Nr. 04 | Winter 2012 | www.ijug.eu

Java aktuell

Java aktuell

Das Magazin der Java-Community

Die „Bohne“ in voller Blüte

Besser mit Java entwickeln

Java aktuell Update

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

iJUG Verbund

Wolfgang Taschner
Chefredakteur Java aktuell

Business as usual

Das diesjährige Java Forum war ein voller Erfolg für die Java User Group Stuttgart. Mit mehr als 1.200 Besuchern war die Veranstaltung ausverkauft. Trotz der räumlichen Enge gab es fast überall zufriedene Gesichter. Das war nicht immer so. Als Oracle vor zwei Jahren die Sun-Übernahme bekannt gab und damit die Herrschaft über Java übernahm, war an gleicher Stelle eine große Verunsicherung, oft sogar eine ablehnende Haltung gegenüber dem Datenbank-Hersteller spürbar.

Noch vor einem Jahr gab es bei der Vorstellung von Java 7 viele Skeptiker. Der Ausfall der Live-Übertragung nach Stuttgart hatte für manche Besucher sogar symbolischen Charakter. Eines war klar: Oracle musste sich das Vertrauen der Community erst erwerben.

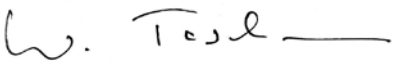
Heute haben sich die Wogen geglättet. Der Oracle-Stand auf dem Java Forum Stuttgart war gut besucht, Evgenia Rosa und Peter Doschkinow gingen in ihrer freundlichen Art auf alle Anliegen der Besucher ein und die Stuhlreihen bei den Vorträgen der beiden anderen Oracle-Mitarbeiter Wolfgang Weigend und Dalibor Topic waren gefüllt. Bleibt zu hoffen, dass die gute Stimmung weiter anhält, auch wenn sich kürzlich durch die Ankündigung von Oracle, dass Jigsaw nicht in der nächsten Java-Version enthalten sein wird, schon wieder erste Verstimmungen andeuten. Das war aber zu Sun-Zeiten nicht anders.

Peter Doschkinow und Wolfgang Weigend haben sich auch an dieser Ausgabe mit jeweils einem richtungsweisenden Artikel zu JAX-RS 2.0 und Java FX 2.1 beteiligt. Scott Kovatch, ein weiterer Oracle-Mitarbeiter, äußert sich zu Java und Mac OS X. Darüber hinaus kommen erfahrene Experten zu Wort, beispielsweise Markus Eisele zu Java EE 7, Tobias Unger und Jochen Traunecker zu Apache Camel und Apache Synapse, Dominik Schadow zu Apache Camel Security oder Marcel Bruch und Andreas Sewe zu Eclipse Code Recommenders. Von Microsoft und SAP melden sich die Evangelisten zu Wort, Holger Sirtl stellt das neue Release von Windows Azure vor und Matthias Steiner die SAP NetWeaver Cloud. Ich freue mich auf Ihr Feedback an redaktion@ijug.eu.

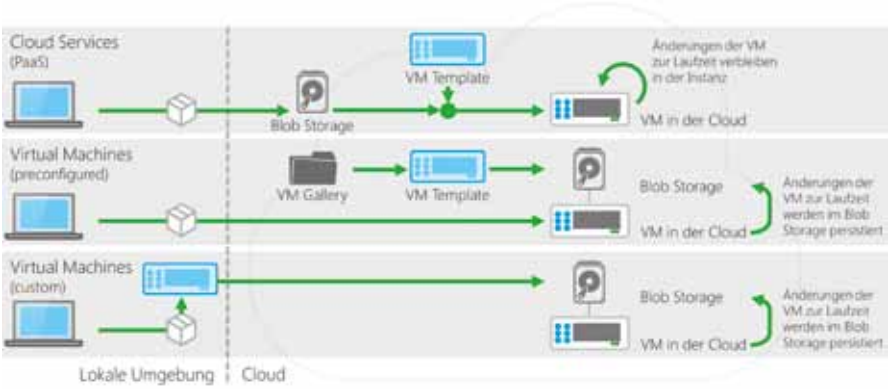
Falls Sie sich mit einem Artikel in der nächsten Ausgabe beteiligen möchten, schreiben Sie mir bitte vorab eine E-Mail. Die Ausgabe 01/2013 erscheint am 5. Dezember 2012, Redaktionsschluss ist am 1. Oktober 2012. Mit dem Motto „Aus der Community – für die Community" hat sich die Java aktuell zu einer der führenden Zeitschriften für Entwickler im deutschsprachigen Raum entwickelt. Dafür danke ich allen Autoren, die ihre Erfahrungen mit den Leserinnen und Lesern teilen.

Bleibt mir nur noch, Sie auf die DOAG 2012 Konferenz + Ausstellung vom 20. – 22. November 2012 in Nürnberg hinzuweisen. Neben zahlreichen interessanten Vorträgen erfahrener Java-Entwickler und -Architekten gibt es auch jeweils eine Keynote von Patrick Curran, Leiter des Java Community Process (JCP), sowie von Ed Burns, Spec Lead für JavaServer Faces.

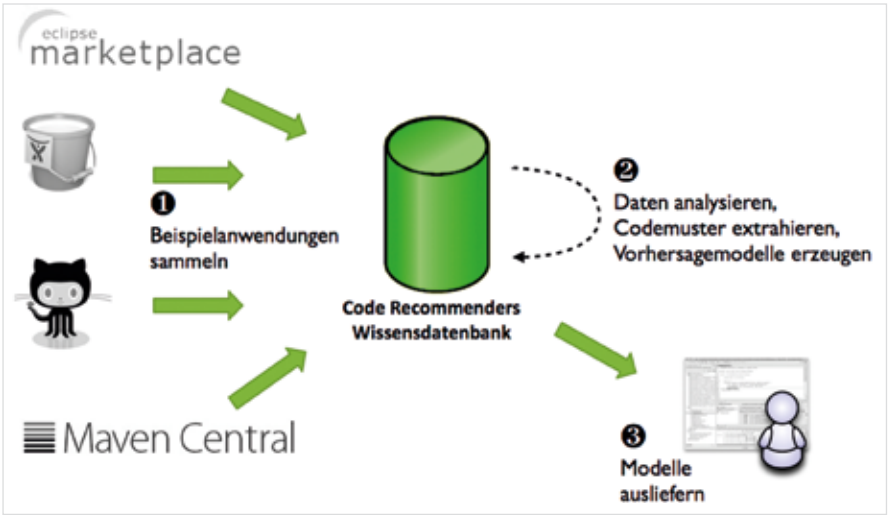
Ihr



Mit Java EE 7 für die Cloud entwickeln, Seite 14



Windows Azure bietet neue Möglichkeiten für Java-Entwickler, Seite 39



Eclipse Code Recommenders: eine Erweiterung der Java-Development-Tools, Seite 44

Das Java-Tagebuch

Andreas Badelt, Leiter der DOAG SIG Java

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

11. April 2012

JDK 8: Aktualisierter Zeitplan vorgeschlagen

Mathias Axelsson, Lead Release Manager für das Oracle JDK 8, das im Wesentlichen aus dem OpenJDK besteht, hat Milestones für die Implementierung vorgeschlagen – ohne diese schon mit den Features zu verknüpfen. Im Vergleich zum ursprünglichen Fahrplan nach der Neuaufteilung von JDK 7 und 8 („Plan B") ergibt sich inzwischen eine Verspätung von rund neun Monaten. Milestone 1 soll am 24. April 2012 sein, die weiteren folgen dann jeweils im Abstand von sechs bis zwölf Wochen. M6 „feature complete" ist dann für den 30. Januar 2013 geplant. Aufgrund der vielen Features sieht er einen Stabilisierungs-Zeitraum in ähnlicher Länge wie für das JDK 7 vor, sodass sich ein Release-Datum im September 2013 ergibt. Dazu soll in zwei Schritten die Mindest-Priorität der Bugs erhöht werden, für die Fixes aufgenommen werden – ab Mitte Juni 2013 nur noch „Showstopper". Das bezieht sich zunächst nur auf Änderungen im Rahmen der Maintenance, ausgenommen sind die im Umbrella JSR aufgelisteten Component JSRs wie „Lambda Expressions" und das neue Modulsystem „Jigsaw" – diese haben noch etwas mehr Zeit. Um Probleme mit (zu) spät entdeckten Bugs zu vermeiden (wie im Falle des JDK 7 und Lucene), schlägt Axelsson ein Datum Anfang April 2013 vor, zu dem alle Tests abgeschlossen und alle Bugs gemeldet sein sollen, verbunden mit einem „proaktiven Ansatz" für das frühe Testen und Melden von Fehlern – was auch immer das heißen soll. Bugs, die später gemeldet werden, können laut Axelsson eventuell nicht mehr rechtzeitig behoben werden. Letzteres ist nun nichts Neues und auch keine Änderung gegenüber JDK 7. Der „proaktive Ansatz" sollte nicht nur aus der Hoffnung bestehen, dass die Community, insbesondere die vielen großen Open-Source-Projekte, dazugelernt haben und früher eigene Tests durchführen – Hoffnung ist ja bekanntlich keine Strategie. Auch Oracle sollte hier einiges beisteuern, um den Erfolg – ein von Anfang an stabiles JDK 8 – sicherzustellen:

- http://mail.openjdk.java.net/pipermail/jdk8-dev/2012-April/000942.html
- http://mail.openjdk.java.net/pipermail/jdk8-dev/2012-April/000941.html
- http://openjdk.java.net/projects/jdk8/

26. April 2012

Java 7 wird Default-JRE

Zehn Monate nach dem ersten Release von Java SE 7 kommt nun Update 4 heraus. Es enthält unter anderem JavaFX 2.1 und den lang ersehnten G1 Garbage Collector. Außerdem ist es das erste vollständige Release 7 für Mac OS (inklusive JDK und JavaFX) und damit auch das erste „richtige" Mac-OS-Release, das unter Leitung von Oracle entstanden ist. In Zukunft sollen die Versionen für Mac OS zeitgleich mit denen für Linux, Windows und Solaris freigegeben werden, wie man im Blog von Henrik Ståhl nachlesen kann, dem obersten Produkt-Strategen für die Java-Plattform. Update 4 ist das erste „Consumer Release" und wird damit als Default-JRE auf „java.com" veröffentlicht, was bedeutet, dass bei automatischen Upgrades auf SE 7 aktualisiert wird. Diese Entscheidung stößt aber nicht überall auf Zustimmung, da es immer noch Probleme gibt, die für viele Anwender eine Aktualisierung unmöglich machen. Diese müssen gegebenenfalls die automatische Aktualisierung ausschalten. Bereits im Frühjahr hatte Oracle den kostenlosen Bugfix-Support für Java SE 6 bis

November 2012 ausgedehnt. Henrik Ståhl weist in seinem Blog darauf hin, dass dieser Support normalerweise sechs Monate nach der ersten Freigabe des nächsten „Major Release" endet. Trotzdem gibt es auch hier Widerspruch von Anwendern aus den oben genannten Gründen. Der iJUG will daher eine Umfrage zur Akzeptanz von Java SE 7 starten: https://blogs.oracle.com/henrik/entry/oracle_jdk_and_javafx_sdk

26. April 2012

JCP-Webseite überarbeitet

Parallel zu den Veränderungen in der Organisation und im Prozess hat der JCP seine Webseite überarbeitet, die gerade live gegangen ist. Schwerpunkte sind die generelle Verbesserung der Benutzerfreundlichkeit sowie das leichtere Auffinden relevanter Informationen: https://blogs.oracle.com/jcp/entry/jcp_org_site_improvements_released

26. April 2012

Umfrage zu Java-Web-Frameworks: JSF und GWT bald Kopf an Kopf?

Die Orientation in Objects GmbH hat die Ergebnisse einer Umfrage zu Java-Web-Frameworks veröffentlicht. Auch wenn man bezüglich der Repräsentativität vorsichtig sein muss, sind doch einige interessante Aspekte dabei: Momentan liegt bei der Nutzung „Java Server Faces" noch deutlich vorn, gefolgt von „Spring MVC" und dem Google Web Toolkit. Es deute sich aber eine Konsolidierung zugunsten von JSF und GWT an. Gefragt wurde allerdings nur nach den bereits erwähnten Frameworks plus Grails, Struts 2 und „Sonstigen". Apache Wicket blieb zum Beispiel außen vor, zumindest wurden die Sonstigen nicht weiter analysiert. Ein weiterer interessanter Aspekt ist die Art der Auswahl: „Java Server Faces" wurden deutlich öfter einem Projekt vorgegeben als andere Frameworks, in der Regel gestützt durch vorhergehende Analysen oder Prototypen. Dies, so die Autoren, könne an der politischen Bedeutung von JSF als Standard liegen. Gleichzeitig ist die Unzufriedenheit der Entwickler mit Java Server Faces deutlich geringer als mit dem GWT – wobei die Frage leider unbeantwortet bleibt, ob dies eher technische oder eben politische Gründe hat, da dem Projekt keine Wahl gelassen wurde: http://www.oio.de/public/java/java-web-frameworks-vergleich/jsf-vs-gwt-studie.htm

30. April 2012

Apache TomEE v1.0 freigegeben

Die Apache Software Foundation hat das Release 1.0 von TomEE freigegeben, dem auf Tomcat basierenden „EE Application Server". Dieser könnte den etablierten Application-Servern Konkurrenz machen, auch wenn er nicht den vollen Umfang von Java EE 6 unterstützt, sondern „nur" für das reduzierte Web-Profile zertifiziert ist. Die Fertigstellung von TomEE resultiert aus der gemeinsamen Anstrengung vieler Apache-Projekte, deren Arbeit in den Application-Server einfließt – etwa OpenEJB, OpenWebBeans, OpenJPA und MyFaces: https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces27

1. Mai 2012

James Gosling spricht

James Gosling hat sich in seinem Blog zum laufenden Android-Rechtsstreit geäußert. Unter anderem sorgt er sich um die Folgen, sollte Oracle im Falle der Urheberrechtsverletzungen gewinnen, die noch vor den Patentverletzungen verhandelt werden. Zum bekannten Phänomen des „Patent-Trolls" könnte dann der „Copyright Troll" hinzukommen. Entgegen seiner bekannten Meinung würde Oracle es aber niemals zu irgendeinem der „Alptraum"-Szenarien kommen lassen, da dies ihren eigenen Interessen widerspräche. Dann findet er sogar erstaunlich warme Worte für Oracle: Sie seien bislang unerwartet gute Verwalter von Java gewesen. Abschließend spricht er noch über Patente: Was die Java-Community angetrieben habe, sei immer die Interoperabilität gewesen und um diese zu erzwingen, habe Sun Patente eingesetzt: „Wenn Du die Spec befolgst, kannst Du sie kostenlos nutzen. Ein gutes Resultat für Entwickler aus einem falschen Patentsystem heraus": http://nighthacks.com/roller/jag/entry/comments_around_oracle_v_google

11. Mai 2012

Der JCP wirbt um Startups

Das Project Management Office des JCP hat ausgehend von Diskussionen in den Executive Committees eine Offensive gestartet, um mehr Java-Endanwender einzubinden. Insbesondere kleine Unternehmen und Startups sind im Fokus. Ziel ist es, den Prozess noch besser in die vielfältige Community einzubinden. Dies hatte zuletzt ja gut für die Java-User-Groups geklappt: http://jcp.org/

21. Mai 2012

Java 7 wird von Entwicklern angenommen

Die Akzeptanz von Java SE 7 ist entgegen den Diskussionen um das automatische Upgrade ziemlich gut, jedenfalls unter Entwicklern. Das ist zumindest in einer Studie von ZeroTurnaround nachzulesen, auf die sich Henrik Ståhl in seinem Blog bezieht. Danach benutzen 88 Prozent der Befragten Java SE 6, aber immerhin auch schon 23 Prozent SE 7. Dies sei eine erstaunlich gute Akzeptanz, so die Autoren der Studie. Das könnte auch daran liegen, dass die Entwickler seit Ende 2006 auf das nächste große Release warten müssten und entsprechend gierig auf neue Features waren: https://blogs.oracle.com/henrik/entry/java_7_adoption_at_23

27. Mai 2012

java.net-Umfrage: Entwickler zufrieden mit „Plan B"

In einer Umfrage auf „java.net" haben 60 Prozent der Teilnehmer geantwortet, dass die Entscheidung richtig war, einige Features auf Java SE 8 zu verschieben und dafür SE 7 früher (oder mit weniger Verspätung) herauszubringen – was der Java-Community von Mark Reinhold, Oracle, als „Plan B" offeriert worden war: http://weblogs.java.net/blog/editor/archive/2012/05/27/poll-result-developers-agree-java-7-java-8-plan-b-decision

8. Juni 2012

Java ME: Neues Nokia SDK 2.0

Totgesagte leben länger. Nokia hat eine neue Serie von Smartphones mit Touch-

screen im unteren Preissegment herausgegeben. Alle Devices unterstützen Java ME. Dazu gibt es, als Beta-Version, das neue Nokia SDK 2.0 for Java. Damit bestätigt sich einmal mehr, dass das Einsatzgebiet für Java ME die sogenannten „feature phones" sind. Böse Zungen würden behaupten: Das sind die Geräte ohne besondere Features. Allerdings scheint das – siehe Touchscreen – auch nicht mehr zu stimmen: https://blogs.oracle.com/java/entry/new_nokia_sdk_2_0 und http://www.linux-magazin.de/NEWS/Nokia-erneuert-Java-SDK-fuer-Series-40

12. Juni 2012

JCP.Next-Projekt schreitet voran

Der Umbau des Java Community Process macht Fortschritte: JSR-355 „JCP Executive Merge" ist ohne Gegenstimmen angenommen worden. Damit steht nun fest, dass es in Zukunft nur noch ein Executive Committee für SE/EE und ME geben wird, das dann aus 25 (statt bislang jeweils 16) Mitgliedern bestehen wird. Gleichzeitig werden Mitglieder nur noch für zwei statt drei Jahre gewählt, wobei jedes Jahr die Hälfte der Sitze neu gewählt wird (nur Oracles Sitz als 25ter ist permanent). Zeitgleich ist der dritte JCP.Next JSR (358) eingereicht worden. In diesem geht es hauptsächlich um eine Überarbeitung des Java Specification Participation Agreement, das die Rechte und Pflichten der JCP-Mitglieder regelt. Dieser JSR wird in den Blogs bereits als Langläufer angekündigt (zumindest im Vergleich mit seinen Vorgängern) und soll mindestens ein Jahr benötigen: http://java.net/projects/jsr348/pages/Home

13. Juni 2012

JRE-Auto-Update auf 7 und die Oracle E-Business Suite

Offensichtlich hat Oracle sich mit dem bereits erwähnten Auto-Update des JRE auf Version 7 selbst ein Bein gestellt. Die E-Business Suite ist noch nicht für Java 7 zertifiziert. Daher werden die Anwender nun von Oracle aufgefordert, auf ihren Rechnern das Auto-Update zu deaktivieren und manuell auf die jeweils aktuellste Java-6-Version zu aktualisieren, um Sicherheitslücken zu vermeiden – jedenfalls so lange, bis die Zertifizierung für Java SE 7 vorliegt: https://blogs.oracle.com/stevenChan/entry/bulletin_disable_jre_auto_update und http://www.doag.org/de/home/aktuelle-news/article/inkompatibilitaeten-zwischen-jre-7-und-e-business-suite-windows-jre-auto-update-sollte-deaktiviert.html

13. Juni 2012

JavaFX-Entwicklung jenseits der GUI-Spielereien

JavaFX findet immer mehr Verbreitung und viele Entwickler machen sich Gedanken, wie das optimale Design darauf basierender Anwendungen aussehen sollte und wie sie in die bestehende Applikations-Landschaft eingebaut werden können. Entsprechend wächst die Anzahl von Artikeln und Tutorials im Internet zu diesen Themen. So wurden auf dem Oracle Technology Network gerade neue Tutorials von Adam Bien und Jim Weaver veröffentlicht mit dem Anspruch, JavaFX-Entwicklung auf „Enterprise-Niveau" zu zeigen: https://blogs.oracle.com/java/entry/best_practices_for_javafx_21 und https://blogs.oracle.com/java/entry/the_enterprise_side_of_javafx

20. Juni 2012

„Adopt OpenJDK"-Initiative will mit JUGs einen „Bugathon" durchführen

Eine andere Form der Community-Beteiligung, diesmal deutlich näher am Code, hat die „Adopt OpenJDK"-Initiative ins Leben gerufen, ein Ableger von „Adopt a JSR", hinter dem unter anderem die „London Java Community" steht. Es soll zunächst von drei Java-User-Groups ein „Bugathon" für das OpenJDK 8 durchgeführt werden, also ein Event, bei dem die Mitglieder dieser JUGs unter Anleitung versuchen, möglichst viele Bugs aufzuspüren und zu dokumentieren. Wenn das Format erfolgreich ist, wird es weitere Bugathons mit anderen JUGs geben: http://weblogs.java.net/blog/editor/archive/2012/06/20/java-user-groups-sought-adopt-openjdk-bugathon-pilot-program und http://java.net/projects/adoptopenjdk/pages/AdoptOpenJDK

27. Juni 2012

Eindrücke vom öffentlichen JCP-Executive-Committee-Meeting

„java.net"-Chefredakteur Kevin Farnham berichtet in seinem Blog über seine Eindrücke vom ersten öffentlichen Meeting der Executive Committees im Jahre 2012. Etwas beunruhigend sind seine Ausführungen darüber, dass es vielleicht zu optimistisch sei, von einem Abschluss des JCP-Umbaus innerhalb weniger Jahre auszugehen. Jede Änderung werde von den beteiligten Firmen und ihren Anwälten genauestens begutachtet – letztlich gehe es um Unternehmens-Interessen. Daher hier ein Aufruf an alle Leserinnen und Leser: Engagiert Euch im JCP – er heißt nicht umsonst „Java Community Process" und sollte nicht von Firmen-Anwälten dominiert werden. Java lebte und lebt von der breiten Beteiligung aller, unabhängig von Firmeninteressen: http://weblogs.java.net/blog/editor/archive/2012/06/27/first-jcp-executive-committee-public-meeting-sets-sight-future

27. Juni 2012

Eclipse und NetBeans verdrängen IDEs für „Embedded Software"

Eclipse wird immer häufiger auch für die Entwicklung von „Embedded Software" genutzt. Sie verdrängt dort die proprietären IDEs, da die Hersteller sich offensichtlich versprechen, von der Community-basierten Entwicklung zu profitieren, indem sie nur noch spezielle Features selbst implementieren. Jetzt bekommt Eclipse auch in diesem Bereich Konkurrenz von einer anderen großen Java-IDE: Microchip hat seine Spezial-IDE auf Basis von NetBeans umgesetzt: https://blogs.oracle.com/geertjan/entry/eclipse_and_netbeans_replacing_embedded

28. Juni 2012

Groovy 2.0 freigegeben

Die auf der JVM laufende, dynamische Skriptsprache Groovy ist jetzt in der Version 2.0 freigegeben. Diese nutzt viele Neuerungen von Java SE 7 wie die Invoke-Dynamic-Funktion und einige der Syntax-Verbesserungen von „Project Coin", die in Groovy übernommen wurden, soweit nicht schon vorhanden. Auch eine statische Typ-Prüfung wird nun angeboten – für eine dynamische Sprache eine nicht ganz so offensichtliche, aber für bestimmte Einsatzgebiete sehr sinnvolle Verbesserung. Etwas merkwürdig bei der ganzen Aktivität ist, dass der JSR für die Integration von Groovy mit der JVM (JSR-241) seit 2004 keine sichtbaren Fortschritte macht und aufgrund erfolgloser Suche nach einem neuen Spec-Lead inzwischen den Status „schlafend" hat. Aber es geht ja offensichtlich auch so voran: http://www.heise.de/developer/meldung/Groovy-2-0-fuehrt-statische-Kompilierung-und-Typpruefung-ein-1627754.html

6. Juli 2012

Firmen unterstützen „Adopt OpenJDK"

Die ersten Firmen unterstützen die Adopt OpenJDK-Initiative. CloudBees bietet Open-Source-Projekten, die frühzeitig mit dem JDK 8 testen wollen, jetzt Builds mit einer aktuell gehaltenen JDK-8-Version an, damit diese ohne großen Aufwand frühzeitig mit der neuen Version testen können. Damit sollen Probleme wie beim Launch von JDK 7 im Sommer 2011 vermieden werden. Einige Open-Source-Projekte meldeten damals wenige Wochen vor der offiziellen Freigabe schwerwiegende Fehler, die dann erst mit Update 1 behoben wurden: http://blog.cloudbees.com/2012/07/adopt-openjdk.html

7. Juli 2012

JSR Updates: SIP Servlet & Co.

Neues vom JCP: Ein neuer JSR zu „SIP Servlet 2.0" wurde eingereicht (JSR-359). Außerdem sind der „Final Draft" für JSR-355 „Executive Committee Merge" und ein „Early Draft für die Servlet 3.1-Spezifikation veröffentlicht worden: https://blogs.oracle.com/jcp/entry/jsr_updates8

11. Juli 2012

iJUG-Umfrage: Support für JDK 6 endet zu früh

Der kostenlose Support für das JDK 6 endet zwar erst im November, dies ist nach Empfinden vieler Anwender aber zu früh: An der Umfrage des iJUG haben 234 Anwender teilgenommen, knapp die Hälfte davon ist dieser Meinung (siehe Seite 38). Laut Tobias Frech (iJUG-Vorstand und Board-Mitglied der Java User Group Stuttgart) ist das Thema „Update auf Java 7" bei vielen Herstellern Java-basierter Software noch gar nicht richtig angekommen: http://www.ijug.eu/index.php?option=com_content&view=article&id=61:umfrage-des-ijug-zu-java-7&catid=2:news&Itemid=4

17. Juli 2012

Project Jigsaw soll auf JDK 9 verschoben werden

Mark Reinhold berichtet in seinem Blog über „Challenges" bei der Entwicklung eines so tiefgreifenden Features wie dem Modulsystem Jigsaw und dass die Zeit nach dem aktuellen Release-Plan zu knapp wird. Er schlägt daher vor, Jigsaw auf das nachfolgende Release zu verschieben. Außerdem empfiehlt er für die Zukunft einen festen „Release Train", der alle zwei Jahre ein neues Release liefert mit allen Features, die es rechtzeitig geschafft haben. Die Nachricht macht in der Community natürlich binnen weniger Stunden die Runde – zum einen natürlich wegen der Enttäuschung um Jigsaw, das ja bereits aus JDK 7 herausgefallen ist. Zum anderen wird die Diskussion eröffnet, ob die pünktliche Ankunft eines Release wichtiger ist oder die Tatsache, dass es die erwarteten „Passagiere" enthält. Nachdem die Diskussion sich etwas abgekühlt hat, gibt es aber auch Kommentare dahingehend, dass Jigsaw einfach zu riskant ist, um es in einer halbfertigen Variante herauszubringen. Es wird sich wohl in Kürze entscheiden, ob der Passagier an Bord ist oder nicht: http://mreinhold.org/blog/late-for-the-train

Andreas Badelt
andreas.badelt@doag.org

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

UPDATE

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

Fortschritte bei JAX-RS 2.0: Neuer Standard für REST-basierte Web-Services für Java

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

JAX-RS 2.0 gehört zusammen mit JPA 2.1 zu den ersten Spezifikationen, die als Teil von Java EE 7 vom Executive Committee des JCP abgesegnet wurden. Die Arbeiten an JAX-RS 2.0, registriert unter JSR-339, haben schon Anfang 2011 begonnen. Seitdem wurden bereits drei Early-Draft-Reviews veröffentlicht, das letzte am 7. Juni 2012. Der Artikel resümiert den aktuellen Stand von JAX-RS und kommentiert einige der wichtigen neuen Features im JAX-RS 2.0 (Early Draft Release 3).

REST-basierte Architekturen und Web-Services erfreuen sich in den letzten Jahren zunehmender Beliebtheit. Der Hauptgrund dafür liegt darin, dass sie eine leichtgewichtigere Interoperabilität im Vergleich zu „SOAP/WS*"-Web-Services ermöglichen und besser zu der HTTP-basierten Infrastruktur des Internets passen. Sowohl Application-Server-Hersteller als auch Cloud-Provider bieten heutzutage standardmäßig REST-APIs an, um ihre Dienste einfacher für eine breite Klasse von Anwendungen in verschiedenen Sprachen wie Java, JavaScript, Perl, PHP, Phyton, Ruby und auf diversen Clients wie PC, Android- oder iPhone-Telefonen verfügbar zu machen.

Auch wenn man REST-basierte Web-Services für Java mit Standard-JDK-Mitteln etwa unter Verwendung von java.net.URL, java.net.HttpUrlConnection etc. implementieren kann, bietet JAX-RS dem Entwickler eine höhere API-Abstraktion und bessere Produktivität an. Die letzte abgeschlossene JAX-RS-Spezifikation, JAX-RS 1.1, ist Bestandteil von Java EE 6 und eine ihrer begehrtesten Technologien, kann jedoch auch außerhalb von Java EE in einem Servlet-Container eingesetzt werden. Es handelt sich um ein „server-side", Annotation-basiertes Framework, das im Wesentlichen HTTP-Requests auf Java-Methoden-Aufrufe abbildet und als eine Art „Domain Specific Language" (DSL) für den Umgang mit dem HTTP-Protokoll angesehen werden kann.

Mit JAX-RS ist es sehr einfach, durch die Verwendung von wenigen Annotations in Java implementierte Anwendungsdienste als REST Web Services zu exponieren. Das folgende Beispiel in Abbildung 1 zeigt, wie HTTP-Requests mit Parametern durch standardisierte JAX-RS-Annotations-Java-Methoden zugeordnet werden und wie ihre Return-Werte wiederum durch die Verwendung entsprechender Serializer für die Generierung der HTTP-Responses genutzt werden.

In diesem Beispiel wird ein „HTTP GET"-Request mit dem URI-Path „/atm/123456/balance?pin=3711" der „balance"-Methode zugeordnet, die mit den Argumenten „card=123456" und „pin=3711" aufgerufen wird. Das Ergebnis wird mit dem Standard-Serializer für String als MIME-Type „text/plain" als HTTP-Response zurückgegeben.

In Tabelle 1 sind die wichtigsten JAX-RS-Begriffe aufgelistet. Tabelle 2 zeigt die Annotations für das Mapping einer Java-Klasse und ihrer Methoden auf eine Web-Resource im Sinne von REST und in Tabelle 3 sind die Annotations für Method-Parameter zur Extraktion der Informationen aus dem HTTP-Request zusammengefasst.

Die Referenz-Implementierung von JAX-RS ist Jersey von Oracle. Jersey wird sowohl im WebLogic- als auch im GlassFish-Application-Server eingesetzt, um ihre Management- und Monitoring-Interfaces als RESTful-Web-Services bereitzustellen. JAX-RS ist sehr erfolgreich, gut verbreitet und hat neben Jersey mehrere alternative Implementierungen wie RESTeasy von JBoss und Apache CXF. Was jedoch bisher in JAX-RS gefehlt hat und von vielen in der Java-Community gefordert wurde, sind:

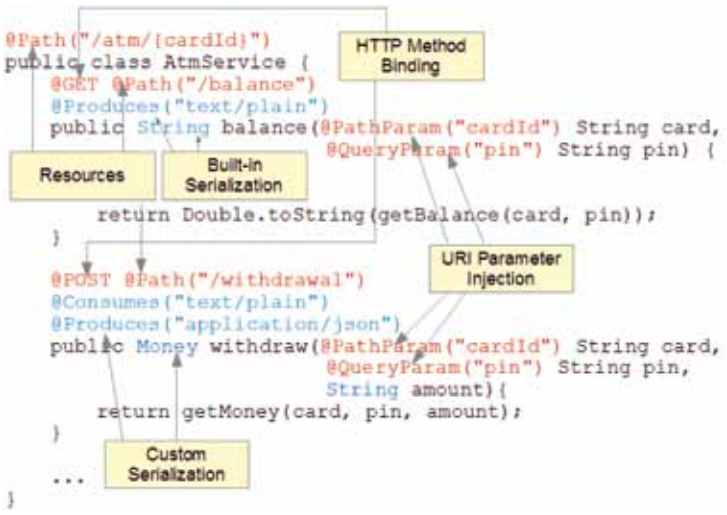


Abbildung 1: Beispiel für die Anwendung des JAX-RS API

Begriff	Bedeutung
Resource class	Java Klasse, die JAX-RS Annotations nutzt, um eine Web Resource zu implementieren.
Provider	Implementiert eine JAX-RS-Erweiterungs-Schnittstelle. Typische Beispiele sind Entity-Provider zur Custom-Serialization, Filter und Interceptoren.
Invocation	Client-API-Objekt, das konfiguriert werden kann, um HTTP-Requests abzusetzen.
Target	Empfänger einer Invocation, identifiziert mit einem URI.

Tabelle 1: JAX-RS-Terminologie

Annotation	Bedeutung
@Consumes	Gibt eine Liste von MIME-Types, die akzeptiert werden.
@Produces	Gibt eine Liste von MIME-Types, die ausgegeben werden.
@Path	Bestimmt den relativen Pfad für eine Resource. Angewendet auf eine Klasse, dient diese als Root-Resource, angewendet auf eine Methode als eine Sub-Resource.
@GET	Die annotierte Methode behandelt HTTP-GET-Requests. Zum Lesen, gegebenenfalls vom Cache.
@POST	Die annotierte Methode behandelt HTTP-POST-Requests. Zur Erstellung/Update einer Entity ohne bekannte ID.
@PUT	Die annotierte Methode behandelt HTTP-PUT-Requests. Zur Erstellung/Update einer Entity mit bekannter ID.
@DELETE	Die annotierte Methode behandelt HTTP-DELETE-Requests. Zur Löschung einer Entity.
@HEAD	Die annotierte Methode behandelt HTTP-HEAD-Requests. Wie GET ohne Response.

Tabelle 2: Abbildung der Standard-HTTP-Methoden und MIME-Types

Annotation	Bedeutung	Beispiel
@PathParam	Extrahiert einen Parameter vom URI	@PathParam(“cardId“)
@QueryParam	Extrahiert den Wert des Query-Parameters vom URI	@QueryParam(“pin“)
@QueryParam	Extrahiert den Wert eines Cookie vom HTTP-Header	@CookieParam(“JSESSIONID“)
@HeaderParam	Extrahiert den Wert eines Headers vom HTTP-Header	@CookieParam(“Accept“)
@FormParam	Extrahiert den Wert eines Form-Parameters	@FormParam(“name“)
@MatrixParam	Extrahiert den Wert eines Matrix-Parameters	@MatrixParam(“name“)

Tabelle 3: Abbildung der HTTP-Request-Parameter

Client-API, „Filter und Interceptor“-Framework, MVC-Framework, Validierung und Unterstützung für Hypermedia, asynchrone Verarbeitung, CDI sowie MIME-Type-Ermittlung. Bis auf das MVC-Framework wurden alle diese Anforderungen in JAX-RS 2.0 adressiert. Der Entwurf eines neuen MVC-Frameworks in JAX-RS wird momentan von der Expert-Group als unpassend empfunden. Da JAX-RS-Resources als Controller angesehen werden können, wird sich JAX-RS 2.0 mehr darauf konzentrieren, alternative View-Technologien zu integrieren. Nachfolgend sind einige der neuen Features in JAX-RS 2.0 näher erläutert.

Client-API

Das neue Client-API, untergebracht im Package „javax.ws.rs.client“, ergänzt das bisherige, rein serverseitige API von JAX-RS 1.1 und ermöglicht einem Client den standardisierten Zugriff auf Web-Resources, die nicht unbedingt mit JAX-RS implementiert sind. Das Ziel ist, effiziente, wiederverwendbare, clientseitige Lösungen zu ermöglichen, die auf bewährten Implementierungstechniken der HTTP-Kommunikation aufsetzen. Das Client-API ist ein „fluent“, low-level API und nutzt das Builder-Pattern. Listing 1 zeigt einen HTTP-GET-Request von „http://.../atm/123456/balance?pin=3711“ (siehe auch Abbildung 1).

Listing 2 zeigt analog ein HTTP-POST-Request von „http://.../atm/123456/withdrawal?pin=3711“ (siehe auch Abbildung 1).

Die JAX-RS-2.0-Client-Runtime würde dabei dieselbe Custom-Serialisierung (implementiert durch die vom Framework vorgesehenen MessageBodyReader/Writer) zur Konvertierung zwischen der Money-Instanz und ihrer JSON-Darstellung nutzen wie auch auf der Server-Seite. Mit dem generischen Command-Interface „Invocation“ unterstützt das Client-API das häufig verwendete Muster der Trennung von Verantwortlichkeiten. Insbesondere die Vorbereitung und Ausführung von HTTP-Requests können getrennt werden, sodass derjenige, der einen synchronen oder asynchronen HTTP-Request absetzten möchte, sich nicht darum kümmern muss, wie er vorab konfiguriert wurde.

Filter- und Interceptoren

Filter und Interceptoren sind Interfaces, die für Fortgeschrittene und Framework-Entwickler vorgesehen sind, um sie in die Lage zu versetzen, an definierten Erweiterungspunkten auf der Client- und Server-Seite die HTTP-Requests und -Responses zu modifizieren. Diese Möglichkeit gab es auch in früheren JAX-RS-1.0/1.1-Implementierungen, wenn auch mit leicht unterschiedlicher Semantik, die nun im Rahmen von JAX-RS 2.0 vereinheitlicht und standardisiert wird.

Die Filter dienen zur Änderung der „Request/Response"-Header oder Return-Codes und sind im aktuellen Early Draft auf der Client- und Server-Seite unterschiedlich. Das hängt damit zusammen, dass auf der Server-Seite mehr Erweiterungspunkte vorgesehen sind, nämlich vor und nach der Zuordnung des Request zur Target-Resource, um diese auch beeinflussen zu können. Mit clientseitigen Filtern kann man beispielsweise Caching-Strategien durch die Verwendung zusätzlicher HTTP-Header implementieren. Serverseitige Filter können genutzt werden, um zum Beispiel HTTP-Methoden zu überschreiben, eine Content-Negotiation auf der Basis des URL-Suffix (.txt, .json, .xml) zu implementieren oder den HTTP-Header mit Cache-Control, Etag, Last-Modified zu dekorieren oder digital zu signieren. Für jeden Erweiterungspunkt lassen sich Filter-zu-Filter-Chains verketten. Die Verwendung der „@BindingPriority"-Annotation legt die Ausführungs-Reihenfolge der Filter in einer Filter-Chain fest.

