Java Virtual Machine für Mikrocontroller und wurde zuerst auf AVRs – dabei handelt es sich um die 8-Bit-Mikrocontroller-Familie von Atmel – und der Arduino-Plattform entwickelt. Es ist Open Source, wird auf SourceForge gehostet und zielt insbesondere auf Mikrocontroller mit wenig Ressourcen an Flash- und RAM-Speicher ab. HaikuVM steht in der Tradition und wurde inspiriert von vielen anderen Java VMs für Mikrocontroller, beispielsweise leJOS (mit TinyVM), TakaTuka, Toba, bajos, Darjeeling, JC, SlimVM und NanoVM.

Insbesondere NanoVM zeigt, dass es möglich ist, eine Java VM auch für Mikrocontroller mit sehr wenigen Ressourcen zu implementieren. So laufen NanoVM und HaikuVM auch auf einem ATmega8 Mikrocontroller mit nur 8K Flash und 1K RAM – das ist selbst im Vergleich zum LEGO MINDSTORMS NXT Controller eher winzig. Im Unterschied zu NanoVM implementiert HaikuVM aber ausnahmslos alle Bytecode-Instruktionen von Java in der Version 6. Spannende Features wie Threads (inklusive *synchronized*), Exceptions und Garbage Collection werden unterstützt.

Haikufy your World!

Wie funktioniert nun HaikuVM? Es analysiert den Java-Bytecode und übersetzt diesen in C-Datenstrukturen. Diese werden zusammen mit der Java VM (HaikuVM selbst ist in C geschrieben) durch einen C-Crosscompiler (hier: avr-gcc) für die Zielplattform kompiliert. Dieser Prozess wird „Haikufying" genannt und ist als Framework in Java implementiert (**Abb. 1**).

Um gleich einem naheliegenden Missverständnis vorzubeugen: Es wird hier kein Java-Bytecode (also *class*-Dateien) direkt auf den Mikrocontroller geladen und dort dann von einer vorinstallierten Java VM (hier: HaikuVM) interpretiert. Denn eine vollständige Java VM, eventuell noch inklusive einer vollständigen Laufzeitumgebung, ist viel zu groß für einen kleinen Mikrocontroller.

Die Kunst, also das Haikufying, besteht darin, im Vorfeld die Java-Anwendung zu analysieren und danach die Java VM so zu reduzieren, dass sie nur das enthält, was von der Anwendung auch wirklich gebraucht wird. Wird beispielsweise kein *double*- bzw. *float*-Typ gebraucht – diese Datentypen nutzen Programmierer von Mikrocontrollern eher ungern –, dann wird die Java VM von Haus aus ohne die Bytecode-Implementierungen für *double*- bzw. *float*-Operationen erzeugt. Wenn kein Threading genutzt wird, entfallen automatisch Implementierungen für den entsprechenden Bytecode usw. Aber auch auf Methoden- und Klassenebene wird alles eingespart, was nicht aufgerufen bzw. benötigt wird. Ich war überrascht zu sehen, wie viel Komplexität selbst bei realistischen Anwendungen wegfällt, wodurch die benötigte Java VM inklusive Anwendung auf ein Minimum schrumpft. Nimmt man hinzu, dass Bytecode nur ein Viertel des Platzes von kompiliertem C-Code benötigt, hat man plötzlich Platz für deutlich größere Programme als mit C alleine möglich gewesen wäre.

Der Zugriff auf die Mikrocontroller-Hardware (Teil 1)

HaikuVM wurde entwickelt, um direkt und hardwarenah programmieren zu können. „Direkt" bedeutet hier ganz ohne die Verwendung zusätzlicher Infrastruktur (z. B. Unix-Gerätetreiber) oder eines API, das den direkten Hardwarezugriff hinter Funktionen verbirgt, bzw. des Java Native Interface. Natürlich geht das nicht ohne Kenntnis der Funktionsweise der I/O-Register der konkreten Hardware. Man muss sich also die Mühe machen, das Manual zum eingesetzten Mikrocontroller durchzulesen.

In diesem Artikel möchte ich das einfache Beispiel (Blinky) der Ansteuerung einer LED bemühen, um die Verbindung zur Hardware zu beschreiben. Die LED ist am ATmega328p und dort am Pin PB5/SCK angeschlossen. Dieser lässt sich über das Bit 5 von Register B kontrollieren. In *main* von Listing 1 wird zunächst der Pin als Output konfiguriert. In der anschließenden Endlosschleife wird dieser Ausgang im Sekundentakt alternierend auf „High" und „Low" gesetzt, wodurch die LED blinkt.

Beim ATmega328p kontrolliert man mittels des 8-Bit-Registers DDRB die Richtung der Pins von Port B. Mit

Listing 1

```
package de.javamagazin;

import static haiku.avr.AVRConstants.*;

public class Blinky {
  // LED is connected with port B and Bit 5.
  // give it a name:
  private static final int LED = 1 << 5;

  public static void main(String[] args) throws InterruptedException {
    // initialize the digital pin as an output.
    DDRB |= LED;
    // this loop runs forever:
    while (true) {
      PORTB |= LED; // turn the LED on (HIGH is the voltage level)
      Thread.sleep(1000); // wait for a second
      PORTB &= ~LED; // turn the LED off by making the voltage LOW
      Thread.sleep(1000); // wait for a second
    }
  }
}
```

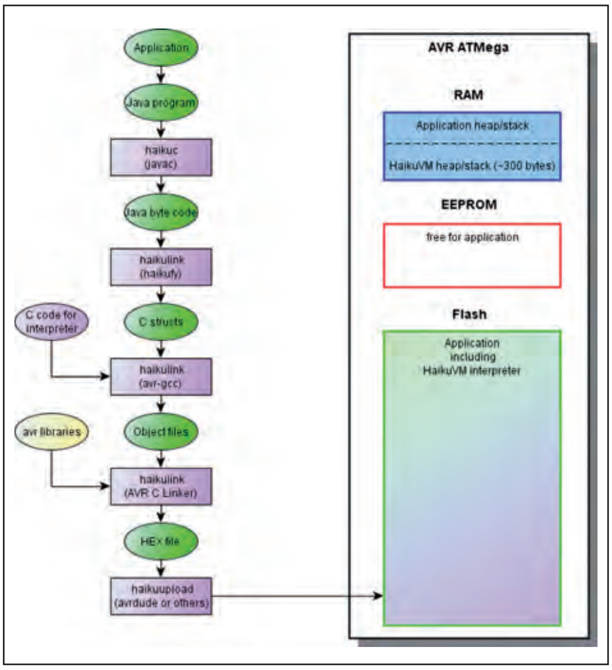


Abb. 1: Haikufying

dem 8-Bit-Register PORTB lässt sich der Zustand dieser Pins kontrollieren. Es bleibt die Frage, woher HaikuVM die Variablen *DDRB* und *PORTB* kennt. Einfache Antwort: Diese sind in *haiku.avr.AVRConstants* definiert. Aber wie werden sie dort mit den ATmega328p-Registern verknüpft? Um das zu demonstrieren, nehmen wir den Import von *haiku.avr.AVRConstants* heraus und definieren die Variablen direkt (Listing 2).

Listing 2

```
package de.javamagazin;

public class Blinky {
  // LED is connected with port B and Bit 5.
  // give it a name:
  private static final int LED = 1 << 5;

  @NativeCVariable8
  private static volatile int DDRB;
  @NativeCVariable8
  private static volatile int PORTB;

  public static void main(String[] args) throws InterruptedException {
    // initialize the digital pin as an output.
    DDRB |= LED;
    // this loop runs forever:
    while (true) {
      PORTB |= LED; // turn the LED on (HIGH is the voltage level)
      Thread.sleep(1000); // wait for a second
      PORTB &= ~LED; // turn the LED off by making the voltage LOW
      Thread.sleep(1000); // wait for a second
    }
  }
}
```

Listing 2 zeigt, dass das verwendete Zaubermittel Annotationen sind. Die hier verwendete *@NativeCVariable8* informiert HaikuVM, dass die folgende Java-Variable direkt auf eine gleichnamige und existierende 8-Bit-C-Variable abgebildet werden soll (ganz unabhängig von ihrem Java-Typ). Voilà – das reicht, um die Verbindung zwischen Java und der Hardware herzustellen. Die Frage, welche Variablen und Register der AVR-C-Compiler für die jeweils spezifische Hardware (hier ATmega328p) anbietet, beantwortet ein Blick ins entsprechende AVR-Manual.

Und es wäre nicht Java, wenn man nicht auch zwei LEDs mit unterschiedlicher Frequenz durch die Verwendung von Threads blinken lassen könnte (Listing 3). Schon hier fängt die Schwierigkeit für C-/C++-Hobbyisten an: Sie müssen sich auf die Suche nach einer Threading Library für ihren Mikrocontroller machen oder, wenn es eine solche nicht gibt, das Problem eben anders lösen.

Der Zugriff auf die Mikrocontroller-Hardware (Teil 2)

Bei Elektronikhobbyisten und Freunden von Mikrocontrollern erfreut sich gerade die Arduino-Plattform einer großen Beliebtheit. Die Arduino-Hardware programmiert man in C++. In Sachen Programmierung geht man dort einen anderen Weg: Es wird versucht, von einer allzu hardwarenahen Programmierung durch ein API zu abstrahieren. Dazu gehört auch, dass die Pins eines Arduinos durchnummeriert werden und nur über diese

Listing 3

```
package de.javamagazin;

import static haiku.arduino.api.Arduino.*;

public class BlinkWithThread {
  // LED1 is connected with PORTB and Bit 5.
  // LED2 is connected with PORTB and Bit 4.
  // give them a name:
  private static final int LED1 = 1 << 5;
  private static final int LED2 = 1 << 4;

  private static void delay(int ms) {
    try {
      Thread.sleep(ms);
    } catch (InterruptedException e) {
    }
  };

  public static void main(String[] args) throws InterruptedException {
    // initialize the digital pins as an output.
    DDRB |= (LED1|LED2);

    new Thread("for LED2") {
      public void run() {
        // this loop runs forever:
        while (true) {
          PORTB |= LED2; // turn the LED2
          delay(500); // wait for 500 milliseconds
          PORTB &= ~LED2; // turn the LED2 off
          delay(500); // wait for a second
        }
      }
    }.start();

    // this loop runs forever:
    while (true) {
      PORTB |= LED1; // turn the LED1 on (HIGH is the voltage level)
      delay(1000); // wait for a second
      PORTB &= ~LED1; // turn the LED1 off by making the voltage LOW
      delay(1000); // wait for a second
    }
  }
}
```

Listing 4

```
package de.javamagazin;

import static haiku.arduino.api.Arduino.*;

public class BlinkArduino {
  static byte ledPin = 13; // LED connected to digital pin 13

  public static void main(String[] args) {
    init();
    pinMode(ledPin, OUTPUT); // sets the digital pin as output
    while(true) {
      digitalWrite(ledPin, HIGH); // sets the LED on
      delay(1000); // waits for a second
      digitalWrite(ledPin, LOW); // sets the LED off
      delay(1000); // waits for a second
    }
  }
}
```

Listing 5

```
package de.javamagazin;

import static haiku.arduino.api.Arduino.*;

public class BlinkProcessing {
  static byte ledPin = 13; // LED connected to digital pin 13

  public static void setup() {
    pinMode(ledPin, OUTPUT); // sets the digital pin as output
  }

  public static void loop() // run over and over again
  {
    digitalWrite(ledPin, HIGH); // sets the LED on
    delay(1000); // waits for a second
    digitalWrite(ledPin, LOW); // sets the LED off
    delay(1000); // waits for a second
  }
}
```

Nummer auf einen Pin zugegriffen werden kann. Das API liegt jeweils als Library für viele AVR- aber auch ARM-Mikrocontroller vor, die zur Arduino-Plattform zählen. Diese Bibliotheken lassen sich einfach in HaikuVM einbinden, denn sie stehen durch die Klasse *haiku.arduino.api.Arduino* zur Verfügung und lassen sich so in der Java-Anwendung nutzen. Listing 4 zeigt, wie dies am Beispiel von Blinky aussieht.

Der MicroKernel als HaikuVMs wichtigste Innovation

Genau genommen programmiert man Arduinos nicht in C++, sondern in Processing. Processing strukturiert den Programmaufbau in folgende Phasen vor:

- Initialisierung der Hardware (*init*)
- Initialisierung der Anwendung (*setup*)
- Die eigentliche Anwendung (*loop*)

Das heißt, dass ein Processing-Programm keine *main*-Funktion kennt, und im Fall von *BlinkArduino* hätte man bezüglich des Programmaufbaus eher einen Quellcode im Stil von Listing 5 erwartet.

Es fällt sofort auf, dass der Aufruf von *init* aus Listing 4 fehlt und *loop* nicht wirklich eine Schleife ist (sondern nur so heißt). Diese Lücke kann in HaikuVM durch die Verwendung eines MicroKernels – die vielleicht wichtigste Innovation von HaikuVM – geschlossen werden. Ein MicroKernel ist selbst in Java geschrieben, dient im Wesentlichen der Initialisierung und Abstraktion der Hardware und ruft letztlich die Anwendung auf. Dies erreicht man, indem man beim Haikufying zusätzlich (und optional) einen MicroKernel angibt. Dieser wird mit der Anwendung verwoben, und zwar derart, dass der Mikrocontroller zur Laufzeit mit dem MicroKernel gebootet wird (und nicht mit der eigentlichen Anwendung). Listing 6 verdeutlicht das mit einem passenden MicroKernel für *BlinkProcessing*.

Ein MicroKernel ist zunächst eine beliebige Klasse, die das Interface *haiku.vm.MicroKernel* implementiert. Sie sollte eine *main*-Funktion aufweisen, denn nur mit dieser startet Java/HaikuVM zur Laufzeit. Wie in Listing 6 zu sehen, wird zunächst durch den Aufruf von *init* die Hardware initialisiert. Dann erfolgt der Aufruf von *haiku.vm.HaikuMagic.setup*. Hierin steckt, wie der Name schon nahe legt, das kleine bisschen Magie, über das der MicroKernel mit der eigentlichen Anwendung verwoben wird. Während des Haikufyings, also offline, wird im Anwendungscode nach einer Methode *setup* gesucht, und diese wird zur Laufzeit an dieser Stelle aufgerufen, in unserem Fall also *de.javamagazin.BlinkProcessing.setup*. Entsprechendes geschieht mit *haiku.vm.HaikuMagic.loop*. Dieses wird (offline) durch den Aufruf von *de.javamagazin.BlinkProcessing.loop* substituiert. Dieses Verweben könnte man zwar auch zur Laufzeit mittels Reflection machen, allerdings wird Reflection von HaikuVM nicht unterstützt und würde überdies das Programm so aufblähen, dass die Chance schwindet, es in einem kleinen Mikrocontroller unterzubringen.

Listing 6

```
package de.javamagazin;

public class MicroKernelProcessing implements haiku.vm.MicroKernel {

  public static void main(String[] args) {
    haiku.arduino.api.Arduino.init();
    haiku.vm.HaikuMagic.setup();
    while (true) {
      haiku.vm.HaikuMagic.loop();
    }
  }
}
```

Runtime Library und Portierbarkeit

Innerhalb des HaikuVM-Projekts wurde auf die Entwicklung einer proprietären Runtime verzichtet und stattdessen die Runtime von leJOS geborgt, die mit dem Download von HaikuVM zur Verfügung steht. Diese Library ist relativ vollständig und überdies so aufgebaut, dass viele Funktionen direkt in Java implementiert sind und somit nicht mittels JNI auf C-Funktionen bzw. Libraries ausgewichen werden muss. So ist beispielsweise *java.lang.Math* komplett in Java implementiert. Damit stehen wichtige Funktionen wie *sin*, *cos*, *sqrt* etc. auch dann zur Verfügung, wenn für die entsprechende Zielhardware eine Mathematik-Library (*libm*) fehlt.

Für viele Nutzer von Mikrocontrollern ist die Verwendung von 32-Bit *int* oder 64-Bit *double* ungewöhnlich und meist der Overkill für die Mikrocontroller-Anwendung. Sie wird es freuen zu lesen, dass HaikuVM zwar den 32-/64-Bit-Modus der Java-Standardtypen beherrscht, aber daneben auch noch den 16-/32-Bit-Mode als Standard anbietet. Hier sind die Typen *double* bzw. *long* 32 Bit lang und *float* bzw. *int* nur 16 Bit lang. Das schont die Ressourcen der kleinen Mikrocontroller.

Für den Fall, dass dies nicht reicht, ist die Java VM von HaikuVM vollständig in C geschrieben (zero assembler) und erleichtert damit die Portierbarkeit auf andere Plattformen. HaikuVM ist zwar eine ausgereifte Java VM für Mikrocontroller; trotzdem suche ich noch Mitstreiter für den Ausbau der Arduino-Unterstützung und Hilfe beim Einbau von Java-Remote-Debugging. In diesem Sinne: Meldet euch doch bei mir [1]!

Bob Genom wurde nie geboren und lebt in einer Matrix, in der die Wirklichkeit keine Rolle zu spielen scheint. Hauptberuflich spielt er gerne mit weißen Kaninchen. Seine Freizeit verbringt er mit der Zucht von Programmen für Mikrocontroller.

Links & Literatur

[1] http://haiku-vm.sourceforge.net/

[2] http://arduino.cc/

Ein Client für das Internet of Things für die Hosentasche

MyMQTT

Maschine-zu-Maschine-Kommunikation (M2M) und das Internet der Dinge spielen eine immer zentralere Rolle in der mobilen Welt. instant:solutions hat im April eine Android-App namens *MyMQTT* veröffentlicht, um mithilfe des De-facto-Standardprotokolls für das Internet der Dinge, MQTT, schnell und spielerisch in das Thema einzusteigen. Grund genug, die App genauer unter die Lupe zu nehmen.

von Dominik Obermaier

Das schlanke Protokoll MQTT ist durch seine hochskalierbare Publish/Subscribe-Architektur und den minimalen Protokoll-Overhead besonders für den Einsatz zur Kommunikation mit mobilen Geräten geeignet. Das Grundprinzip ist, dass sich verschiedenste Geräte bei einem zentralen Server (Message Broker) anmelden und dort Topics, also Themen, abonnieren (Subscribe). Sie erhalten dann sofort die Nachrichten, die andere Geräte zu diesem Topic senden (Publish). Die komplette Kommunikation findet also rein über Themen statt, und die kommunizierenden Geräte kennen sich in der Regel nicht direkt.

MyMQTT

Die kostenlose App MyMQTT ist zum Zeitpunkt des Schreibens in Version 1.0 im Google Play Store verfügbar [1] und erfordert mindestens Android 2.1 oder höher. Grundsätzlich ist MyMQTT ein Tool, um MQTT-Nachrichten an einen Broker zu senden oder dort verschiedene Topics zu abonnieren und die Nachrichten dann zu empfangen. Um eines vorwegzunehmen: Diese Dinge macht MyMQTT wirklich sehr gut.

Die App hat ein sehr schickes User Interface und wirkt sehr aufgeräumt und klar strukturiert. Die Navigation in der App funktioniert ausschließlich über das ausklappbare Menü, das so genannte *Slide-out Menu*). Über dieses Menü erreicht man alle Menüpunkte jederzeit und kann so sehr schnell zwischen der Publish- und der Subscribe-Funktionalität wechseln.

Verbindungsaufbau zu einem MQTT Broker

Bevor mit MyMQTT losgelegt werden kann, muss zunächst ein öffentlicher MQTT Broker eingetragen werden. Im SETTINGS-Bereich der App hat man dazu die Möglichkeit. Im Normalfall muss einfach nur der URL zu einem MQTT Broker eingetragen werden (z. B. broker.mqttdashboard.com, um einen öffentlichen HiveMQ MQTT Broker [2] zu benutzen). Optional können noch der Port und eine Username-Passwort-Kombination eingetragen werden. Wenn die benötigten Informationen für den Verbindungsaufbau gespeichert wurden, verbindet sich die App automatisch mit dem Broker, und es kann sofort losgelegt werden. Im Fehlerfall macht die App automatisch einen erneuten Verbindungsversuch. Oben im ausklappbaren Menü sieht man jederzeit, mit welchem Broker man verbunden ist. Wenn keine Verbindung zum Broker besteht, verwandelt sich die Informationsbox in einen Button, der einen Verbindungsversuch bei einem Klick auslöst.

Dashboard und Subscriptions

Das Herzstück der App dürfte wohl das Dashboard sein. Hier werden alle eingehenden Nachrichten chronologisch aufgelistet. Sehr schön ist, dass jedes Topic seine eigene Farbe im Dashboard bekommt und man so bei sehr vielen Subscriptions sofort erkennen kann, welche Nachricht auf welchem Topic eingegangen ist. Selbst sehr große Mengen an eingegangenen Messages verkraftet die App gut. Alle eingegangenen Nachrichten können bei Bedarf mit dem *Clear List*-Button gelöscht werden, worauf das Dashboard wieder mit neuen eingehenden Nachrichten gefüllt werden kann. Mithilfe des Menüpunkts SUBSCRIBE können die verschiedenen Topics abonniert werden. Sehr schön ist, dass, wenn noch kein Abonnement vorhanden ist, ein Beispiel angezeigt wird, wie ein Subscribe aussehen könnte. Toll ist auch, dass bei den Subscriptions alle MQTT Wildcards unterstützt werden.

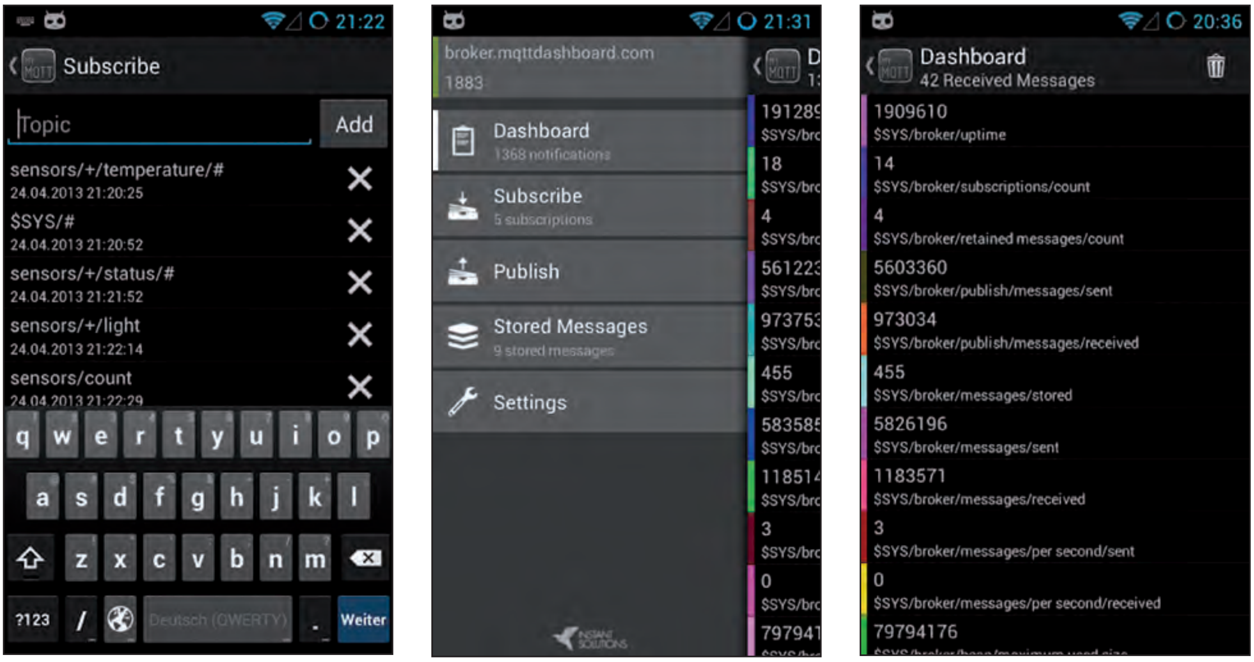


Abb. 1 bis 3: Screenshots von der MyMQTT-App (Quelle: [1])

Publish

Über den Menüpunkt Publish gelangt man zu der Ansicht, mit der man Nachrichten an den MQTT Broker senden kann. Die App bietet jeweils ein Textfeld für das Topic und ein Textfeld für die Payload, also die eigentliche Nachricht, an. Nützlich wäre hier noch die Möglichkeit, Dateien als Nachricht statt als Text zu senden, was aber in der Praxis meist nicht relevant ist. Negativ fällt auf, dass das Quality of Service Level – MQTT definiert drei verschiedene – nicht eingestellt werden kann und auf dem niedrigsten Level gesendet wird. Glücklicherweise ist dieses essenzielle Feature schon auf der Roadmap der Entwickler und wird in den nächsten Versionen der App enthalten sein. Positiv fällt hingegen auf, dass die Anzahl, wie oft man die Nachricht an den Broker senden möchte, konfiguriert werden kann.

Stored Messages

Ein sehr nützliches und herausstechendes Feature ist, dass Publish-Nachrichten gespeichert werden und jederzeit mit einem Klick erneut gesendet werden können. Das ist besonders praktisch, wenn man die MyMQTT-App verwendet, um andere Apps oder Integrationen zu testen. Um das Feature zu nutzen, kann man in der Publish-Ansicht auf Save Message klicken, und die Nachricht kann bequem im Menüpunkt Stored Messages erneut mit einem Klick gesendet werden.

Kommende Features

Wie schon erwähnt, ist MyMQTT momentan in Version 1.0 erhältlich. Es gibt noch kleinere Instabilitäten, wenn man z. B. den Broker im laufenden Betrieb wechselt oder wenn der Broker Fehlercodes schickt. Diese fallen aber in der täglichen Benutzung der App nicht ins Gewicht. Die bisher angekündigten zusätzlichen Features für spätere Versionen klingen vielversprechend. Dazu gehören:

- Unterstützung für SSL
- Android Notifications, wenn eine neue Nachricht eintrifft
- Export und Import von gespeicherten Nachrichten
- Unterstützung für MQTT Quality of Service Levels

Fazit

MyMQTT macht einen sehr guten Eindruck, und die Oberfläche kann sich wirklich sehen lassen (Screenshots **Abb. 1** bis 3). Funktional deckt die App alles ab, was ein Einsteiger in das Thema MQTT braucht, und sie ist eine echte Bereicherung für Entwickler mit Projekten, in denen MQTT verwendet wird. Somit kann für die kostenlose MyMQTT-App eine uneingeschränkte Empfehlung für Entwickler und Personen, die sich für MQTT interessieren, ausgesprochen werden.

Dominik Obermaier entwickelt bei der dc-square GmbH M2M-Lösungen und ist verantwortlich für die Architektur von Webapplikationen auf Basis von Java EE 6.