Die Interceptoren „ReaderInterceptor „und „WriterInterceptor" sind auf der Client- und Server-Seite gleich und werden zur Manipulation des Message-Body genutzt, indem sie die„readTo"-Methode vom „MessageBodyReader" und die „writeTo"-Methode vom „MessageBodyWriter" abfangen (wrappen). Interceptoren vom gleichen Typ lassen sich zu Interceptor-Chains verketten und wie bei Filter-Chains wird ihre Ausführungs-Reihenfolge über „@BindingPriority" bestimmt. Anders als bei einem Filter jedoch muss ein Interceptor explizit die „proceed"-Methode vom Context aufrufen, um die Interceptor-Ausführung in der Interceptor-Chain fortzusetzen. Das folgende Beispiel in Listing 3 zeigt einen rudimentären GZIP-Interceptor, der den HTTP-Message-Body am Sende-Endpunkt komprimiert beziehungsweise am Empfänger-Endpunkt dekomprimiert.

Validierung

Zur besseren Code-Qualität in JAX-RS-Anwendungen ist es sinnvoll, die Request- und Response-Daten zu validieren. Momentan muss diese Funktionalität im Anwendungs-Code programmiert sein. In JAX-RS 2.0 wird die Bean Validation Specification (JSR-330) genutzt, um deklarativ und standardisiert Restriktionen für die

```
Client client = ClientFactory.newClient();
String balance = client.target("http://.../atm/{cardId}/ba-
lance")
                       .pathParam("cardId", "123456")
                       .queryParam("pin", "3711")
                       .request("text/plain")
                       .get(String.class);
```

Listing 1

```
Client client = ClientFactory.newClient();
Money balance = client.target("http://../atm/{cardId}/with-
drawal")
                       .pathParam("cardId", "123456")
                       .queryParam("pin", "3711")
                       .request("application/json")
                       .post(text("100"), Money.class);
```

Listing 2

```
@Provider
class GzipInterceptor implements ReaderInterceptor, Writer-
Interceptor {
    @Override
    Object aroundReadFrom(ReaderInterceptorContext ctx)
throws … {
        InputStream old = ctx.getInputStream();
        ctx.setInputStream(new GZIPInputStream(old));
        try {
            return ctx.proceed();
        } finally {
            ctx.setInputStream(old);
        }
    }

    @Override
    void aroundWriteTo(WriterInterceptorContext context)
throws … {
        OutputStream old = ctx.getOutputStream();
        GZIPOutputStream gzipOutputStream = new
GZIPOutputStream(old);
        ctx.setOutputStream(gzipOutputStream);
        try {
            ctx.proceed();
        } finally {
            gzipOutputStream.finish();
            ctx.setOutputStream(old);
        }
    }
    …
}
```

Listing 3

Daten, die aus Query-, Header-Parameter und Entity-Bodies extrahiert oder von Resource-Methoden zurückgegeben werden, zu definieren (siehe Listing 4).

Die „@NotNull"-Annotation stammt vom Bean-Validation-API und stellt sicher, dass die Form-Parameter „firstName" und „lastName" nicht „null" sind. Die „@StandardUser"- und „@Email"-Annotations sind Custom-Annotations, deren Validierungs-Klassen vom Anwendungsentwickler nach den Richtlinien von JSR-330 bereitgestellt werden. „@Valid" ist eine vordefinierte Bean-Validation-Annotation, die bewirkt, dass alle Restriktionen für den User, der aus dem Request-Body extrahiert wurde, erfüllt sind, und veranlasst somit die Überprüfung, die hinter „@StandardUser" steckt.

Falls die deklarierten Restriktionen nicht erfüllt sind, wird die zugehörige Resource-Methode nicht ausgeführt und ein Response mit dem Status-Code 400 (bad request) sowie die Liste der festgestellten Verletzungen werden dem Client zurückgegeben.

```
@StandardUser
class User { ... }

@Path("/users")
class UsersResource {
      @POST
      @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
      public void registerUser(
            @NotNull @FormParam("firstName") String first-
Name,
            @NotNull @FormParam("lastName") String last-
Name,
            @Email @FormParam("email") String email) {
                  ...
            }

      @POST
      @Consumes(MediaType.APPLICATION_JSON)
      public void registerUser(@Valid User user) {
            ...
      }
 }
```

Listing 4

```
// Server API
Response res = Response.ok(order)
      .link("http://.../orders/1/ship", "ship")
      .build();

// Client API
Response order = client.target(…)
      .request(„application/xml").get();
if (order.getLink("ship") != null) {
      Response shippedOrder = client
            .target(order.getLink("ship"))
            .request("application/xml").post(null);
      …
}
```

Listing 5

Hypermedia

Die Verlinkung von Resources ist eines der wichtigsten REST-Prinzipien, auch „Hypermedia As The Engine Of Application State" (HATEOS) genannt. Es wird zwischen Struktur (structural)- und Übergangs (transitional)-Links unterschieden. Struktur-Links vermeiden, dass die komplette Repräsentation einer komplexstrukturierten Resource übertragen wird, und ermöglichen Lazy-Loading. Der Client kann solche Links verfolgen, um an die gewünschten Detail-Informationen heranzukommen. Übergangs-Links werden genutzt, um den Zustand einer Resource zu aktualisieren oder zum nächsten Anwendungszustand zu gelangen. JAX-RS 2.0 unterstützt nur Übergangs-Links. Das Beispiel in Listing 5 verdeutlicht, wie in einem üblichen Szenario auf der Server-Seite ein Übergangs-Link erstellt und auf dem Client konsumiert wird.

Serverseitig wird der angeforderten Bestellung eine Bestellnummer zugeordnet und der speziell benannte Link zur Versendung als Header-Parameter in der Response zurückgeschickt. Falls der Client in der Response von seiner Bestellung einen Link mit dem erwarteten Namen „ship" vorfindet, folgt er dem Link, um ihre Versendung anzustoßen.

Asynchrone Verarbeitung

JAX-RS 2.0 unterstützt die asynchrone Verarbeitung sowohl auf der Client- als auch auf der Server-Seite und ermöglicht somit die effizientere Nutzung von Threads. Ein Thread auf dem Client, der einen HTTP-Request abgesetzt hat, könnte auch für Updates der Benutzeroberfläche zuständig sein. Falls dieser Thread blockiert wird, während er auf Antwort vom Server wartet, würde die vom Benutzer empfundene Performance der Anwendung darunter leiden. Ähnlich ist es auf der Server-Seite: Ein Thread, der gerade einen Request verarbeitet, indem er auf das Ergebnis einer Datenbank-Abfrage wartet, wäre für andere, gleichzeitig ankommende Requests von anderen Clients nicht verwendbar und das könnte schnell die Verfügbarkeit des angebotenen Service gefährden. Deshalb ist diese neue Möglichkeit der asynchronen Verarbeitung wichtig, um hochverfügbare und hoch performante Anwendungen mit JAX-RS bauen zu können.

Bei der asynchronen Verarbeitung auf der Server-Seite informiert die Resource-

Methode die JAX-RS Runtime über die „@Suspend"-Annotation, dass das Ergebnis nicht beim Return der Methode, sondern später geliefert wird, sodass die Client-Verbindung zunächst angehalten werden kann. Die Verarbeitung selbst wird in einem neuen Thread ausgeführt und, wenn abgeschlossen, wird das Ergebnis über „resume" des injizierten „ExecutionContext" von der JAX-RS-Runtime und über die vorher angehaltene Client-Verbindung an den Client ausgeliefert, wie das Beispiel in Listing 6 zeigt. Falls die Resource-Klasse in diesem Beispiel mit einer EJB implementiert wird – was in JAX-RS durchaus möglich ist –, könnte man sich das explizite Thread-Management sparen und es dem EJB-Container überlassen (siehe Listing 7).

Auf der Client-Seite kann eine asynchrone Verarbeitung veranlasst werden, wenn beim Request-Aufbau die „async"-Methode aufgerufen wird. Dann wird sofort ein „Future"-Objekt zurückgegeben, das zum Monitoring, zur Extraktion des Ergebnisses oder zur Stornierung des Request verwendet werden kann (siehe Listing 8). Durch Nutzung der Asynchronität in JAX-RS 2.0 können Client- und Server-Entwickler unabhängig voneinander ihre Thread-Resources entsprechend ihren Bedürfnissen optimieren und müssen nicht wissen, ob der Aufruf auf der anderen Seite der Verbindung synchron oder asynchron ist.

```
@Path("/async/longRunningOp")
class DbResource {
    @Context
    ExecutionContext ctx;

    @GET
    @Suspend
    public void longRunningOp() {
        Executors.newSingleThreadExecutor().submit(
                new Runnable() {
                    public void run() {
                        String result = longQueryFromData-
base();
                        ctx.resume(result);
                    }
                });
    }
}
```

Listing 6

```
@Stateless
@Path(«/async/longRunningOp»)
class DbResource {
    @GET
    @Suspend
    @Asynchronous
    public void longRunningOp(@Context ExecutionContext ctx)
{
        String result = longQueryFromDatabase();
        ctx.resume(result);

    }
}
```

Listing 7

```
String mybalance;
Client client = ClientFactory.newClient();
Future<String> result = client.target("http://.../atm/{car-
dId}/balance")
                        .pathParam("cardId", "123456")
                        .queryParam("pin", "3711")
                        .request("text/plain")
                           .async()
                         .get(String.class);
…
if (result.isDone())
   mybalance = result.get();
```

Listing 8

Fazit

Weil JAX-RS 2.0 als Bestandteil des Web-Profile in Java EE 7 vorgesehen ist (JAX-RS 1.1 ist nicht im Web-Profile von Java EE 6 enthalten), ist in Zukunft mit einer noch größeren Verbreitung dieser nützlichen Technologie zu rechnen.

Peter Doschkinow
peter.doschkinow@oracle.com

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

Links

1. JSR-339: http://jcp.org/en/jsr/detail?id=339
2. JAX-RS-2.0-Spezifikation: http://download.oracle.com/otn-pub/jcp/jaxrs-2_0-edr3-spec/jaxrs-2_0-edr3-spec.pdf
3. JAX-RS-2.0-API und Java-Docs: http://download.oracle.com/otn-pub/jcp/jaxrs-2_0-edr3-spec/jaxrs-2_0-edr3-api.zip
4. JAX-RS-Referenz-Implementierung Jersey: http://jersey.java.net/
5. GlassFish mit integrierter Jersey-Implementierung von JAX-RS: http://glassfish.java.net

Java EE 7:
Entwickeln für die Wolken

Markus Eisele, msg systems ag

Java EE 7 wirft seine Schatten voraus. Erste Ideen und Ziele wurden bereits auf der JavaOne 2011 vorgestellt. Mittlerweile sind schon mehrere enthaltene Spezifikationen mit neuen Java Specification Requests (JSR) im Java Community Process (JCP) an den Start gegangen. Der Artikel gibt einen Überblick über die Ziele, den Zeitplan sowie den aktuellen Stand der Spezifikationen und erläutert Neuheiten.

Bereits auf der JavaOne 2011 wurde über die Inhalte der neuen Java-EE-7-Spezifikation gesprochen. Mit dem Oberthema „Cloud" konnte bereits früh ein erster Eindruck von der generellen Richtung gewonnen werden. Mehr als ein Jahr danach sind die Bemühungen zur Finalisierung in vollem Gang und die Details sind auch nicht mehr so vage wie noch im letzten Jahr. Java EE 6 wurde im Dezember 2009 unter der Leitung von Roberto Chinnici erfolgreich beendet. Die Referenz-Implementierung GlassFish stand pünktlich bereit. Im März 2011 nun startete der JSR 342 (Java EE 7) offiziell unter der Leitung von Linda DeMichiel. Nach Chinnicis Ausscheiden aus Oracle hat die langjährige JPA Leaderin nun die Führung des nächsten Prestige-Objekts von Oracle übernommen.

Abbildung 1: Überblick Cloud-Begrifflichkeiten
© msg systems ag

Entwickeln für die Wolken

Das Hauptthema von Java EE 7 ist „die Cloud". Auch wenn bereits seit Längerem einige Unternehmen Java EE im Cloud-Umfeld erfolgreich einsetzen, blieben bisher noch viele Probleme ungelöst. Das Ziel ist, dass sich Java-EE-7-basierte Produkte vereinfacht in privaten oder öffentlichen Clouds einsetzen lassen und ihre Dienste sowohl horizontale Skalierung (Elastizität) als auch Mandantenfähigkeit besser unterstützen. In Platform-as-a-Service-(PaaS)-Modellen können Anwendungen von Kunden in der Cloud bereitgestellt werden. Diese profitieren dabei von Leistungen der Hosting-Umgebung, die parallel auch anderen Anwendungen anderer Kunden zur Verfügung stehen – mit den entsprechenden Garantien in Bezug auf Sicherheit, Isolation, Quality of Service etc.

Seit den ersten Versionen hat sich Java EE als eine verwaltete Umgebung ausgezeichnet, in der Zugriffe auf Betriebssysteme und andere externe Ressourcen, wie beispielsweise relationale Datenbanken, über detaillierte Container-Konzepte gekapselt zur Verfügung stehen. Dieses Container-basierte Modell ermöglicht es bereits heute, dass portable Anwendungen auf einzelnen Rechnern oder auch für große Cluster-Installationen ohne grundlegende Änderungen an der Programmierung verwendet werden können. Java EE 7 ist in diesem Sinne eine konsequente Weiterentwicklung dieser Paradigmen und umfasst daher voraussichtlich nur einige inkrementelle Änderungen an den bestehenden (und beliebten) Java-EE-Paradigmen. Dabei wird Java EE 7 die bestehende Architektur um betriebliche Aspekte von Clouds erweitern. Neben der Einführung neuer Rollen, beispielsweise des PaaS-Administrators, steht auch die Überarbeitung der Sicherheits-Architektur im Fokus.

Alle bereits in Vorgänger-Versionen verfügbaren Technologien sollen dabei um die Kernprinzipien der Cloud erweitert werden. Dabei sind speziell solche Technologien davon betroffen, die Zugriffe auf Ressourcen bereitstellen (JPA, JDBC, JMS). Darüber hinaus sind weitergehende Meta-Informationen geplant, um die Eigenschaften von Anwendungen bezogen auf ihr Verhalten in der PaaS-Landschaft zu beschreiben. Auch das Thema „Modularität von Enterprise-Java-Anwendungen" soll erneut beleuchtet werden. Besonders wichtig ist die Tatsache, dass es sich bei Java EE 7 um eine Datums-getriebene Version handelt. Was bis Q1 (vielleicht auch frühes Q2) 2013 nicht fertig ist, wird erst in Java EE 8 einfließen.

Java EE 7 im XaaS-Dschungel

Ausgehend von der allgemeinen Definition des National Institute of Standards and Technology (NIST) fokussiert sich EE 7 auf die Funktionalitäten und Fähigkeiten mit Hinblick auf den sogenannten „Platform-as-a-Service-Bereich". Im Unterschied zu

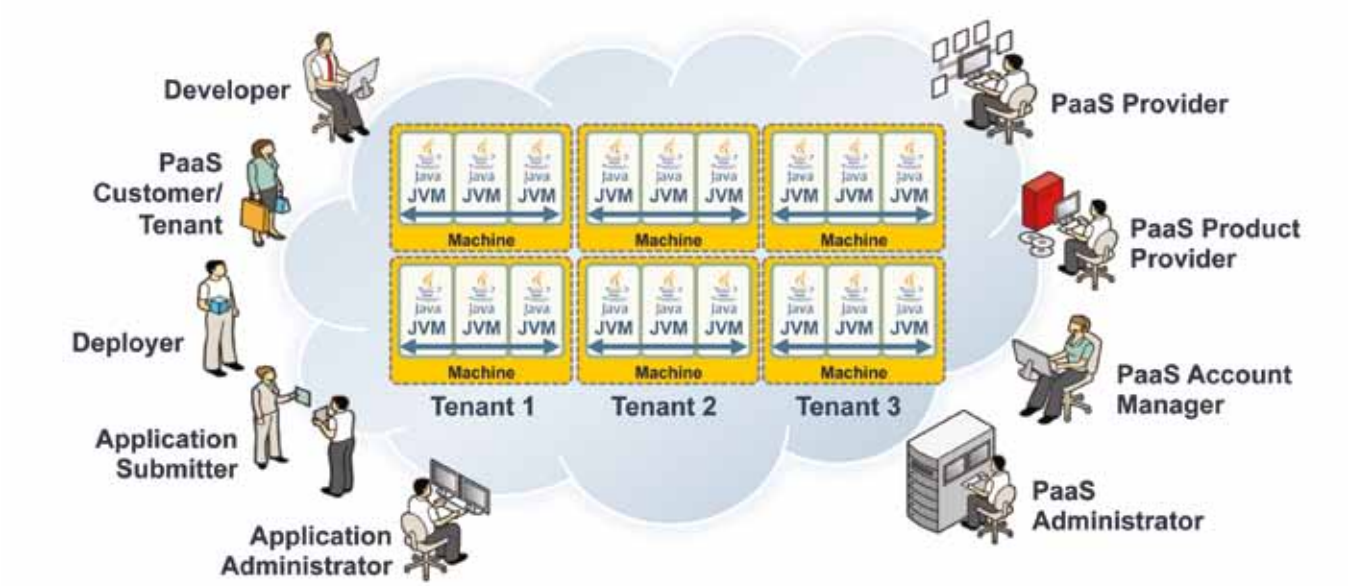


Abbildung 2: Java EE 7 Platform Roles © Oracle

einem klassischen Infrastucture-as-a-Service (IaaS), wo es nur um Basis-Strukturen (CPU, Speicher, Netzwerk) geht, soll mit dem PaaS-Ansatz eine integrierte Laufzeitumgebung für Java-EE-Anwendungen bereitgestellt werden (siehe Abbildung 1).

Dies ist klassisch bereits von anderen Web-Hosting-Plattformen bekannt. Wichtigste Merkmale sind dabei die vergleichsweise restriktiven Freiheiten für Anwender. Sie haben zwar volle Kontrolle über ihre Anwendungen, die Konfigurations-Einstellungen der zugrunde liegenden Container sind allerdings zum großen Teil nur eingeschränkt veränderbar. Ein physikalischer Zugriff auf die Basis-Infrastruktur ist zumeist gar nicht möglich. Der Vorteil ist, dass sich die Nutzer von PaaS-Angeboten nicht mehr explizit um Datenbank-Zugriffe, Skalierbarkeit, Zugriffskontrollen etc. kümmern müssen, da diese bereits von der Plattform bereitgestellt sind. Die Abgrenzung zu SaaS steht mit der getroffenen Definition außer Frage. Leider ist das nicht uneingeschränkt so. Auch wenn sich die Plattform-Spezifikation auf PaaS fokussiert, finden sich bei vielen enthaltenen Spezifikationen wie dem Java Persistence API (JPA) durchaus auch SaaS-Ansätze. In der Spezifikation findet sich das dann als „Basis-Form von SaaS" wieder (EE.2.11.1). Vergleichsweise vollständig ausgearbeitet ist das neue Rollenmodell für die Plattform (siehe Abbildung 2).

Die erwartete Zweiteilung zwischen Nutzern auf der linken und PaaS-Anbietern auf der rechten Seite ist nachvollziehbar. Die Trennung auf Nutzerseite ist auch angelehnt an bereits Bekanntes. Dennoch ändert sich damit die Arbeit mit der Laufzeitumgebung erheblich. Wurden bis EE 6 notwendige Ressourcen konfiguriert (Datenbank, LDAP, Messaging) und anschließend die Anwendung „deployed", verteilen sich diese Aufgaben nun auf mehrere Schultern. Neben den Anwendungen müssen jetzt auch die notwendigen Ressourcen provisioniert werden, da Entwickler und Administratoren auf Kundenseite keinen direkten Zugriff mehr auf die Laufzeit-Umgebung haben. Beim Vergleich aktueller Angebote von Anbietern wie Jelastic, OpenShift oder auch Googles Appengine bekommt man einen ersten Eindruck von dem veränderten Modell. Für die Entwickler bleibt allerdings erst einmal alles beim Alten. Auch die drei genannten Kandidaten unterstützen ein vergleichsweise leichtgewichtiges Entwickeln und bieten einen guten Werkzeugkasten, um die gefühlt hohen Auswirkungen von PaaS- und SaaS-Konzepten auf die Anwendungsentwicklung auf ein erträgliches Maß zu reduzieren.

Selbstverständlich ist, dass die EE 7 jetzt SE 7 voraussetzt und bereits Metadaten für Ressourcen beinhaltet (DataSourceDefinition, JMSConnectionFactoryDefinition, JMSDestinationDefinition, MailSessionDefinition, ConnectorResourceDefinition).

Neben einer Standard Data Source (java:comp/defaultDataSource) wird es auch eine Default JMS Connection Factory (java:comp/defaultJMSConnectionFactory) geben. Eine Mandanten-Identifikation lässt sich zukünftig via java:comp/tenantId beziehen. Auch dem „Pruning" gegenüber, also dem Entfernen von veralteten Spezifikationsumfängen, zeigt sich die aktuelle Expert Group offen. EJB Entity Beans, EJB QL, JAX-RPC, Deployment API und JAXR werden nicht mit in die moderne Welt übernommen und sollen verschwinden.

Vereinfachte Entwicklung

Die Überschrift über der letzten Spezifikationsversion EE 6 soll unvermindert gelten. Nachdem sich in den vergangenen Jahren deutlich Optimierungspotenzial bei der Zusammenarbeit einzelner Spezifikationen im Umfeld der Dependency Injection gezeigt hat, sind hier alle gefragt, entsprechend tätig zu werden. Auch das Vorhandensein verschiedener Managed Beans in der Plattform hat immer wieder zu Diskussionen darüber geführt, welche noch für EE 7 gelöst werden sollen. Aktuell finden sich nur vereinzelt Beispiele in den Bug-Trackern der beteiligten Spezifikationen, aber es bleibt die Hoffnung, dass mehrere der aus EE 6 bekannten Ungereimtheiten verschwinden. Darüber hinaus wird schon im Rahmen der

Java EE 7, die auf Java SE 7 aufbaut, darauf geachtet, dass die mit Java SE 8 erwarteten Sprachänderungen soweit möglich schon heute berücksichtigt werden. Ziel ist es, dass Entwickler die neuen Features der Sprache auch in EE 7 bereits dann nutzen können, wenn SE 8 verfügbar ist.

Ganz offensichtlich ist das bisher Umschriebene vergleichsweise dünn vor dem Hintergrund der notwendigen Änderungen von EE 6 hin zum PaaS-getriebenen EE 7. Leider sind bisher weitergehende Überlegungen noch recht selten in der Expert Group. Neben einem detaillierteren Dokument zum Thema „Security Manager" und einem Konzept über die „Credential und SSL"-Konfiguration in EE 7 gibt es nach aktuellem Stand kaum mehr Substanzielles, über das es sich zu berichten lohnt. Hier bleibt also abzuwarten, wie die fertige Version aussehen wird.

Updates von Spezifikationen

Der EE-7-Schirm ist ja nur ein Teil des Ganzen. In Summe sind wieder mehr als 30 einzelne Spezifikationen Bestandteile. Und jede wird ihre Auffassung von PaaS und SaaS zum Gesamt-Mix hinzufügen. Abbildung 3 zeigt einen Überblick über die wichtigsten Erweiterungen bestehender JSRs, aber auch über die neu hinzugekommenen.

Allen voran werden neue Web-Standards Einzug in die Java-EE-Plattform halten. Neben einer erstklassigen Unterstützung für HTML 5 sollen das vor allem auch WebSockets („JSR 356: Java API for WebSocket") und ein modernes http-Client-API sein (Teil von „JSR 339: JAX-RS 2.0, The Java API for RESTful Web Services"). Der HTML-5-Anteil wird dabei komplett von den Java Server Faces („JSR 344: JavaServerTM Faces 2.2") aufgefangen. Neben entsprechenden Formularen sollen hier das neue Sektionskonzept sowie entsprechende Metainformationen möglich werden. Endlich wird es eine standardisierte Upload-Komponente geben und auch hier kann Dependency Injection jetzt in allen JSF-Artefakten verwendet werden – dies umfasst auch Konverter und Validatoren. Das Programmiermodell wird noch erweitert und bietet die Möglichkeit einer vollständig programmatischen Erstellung einer JSF-Anwendung inklusive Custom Components.

Nahe dran an den neuen Web-Standards ist der „JSR 353: Java API for JSON Processing". Lange musste gezittert werden, ob es tatsächlich zu diesem JSR kommt. Seit Ende letzten Jahres liegt er vor und stellt zukünftig mit JSONP (jsonp.java.net) die Referenz-Implementierung für die Arbeit mit JSON in Java. Neben einem Objektmodell (ähnlich DOM) gibt es auch ein Streaming-API (ähnlich StAX). JsonBuilder, JsonReader und JsonWriter werden um JsonGenerator und JsonParser angereichert.

Für „JSR 338: Java Persistence (JPA) 2.1" stehen einige kleinere Änderungen auf dem Plan. Bisher bekannt sind die bessere Integration von Stored Procedures (@NamedStoredProcedureQuery, @StoredProcedureQuery) sowie die Erweiterung des Criteria-API um Massendaten-Funktionen (update/delete). Benutzerdefinierte Datenbank-Funktionen werden ebenfalls aus JPA heraus benutzbar sein. Dass das Mandanten-Konzept angelehnt an EclipseLink ebenfalls Bestandteil der Spezifikation wird, bleibt stark anzunehmen, auch wenn es im Early Draft Review bisher nicht enthalten ist.

Abbildung 3: Heiße Kandidaten © Oracle

Größte Überraschung ist sicherlich der JSR 339. JAX-RS bekommt mit der Major-Versionsnummer 2.0 nicht nur optisch eine Überarbeitung, inhaltlich tut sich hier ebenfalls unglaublich viel. Das neue Client-API ist das prominenteste Beispiel (siehe Listing 1). Von ihm gibt es auch eine asynchrone Variante. Zusätzlich stehen serverseitig jetzt neue Funktionen zur Verfügung. Filter und Interceptors sind nur der Anfang. Eine Integration in JSR 349 Bean Validation 1.1 ist endlich vorhanden und auch die serverseitige Content Negotiation ist integriert.

Der „JSR 346: Contexts and Dependency Injection for Java EE 1.1" versteht sich als einfache Auffrischung und führt doch nach aktuellem Early-Draft-Review einige Erweiterungen und Änderungen ein. Ein „embedded mode" dient zum Starten außerhalb eines Java-EE-Containers. Die Reihenfolge von Interceptors und Decorators lässt sich jetzt global verändern. Die Integration mit dem „JSR 340: Java Servlet 3.1 Specification" wird verbessert, unter anderem durch die Möglichkeit, den BeanManager aus dem ServletContext zu beziehen, den ServletContext zu injizieren sowie Conversations in Servlets zu verwenden. Beans können deklarativ mit einem „@Veto" versehen sein sowie ihre Attribute mithilfe von BeanAttributes überschrieben werden.

Besonders hervorzuheben ist der „JSR 349: Bean Validation 1.1". Hier hat die Expert Group sehr offen mit der Gemeinschaft zusammengearbeitet und konsequent die Anfänge zu einem runden Gesamtpaket weiterentwickelt. Listing 2 zeigt die wichtigste neue Funktion, die Methoden-Validierung.

Auch der „JSR 343: Java Message Service 2.0" ist etwas Besonderes. Nach fast zehn Jahren wurde das API vollständig überarbeitet. Deutlich weniger redselig, weniger zu verwendende Objekte, Integration mit Java 7 (Autocloseable) und Zusammenarbeit mit JSR 346 führen zu erkennbar gesteigerter Produktivität und lesbarerem Sourcecode.

In der Sammlung fehlen noch ein paar Spezifikationen. Neben JAX-RS 2.0 (JSR 339) sind das Servlet 3.1 (JSR 340), Expression Language 3.0 (JSR 341) und EJB 3.2 (JSR 345). Auch hier tut sich einiges. Die Early-Draft-Review-Versionen sind bereits verfügbar und ein kurzer Blick bestätigt Änderungen, Vereinfachungen, Verbesse-

rungen. Allerdings jeweils im Rahmen kleinerer Weiterentwicklungen. Bisher noch offen sind Nachfolger oder Wartungs-Releases für folgende Spezifikationen:

- Dependency Injection for Java 1.1 (JSR 330)
- Commons Annotations 1.2 (JSR 250)
- JAX-WS 2.3(JSR 224)
- JTA 1.2 (JSR 901)
- Connector Architecture 1.7 (JSR 322)

Neue Technologien

Als neue Technologien werden voraussichtlich die folgenden neuen Spezifikationen Einzug in die Java-EE-Plattform halten:

- Concurrency Utilities 1.0 (JSR 236)
- JCache – Java Temporary Caching API 1.0 (JSR 107)
- Data Grids for the Java Platform (JSR 347)

Die hier aktuell verfügbaren Informationen sind allerdings durchweg überschaubar. Im Rahmen des JCP liegt kein offizielles Dokument vor. Ausnahmen sind zum einen der JSR 107. Beim ältesten, offenen JSR (aus dem Jahr 2001) gibt es durchweg Arbeitsergebnisse auf GitHub (https://github.com/jsr107/) und als Google Doc. Auch eine Referenz-Implementierung liegt bereits als 0.6-SNAPSHOT vor. Den JSR 347 findet man ebenfalls auf GitHub (https://github.com/datagrids/). Das Wiki bildet hier den Ansatzpunkt für weitere Recherchen. Die Concurrency Utilities werden als Java.net-Projekt (http://java.net/projects/concurrency-ee-spec/) konzipiert. Seit Juni 2012 liegt ein Early Draft Candidate im Download-Bereich vor. Ob und wann die erfolgte Arbeit aller drei JSRs offiziell in den JCP überführt wird, ist offen.

```
Client client = ClientFactory.newClient();
String balance = client.target("http://.../atm/balance")
.request("text/plain")
.get(String.class);
```

Listing 1

```
@MethodValidated
public class OrderService {
public OrderService(@NotNull CreditCardProcessor creditCard-
Processor) { . . . }
public void placeOrder(
@NotNull
@Size(min=3, max=20) String customerCode,
@NotNull @Valid Item item,
@Min(1) int quantity) { . . . }
@NotNull @Size(min=1)
public Set<CreditCardProcessor> getCreditCardProcessors() {
... }
@NotNull @Future
public Date getNextAvailableDeliveryDate() { ... }
}
```

Listing 2

Fazit

Es bewegt sich einiges im Java-EE-Bereich. Der Regenschirm (Umbrella) JSR gibt die Richtung vor und alle anderen folgen. So zumindest die Theorie. Schaut man auf die tatsächlichen Änderungen der einzelnen JSRs, wird man den Eindruck nicht los, als wäre das Cloud-Thema noch nicht so richtig bei JSF, JPA, EJB & Co. angekommen. Es bleibt also ein gemischtes Gefühl beim aktuellen Blick auf den Standard. Aber vielleicht lässt sich das auch mit der PaaS-Ausrichtung erklären. Cloud im Kontext einzelner Spezifikationen würde ja viel mehr SaaS unterstützen und genau das ist ja aktuell nicht im Fokus.

Es bleibt also spannend. Ein abschließendes Urteil kann man sich wohl erst nach Abschluss der Arbeiten an der aktuellen Plattform machen.

Auch politisch hat sich in den Jahren seit der Übernahme von Sun vieles getan. Der JCP wird offener und auch die Java-EE-JSRs sollen da keine Ausnahme darstellen. Alle EG-Mailinglisten und Issue-Tracker sind öffentlich zugänglich und die einzelnen JSRs sollen auch die JCP-Version 2.8 übernehmen, sobald diese vorliegt. Parallel zur Weiterentwicklung der Plattform selber ist auch der GlassFish 4.0 als Referenz-Implementierung in ersten Promoted Builds verfügbar. Hier lassen sich erste Integrationen anschauen und ausprobieren.

Links

1. Java EE Spec Downloads: http://java.net/projects/javaee-spec/downloads
2. JSR 342 Early Draft Review: http://java.net/downloads/javaee-spec/JavaEE_Platform_Spec_EDR.pdf
3. GlassFish 4.0 Promoted Builds: http://dlc.sun.com.edgesuite.net/glassfish/4.0/promoted/

Markus Eisele
markus.eisele@msg-systems.com

Markus Eisele arbeitet bei der msg systems ag in München. Er ist Java EE 7 EG und GlassFish Community Member und betreibt einen englischsprachigen Blog über Java EE, GlassFish und WebLogic unter http:// blog.eisele.net

Mit JavaFX entwickeln

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

Oracle bietet JavaFX als strategische Technologie für die Entwicklung von Client-Web-Anwendungen innerhalb der Java-Plattform für den Desktop. JavaFX 2 erfährt eine Renaissance mit nachhaltiger Verstärkung der zuständigen Entwicklungs-Teams und klar definierter Roadmap für die kommenden Versionen mit dem Ziel, JavaFX 3.0 im Java SE Development Kit (JDK) 8 zu paketieren und auszuliefern. Die aktuelle Version JavaFX 2.1 wird bereits mit dem JDK 7u4 und höheren JDK-Versionen für MS Windows und Mac OS X angeboten.

Die eigens für JavaFX 1.0 geschaffene Skriptsprache ist mit JavaFX 2.0 vollständig verschwunden. Man setzt jetzt auf eine einheitliche Java-Entwicklung mit neuer JavaFX-Bibliothek für den Desktop. Das Augenmerk liegt dabei auf verbesserter Browser-Integration, dem UI-Styling, Animationen und höherer Geschwindigkeit der Grafikverarbeitung. Die verfügbaren APIs sind vereinfacht worden, sodass sie den Java-Entwicklungsprozess mit seinen Werkzeugen besser unterstützen. Damit gestaltet sich der Einsatz von JavaFX deskriptiv und programmatisch und führt zu einer schnelleren Umsetzung bei der Entwicklung von Desktop-Anwendungen. JavaFX 2.0 liefert einige UI-Controls (Charts, Tabellen, Menüs) und ein API für

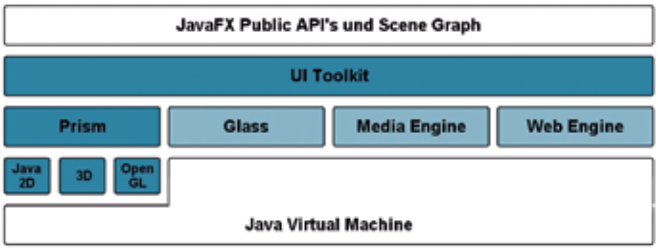


Abbildung 1: JavaFX-Architektur

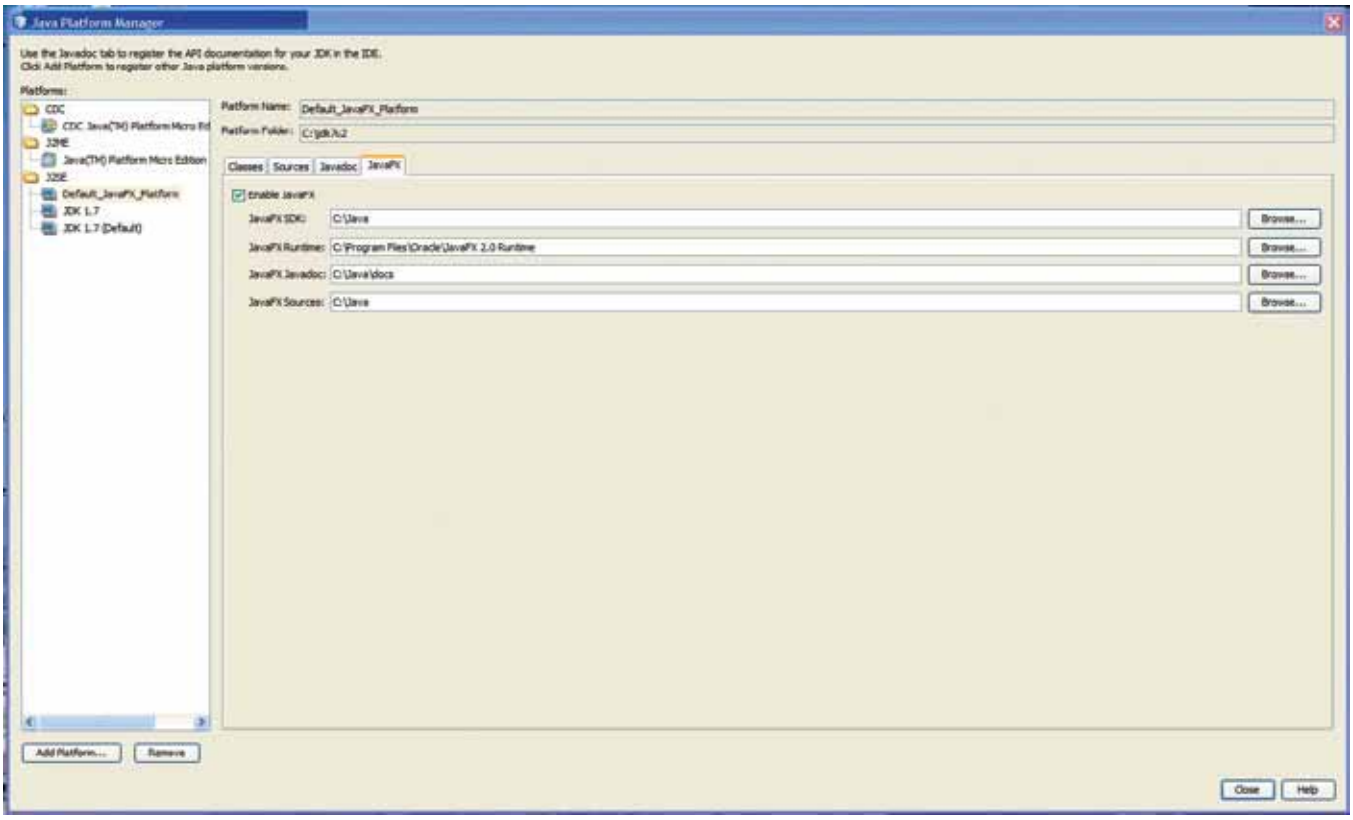


Abbildung 2: NetBeans-Java-Plattform-Manager mit JavaFX

selbsterstellte Controls mit der Absicht, weitere folgen zu lassen, um den konkreten Bedarf an ausgereifter Desktop-Funktionalität zu decken und langfristig Swing abzulösen.