@dobermai

Links & Literatur

[1] https://play.google.com/store/apps/details?id=at.tripwire.mqtt.client

[2] http://www.hivemq.com/

Die beliebtesten Miniaturcomputer im Überblick

Sieben Zwerge

Sie liebäugeln mit einem dieser angesagten Liliputaner-PCs, aus denen Bastler und Kreativhacker, aber auch Enterprise-Softwareentwickler die raffiniertesten Miniaturinfrastrukturen bauen, LED-Lampen leuchten oder Temperaturen anzeigen lassen? Sie suchen die geeignete Einplatine für ein günstiges Testsystem, möchten Ihr Gewächshaus bewässern oder Ihren Sprösslingen das Programmieren beibringen? Dann sind Sie hier richtig. Die folgenden Seiten präsentieren sieben der derzeit populärsten Open-Source-Computer für die Hosentasche.

von Diana Kupfer

Ethernut: Netz trifft Hardware

Am Anfang war das Netz: Das älteste der hier vorgestellten Boards aus der deutschen Elektronikschmiede egnite erwuchs aus dem Vorhaben, TCP/IP auf 8-Bit-Mikrocontrollern zu implementieren. Der britische Softwareentwickler Dave Hudson schrieb Ende der 1990er Jahre ein entsprechendes Open-Source-Betriebssystem mit dem Namen „Liquorice". Das Hardwarependant, ein AVR-Mikrocontroller-Board auf Basis des ATmega103 und des Ethernet-Controllers RTL-8019AS, entwickelte Harald Kipp. Der nannte das Board „Ethernut", übernahm wenig später auch das OS-Projekt von Hudson, taufte dieses in „Nut/OS" um und registrierte diese Hard- und Softwaresymbiose. Das Ethernut-Board zeichnet sich durch ein großes Spektrum an Schnittstellen aus. Eine Besonderheit ist zudem die mitgelieferte Software Nut/OS, ein sehr schlichtes eingebettetes Realtime-Multitasking-Betriebssystem (RTOS) unter BSD-Lizenz,

Bild: egnite.de

Ethernut 5

CPU: AT91SAM9XE (ARM9-basiert, 200 MHz) | NOR-Flash-Speicher: 512 KB | NAND-Flash: 1 GB | Statischer RAM: 32 KB | SDRAM: 128 MB | Serieller Flash-Speicher: 4 MB | Schnittstellen: Ethernet, 2 x USB, RS-232, Erweiterungsstecker mit 15 digitalen Ein- und Ausgabekanälen, 2 analoge Eingänge, 20-pol. JTAG-Stecker, SD-Kartenslot, Bus für externe Speichererweiterungen u. a., Stromversorgung über USB, Ethernet oder externes 5-24-V-Netzteil | 3 Status-LEDs | Echtzeituhr | Abmessungen: 98 x 78 x 17 mm | Gewicht: 75 g | Preis: ca. 220 Euro

Bild: arduino.cc

Arduino Uno R3

Mikrocontroller: ATmega328 | Betriebsspannung: 5 V | Digitale Ein- und Ausgabekanäle: 14 (6 davon mit PWM-Ausgang) | Eingangsspannung (empfohlen) 7–12 V | Analoge Eingabekanäle: 6 | Flash-Speicher: 32 KB (ATmega328), davon 0,5 KB für Bootloader | SRAM: 2 KB (ATmega328) | EEPROM: 1 KB (ATmega328) | Taktfrequenz: 16 MHz | Schnittstelle: USB | Abmessungen: 68,6 x 53,3 mm | Gewicht: 80 g | Preis: ca. 20 Euro (Arduino Store) das leicht auf andere Hardwareplattformen portiert werden kann. Außerdem glänzt es durch modulares Design, Event-Queues, dynamisches Speichermanagement, Filesystem-Unterstützung, Stream-I/O-Funktionen und kooperatives Multi-Threading. Neben Nut/OS wird auch Linux unterstützt. Anwendungen werden in der Sprache C geschrieben. Implementiert wird in diesem Projekt ein eigener Netzwerk-Stack, der den Namen Nut/Net trägt. Das aktuellste Ethernut-Board trägt die Versionsnummer 5.

Arduino: Kunst trifft Technik

Wo Kunst und Technik zusammentreffen, entstehen kreative Gadgets: Seinen Anfang nahm das Arduino-Projekt 2005 in der norditalienischen Stadt Ivrea. Massimo Banzi, damals Professor am Institut für Design, suchte nach einer Möglichkeit, wenig technikaffinen Studenten den Umgang mit elektronischen Hilfsmitteln zu erleichtern. Gemeinsam mit dem Ingenieur David Cuartielles entwickelte er die Idee, einen Mikrocontroller zu entwerfen, der sich im Handumdrehen in Designprojekte einbetten lässt. Das Kern-Entwicklerteam komplettieren Tom Igoe und David Mellis. Die Hardware wurde beim Elektronikhersteller Gianluca Martino in Auftrag gegeben. So entstand Arduino, eine offene Soft- und Hardwareplattform. An Bord ist eine IDE auf Grundlage von Processing und Wiring – beides wiederum auf Java basierend. Inzwischen ist ein ganzes Ökosystem um den kleinen Italiener entstanden: Nicht nur hat die Arduino-Familie selbst reichlich Nachwuchs bekommen. Auch zahlreiche Klone und Arduino-kompatible kleinere und passgenauere Varianten wie der TinyDuino setzen die Vision Banzis und seiner Partner in diversen Anwendungsszenarien in die Tat um. Nicht nur die produktive Community und ihr ansteckender Idealismus, sondern auch die Erweiterbarkeit der Plattform begründet ihren Erfolg. So können durch aufsteckbare *Shields* je nach Bedarf neue Funktionalitäten hinzugefügt werden. Derzeit gibt es knapp 300 Shields von über 100 Herstellern. Benannt wurde das Arduino-Projekt übrigens nach einer Bar, deren Name wiederum auf den mittelalterlichen Herrscher Arduin von Ivrea zurückgeht.

Vergleichbar: Digispark, ein äußerst günstiges, winziges Mini-USB-Board.

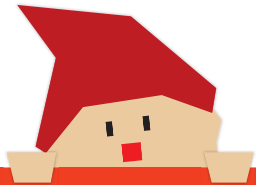



Bild: arduino.cc

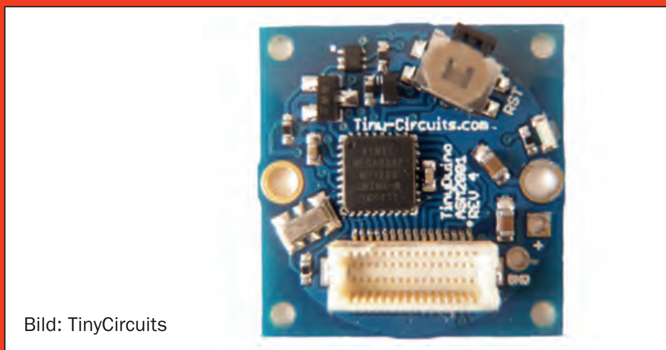


Bild: TinyCircuits

Arduino Due

Erster ARM-basierter Arduino | Mikrocontroller: AT91SAM3X8E | Betriebsspannung: 3,3 V | Eingangsspannung (empfohlen): 7–12 V | Digitale Ein- und Ausgabekanäle: 54 (12 mit PWM-Ausgang) | Analoge Eingabekanäle: 12 | Analoge Ausgabekanäle: 2 (DAU) | Flash-Speicher: 512 KB, komplett für Useranwendungen verfügbar | SRAM: 96 KB (zwei Module: 64 KB und 32 KB) | Taktfrequenz: 84 MHz | Schnittstelle: USB | Abmessungen: 101,6 x 53,3 mm | Gewicht: 50 g | Preis: ca. 39 Euro (Arduino Store)

TinyDuino (Hersteller: TinyCircuits)

Arduino- und LilyPad-kompatibel | Erweiterbar mit TinyShield Boards | Optionaler Akkuanschluss für CR1612-CR1632-Knopfzellen | Mikrocontroller: Atmel ATmega328P | Flash-Speicher: 32 KB | 2 KB RAM | 1 KB EEPROM | Taktfrequenz: 8 MHz | Betriebsspannung: 1,8–5,5 V | 20 I/Os (14 Digital, 6 Analog/Digital I/O) | Arduino Bootloader ist vorprogrammiert (benötigt 0,5 KB Flash-Speicher) | Abmessungen: quadratische Variante: 20 x 20 mm, kreisförmige Variante: 20 mm Durchmesser | Höhe: 0,61 mm | Preis: 19,95 US-Dollar

BeagleBoard: Ångström Linux trifft Cloud9

2008 erblickte das BeagleBoard aus dem Hause Texas Instruments unter Zusammenarbeit mit Digi-Key das Licht der Elektronikwelt. Das Hardwaredesign ist, wie bei Arduino und Co., Open Source. Einerseits ist die Einplatine ein optimaler Schaukasten für die ARM-basierte Prozessorarchitektur Open Multimedia Applications Platform (OMAP) von Texas Instruments, andererseits dient sie, wie der Raspberry Pi, pädagogischen Zwecken. Das Original-BeagleBoard ist mit einer ARM-Cortex-A8-CPU ausgestattet, auf der Linux, FreeBSD, RISC OS, Symbian und Android laufen. Mittlerweile sind drei weitere Modelle auf dem Markt: das BeagleBoard-xM, das BeagleBone und der jüngste Spross der Beagle-Familie, das BeagleBone Black. An Software unterstützt Letzteres u. a. Ångström Linux, Android und Ubuntu. Vorinstalliert ist neben Ångström Linux die Cloud9 IDE. Das Besondere am BeagleBoard und BeagleBone ist deren Schnittstellenvielfalt. *Über immerhin 65 digitale Ein- und Ausgabe-Pins und sieben analoge Eingänge ist das BeagleBone Black erweiterbar* – das ist sogar mehr, als der Arduino Due bieten kann. Allerdings ist das schwarze BeagleBone mit Preisen um die 55 Euro auch etwas teurer.

PandaBoard: Minimaldesign trifft Anspruch

Ebenfalls aus dem Hause Texas Instruments stammt das PandaBoard, mit Preisen um die 200 Euro etwas teurer als die BeagleBoard-Familie. Allerdings ist es mit einem Dual-Core-Prozessor und 1 GB Arbeitsspeicher auch etwas besser ausgestattet. Das Hardwaredesign ist wie bei den anderen Boards ebenfalls Open Source. Die neueste Version ist das PandaBoard ES, das erste auf OMAP4460-basierende Board, das im Gegenteil zu seinem Vorgänger außerdem die DSI-Spezifikation (Display Serial Interface) unterstützt. Eine Ethernet-Schnittstelle, WLAN- und Bluetooth-Unterstützung sind ebenfalls an Bord. Auf einen Flash-Speicher muss man allerdings verzichten.

Raspberry Pi: Pädagogik trifft Technik

Kreative Gadgets entstehen auch dort, wo Pädagogen und Techniker in Dialog treten – in diesem Fall in Großbritannien. Inspiriert wurde das Projekt Raspberry Pi vom Erfolg des Bildungsprojekts BBC Micro, einem 1981 von der Firma Acorn Computer hergestellten Mikrocomputer (der damalige Begriff für „Personal Computer"). Bereits 2006, ein Jahr nach der Arduino-Initialzündung, gab es erste Pläne für einen entsprechenden Atmel-ATmega644-basierten Mikrocontroller. 2009 gründete Eben Upton, damals erst 31 Jahre jung, gemeinsam mit Lehrern, Universitätsangestellten und Technikbegeisterten die Raspberry Pi Foundation. Ziel der gemeinnützigen Stiftung ist es, Kindern und Jugendlichen einen spielerischen Zugang zur Infor-

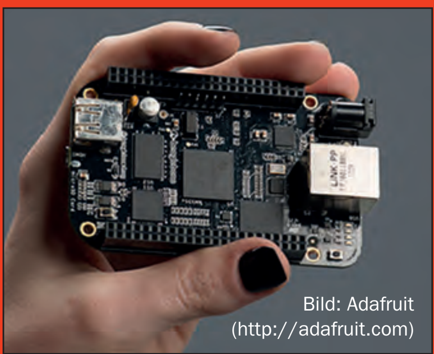


Bild: Adafruit (http://adafruit.com)

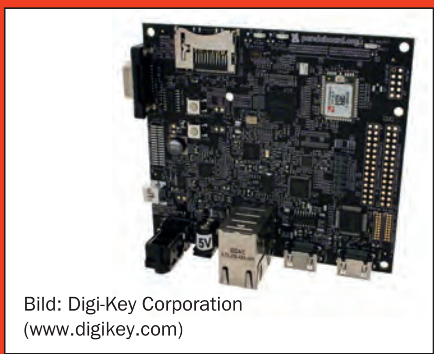


Bild: Digi-Key Corporation (www.digikey.com)

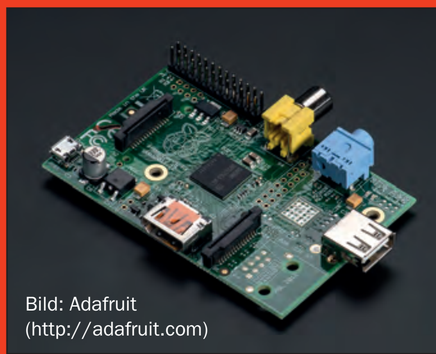


Bild: Adafruit (http://adafruit.com)