Java als Programmiermodell für JavaFX 2

Die Einführung von Java APIs für JavaFX bietet mit Generics, Annotationen und Multithreading die gewohnte Java-Programmiermodell-Unterstützung mit vorhandenen Sprachmerkmalen an. Beim Design der APIs versuchte man bewusst eine Alternative zu dynamisch typisierten Sprachen zu schaffen. Die Funktionalität von JavaFX wird dem Entwickler über die APIs direkt angeboten, um vorhandene Java-Werkzeuge für Code Refactoring, Debugging und Profiling auch für die Erstellung von JavaFX-Anwendungen nutzen zu können.

Architektur für schnelle Grafikverarbeitung

Die neue JavaFX-Grafik-Engine unterstützt moderne Grafikprozessoren (Graphics Processing Units). Die Hardwarebeschleunigte Grafik-Pipeline „Prism" bildet das Fundament der Grafik-Engine für schnelles Rendering von 2D- und 3D-Elementen (siehe Abbildung 1). Web-Seiten, die in einer JavaFX-Anwendung als Web-Komponente eingebunden sind, werden mittels Web-Kit-HTML-Rendering-Technologie und Prism dargestellt. Prism spielt mit dem Glass-Windowing-Toolkit zusammen. Damit macht es die Darstellung umfangreicher grafischer Oberflächen schneller und wird in getrennten Threads für die Anwendung und das Rendering im Hintergrund synchronisiert, ohne dass es der Entwickler bemerkt. Ein JavaFX-Plug-In ermöglicht das Laden von Prism-basierten Applets im Browser. Die neue JavaFX Media Engine unterstützt den schnellen Ablauf von „Web Multimedia Content Video und Audio" in der Playback-Funktion mit dem GStreamer-Multimedia-Framework.

Entwicklungsumgebungen und Scene Builder

JavaFX lässt sich in nahezu jeder Java-Entwicklungsumgebung verwenden. Zur Auswahl stehen NetBeans, Eclipse IDE über das JavaFX-Eclipse-Integrationsprojekt „e(fx)clipse", IntelliJ IDEA und JDeveloper/ADF. JavaFX-Unterstützung besteht auch für Groovy über das GroovyFX-API und für Scala mit ScalaFX (Scala Bindings for JavaFX 2). Die Verwendung von JavaFX in der Entwicklungsumgebung NetBeans mit dem Windows-XP-Betriebssystem erfolgt durch Konfiguration mit dem Java-Plattform-Manager über die Definition einer Java-Plattform-Version (JDK 6 oder JDK 7) und Einbindung der JavaFX 2.0 Runtime mit der Datei „fxrt.jar" aus dem Verzeichnis „\JavaFX 2.0 Runtime\lib" (siehe Abbildung 2).

```
package javafxapplication1;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class JavaFXApplication1 extends Application {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Hello World!");
        Button btn = new Button();
        btn.setText("Say, 'Hello World'");
        btn.setOnAction(new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello Alexander!");
            }
        });

        StackPane root = new StackPane();
        root.getChildren().add(btn);
        primaryStage.setScene(new Scene(root, 300, 250));
        primaryStage.show();
    }
}
```

Listing 1

Durch Einführung der neuen XML-basierten, deklarativen Markup-Sprache FXML kann das Benutzer-Interface der JavaFX-Anwendung unabhängig von der jeweils verwendeten Entwicklungsumgebung aufgebaut werden, ohne IDE-spezifische Fragmente nachpflegen zu müssen. Bei Layout-Änderungen ist das vorteilhaft, da keine erneute Code-Kompilierung notwendig ist.

Der neue JavaFX Scene Builder 1.0 als Beta-Version ist ein visuelles Layout-Werkzeug zum Design von JavaFX-Anwendungsoberflächen, ohne kodieren zu

```
/* JavaFX Ensemble Sequential Transition */
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.event.EventHandler;

import javafx.animation.*;
import javafx.scene.Node;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.util.Duration;

/**
 * A sample in which various transitions are executed sequential-
ly.
 *
 * @related animation/transitions/ParallelTransition
 * @see javafx.animation.SequentialTransition
 * @see javafx.animation.Transition
 */
public class SequentialTransitionSample extends Application {

    private SequentialTransition sequentialTransition;

    private void init(Stage primaryStage) {
        Group root = new Group();
        primaryStage.setResizable(false);
        primaryStage.setScene(new Scene(root, 400,100));
        // create rectangle
        Rectangle rect = new Rectangle(-25,-25,50, 50);
        rect.setArcHeight(15);
        rect.setArcWidth(15);
        rect.setFill(Color.GREEN);
        rect.setTranslateX(50);
        rect.setTranslateY(50);
        root.getChildren().add(rect);
        // create four transitions as fade, translate, rotate and
        scale
        FadeTransition fadeTransition =
            new FadeTransition(Duration.seconds(1), rect);
        fadeTransition.setFromValue(1.0f);
        fadeTransition.setToValue(0.3f);
        fadeTransition.setCycleCount(2);
        fadeTransition.setAutoReverse(true);
        TranslateTransition translateTransition =
            new TranslateTransition(Duration.seconds(2), rect);
        translateTransition.setFromX(50);
        translateTransition.setToX(375);
        translateTransition.setCycleCount(2);
        translateTransition.setAutoReverse(true);
        RotateTransition rotateTransition =
            new RotateTransition(Duration.seconds(2), rect);
```

müssen. Im Scene Builder Editor können UI-Komponenten per „Drag & Drop" im Arbeitsfeld der Oberfläche platziert werden, deren Eigenschaften verändert und StyleSheets zugeordnet werden können. Für das gewählte Layout wird automatisch das passende FXML erzeugt und in einer FXML-Datei abgelegt. Diese kann dann in ein Java-Projekt einer Entwicklungsumgebung integriert werden. Die JavaFX-Oberfläche lässt sich dort mit der Applikationslogik verbinden. Das eingebaute Scene Builder Preview macht im Menü über „Preview in Window" die Oberfläche von „IssueTracking.fxml" auf dem Desktop sichtbar und verwendet sie gemäß ihrer Funktionalität.

Wie im JavaFX Scene Graph in Abbildung 3 dargestellt, verwendet JavaFX zuerst eine „Stage", um die Bühne zu eröffnen und darin eine „Scene" mit Parent- und Child-Nodes anzuordnen. Diese wird mit Layout und Controls verknüpft und in der JavaFX-Anwendung über ein Event-Framework gesteuert. Dies zeigt das Code-Beispiel „javafxapplication1" (Hello World) im Listing 1, bei dem ein Button „Say Hello World" in einer Oberfläche erscheint und beim Betätigen der gedrückte Zustand über ein „ActionEvent" vom Event-Handler ausgewertet sowie im IDE-Output der String „Hello Alexander!" ausgegeben wird. Die JavaFX-Klasse „Scene" (public class javafx.scene.Scene) ist der Ursprung jeglichen Contents im „Scene Graph". Der Scene-Hintergrund füllt sich, indem die Variable „fill" angegeben wird. Die Gruppe der Nodes einer Content-Sequenz wird dann für die „Scene" gerendert.

Die Beispielanwendung „JavaFX Ensemble" stellt eine Auswahl von Anwendungen für den Entwickler bereit, um einzelne JavaFX-Merkmale wie Animationen, Charts, Controls, Grafiken in 2D und 3D, Layout, Media, HTML und Web View zu demonstrieren und den Quell-Code in die Entwicklungsumgebung übernehmen zu können. Das Code-Beispiel „Sequential Transition" (siehe Listing 2) zeigt anschaulich, wie sequenzielle Übergänge in JavaFX programmiert werden. Ein grüner Würfel (siehe Abbildung 4) wird dabei ein- und ausgeblendet, jeweils nach rechts und links waagerecht verschoben, um 180 Grad nach rechts und links gedreht, vergrößert und verkleinert, wieder nach rechts und links gedreht, nach rechts und

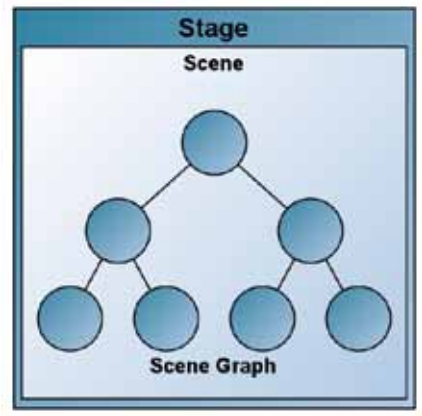


Abbildung 3: JavaFX Scene Graph

links verschoben etc. Zuerst erzeugt man den Würfel als „Rectangle“ und durchläuft dann die vier Übergänge „fadeTransition“ (Ein-/Ausblenden), „translateTransition“ (Seitwärtsbewegung), „rotateTransition“ (Drehung) und „scaleTransition“ (Verkleinern/Vergrößern) nacheinander.

Die beschriebenen Beispiele sind per Download der Datei „javafx_samples-2_1_0.zip“ als JavaFX 2.1-Samples für MS Windows und Mac OS X verfügbar und können mit der Entwicklungsumgebung verwendet werden. Als nützliche Informationsquelle erweist sich FX Experience (siehe http://fxexperience.com/) mit technischen Erfahrungsberichten, der Verfügbarkeit neuer UI-Controls und weiteren Beispielen.

JavaFX in Swing-Anwendungen einbinden

Die Einbindung von JavaFX in Swing-Anwendungen erfolgt mit der JFXPanel-Komponente (siehe Abbildung 5). Listing 3 zeigt die Klasse „JFXPanel“ (public class JFXPanel, extends javax.swing.JComponent). Der darzustellende Inhalt ist mit der Methode „setScene(javafx.scene.Scene)“ spezifiziert, die eine Instanz der JavaFX-Scene erhält. Nach deren Zuordnung wird sie automatisch nachgezeichnet und alle Input- und Fokus-Events gelangen direkt zur Scene. Bei der Verwendung von JFXPanel gibt es die Einschränkung, dass die Swing-Komponente nur über den Event-Dispatch-Thread erreichbar sein sollte. Einzige Ausnahme ist der Aufruf der Methode „setScene(javafx.scene.Scene)“ vom Event-Dispatch- oder vom JavaFX-Anwendungs-Thread. Listing 4 zeigt das typische Pattern zur Verwendung von JFXPanel.

```
        rotateTransition.setByAngle(180f);
        rotateTransition.setCycleCount(4);
        rotateTransition.setAutoReverse(true);
        ScaleTransition scaleTransition =
            new ScaleTransition(Duration.seconds(2), rect);
        scaleTransition.setToX(2);
        scaleTransition.setToY(2);
        scaleTransition.setCycleCount(2);
        scaleTransition.setAutoReverse(true);
        // create sequential transition to do four transitions one
        after another
        sequentialTransition = new SequentialTransition();
        sequentialTransition.getChildren().addAll(
                fadeTransition,
                translateTransition,
                rotateTransition,
                scaleTransition);
        sequentialTransition.setCycleCount(Timeline.INDEFINITE);
        sequentialTransition.setAutoReverse(true);
    }

    public void play() {
        sequentialTransition.play();
    }

    @Override public void stop() {
        sequentialTransition.stop();
    }

    public double getSampleWidth() { return 400; }

    public double getSampleHeight() { return 100; }

    @Override public void start(Stage primaryStage) throws Excep-
tion {

         init(primaryStage);
         primaryStage.show();
         play();
     }
     public static void main(String[] args) { launch(args); }
}
```

Listing 2

```
Class JFXPanel

java.lang.Object
  java.awt.Component
    java.awt.Container
      javax.swing.JComponent
        javafx.embed.swing.JFXPanel
```

Listing 3

Open-Source-Projekt OpenJFX

OpenJFX (siehe http://openjdk.java.net/projects/openjfx/) wurde im November 2011 durch die OpenJDK Community etabliert und ist eine wichtige Anlaufstelle für Entwickler, die sich mit JavaFX beschäftigen. Das Ziel des OpenJFX-Projekts ist der Aufbau eines neuen Java-Client-Toolkits mit einer eigenständigen Java-Spezifikation (JSR), die den JavaFX-Source-Code enthält. Die Beteiligung der Entwicklergemeinschaft ist von erheblicher Bedeutung für JavaFX, um die Einsatzfähigkeit zu erhöhen und die Funktionalität mit weiteren UI-Controls anzureichern.

Fazit

Durch die neu geschaffene technische Ausrichtung von JavaFX und die steigende Verbreitung in der Java-Community deutet sich ein positiver Trend für den zu erwartenden Einsatz von JavaFX in Desktop-Anwendungen an. Auch die Verstärkung der Oracle-Entwicklergruppe bekräftigt die Substanz von JavaFX und sichert die technische Umsetzung der geplanten Bündelung von JavaFX 3.0 mit dem JDK 8. Jedoch entscheiden die Erfahrung der Entwickler im Umgang mit JavaFX und der Reifegrad von JavaFX bis zum Erscheinungszeitpunkt vom JDK 8 über die künftige Verwendung dieser neuen Technologie für Desktop-Anwendungen. Diese Faktoren werden auch wegweisend für die Ablösung von existierender Swing-Technologie in Java-Anwendungen sein.

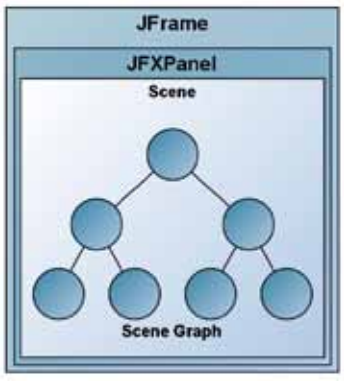


Abbildung 5: JFXPanel-Komponente in Swing

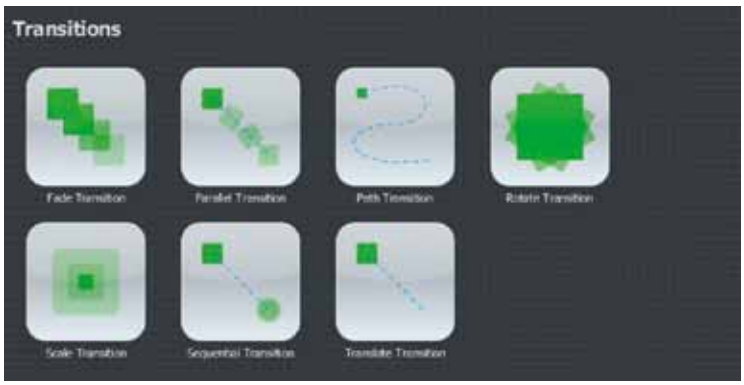


Abbildung 4: JavaFX-Ensemble-Transitions-Anwendungsbeispiele

```
public class Test {

    private static void initAndShowGUI() {
        // This method is invoked on Swing thread
        JFrame frame = new JFrame("FX");
        final JFXPanel fxPanel = new JFXPanel();
        frame.add(fxPanel);
        frame.setVisible(true);

        Platform.runLater(new Runnable() {
            @Override
            public void run() {
                initFX(fxPanel);
            }
        });
    }

    private static void initFX(JFXPanel fxPanel) {
        // This method is invoked on JavaFX thread
        Scene scene = createScene();
        fxPanel.setScene(scene);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                initAndShowGUI();
            }
        });
    }
}
```

Listing 4

Wolfgang Weigend
wolfgang.weigend@oracle.com

Wolfgang Weigend, Systemberater für die Oracle Fusion Middleware bei der Oracle Deutschland B.V. & Co. KG, ist zuständig für Java-Technologie und -Architektur mit strategischem Einsatz bei Großkunden. Er verfügt über langjährige Erfahrung in der Systemberatung und im Bereich objektorientierter Softwareentwicklung mit Java.

Integration à la Apache

Tobias Unger und Jochen Traunecker, GRIDSOLUT GmbH & Co.KG

Die Apache Software Foundation schickt zwei Frameworks zur Integration von Anwendungen ins Rennen: Apache Camel und Apache Synapse. Der Artikel zeigt die wichtigsten Gemeinsamkeiten und Unterschiede von Camel und Synapse auf.

Die Externalisierung interner Dienste in Form offener Schnittstellen (kurz: APIs) stellt neue Anforderungen an Integrations-Lösungen. Unternehmen wollen interne Daten wie zum Beispiel Produkt-Kataloge zur externen Nutzung bereitstellen, sodass diese leicht in mobile Applikationen eingebunden werden können. Das externe API soll dabei meist als Web- oder REST-Service bereitgestellt werden, findet jedoch intern häufig kein unmittelbar passendes Gegenstück. So entsteht das API maßgeblich durch Integration vorhandener Systeme und Dienste. Die Technologien, über die diese Systeme und Dienste integriert werden können, sind dabei sehr heterogen.

Datei- und nachrichtenbasierte Systeme, Web-Services oder die im Java-Umfeld verbreiteten Spring- und JEE-Anwendungen müssen mit dem eingesetzten Integrations-Framework integriert werden können. Im Open-Source-Umfeld gibt es dazu eine Reihe von Integrations-Frameworks, auf deren Basis sich solche Aufgaben angehen lassen. Die Apache Software Foundation schickt gleich zwei Kandidaten ins Rennen: Apache Camel (kurz: Camel) [1] ist ein klassisches Integrations-Framework, wohingegen Apache Synapse (kurz: Synapse) [2] in die Kategorie „Lightweight ESB" fällt. Der Einblick in die wichtigsten Gemeinsamkeiten und Unterschiede von Camel und Synapse beschränkt sich auf die eigentliche Nachrichtenverarbeitung und deren Realisierung in den beiden Frameworks. Darüber hinaus stellen die Autoren einen kleinen Kriterienkatalog auf, der in einer konkreten Projektsituation zur Auswahl eines Frameworks herangezogen werden kann.

Selbstbeschreibung

Auf der Projekt-Webseite ist Camel als vielseitiges Open-Source-Integrations-Framework beschrieben. Grundlage sind die Enterprise Integration Patterns (EIP) aus dem gleichnamigen Buch von Gregor Hohpe und Bobby Woolf [3]. Camel ermöglicht es, Regeln für Routing und Mediation von Nachrichten nach der „Pipes & Filter"-Architektur zu definieren. Diese Regeln können in mehreren Sprachen definiert sein, wobei Java und XML die populärsten sind. Darüber hinaus beschreibt sich Camel als kleine Bibliothek mit minimalen Abhängigkeiten, die sich leicht in eine beliebige Java-Applikation einbetten lässt.

Synapse wird dagegen auf der Projekt-Webseite als leichtgewichtiger, hoch performanter Enterprise Service Bus (ESB) positioniert. Als wichtigste Merkmale sind die asynchron arbeitende Mediation-Engine sowie die exzellente Unterstützung von XML, Web Services und REST hervorgehoben. Letztere basiert maßgeblich auf der Apache Axis2 Web Service Engine (Axis2).

Aus den jeweiligen Selbst-Positionierungen geht hervor, dass sowohl Camel als auch Synapse mit einer ähnlichen Intention ans Werk gehen und als Engine zur Integration, zur Mediation und zum Routing positioniert werden. Hier tritt bereits ein erster Unterschied in der jeweiligen Ausrichtung zutage. Während Camel seine leichte Einbettbarkeit betont, wobei die Einbettungen in Plain-Java-Applikationen, Spring-Applikationen und in Apache ActiveMQ sicher die wichtigsten Vertreter sind, wird die Einbettbarkeit bei Synapse nicht explizit genannt. Im Duo mit Axis2 nennt die Webseite von Synapse zwei Möglichkeiten des Deployments: als Stand-alone und als Web-Applikation. Dennoch kann auch Synapse in eigene Anwendungen eingebettet werden; dazu muss lediglich ein Java-Interface implementiert sein. Im Gegensatz zu Camel scheint die Einbettbarkeit bei den Nutzern von Synapse jedoch nicht relevant zu sein.

Abschließend sei noch erwähnt, dass beide Frameworks ihr reichhaltiges Angebot an unterstützten Protokollen und Datenformaten betonen. Kurz gesagt unterstützen beide die gebräuchlichsten Protokolle und Formate, reduziert auf die schiere Anzahl hat Camel hier die Nase vorn.

Architektur und Nachrichtenformat

Abbildung 1 zeigt die Architektur von Camel. Im Mittelpunkt stehen die Interaction Engine und das Routing. Prozessoren stellen Module dar, die in der Lage sind, Nachrichten zu verarbeiten. Camel stellt diverse Prozessoren bereit, die dedizierte Aufgaben übernehmen wie die Transformation von Nachrichten oder das Implementieren spezifischer EIP-Pattern. Bei Bedarf können Prozessoren selbst implementiert sein. Über Endpunkte kommuniziert Camel mit sogenannten „Komponenten". Endpunkte stellen, abstrakt gesehen, eine uniforme Schnittstelle zur Kommunikation mit Komponenten dar, über die Nachrichten sowohl an eine Komponente geschickt als auch von einer Komponente empfangen werden können. Endpunkte werden über URIs adressiert, wodurch es der Routing Engine ermöglicht wird, Nachrichten an die entsprechenden Komponenten zu senden. Komponenten können wiederum vielfältige Aufgaben wahrnehmen. Hauptsächlich dienen sie dem Versand und Empfang von Nachrichten, beispielsweise über JMS und Web-Services.

Nachrichten werden innerhalb von Camel als sogenannter „Exchange" verschickt. Dieser stellt gewissermaßen eine Art „Wrapper" um den eigentlichen Inhalt der Nachricht dar, der zusätzliche Informationen speichern kann. Dies ist insofern wichtig, als die „Pipes & Filter"-Architektur an sich statuslos ist und somit alle Informationen zur Verarbeitung aus dem Exchange selbst kommen müssen. Der Exchange enthält wiederum eine Message, die schließlich den Inhalt der eigentlichen

Nachricht repräsentiert. Als Inhalt einer Nachricht akzeptiert Camel jedes beliebige Java-Objekt, sei es ein POJO oder auch ein in DOM geparstes XML-Dokument. Des Weiteren kann eine Message Header speichern, was hilfreich ist, um HTTP-Header oder JMS-Header mitzuführen.

Abbildung 2 zeigt die Architektur von Synapse. Die zentrale Komponente hier ist die Mediation Engine. Nachrichten werden von Endpunkten empfangen beziehungsweise an Endpunkte gesendet, wobei ein Endpunkt durch eine URL definiert ist. Im Gegensatz zu Camel basiert die interne Kommunikation von Synapse auf dem kanonischen Nachrichtenformat SOAP. Eingehende Nachrichten müssen zunächst in SOAP-Nachrichten überführt und ausgehende Nachrichten wiederum aus SOAP-Nachrichten in das jeweilige Zielformat transformiert werden.

Um nicht immer für jedes mögliche Nachrichtenformat (CSV, POX, JSON etc.) und dessen Transportmöglichkeiten (HTTP, JMS, AMQP, Email, File etc.) jeweils eine individuelle Komponente zur Nachrichten-Transformation erstellen zu müssen,

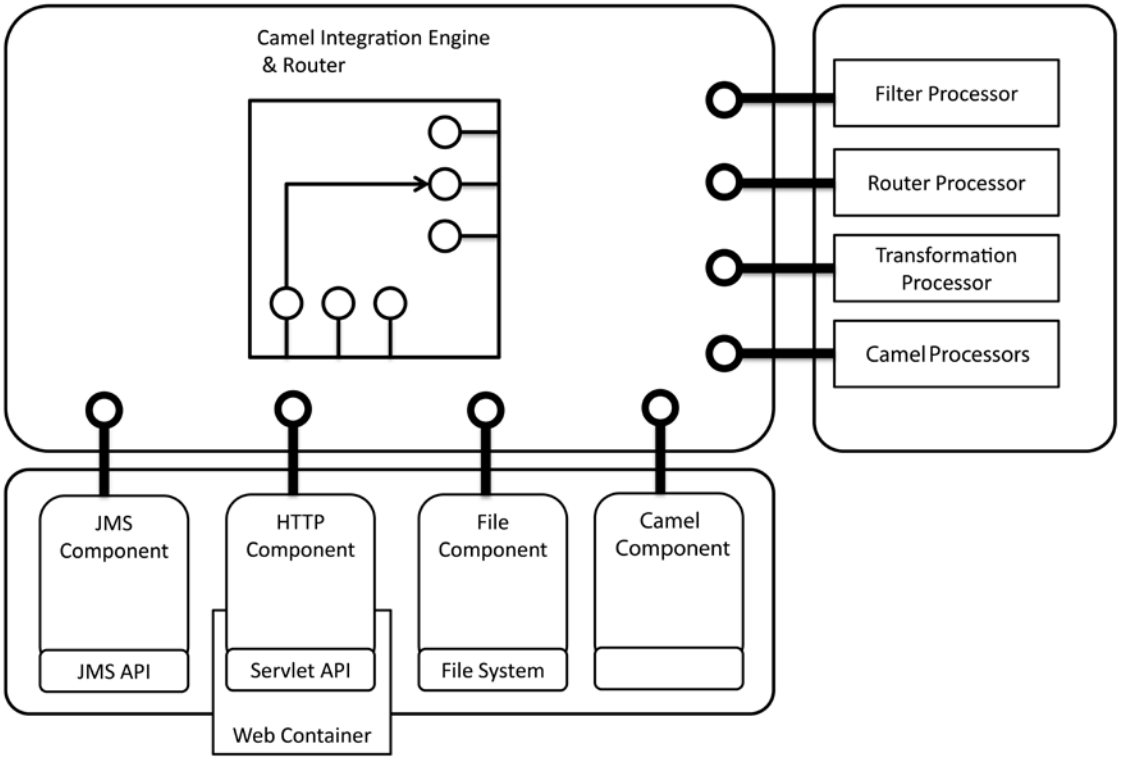


Abbildung 1: Architektur von Camel

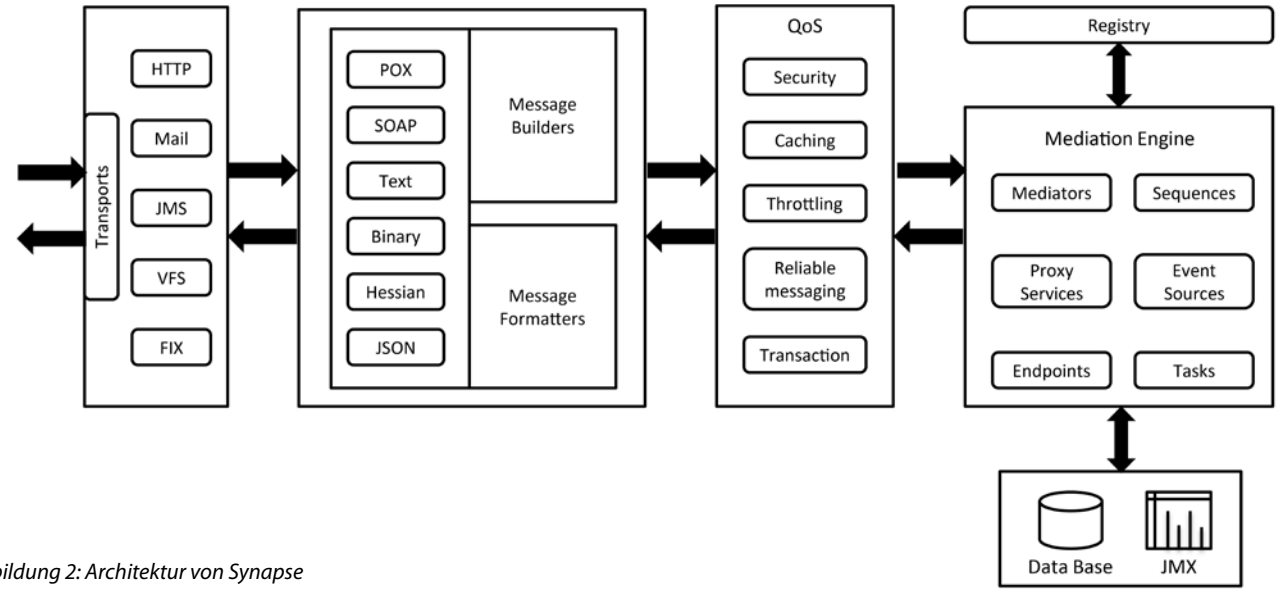


Abbildung 2: Architektur von Synapse

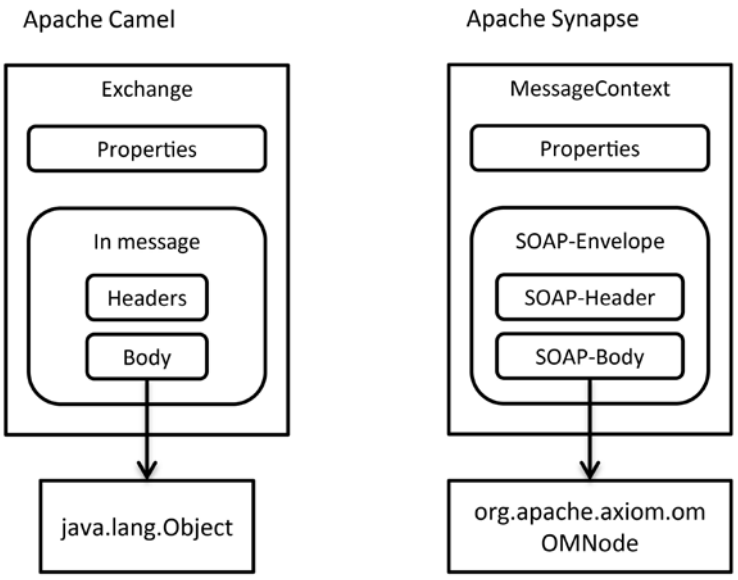


Abbildung 3: Gegenüberstellung Nachrichtencontainer Camel und Synapse (vereinfacht)

durchlaufen alle Nachrichten sowohl beim Eintritt als auch beim Austritt aus der Synapse Mediation Engine fest definierte Komponenten. Eingehende Nachrichten werden erst von einem Transport empfangen, der ein Protokoll spricht wie HTTP oder JMS.

Als nächster Schritt wird die Nachricht in einem Message Builder verarbeitet, der meist anhand des Content Types der Nachricht ausgewählt wird. Die Hauptaufgabe des Message Builders besteht darin, aus der eingehenden Nachricht, die durch den Transport durchgereicht wurde, mittels Apache Axiom eine SOAP-Nachricht zu bauen. Ähnlich wie Camel packt Synapse die Nachricht in einen Wrapper ein, der bei Synapse „Message Context" heißt. Auch hier können zusätzliche Informationen abgelegt und mit der Nachricht mitgeführt werden.

Ausgehende Nachrichten werden durch die Message Formatter aufbereitet und anschließend über einen Transport verschickt. Die Auswahl der entsprechenden Transporte und Message Formatter erfolgt durch die URL des Endpunkts, an den die Nachricht verschickt werden soll, und zusätzlich eventuell durch den Content Type. Als letzte Komponente ist die QoS-Komponente zu nennen, die beispielsweise für Throtteling oder auch Caching verantwortlich ist.

```
curl -i -H "Accept: application/json" -H "Content-Type:
application/json" -X POST -d ,{GetQuote:{symbol:"IBM"}}'
http://localhost:8280/services/MessageFormat1
```

Listing 1

```
<?xml version='1.0' encoding='utf-8'?>
 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.
org/soap/envelope/">
      <soapenv:Body>
            <GetQuote>
                  <symbol>IBM</symbol>
            </GetQuote>
      </soapenv:Body>
 </soapenv:Envelope>
```

Listing 2

Nachrichtenformat als grundlegender Unterschied

Für Camel ist die eigentliche Nachricht (Message Body) innerhalb des Nachrichtenkontextes (Exchange) ein beliebiges Java-Objekt (siehe Abbildung 3). Synapse beschneidet dieses generische Konstrukt dahingehend, dass ein Axiom OMNode als SOAP-Body erwartet wird. Für beide Design-Entscheidungen gibt es jeweils Argumente, die das Für und Wider unterstreichen. Schlussendlich kann bei Camel-basierten Projekten auch jede Nachricht in eine „XML-Infoset"-konforme Repräsentation überführt werden.

Aber auch Synapse bietet Freiheitsgrade in der Nachrichten-Repräsentation. Dazu betrachtet man beispielsweise ein klassisches Proxy-Szenario, bei dem eine JSON-Nachricht per HTTP an Synapse geschickt und von Synapse ebenfalls als JSON-Nachricht unverändert per JMS an eine Queue durchgereicht werden soll. Listing 1 zeigt, wie dieses Beispiel leicht mit cURL nachgestellt werden kann. Anhand des Content-Types „application/json" wird von Synapse der JSON-MessageBuilder ausgewählt, der die JSON-Nachricht in XML wandelt und schließlich als SOAP-Nachricht weiterreicht (siehe Listing 2). Beim Senden der Nachricht läuft dieser Transformationsprozess erneut ab und aus der XML-Nachricht wird wieder eine JSON-Nachricht erstellt.

Da in diesem Beispiel keine Bearbeitung der Nachricht erfolgt, kann auf die Nachrichtentransformation gänzlich verzichtet werden. Am einfachsten ist es dabei, wenn man Synapse anweist, die Nachricht als Content-Type „text/plain" zu verarbeiten (siehe Listing 3). Dies veranlasst Synapse, einen entsprechenden Message Formatter beim Empfangen und einen geeigneten Message Builder beim Versenden der Nachricht auszuwählen. Listing 4 verdeutlicht das Ergebnis. Die ursprüngliche JSON-Nachricht wurde nicht in XML umgewandelt, sie ist lediglich als TextNode beziehungsweise String in der SOAP-Nachricht eingebettet.

Modellierung von Integrationen

Sowohl Camel als auch Synapse stellen eine Sprache bereit, mit der die Nachrichtenverarbeitung modelliert werden kann. Bei Camel heißt das entsprechende Konstrukt „Route“ und bei Synapse „Proxy“. Beide Frameworks bieten ein XML-Rendering ihrer Sprachen an. Bei Camel können Routen auch als Java-Objekte definiert und aufgebaut sein, darüber hinaus werden Groovy und Scala unterstützt (DSL).

Ein erster Blick auf die von Camel definierte Sprache offenbart gleich die enge Anlehnung an die oben bereits erwähnten EIPs. Diese lassen sich mit der DSL von Camel sehr gut umsetzen. Listing 5 zeigt ein einfaches Beispiel einer Route. In Camel definiert eine Route den Weg, den eine Nachricht nimmt, um von einer Quelle (etwa einer Queue) zu einem Ziel (etwa einer Datei) zu gelangen, die jeweils durch Endpunkte repräsentiert sind. Zwischen Quelle und Ziel lassen sich beliebige weitere Bearbeitungs- oder Entscheidungsschritte wie Filter oder Router einfügen. Im Beispiel beschreibt das „From“-Element den Endpunkt, an dem auf eingehende Nachrichten gehört werden soll und der durch ein URI beschrieben ist. Die eingehende Nachricht wird an die nachfolgenden „To“-Elemente der Reihe nach weitergereicht. Ein „To“-Element veranlasst Camel, den Exchange an die jeweiligen Endpunkte weiterzureichen. Im Falle eines „Request-Reply“-Patterns wird die Nachricht, die aus dem letzten Verarbeitungsschritt als Ausgabe herauskommt, an den ursprünglichen Sender zurückgeschickt.

Typischerweise wird Synapse zur Vermittlung des Nachrichtenflusses zwischen einem Client und einer Service-Implementierung verwendet. Zur Modellierung dieses Nachrichtenflusses stellt Synapse eine XML-basierte DSL bereit. Diese bildet im Gegensatz zur Camel-DSL nicht direkt EIPs ab, jedoch lassen sie sich nach Meinung der Autoren auch mit der Synapse-DSL gut umsetzen. Wichtigstes Element ist der Proxy, der als Container für die Definition des Nachrichtenflusses dient. Listing 6 zeigt das Beispiel eines Proxy-Service. Das Attribut „Transports“ beschreibt die Menge der Transport-Protokolle, für die Synapse Endpunkte des entsprechenden Proxy generieren soll. Ein solcher „Endpoint“ definiert, wohin der Proxy eine Nachricht

```
curl -i -H "Accept: text/plain" -H "Content-Type: text/
plain" -X POST -d "{GetQuote:{symbol:"IBM"}}' http://local-
host:8280/services/MessageFormat1
```

Listing 3

```
  <?xml version='1.0' encoding='utf-8'?>
 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.
org/soap/envelope/">
      <soapenv:Body>
            <text xmlns="http://ws.apache.org/commons/ns/pa
yload">{GetQuote:{symbol:"IBM"}}</text>
      </soapenv:Body>
 </soapenv:Envelope>
```

Listing 4

```
<route>
  <from uri="cxf:bean:stockquote?dataFormat=MESSAGE" />
  <!-- log input received -->
  <to uri="log:input" />
  <!-- send proxied request to real web service -->
  <to ref="stockEndpoint" />
  <!-- log answer from real web service -->
  <to uri="log:output" />
</route>
```

Listing 5

```
<proxy xmlns="http://ws.apache.org/ns/synap-
se" name="PriceInfoProxy" transports="https,http"
statistics="disable" trace="disable" startOnLoad="true">
    <target>
       <inSequence>
          <log level="full" separator="," />
          <send />
       </inSequence>
       <outSequence>
          <log level="full" separator="," />
          <send />
       </outSequence>
       <endpoint>
          <wsdl service="StockQuote" port="StockQuoteSoap"
uri="http://www.webservicex.net/stockquote.asmx?WSDL">
          </wsdl>
       </endpoint>
    </target>
 </proxy>
```

Listing 6

senden soll. Im Beispiel wird ein WSDL-Endpunkt verwendet, um den Service zu spezifizieren, an den die Nachricht geschickt werden soll.

Synapse stellt eine Menge an Endpunkten bereit, die zum Beispiel auch ein Load-Balancing ermöglichen. Auf dem Weg vom Eingangs-Endpunkt zu einem Ausgangs-Endpunkt durchläuft die Nachricht „Sequences", in denen die Nachricht durch Mediatoren bearbeitet werden kann. Von diesen Mediatoren steht ein kompaktes, aber ausreichendes Angebot zur Verfügung. Da es sich im Beispiel um ein „Request-Reply"-MEP handelt, existiert neben der „inSequence" noch eine „outSequence", durch die die Antwort-Nachricht geleitet wird. Daneben kann eine Sequenz zur Fehlerbehandlung angegeben werden. Weitere Sequenzen lassen sich bei Bedarf hinzufügen, etwa bei bedingten Verzweigungen.

Service Chaining

Ein Pattern, das bei Integrationsaufgaben häufig anzutreffen ist, ist das „Service Chaining". Dabei wird die vom Aufrufer gesendete Nachricht der Reihe nach an mehrere Services geschickt und dann wieder an den ursprünglichen Aufrufer zurückgesendet. Mit Camel lässt sich dieses Pattern direkt mit einer Reihe von „To"-Elementen abbilden, wie in Listing 5 gezeigt.

Bei Synapse ist die Abbildung des „Service Chaining"-Patterns auch leicht möglich, dennoch kommt hier ein anderes Verständnis von Mediation zum Tragen. Bei Synapse dienen Mediatoren mehrheitlich zur Bearbeitung der Nachricht. Dazu zählen das Transformieren einer Nachricht, aber auch das Logging von Nachrichten und das aktive Überwachen von QoS-Anforderungen. Sobald eine Nachricht zu einem weiteren Service geschickt werden soll, beendet der Send-Mediator die Sequence. Abbildung 4 zeigt dieses Verhalten. Service „A" schickt eine Nachricht an einen Proxy-Service (in Synapse beschrieben). Dabei wird die Nachricht durch die In-Sequence geschickt, ein Logging-Mediator (M1) aufgerufen und nach weiteren Bearbeitungsschritten (M2 … M4) durch den Send-Mediator an den Endpunkt des Service „B" geschickt. Mit dem erfolgreichen Versand dieser Nachricht ist die In-Sequence beendet. Die darauf folgende Antwort des aufgerufenen Service B wird selbst wiederum durch eine dedizierte Sequence (MySequence) verarbeitet. Dieses Verhalten wiederholt sich für den Aufruf von Service „C". Schlussendlich erhält Service „A" die Antwort aus der Out-Sequence. Sollte während der Nachrichtenverarbeitung innerhalb des Proxy-Service ein Fehler auftreten, wird die Fault-Sequence abgearbeitet und im Beispiel eine Antwort an Service „A" zurückgegeben.

Transformationen

Die Transformation der in den Nachrichten enthaltenen Daten ist bei Integrationsprojekten das Brot-und-Butter-Geschäft schlechthin. Da Camel beliebige Java-Objekte als Nachrichten akzeptiert, existiert ein breites Spektrum an Möglichkeiten zur Nachrichten-Transformation. Transformationen erfolgen mithilfe der EIPs Message Translator und Content Enricher, die dies durch spezifischen Java-Code ermöglichen. Daten lassen sich auch durch Camel-Komponenten transformieren. Beispielsweise existiert eine Komponente zur Transformation mittels XSLT. Camel kennt außerdem sogenannte „Data Formats", durch die Daten innerhalb einer Route mit „Marshall" und „Unmarshall" transformiert werden können. Dies ist zum Beispiel dann nützlich, wenn man JSON-Serialisierungen in Java-Objekte überführen möchte. Darüber hinaus können für Transformationen noch der Template-Mechanismus (basierend auf Apache Velocity) und der Camel-eigene Typ-Converter-Mechanismus eingesetzt werden. Auch implizite Transformationen in Komponenten sind möglich. Die Apache-CXF-Komponente ist beispielsweise in der Lage, SOAP-Nachrichten direkt in Java-Objekte zu wandeln, wie in JAX-WS und JAXB spezifiziert.

Bei Synapse erfolgt eine Nachrichten-Transformation häufig schon beim initia-

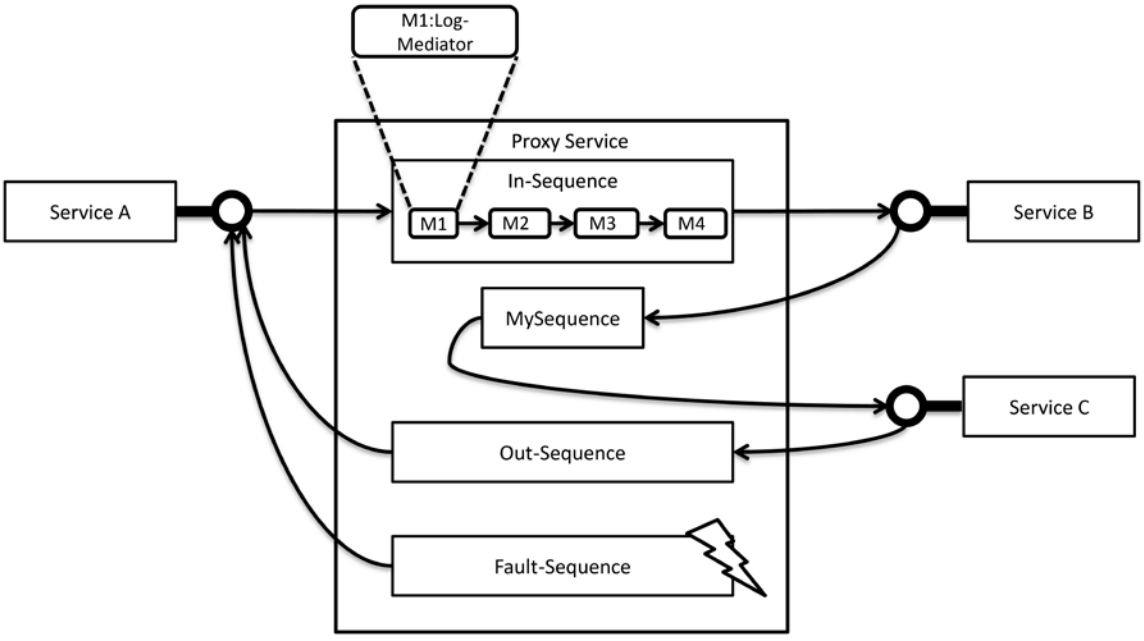


Abbildung 4: Bei Synapse dienen Mediatoren mehrheitlich zur Bearbeitung der Nachricht

len Empfang einer Nachricht direkt nach dem Transport auf der Ebene der Message-Builder (Message-Formatter beim Senden). Sobald eine beliebige Nachricht durch den Message-Builder in eine SOAP-Nachricht überführt wurde, steht mit dem XSLT-Mediator ein mächtiges Instrument zur Nachrichten-Transformation zur Verfügung. Darüber hinaus können eigene Transformationen in Java selbst oder in diversen Skriptsprachen (JavaScript, Groovy etc.) formuliert werden. Mit dem Smooks-Mediator steht die ganze Mächtigkeit des Smooks-Frameworks zur Datentransformation auch in Synapse zur Verfügung.

```
<route id="login">
  <description>Handle login requests</description>
  <from uri="mom:queue:login.request" />
  <bean ref="MessageCrypto" method="decrypt" />
  <unmarshal ref="json" />
  <bean ref="AuthBean" method="checkLogin" />
  <bean ref="AuthBean" method="checkUserRights" />
  <marshal ref="json" />
  <bean ref="MessageCrypto" method="encrypt" />
</route>
```

Listing 7

Direkter Aufruf von Java-Logik

Eine Stärke von Camel besteht darin, Java-Objekte innerhalb einer Camel Route direkt aufzurufen und deren Ein- und Ausgabeparameter als Java-Objekte weiterzureichen. Damit kann komplexe Geschäftslogik durch die Kombination mehrerer Java-Beans implementiert und über die vielfältigen Kommunikationsmöglichkeiten von Camel verfügbar gemacht werden (siehe Listing 7). Die Eingabedaten werden aus JSON in Java-Objekte gewandelt, an zwei Beans weitergereicht und wieder ins JSON-Format zurückkonvertiert.

Bei Synapse hingegen müsste man in diesem Fall immer zuerst eine SOAP-Nachricht erzeugen, auch wenn nur Java-Objekte von einer Bean zur anderen gereicht werden sollen. Zwar scheint die jeweilige Transformation aus Java-Objekten in SOAP-Nachrichten bei jedem Aufruf einer Java-Bean vermeidbarer Aufwand zu sein, doch verliert dieser vermeintliche Nachteil rapide an Relevanz, sobald Camel nicht in derselben VM läuft wie die aufzurufenden Java-Beans. Wird Camel zum Beispiel als eigenständiger Integrations-Server betrieben, dann führt ein scheinbar direkter Aufruf zweier Java-Beans aus einer Camel Route heraus stets zu einem entfernten Methoden-Aufruf, der eine Serialisierung der Aufruf-Parameter und Antworten bedingt. In diesem Fall würde sich der Aufruf der entfernten Java-Beans – analog zum Synapse-Ansatz – über Web-Services anbieten.

Community, professioneller Support und Erweiterungen

Um beide Apache-Frameworks und -Projekte hat sich eine aktive Community gebildet, die sowohl die Frameworks weiterentwickelt als auch durch Mailing-Listen und Foren die tägliche Integrationsarbeit unterstützt. Darüber hinaus bieten diverse Unternehmen professionellen Support als Dienstleistung an. Für Camel werden Anbieter von kommerziellem Support auf den Webseiten des Projekts gelistet (FuseSource, OpenLogic, Savoir Technologies Inc. und Talend Inc.). Das Synapse-Projekt erwähnt keine kommerziellen Dienstleister, ein Blick auf die Unterstützer des Projekts zeigt jedoch, dass auch hier ein Unternehmen aktiv ist, das professionellen Support anbietet (WSO2 Inc.).

Über den reinen Support hinausgehend bieten Unternehmen Produkte an, die jeweils entweder Camel oder Synapse als Integrations-Komponente beinhalten. Exemplarisch für Camel ist das Produkt-Portfolio von FuseSource, das vom Fuse ESB über Fuse Message Broker bis hin zur Fuse IDE reicht. Welche dieser Produkte nur im Rahmen eines Dienstleistungsvertrags erhältlich sind, ist auf der Web-Präsenz des Herstellers ersichtlich. Einen bemerkenswerten Weg hinsichtlich des Open-Source-Gedankens geht im Zusammenhang mit Synapse das Unternehmen WSO2. Dieses Unternehmen entwickelt ein breites Spektrum an Middleware-Software-Produkten, angefangen vom WSO2 ESB bis hin zu PaaS-Lösungen für das eigene Rechenzentrum, die ausnahmslos der Apache2-Lizenz unterliegen. Ein Blick auf die Webseite dieses Unternehmens ist sicherlich lohnenswert.

Kriterien zur Entscheidungsfindung

Erstes Zwischen-Fazit aus den genannten Gemeinsamkeiten ist, dass es keinen eindeutigen Sieger gibt. Um aber dennoch einen Weg aufzuzeigen, wie man sich für ein Integrations-Framework entscheiden kann, sind abschließend noch einige Kriterien erläutert.

In ein Integrationsprojekt sind im Allgemeinen viele Rollen involviert, etwa Kunden, Manager, Architekten, Entwickler etc. Die Autoren haben die Erfahrung gemacht, dass Integrationsprojekte deutlich besser verlaufen, wenn sich alle Rollen mit der Auswahl des Frameworks identifizieren können.

Wichtige Anhaltspunkte zur Evaluation der Frameworks liefert meist die eigene Architektur- und Technologie-Strategie. Setzt ein Unternehmen konsequent auf eine SOA und will diese technisch mithilfe von Web-Services realisieren, gelten ganz andere Kriterien zur Evaluation, als wenn ein Unternehmen auf reine Java-Architekturen setzt und diese mittels JEE oder Spring umsetzt. Prinzipiell eignen sich beide hier vorgestellten Frameworks für beide Einsätze. Bei reinen Web-Service-Projekten ist nach Ansicht der Autoren Synapse etwas im Vorteil aufgrund der flexiblen Architektur von Axis2. Auch kommt hier das normalisierte Nachrichten-Format basierend auf SOAP voll zum Tragen. Im reinen Java-Umfeld bietet Camel hingegen mehr Möglichkeiten. Vor allem die direkte Einbettbarkeit in Java-Anwendungen sei hier erwähnt.

Die potenziellen Einsatzszenarien sollten prototypisch genau getestet werden. Dabei ist darauf zu achten, dass die Szenarien möglichst alle späteren Einsatzbereiche abdecken und auf Grundlage der Architektur- und Technologie-Strategie erfolgen. Die Prototypen geben dem Pro-

jektteam wichtige Erfahrungen, um später Programmier-Richtlinien abzuleiten. Des Weiteren lassen sich durch den Test auch Erfahrungen darüber sammeln, wie es um die Qualität und Konfigurierbarkeit der verwendeten Komponenten bestellt ist. Im Allgemeinen sind nicht alle Komponenten eines Frameworks auf dem gleichen qualitativen Stand, sodass man Alternativen rechtzeitig identifizieren kann.

Ein sehr wichtiges Kriterium innerhalb einer Evaluation ist natürlich die Modellierungs- beziehungsweise Programmiersprache. Als ein Maß für die Vollständigkeit der Sprache kann auf die Umsetzbarkeit der EIPs hin getestet werden. Hier besticht Camel natürlich durch eine intuitive Sprache direkt auf Basis der EIPs. Erfahrungsgemäß versetzt diese Sprache Entwickler schnell in die Lage, produktiv zu arbeiten. Das ursprüngliche Anwendungsfeld von Synapse liegt im Erstellen von Proxys für Web-Services. Sollen solche Proxys über möglichst viele Protokolle und Formate angesprochen werden, ist dies einfacher zu erledigen als etwa bei Camel. Dennoch erlaubt auch Synapse die Umsetzung der EIPs – nur nicht so intuitiv.

Für den produktiven Einsatz der Frameworks halten die Autoren außerdem das Tooling für sehr wichtig. Leider bietet keines der Frameworks selbst einen grafischen Editor an. Hier muss auf andere Angebote zurückgegriffen werden, die leider nur teilweise kostenlos und auch nur teilweise unter Open-Source-Lizenz im Quellcode verfügbar sind. Wer die Integrationslogik manuell erstellen will, kann diesen Punkt getrost ignorieren. Die Integration der Frameworks in Entwicklungs- und Betriebsprozesse ist ein oft unterschätzter Punkt bei der Auswahl, der schon in der Evaluationsphase bedacht und getestet werden sollte. Die Zusammenarbeit mit Tools zu Build-Management, Versionsverwaltung oder Issue-Tracking sollte nahtlos funktionieren, um einen effizienten Einsatz zu gewährleisten.

Neben dem Framework sollte als Teil der Evaluation auch die zugehörige Community betrachtet werden. Sowohl das Camel- als auch das Synapse-Projekt besitzen eine aktive Community, wobei die Synapse-Community etwas größer und aktiver zu sein scheint. Oft gilt es auch der Roadmap Beachtung zu schenken, um zu sehen, wie die Weiterentwicklung der Frameworks geplant ist. Dennoch gibt es bei Open-Source-Projekten natürlich keine Versicherung dafür, dass die Roadmap auch wie geplant umgesetzt wird. Dies lässt sich durch Beteiligung am Projekt aktiv beeinflussen.

Unterstützung kommerzieller Art gibt es für beide Frameworks. Darüber hinaus erweitern diese Anbieter die Frameworks um weitere Funktionen. Hier ist darauf zu achten, ob diese Erweiterungen als Open Source zur Verfügung stehen oder nicht.

Adapter an kommerzielle Systeme wie SAP können ein wichtiges Entscheidungskriterium sein. Hier muss bei beiden Frameworks auf Angebote Dritter zurückgegriffen werden, wobei sich das aktuelle Angebot an Adaptern und deren Lizenzbedingungen erheblich unterscheidet.

Zu guter Letzt sollte man die Listen der umgesetzten Standards und Features der Frameworks vergleichen. Im Allgemeinen bieten beide Frameworks eine große und im Normalfall ausreichende Liste an Standards und Features, aber im Detail gibt es auch hier Unterschiede. Camel führt zwar quantitativ bei der Anzahl, dennoch sollte hier ein fokussierter Blick basierend auf der Projektsituation erfolgen.

Fazit

Sowohl Camel als auch Synapse stellen flexible Mittel zum Empfang, zur Bearbeitung und Weiterleitung von Nachrichten zu Verfügung. Zur Modellierung von Integrationsmustern bieten beide Apache-Projekte domänenspezifische Sprachen an (DSL), hierbei sind die DSLs von Camel intuitiver in der Handhabung. Beide Projekte ermöglichen grundsätzlich die Einbettung in eigene Applikationen, dies gelingt Camel besonders gut. Da in Camel intern die Nachrichten in Form von Java-Objekten (java.lang.Object) geführt werden, ist die direkte Integration eigener Java-Komponenten (Java Beans) zur Nachrichtenbearbeitung leicht möglich.

Die Prinzipien serviceorientierter Architekturen sind in Entwicklungsprojekten sowohl mit Camel als auch mit Synapse umsetzbar. Durch die konsequente Trennung der Nachrichtenverarbeitung in Transport, Message-Builder beziehungsweise Message-Formatter können zum Beispiel Multi-Protokoll-Proxys mit Synapse sehr elegant erstellt werden. Synapse hat sich dafür entschieden, ausschließlich SOAP als internes Nachrichtenformat zuzulassen. Damit lassen sich Web-Services entsprechend leicht integrieren.

Beide Frameworks eignen sich nach Ansicht der Autoren gut, um Integrationsaufgaben anzugehen. Camel hat sicher die Nase etwas weiter vorn, falls es um den puren Java-Aspekt und die Einbettbarkeit geht. Auch ist die direkte Umsetzung der EIPs innerhalb der Camel DSL von Vorteil. Synapse steht hier jedoch nicht wirklich zurück, macht aber umso mehr Freude, wenn man bereit ist, sich mit SOA, REST und Web Services auseinanderzusetzen. Als finales Fazit bleibt, dass der Weg zum passenden Integrations-Framework nur über eine ausführliche Evaluation geht.

Weitere Informationen

http://camel.apache.org/
http://synapse.apache.org/
http://www.eaipatterns.com/

Tobias Unger
tobias.unger@gridsolut.de
Jochen Traunecker
jochen.traunecker@gridsolut.de

Tobias Unger (Dipl.-Inf.) verfügt über mehrjährige Erfahrung über BPM und Java Enterprise" sowohl im praktischen wie auch im wissenschaftlichen Gebiet, insbesondere in den Bereichen „Prozessausführung" und „Aufgabenverwaltung". Er ist Autor zahlreicher Publikationen und als Sprecher auf Konferenzen aktiv.

Jochen Traunecker (Dipl.-Ing. sc.agr, Dipl.-Inf.) ist Gründer und geschäftsführender Gesellschafter der GRIDSOLUT und kann auf viele Jahre Praxis im Bereich des Software-Engineerings zurückblicken.

Apache Camel Security – Route Security

Dominik Schadow, Trivadis GmbH

Nachdem der erste Artikel zur Apache Camel Security in der letzten Ausgabe der Java aktuell die Sicherung der Nutzdaten mit den Crypto- und XMLSecurity-Datenformaten zum Thema hatte, geht es nun um die Absicherung der eigentlichen Route.

Camel kann mit verschiedenen Komponenten ergänzt werden, sodass man das Rad nicht zwangsläufig komplett neu erfinden muss. Die von Camel out-of-the-box gebotenen Komponenten repräsentieren dabei gleich zwei der bekanntesten Vertreter aus dem Bereich der Authentifizierung und Autorisierung: Apache Shiro und Spring Security.

Neben der Verschlüsselung und/oder Signierung von Nachrichten gibt es im Apache-Camel-Umfeld üblicherweise weitere Anforderungen an die Sicherheit. So sollen beispielsweise nicht in jedem Integrationsprojekt alle Routen allen Benutzern oder Benutzergruppen gleichermaßen offenstehen. Vielfach taucht der Wunsch auf, bestimmte Routen komplett oder auch abschnittsweise nur für bestimmte Benutzer oder Benutzergruppen zu öffnen und andere mit oder ohne Exception abzuweisen. In Camel bieten sich dazu, je nach Projektvorgaben oder persönlichen Vorlieben, die beiden Komponenten Apache Shiro und Spring Security an.

Wirft man einen Blick auf die in Camel angebotenen Kern-Features dieser beiden Komponenten, stellt man gewisse Ähnlichkeiten und auch einige Überschneidungen fest. Die gleichzeitige Verwendung ist daher kaum sinnvoll, allein schon wegen der notwendigen doppelten Konfiguration. Stattdessen sollte man sich entweder für Apache Shiro oder für Spring Security entscheiden. Wer dann bereits eines dieser Frameworks außerhalb des Camel-Kontextes eingesetzt hat, kann besonders schnell in die Verwendung der jeweiligen Komponente mit Camel einsteigen.

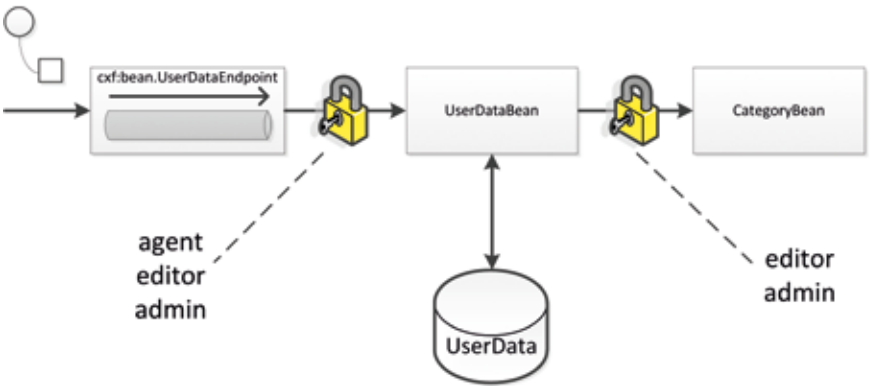


Abbildung 1: Die Camel-Route des Beispielprojekts im Überblick

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-shiro</artifactId>
  <version>2.9.2</version>
</dependency>
```

Listing 1

Das Beispielprojekt

Unser Beispiel entspricht nahezu der ursprünglichen Variante aus dem Artikel in der letzten Ausgabe. Um die Routen übersichtlicher zu halten, wird im Folgenden nur die ungesicherte Operation „findUserData(int ssn)" als Ausgangsbasis verwendet. Die beiden gesicherten Varianten, die im Laufe dieses Artikels für diese Operation entstehen werden (siehe Abbildung 1), lauten dementsprechend „findUserDataShiro(int ssn)" und „findUserDataSpring(int ssn)".

Abgesichert werden zwei unterschiedliche Abschnitte der Route: So soll die Operation „findUserData[Shiro|Spring]" nur von Mitgliedern einer der vorhandenen Benutzergruppen (agent, editor oder admin) aufgerufen werden können. Nicht autorisierte Aufrufe bei unbekannten Benutzern oder falscher Gruppe sollen zu einer Exception und zum Abbruch der Verarbeitung führen. Der folgende Aufruf der „CategoryBean" mit der Operation „CalculateCategory" soll dann Mitgliedern der beiden Benutzergruppen „editor" und „admin" vorbehalten sein. Bei Benutzern der Gruppe „agent" soll dieser Aufruf einfach nicht durchgeführt und auch keine Exception ausgelöst werden. Im Beispiel findet hierbei zur Verdeutlichung nur ein Eintrag im Log statt.

Alle nachfolgenden Listings zeigen nur Ausschnitte des Source Codes. Das vollständige Beispielprojekt findet sich auf GitHub unter https://github.com/dschadow/CamelSecurity im SpringSource-Tool-Suite-Projekt „CamelRouteSecurity".

Apache Shiro

Apache Shiro [1] ist ein junges und mächtiges Security-Framework, unter anderem zur Authentifizierung, Autorisierung, Session-Verwaltung und Kryptografie. Die Camel-Shiro-Komponente [2] stellt mit

Authentifizierung und Autorisierung jedoch nur den im Camel-Umfeld sinnvollen Funktionsumfang zur Verfügung. Beliebige Abschnitte einer Route können so mit Shiro abgesichert sein. Um die Shiro-Komponente im Projekt zu verwenden, muss zunächst die Maven-POM-Datei erweitert werden (siehe Listing 1).

Damit die so für Shiro festgelegten Einstellungen anschließend in einer Camel-Route verwendet werden können, muss eine „ShiroSecurityPolicy" mit einer Konfigurationsdatei initialisiert werden.

Konfiguration und Verwendung einer Policy-Datei

Apache Shiro verwendet für die Konfiguration von Policy-Dateien eine „.ini"-Datei, wie Listing 2 mit der „shirosecuritypolicy.ini" zeigt. Da es in diesem Artikel primär um Camel Security und nicht um die Shiro-Konfiguration geht, stellt die hier entwickelte „.ini"-Datei die einfachste aller Möglichkeiten dar. Statt der Passwörter im Klartext (oder auch verschlüsselt beziehungsweise als Hash) bietet Apache Shiro selbstverständlich auch die Anbindung von LDAP, Datenbanken etc. an, wodurch diese sensiblen Daten in gesicherter Form zur Verfügung gestellt werden.

Im Beispiel werden somit die drei verschiedenen Benutzer „userAgent", „userEditor" und „userAdmin" festgelegt und diesen jeweils eine der drei vorhandenen Rollen „agent", „editor" und „admin" zugewiesen. Die Berechtigungen der Rollen wie „findUserDataShiro" entsprechen dabei den IDs der Routen. Prinzipiell können Berechtigungen beliebig benannt werden, eine gewisse Ähnlichkeit zu den IDs der Routen macht diese aber leichter verständlich. Im Beispiel erhält die Gruppe „agent" somit alle Rechte für „findUserDataShiro", die Gruppe „editor" für „findUserDataShiro" und „calculateCategory" und „admin" darf schließlich alles. Was genau hinter diesen Rechten steht und auf welche Routen(-abschnitte) damit Bezug genommen wird, bleibt an dieser Stelle noch offen. Grundsätzlich wird eine Policy-Datei über eine „ShiroSecurityPolicy" instanziiert und kann anschließend an beliebiger Stelle der Route verwendet werden (siehe Listing 3).

Es ist dabei durchaus ein großer Unterschied, ob die Policy direkt nach „from"

```
[users]
  userAgent = secret1, agent
  userEditor = secret2, editor
  userAdmin = secret3, admin

[roles]
  agent = trivadis:findUserDataShiro:*
  editor = trivadis:findUserDataShiro:*,
           trivadis:calculateCategory:*
  admin = trivadis:*
```

Listing 2

```
ShiroSecurityPolicy ssp = new
  ShiroSecurityPolicy("classpath:shirosecuritypolicy.ini");
      // ...
      from("direct:findUserDataShiro").policy(ssp).
       beanRef("userDataBean",
      "loadUserData").to("direct:calculateCategory");
```

Listing 3

```
@Component
public class UserDataShiroRouteBuilder extends SpringRoute
Builder {
  @Override
  public void configure() throws Exception {
    final byte[] passPhrase = "CamelSecureRoute".getBytes();

    List<Permission> permissionsList = new
      ArrayList<Permission>();
    permissionsList.add(new
      WildcardPermission("trivadis:findUserDataShiro:*"));

    ShiroSecurityPolicy ssp = new
      ShiroSecurityPolicy("classpath:shirosecuritypolicy.ini",
      passPhrase, true, permissionsList);
    onException(UnknownAccountException.class,
      IncorrectCredentialsException.class,
       LockedAccountException.class,
      AuthenticationException.class)
       .to("mock:authenticationException");
    from("direct:findUserDataShiro").
      routeId("findUserDataShiro").policy(ssp).beanRef(
       "userDataBean",
      "loadUserData").to("direct:calculateCategoryShiro");
  }
}
```

Listing 4

```
List<Permission> permissionsList = new
  ArrayList<Permission>();
Permission permission = new
  WildcardPermission("trivadis:calculateCategory:*");
permissionsList.add(permission);

ShiroSecurityPolicy ssp = new
  ShiroSecurityPolicy("classpath:shirosecuritypolicy.ini",
  passPhrase, true, permissionsList);
// onException wie in UserDataShiroRouteBuilder zuvor    onExceptio
n(CamelAuthorizationException.class).continued(true);      from("di
rect:calculateCategoryShiro").routeId(
  "calculateCategoryShiro").policy(ssp).beanRef(
  "categoryBean", "processData");
```

Listing 5

```
@Test
public void testCompleteRouteWithValidUser() throws Exception {
  ShiroSecurityToken sst = new
    ShiroSecurityToken("userEditor", "secret2");
  ShiroSecurityTokenInjector ssti = new
    ShiroSecurityTokenInjector(sst, passPhrase);
  UserData ud = template.requestBodyAndHeader(
    "direct:findUserDataShiro", 1234567890,
    "SHIRO_SECURITY_TOKEN", ssti.encrypt(), UserData.class);
  assertEquals(USERDATA_COMPLETE, userData.toString());
}

@Test
public void testFirstRouteWithValidUser() throws Exception {
  ShiroSecurityToken shiroSecurityToken = new
    ShiroSecurityToken("userAgent", "secret1");
  ShiroSecurityTokenInjector ssti = new
    ShiroSecurityTokenInjector(sst, passPhrase);
  UserData ud = template.requestBodyAndHeader(
    "direct:findUserDataShiro", 1234567890,
    "SHIRO_SECURITY_TOKEN", ssti.encrypt(), UserData.class);
  assertEquals(USERDATA_PARTIAL, userData.toString());
}
```

Listing 6

oder nach „beanRef“ zur Route hinzugefügt wird. Deutlich wird dies bei einem ungültigen Benutzer oder einem ungültigen Passwort. Folgt „policy“ direkt „from“ wie im Listing zuvor, enthält die Nachricht den vom Client (beziehungsweise JUnit-Test) eingegebenen Wert, beispielsweise „1234567890“. Folgt „policy“ dagegen erst auf „beanRef“, wurde der erste Teil der Route schon ausgeführt, bevor es zur Exception kommt. Die Nachricht enthält daher schon die in der Bean ermittelten Daten und gibt damit womöglich bereits sensible Daten preis.

Obiger Code-Ausschnitt lässt allerdings vermissen, welche der in der „shirosecuritypolicy.ini“ festgelegten Berechtigungen für welchen Abschnitt der Route gelten. Die Java-Konfiguration (auch Apache Shiro kann per XML in Spring konfiguriert werden, allerdings fühlt sich hier die Java-Variante deutlich einfacher und intuitiver an) der Route wird damit etwas ausführlicher, wie Listing 4 zeigt.

Wichtig ist damit zumindest ein Permission-Objekt (in der Regel aber mehrere), das die Verbindung zwischen den Berechtigungen aus der Policy-Datei mit der konkreten Route herstellt. Die hier alleine verwendete „WildcardPermission („trivadis:findUserDataShiro:*“)“ wird so zur „permissionsList“ hinzugefügt und dem „ShiroSecurityPolicy“-Konstruktor übergeben. Die „onException“-Einträge leiten gleichzeitig alle im Umfeld der Authentifizierung auftretenden Exceptions an einen mock-Endpoint weiter. Exceptions führen somit zum Abbruch der Route (dazu gleich mehr im zugehörigen JUnit-Test „ShiroTest“).

Der zweite Abschnitt der Shiro-Route, der Aufruf der „CategoryBean“, soll laut Anforderung nur noch für die Benutzergruppen „editor“ und „admin“ möglich sein. Allerdings soll bei Benutzern der Gruppe „agent“ keine Exception geworfen, sondern die Nachricht im aktuellen Zustand, also ohne berechnete Kategorie zurückgegeben werden. Der „CategoryShiroRouteBuilder“ ist dabei überwiegend identisch zum „UserDataShiroRouteBuilder“ aufgebaut. Die Permission verweist lediglich auf „WildcardPermission(«trivadis:calculateCategory:*»)“ und die Route ist entsprechend angepasst. Wichtig ist allerdings die Angabe des weiteren onException-Eintrags „onException(CamelAuthorizationException.class).continued(true)“. „continued(true)“ sorgt dafür, dass die Exception nicht an den Aufrufer zurückgelangt und vor allem die restliche Route ganz normal verarbeitet wird. Mit der „continued“-Angabe sollte man bei allen Routen vorsichtig sein. Unter Umständen wird dadurch eine „CamelAuthorizationException“ ignoriert und der nächste Channel führt die Verarbeitung fort, obwohl die gesamte Route hätte abgebrochen werden müssen (siehe Listing 5).

Testen der Route

War der Aufruf von verschlüsselten oder signierten Apache-Camel-Funktionen im letzten Artikel noch transparent für den Anwender etwa über soapUI möglich, benötigt die Shiro-Integration einen Header-Eintrag „SHIRO_SECURITY_TOKEN“ mit ei-

nem verschlüsselten Benutzernamen und Passwort als Wert. Tests sind im Vergleich dazu am einfachsten über einen JUnit-Test (Klasse „ShiroTest“) möglich (siehe Listing 6).

Diese beiden Tests (weitere finden sich in der Klasse) zeigen die Nachrichten-Inhalte bei unterschiedlichen Benutzergruppen an. Der erste Test ist dabei vollkommen valide, beide Routen-Abschnitte werden erfolgreich durchlaufen. Die erwartete Rückgabe enthält daher ein vollständiges UserData-Objekt, samt berechneter Kategorie. Der zweite Test scheitert, mangels Berechtigung des Benutzers userAgent, beim Ausführen des zweiten Routen-Abschnitts und enthält im zurückgelieferten UserData-Objekt somit anstatt der berechneten Kategorie eine Null.

Typische Fehler und Probleme

Sämtliche Exceptions rund um die Authentifizierung und Autorisierung sind wie immer in Camel gewrappt. Statt einer „IncorrectCredentialsException“ erhält der Aufrufer somit zunächst eine allgemeine „CamelExecutionException“ und muss sich über „getCause()“ zur auslösenden Ursache vorarbeiten.

Fehlt das „SHIRO_SECURITY_TOKEN“ im Header der Nachricht, wirft Camel in Version 2.9.1 eine leider unbrauchbare„NullPointerException“. Ab 2.9.2 verrät „getCause()“ deutlich hilfreicher einen fehlenden Eintrag im Header mit „org.apache.camel.NoSuchHeaderException: No ‚SHIRO_SECURITY_TOKEN‘ header available“.

Spring Security

Als Alternative zu Apache Shiro steht Spring Security [3] zur Verfügung. Vielen dürfte die Verwendung dieses Frameworks bzw. der zugehörigen Camel-Komponente auch näher liegen, wird doch ohnehin schon Spring im Projekt eingesetzt. Die Camel-Spring-Security-Komponente [4] bietet, wie bereits die Camel-Shiro-Komponente, einen im Vergleich zum Original eingeschränkten Funktionsumfang rund um Authentifizierung und Autorisierung. Spring-typisch findet hier sämtliche Konfiguration problemlos in den bekannten XML-Dateien statt. Zur Verwendung von Spring Security muss die folgende Abhängigkeit in der POM-Datei eingegeben sein (siehe Listing 7).