BeagleBone Black

Prozessor: TI Sitara AM335x ARM Cortex A8 (1 GHz) | SDRAM: 512 MB DDR3L | Flash-Speicher: 2 GB | GPU: PowerVR SGX530 mit 3-D-Grafikbeschleuniger | Betriebsspannung: 5 V | Schnittstellen: 65 digitale Ein- und Ausgabekanäle, USB, Micro-USB, microSD, HDMI u. a. | Konnektivität: Ethernet | Abmessungen: 86 x 53 mm | Gewicht: 40 g | Preis: ca. 55 Euro

PandaBoard ES

Prozessor: ARM Cortex-A9 MPCore (Dual-Core, 1,2 GHz) | RAM: 1 GB DDR2 | GPU: PowerVR SGX540 | Schnittstellen: 3 x USB, 1 x RS 232, Display: HDMI, DVI-D, Audio: 3,5" Ein-/Ausgang, HDMI-Ausgang | Full-HD-Multistandard-Video-Ein- und Ausgabe | Konnektivität: Ethernet, WLAN, Bluetooth | 2 konfigurierbare Status-LEDs | Abmessungen: 114,3 x 101,6 mm | Gewicht: 81,5 g | Preis: ca. 180 Euro (Vertrieb siehe: http://pandaboard.org/content/buy)

Raspberry Pi, Modell A

Ohne Ethernet-Schnittstelle, Stromverbrauch ca. 1/3 des B-Modells | Prozessor: ARM-1176JZF-S Core (700 MHz) | GPU: Broadcom VideoCore IV | 256 MB RAM | Versorgungsspannung: 5 V (über Micro-USB) | Schnittstellen: 1 x USB, HDMI-Port | 3,5 mm Audioanschluss | SD-Kartenslot | Micro-USB-Power-Port | 2 Status-LEDs | Abmessungen: 85,60 x 53,98 x 17 mm | Gewicht: 45 g | Preis: ca. 28 Euro (Vertrieb über http://www.farnell.com oder http://uk.rs-online.com/web) matik zu verschaffen. 2011 gingen die ersten Boards in die Produktion, Anfang 2012 kamen die ersten Raspberry Pis auf den Markt. Bis 22. Mai waren bereits 20 000 Exemplare verkauft. Die Markteinführung in den USA im April 2013 war ein immenser Erfolg: Binnen weniger Tage war die Einplatine restlos ausverkauft.

Wie das Arduino-Board gibt es auch den „Himbeerkuchen“ in unterschiedlichen Geschmacksrichtungen. Vielfalt ist hier allerdings weniger auf der Hardware- und mehr auf der Softwareseite zu finden. Das gängigste Betriebssystem ist derzeit Raspbian, Debian-basiert und selbstverständlich Open Source. Daneben sind eine Reihe anderer Betriebssysteme, teilweise durch Portierung, möglich bzw. werden in Kürze verfügbar sein, darunter Android, Firefox OS oder Googles Chromium OS.

Cubieboard: Einplatine trifft Massenspeicher

Mit dem Erfolg der Minicomputer wachsen auch die Ansprüche ihrer Nutzer: Auf der Wunschliste vieler Entwickler standen neben einem SATA-Anschluss für Massenspeicher auch andere Erweiterungen. Das war für Tom Cubie von der chinesischen Firma Allwinner Technology Anlass genug, mit einem Team das nach ihm benannte Cubieboard zu entwerfen. So bekamen Eichhörnchen, Beagle, Pandabären und Co. 2012 Konkurrenz durch die Goldstumpfnase – so der zoologische Name des possierlichen Äffchens, das dem Design des Cubieboard-Maskottchens Pate stand. Neben 1 GB Arbeitsspeicher besitzt das Cubieboard einen SATA-Anschluss für den Betrieb von einfachen 2,5-Zoll-Notebook-Festplatten oder SSDs, einen Ethernet- sowie einen Infrarotanschluss. Werksseitig ist bei den neueren Geräten Android Ice Cream Sandwich installiert. Linux-basierte Betriebssysteme werden ebenfalls unterstützt.

Tinkerforge: Cross-Platform trifft Bastelfreude

Noch stärker als die anderen hier vorgestellten Plattformen setzt der deutsche Hersteller Tinkerforge auf das Baukastenprinzip: Per USB steuerbare Mikrocontroller, die so genannten *Bricks*, werden per Kabel mit *Bricklets* verbunden. Das sind Module, die die Bricks um bestimmte Funktionen und Fähigkeiten bereichern, wie z. B. Sensoren. Per *Master Extensions*, vergleichbar mit den Arduino-Shields, können die *Bricks* erweitert werden. Das Besondere: Tinkerforge unterstützt alle gängigen Plattformen: Linux, Windows, Mac OS X, Android und iOS. Unterstützt werden außerdem die Programmiersprachen C/C++, C#, Delphi, Java, PHP, Python, Ruby und VB.NET. Sowohl die Hard- als auch – offensichtlich – die Software sind Open Source. 2012 gewann Tinkerforge den Preis „Produkt des Jahres“ der Computerzeitschrift CHIP.

Bild: Adafruit (http://adafruit.com)

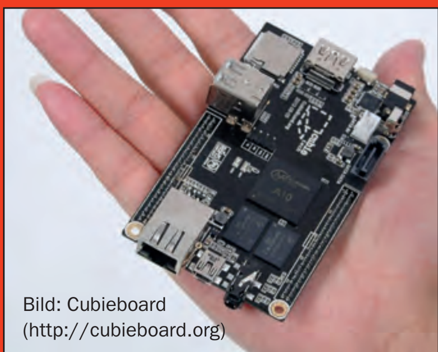


Bild: Cubieboard (http://cubieboard.org)

Bild: Tinkerforge

Raspberry Pi, Modell B

Mit Ethernet-Schnittstelle, Stromverbrauch max. 3,5 W/700 mA | Prozessor: ARM1176JZF-S Core (700 MHz) | GPU: Broadcom VideoCore IV | 512 MB RAM | Versorgungsspannung: 5 V (über Micro-USB) | Schnittstellen: 2 x USB über internen Hub, Ethernet, HDMI-Port | 3,5 mm Audioanschluss | 2 USB-Anschlüsse | SD-Kartenslot | 5 Status-LEDs | Abmessungen: 85,60 x 53,98 x 17 mm | Gewicht: 45 g | Preis: ca. 35 Euro

Cubieboard

Prozessor: ARM Cortex-A8 (1 GHz) | RAM: 1 GB DDR3 | GPU: Mali 400 | Flash-Speicher: 4 GB | Schnittstellen: 3 x USB, Steckplatz mit 96 Pins (können als I2C- oder SPI-Schnittstellen verwendet werden, als Anschluss für LC-Displays oder Sensoren gedacht) | microSD, SATA, Video: HDMI (Full-HD-Multistandard-Ausgabe), Audio: 3,5 mm, HDMI | Konnektivität: Ethernet, Infrarot, WLAN optional | Preis: ca. 60 Euro

Tinkerforge Brick

Mikrocontroller: ARM-basiert, 32 Bit | Schnittstellen: USB, 2 Bricklet-Anschlüsse | Abmessungen: 40 x 40 x 17 mm | Gewicht: je nach Modell zwischen 12 und 20 g | Preis: je nach Modell zwischen 30 und 100 Euro (Master Brick: 30 Euro)

Auf einen Blick

Arduino

Gebräuchlichste Modelle: Arduino Uno, Arduino Due, zahlreiche weitere Modelle und Klone, mehrere hundert *Shields* (Erweiterungen)

Betriebssystem: plattformübergreifend

Entwicklungsumgebung: Arduino IDE (Java, basiert auf Processing und Wiring)

Gängige Programmiersprachen: C, C++

Projektwebsite: http://arduino.cc

Raspberry Pi

Modelle: A (ohne Ethernet), B (mit Ethernet)

Betriebssysteme: zahlreiche Linux-Distributionen, z. B. Raspbian (basiert auf Debian), Android, Arch Linux ARM, R-Pi Bodhi Linux, Firefox OS, Gentoo Linux, Google Chromium OS, Raspberry Pi Fedora Remix, Slackware ARM, QtonPi (basiert auf Qt), WebOS. Außerdem: AROS, Haiku, Plan 9, RISC OS, Unix

Gängige Programmiersprachen: diverse: Python (als Hauptprogrammiersprache vorgesehen), BBC Basic, C, C++, Perl, Ada, Assembler, Java, Lua, Go, JavaScript u. a.

Projektwebsite: http://raspberrypi.org

PandaBoard

Modelle: PandaBoard, PandaBoard ES

Betriebssysteme: Linux-Distributionen, auch Android, Firefox OS, FreeBSD, Entwicklerversion von RISC OS 5, QNX Neutrino 6.5.0. Geplant: OpenBSD-Unterstützung, optimierte Android- und Ubuntu-Versionen gibt es bei der Linaro Foundation (http:// linaro.org)

Gängige Programmiersprachen: C, C++, Java

Projektwebsite: http://pandaboard.org

Ethernut

Modelle: Ethernut 1, 2, 3, 5

Betriebssystem: Nut/OS (vorinstalliert), Linux

Gängige Programmiersprachen: C

Projektwebsite: http://ethernut.de

BeagleBoard

Modelle: BeagleBoard, BeagleBoard-xM, BeagleBone, BeagleBone Black

Betriebssysteme: Linux, FreeBSD, RISC OS, Symbian, Android
BeagleBone Black: Ångström Linux und Cloud9 IDE vorinstalliert

Gängige Programmiersprachen: verschiedene, u. a. C, C++, Java, Python, Perl

Projektwebsite: http://beagleboard.org

Cubieboard

Modelle: Cubieboard

Betriebssysteme: Linux-Distributionen, vorinstalliert auf älteren Geräten: Buildroot-basiertes System, auf neueren: Android Ice Cream Sandwich

Gängige Programmiersprachen: C/C++

Projektwebsite: http://cubieboard.org

Tinkerforge

Modelle: Bricks, Bricklets und Master Extensions in diversen Ausführungen

Betriebssysteme: alle gängigen Plattformen: Linux, Windows, Mac OS X, Android und iOS

Gängige Programmiersprachen: C/C++, C#, Delphi, Java, PHP, Python, Ruby, VB.NET

Projektwebsite: http://tinkerforge.com

NEUE SERIE

©iStockphoto.com/avdeev007

Codequalität in Alt- und Wartungsprojekten

Forever young

In jedem neuen Softwareprojekt wird, oder zumindest sollte es so sein, der Qualitätssicherung von Beginn an ein hoher Stellenwert zugeschrieben. Dies umfasst nicht nur eine hohe Testabdeckung der fachlichen Logik, sondern spiegelt sich auch im Aufbau des Quellcodes selbst wider. Qualitätssicherung in Alt- und Wartungsprojekten, die bereits ein Jahrzehnt oder länger existieren, ist hingegen eine zeitintensive Mammutaufgabe, deren Einführung gut überlegt sein will. Allgemeine Probleme und erste Lösungsansätze mit Checkstyle, PMD und einem Code Formatter stehen im Fokus des ersten Teils dieser Artikelserie.

von Daniel Winter

Sobald alle Stories abgeschlossen, Features programmiert, Testfälle bestanden sind und die Software erfolgreich beim Kunden ihren Dienst verrichtet, beginnt die längste und wohl zäheste Phase im Softwareleben: die Wartung. Diese ist meist durch das Beheben von Fehlern gekennzeichnet (**Abb. 1**). Dabei kann zu Beginn noch auf das umfassende Fachwissen der ursprünglichen Entwickler zurückgegriffen werden, die nicht nur bestimmte Architekturentscheidungen getroffen haben, sondern auch die fachliche Qualifikation besitzen, auftretende Fehler schnell zu identifizieren und zu beheben. Mit der Zeit jedoch geht dieses Wissen verloren, sei es durch die Bearbeitung von zahlreichen neuen Projekten oder durch die Fluktuation bei den Mitarbeitern selbst. Neue, motivierte Entwickler kommen mit der nun schon in die Jahre gekommenen Software in Kontakt und müssen die aktuellen Fehler bearbeiten oder sogar neue Erweiterungen implementieren und testen.

Artikelserie

Teil 1: Erste Lösungsansätze mit Checkstyle, PMD und einem Code Formatter
Teil 2: Höhere Testabdeckung mit Mockito

Never change a running system

Am einfachsten und für die neuen Mitarbeiter wahrscheinlich auch am reizvollsten wäre eine komplette Neuimplementierung der Anwendung. Nicht nur könnte man so alle alten Fehler korrigieren; es bestünde auch noch die Möglichkeit, moderne Technologien einzusetzen, neue Frameworks zu evaluieren und aktuelle Entwurfsmuster (Design-Patterns) in die Architektur mit einfließen zu lassen. Insgesamt, so die Intention, würde die Anwendung schneller reagieren, besser aussehen, Erweiterungen wären einfacher zu realisieren und das verloren gegangene Fachwissen im Unternehmen wäre wiederhergestellt.

Die Realität sieht jedoch anders aus. Anwendungen, die bereits seit Jahren erfolgreich und ohne größere Probleme in Unternehmen eingesetzt werden, werden nicht einfach ausgetauscht. Dabei spielen nicht nur die auftretenden Kosten einer kompletten Neuentwicklung eine Rolle. Hinzu kommt, dass der Kunde schlicht keinen Sinn darin sieht, ein Produkt ein zweites Mal entwickeln zu lassen, das bereits existiert, tief in die eigene Systemlandschaft integriert ist und mit dem die Mitarbeiter nach langer Einarbeitungszeit vertraut sind. Getreu dem Motto „Never change a running system".