```
<dependency>
 <groupId>org.apache.camel</groupId>
 <artifactId>camel-spring-security</artifactId>
 <version>2.9.2</version>
</dependency>
```

Listing 7

```
<authorizationPolicy id="agent"
  access="ROLE_AGENT, ROLE_EDITOR, ROLE_ADMIN"
  authenticationManager="authenticationManager"
  accessDecisionManager="accessDecisionManager"
  xmlns="http://camel.apache.org/schema/spring-security"/>
```

Listing 8

```
<policy ref="agent">
  <bean ref="userDataBean" method="loadUserData" />
  <to uri="direct:calculateCategorySpring" />
</policy>
```

Listing 9

```
<onException>
  <exception>
    org.springframework.security.access.AccessDeniedException
  </exception>
  <handled><constant>true</constant></handled>
  <process ref="accessDeniedProcessor" />
</onException>
```

Listing 10

```
public class AccessDeniedProcessor implements Processor {
  @Override
  public void process(Exchange exchange) throws Exception {
    UserData body = exchange.getIn().getBody(UserData.class);
    Exception ex =
      exchange.getProperty(Exchange.EXCEPTION_CAUGHT,
      Exception.class);
    // logger
    exchange.getIn().setBody(body);
  }
}
```

Listing 11

```
@Test
public void testRouteWithValidEditorUser() throws Exception {
  UsernamePasswordAuthenticationToken token =
    new UsernamePasswordAuthenticationToken("userEditor",
    "secret2");
  Subject subject = new Subject();
  subject.getPrincipals().add(token);

  UserData userData =
    template.requestBodyAndHeader("direct:findUserDataSpring",
    1234567890, Exchange.AUTHENTICATION, subject,
    UserData.class);
  assertEquals(USERDATA_COMPLETE, userData.toString());
}
```

Listing 12

```
Principal: org.springframework.security.authentication.UsernamePas
swordAuthenticationToken@67fe8a64: Principal: userAgent; Credenti-
als: [PROTECTED]; Authenticated: false; Details: null; Not granted
any authorities
```

Listing 13

Konfiguration

Der Dreh- und Angelpunkt bei der Verwendung von Spring Security mit Apache Camel sind authorizationPolicy-Elemente (siehe Listing 8).

Neben der ID müssen zusätzlich die erlaubten Rollen per „access“, ein „authenticationManager“ und ein „accessDecisionManager“ angegeben werden. Diese beiden Manager bieten keine wirklichen Überraschungen und basieren auf Spring-Security-Standards. Lediglich beim access-Attribut ist eine Angabe von mehreren, durch Komma separierten Rollen möglich. Soll nun ein vorhandener Routenabschnitt, beispielsweise der Aufruf einer weiteren Route oder auch nur einer Bean, geschützt werden, wird der zu schützende Teil einfach in ein policy-Element mit einem ref-Attribut, beispielsweise „ref=“agent““, eingeschlossen (siehe Listing 9).

Das ref-Attribut verweist dabei auf genau eine „authorizationPolicy“, eine Kombination mit beispielsweise einer Kommagetrennten Liste ist leider nicht möglich. Damit dürfte auch klar werden, warum das „authorizationPolicy-agent“-Beispiel zuvor im access-Attribut mehrere Rollen listet: Auf Routen, auf die ein Benutzer mit der Rolle „agent“ Zugriff hat, dürfen auch die Rollen „editor“ oder ein „admin“ zugreifen.

Ohne weitere Konfiguration sind die Routen damit vollständig abgesichert. Allerdings endet der Aufruf der zweiten Route für Benutzer mit der Rolle „agent“ mit einer „AccessDeniedException“. Allgemein ausgedrückt: Wer keine oder ungenügende Rechte hat, wird abgewiesen. In den meisten Fällen ist das sicherlich auch so gewünscht, allerdings soll im Beispiel anstelle der Exception der Nachrichteninhalt zurückgeliefert werden, zumindest soweit er zum Zeitpunkt der Exception schon existiert. Bei der Rolle „agent“ wären das etwa die Benutzerdaten zur eingegebenen ID, jedoch ohne berechnete Kategorie.

Um diese Vorgabe umzusetzen, sind ein zusätzlicher Prozessor und eine „onException“-Konfiguration notwendig. Das onException-Element legt dabei fest, dass eine „AccessDeniedException“ behandelt wird und einen Camel Processor zu durchlaufen hat (siehe Listing 10).

Auf die Tests mit ungültigem Benutzer und/oder Passwort hat diese Konfigurationsänderung keine Auswirkungen. In diesem Fall lautet die Exception „BadCredentialsException“ und wird damit von dieser „onException“-Konfiguration nicht erfasst.

Auch der „AccessDeniedProcessor“ birgt keine wirklichen Überraschungen. Er protokolliert lediglich die abgefangene Exception sowie den ursprünglichen Nachrichteninhalt und gibt den bis zu diesem Zeitpunkt vorhandenen Nachrichteninhalt an den Aufrufer zurück (siehe Listing 11).

Testen der Route

Auch mit der Camel-Spring-Security-Komponente fällt das Testen über einen JUnit-Test (Klasse „SpringSecurityTest“), nicht zuletzt wegen des erforderlichen Authentication Headers, deutlich leichter. Beispielhaft sei hier eine der Testmethoden aufgeführt (siehe Listing 12).

Das so eingefügte Subject samt „UsernamePasswordAuthenticationToken“ für den Benutzer „userEditor“ sorgt im Message-Header für folgenden Eintrag, der von Spring Security bei Verwendung eines policy-Elements überprüft wird (siehe Listing 13).

Da dieser Benutzer über alle notwendigen Rechte verfügt, ist das zurückgelieferte UserData-Objekt vollständig und enthält auch die berechnete Kategorie.

Typische Fehler und Probleme

Auch mit Spring Security sind sämtliche Exceptions gewrappt. Statt einer „BadCredentialsException“ liefert Camel zunächst eine allgemeine „CamelExecutionException“ zurück. Über „getCause()“ gelangt man dann auch hier zur eigentlichen Ursache.

Die oben gezeigte „onException“-Konfiguration ist nicht immer für andere Routen(-abschnitte) folgenlos. Bei nicht eindeutiger Exception ist es dann notwendig, mit einem choice-Element zu überprüfen, mit welcher Rolle oder welchem Benutzer die Exception aufgetreten ist. Diese kann dann je nach Ergebnis unterschiedlich behandelt werden.

Fazit

Auch bei der in Integrationsprojekten häufiger anzutreffenden Anforderung zur Absicherung von Routen per Authentifizierung und Autorisierung bietet Apache Camel wie gezeigt zahlreiche verschiedene Möglichkeiten an. Die Camel-Varianten der Security-Frameworks Apache Shiro

und Spring Security enthalten dabei nur die Features, die im Camel-Umfeld den größten Nutzen versprechen: Authentifizierung und Autorisierung.

Die grundsätzlich angebotene Funktionalität ist in beiden Komponenten ähnlich. Ob man sich nun für Apache Shiro oder Spring Security entscheidet, hängt neben den Kenntnissen im Entwicklerteam sicherlich auch von den weiteren im Projekt eingesetzten Frameworks ab. Für Spring-Entwickler liegt der Schritt zu Spring Security einfach näher. Subjektiv betrachtet lässt sich Spring Security auch leichter und intuitiver für den Camel-Einsatz konfigurieren. Gleichzeitig stellt Spring Security, im Gegensatz zum deutlich jüngeren Apache Shiro, auch eine umfangreichere Dokumentation zur Verfügung. Wer bisher weder das eine noch das andere Framework verwendet hat, dürfte mit Spring Security einen reibungsloseren Start hinlegen.

Die Konfiguration ist mit beiden Frameworks, vor allem je nach Anzahl der unterschiedlichen Rollen und Routen, aufwändig. Das ist allerdings nicht unbedingt Camel selbst oder den beiden Security-Frameworks Apache Shiro und Spring Security geschuldet, sondern allgemein der leider vorhandenen Komplexität beim Thema „Sicherheit". Außerdem kommt hinzu, dass hier Sicherheitsaspekte mit Routing-Aspekten vermischt werden. Eine (fachlich motivierte) Route enthält so neben der eigentlichen Routen-Logik auch Betriebsaspekte rund um die Sicherheit. Eine derartige Vermischung erhöht die Komplexität deutlich und macht die Konfiguration schwieriger verständlich und auch wartbar.

Doch trotz dieser Kritik: Zusammen mit der im vorherigen Artikel gezeigten Payload Security können Camel-Routen somit nun umfassend gesichert und Nachrichten sicher ausgetauscht werden. Das nächste Integrationsprojekt mit Apache Camel kann daher nicht nur zahlreiche Systeme einfach und flexibel einbinden, sondern gleichzeitig auch bei der Sicherheit einen großen Schritt in die richtige Richtung machen.

Weiterführende Informationen

[1] http://shiro.apache.org
[2] http://camel.apache.org/shiro-security.html
[3] http://www.springsource.org/spring-security
[4] http://camel.apache.org/spring-security.html

Dominik Schadow
dominik.schadow@
trivadis.com

Dominik Schadow arbeitet als Senior Consultant im Bereich „Application Development" bei der Trivadis GmbH in Stuttgart. Neben seinem Fokus auf Java-Enterprise-Applikationen und -Architekturen sowie Integrationsthemen rund um Apache Camel beschäftigt er sich als Discipline Manager Application Development Security mit dem Thema „Sichere Software-Entwicklung". In seiner Freizeit leitet er das Open-Source-Projekt „JCrypTool" , mit dem Anwendern die Kryptografie näher gebracht werden soll.

„Ich hasse Kaffee, aber ich mag Java und Mac OS X …“

Java auf dem Mac führt ein unberechtigtes Schattendasein in der Java-Community. Ein Interview dazu mit Scott Kovatch, dem Mac OS X Projektleiter für OpenJDK.

Wie entwickelt sich Java auf dem Mac?
Scott Kovatch: Im Oktober 2010 haben sich Apple und Oracle zusammengetan und Mac Ports für OpenJDK angekündigt. Dabei handelt es sich um ein Open-Source-Projekt. Damit hatte Apple grundsätzlich die Möglichkeit, seinen gesamten Code wieder in das OpenJDK zurückzubringen. Wir brauchten ein neues Projekt, weil das Abstract-Window-Toolkit (AWT) auf dem Mac in vielen Teilen einfach zu dicht am OS dran war. Es gab viele interne APIs und ähnliche Dinge. Ein aus Ingenieuren von Apple und Oracle zusammengesetztes Team hat harte Arbeit geleistet und ein neues AWT für Java 7 entwickelt. Der gesamte Code ist Teil des Projekts, das sich jeder ansehen kann.

Ist das eine Art Kooperationsvertrag, so ähnlich wie mit IBM?
Scott Kovatch: Ja, das stimmt so weit. Es heißt allerdings keineswegs, dass Apple jetzt seine Zelte abbricht, etwa salopp ausgedrückt.

Und wie ist der heutige Stand der Dinge?
Scott Kovatch: Mehrere unterschiedliche Gruppen hegen Interesse an diesem Projekt – allen voran Apple. Sie wollen sicherstellen, dass sowohl Kunden als auch Entwickler Java direkt von der Quelle bekommen, sprich von Oracle. Oracles Ziel ist es sicherzugehen, dass Java weiterhin offen, überall zugänglich und aktuell ist. Der große Vorteil für Entwickler besteht darin, dass sie das neue JDK zeitgleich plattformübergreifend erhalten. Im Laufe des nächsten Jahres wird es also auf dem Mac keine zwei oder drei Monate veraltete Version gegenüber Windows und Solaris mehr geben. Sie wird zum selben Zeitpunkt herauskommen wie für Windows und Solaris.

Wenn es dann bestimmte Updates für die kommerzielle Version gibt, die dann auf dem Markt ist, wird es auch zeitgleich eine Mac-Version geben?
Scott Kovatch: Das ist richtig. Als Nächstes steht Java 7 Update 6 an. Das wird Mitte 2012 erwartet, und wir gehen davon aus, dass es dann plattformübergreifend verfügbar sein wird.

Was ist der augenblickliche Stand dieser Übergangsphase?
Scott Kovatch: Für Mac OS 10.6 gibt es Java bereits. Dabei handelt es sich um eine Java-6-Version. Auch wenn auf Mac OS 10.7 Java nicht vorhanden ist, kann es einfach installiert werden. Es reicht, einfach nur „Java“ einzugeben oder irgendetwas mit Java zu machen, und schon wird gefragt, ob man Java installieren möchte. Künftig wird Java 7 für den Mac immer über Oracle erhältlich sein. Sobald Oracle Java 7 freigibt, wird Apple das OS abändern, um den automatischen Download von Java 6 zu unterbinden und den User auf java.com oder oracle.com zu verweisen, wo die aktuellste Version von Java 7 heruntergeladen werden kann.

Was ist mit den Anwendungen, die bereits vorhanden sind oder die von einem bestimmten JRE abhängig sind? Werden die dann auf die nächste Java-Version umgestellt?
Scott Kovatch: Wer bereits eine Java-6-basierte Anwendung nutzt, muss diese App für Java 7 neu installieren. Mit Java 7 lässt sich die Apple-Code-Basis nicht nutzen. Das wird aber völlig unkompliziert sein, denn wir stellen alle dazu benötigten Tools bereit. Für die Zukunft gehen wir davon aus, dass die meisten Java-Apps gebündelt werden, um dann in den Mac-App-Store zu kommen.

Wie geht App-Store mit den Java-Apps um?
Scott Kovatch: Es gibt die weitverbreitete Fehleinschätzung, dass Java-Apps im App-Store nicht zugelassen sind. Nicht erlaubt sind Apps, die auf einer vorinstallierten Java-Version basieren. Sprich, die einzige Möglichkeit, eine Java-App in den Store zu bringen, besteht darin, Java in der App zu bündeln.

Allerdings gibt es auch einen Haken an der Sache: Man kann keine JREs zwischen den Apps austauschen.
Scott Kovatch: Das ist richtig. Der andere Nachteil besteht darin, dass im Falle der

Veröffentlichung einer neuen JRE-Version die App neu gebündelt und dann an den App-Store geschickt werden muss. Das heißt im Grundsatz: Apple übernimmt die Verantwortung für die Bandbreite, um unsere Updates für uns zu hosten. Bekommt jemand also eine neue Version oder braucht er ein neues JRE, dann lädt er es im App-Store hoch und User aktualisieren sie auf diesem Wege.

Wie ist der Stand beim „Mac OS X OpenJDK"-Projekt?

Scott Kovatch: Es ist so gut wie alles inzwischen implementiert. Wir haben auf der JavaOne 2011 eine Beta-Version veröffentlicht. Betas werden auch im Laufe der nächsten Monate weiter herauskommen. Jetzt ist aber auch wirklich so gut wie alles dabei, auch Hotspot, an das man sich lange Zeit nicht herangewagt hat, weil alles Intel-spezifischer Code ist. Ist Hotspot einmal zum Laufen gebracht, braucht man so gut wie nichts mehr damit zu machen. Es gibt da noch ein paar Dinge hinsichtlich Bedienbarkeit, die im Augenblick noch nicht richtig funktionieren. Man muss schauen, ob diese überhaupt umgesetzt werden. Noch hat keiner herausgefunden, wie man es am besten macht.

Und die Core Library sitzt?

Scott Kovatch: So ist es. Gut, dass Mac OS Unix-basiert ist. So konnte eine ganze Menge Code von Solaris und Linux einfach übernommen werden, und das läuft einwandfrei. An der NIO Version 2 ist noch etwas zu machen und einige Async-IO-Implementierungen sind noch auszuarbeiten. Ein paar Probleme gibt es auch noch mit IPv6. Ich weiß aber, dass das Core-Library-Team sich der Probleme durchaus bewusst ist und intensiv daran arbeitet.

Man kann sich gut vorstellen, dass das eigentliche Problem in dem Moment entstand, als klar wurde, dass einige native Parts der AWT so stark abhängig sind und dass die zahlreichen Schwierigkeiten genau hier zu erwarten sind.

Scott Kovatch: Ja, das stimmt. AWT wurde für Java 7 von vorn bis hinten komplett umgeschrieben. Was man jetzt hat, ist ein NSWindow auf Spitzen-Niveau. Sämtliche Inhalte dieses Fenster werden jetzt vollständig in Java wiedergegeben. Das Aqua Look & Feel ist geblieben – es war auch schon in 4 und 6 da und wird auch für die 7er-Version übernommen. Aber die Implementierung von dem, was unter der Haube steckt, das Aqua Look & Feel, wird zur Umsetzung der AWT-Bedienelemente genutzt. Man weiß zwar, dass AWT eine etwas veraltete Technologie ist, dennoch werden die Aqua-Swing-Bedienelemente für die Implementierung von AWT eingesetzt. Das ist also quasi das Gegenteil dessen, was man erwarten würde.

Es gibt noch eine ganze Reihe anderer Dinge in Java. Insbesondere wenn es um Releases geht. Plug-in gehört dazu, und Installationen. Wie weit ist man damit?

Scott Kovatch: Plug-in und Web-Start gehören zu den Nicht-Open-Source-Teilen von JDK. Ich arbeite im Moment intensiv daran. Das ist im Prinzip der Hauptgrund, warum wir noch nicht von einer Full-Version von JDK sprechen. Sämtliche Entwürfe und das Plug-in erfolgen jetzt in Core Animation. Wer jetzt schon mit der Entwicklung von Applets vertraut ist, der findet eine neue Plug-in-Version, mit der ein Applet in einem anderen Prozess eingeführt wird. Damit lässt sich dann eine ganze Reihe von Dingen anstellen, die vorher nicht möglich waren. Hinzu kommen einige andere Pluspunkte wie Sicherheit und Stabilität. Dieses Modell läuft auf dem Mac, allerdings mit der neuen AWT-Version, die das Entwurfsmodell verändert. Das hat einiges an Arbeit gekostet. Das andere spannende Thema bei den Plug-ins für Mac ist, dass es ein eigenes JRE-Bundle geben wird. Für den End-User heißt das, es wird komplett in Java programmiert sein. Man kann dann dies und das installieren, ohne dass man sich jemals wieder Gedanken um Java machen muss.

Es gibt eine Entwickler-Version und eine End-User-Version. Worin unterscheiden sich diese?

Scott Kovatch: Java SE 7 Update 4 wird ein voll funktionstüchtiges JDK beinhalten. Die Herzstücke aus VM, AWT, den Core Libraries und all diesen Dingen werden somit hochwertig sein, damit Entwickler eine komplette Anwendung erstellen können. Es wird alle Tools geben, die man als User erwartet. Es wird sich genauso gut arbeiten lassen wie vorher auch. Man wird es dann per Drop-in ablegen und direkt mit NetBeans loslegen können. Update 6 wird dann die End-User-Version. Sie beinhaltet das Plug-in sowie den notwendigen Support, damit Web-Start-Apps, Jar-Anwendungen und Applets mit Java 7 laufen.

Falls jemand möchte, kann er dann mitmachen? Es ist ja immerhin ein offenes JDK-Projekt.

Scott Kovatch: Klar, wer mitmachen möchte, kann das ganz einfach tun. Man braucht einen Mac, das versteht sich von selbst. Dann besorgt man sich Xcode und installiert die Java-Updates. Auf diese Weise erhält man sämtliche APIs, die Apple freigegeben hat, um das Projekt aufzusetzen und zum Laufen zu bringen. Daneben haben wir eine Wiki-Seite eingerichtet, auf der sämtliche Anleitungen zu finden sind, die man braucht, um loszulegen. Im Grunde zeigt der Mercurial-Befehl zum Kopieren des Workspace an, wie es geht, wie man es installiert, und von da an kann man einfach loslegen, um Bugs zu finden und den einen oder anderen Patch einzureichen.

Aus User-Sicht ist das Ganze bis zu diesem Punkt ziemlich leicht. Wir haben eine veröffentlichte Version und eine Beta-Version.

Scott Kovatch: Stimmt, wir haben die Beta-7-Version, die auf java7.java.net vorgestellt wird. Die wird auch im Laufe der nächsten Monate veröffentlicht und hier und da noch aufgewertet. Man kann aber auch heute schon eine Beta-Version bekommen und direkt anfangen, damit zu arbeiten.

Wie schaut es mit Java FX aus?

Scott Kovatch: Java FX wird als Co-Bundle mit jeder veröffentlichten JDK-Version für Mac kommen. Das startet mit Update 4 und Update 6 und setzt sich dann fort. Java FX wird es immer zeitgleich mit dem Rest von JDK geben. User müssen also nichts weiter tun.

Spannend daran ist, dass zum ersten Mal Java FX in Kombination mit JDK herauskommen wird.

Scott Kovatch: Ja, ich denke, das ist korrekt. Ich glaube, Update 4 wird die erste

Impressum

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

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

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

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

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

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

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

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

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

Java aktuell
Magazin der Java-Community

Version sein, in der FX überall mit dem Rest von JDK im Bundle kommt. Java auf dem Mac war schon immer ein großartiges Qualitätsprodukt. Dass Oracle es jetzt hat, ist deshalb gut, weil die Dinge nun zeitgleich stattfinden, Oracle es steuern kann und wir damit in der Lage sind sicherzustellen, dass es keinerlei Unterschiede im Systemverhalten gibt. Nicht, dass Apple einen schlechten Job gemacht hätte. Aber der direkte Kontakt zu sämtlichen Ingenieuren bei Oracle macht es für alle leichter, sich auf ein bestimmtes Systemverhalten zu einigen.

Da steckt bestimmt viel Engagement in der Zusammenarbeit zwischen Apple und Oracle.

Scott Kovatch: Das ist richtig. Wir haben immer noch ein hervorragendes Arbeitsverhältnis mit Apple. Die Apple-Ingenieure haben mächtig viel Zeit investiert und sie haben daran genauso hart gearbeitet wie wir von Oracle. Ihr Arbeitsbeitrag wird allerdings im Laufe der kommenden Monate langsam auslaufen und Oracle dann direkteren Einfluss auf das Projekt nehmen. Aber sie setzen alles daran, damit wir eine qualitativ hochwertige Java-Version für Mac herausbringen.

Hinweis: Das Interview wurde auf https://blogs.oracle.com/javaspotlight/entry/java_spotlight_episode_83_scott veröffentlicht.
Aus dem Amerikanischen übersetzt von Sabine Redlich.

Scott Kovatch
scott.kovatch@oracle.com

Scott Kovatch ist bei Oracle als Lead Engineer für die Mac Ports tätig und seit 1996 in der ein oder anderen Form mit JavaVMs unterwegs. Er hat auch eine Weile für MatchWorks gearbeitet. Außerdem gehörte er über acht Jahre zum Apple Java-Team und hat auch einige Jahre im SWT-Team mit IBM zusammengearbeitet.

Für die deutsche Java-Community kommt der Zeitpunkt zu früh, an dem Oracle den kostenlosen Support für JDK 6 einstellen will

Der iJUG hat über seine Online-Medien sowie auf dem Java Forum Stuttgart eine Umfrage zu Java 7 durchgeführt. Demnach setzen erst 51 der 234 Befragten Java 7 bereits produktiv ein. Weitere 33 nutzen aufgrund von Problemen im produktiven Einsatz Java 7 nur zum Testen. Knapp die Hälfte aller Befragten (116) hält es für nicht akzeptabel, dass Oracle den kostenlosen Support (Updates, Patches) für JDK 6 bereits im November 2012 einstellen will. Sie sind über diesen Termin hinaus auf das Produkt angewiesen.
Tobias Frech, iJUG-Vorstand und Board-Mitglied der Java User Group Stuttgart meint dazu: „Das Thema „Update auf Java 7" ist bei vielen Herstellern von Java-basierter Software noch gar nicht richtig angekommen. Oracle geht zwar mit erfreulichem Beispiel voran und stellt das mit Java 7 kompatible Release seiner Fusion Middleware zur Verfügung. Jedoch gilt auch hier, dass ein Update auf die neueste Version notwendig ist."
„Der Druck von Oracle auf die Java-Comunity ist zu hoch, die Anwender können damit nicht Schritt halten", sagt Fried Saacke, Vorstandsvorsitzender des iJUG. „Wir adressieren damit die dringende Bitte an Oracle, den Support um ein Jahr zu verlängern, damit Zeit genug bleibt, die Anwendungen umzustellen."
Auf die Frage, ob das neue JDK 7 ihre Bedürfnisse erfüllt, antworteten 73 der Befragten mit „ja", 84 mit „weitgehend", 25 mit „nur zum Teil" und 19 mit „nein".

Windows Azure Reloaded – neue Möglichkeiten für Java-Entwickler

Holger Sirtl, Microsoft Deutschland GmbH

Java-Entwickler hatten bislang gewisse Vorbehalte gegenüber der Bereitstellung von Anwendungen auf Windows Azure. Bislang als Platform-as-a-Service-Angebot konzipiert, mussten Paketierung und Deployment Java-basierter Anwendungen speziell für Azure erfolgen. Mit dem Juni-Release von Windows Azure ändert sich dies grundlegend. Einige der dort enthaltenen Änderungen wie der Betrieb eigener beziehungsweise vorgefertigter persistenter virtueller Maschinen oder virtuelle Netzwerke zwischen lokalen und Cloud-Ressourcen machen Azure für Java-Entwickler zur ernsthaften Alternative für die Bereitstellung eigener Anwendungen in der Cloud.

Windows Azure war lange Zeit primär ein Platform-as-a-Service-Angebot. Es stellte verschiedene Plattform-Dienste bereit, die unter anderem dazu genutzt werden konnten, speziell paketierte Anwendungen in der Cloud auszuführen. Setup, Betrieb und Wartung der hierzu benötigten virtuellen Maschinen erfolgte vollautomatisiert durch Windows Azure. Hierüber war es möglich, mit minimalem administrativen Aufwand sehr schnell Umgebungen für hochskalierende Web-Anwendungen in der Cloud bereitzustellen. Dieses Ausführungsmodell für Websites steht weiterhin zur Verfügung und wird als „Windows Azure Cloud Services" bezeichnet. Im Betrieb bietet es durch automatisiertes Management (Patches und Upgrades der Gast-Betriebssysteme in den VMs werden automatisch eingespielt) und insbesondere bei der Skalierung der Umgebung (im Bedarfsfall kann die Zahl der Instanzen und die Konfiguration des vorgeschalteten Loadbalancers automatisch angepasst werden) hohen Komfort. Aus Sicht von Java-Entwicklern besitzen Windows Azure Cloud Services aber zwei Einschränkungen: Zum einen müssen Anwendungen speziell für Windows Azure paketiert werden. Dabei müssen auch alle für Java benötigten Laufzeitkomponenten (inkl. JDK, etwaige Application Server etc.) in das Deployment-Paket aufgenommen werden. Windows Azure erstellt dann auf Basis verschiedener VM-Vorlagen und entsprechender im Deployment-Paket enthaltener Artefakte VM-Instanzen und führt diese aus. Die zweite Einschränkung betrifft die Persistierung von Daten. Die in den virtuellen Maschinen von Cloud Services enthaltenen Festplattenlaufwerke behalten ihre Inhalte nur für die Laufzeit der virtuellen Maschine. Sobald eine VM einem Re-Image unterzogen wird (etwa bei Ausfall der betreffenden Hardware), werden die Laufwerks-Inhalte auf den Zustand zum Zeitpunkt des Deployment zurückgesetzt. Bei einer Speicherung in

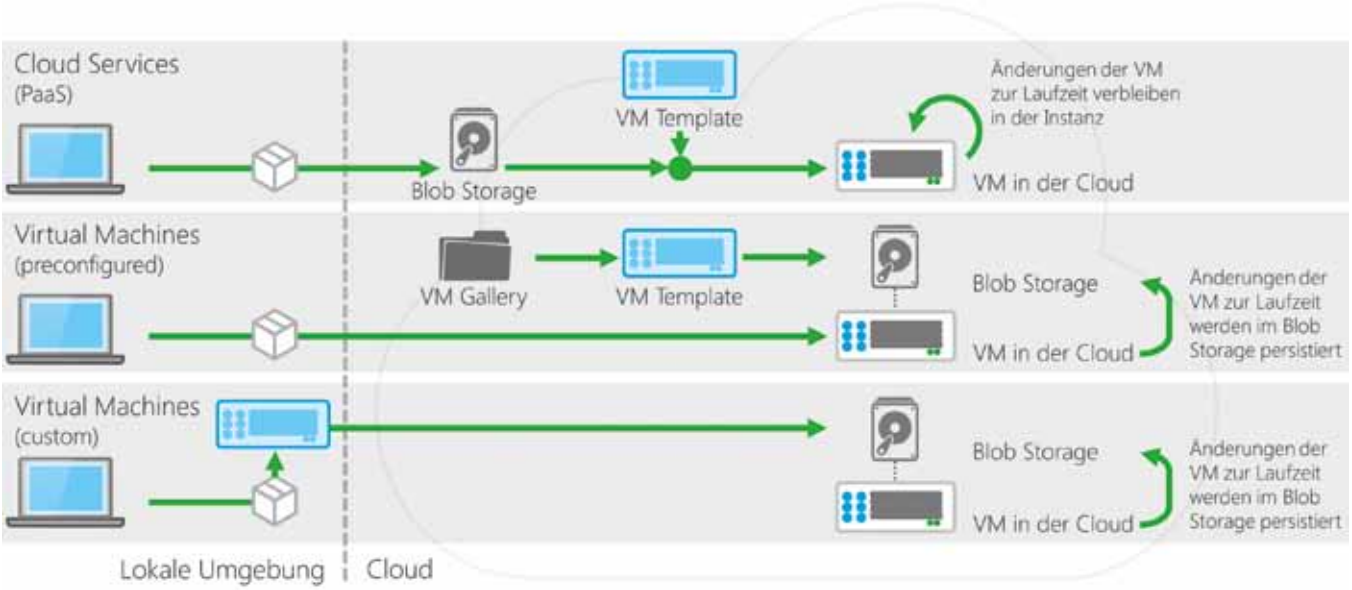


Abbildung 1: Ausführungsmodelle für Java-basierte Anwendungen in Windows Azure

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<ServiceDefinition xmlns="http://schemas.microsoft.com/ServiceHo-
sting/2008/10/ServiceDefinition" name="WindowsAzureProject">
  <WorkerRole name="WorkerRole1" vmsize="Small">
    <Startup>
      <Task commandLine="util/.start.cmd startup.cmd"
executionContext="elevated" taskType="simple"/>
    </Startup>
    <Runtime executionContext="elevated">
      <EntryPoint>
            <ProgramEntryPoint commandLine="run.cmd"
setReadyOnProcessStart="true"/>
      </EntryPoint>
    </Runtime>
    <Imports />
    <Endpoints>
      <InputEndpoint localPort="8080" name="http" port="80"
protocol="tcp"/>
    </Endpoints>
  </WorkerRole>
</ServiceDefinition>
```

Listing 1: Definitionsdatei eines Windows Azure Cloud Service

```
SET SERVER_DIR_NAME=apache-tomcat-7.0.28
SET WAR_NAME=HelloWorld.war

rd "\%ROLENAME%"
mklink /D "\%ROLENAME%" "%ROLEROOT%\approot"
cd /d "\%ROLENAME%"

cscript /NoLogo "util\download.vbs" "http://hsirtl.blob.core.win-
dows.net/tomcat/jdk.zip" "jdk.zip"
cscript /NoLogo "util\unzip.vbs" jdk.zip „%CD%"

cscript /NoLogo "util\download.vbs" "http://hsirtl.blob.core.win-
dows.net/tomcat/tomcat.zip" "tomcat.zip"
cscript /NoLogo "util\unzip.vbs" tomcat.zip „%CD%"

copy %WAR_NAME% "%SERVER_DIR_NAME%\webapps\%WAR_NAME%"
cd "%SERVER_DIR_NAME%\bin"
set JAVA_HOME=\%ROLENAME%\jdk1.7.0_05
set PATH=%PATH%;%JAVA_HOME%\bin
cmd /c startup.bat

@ECHO OFF
if %ERRORLEVEL%==0 exit %ERRORLEVEL%
choice /d y /t 5 /c Y /N /M "*** Windows Azure startup failed -
exiting..."
exit %ERRORLEVEL%
```

Listing 2: Startup-Skript zur Installation von Tomcat in einer Cloud-Service-Instanz

einem der VM-internen Laufwerke (etwa in einer MySql-Installation in der VM) waren die Inhalte also nicht vor Verlust geschützt. Sofern die Datenspeicherung außerhalb der VM vorgenommen wird (beispielsweise in Windows Azure Storage oder SQL Database, dem relationalen Datenbankdienst von Windows Azure), ist dies kein Problem.

Das Juni-2012-Release – Windows Azure wird erwachsen

Mit dem Juni-2012-Release hat sich nun die Situation grundlegend geändert. Eine Vielzahl an Neuerungen machen Azure nun auch für Java-Entwickler zu einer interessanten Möglichkeit, Services der Plattform in eigenen Programmen zu nutzen und eigene Anwendungen in der Cloud zu betreiben. Neben Media Services, die die Implementierung von Workflows zur Erstellung, Verarbeitung und Ausbringung von Medieninhalten ermöglichen, einem erweiterten Caching-Service zur In-Memory-Zwischenspeicherung von Daten, bieten insbesondere die Windows Azure Virtual Machines und das Virtual Network gänzlich neue Möglichkeiten zum Aufbau und Betrieb von Java-basierten Cloud Apps in Windows Azure.

Drei Ausführungsmodelle für Java-basierte Cloud-Anwendungen

Mit den Virtual Machines stehen für Java-Entwickler nun drei Möglichkeiten offen, ihre Anwendungen auf Windows Azure zu betreiben: Cloud Services, vorkonfigurierte und selbst erstellte Virtual Machines. Abbildung 1 skizziert die Alternativen.

Bei Cloud Services werden die auszuführenden Anwendungen als Deployment-Paket im Blob-Storage (Blob = binary large objects) von Windows Azure gespeichert. Azure erstellt daraufhin auf dieser Basis entsprechende virtuelle Maschinen.

Bei Virtual Machines haben Nutzer die Möglichkeit, auf Basis von vorgefertigten oder eigenen virtuellen Maschinen Anwendungen in Windows Azure zu betreiben. Im Gegensatz zu Cloud Services werden dabei die VM-Images über sogenannte „Disks“ im Blob-Storage gehalten und zur Laufzeit aktualisiert. Änderungen bleiben somit erhalten, selbst wenn die VMs einem Re-Image unterzogen werden.

Cloud Services

Cloud Services bleiben weiterhin die beste Alternative in Szenarien, in denen minimaler administrativer Aufwand bei Bereitstellung und Betrieb einer Cloud-Anwendung betrieben werden soll. Zur Bereitstellung muss nur ein Deployment-Paket über das Service-Portal [1] nach Windows Azure geladen werden. Das Setup der zum Betrieb benötigten virtuellen Maschinen erfolgt vollautomatisch. Entwickler, die Eclipse als Entwicklungsumgebung verwenden, können sich unter [2] das „Windows Azure Software Development Kit für Java" (einschließlich eines Eclipse-Plug-ins) herunterladen. Darin enthalten ist eine Projekt-Vorlage, die es Entwicklern erleichtert, ein Azure-Deployment-Paket zu erstellen. Abbildung 2 zeigt den Projektaufbau.

In den über das Plug-in erstellten Projekten sind unter anderem Zertifikate für Remote-Desktop-Zugriffe auf VM-Instanzen des Cloud Service, ein Ant-Build-Skript sowie Kompilierungs- und Paketierungs-Werkzeuge enthalten. Zentrale Bedeutung für das automatische Setup der virtuellen Maschinen hat die Service-Definitionsdatei „csdef". Dort sind die Komponenten des Service sowie die Größen der zu konfigurierenden virtuellen Maschinen definiert. Listing 1 zeigt beispielhaft eine Definitionsdatei.

Um Java-basierte Anwendungen in den Cloud-Service-Instanzen ausführen zu können, müssen noch eine Java Runtime und eventuell benötigte Anwendungsserver installiert sein. Dies geschieht am einfachsten über eine sogenannte „Startup-Task". Diese ist in der Definitionsdatei festgelegt. In Listing 1 wird angegeben, dass das Skript „startup.cmd" beim Hochfahren einer Instanz ausgeführt werden soll. Listing 2 zeigt ein Beispiel für dieses Skript.

In diesem Skript werden das JDK und Tomcat heruntergeladen und installiert. Eine als WAR-Datei mitpaketierte Java-Anwendung wird in diese Umgebung installiert und ausgeführt. Vorteil der Verwendung von Cloud Services als Ausführungs-Alternative ist die Möglichkeit, bei sich ändernder Last schnell Instanzen des Service hinzuzuschalten beziehungsweise herunterzufahren. VM- und Umgebungs-Management werden durch Windows Azure übernommen. Die Zahl der Instanzen kann unter anderem über einen Schieberegler im „Windows Azure Management Portal" geändert werden. Auch eine automatisierte Anpassung ist möglich.

Abbildung 2: Projektaufbau eines Windows-Azure-Projekts in Eclipse

```
C:\PS>New-AzureVMConfig -Name "MySUSEVM2" -InstanceSize
ExtraSmall -ImageName (Get-AzureVMImage)[7].ImageName `
| Add-AzureProvisioningConfig -Linux -LinuxUser $lxUser
-Password $adminPassword ` | New-AzureVM -ServiceName
"MySvc2" -AffinityGroup "Contoso"
```

Listing 3: Erstellen einer Linux-basierten Virtual Machine über ein Powershell-Cmdlet

Virtual Machines für Windows und Linux

Die neu in der Plattform enthaltenen Virtual Machines erweitern die Möglichkeiten, Anwendungen in der Cloud auszuführen, bedeutend. Sie erlauben die Ausführung flexibel konfigurierbarer virtueller Maschinen auf Windows Azure. Im Gegensatz zu Cloud Services bleiben hierbei Änderungen zur Laufzeit in den VMs erhalten, selbst wenn ein Hardware-Defekt auftritt oder aus anderen Gründen ein Re-Image der VMs erfolgt. Erreicht wird dies dadurch, dass von den verwendeten Images zunächst im Blob-Storage Drives erzeugt werden, die dann wiederum Basis für die Ausführung als VM sind. Blob-Storage weist einige Eigenschaften auf, die hohe Ausfallsicherheit für die gespeicherten Daten sicherstellen: Inhalte werden im ausgewählten Rechenzentrum grundsätzlich dreifach gespeichert. Die drei Kopien werden stets synchron gehalten, Schreibzugriffe erst dann als erfolgreich quittiert, wenn alle drei Kopien erfolgreich geschrieben sind. Dabei liegen sie an verteilten Orten im Rechenzentrum, um bei Ausfall eines Speichermediums zwei Kopien zu behalten, die dann um eine weitere Kopie wieder ergänzt werden. Darüber hinaus ist Geo-Replikation möglich. Dabei werden die Inhalte noch in ein entferntes Rechenzentrum repliziert. Diese Replikation erfolgt asynchron.

Virtual Machines lassen sich auf zwei Arten erzeugen. Eine Möglichkeit ist die Auswahl vorbereiteter Images aus der VM Gallery im Management-Portal. Über die Option „Quick Create" kann mit zwei Mausklicks eine neue VM erzeugt werden (siehe Abbildung 3). Dort wird eine Windows-Server-2012-Instanz in einem Rechenzentrum in Nord-Europa erzeugt.

Folgende Windows-Versionen werden in Virtual Machines unterstützt:

- Windows Server 2008 R2
- Windows Server 2012
- Windows Server 2008 R2 mit SQL Server 2008 oder 2012 (sobald allgemein verfügbar)

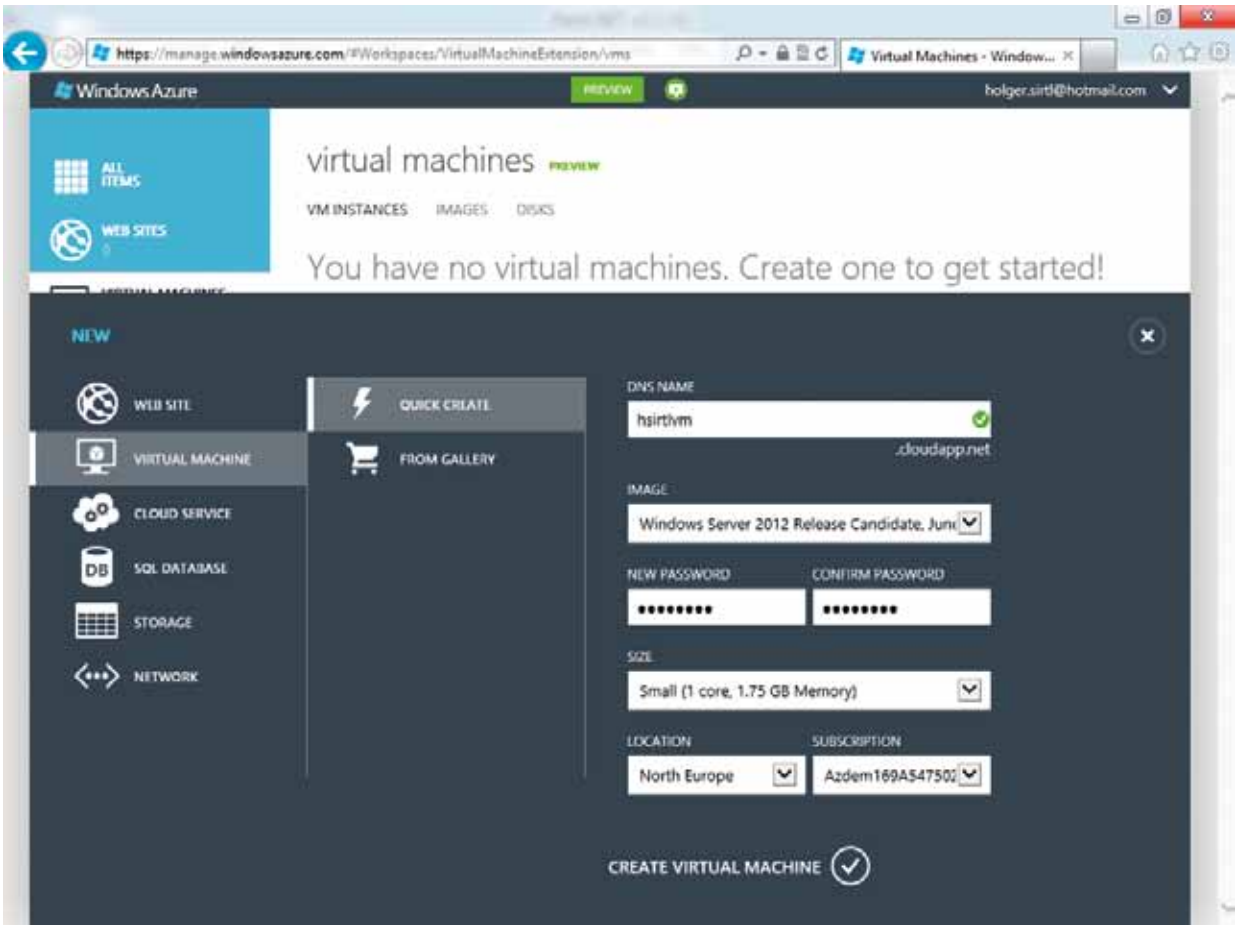


Abbildung 3: Erstellen einer Virtual Machine mittels Quick Create

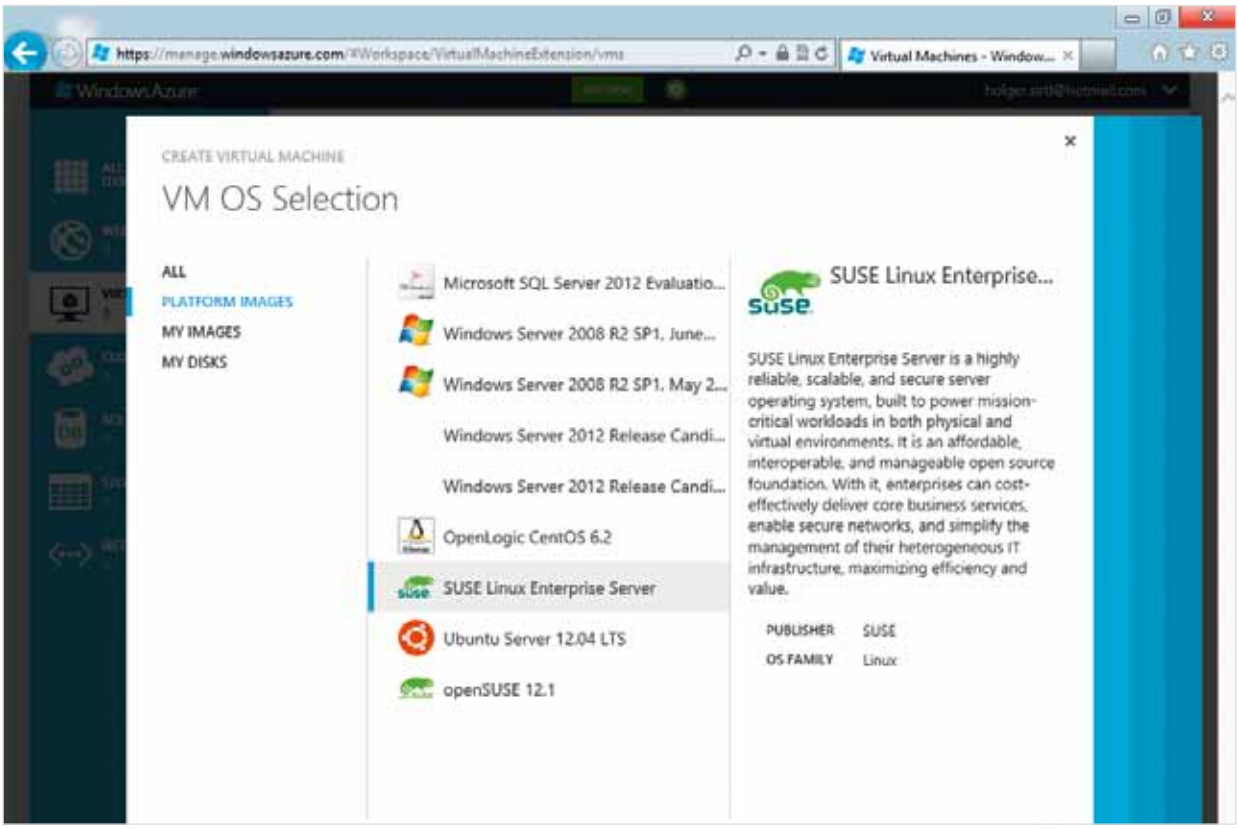


Abbildung 4: Erstellen einer Linux-basierten Virtual Machine über das Azure Portal

Neben Windows werden bei Virtual Machines auch verschiedene Linux-Distributionen unterstützt:

- SUSE Linux Enterprise Server 12 SP2
- openSuse 12.1
- OpenLogic CentOS 6.2
- Ubuntu Server 12.04 LTS

Abbildung 4 zeigt die unterstützten Versionen im Portal. Nachdem in den Folgemasken noch Administrator-Kennung, etwaige Ports und Beziehungen zu anderen VMs festgelegt wurden, wird eine entsprechende VM erzeugt. Dabei wird die Remote-Desktop-Funktion aktiviert, sodass eine direkte Verbindung in die VM aufgebaut und dann alle für die Ausführung einer Java-basierten Anwendung benötigten Software-Komponenten installiert werden können.

Wem der Weg über das Management-Portal nicht zusagt, erhält über verschiedene Powershell-Cmdlets die Möglichkeit, sämtliche Setup-Funktionen für die Erstellung von VMs auch über die Kommandozeile aufzurufen. Listing 3 zeigt ein Powershell-Cmdlet, mit dem eine neue Linux-basierte VM erstellt wird.

Mit dem Upload entsprechender Zertifikate können „Remote Desktop"- oder „Remote Shell"-Verbindungen zu den VMs aufgebaut werden. Die Installation Java-basierter Anwendungen erfolgt dann wie in klassischen, lokal betriebenen Umgebungen. Neben der Auswahl einer vorgefertigten VM lassen sich auch selbst lokal erstellte VM-Images nach Windows Azure laden und dort ausführen. Hierzu muss von der gewünschten VM lokal eine virtuelle Festplatte (VHD) erstellt, über das Portal als Image gespeichert und darauf basierend eine virtuelle Maschine erzeugt und ausgeführt werden. Über diesen Weg können also VMs lokal nach eigenen Wünschen erstellt, getestet und dann in der Cloud betrieben werden. Mit dieser maximalen Flexibilität beim Setup der Umgebung geht allerdings auch die Verpflichtung einher, die Umgebung zur Laufzeit selbst zu betreuen, also Patches und Upgrades selbst einzuspielen. Bei Windows Azure Cloud Services wird diese Aufgabe durch Windows Azure übernommen.

Der iJUG im Dialog

Der iJUG ist mit den bedeutendsten Herstellern im Java-Umfeld in Kontakt, um Informationen aus erster Hand weiterzuleiten und Fragen der Mitglieder direkt zu klären. Derzeit sind folgende Dialoge eingerichtet:

- *Dialog mit Oracle, Ansprechpartner: Fried Saacke und Andreas Badelt, fragen-zu-oracle@ijug.eu*
- *Dialog mit SAP, Ansprechpartner: Hagen Stanek, fragen-zu-sap@ijug.eu*
- *Dialog mit IBM, Ansprechpartner: Rainer Anglett, fragen-zu-ibm@ijug.eu*
- *Dialog mit Red Hat, Ansprechpartner: Tobias Frech, fragen-zu-red-hat @ijug.eu*
- *Dialog mit VMWare, Ansprechpartner: Frank Schwichtenberg und Oliver Szymanski, fragen-zu-vmware@ijug.eu*

Virtual Network

Auf Netzwerk-Ebene werden Virtual Machines durch das neu eingeführte Virtual Network ergänzt. Mit diesem ist es möglich, IPsec-gesicherte Netzwerk-zu-Netzwerk-Verbindungen aufzubauen. Dabei kann durch Übernahme eigener IPv4-Adressen für Windows-Azure-Instanzen ein gemeinsamer Adress-Raum für lokale und Cloud-basierte Ressourcen erstellt werden. In einem solchen Netzwerk lassen sich VM-Instanzen direkt über den Hostnamen ansprechen. Beim Setup wird zwischen einem Windows-Azure-VPN-Gateway eine Verbindung zu einem lokalen VPN-Hardware-Gateway etabliert. Als Gateway werden verschiedene Produkte von Cisco und Juniper unterstützt. Über diese Verbindung wird dann innerhalb des Virtual Networks der Netzwerk-Traffic abgewickelt. Für Ressourcen außerhalb des VPN können zur Kommunikation Endpoints definiert werden. Clients können sich mit einem Endpoint via TCP oder UDP verbinden. Das TCP-Protokoll schließt HTTP- und HTTPS-Kommunikation ein.

Im Zusammenspiel mit Virtual Machines ergeben sich hierdurch umfangreiche Möglichkeiten zur Optimierung der eigenen Infrastruktur. Lokal betriebene VMs können ohne Änderung nach Azure verschoben beziehungsweise von dort auch wieder zurück ins eigene Rechenzentrum migriert werden. Durch Einsatz des Virtual Networks kann dies für die betroffenen Systeme vollständig transparent geschehen. Die Adressierung untereinander bleibt gleich. Darüber hinaus ist es auch möglich, für die Grundlast Systeme lokal zu betreiben und bei Lastspitzen Ressourcen auf Azure hinzuzuschalten und einen gemeinsamen Systemverbund zu schaffen.

Fazit

Für Java-Entwickler ergeben sich mehrere Möglichkeiten, Anwendungen auf Azure zu betreiben. Soll der administrative Aufwand minimiert und Wert auf schnelle Skalierung der Anwendung gelegt werden, eignen sich Cloud Services ideal zum Betrieb der Anwendung. Datenspeicherung sollte hierbei außerhalb der betreffenden VMs erfolgen, um Datenverlust bei einem Re-Image zu vermeiden.

Windows Azure Virtual Machines sind eine Alternative zum Betrieb eigener oder vorgefertigter virtueller Maschinen in der Cloud. Datenspeicherung erfolgt dabei in Laufwerken, die auf Blob-Storage abgebildet werden. Damit wird Datenverlust auch bei einem Re-Image vermieden. Im Zusammenspiel mit dem Virtual Network können hiermit komplexe, hybride Systemkonfigurationen erstellt werden, die sowohl lokal betriebene als auch Azure-basierte VMs umfassen.

Referenzen

[1] https://manage.windowsazure.com
[2] http://www.windowsazure.com/en-us/develop/java/
[3] http://blogs.msdn.com/hsirtl

Holger Sirtl
holger.sirtl@microsoft.com

Holger Sirtl ist seit 2006 als Architecture Evangelist bei Microsoft in München tätig und berät in dieser Rolle Unternehmen im Aufbau Cloud-basierter Anwendungsarchitekturen. Schwerpunktthemen seiner Arbeit sind Cloud Computing und die Windows Azure Plattform. Vor seinem Einstieg bei Microsoft arbeitete Holger Sirtl sechs Jahre lang als Technologieberater für eine international führende Unternehmensberatung sowie zwei Jahre lang als Senior-IT-Projektmanager für einen großen deutschen Energieversorger.

Eclipse Code Recommenders: Liest du noch [Quellcode] oder programmierst du schon?

Marcel Bruch und Andreas Sewe, TU Darmstadt

Mit dem Eclipse-Juno-Release hat auch ein neues Eclipse-Projekt seine Version 1.0 erreicht. Eclipse Code Recommenders ist eine Erweiterung der Java-Development-Tools, die die Konzepte des Web 2.0 auf die IDE überträgt. Mit der Integration von Code Recommenders in das Eclipse-RCP-Package und die Eclipse IDE for Java Developers ist es jetzt an der Zeit, das Projekt ein wenig näher vorzustellen und den einen oder anderen Blick hinter die Kulissen zu werfen.

Ein Code Recommender ist ein Werkzeug, das aufzeigt, wie man ein API typischerweise verwendet, also welche Methoden man auf einem Objekt üblicherweise aufruft, welche Methoden der Basisklasse man für gewöhnlich überschreibt und Ähnliches. Wenn jetzt mehrere Code Recommender für unterschiedlichste Aufgaben zusammenkommen, hat man – auf Englisch – „Code Recommenders". Es besteht also aus einer Menge verschiedener Werkzeuge, die Programmierer während der Software-Entwicklung im Umgang mit (fremden) APIs unterstützen und so dabei helfen, Fehler im Umgang mit diesen Frameworks zu vermeiden sowie Einarbeitungskosten zu sparen.

Das Interessante an Code Recommenders ist, dass das Wissen, wie ein API korrekt verwendet wird, nicht mühsam manuell spezifiziert werden muss, sondern vollautomatisch aus dem Code bestehender Anwendungen extrahiert und direkt innerhalb der Entwicklungsumgebung in Form von intelligenter Code-Completion, erweiterter API-Dokumentation, Bug-Detection-Systemen und personalisierten Code-Suchmaschinen wieder zur Verfügung gestellt wird.

```
public class MyDialog extends Dialog {

   private Text text;

   public MyDialog(Shell s) {
       super(s);
       text = new Text(s, SWT.BORDER);
   }

   // debug code:
   public static void main(String[] args) {
       Shell s = new Shell();
       Dialog d = new MyDialog(s);
       d.open();
   }
```

Listing 1: Erster Versuch mit UI-Code im Konstruktor

Frameworks = Kostenersparnis?

Arbeitet man regelmäßig mit neuen APIs? Oder ist das Projekt so groß, dass man sich nur schwer für jede ihrer Klassen merken kann, wie man diese korrekt benutzt? Das ist normal. Sehr normal. Tatsächlich zeigt eine Studie der Carnegie Mellon University, dass Software-Entwickler gerade mal ein Viertel ihrer Zeit tatsächlich mit dem Schreiben und Testen von Code verbringen. Die restliche Zeit wird unter anderem mit dem Lesen von Dokumentation, dem Verstehen von Quellcode und vor allem mit der Suche nach Beispielcode und Tutorials im Web verbracht (Quelle: IEEE Transactions of Software Engineering, Vol. 32, No. 12).

Ob In-House-Bibliothek oder Open-Source-Projekt – jedes Framework, das in Anwendungen zum Einsatz kommt, muss zuerst einmal erlernt werden, bevor es effizient einsetzbar ist. Für jedes API ist zu verstehen, wann welche Methoden in welcher Reihenfolge aufgerufen werden müssen. Um diesen Lernprozess möglichst klein zu halten, stellen Framework-Entwickler jede Menge Dokumentationen (API-Dokumentation, Tutorials, Code Snippets) zur Verfügung. Dennoch bleibt oft unklar, wie genau man seine Anforderungen mit dem Framework lösen kann. Doch wenn die existierende Dokumentation keine Hilfestellung mehr gibt, wird der Quellcode anderer Programme, die das Framework bereits auf ähnliche Weise benutzt haben, zu einer neuen und interessanten Informationsquelle …

Dieser Artikel zeigt anhand eines kleinen Beispiels, wie IDEs ihre Nutzer beim Erlernen neuer Frameworks besser unterstützen und vor Fehlern bewahren können, indem sie aus dem Quellcode anderer Anwendungen die korrekte Nutzung des Frameworks erlernen und dieses Wissen dem Entwickler wieder zur Verfügung stellen.

Beispielszenario

Ein einfaches Beispiel zeigt vermutlich am besten, wie Code Recommenders funktioniert und welche Probleme es löst. Nehmen wir also an, wir möchten einen einfachen Eingabe-Dialog mithilfe des Standard Widget Toolkits (SWT) und JFace implementieren. Als Basisklasse für den eigenen Dialog wählen wir die Klasse „org.eclipse.jface.dialogs.Dialog". Die IDE generiert daraufhin einige Zeilen Quellcode. Dieses Gerüst unserer Dialog-Klasse kompiliert zwar, doch wenn wir den Code ausführen, passiert – wenig überraschend – nicht viel (siehe Abbildung 1).

Als Nächstes fügen wir im Konstruktor unseren eigenen UI-Code hinzu in der Hoffnung, dass dies dafür der richtige Ort sei (siehe Listing 1). Wenn wir den Code jedoch erneut ausführen, passiert immer noch nicht mehr. Unser neu angelegtes Text Widget erscheint nicht mal.

Konzeptionell finden wir auf die Schnelle keinen Fehler in unserem Code und wenden uns deshalb hilfesuchend an die

Abbildung 1: Leerer Basisdialog

Klassen-Dokumentation von Dialog. Leider finden wir hier nur eine abstrakte Beschreibung darüber, was überhaupt ein Dialog sei, jedoch keinerlei Informationen darüber, wie wir die konkrete Dialog-Klasse erweitern müssen, um unsere eigenen Inhalte darzustellen.

Es bleibt uns also nichts anderes übrig, als die API-Dokumentation aller 56 potenziell überschreibbaren Methoden manuell zu untersuchen und per „Trial & Error" herauszufinden, wo wir unseren UI-Code eigentlich hätten platzieren müssen. Wem das zu lange dauert, der befragt die Suchmaschine seiner Wahl nach geeigneten Tutorials oder öffnet die Typ-Hierarchie seiner IDE und analysiert manuell alle verfügbaren Subtypen von Dialog, um aus diesen funktionierenden Code-Beispielen zu lernen, wie andere Entwickler die Klasse „Dialog" typischerweise erweitert haben.

Würden wir dieses manuelle Suchen mit etwas mehr System durchführen, kämen wir rasch zu dem Ergebnis, dass fast alle Sub-Klassen die Methoden „Dialog.createDialogArea()" und viele Klassen die Methode „Dialog.okPressed()" überschreiben. Diese Methoden scheinen demnach im besonderen Maße für Sub-Klassen interessant zu sein. Zu schade, dass wir bis dahin bereits eine halbe Stunde unserer Zeit mit Ausprobieren, dem Lesen der Dokumentation und der Suche und Analyse von Code-Snippets verbracht haben …

Genau hier setzt Code Recommenders zum ersten Mal an. Anstatt es dem Benutzer zu überlassen, Beispielcode im Web zu finden und manuell zu untersuchen, analysiert Code Recommenders große Software-Repositories vollautomatisch und extrahiert so die Informationen darüber, welche Methoden der Klasse „Dialog" üblicherweise von Sub-Klassen überschrieben werden. Die so gewonnenen Informationen werden dann in Form von einfachen Bayes'schen Netzen abgelegt (Hinweis: Bayes'sche Netze halten beispielsweise auch das Postfach Spam-frei) und mittels einer speziellen Code-Completion wieder in die IDE integriert (siehe Abbildung 2).

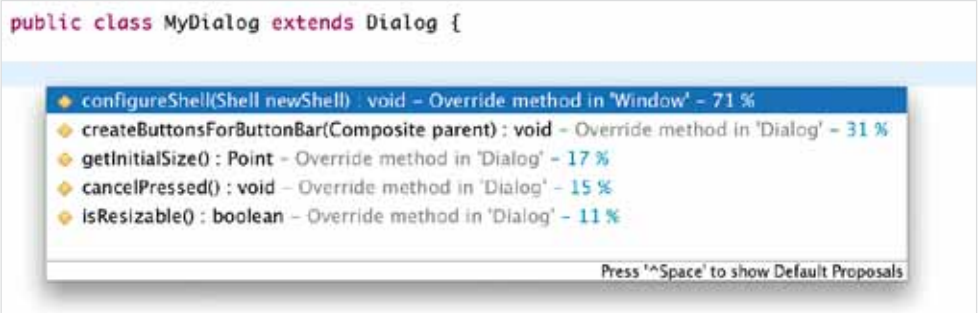


Abbildung 2: Code Recommenders' intelligente Overrides-Completion. Die Prozentwerte hinter den jeweiligen Vorschlägen geben die Wahrscheinlichkeit dafür an, dass die vorgeschlagenen Proposals auch tatsächlich ausgewählt werden.

Die intelligente Overrides-Completion-Engine von Code Recommenders hilft Software-Entwicklern also, schneller die relevanten Methoden einer Basisklasse zu identifizieren, indem sie die unnötigen Methoden ausfiltert oder (optional) einfach über den bestehenden Vorschlägen anzeigt. Das Wort „intelligent" scheint hier vielleicht noch ein wenig großzügig bemessen, doch sei schon mal erwähnt, dass sich die Vorschläge mit jeder neuen Information (etwa einer weiteren überschriebenen Methode) immer wieder neu anpassen und die Code-Completion-Engine somit erkennt, worauf ein Entwickler zusteuert. Dazu später mehr. Neben der Darstellung der Vorhersagen innerhalb der Eclipse-Code-Completion stellt Code Recommenders diese Informationen auch in einer separaten Javadoc View (siehe Abbildung 3) dar.

Nach ein wenig Literatur-Recherche haben wir nun die beiden relevanten Methoden „createDialogArea()" und „okPressed()" identifiziert und überschrieben. Jetzt geht es an die eigentliche Implementierung. Dass für die Benutzereingabe ein Objekt der Klasse „org.eclipse.swt.widgets.Text" benötigt wird, leuchtet recht schnell ein

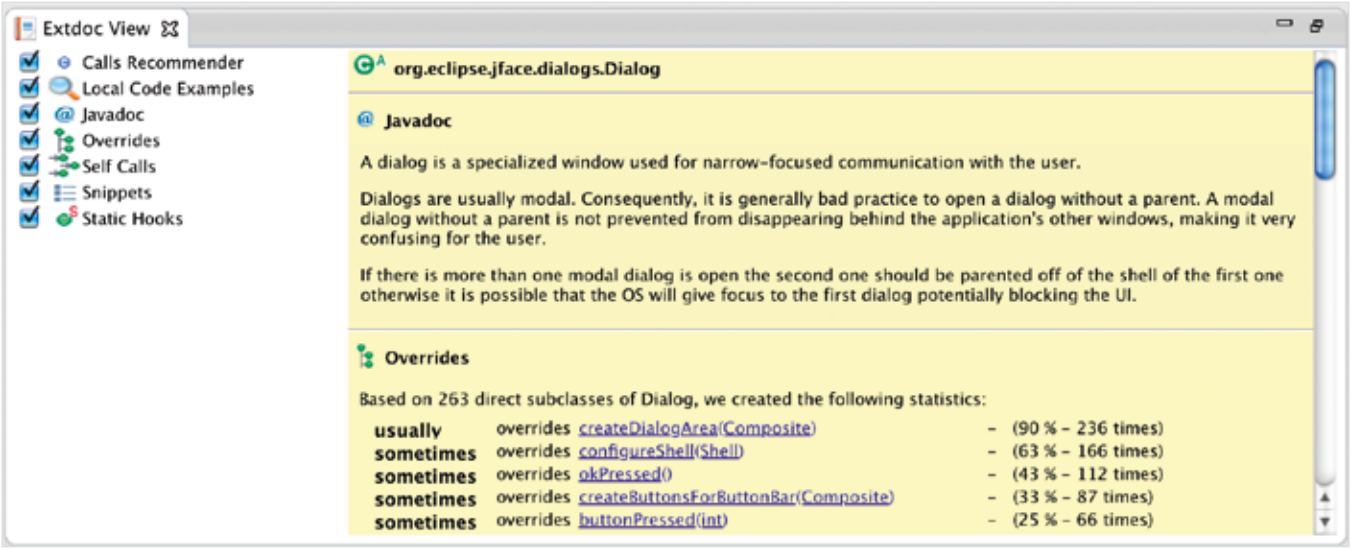


Abbildung 3: Extended Javadoc View

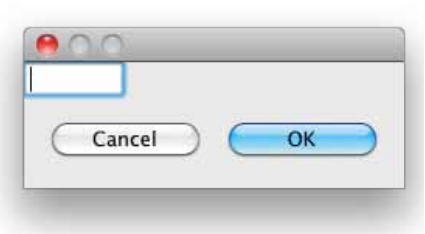


Abbildung 4: Dialog mit Textfeld

und die Wahl des Konstruktoraufrufs geht auch noch flüssig von der Hand. Doch wie geht es jetzt weiter? Ein kurzer Blick auf das UI (siehe Listing 2 und Abbildung 4) zeigt uns, dass alles grundsätzlich in die richtige Richtung läuft – wenn auch nicht so hübsch, wie wir es gerne hätten.

```
public class MyDialog extends Dialog {

   private Text text;

   @Override
   protected Control createDialogArea(Composite p) {

       text = new Text(p, SWT.BORDER);

       return p;
   }
```

Listing 2: Unvollständige Verwendung von Text

```
public class MyDialog extends Dialog {

   private Text text;

   @Override
   protected Control createDialogArea(Composite p) {
       text = new Text(p, SWT.BORDER);
       text.
```

setLayoutData(Object layoutData) : void – Control – 80 %
setText(String string) : void – Text – 53 %
addModifyListener(ModifyListener listener) : void – Text – 38 %
Press '^Space' to show Default Proposals

Abbildung 5: Intelligent-Call-Completion für „text" in createDialogArea()"

```
public class MyDialog extends Dialog {

   private Text text;

   @Override
   protected Control createDialogArea(Composite p) {
       text = new Text(p, SWT.BORDER);
       text.
```

dynamic 'Text' – VL-6 – 24 %
dynamic 'Text' – VL-13 – 14 %
dynamic 'Text' – VL-7 – 8 %
dynamic 'Text' – VL-43 – 6 %
dynamic 'Text' – VL-51 – 5 %
dynamic 'Text' – VL-53 – 3 %
Press '^Space' to show Chain Proposals (Recommender)

```
text.addModifyListener(listener);
text.setLayoutData(layoutData);
text.setText(string);
```

Abbildung 6: Template-Completion für „text" in „createDialogArea()"

Fein-Tuning ist also angesagt. Aber wie? Ein Blick auf das API von Text lässt uns staunen: Es besteht aus sage und schreibe 164 öffentlichen Methoden. Aber welche sind für uns in „createDialogArea()" tatsächlich relevant?

Hier kommt die zweite Code-Completion-Engine zum Zug: die intelligente Call-Completion. Ähnlich wie im Beispiel für „Dialog" können Muster der korrekten Verwendung einer Framework-Klasse direkt aus dem Quellcode bestehender Beispiel-Anwendungen extrahiert und die so gewonnenen Informationen wieder in die Code-Completion integriert werden. So würde Code Recommenders in diesem Szenario nicht alle 164 potenziell möglichen Methoden vorschlagen, sondern nur noch die drei wahrscheinlichsten – nämlich „setLayoutData()", „setText()" und „addModifyListener()" (siehe Abbildung 5).

Neben der Vorhersage einzelner Methoden-Aufrufe kann Code Recommenders auch vollautomatisch ganze Code-Snippets aus dem Quellcode bestehender Anwendungen generieren und diese in der Code-Completion präsentieren. Dies beschleunigt den Entwicklungsprozess noch weiter (siehe Abbildung 6).

Werfen wir noch einen letzten Blick auf die Implementierung von „MyDialog.okPressed()". Hier müssen wir den Inhalt des Text-Widgets auslesen und für die Weiterverarbeitung zwischenspeichern. Triggern wir Standard-Code-Completion auf der Variablen „text", erhalten wir erwartungsgemäß wieder alle 164 potenziell möglichen Methoden – völlig unabhängig davon, dass in diesem Kontext lediglich eine Methode wirklich sinnvoll ist, „getText()" (siehe Abbildung 7).

Das letzte Beispiel zeigt eine wichtige Eigenschaft der intelligenten Call-Completion, nämlich deren Kontext-Abhängigkeit. Auch für die intelligente Call-Completion verrichten Bayes'sche Netze ihren Dienst, die für jede Variable verschiedene Informationen aus dem Quellcode auswerten:

- Wie wurde eine Variable definiert?
- Wo wurde sie definiert?
- Welche Methoden wurden bereits auf dieser aufgerufen?

Aus diesen Informationen errechnen sich für jede Variable völlig unterschiedliche

Wahrscheinlichkeiten, die sich durch jede weitere Information ständig weiter verfeinern. Mit jedem weiteren Methoden-Aufruf lernt Code Recommenders somit mehr über das Ziel und wohin der Entwickler steuert.

Daten

Wie kommt Code Recommenders zu seinen Vorhersagemodellen? Im ersten Schritt muss Code Recommenders Beispiel-Anwendungen für die Analyse sammeln. Die Daten hierfür können aus Software-Repositories wie „Maven Central" oder dem „Eclipse Marketplace" stammen, aber auch direkt von Sites wie „GitHub" oder „Bitbucket" beziehungsweise aus dem firmeneigenen Versions-Kontrollsystem.

In einem zweiten Schritt werden diese Daten analysiert. Dabei werden wertvolle API-Verwendungsmuster extrahiert, in einer Modell-Datenbank zusammengefasst und schließlich zum Download bereitgestellt. Sie können nun on-Demand von der IDE des Entwicklers heruntergeladen werden, um dann als Basis für Vorhersagen zu dienen, etwa innerhalb der Eclipse-Code-Completion oder der erweiterten Java-Dokumentation (siehe Abbildung 8).

Fazit

Dieser Artikel hat ein kurzes Schlaglicht auf drei der fünf neuen Code-Recommenders-Completion-Engines für Java geworfen (Overrides-, Call- und Templates-Completion). Für Eclipse Kepler (Eclipse 4.3, Juni 2013) sind jedoch schon wieder eine Reihe neuer Code Recommender in Arbeit. Ein Teilteam arbeitet derzeit an weiteren intelligenten Code-Completion-Engines – unter anderem an einer intelligenten Parameter-Completion-Engine. Denn welcher Entwickler mit dem Standard Widget Toolkit (SWT) kennt nicht die durch SWT-Style-Konstanten verursachten Probleme, wenn man zum Beispiel einen Listener hinzufügen oder ein Text-Widget erzeugen will? Aber welche Konstante oder sogar welches Objekt man als Parameter einem Methoden-Aufruf übergeben will, lässt sich genauso erlernen wie das Wissen darüber, welche Methoden auf einem Objekt aufgerufen werden sollen.

Ein weiterer Bereich mit aufregenden Fortschritten betrifft das Mining von komplexen Code-Snippets aus Beispielcode.

```
public class MyDialog extends Dialog {

    private Text text;

    @Override
    protected void okPressed() {
        text.
```

getText() : String - Text - 97 %

Press '^Space' to show Default Proposals

Abbildung 7: Intelligente Call-Completion für „text" in „okPressed()"

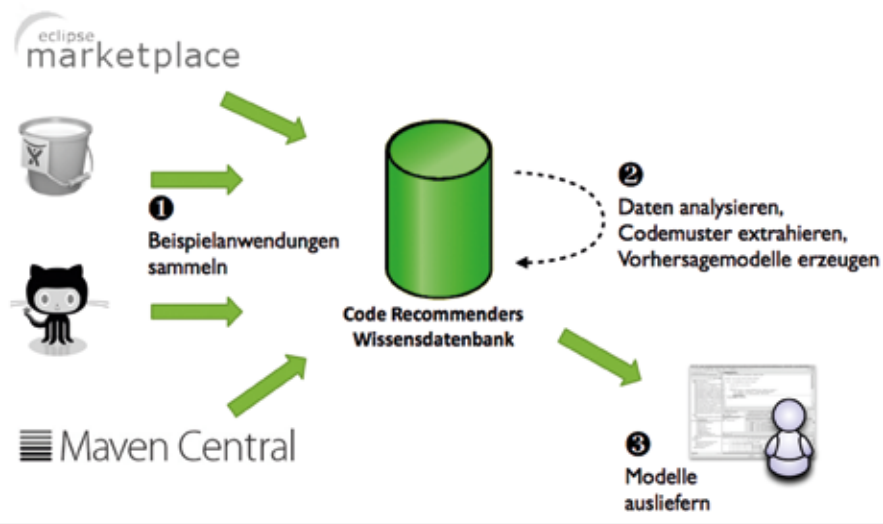


Abbildung 8: Modell-Generierung und -Verteilung in Code Recommenders

Jeder Entwickler kennt die Code-Templates, die seine IDE zum Beispiel für das Iterieren über ein Array oder das Durchführen von Typcasts anbietet. Aber viel häufiger als diese einfachen Templates bräuchte man fallspezifische Code-Templates, die zeigen, wie man ein Framework-Objekt korrekt benutzt und wie es mit anderen Objekten zusammenarbeitet. Die Eclipse-SWT-Templates sind ein gutes Beispiel für solche fallspezifischen Templates. Doch diese zu erzeugen ist sehr zeitaufwändig; daher existieren auch nur wenige. Das Code-Recommenders-Team arbeitet derzeit an Werkzeugen, die häufige Codemuster aus Beispielcode extrahieren und diese als Templates wieder in der IDE verfügbar machen, vollautomatisch. Andere Bereiche umfassen die Arbeit an Suchmaschinen für Stacktraces beziehungsweise Beispielcode für Eclipse und weitere Ideen. Mehr dazu, wenn diese Werkzeuge reif sind …

Marcel Bruch
bruch@cs.tu-darmstadt.de
Andreas Sewe
sewe@cs.tu-darmstadt.de

Marcel Bruch ist Projektleiter von Eclipse Code Recommenders, Eclipse-Trainer und Wissenschaftler an der TU Darmstadt. Sein Ziel ist es, mithilfe der Big (Software Development) Data neue Wege in der Software-Entwicklung zu beschreiten und IDEs intelligenter zu machen.

Andreas Sewe ist Wissenschaftler an der TU Darmstadt, wo er zu Programmiersprachen und virtuellen Maschinen forscht. Sein Ziel ist es, eine Vielzahl von Programmiersprachen von intelligenten IDEs profitieren zu lassen.

Webservice-Lasttests mit loadUI

Sebastian Steiner, Trivadis AG

loadUI ist ein Test-Werkzeug zum Erstellen, Durchführen und Auswerten von Lasttests. Dieser Artikel zeigt auf, welchen Mehrwert und welche Möglichkeiten loadUI bietet und wie man damit arbeitet.

Lasttests gehören zu den sogenannten „nicht-funktionalen Tests". Primärziel ist, deren Verhalten unter Last zu beobachten, und nicht, funktionale Fehler in der zu testenden Software zu finden. Grundsätzlich gilt: Damit ein Lasttest sinnvoll durchgeführt werden kann, sollte sich das System bereits in einem funktional stabilen Zustand befinden.

Lasttests lassen sich nicht ohne Tool-Unterstützung durchführen. Das größte Problem stellt die Generierung einer hohen Last dar. loadUI ist ein noch junges, aber bereits mächtiges Tool zur automatisierten Durchführung von Lasttests.

loadUI

Die erste Version von loadUI wurde im September 2010 veröffentlicht. Sie ist darauf ausgelegt, möglichst einfach und intuitiv bedienbar zu sein. Tests lassen sich per „Drag & Drop" von bestehenden Komponenten zusammenklicken. Programmier-Kenntnisse sind nicht nötig.

Lasttests lassen sich direkt in der loadUI-GUI starten und zur Laufzeit modifizieren. Die Auswertung kann über integrierte Statistik-Komponenten im Programm selbst erfolgen. Verteiltes Testen wird ebenso unterstützt wie die Automatisierung via Konsolen-Skripts.

Die schwedische Firma SmartBear Software hat loadUI entwickelt. Das ist dieselbe Firma, von der auch soapUI stammt. Die Möglichkeit, bestehende soapUI-Tests in loadUI als Lasttests laufen zu lassen, ist das Haupt-Argument, das für den Einsatz von loadUI spricht.

soapUI

Kernfunktionalität von soapUI, einem populären Test-Framework für serviceorientierte Lösungen, ist das Testen von SOAP-Webservices. Daneben lassen sich jedoch auch Datenbanken, REST-Services oder JMS-Queues mit soapUI testen. Die umfangreichen Einsatzmöglichkeiten und die Tatsache, dass viele Firmen bereits soapUI einsetzen, machen die nahtlose Integration in loadUI so wertvoll. Weitergehende Informationen und Tutorials zu soapUI stehen unter www.soapui.org im Internet.

Erste Schritte mit loadUI

Der loadUI-Arbeitsbereich besteht aus drei Hierarchie-Levels. Einstiegspunkt ist der sogenannte „Workspace". Hier werden Projekte, Agenten für verteilte Tests und weitere globale Einstellungen verwaltet. loadUI speichert standardmäßig alle anfallenden Dateien im Ordner „.loadUI", der im aktuellen User-Home liegt. Ein Projekt entspricht immer einer XML-Datei, was das Arbeiten mit Versions-Kontrollsystemen erschwert. loadUI selbst unterstützt keine Versionierung. Das Einpflegen von Änderungen muss also auf XML-Ebene erfolgen. Die Nachvollziehbarkeit wird dadurch erschwert und das manuelle Zusammenführen zweier Versionen im Konfliktfall ist mühsam.

Klickt man im Workspace auf ein Projekt, öffnet sich die Project View. Dort sind die Testfälle ersichtlich, die im Projekt enthalten sind, und diese lassen sich einzeln oder als Gesamtes starten (siehe Abbildung 1). Das Anklicken eines Testfalls öffnet die „Testcase View". Das ist der Bereich, in dem die eigentliche Testlogik implementiert wird.

Man findet sich schnell im loadUI-Arbeitsbereich zurecht. Fast alle Objekte lassen sich anklicken und per „Drag & Drop" bewegen. Oftmals ist es angenehm so zu arbeiten, manchmal erscheint es auch unnötig kompliziert. Um Zahlenwerte zu definieren, wäre beispielsweise ein simples Eingabefeld viel praktischer, als am „Rädchen" einer Komponente zu drehen.

Komponenten

Die Komponenten bilden das Herzstück von loadUI. Aus ihnen werden die Lasttests erstellt. Alle Komponenten besitzen Ein- und Ausgänge, um sie mit anderen Komponenten zu verknüpfen. Die Komponenten werden in Gruppen zusammengefasst. Nachfolgend eine Übersicht:

- *Generators*
  Sie steuern die Art und Menge der Last. Generatoren werden mit einem Runner verbunden und senden diesem in definierten Intervallen ein Signal, damit er eine Aktion ausführt. Mit den unterschiedlichen Generatoren lassen sich verschiedene Arten von Last generieren: fixe Last, variable Last, Spitzen oder auch eine zufällige Last. Ein Testfall kann 1 bis n Generatoren enthalten. Alle lassen sich separat ein- und ausschalten.
- *Scheduler*
  Ein Scheduler ist eine Art Zeitschaltuhr, um Generatoren ein- und auszuschalten. Verbundene Generatoren werden zu den definierten Zeiten aktiviert beziehungsweise deaktiviert. Somit lässt sich die Last während eines Testlaufs besser variieren.
- *Runners*
  Das sind die Komponenten, die die Testlogik ausführen. Ein Runner kann ein Groovy-Skript, einen Betriebssystem-Befehl oder einen soapUI-Testfall ausführen. Runner können auch http-Requests absenden, um eine Webseite aufzurufen. Jeder Runner verfügt über einen Eingang und mindestens drei Ausgänge. Über den Eingang wird er mit einem Generator verbunden. Die Ausgänge lassen sich mit den Analysis-Komponenten verbinden und liefern Informationen und Statistiken zum aktuellen Testlauf.