Mit Ausdauer zur Codequalität

Die Hauptaufgabe der neuen Entwickler ist es deshalb, das alte System am Leben zu erhalten. Um Fehler zu beheben, muss man die grundlegende Architektur kennenlernen, sich durch Dutzende Klassen debuggen, Kommentare studieren und vielleicht sogar mehrere hundert Seiten lange Dokumentationshandbücher wälzen. Es muss sich schlicht eingearbeitet und das Projekt kennengelernt werden. Dass man nicht die erste Person ist, die diese Wege beschreitet, wird spätestens am Quellcode deutlich. Wenn viele Entwickler zu unterschiedlichen Zeiten an einem Projekt arbeiten, ohne dass eine einheitliche Formatierung definiert ist, gibt es mindestens doppelt so viele Codestile. Ebenso können mit der Zeit verschiedene Architekturmuster bei der Erweiterung eingesetzt worden sein. Als engagierter Mitarbeiter verspürt man nun natürlich den Drang, diese Dinge zu ändern, die Codestile zu vereinheitlichen und Methoden oder ganze Klassen einem Refactoring zu unterziehen, um die unter Umständen verloren gegangene Architektur wieder hervorzuheben. Bevor aber begonnen und versucht wird, die Applikation an mehreren Fronten zu verbessern, sollte zunächst folgende Frage gestellt werden: Ist die Verbesserung der Codequalität für mein Projekt sinnvoll und überhaupt umsetzbar?

Um diese Frage zu beantworten, muss zuerst Klarheit geschaffen werden, ob die benötigte Zeit genehmigt wird. Die Codequalität in einem Altprojekt zu verbessern, ist zeit- und arbeitsintensiv, besonders dann, wenn nebenbei noch die tägliche Arbeit verrichtet werden muss. Ein ausreichend großes Zeitbudget, verbunden mit der nötigen Ausdauer des dazugehörigen Personals, bilden die Grundvoraussetzungen für die Phase der Qualitätsverbesserung – zwei Ressourcen, die eigentlich immer zu knapp bemessen sind. Ein weiterer wichtiger Punkt ist die Planungssicherheit in Bezug auf die weitere Nutzung der Anwendung beim Kunden für die nächsten Jahre. Der eigentliche Sinn,

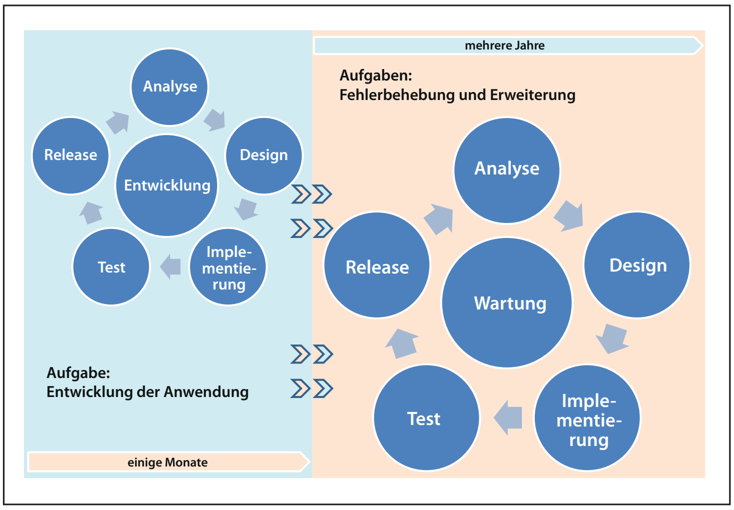


Abb. 1: Lebenszyklus der Software, unterteilt in Entwicklung und Wartung

die Codequalität zu steigern, besteht darin, die spätere Erweiterung und Fehlerbehebung zu vereinfachen und zu beschleunigen. Besitzt man nun neben dem eigenen Willen, etwas zu verbessern, auch noch ein gewisses Kontingent an Zeit und Budget, könnte die eigentliche Arbeit doch beginnen, oder?

Erfahrung, Motivation, Konsequenz und Zurückhaltung

Nein. Änderungen an einem Altprojekt, sei es durch unsachgemäße, manuell durchgeführte Codeformatierung oder durch Refactoring, bergen immer Gefahren. So kann allein das falsche Einfügen von Klammern in *if-else*-Anweisungen die gesamte Programmlogik verändern. Vielleicht mag so ein Fehler auf den ersten Blick banal erscheinen und erfahrenen Entwicklern in normalen Projekten auch nicht passieren. Aber bei Altprojekten ist es nicht unüblich, tausende Zeilen schlecht formatierter Klassen zu bearbeiten. Wenn dann auch noch gleichzeitig beim Kunden ein Fehler im Produktivsystem zu beheben ist, kann schnell der Überblick verloren gehen. Im Nachhinein die entsprechende Codezeile zu identifizieren kann sehr aufwändig und mitunter auch frustrierend sein. Wenn schon das Verändern von einzelnen Zeilen eine gewisse Gefahr birgt, muss beim Refactoring von Methoden oder ganzen Klassen erst recht vorsichtig agiert werden. Alter Code enthält zahlreiche, unkommentierte Bugfixes und Änderungen in der Fachlogik, die in den Dokumentationen und Handbüchern nicht kommentiert sind. Zusätzlich besteht die Gefahr, dass neue Fehler eingebaut, aber erst im Betrieb sichtbar werden. Fehlen dazu auch noch automatisierte Tests, ist ein Refactoring recht kompliziert. Es sollte dann nur in den wenigsten Fällen erfolgen und einen triftigen Grund voraussetzen. Weiterhin müssten bei einem umfassenden Refactoring die nötigen regressiven Integrationstests ausgearbeitet und implementiert werden, um eine vollständige Funktionsabdeckung aller Sonderfälle zu gewährleisten. Bei dem Einsatz neuer Technologien und Frameworks ist in der Planung der zusätzliche Zeitaufwand für das Lösen unbekannter Probleme mit einzubeziehen.

Bevor also die Codequalität verbessert werden kann, benötigt man die erforderliche Erfahrung mit dem Altsystem, um mögliche Auswirkungen abzuschätzen. Eine intensive Einarbeitung ist somit unerlässlich. Wurde der Prozess der Codequalitätsverbesserung erst begonnen, muss der Entwickler fokussiert auf sein Ziel sein und motiviert bleiben, da das Verändern von altem Code für die meisten nicht zu den interessantesten Aufgaben in der Softwareentwicklung gehört. Hat man schlussendlich die Phase erreicht, um alle Änderungen in das Produktivsystem zu überführen, sind eine umfassende Qualitätssicherung und die Beachtung von ITIL-Prozessen zur Vermeidung von Fehlern und Ausfällen essenziell.

Check and Style

Ein erster Schritt zur Verbesserung der Codequalität ist die Überprüfung und Anpassung der Software mit dem statischen Codeanalysetool Checkstyle [2]. Dabei setzt Checkstyle auf einen fest definierten Regelsatz, um einen einheitlichen Programmierstil sicherzustellen. Es existieren Regeln für verschiedene Komponenten, wie für:

- Annotationen
- Klassendesign
- Namenskonventionen
- Verwendung von Leerzeichen
- und viele weitere (http://checkstyle.sourceforge.net/checks.html)

Checkstyle kann dabei als Plug-in direkt in Eclipse, Hudson und Jenkins integriert werden. In Eclipse bietet sich dem Entwickler die Möglichkeit, seinen geschriebenen Quellcode sofort auf Regelverstöße zu überprüfen und zu ändern. Eine Einbindung von Checkstyle in den Continuous-Integration-Prozess und damit in die Daily Builds bietet den Vorteil, dass das gesamte Projektteam auf die Fehler aufmerksam wird und diese sofort beheben kann.

Listing 1

```
<!-- Überprüft Javadoc bei public – Klassen und Interfaces  -->
<module name="JavadocType">
  <property name="scope" value="public"/>
</module>

<!-- Überprüft Javadoc bei public – Methoden, dabei brauchen
RuntimeExceptions nicht beschrieben werden-->
<module name="JavadocMethod">
  <property name="scope" value="public"/>
  <property name="allowUndeclaredRTE" value="true"/>
</module>

<!-- Überprüft Javadoc bei private, protected und public – Variablen -->
<module name="JavadocVariable"/>
```

Die eigentliche Hauptaufgabe ist es aber nun, den entsprechenden Regelsatz für das Altprojekt zu definieren. Bei neuen Softwareprojekten werden die Regeln entweder vom Kunden vorgegeben oder im Projektteam aufgestellt. Die spätere Software wird damit auf Basis dieser Regeln programmiert. Bei einem Altprojekt ist das der falsche Ansatz. Hier liegt ein über die Jahre gewachsenes System vor, bei dem zahlreiche Entwickler bereits ihre Vorstellungen eines guten (oder schlechten) Programmierstils umgesetzt haben. Würde der Standardregelsatz oder der eines aktuellen Projekts verwendet, wären die Regelverletzungen zu zahlreich, um sie zu beheben. Eine Anwendung der Standardregeln von Checkstyle in meinem eigenen Wartungsprojekt führte zu dem Ergebnis, dass über 80 000 Verstöße vorlagen. Kein Entwickler hätte wohl die Zeit oder die Motivation, so umfangreiche Veränderungen vorzunehmen. Wahrscheinlich könnte danach niemand mehr die Korrektheit der Software gewährleisten. Wie aber ist es dann möglich, Checkstyle in einem Altprojekt einzusetzen?

Die Lösung ist, die Regeln basierend auf der Anwendung zu erstellen. Auf den ersten Blick sieht dieses Vorgehen nicht sehr vielversprechend und erfolgreich aus. Man möchte meinen, dass Regeln, die das aktuelle Altsystem widerspiegeln und dessen Fehler nicht aufzeigen, überflüssig wären. Der Vorteil wird erst durch den kombinierten Einsatz mit einem Codeformatierungstool ersichtlich. Dazu sollte die Konzentration zuerst auf die Regeln gelegt werden, die sich mit der reinen Codeformatierung befassen, wie der Überprüfung von Codeblöcken und der Verwendung von Leerzeichen. Hier kann der aktuelle Entwickler oder das Projektteam, unabhängig vom bisher eingesetzten Programmierstil, die eigenen Vorstellungen umsetzen. Eine einheitliche Klammersetzung bei Methodendefinitionen und Anweisungen erhöhen die Lesbarkeit enorm. Überhaupt ist eine konsequente Klammersetzung (auch bei einzeiligen Anweisungen) durchaus sinnvoll, da Methoden in Altprojekten oft die Eigenschaft haben, mehrere hundert Zeilen lang zu sein. Dadurch erhöht sich nicht nur die Übersichtlichkeit. Der Quellcode kann auch einen Teil seiner Komplexität verlieren, weil zum Beispiel Berechnungen für eventuelle Sonderfälle klarer abgegrenzt sind. Weitaus vorsichtiger müssen die Regeln behandelt werden, die folgende Aspekte überprüfen:

- Klassendesign
- Zugriffsmodifikatoren
- Größe von Dateien, Klassen und Methoden
- Namenskonventionen
- Java-Kommentare

Hier empfiehlt sich ein prototypischer Testlauf mit den Standardvorgaben der Regeln. Das Ergebnis ist in den meisten Fällen wohl katastrophal, da mit Sicherheit gegen fast alle Regeln wiederholt verstoßen wird. Prädestiniert sind dabei besonders die Größenüberprüfungen und die Zugriffsmodifikatoren. Es werden zahlreiche

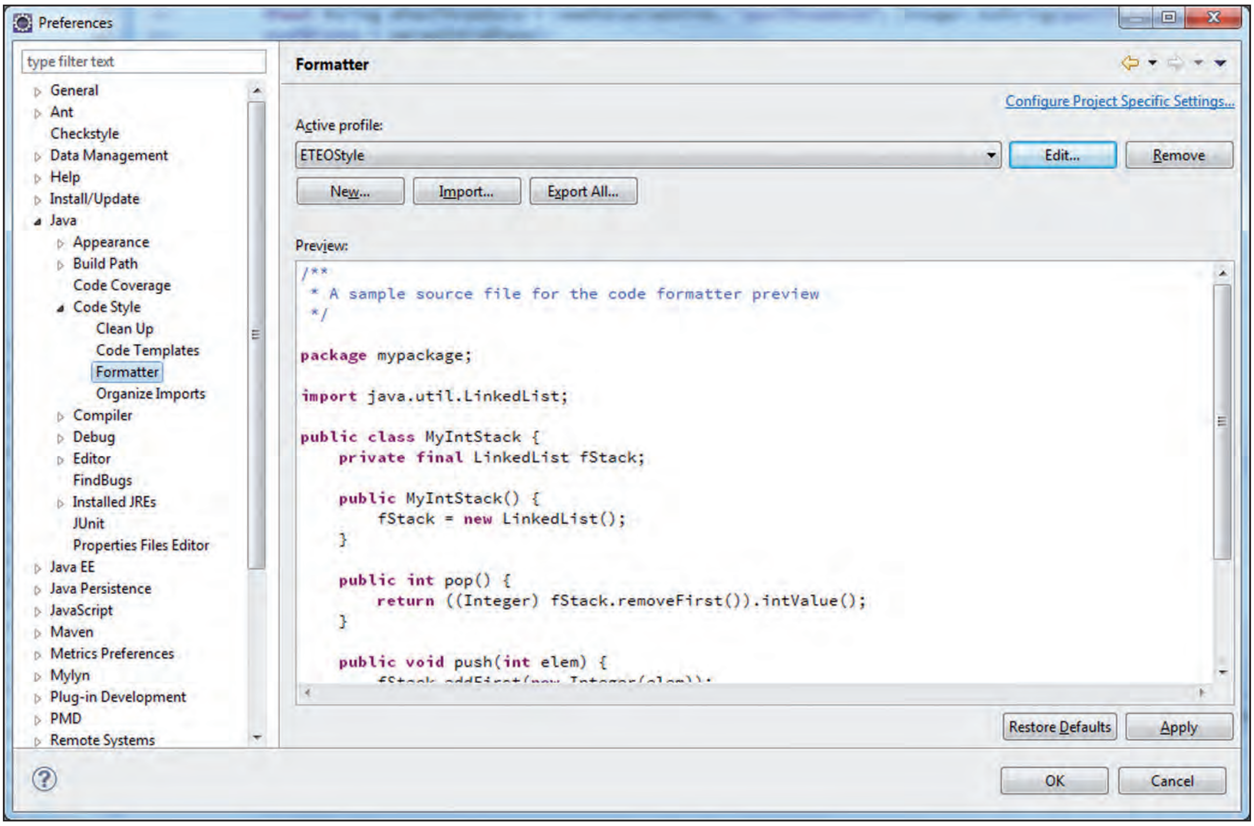


Abb. 2: Codeformatierungstool in Eclipse

Methoden und Klassen existieren, die die vorgegebene maximale Größe weit überschreiten oder Redundanz bei den Zugriffsmodifikatoren aufweisen (so sind z. B. Methoden in Interfaces automatisch *public* und *abstract*). Hier stehen einem nun drei Lösungsmöglichkeiten zur Verfügung: Zum einen kann man die Regel anpassen, etwa indem man die maximal erlaubte Methodenlänge erhöht, zum anderen den Code, indem zum Beispiel die Methode *refactored* wird. Oder man entfernt die Regel aus dem Regelsatz. Das umfassende Refactoring aufgrund von Checkstyle ist meist keine Option. Würde dieser Ansatz dauerhaft verfolgt, wäre die aufzuwendende Zeit wohl selbst vor dem eigenen Management kaum zu vertreten und sollte deshalb am besten unterbleiben. Nur wenn sich der manuell zu tätigende Aufwand in Grenzen hält und die Codequalitätsverbesserung signifikant ist, kann in einigen Fällen ein Refactoring durchgeführt werden. Die Anpassung der Regel hört sich logischer an, ist aber auch nur begrenzt umsetzbar. Eine maximale Methodenlänge von 1 000 Zeilen würde zwar den alten Code nun regelkonform erscheinen lassen, hätte aber keinen Mehrwert für neu geschriebenen Quellcode, weil dieser die Regel immer erfüllen würde. Kann also weder der Quellcode noch die Regel angepasst werden, ist das einzig sinnvolle Mittel, die Überprüfung aus dem Regelsatz zu entfernen. Warnungen, die nicht behoben werden können, demotivieren, erfüllen keinen Zweck und sind daher zu vermeiden. Weiterhin könnten Warnungen, die neuen Quellcode betreffen, aus der schieren Masse nur schwer identifiziert werden.

Der Regelsatz für die Java-Kommentare ist wohl nur in den wenigsten Altprojekten noch einsetzbar. Zu häufig werden Kommentare überhaupt nicht geschrieben oder wichtige Bugfixes im Quellcode nicht kenntlich gemacht. Sollten Sie aber dennoch das Glück haben, durchgehend Javadoc-Kommentare in Ihrem Projekt zu besitzen, führen Sie diese weiter, aktualisieren Sie sie und sichern Sie sich zusätzlich mit möglichst vielen Checkstyle-Regeln ab (Listing 1.). Der Quellcode und die dazugehörigen Kommentare sind meist der erste und manchmal auch der einzige Einstiegspunkt bei Problemen in einem Altprojekt. Javadoc und zusätzliche Kommentare helfen auch, schwer verständlichen Quellcode zu begreifen. Eine weitere wichtige Regel für ein Altprojekt ist das Verbieten der Annotation *@unused* (Listing 2). Mit diesem simplen Mechanismus kann relativ einfach sichergestellt werden, dass als unbenutzt deklarierter Code gelöscht wird. Nach einem gewissen Zeitraum wird kein Entwickler solchen Code wiederverwenden, weil die technischen und/oder fachlichen Gründe, die dazu führten, dass der Quellcode keine Verwendung mehr findet, unklar sind.

Listing 2

```
<!-- Erlaubt alle Warnungen, bis auf @unused  -->
<module name="SuppressWarnings">
  <property name="format" value="^unused$"/>
</module>
```

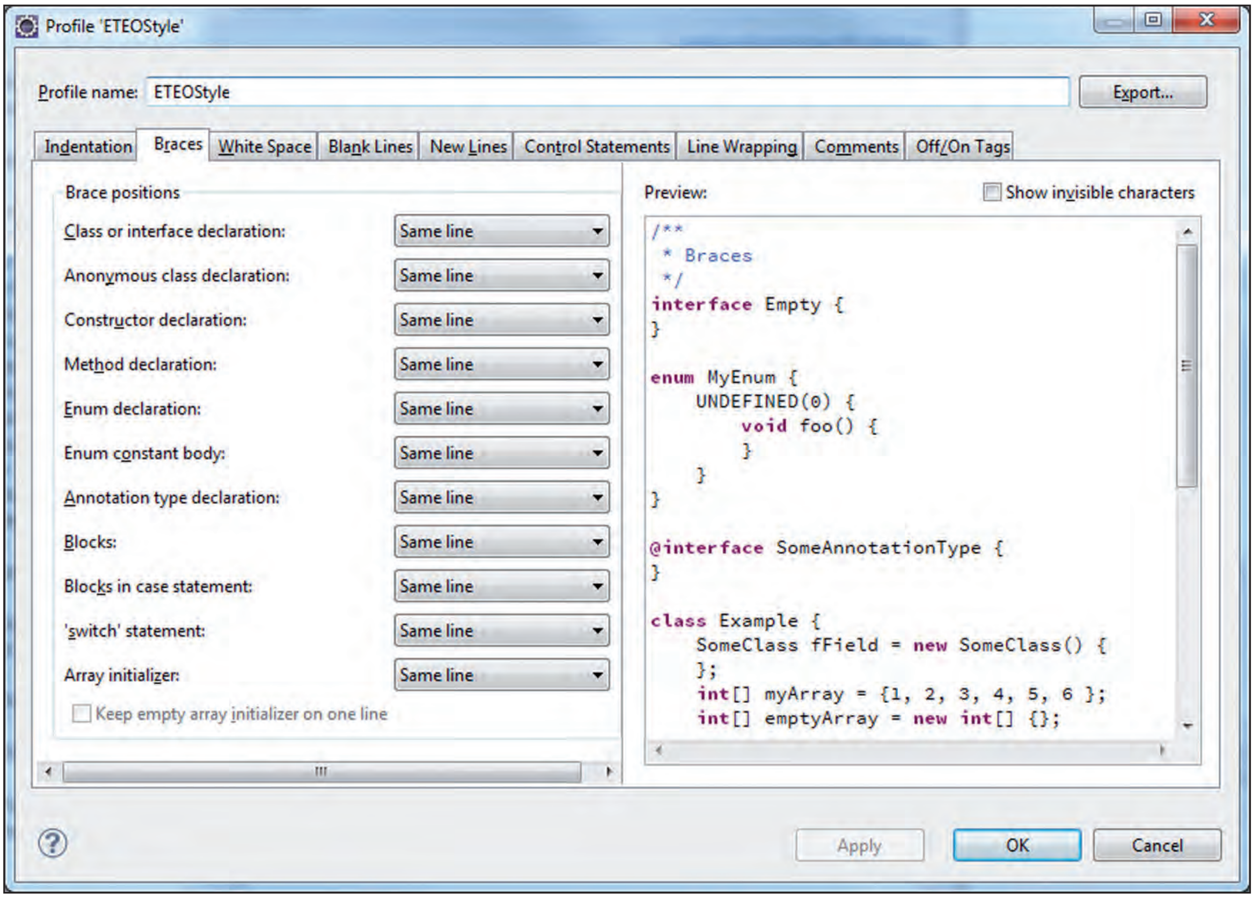


Abb. 3 Einstellungen des Codeformatierungsprofils

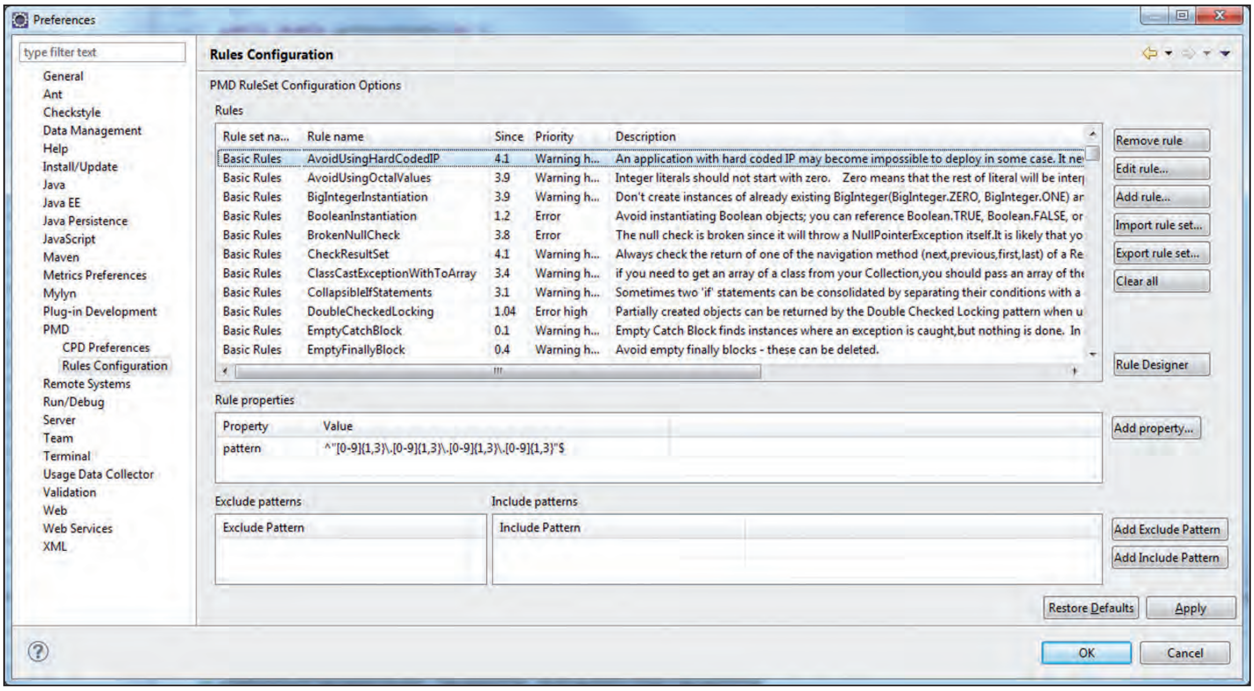


Abb. 4: PMD-Regeln

Automatisch formatieren

Bei der Erstellung eines brauchbaren Checkstyle-Regelsatzes für Altprojekte gibt es kein Patentrezept. Vielmehr werden sich die Regeln von Projekt zu Projekt unterscheiden, basierend auf dem zugrunde liegenden Quellcode. Um trotzdem eine Codequalitätsverbesserung mit Checkstyle zu erreichen, ist ein hohes Maß an Erfahrung mit dem Altprojekt vonnöten, sowie Zeit, um abzuwägen, ob die einzelnen Regeln anwendbar sind. Hat man sich schließlich auf einen Regelsatz geeinigt, muss sichergestellt werden, dass alle Klassen eine regelkonforme Formatierung aufweisen. Dies kann mit einem Codeformatierungstool erreicht werden, wie es beispielsweise in Eclipse standardmäßig integriert ist (**Abb. 2**, Window | Preferences | Java | Code Style | Formatter).

In den angezeigten Menüpunkten können die durch Checkstyle festgelegten Regeln in das entsprechende Profil eingetragen (**Abb. 3**) und nach erfolgreichem Anlegen (oder Editieren) die Formatierung auf das gesamte Projekt angewendet werden. Für die verbliebenen Warnungen kann entweder eine manuelle Anpassung oder die Entfernung aus dem Regelsatz erfolgen.