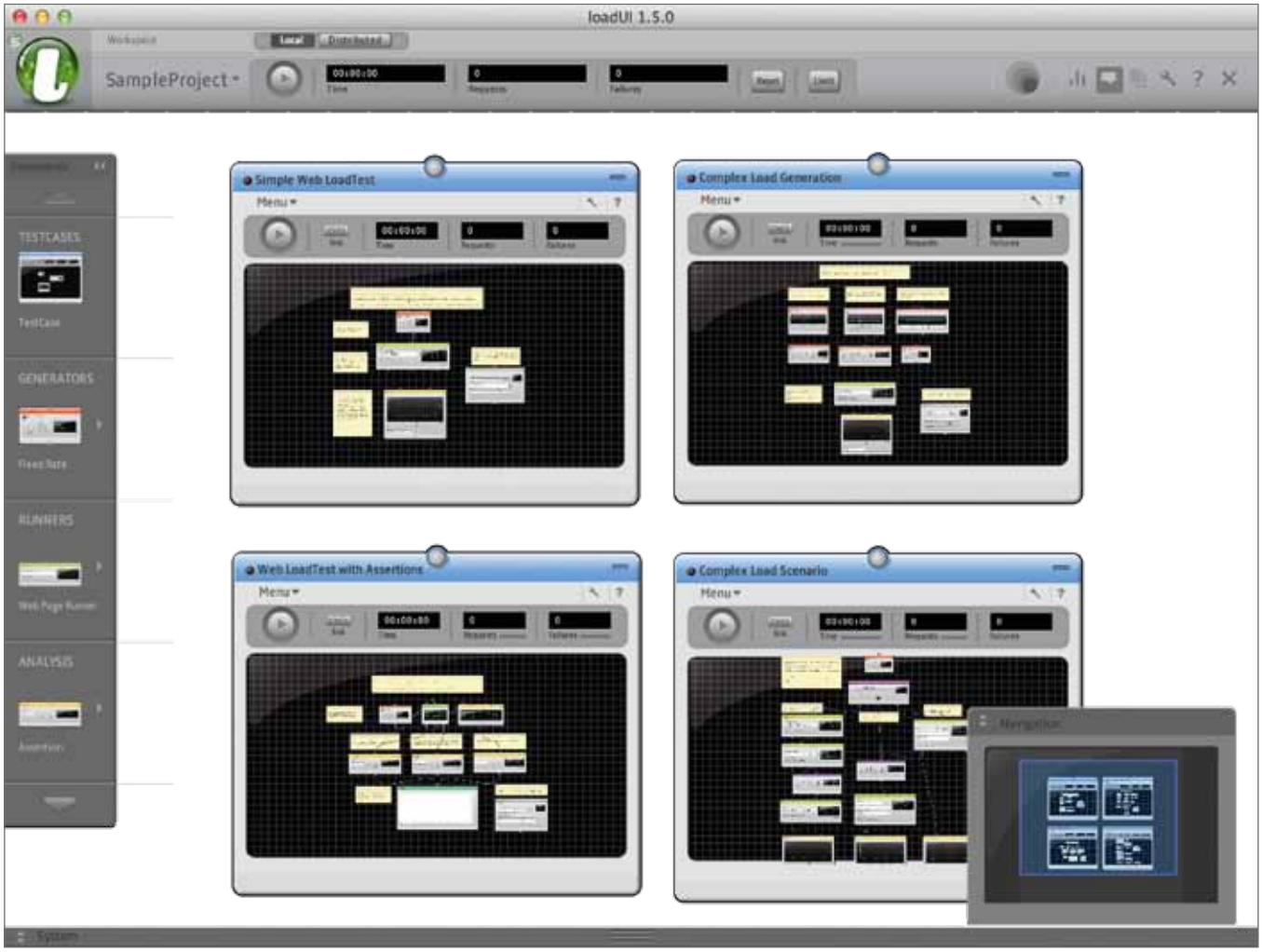


Abbildung 1: Die loadUI-Projektansicht im Überblick

- *Analysis*
  Diese Komponenten analysieren die Resultate, die ein Runner liefert. Einerseits können unterschiedliche Statistiken dargestellt werden, anderseits können die Resultate auf gewisse Eigenschaften wie Laufzeiten oder Fehleranzahl geprüft werden. Mit loadUI Version 1.5 wurde eine „Statistics Workbench" eingeführt. Sie dient als mächtigere Alternative zu den Analysis-Komponenten. Mehr dazu im Nachfolgenden.
- *Flow Control*
  Davon gibt es lediglich zwei Komponenten. Splitter verteilen eingehende Nachrichten zufällig oder gleichmäßig auf die verbundenen Runner. Die Delay-Komponente pausiert eingehende Nachrichten für einen definierten Zeitraum.
- *Output*
  Dies entspricht in etwa einem Log-Fenster. Die Table-Log-Komponente – die einzige Komponente dieser Kategorie – erlaubt das tabellarische Anzeigen von Resultaten und Nachrichten.
- *Custom Components*
  Verfügt man über Programmierkenntnisse, lässt sich loadUI einfach mit eigenen Komponenten erweitern. Dies wird hier jedoch nicht behandelt und kann im „Developers Corner" auf www.loadui.org nachgelesen werden.

Abbildung 2 zeigt ein Beispiel, wie ein einfacher Testfall, bestehend aus einigen der oben genannten Komponenten, aufgebaut sein kann. Der Testfall enthält einen „Fixed Rate Generator", der zehnmal je Sekunde einen „Web Page Runner" anstößt. Dieser ruft die definierte Web-Adresse auf und sendet verschiedene Kennzahlen an die verbundene Statistik-Komponente. Diese stellt die Kennzahlen in einem einfachen Diagramm dar.

soapUI-Integration

Die Integration von soapUI ist das Alleinstellungsmerkmal, das loadUI von anderen Konkurrenzprodukten unterscheidet. Der Aufwand, um bestehende soapUI-Artefakte in loadUI zu verwenden, ist minimal. Unterstützt wird sowohl die Open-Source-Variante von soapUI als auch die kostenpflichtige Pro-Version.

Die MockService-Komponente erlaubt es, einen soapUI-MockService in loadUI laufen zu lassen. Man wählt das entsprechende soapUI-Projekt an und in einer Dropdown-Liste werden alle verfügbaren MockServices angezeigt. Der ausgewählte MockService wird beim Starten des Testfalls ebenfalls gestartet. Damit lassen sich beispielsweise Prozesse simulieren, die von den zu testenden Komponenten aufgerufen werden, aber nicht Bestandteil des Tests sind. Ein weiterer Anwendungsfall ist das Simulieren von Systemkomponenten,

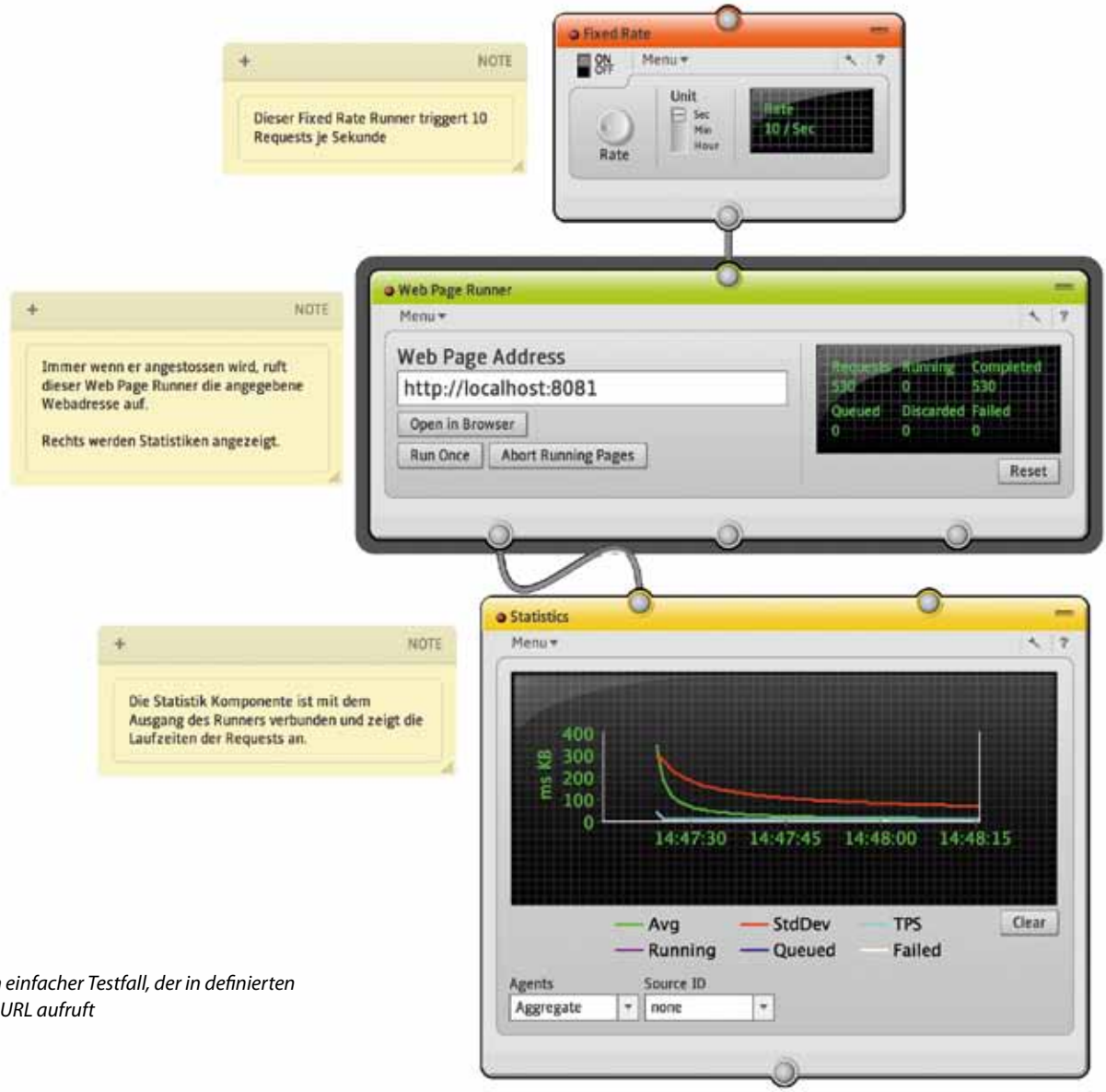


Abbildung 2: Ein einfacher Testfall, der in definierten Abständen eine URL aufruft

die sich in der Entwicklung befinden und noch nicht regulär aufgerufen werden können.

Wesentlich interessanter ist aber die soapUI-Runner-Komponente (siehe Abbildung 3). Auch hier ist die Konfiguration einfach: Der gewünschte soapUI-Testfall wird in einer Dropdown-Liste angewählt und der Runner mit einem Generator verbunden – schon ist der soapUI-Testfall in loadUI einsetzbar.

Die Ausführung des Testfalls läuft genau gleich ab, als wenn er in soapUI gestartet worden wäre. Eventuell vorhandene Setup- und Teardown-Skripte oder Eventhandler werden ganz normal aufgerufen. Müssen einem soapUI-Testfall für die Ausführung Parameter übergeben werden, können diese in den Einstellungen des Runners – im Reiter „Properties" – definiert werden.

Verteiltes Testen

Wird die Ausführung eines Tests auf mehrere, voneinander unabhängige Computer verteilt, spricht man von „verteiltem Testen". Es gibt unterschiedliche Gründe, um verteilte Tests durchzuführen: Einerseits entspricht es mehr der Realität, wenn die Aufrufe von unterschiedlichen Quellen kommen. Anderseits stößt ein handelsüblicher Laptop oder PC beim Ausführen eines Lasttests auch schnell an die Grenzen seiner Leistungsfähigkeit.

loadUI bietet die Möglichkeit, einen Testfall auf einen oder mehrere sogenannte „Agents" zu verteilen. Der loadUI-Agent wird mit einer separaten Installationsdatei auf dem Computer installiert. Ein Computer, der als Agent eingesetzt wird, benötigt keine Komplett-Installation von loadUI.

Die Kommunikation mit den Agenten geschieht via HTTPS-Protokoll. Agenten, die sich in demselben Subnetz befinden wie der Haupt-Computer, werden von loadUI automatisch erkannt. Ansonsten kann man sie sich über den „Agent Manager" hinzufügen. Dieser lässt sich am unteren Rand der Workspace- und Projekt-Ansicht einblenden. Dort werden auch die Testfälle per „Drag & Drop" auf die konfigurierten Agenten verteilt.

loadUI bietet zwei unterschiedliche Modi zur Testausführung an. Im lokalen Modus kommen alle Testfälle auf dem Computer, auf dem loadUI läuft, zur Ausführung. Im verteilten Modus werden die Testfälle auf die Agenten verteilt – so wie es im Agenten-Manager konfiguriert wurde – und dort gestartet. Informationen zum Testlauf kommen an den Haupt-Computer zurück. Wichtig ist hierbei, dass jeder Agent seine eigene Kopie des Testfalls ausführt. Enthält also ein Testfall, der auf vier

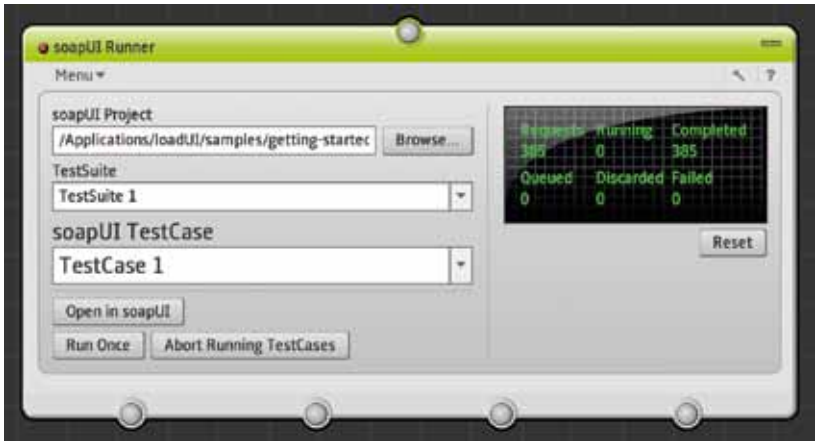


Abbildung 3: Die soapUI-Runner-Komponente

Agenten verteilt wurde, einen Generator, der 100 Requests pro Sekunde erzeugt, wird eine Gesamtlast von 400 Request pro Sekunde generiert. Auf der loadUI-Homepage befindet sich ein Tutorial darüber, wie sich ein Agent in der Amazon Elastic Compute Cloud (Amazon EC2) installieren und verwenden lässt.

Test-Automatisierung

Ein wichtiger Faktor beim Testen besteht darin, die Tests regelmäßig auszuführen. Dies gilt auch für Lasttests. loadUI enthält Konsolen-Skripte, mit denen Tests via Kommandozeile ausgeführt werden können. Wird dieses in ein Ant-Skript gepackt, lassen sich loadUI-Tests ohne großen Aufwand in eine Continuous-Integration-Umgebung einbinden.

Verschiedene Kommandozeilen-Parameter können die Testausführung steuern. Es lassen sich Zeit- oder Mengen-Limits dafür definieren, wann die Ausführung gestoppt werden soll. Der Testlauf kann auf spezifische Testfälle begrenzt werden und auch verteiltes Testen mit Agenten ist möglich. Eine vollständige Referenz aller möglichen Parameter ist auf der loadUI-Homepage zu finden.

Testresultate auswerten

Die momentan aktuelle Version 1.5 von loadUI besitzt den Code-Namen „The Statistics Release". Offensichtlichste Neuerung ist die sogenannte „Statistics Workbench", um Testresultate zu archivieren, zu vergleichen und auszuwerten.

Dieser Arbeitsbereich besteht aus zwei Ansichten. In der „Result View" werden automatisch die Resultate der letzten Testläufe gespeichert. „Current Run" repräsentiert hierbei den aktuellen Testlauf. Es ist möglich, bereits zur Laufzeit eines Tests Statistiken zu diesem zu erstellen. Ein Doppelklick auf eines der gespeicherten Resultate öffnet die „Analysis View". Dort lassen sich zu den gespeicherten Resultaten Diagramme mit unterschiedlichen Kennzahlen erstellen. Das Aussehen der Diagramme lässt sich beliebig konfigurieren und sie können per Knopfdruck als PDF exportiert werden. Es gibt ein Set an vordefinierten Diagrammen, die sich per „Drag & Drop" hinzufügen lassen. Abbildung 4 zeigt ein Diagramm, das die Anzahl der Aufrufe je Sekunde für vier verschiedene Runner anzeigt.

Ein nettes Feature ist, zwei Testläufe miteinander zu vergleichen. Deren Resultate werden dann überlagert in den Diagrammen dargestellt.

Abbildung 4: Auswertung eines Testlaufs mithilfe der „Statistics Workbench"

Fazit

Wer mit soapUI arbeitet, sollte sich loadUI auf jeden Fall einmal ansehen. Die Tutorials auf der offiziellen Homepage sind dafür ein guter Einstiegspunkt. Der loadUI-Arbeitsbereich kommt aufgeräumt daher, die Arbeit darin ist intuitiv. Mit den vorhandenen Komponenten lässt sich bereits sehr viel machen. Der Funktionsumfang lässt sich mit „Custom Components" nochmals erweitern. Auf der loadUI-Website steht unter www.loadui.org/Custom-Components eine Auswahl von fertigen Komponenten zum Download bereit.

Negativ fällt auf, dass loadUI tendenziell einen hohen Speicherbedarf hat und nach längerem Gebrauch zum Teil etwas träge reagiert. Dann hilft es, das Programm zu beenden und neu zu starten.

Eine Unterstützung für Composite Projects – so wie es in soapUI Pro möglich ist – wäre ebenfalls wünschenswert. Mittlerweile ist auch eine kostenpflichtige Pro-Version von loadUI angekündigt. Man darf gespannt sein, welche Verbesserungen diese Version mit sich bringt.

Literatur und Links

- www.loadui.org
- www.soapui.org

Sebastian Steiner
sebastian.steiner@trivadis.com

Sebastian Steiner ist seit 2002 in der Informatik tätig und seit 2004 im Bereich der Software-Entwicklung mit Schwerpunkt Java. Von 2007 bis 2010 absolvierte er ein Informatikstudium an der Berner Fachhochschule. Seit Herbst 2010 arbeitet Sebastian Steiner für die Trivadis AG als Consultant im Java- und SOA-Umfeld.

Tools für Web-Applikations-Lasttests

Michael Jerger, Informatikbüro Jerger

Lasttest bedeutet operieren im Grenzbereich – und damit gibt es leider auch jede Menge Raum für Überraschungen und Fehler. Der Artikel stellt eine durchgängige und abgestimmte Tool-Kette für Lasttests vor und bietet damit für all diejenigen einen sicheren Hafen, die sich neben all den anderen Lasttest-Problemen nicht auch noch intensiv um die Tooling-Frage kümmern wollen.

Die vorgestellten Tools sind natürlich keine magische Allzweckwaffe, sondern funktionieren nur unter bestimmten Voraussetzungen. Die wesentlichen zwei Rahmenbedingungen sind dabei:

- Messdaten werden klassisch in Logfiles geschrieben und nach dem Lasttest ausgewertet.
- Die Auswertung der Messwerte hat die Dimensionen „Zeit" und „Request", wobei jeder Request über alle Systeme hinweg nachverfolgt werden kann.

Abbildung 1 zeigt eine beispielhafte Anordnung der erwähnten Systeme. LES steht dabei für „Last-erzeugendes System" und LP1 bis LP3 sind Messpunkte, an denen für jeden Request individuell die Zeiten gemessen werden.

Umrechnung zwischen Seiten und Requests

Die erste Hürde liegt typischerweise in der Spezifikation der gewünschten Performance. Kunden sprechen vielleicht von Benutzern oder Seiten, Techniker denken eher aber in Requests. Bei der Umrechnung dieser Größen helfen die folgenden beiden Tools:

- *Firebug*
  Firebug (https://addons.mozilla.org/firefox/addon/firebug/) vermittelt (unter dem Reiter „Netzwerk") recht gut, wie viel Requests für die Darstellung einer Seite notwendig sind (siehe Abbildung 2).
- *Browserscope*
  Wer in Firebug genau hinsieht, kann schon erkennen, dass Browser die Ressourcen einer Seite blockweise abholen. Unterschiedliche Browser verwenden unterschiedliche Blockgrößen. Das Projekt „Browserscope" (http://www.browserscope.org) liefert hier für alle Browser und alle Versionen einen nützlichen Überblick über solche Werte (siehe Abbildung 3).

Lasterzeugung

Zur Lasterzeugung gibt es sehr viele unterschiedliche Tools. Nachfolgend aufgezählte Faktoren sind in dieser Zusammenstellung wichtig:

- Um Requests über alle Systeme hinweg identifizierbar zu machen, müssen in irgendeiner Form-Request-IDs injiziert werden können
- Tests sollten effektiv formuliert werden können – am besten mit einer vollständigen Script-Sprache
- Das Erzeugen von Last mit vielen Server-Instanzen sollte mit möglichst wenig Aufwand unterstützt werden

Aus der Menge der möglichen Kandidaten sind hier stellvertretend die folgenden beiden Tools vorgestellt:

- *JMeter*
  Mit JMeter (http://jmeter.apache.org/) können komplexe Tests formuliert werden. Gleichzeitig bietet JMeter die Möglichkeit, viele Last-erzeugende Instanzen komfortabel zu koordinieren. Allerdings muss dafür die entsprechende Anzahl von Server-Instanzen bereitstehen. Für alle Szenarien, in denen kompliziertere Anmeldeverfahren zum Einsatz kommen, ist JMeter das Tool der Wahl (siehe Abbildung 4).

- *Loadimpact*
  Loadimpact (http://loadimpact.com/) ist ein Lasttest-Cloud-Service, mit dem in kürzester Zeit beliebig viel Last-erzeugende Instanzen zum Einsatz gebracht werden können. Die Möglichkeiten, Lasttest-Szenarien zu erstellen, sind nicht so umfangreich, reichen aber für einfache Tests aus. Mit Loadimpact als Cloud-Service ist es möglich, die Last von überall auf der Welt zu erzeugen. Somit ist ein sehr benutzernaher Eindruck der getesteten Web-Applikation möglich (siehe Abbildung 5).

Betriebssystem und Java-VM

Neben der Lasterzeugung ist die Datenmessung ein weiterer zentraler Aspekt von Lasttests. Um den allgemeinen Gesundheitszustand der beteiligten Systeme zu

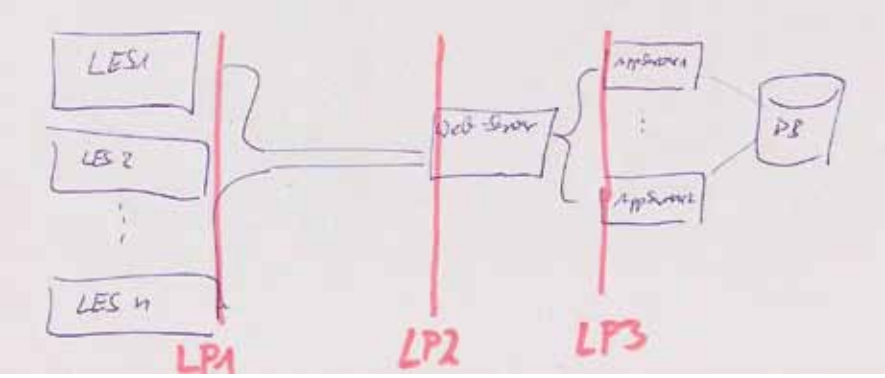


Abbildung 1: Messpunkte bei einer exemplarischen Web-Applikation

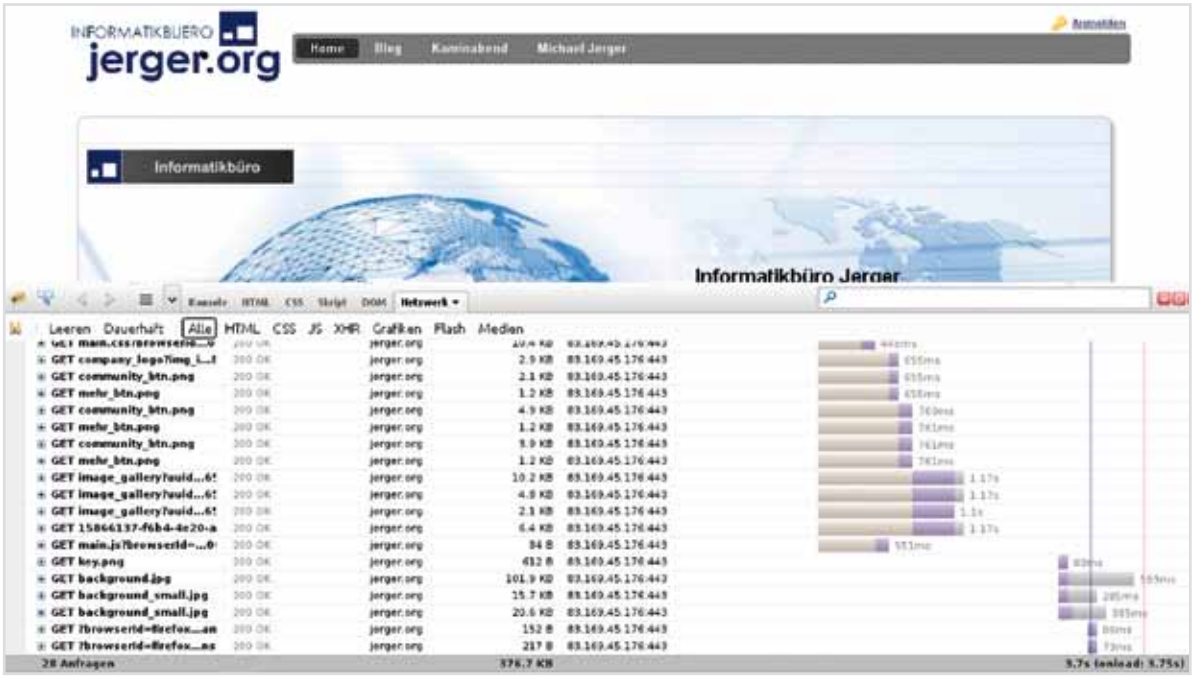


Abbildung 2: Das Laden einer Seite in der Netzwerkansicht

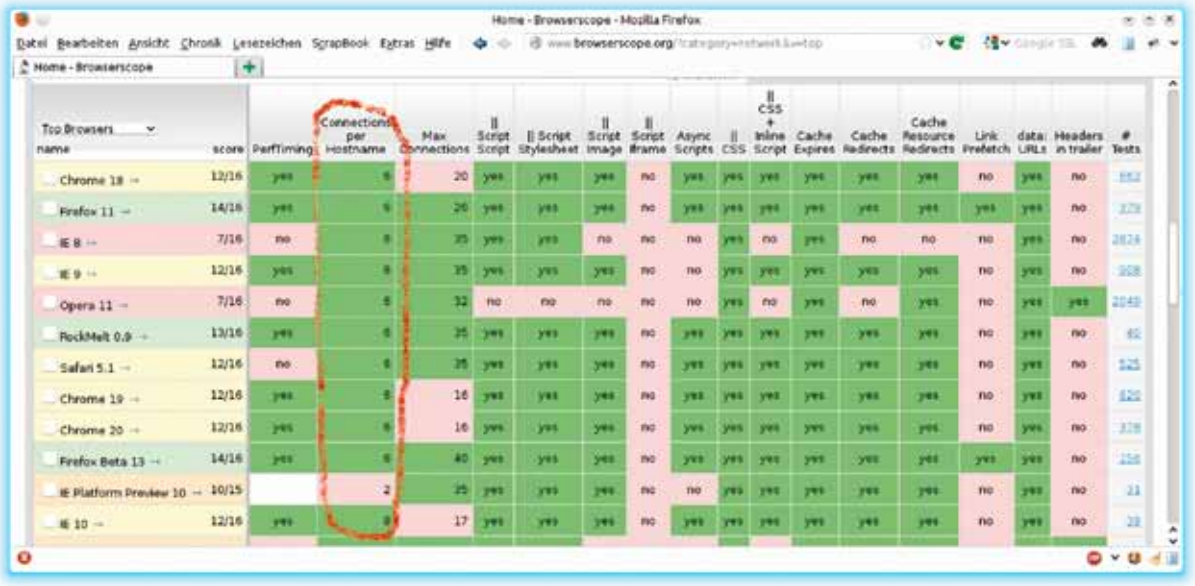


Abbildung 3: Überblick über die Defaultwerte aller Browser und Versionen

messen, eignet sich für Linux das Paket „System Activity Recorder" (SAR). Damit können die aus Top bekannten Werte laufend erfasst und nach dem Lasttest als Komma-separierte Liste ausgegeben werden (siehe Listing 1). Für eingesetzte Java-VMs ist vor allem deren Speicherverbrauch interessant. Der Garbage Collector erlaubt mit der folgenden Option, alle Speicherbewegungen in ein Logfile zu schreiben (siehe Listing 2).

Web- und Applikationsserver

Als Messdaten aus Web- und Applikationsserver kann das jeweilige Access-Log verwendet werden. Apache httpd oder tomcat bieten den Parameter „%D" für die Messung der Request-Response Zeit an. Die im Last erzeugenden System injizierten Request-Id's werden als URL-Bestandteil typischerweise ebenfalls geloggt. Damit ist die Request-Verfolgung möglich (siehe Listing 3 und 4). Die hinteren beiden Zahlen in den beiden Listings geben die verbrauchte Zeit an. Beachtenswert: „tomcat" gibt Zeiten in Millisekunden aus, „httpd" hingegen in Mikrosekunden. Für eine Umrechnung ist hier also der Faktor 1.000 notwendig.

Log-Transformation

Nach einem Lasttest müssen die unterschiedlichen Logfiles aus allen beteiligten Systemen zusammengeführt werden. Mit dem BI-Tool Pentaho (http://community.

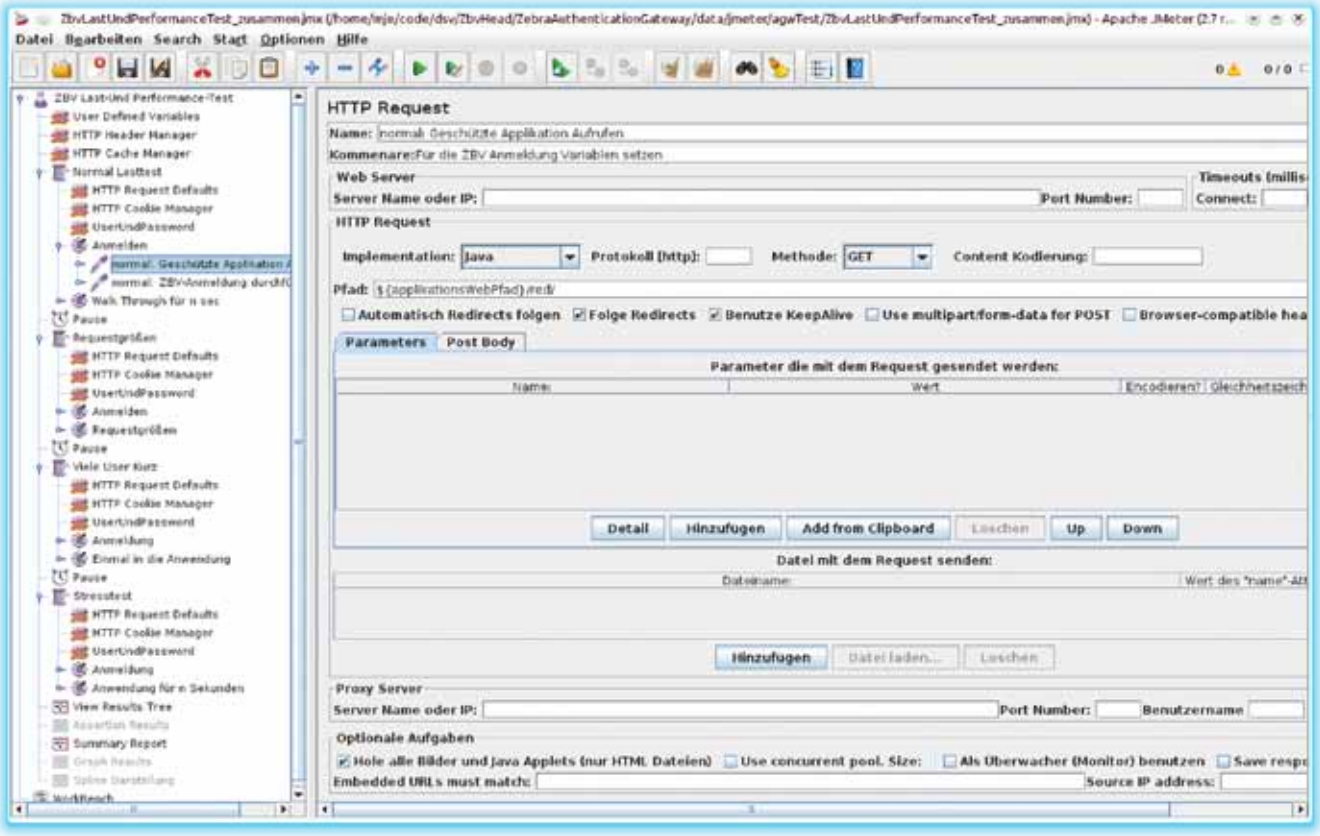


Abbildung 4: Ein Test-Szenario unter Jmeter

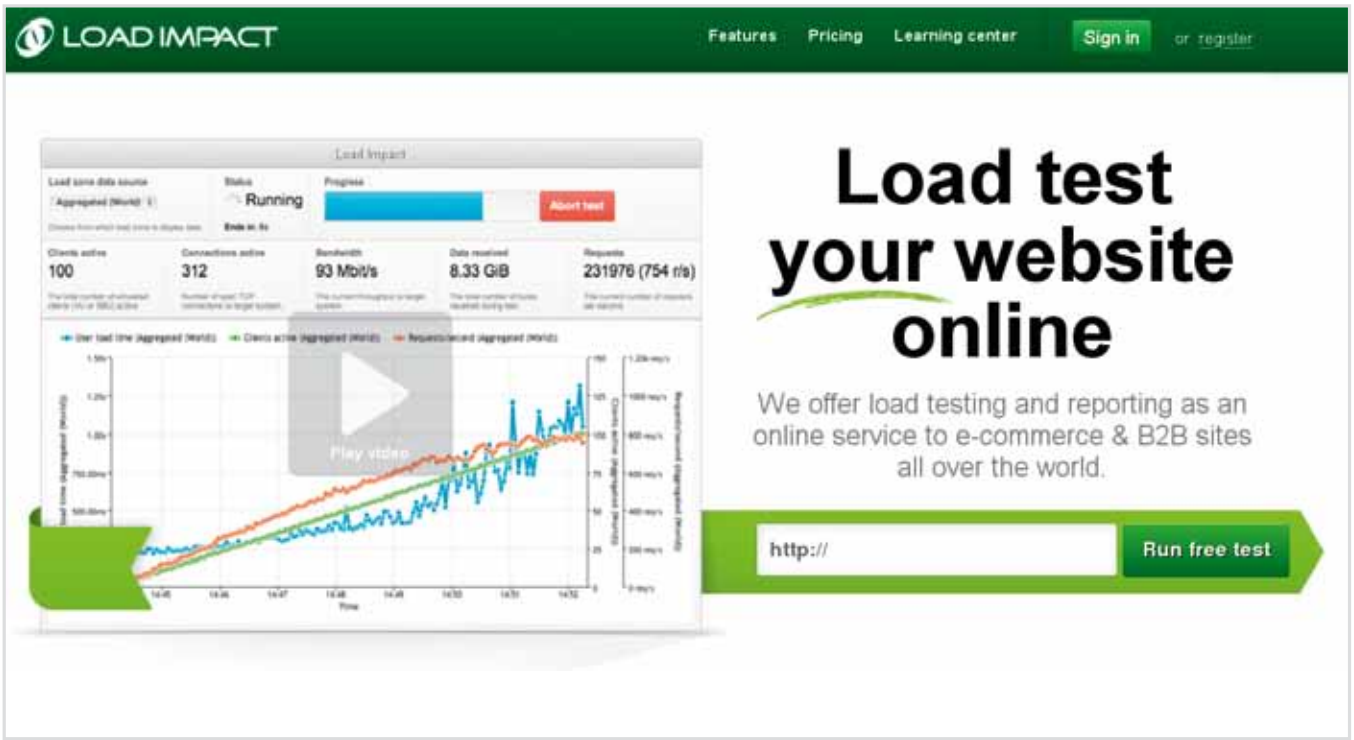


Abbildung 5: Mit Loadimpact testen aus der ganzen Welt

```
CPU:        sadf -d /var/log/sysstat/sa25 -- -u
Hostname    Inter-    Timestamp     CPU      %user     %nice     %system   %iowait   %idlew
            val
www.        600       2012-06-15    0.28     0.00      0.06      0.08      0.00      99.57
jerger.                06:55:011
org
Disk:       sadf -d /var/log/sysstat/sa25 -- -b
hostname    inter-    timestamp     tps      rtps      wtps      bread/s   bwrtn/s
            val
www.        600       2012-06-15    0.00     0.00      0.00      0.00      0.00
jerger.                06:55:01
org
Mem:        sadf -d /var/log/sysstat/sa25 -- -r
hostname    inter-    timestamp     kbmem-   kbme-     %me-      kbbuf-    …
            val                     free     mused     mused     fers
www.        600       2012-06-15    85824    3059904   97.27     0         …
jerger.                06:55:01
org
Load: sadf -d /var/log/sysstat/sa25 -- -q
hostname    inter-    timestamp     runq-    plist-    ldavg-1   ldavg-5   ldavg-15
            val                     sz       sz
www.        600       2012-06-15    0        207       0.09      0.08      0.03
jerger.                06:55:01
org
Network:    sadf -d /var/log/sysstat/sa25 -- -n DEV
hostname    inter-    timestamp     IFACE    rxkB/s    txkB/s
            val
www.        600       2012-06-15    lo       12.86     12.86
jerger.                06:55:01
org
```