Ein Prise PMD

Neben Checkstyle kann eine weitere Überprüfung des Altprojekts mit PMD [1] erfolgen. Während Checkstyle sein Hauptaugenmerk auf die Formatierung des Quellcodes richtet, analysiert PMD den Programmierstil, wie das Design der Klassen, und gibt Empfehlungen auf Basis von Best Practices. PMD ist ebenfalls als Eclipse-, Jenkins- und Hudson-Plug-in [4] verfügbar. Im Gegensatz zu Checkstyle (in Verbindung mit einem Codeformatierungstool) besteht hier jedoch kaum die Möglichkeit, automatisiert Änderungen vorzunehmen und das Altprojekt an die PMD-Regeln anzugleichen. Um manuelle Anpassungen zu vermeiden, empfiehlt sich eine alternative Vorgehensweise. Anstatt aufwändig alle Regeln wie bei Checkstyle auf deren mögliches Verbesserungspotenzial für den Quellcode zu überprüfen, sollte der PMD-Regelsatz eines ähnlichen, aktuellen Softwareprojekts als Grundlage dienen. Eine erste Filterung von wichtigen Regeln wurde von dem dortigen Projektteam bereits getroffen. Gleichwohl müssen die Regeln natürlich ein zweites Mal überprüft werden. Hierfür bietet sich die direkte Darstellung in Eclipse an (**Abb. 4**, Window | Preferences | PMD | Rules Configuration). Nicht nur, dass die Regeln nach Kategorien sortiert sind, auch eine kurze Beschreibung wird angezeigt, die meist neben dem aussagekräftigen Regelnamen selbst ausreicht, um den Zweck der Überprüfung zu verstehen. Durch ausreichend Erfahrung mit dem Quellcode des Altprojekts sollte es dem Entwickler möglich sein, alle Regeln für das Altprojekt ohne umfangreiche Analyse und Suche in eine der folgenden drei Kategorien einzuteilen:

- Sinnvolle und zutreffende Regeln
- Zu entfernende Regeln
- Regeln, die möglicherweise entfernt werden sollten und/oder bei denen eine weitere Verifizierung sinnvoll erscheint

Listing 3

```
<!-- Identifiziert unbenutzten Quellcode  -->
<rule ref="ruleset/unusedcode.xml/UnusedPrivateField"/>
<rule ref="ruleset/unusedcode.xml/UnusedLocalVariable"/>
<rule ref="ruleset/unusedcode.xml/UnusedPrivateMethode"/>
<rule ref="ruleset/unusedcode.xml/UnusedFormalParameter"/>
```

Dabei können die letzten zwei Listen recht schnell anwachsen. Sollten Regeln in die letzte Kategorie einsortiert werden, muss genau abgewogen werden, inwieweit manuelle Formatierungen oder Refactorings sinnvoll und zeitlich machbar sind. Besonders Regeln der folgenden Kategorien sollten besser entfernt werden, als den Versuch zu unternehmen, das Altprojekt anzupassen:

- Design
- Migration
- Quellcodegröße

Als Vorzeigebeispiele können hier wiederum die Regeln genannt werden, die die Länge von Methoden und Klassen oder die Anzahl der Übergabeparameter einer Methode überprüfen. Berechtige Ausnahmen bilden, wie bei Checkstyle, die Regeln, die unbenutzten Code identifizieren (Listing 3). Nicht nur, dass das Löschen zügig erledigt ist, es wird ebenfalls die Übersichtlichkeit in den Klassen verbessert und die Einarbeitung für neue Mitarbeiter erleichtert. Analog zu Checkstyle gibt es auch für PMD keinen allgemeingültigen Regelsatz. Vielmehr muss man darauf vorbereitet sein, eine hohe Anzahl an Regeln zu entfernen. Dadurch kann aber immer noch eine, wenn vielleicht auch nur geringe, Codequalitätsverbesserung erreicht werden.

Ausblick

Im zweiten Teil der Artikelserie zeige ich, wie mit dem Framework Mockito [5] die Testabdeckung von neuem und altem Quellcode erhöht und verbessert werden kann. Weiterhin erkläre ich, wie man am besten mit alten fachlichen und technischen Fehlern umgeht und warum eine umfangreiche Dokumentation nicht nur der Zierde dient. Abschließend werden Möglichkeiten aufgezeigt, um die verbesserte Codequalität auch bei zukünftigen Erweiterungen und Fehlerbehebungen zu erhalten.

Daniel Winter studierte Informatik an der FH Zittau/Görlitz und arbeitet seit 2011 als Consultant für Softwareentwicklung bei der Saxonia Systems AG (daniel.winter@saxsys.de). Sein aktueller Fokus liegt in den Möglichkeiten der Codequalitätsverbesserung bei Projekten jedweder Art. Ausgiebige Erfahrungen in diesem Bereich sammelte er bei der Wartung einer Leasing-Refinanzierungssoftware.

Links & Literatur

[1] PMD: http://pmd.sourceforge.net/

[2] Checkstyle: http://checkstyle.sourceforge.net/

[3] Checkstyle Eclipse – Plug-in: http://eclipse-cs.sourceforge.net/

[4] Checkstyle Jenkins – Plug-in: https://wiki.jenkins-ci.org/display/JENKINS/Checkstyle+Plugin

[5] Mockito: http://code.google.com/p/mockito/

Warum guter Code und agile Tests ein schönes Paar sind

Drum prüfe, wer sich ewig bindet

What you get is what you see – was passiert jedoch, wenn man beim Testen nicht sieht, was man sehen wollte? Werden Tests und Entwicklung zu sehr getrennt, dann kann „zurück auf Los“ ein sehr weiter Weg sein. Besonders ein großer zeitlicher Abstand zwischen der Codeerstellung und den Tests kann die Entwicklungsgeschwindigkeit deutlich verlangsamen. Je später mangelhafte Codequalität offen gelegt wird, umso aufwändiger wird es, sie zu verbessern. Agiles Testen verfolgt daher den Ansatz, Testen und Entwickeln eng zu verzahnen.

von Maynard Harstick und Daniel Knapp

Automatisierte, einfache Tests von Anfang an reduzieren die Zahl komplexer Tests am Ende. Außerdem fungieren die Tests als Sicherheitsnetz für Entwickler beim Refaktorisieren. Dadurch bleibt der Code flexibel und kann kontinuierlich an die gegebenen Anforderungen angepasst werden. Eine hohe Testabdeckung wird so zum Schlüssel für bessere Softwarequalität.

Es klingt paradox und ist doch nur eine Frage von Ursache und Wirkung: Softwaretests stehen für einen Anfang, unabhängig davon, an welcher Stelle des Produktionsprozesses sie platziert sind. Das gilt zumindest dann, wenn sie nicht hundertprozentig fehlerfrei durchlaufen werden. Ein Test legt die funktionale Qualität der Software offen; verändern kann und soll er sie nicht. Insofern ist er die Instanz, die entscheidet, ob die Marschrichtung hin zur Auslieferung eingeschlagen wird oder zurück an den Start. Aber das ist nur die Wirkung. Die Ursache liegt nicht im Test selbst, es sei denn, er wurde fehlerhaft aufgesetzt. Sie liegt im Code. Und damit verlangt jeder Fehler im Code, an den Anfang der Programmierung oder gar der Konzeptionierung zurückzukehren.

Diese Rückkehr kann aufwändig und teuer werden, und das umso mehr, je größer das Delta zwischen Entwicklung und Eintreffen des Feedbacks wird. Gerade der klassische Ansatz, der Entwicklung und Test personell wie zeitlich komplett trennt, kann hier problematisch werden. Anstatt also das Testen einer eigenen – und autark agierenden – Abteilung zu überlassen, bringt die agile Vorgehensweise die Testexperten und die Entwickler zusammen, idealerweise ins gleiche Team. Denn der rein „externe“ Blick hat einige Nachteile: Es bedeutet Zeit und Aufwand, sich in Produktinkremente hineinzudenken, die jemand anders erstellt hat. Je länger die Zeitspanne zwischen Programmierung und Feedback wird, desto höher ist die Wahrscheinlichkeit, dass auch der Entwickler inzwischen gedanklich in andere Themen involviert ist und sich dann selbst erneut einarbeiten muss. Nicht zuletzt steigt die Gefahr, dass auf den fehlerhaften Code längst andere Inkremente aufbauen, die dann zwangsläufig mitbetroffen sind – ein Dominoeffekt der unerwünschten Art.

Das ist ein Grund, warum agiles Software Engineering auf konstantes und direktes Feedback setzt und dazu Entwicklung und Testen zu engen Freunden oder Kooperationspartnern macht. Der Test wird zu einer „ausführbaren Spezifikation“, die entweder der Entwickler oder der Tester aufsetzt, sobald die Anforderungen vollständig sind. Unit Tests entstehen vor und beim Schreiben von eigentlichem Produktivcode. Man kann sagen, der Produktivcode wird anhand eines Unit Tests entwickelt. Dabei entspricht der Test einer recht feingranularen Spezifikation der jeweiligen Funktonalität. Insofern erfüllt der Code immer die zugrunde liegende Spezifikation eines Unit Tests. Insgesamt klingt das zunächst, als ob gerade Unit Tests viel Aufwand für die Entwickler verursachen. Tatsächlich sind sie jedoch gleich aus zwei Gründen rationell: Erstens, weil für den Produktivcode eben nur so viel entwickelt wird, bis der die Anforderung spezifizierende Test erfüllt ist. Zweitens, weil die Fülle an Unit Tests am Anfang verhindert, während des Lebenszyklus der Software Bugschulden anzuhäufen.

Natürlich ist es schön, wenn sich Entwickler rein auf die Funktionalität konzentrieren können. Nur bringt es nicht viel, wenn die Zahl der zu fixenden Bugs synchron mit der Lebensdauer der Software steigt und daher mehr und mehr Zeit ausschließlich für Reparaturarbeiten benötigt wird – bis, im Extremfall, nur noch Schadensbegrenzung möglich ist. Diese Art von Hypothek sollte beim agilen Vorgehen gar nicht erst entstehen.

Beispielsweise deswegen nicht, weil der Code praktisch ständig refaktorisiert wird. Für dieses Refaktorisieren bilden die Tests eine Art Sicherheitsnetz, indem sie die Funktionalität einer Software dokumentieren. Mit diesem Schutz können die Entwickler den Code „aufräumen", verschlanken und an ihm feilen, ohne Funktionalitäten zu gefährden.

Test nicht gleich Test

Unabhängig von der Vielzahl an Bezeichnungen, Testtypen und Nomenklaturen lassen sich Testfälle in der so genannten *Testpyramide* (**Abb. 1**) nach ihrer Abstraktionsebene gliedern. Anders gesagt unterscheiden sich die Testfälle der einzelnen Ebenen hinsichtlich ihrer Komplexität und Zeitdauer.

Demnach empfiehlt es sich, die einzelnen Testfälle in der Praxis unterschiedlich einzusetzen. Wie, das zeigt ein Beispiel aus dem Unternehmen arvato infoscore GmbH,

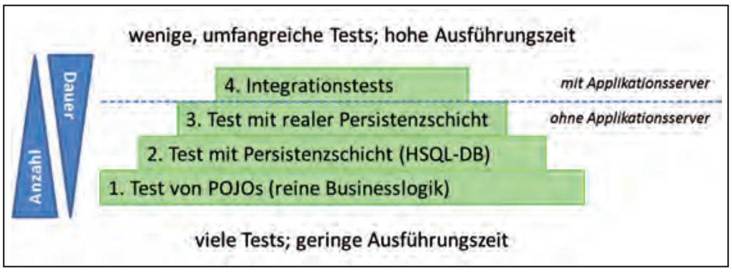


Abb. 1: Testpyramide

ein Dienstleister für ein integriertes und wertorientiertes Management von Kundenbeziehungen und Zahlungsflüssen. Als solcher bearbeitet das Unternehmen bis zu 50 000 Bonitätsanfragen pro Stunde. Die durchschnittliche Antwortzeit beträgt für drei Viertel aller Bonitätsanfragen weniger als eine Sekunde, bei insgesamt ca. 40 Millionen gespeicherten Merkmalen.

Die zugrunde liegende Systemlandschaft ist seit fünfzehn Jahren gewachsen und entsprechend komplex. Einzelne Systemkomponenten hatte die arvato infoscore GmbH selbst entwickelt, andere wurden von Dienstleistern erstellt und später übernommen. Derzeit werden nun die Einzelsysteme in einer plattformbasierten Lösung konsolidiert. Dazu erteilte die arvato infoscore GmbH dem Karlsruher Beratungs- und Entwicklungshaus andrena objects den Auftrag, die Systeme zu analysieren und Empfehlungen abzuleiten. Eine dieser Empfehlungen lautete, die Testabdeckung weiter zu erhöhen, besonders bei den älteren Systembestandteilen. Denn diese hohe Testabdeckung bildet die Voraussetzung für umfangreiche Systemumstrukturierungen, beispielsweise, um eine technische Schichtung einzuführen.

Die angestrebte Erhöhung der Testabdeckung ist in diesem Fall ein Synonym für „Ausbau der automatisierten Tests". Dazu entwickelte andrena objects ein Testkonzept, das primär für ein Teilprojekt pilotiert wurde. Seine beiden wesentlichen Eigenschaften bestehen darin, erstens die Testfälle nach ihrer Abstraktionsebene zu klassifizieren – sprich, die oben erwähnte Testpyramide einzuführen. Zweitens unterstützen entsprechende Hilfsklassen jede Testebene. Dazu nehmen sie nötige Konfigurationen vor und stellen oft benötigte Funktionen bereit. Das ermöglicht den Entwicklern, sich auf das Schreiben der Tests zu konzentrieren.

Innerhalb der Testpyramide bilden codenahe, reine Unit Tests die unterste und breiteste Ebene. Sie testen ausschließlich die Businesslogik, prüfen einzelne Klassen, sind permanent ausführbar und nehmen nur Sekunden an Laufzeit in Anspruch. Gemeinsam bilden sie das primäre Fangnetz, innerhalb dessen die Software restrukturiert und weiterentwickelt werden kann. Rein theoretisch ließe sich die gesamte Funktionalität auf dieser Ebene prüfen.

Die Praxis hingegen ist häufig komplexer, weil Fremdsysteme angebunden werden, im speziellen Fall eine Datenbank. Deshalb stehen auf der zweiten Ebene Integrationstests, in der lokalen Entwicklungsumgebung allerdings nicht gegen die reale Persistenzschicht, sondern gegen eine „HSQL-in-Memory"-Datenbank. Diese virtuelle Datenbank ist auch für mehrere gleichzeitig testende Entwickler schnell ansprechbar und sichert die lokale Unabhängigkeit der Tests und damit die Konsistenz des Zustands. Gleichbleibende Ausgangslagen bedeuten wiederholbare Tests, und die sind ihrerseits eine der Grundbedingungen für agiles Testen. Alle Änderungen der Datenbank nach dem Test zurückzunehmen – was hier der Fall ist –, verhindert die unerwünschten Interferenzen zwischen den einzelnen Tests.

Trotzdem sind natürlich auch Tests gegen die reale Datenbank sinnvoll. Sie laufen in der dritten Ebene und gewährleisten, dass die Persistenzschicht richtig konfiguriert wurde.

Diese drei Arten von Testfällen bewegen sich innerhalb der Java Virtual Machine, im Idealfall werden sie alle innerhalb eines definierten Zeitplans automatisiert ausgeführt, z. B. auf einem Build-Server (wie Hudson oder Jenkins).

Die oberste Schicht an Integrationstests berücksichtigt, dass sich manche Konfigurationen erst nach dem Hochfahren des Servers überprüfen lassen, wenn ein System auf einem Applikationsserver läuft. Tests der Ebene 4 starten zuerst den Server und schicken dann von außen Anfragen an das zu testende System.

Diese Tests werden sowohl manuell als auch automatisiert durchgeführt. Der Testmanager innerhalb des Scrum-Teams leitet diese Tests, die einen hohen Automatisierungsgrad haben. Die Ergebnisse dieser Tests und die enge Zusammenarbeit der Entwickler mit den Testmanagern innerhalb des Sprints helfen hierbei, dann schnell auf eventuelle Fehler zu reagieren.

Unabhängig davon, wer testet: Generell lautet die Faustregel, dass die Anzahl der Tests abnehmen sollte, je höher die Testebene ist. Denn synchron mit der Ebene steigt die Komplexität, nicht nur im Schreiben der Tests, sondern auch in der Ausführung. Wird etwa die Businesslogik in allen denkbaren Varianten auf der untersten, also Unit-Test-Ebene durchgespielt, dann reichen auf der Ebene der Integrationstests exemplarisch ein positives und ein negatives Beispiel. Da die Tests der untersten Ebene die schnellsten und einfachsten sind, sinkt der Gesamtaufwand signifikant. Der abgestufte Einsatz der verschiedenen Testfälle macht es den Entwicklern leichter, ihr Testmanagement in diesem Sinne effizienter zu gestalten.

Fazit

Tests gezielt weitreichend zu automatisieren und, je nach Abstraktionsgrad, unterschiedlich einzusetzen, erhöht die Testabdeckung und damit mittelbar die Codequalität. Damit sinken die ansonsten sehr häufigen, späteren Kosten zur Fehlerbehebung deutlich. Außerdem ist die hohe Testabdeckung elementare Voraussetzung für das Refaktorisieren, das seinerseits zur Basis des agilen Programmierens gehört. Und für agiles Programmieren hat sich das Beispielunternehmen arvato infoscore GmbH klar entschieden. Dazu werden bereits XP-Techniken wie Pair Programming und Test-First eingesetzt. Erste Eindrücke der betroffenen Entwickler und Fachbereiche sind sehr positiv. Im Verbund mit den Testfällen auf verschiedenen Ebenen wurde also die Basis gelegt, die Softwarequalität noch weiter zu erhöhen – sei es aus Anwender- oder aus Entwicklersicht.

Womit dann auch geklärt wäre, warum agiles Testen und guter Code ein wirklich schönes Paar sind. Agiles Testen unterstützt Entwickler dabei, guten Code zu entwickeln. Guter Code ist das Ziel jedes Entwicklers. Und wirtschaftlich sinnvoll.


Maynard Harstick ist Testmanager bei arvato infoscore im Bereich IT-Risk Management. Er ist seit über fünfzehn Jahren in der Softwarequalitätssicherung sowohl im klassischen Entwicklungsbereich als auch im agilen Umfeld tätig.


Daniel Knapp studierte Wirtschaftsingenieurwesen am KIT und arbeitet seit 2005 als Softwareentwickler bei der andrena objects ag. 2010 wurde er zum Leiter des Geschäftsfelds Lösungen berufen. Sein Hauptinteresse gilt dem Agile Software Engineering.

SOA

von Bernhard Löwenstein

W-JAX Speaker

Ein herzliches Grüß Gott zu einer weiteren Ausgabe der Java-Magazin-Retrospektive. Dieses Mal steht die Nummer 9 aus dem Jahr 2005 auf dem Programm. Im Mittelpunkt stand mit dem Thema SOA (Serviceorientierte Architektur) ein besonderer Hype der damaligen Zeit. Die Grundidee bei diesem Architekturstil bestand darin, sämtliche Unternehmensfunktionalitäten in Form von unabhängigen Diensten zur Verfügung zu stellen und dann darauf basierend die einzelnen Geschäftsprozesse umzusetzen. Ins Zentrum sollte dadurch der eigentliche Businessprozess und nicht mehr die pure Technik rücken, wie Chefredakteur Sebastian Meyen im Editorial richtig anmerkte. Als Technologie zur Umsetzung einer SOA empfahlen sich zu dieser Zeit SOAP-basierte Web Services, als Integrationstechnologien boten sich BPEL (Business Process Execution Language) und BPMN (Business Process Model and Notation) an. Festzuhalten ist, dass der SOA-Ansatz damals nicht mehr unbedingt neu war, denn beispielsweise bereits Jini (mittlerweile Apache River) verfolgte schon Jahre davor dieses Paradigma. Auch die Idee, Applikationen auf Basis von Services zusammenzubauen, war nicht revolutionär, sondern vielmehr eine logische Konsequenz aus den jahrzehntelangen Entwicklungen. Was einst mit Funktionen begann, nahm seine Fortsetzung mit Units, Modulen und Bibliotheken, ehe Objekte, Komponenten und in bisher letzter Instanz Dienste folgten. Die Idee dahinter blieb aber stets die gleiche, nämlich Software aus möglichst unabhängigen, wiederverwendbaren Bausteinen zusammenzubauen. Lediglich die Granularität der Bausteine wuchs über die Jahre. Der Artikel prognostizierte eine neue Rolle beim Entwicklungsprozess, und zwar die des SOA-Architekten. Dieser sollte als Bindeglied zwischen Fachbereich und IT-Bereich fungieren und somit für den Brückenschlag zwischen den beiden Welten sorgen. Wie entwickelte sich das Thema nun in den letzten Jahren? SOA war nicht, wie so oft in unserer Branche, nur ein vorübergehender Hype, sondern schaffte tatsächlich den Einzug in die IT. Wer heute Applikationen baut, ist jedenfalls gut beraten, seine Funktionalitäten in Serviceform zu realisieren und seine Prozesse auf Basis dieser Dienste umzusetzen. Im Idealfall wird dafür eine Beschreibungs- oder Modellierungssprache verwendet, wenngleich diese Vorgehensweise wenig verbreitet ist. Anstatt BPEL oder BPMN zum Verknüpfen der einzelnen Services zu einem Gesamtprozess zu verwenden, kommt in den meisten Fällen dafür immer noch herkömmlicher Programmcode zum Einsatz. Auch die explizite Rolle des SOA-Architekten findet man nur selten vor.

Ein weiterer Artikel beschäftigte sich mit statischen Variablen, Methoden und Blöcken in Java. Wie der Autor Michael Wiedeking gleich zu Beginn festhielt, ist das Keyword *static* in einer objektorientierten Sprache mit gewisser Vorsicht zu genießen und oftmals ein Indiz dafür, dass man seine Problemlösung besser nochmals überdenken sollte. So demonstrierte er anhand eines Beispiels, dass die Reihenfolge der statischen Codeteile innerhalb des Java-Quellcodes unbedingt zu beachten ist, da dies sonst zu unerwarteten Phänomenen führt. Diese Erkenntnis überrascht vielleicht den einen oder anderen Java-Entwickler, der bisher davon ausgegangen ist, dass die Reihenfolge der einzelnen Codekonstrukte vollkommen belanglos ist.

Mit Apache Cocoon wurde weiterhin auf ein Framework eingegangen, das seinerzeit groß im Gespräch war. Ein Blick auf die Website zeigt, dass das Projekt nach wie vor weiterentwickelt wird, wenngleich sich der Fokus verändert hat. Standen seinerzeit Webapplikationen im Mittelpunkt, rückt in der dritten Version nun der allgemeinere Anwendungsfall ins Zentrum.

Sich nicht über die Jahre retten konnte sich das ebenfalls in dieser Ausgabe vorgestellte HiveMind-Projekt. Es handelte sich hierbei laut Jakarta um einen „services und configuration microkernel“. Damit ließ sich der Dependency-Injection-Mechanismus für Services über so genannte Service-Points umsetzen. Der Nutzer konnte hierbei unter anderem konfigurieren, wann ein Service erzeugt werden soll und wie Instanzen davon erstellt werden. Konkret konnte er zwischen vier Service-Models wählen: primitive, singleton, threaded und pooled. Doch all die netten Konzepte halfen nichts – seit 15. April 2009 ziert der Satz „Apache HiveMind has been retired.“ die offizielle Apache-Projektseite.

Ich wünsche einen erholsamen Sommer und Happy Coding!

Bernhard Löwenstein (bernhard.loewenstein@java.at) ist als selbstständiger IT-Trainer und Consultant für javatraining.at und weitere Organisationen tätig. Als Gründer und ehrenamtlicher Obmann des Instituts zur Förderung des IT-Nachwuchses führt er außerdem altersgerechte Roboter-Workshops für Kinder und Jugendliche durch, um diese für IT und Technik zu begeistern.

Vorschau auf die Ausgabe 10.2013

Continuous Delivery

Continuous Integration hat seinen Ursprung in der Lehre des Extreme Programming (XP) und ist mittlerweile fester Bestandteil unserer Softwareentwicklung. Continuous Delivery geht einen Schritt weiter und will die letzte Meile bis zur Produktion beschleunigen. Doch die Umsetzung dieser Softwarepipeline ist nicht trivial. Nächsten Monat beschäftigen wir uns daher mit wichtigen Architektur-Patterns für Continuous Delivery und dem Weg von Continuous Integration zu Continuous Delivery.

Aus redaktionellen Gründen können sich Themen kurzfristig ändern.

Die nächste Ausgabe erscheint am 4. September 2013

Querschau

eclipse MAGAZIN **Ausgabe 5.2013 | www.eclipse-magazin.de**

- **Hello Kepler! Alles Wissenswerte rund um das zehnte Simultaneous Release**
- **Test Drive: Einführung in Java 8 und Java Development Tools**
- **Umstieg heißt Umdenken: Migration auf Eclipse 4**

MOBILE TECHNOLOGY **Ausgabe 3.2013 | www.mobiletechmag.de**

- **Bluetooth LE und iOS 6 – Alternative zu NFC?**
- **Monetarisierung von Mobile-Apps: Zanox SDK für iOS und Android**
- **Odyssee der anderen Art: Mit Sencha Touch und Apache Cordova zur mobilen Applikation**

windows .developer **Ausgabe 9.2013 | www.windowsdeveloper.de**

- **NUI: Modellierung von Informationen und Interaktionen**
- **Azure BizTalk Services: Systemintegration als PaaS-Lösung**
- **WP8-Foto-APIs: Der Fotoapparat in der Hosentasche**

Inserenten

Verlag:
Software & Support Media GmbH

Anschrift der Redaktion:
Java Magazin
Software & Support Media GmbH
Darmstädter Landstraße 108
D-60598 Frankfurt am Main
Tel. +49 (0) 69 630089-0
Fax. +49 (0) 69 630089-89
redaktion@javamagazin.de
www.javamagazin.de

Chefredakteur: Sebastian Meyen
Redaktion: Claudia Fröhling, Corinna Kern, Diana Kupfer
Chefin vom Dienst/Leitung Schlussredaktion: Nicole Bechtel
Schlussredaktion: Jennifer Diener, Frauke Pesch
Leitung Grafik & Produktion: Jens Mainz
Layout, Titel: Tobias Dorn, Flora Feher, Karolina Gaspar, Dominique Kalbassi, Laura Keßler, Nadja Kesser, Maria Rudi, Petra Rüth, Franziska Sponer

Autoren dieser Ausgabe:
Stephan Elter, Dino Esposito, Bob Genom, Maynard Harstick, Daniel Knapp, Thomas Leitner, Arne Limburg, Bernhard Löwenstein, Jochen Mader, Christian Meder, Cornelius Moucha, Michael Müller, Dr. Matthias Nagel, Dominik Obermaier, Wolfgang Pleus, Lars Röwekamp, Stefan Siprell, Sebastian Weber, Thomas Wenzlaff, Daniel Winter, Eberhard Wolff

Anzeigenverkauf:
Software & Support Media GmbH
Patrik Baumann
Tel. +49 (0) 69 630089-20
Fax. +49 (0) 69 630089-89
pbaumann@sandsmedia.com

Es gilt die Anzeigenpreisliste Mediadaten 2013

Pressevertrieb:
DPV Network
Tel.+49 (0) 40 378456261
www.dpv-network.de

Druck: PVA Landau
ISSN: 1619-795X

Abonnement und Betreuung:
Leserservice Java Magazin
65341 Eltville
Tel.: +49 (0) 6123 9238-239
Fax: +49 (0) 6123 9238-244
javamagazin@vuservice.de

Abonnementpreise der Zeitschrift:

Inland:	12 Ausgaben	€ 118,80
Europ. Ausland:	12 Ausgaben	€ 134,80
Studentenpreis (Inland)	12 Ausgaben	€ 95,00
Studentenpreis (Ausland):	12 Ausgaben	€ 105,30

Einzelverkaufspreis:

Deutschland:	€ 9,80
Österreich:	€ 10,80
Schweiz:	sFr 19,50
Luxemburg:	€ 11,15

Erscheinungsweise: monatlich