Listing 1: Parameter und Beispiel-Ausgaben für SAR

```
java … -verbose:gc -Xloggc:<file> …
0.068: [GC 5244K->320K(1004928K), 0.0034960 secs]
0.072: [Full GC 320K->241K(1004928K), 0.0037430 secs]
```

Listing 2: Commandline und Beispiel-Ausgaben für das gc-log

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\" %D"
79.125.101.106 … [15/Jun/2012:20:28:41 +0200] "GET …/Leistungsnachweis.png?X-
requestId=spec1621339784894 … 341001
```

Listing 3: Parameter und Bsp. für das httpd access.log

```
<Valve className="org.apache.catalina.valves.AccessLogValve" … pattern="%h %l %u %t
&amp;quot;%r&amp;quot; %s %b %D %S" … />
79.125.101.106 … [15/Jun/2012:20:28:42 +0200] „GET …/Leistungsnachweis.png?X-
requestId=spec1621339784894 … 231 …
```

Listing 4: Parameter und Bsp. für das tomcat access.log

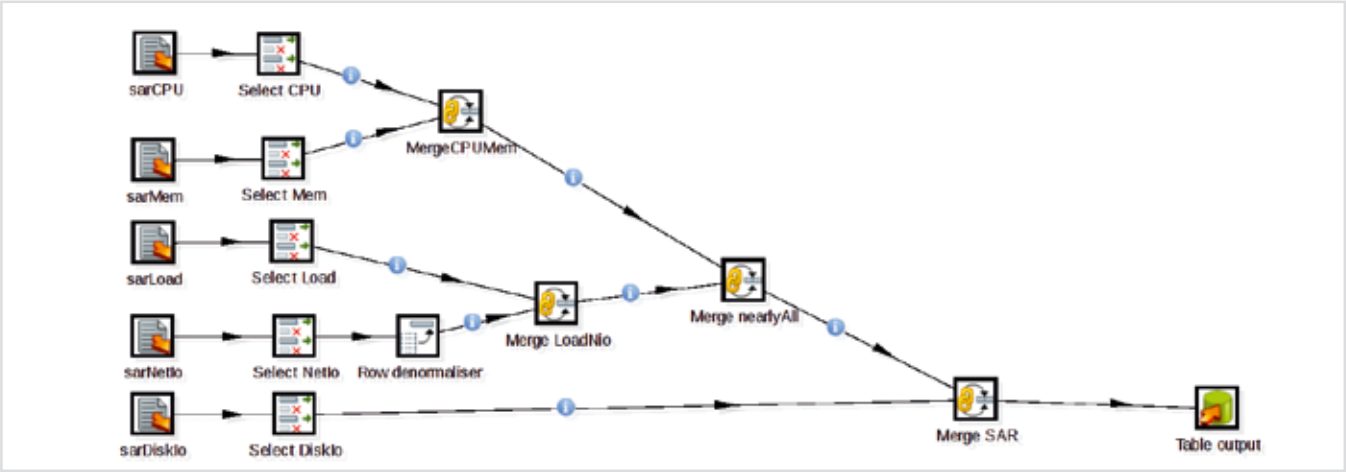


Abbildung 6: Die Transformation für SAR-Werte. Die Ausgabe erfolgt in eine Datenbank.

Abbildung 7: Anzahl der Requests und Requestzeiten sekundenweise aggregiert

pentaho.com/) lässt sich das gut bewältigen. Die Extraktion und Transformation der Daten kann mit Pentaho grafisch definiert werden. Das auftretende Datenvolumen stellt für die ausgewachsene BI-Software keine große Herausforderung dar (siehe Abbildung 6).

Darstellung

Nachdem alle Daten auf diese Weise in einer Datenbank zur Verfügung stehen, ist die weitere Verarbeitung mit SQL und Excel keine große Herausforderung mehr (siehe Abbildung 7).

Fazit

Die vorgestellten Tools haben sich so schon in vielen Lasttests bewährt und stellen eine gute Grundausrüstung dar. Die vorgestellte Tool-Kette ist wenig invasiv und kann leicht um weitere Messmethoden ergänzt werden.

Unter https://www.domaindrivenarchitecture.org/lasttest-web finden Sie alle hier gezeigten Ressourcen (Pentaho Transformation, SQL's, Loadimpact Testscripte) und ein ausführliches Lasttest-Konzept unter einer Creative Common Lizenz.

Michael Jerger
buero@jerger.org

Michael Jerger ist IT-Architekt, Projektmanager und immer noch neugierig. Sein Motto lautet „Demotivation eliminieren bedeutet mit Vergnügen arbeiten" – was stets schnell zu den wirklich interessanten Punkten im Projekt führt und oft auch zu sehr erfolgreichen Teams.

Unbekannte Kostbarkeiten des SDK

Heute: JavaScript

Bernd Müller, Ostfalia, Hochschule für angewandte Wissenschaften

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

Das Java-Scripting-API, definiert als JSR 223 (Scripting for the Java Platform [1]), hatte ursprünglich die Integration von Skript-Sprachen zur Erzeugung von Web-Seiten zum Ziel. Zum Zeitpunkt der Initiierung des JSR im Jahr 2003 waren dies vor allem PHP und Perl. Wenn man heute von „Scripting" im Zusammenhang mit Java spricht, werden häufig andere Sprachen, etwa Groovy oder JPython, eventuell auch Scala genannt. Da diese Sprachen sehr gute Schnittstellen von und zu Java besitzen, wird in der Regel das Scripting-API für diese Sprachen selten verwendet. Für PHP und Perl kommt dieses zumindest nach unserer Kenntnis ebenfalls selten vor, es gehört also zu den unbekannten Kostbarkeiten des SDK. Der Artikel stellt das Scripting-API kurz vor und geht insbesondere auf die Verwendung der im SDK enthaltenen JavaScript-Implementierung ein, da diese durchaus nutzbringend eingesetzt werden kann.

Java und JavaScript

Das im JSR 223 [1] definierte Scripting-API ist seit Java 6 Bestandteil des SDK und im Package „javax.script" durch insgesamt 12 Interfaces, Klassen und Exceptions repräsentiert. Neben dieser allgemeinen Schnittstellen-Definition für verschiedene Skript-Sprachen ist jedoch auch eine Implementierung einer Skript-Sprache im SDK 6 enthalten. Dies ist die JavaScript-Implementierung der Mozilla Foundation mit Code-Namen „Rhino" [2]. Sie implementiert JavaScript nach der Spezifikation ECMA-262 [3]. Rhino unterliegt, wie für die Mozilla Foundation üblich, einer Open-Source-Lizenz und ist vollständig in Java implementiert.

Das Scripting-API

Die Klasse „ScriptEngineManager" realisiert den Einstiegspunkt zur Ermittlung aller bekannten Script-Implementierungen. Sie basiert auf dem Konzept eines Service-Loaders, das wir in Java aktuell Q4/2011 im Rahmen dieser Reihe vorgestellt haben [4]. Listing 1 zeigt den Zugriff auf alle bekannten Script-Implementierungen und gibt einige Details dieser Implementierungen aus.

Die im Listing 2 dargestellte Ausgabe erscheint bei der Ausführung des Programms aus Listing 1 auf dem System des Autors (JDK 7u5), wenn Groovy im Klassenpfad ist. Wie man sieht, registriert sich auch Groovy über den Service-Loader.

Zu betonen ist hier nochmals, dass die JavaScript-Implementierung seit Version 6 im JDK enthalten ist und nicht installiert oder in den Klassenpfad aufgenommen werden muss.

Die Verwendung von JavaScript

Soll JavaScript verwendet werden, ohne nach allen Script-Implementierungen zu suchen, so erlaubt der „ScriptEngineManager" den direkten Zugriff auf eine Implementierung über den Namen (getEngineByName()), die Dateinamenserweiterung (getEngineByExtension()) oder den

```
ScriptEngineManager manager = new ScriptEngineManager();
List<ScriptEngineFactory> factoryList =
            manager.getEngineFactories();
      for (ScriptEngineFactory factory : factoryList) {
            System.out.println(factory.getEngineName()
                  + " " + factory.getEngineVersion()
                  + " / " + factory.getLanguageName()
                  + " " + factory.getLanguageVersion());
      }
```

Listing 1

```
Mozilla Rhino 1.7 release 3 PRERELEASE / ECMAScript 1.8
Groovy Scripting Engine 2.0 / Groovy 1.7.2
```

Listing 2

```
ScriptEngine engine = manager.getEngineByName ("JavaScript");
String userInput = "2 + 3 * 4";
engine.eval ("print (" + userInput + ")");
```

Listing 3

```
ScriptEngine engine = manager.getEngineByName ("JavaScript");
engine.put("x", 2);
engine.put("y", 3);
engine.put("z", 4);
engine.eval (new FileReader("./formula.js"));
Double result = (Double) engine.get("result");
```

Listing 4

```
// Datei ‚formula.js‘
// kompliziertere Formel fuer ‚result‘
result = Math.pow(x, y) * Math.sqrt(z);
```

Listing 5

Mime-Type (getEngineByMimeType()). Als kleines Beispiel definieren wir die Anforderung, dass der Benutzer einen einfachen arithmetischen Ausdruck eingeben können soll, der dann ausgewertet wird. Wir wollen damit einen einfachen Taschenrechner realisieren.

In Java EE kann man dafür auf die Expression Language zurückgreifen. In Java SE muss der Eingabe-String (etwa ein Textfeld in Swing, SWT oder JavaFX) zunächst in seine Tokens zerlegt, der Syntax-Baum aufgebaut und anschließend ausgewertet werden. Bei JavaScript wertet der Code in Listing 3 eine fiktive Benutzereingabe aus und gibt sie nach „System.out“ aus.

Mehrere Benutzereingaben und deren Kombination in einem komplexeren Ausdruck lassen sich ebenfalls leicht realisieren. In Listing 4 werden fiktive Benutzereingaben über ein Language-Binding an die JavaScript-Variablen x, y und z gebunden. Die „eval()“-Methode wertet die JavaScript-Anweisung in der Datei „formula.js“ aus und stellt das Ergebnis über die „get()“-Methode wieder Java zur Verfügung. Listing 5 zeigt die Datei „formula.js“.

Die verwendete „put()“-Methode bindet die Variablen global für die „ScriptEngine“. Das Interface „Bindings“ erlaubt es, eine Menge von Variablen-Bindungen zusammenzufassen und als weiteren Parameter der „eval()“-Methode zu übergeben. Das Interface „ScriptContext“ gestattet die Definition von Kontexten zur Skript-Ausführung. Beispiele beider Konstrukte liegen aber außerhalb des einführenden Charakters dieses Artikels.

Fazit

Die verborgenen Kostbarkeiten „Scripting-API“ und „JavaScript“ sind seit Java 6 im SDK enthalten. Exemplarisch haben wir gezeigt, wie damit einfache, aber auch kompliziertere Arithmetik möglich ist, ohne einen eigenen Interpreter für derartige arithmetische Ausdrücke implementieren zu müssen.

Literatur/Web

[1] http://jcp.org/en/jsr/detail?id=223
[2] http://www.mozilla.org/rhino
[3] http://www.ecma-international.org/publications/standards/Ecma-262.htm
[4] Bernd Müller, Unbekannte Kostbarkeiten des SDK: Der Service-Loader. Java aktuell Q4/2011

Bernd Müller
bernd.mueller@ostfalia.de

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

SAP NetWeaver Cloud: Eine OpenSource-basierte Java-Cloud-Plattform

Matthias Steiner, SAP AG

Die Cloud revolutioniert die Software-Industrie und auch die großen Namen der Branche müssen sich neu orientieren, um zukunftsfähig zu bleiben. SAP möchte mit seiner offenen Cloud-Plattform Entwickler für sich gewinnen und den Einstieg in die Welt der Unternehmens-Software so einfach wie möglich machen.

Die Software-Industrie ist bekanntlich eine sehr schnelllebige Branche, in der ein Trend den nächsten jagt. Zurzeit dominieren „Mobile" und „Cloud" die Medien und die Strategie-Planung in den Unternehmen. Im Unterschied zu vielen anderen Hypes ist jedoch abzusehen, dass diese beiden Technologien den IT-Sektor nachhaltig verändern werden.

Bei näherer Betrachtung fällt auf, dass die Themenkomplexe „Mobile" und „Cloud" eine zentrale Eigenschaft gemein haben: Beide zielen darauf ab, Daten und Dienste einfach und schnell zugreifbar zu machen – jederzeit an jedem Ort. Diese Vision ist durch die stetig steigende Zahl an Smartphones und Tablets mittlerweile vielerorts zur Realität geworden und diese nun als alltäglich wahrgenommene Erfahrung beeinflusst letztendlich auch das Enterprise-Segment. Firmen erwarten zu Recht dieselbe Agilität und Produktivität auch in Bezug auf Unternehmens-Software und es scheint nur schwer vermittelbar zu sein, warum diese Erwartung nicht auch im beruflichen Umfeld umsetzbar sein sollte. Die Cloud verspricht diesem Bedarf an unkomplizierter und schnell implementierbarer Software gerecht zu werden.

Somit erklärt sich, warum dieser Tage zahllose Cloud-Anbieter auf den Markt drängen, um sich eine Position in diesem Segment zu sichern, darunter auch SAP. Cloud-Computing ist eine relativ junge Technologie und der Begriff wird teilweise recht undifferenziert verwendet. Deshalb ist es sinnvoll, den Themenkomplex kurz zu umreißen, um sich dann auf den Teil zu konzentrieren, der für Entwickler von Cloud-Lösungen wohl der interessanteste sein dürfte: Platform-as-a-Service (PaaS).

Die Cloud

Das gängige Erklärungsmodell unterteilt den Begriff „Cloud" in drei aufeinander aufbauende Servicemodelle (siehe Abbildung 1.) Die unterste Schicht – Infrastructure-as-a-Service (IaaS) – entspricht wohl am ehesten dem landläufigen Verständnis des Begriffes „Cloud". Auf dieser Ebene geht es primär um die Bereitstellung und den Betrieb von Hardware, um Kunden unkompliziert und schnell Rechenkapazität zur Verfügung zu stellen. Vereinfacht dargestellt erhalten Kunden in diesem Modell lediglich die Ressourcen, um eigene (virtualisierte) Software zu betreiben. Der Charme hierbei ist, dass große IaaS-Betreiber wie Amazon diese Rechenkapazitäten auf Knopfdruck bereitstellen und nahezu beliebig skalieren können.

Platform-as-a-Service (PaaS) setzt auf IaaS auf und bietet eine einheitliche Programmier- und Laufzeit-Umgebung, um die Entwicklung und den Betrieb von Anwendungen in der Cloud zu erleichtern. Hierfür werden vom PaaS-Anbieter sämtliche Ressourcen wie Datenbank, Speicherplatz und Netzwerk-Anbindung zur Verfügung gestellt und gemanagt, sodass sich Entwickler voll und ganz auf die Realisierung konzentrieren können. Im Idealfall unterstützt eine solche Plattform den gesamten Lebenszyklus einer Anwendung, angefangen von der Entwicklung bis hin zum Betrieb und der Abrechnung mit Endkunden.

Abbildung 1: Cloud-Servicemodelle

Bei Software-as-a-Service (SaaS) spricht man von einem Modell, bei dem Anwendungen nicht gekauft und dann selbst betrieben oder gehostet, sondern von einem SaaS-Anbieter als Service zur Verfügung gestellt werden. Hierbei übernimmt der Anbieter die gesamte Administration und Wartung der Software – der Kunde zahlt lediglich eine nutzungsabhängige Gebühr.

Das SaaS-Modell ist für kleine und mittelständische Unternehmen sehr interessant, da es ihnen ermöglicht, Unternehmens-Software einzuführen oder zu betreiben, ohne große Investitionskosten tätigen zu müssen. Der Einsatz von Cloud-basierten Anwendungen hat auch in Bezug auf laufende Kosten Vorteile, da diese analog zu dem Wachstum einer Firma zunehmen.

Für größere Unternehmen bietet SaaS mehr Flexibilität, da die einzelnen Unternehmenszweige unabhängiger werden und sich bei der Einführung von neuen Anwendungen und Diensten nicht mehr notwendigerweise den langwierigen und komplexen Updatezyklen einer unternehmensweiten IT-Strategie unterordnen müssen.

SAP Store

SAPs Cloud-Strategie dreht sich hauptsächlich um SaaS und PaaS, allerdings betreibt SAP auch die Infrastruktur selbst, um unabhängig von externen IaaS-Anbietern zu sein und alles aus einer Hand liefern zu können. Das Engagement im PaaS-Segment erklärt sich am besten dadurch, indem man erneut die Parallelen zum Mobile Business aufgreift. Hier hat Apple eindrucksvoll bewiesen, dass es entscheidend ist, die Entwickler-Gemeinde für sich zu gewinnen, um eine attraktive Plattform zu etablieren.

Je größer die Anzahl der aktiven Entwickler für eine Plattform ist, desto größer und vielseitiger ist auch die Auswahl an Software, die man über diese Plattform beziehen kann. Diesen Ansatz greift SAP mit der Kombination aus SAP NetWeaver Cloud und dem SAP Store auf und überträgt ihn in den Bereich der Unternehmenssoftware.

SAP NetWeaver Cloud

Bei SAP NetWeaver Cloud handelt es sich um eine Java-basierte Cloud-Plattform, die weitestgehend auf Open-Source-Frameworks basiert. Bei der Auswahl der zu verwendenden Komponenten wurde besonders viel Wert auf die dahinterstehende Unterstützung durch die Entwickler-Gemeinde gelegt, um sicherzustellen, dass nur Frameworks verwendet werden, die weit verbreitet sind und eine solide und aktive Entwickler-Basis besitzen. Diese Herangehensweise führte zu der Entscheidung, viele Projekte im Umfeld von Eclipse und Apache zu verwenden, an deren Weiterentwicklung sich SAP beteiligt oder die sie sogar maßgeblich vorantreibt wie etwa das Eclipse-Virgo-Projekt.

Hintergrund der Idee, hauptsächlich auf Open-Source-Komponenten zu setzen, ist die Absicht, die Einstiegshürde für Entwickler so niedrig wie möglich zu halten, da sie bekannte Komponenten vorfinden und sich nicht extra neues Wissen aneignen müssen, um auf der Plattform zu entwickeln. Diese Herangehensweise scheint auch aus Unternehmenssicht sinnvoll und wünschenswert, da sie die Ausbildung von Mitarbeitern und/oder die Suche nach Fachkräften wesentlich erleichtert und somit dazu beiträgt, die Entwicklungskosten zu senken.

Architektur

Aus technischer Sicht handelt es sich bei SAP NetWeaver Cloud um eine OSGi-Laufzeitumgebung basierend auf der Eclipse-Equinox-Referenz-Implementierung, in die ein Apache-Tomcat-Server integriert ist. Die Modularität von OSGi in Kombination mit einem leichtgewichtigen Server wie Tomcat ist sehr Ressourcen-schonend, was sich positiv auf das Startup-Verhalten des Servers auswirkt. Durch die weite Verbreitung von Tomcat und den Reifegrad der Servlet-Spezifikation erhält man eine solide und ausbaufähige Basis für die Entwicklung von Cloud-Anwendungen, die durch unzählige Open-Source-Projekte ergänzt und erweitert werden kann.

Für die gängigsten Anforderungen in der Anwendungsentwicklung bietet SAP NetWeaver Cloud außerdem eine Reihe von sogenannten „Plattform-Services", die typische Arbeitsschritte vereinfachen und effizienter machen.

Identity Management

Authentifizierung ist ein wichtiges Thema in der Cloud, gerade in Bezug auf SaaS; schließlich will man es den Benutzern nicht zumuten, sich unzählige Passwörter zu merken und sich permanent neu anmelden zu müssen. Um dieses Problem zu lösen, bietet SAP einen ID-Management-Service an, der zentral die Authentifizierung von Benutzerdaten übernimmt und diese dann an den entsprechenden Identitätsanbieter (Identity Provider) im Unternehmens-Netzwerk delegiert. Dieser Ansatz ermöglicht einen sogenannten „Single Sign-On"-Ansatz (SSO) auf Basis des SAML2-Standards, bei dem Backend-Systeme und Cloud-Anwendungen auf einen gemeinsamen Benutzerstamm zugreifen. Alternativ kann natürlich auch deklarative oder programmatische Authentifizierung verwendet werden, wobei erstere konform zur Servlet-Spezifikation in der „web.xml" definiert wird und letztere über ein dediziertes API erreicht werden kann.

Persistenz

In Bezug auf die Datenbank-Anbindung greift erneut der Plattform-Charakter und die Anbindung wird aus Entwickler-Sicht durch den Persistency-Service abstrahiert. Um Zugriff auf die Datenbank zu bekommen, müssen Entwickler lediglich eine Referenz in der „web.xml" ihrer Web-Anwendung deklarieren. Zur Laufzeit kann man dann über JNDI eine Referenz auf die entsprechende DataSource bekommen. Mit dieser Referenz bleibt es Entwicklern freigestellt, ob sie direkt die Datenbank via JDBC und SQL ansprechen wollen oder das Java-Persistence-API (JPA) basierend auf EclipseLink benutzen möchten. Zurzeit bietet SAP NetWeaver

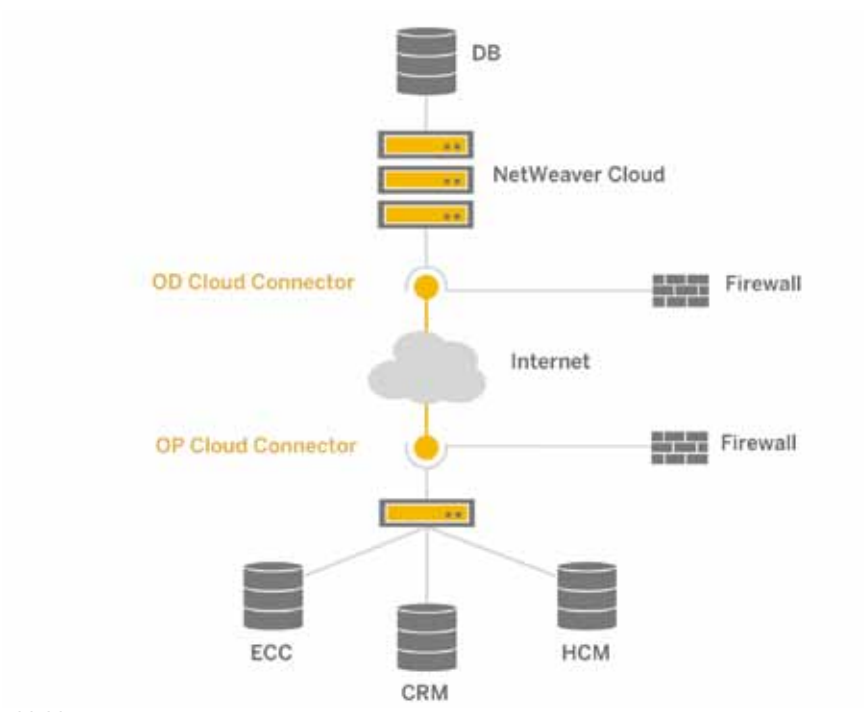


Abbildung 2: Connectivity-Service

Cloud Unterstützung für relationale Datenbanken sowie SAPs neue In-Memory-Datenbank namens „HANA“; in naher Zukunft soll auch Sybase Adaptive Server Enterprise (ASE) als Option angeboten werden.

Konnektivität

Eines der zentralen Themen in den nächsten Jahren wird der Hybrid-Einsatz von klassischen OnPremise-Systemen und neuen Anwendungen und Diensten in der Cloud sein. Hier spielt die Integration existierender Backend-Systeme mit neuen SaaS-Lösungen eine zunehmend größere Rolle, denn sie verspricht einen unmittelbaren Mehrwert, ohne große Veränderungen an bestehenden Systemen und Prozessen notwendig zu machen. SAP nennt diesen Ansatz „Innovation without disruption.“

Um solche Szenarien zu ermöglichen, bietet SAP NetWeaver Cloud den sogenannten „Connectivity Service“, der eine sichere Verbindung zwischen der Cloud und dedizierten Backend-Systemen herstellt. Hierfür muss lediglich eine spezielle Komponente (SAP Cloud Connector) in der OnPremise-IT-Landschaft installiert sein. Über den Cloud Connector wird dann ein permanenter SSL-Tunnel zu dem entsprechenden Cloud-Account hergestellt (siehe Abbildung 2). Die Kommunikation erfolgt dann entweder über HTTP(S) oder in naher Zukunft auch über das SAP-eigene RFC-Protokoll.

Dokumenten-Management

Für das Arbeiten mit unstrukturierten Daten, also Dokumenten, bietet SAPs Cloud-Plattform einen dedizierten Document-Service basierend auf dem „Content Management Interoperability Services“-Standard (CMIS) an. Als Implementierung der CMIS-Spezifikation dient das Apache-Chemistry-Projekt, wodurch Entwickler zusätzlich zu den Web-Service- und AtomPub-Protokollen auch die Möglichkeit haben, direkt über ein Java-API mit dem Content-Management-Repository zu interagieren. Hierbei ist lediglich die Instanziierung der CMIS-Session Plattform-spezifisch; ansonsten benutzen Entwickler lediglich das API von Apache Chemistry.

Für die Zukunft sind weitere Funktionalitäten geplant, die unter anderem das Zusammenspiel mit anderen Lösungen von SAP verbessern und erleichtern sollen. Dies gilt zum einen in Bezug auf bessere Integrationsmöglichkeiten von SaaS-Anwendungen beispielsweise von SuccessFactors, aber zum anderen auch ganz generell in Hinblick auf mobile Szenarien sowie Analytics, soziale Netzwerke sowie generische Integrationslösungen, wie man sie von klassischer Middleware gewohnt ist. Zu guter Letzt ist auch die Unterstützung alternativer Programmiersprachen wie JRuby geplant.

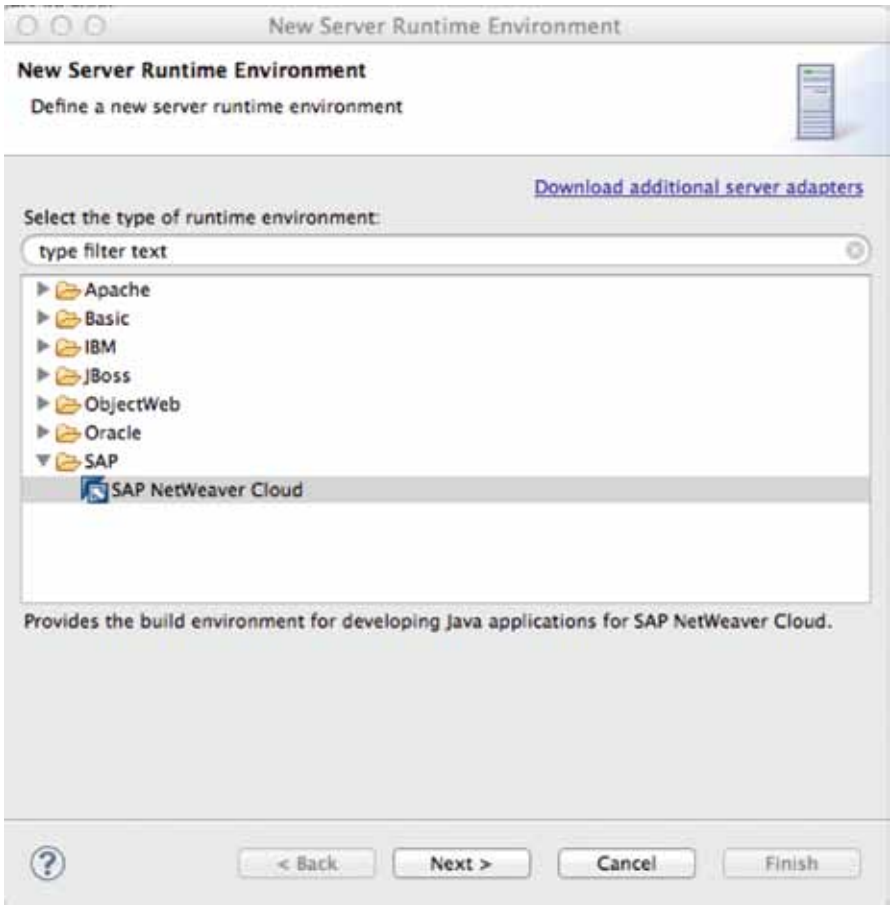


Abbildung 3: Server-Runtime-Konfiguration

Lokale Entwicklung

In Bezug auf die Entwicklungs-Umgebung setzt SAP auf Eclipse und hat die notwendigen Tools in die „Web Tools Platform“ (WTP) integriert. Auch hier war der Gedanke ausschlaggebend, den Einstieg für Entwickler so leicht wie möglich zu machen. Um Web-Anwendungen wie gewohnt zu entwickeln, müssen Entwickler lediglich das Verzeichnis des NetWeaver-Cloud-SDK konfigurieren und können dann NetWeaver Cloud als Laufzeit-Umgebung/Server auswählen (siehe Abbildung 3).

Um Anwendungen effektiv zu entwickeln und vor dem Deployment in die Cloud testen zu können, bietet NetWeaver Cloud die Möglichkeit, einen lokalen Tomcat-Server zu starten. Um ein vergleichbares Laufzeit-Verhalten zur Cloud zu gewährleisten, stehen für diesen Fall Stub-Implementierungen der Plattform-Services bereit, die das Verhalten der Cloud im lokalen Betrieb simulieren beziehungsweise ersetzen.

Monitoring und Application-Management

Um Anwendungen in der Cloud zu überwachen und zu managen, existiert für jeden Account eine dedizierte Website, über die man sich über den aktuellen Status der Anwendungen informieren kann. Von hier aus können Anwendungen ge-

startet und gestoppt sowie Server-Logs untersucht werden. Darüber hinaus können Administratoren auch Auslastung und Zustand der einzelnen Serverknoten einsehen und weitere Status-Informationen überwachen.

Developer Center

NetWeaver Cloud versteht sich als offene Plattform für die Entwicklung von Cloud-Anwendungen, doch dazu gehört mehr, als nur auf offene Standards und Open Source zu setzen. Ein wesentlicher Bestandteil dieser Philosophie beinhaltet auch einen offenen und kontinuierlichen Dialog mit der Entwicklergemeinde und dem Ökosystem. Um diesem Anspruch gerecht zu werden, hat SAP innerhalb des SAP Community Networks unter „developers.sap.com“ ein dediziertes Forum eingerichtet, das „Developer Center“. Auf dieser Plattform kann sich die Community austauschen und mit den neusten Informationen rund um NetWeaver Cloud und andere Technologien bei SAP versorgen. Über diese Website sind auch die Online-Hilfe (siehe help.netweaver.ondemand.com) sowie die Release-Notes der einzelnen Builds öffentlich zugänglich.

Für diejenigen, die nun Interesse verspüren, sich die Plattform näher anzuschauen und auszuprobieren, steht eine 90-Tage-Testversion zur Verfügung. Das NetWeaver-Cloud-SDK wird auf dem Developer Center als Download angeboten. Dort kann man sich auch einen kostenlosen Account für die Test-Landschaft einrichten.

Matthias Steiner
matthias.steiner@sap.com

Matthias Steiner arbeitet als Cloud-Platform-Evangelist bei der SAP AG. Er beschäftigt sich mit den Themen „Cloud“, „Open Source“ und „Web-Anwendungen“. In seiner früheren Rolle als Software-Architekt hat er zahlreiche große Kunden-Projekte basierend auf Java-EE- und SAP-Technologien erfolgreich geleitet. Als aktiver Blogger und überzeugter Nutzer von sozialen Netzwerken gilt sein besonderes Interesse dem Austausch mit der Entwicklergemeinde, unter anderem auf Twitter unter dem Kürzel „@steinermatt“.

Abbildung 1: Dichtes Gedränge auch während der Pausen in der Ausstellung

Java Forum Stuttgart – **ein voller Erfolg**

Stefanie Nolda, Java User Group Stuttgart

Am 5. Juli 2012 richtete die Java User Group Stuttgart (JUGS) unter der Leitung von Dr. Michael Paus zum 15. Mal in Folge das Java Forum Stuttgart aus.

Das Interesse an der Konferenz mit „Java“ als Leitmotiv ist in den letzten Jahren stetig gewachsen, sodass auch dieses Mal mehr als 1.200 Besucher in die Liederhalle Stuttgart strömten, um sich informative Vorträge anzuhören, an den Ständen über neueste Produkte zu informieren oder einfach nur, um sich mit Kollegen auszutauschen.

„Die bewährte Mischung aus Technologie-Vorträgen, Projektberichten und Produkt-Präsentationen traf auch dieses Jahr den Nerv der Java Interessierten“, so Dr. Michael Paus, Vorsitzender der Java User Group Stuttgart. „Nicht zu vergessen die exklusive Bewirtung.“

Die Besucher konnten sich in 42 Vorträgen und an 35 Ausstellungsständen umfassend informieren, kostenlose Zeitschriften mitnehmen, sich an der Jobbörse umsehen oder am köstlichen Buffet bedienen. Zahlreiche Helfer sorgten für einen reibungslosen Ablauf. Aufgrund des erneuten großen Erfolgs haben die Planungen für das nächste Jahr bereits begonnen.

Stefanie Nolda
office@java-forum-stuttgart.de

Fotos: Wolfgang Taschner

Abbildung 2: Selbst im größten Saal waren die Vorträge machmal bis auf den letzten Platz besetzt

Cross-Origin-Resource-Sharing

Dr. Fabian Stäber, NorCom Information Technology AG

Mit Cross-Origin-Resource-Sharing hat das W3C ein Verfahren vorgeschlagen, um REST-Services von verschiedenen Domains browserseitig in einer Web-Anwendung zu integrieren. Dies war durch die Same-Origin-Policy in Web-Browsern bisher nur eingeschränkt möglich. Dieser Artikel stellt Cross-Origin-Resource-Sharing vor und zeigt, wie man das Verfahren für Java-basierte REST-Services nutzen kann.

In klassischen Web-Anwendungen beschränkt sich der browserseitige Teil auf die reine Darstellung von Web-Seiten. Seit einigen Jahren ist jedoch der Trend zu beobachten, dass immer mehr Funktionalität vom Server in den Browser verlagert wird. Diese Entwicklung begann mit der Verbreitung von Ajax, das es dem Browser erlaubt, asynchron im Hintergrund mit dem Server zu kommunizieren und die Ergebnisse dynamisch in das Frontend einzubinden.

Mit der Verbreitung von HTML5 nimmt dieser Trend erneut an Fahrt auf: Die JavaScript-Umgebung, die in HTML5-Browsern zur Verfügung steht, beinhaltet Funktionalitäten wie Offline-Speicher, browserseitige SQL-Datenbanken, WebSockets für die schnellere Kommunikation mit dem Server usw. Gleichzeitig gibt es inzwischen eine Vielzahl von browserseitigen Frameworks, die die Entwicklung von komplexen, strukturierten JavaScript-Anwendungen vereinfachen.

Dieser Trend hat auch Einfluss auf die serverseitige Architektur. Während in klassischen Server-Backends der komplette Stack von der Datenhaltung über die Business-Logik bis hin zum Rendering der HTML-Seiten benötigt wurde, zeichnet sich heute eine Tendenz zu leichtgewichtigen Backends ab, die einfache REST-Services zur Verfügung stellen. Das Auslagern von Funktionalität in den Browser ermöglicht eine hohe Skalierbarkeit von Web-Anwendungen sowie die Vereinfachung der serverseitigen Architektur.

Integration verschiedener Services im Browser

Mit dem Trend zu leichtgewichtigen REST-Services rückt die Frage in den Vordergrund, wie sich mehrere dieser Services in eine Anwendung integrieren lassen. In klassischen serviceorientierten Architekturen erfolgt die Integration serverseitig: Dem Browser steht ein Web-Server zur Verfügung, der intern die verschiedenen Services integriert.

Mit dem aktuellen Trend zu mehr Funktionalität im Browser stellt sich die Frage, wie sich diese Integration browserseitig implementieren lässt. Die Serverseite wäre dadurch auf das Bereitstellen unabhängiger REST-Services beschränkt, während die eigentliche Applikation komplett im Browser läuft.

Abbildung 1 zeigt schematisch eine Beispiel-Architektur. Der Browser lädt die Web-Anwendung etwa als HTML5-App von einem statischen Web-Server. Im Browser kann die Web-Anwendung sowohl mit dem „REST-Service A“ als auch mit dem „REST-Service B“ kommunizieren.

Einschränkungen durch die Same-Origin-Policy

Bei der Realisierung einer solchen Architektur gibt es eine fundamentale Einschränkung: Die Same-Origin-Policy im Browser verbietet es der JavaScript-Anwendung, Inhalte zu lesen oder zu ändern, die von einem anderen Server stammen. Auch die Ajax-Requests unterliegen der Same-Origin-Policy, der Browser erlaubt Ajax-Requests also nur an den Server, von dem die Seite ursprünglich geladen wurde.

Ziel der Same-Origin-Policy ist es, das Ausspähen von Informationen zu verhindern. Zum Beispiel wird unterbunden, dass eine Web-Seite im Hintergrund einen Request an facebook.com schickt, um dort Informationen über den User abzufragen. Die Same-Origin-Policy verhindert aber auch die browserseitige Integration verschiedener REST-Services in einer Anwendung. Um dies in gewünschten Fällen trotzdem zu ermöglichen, hat das W3C das Cross-Origin-Resource-Sharing vorgeschlagen.

Cross-Origin-Resource-Sharing

Cross-Origin-Resource-Sharing (CORS) definiert eine Reihe von neuen Headern im HTTP-Protokoll, die es dem Server ermöglichen, gezielt REST-Aufrufe für bestimmte Ursprungs-Domains zuzulassen. Die Idee hinter CORS ist, dass der Browser vor dem eigentlichen REST-Aufruf in einem Preflight-Request abfragt, ob der Zielserver die Ursprungs-Domain der Seite als ver-

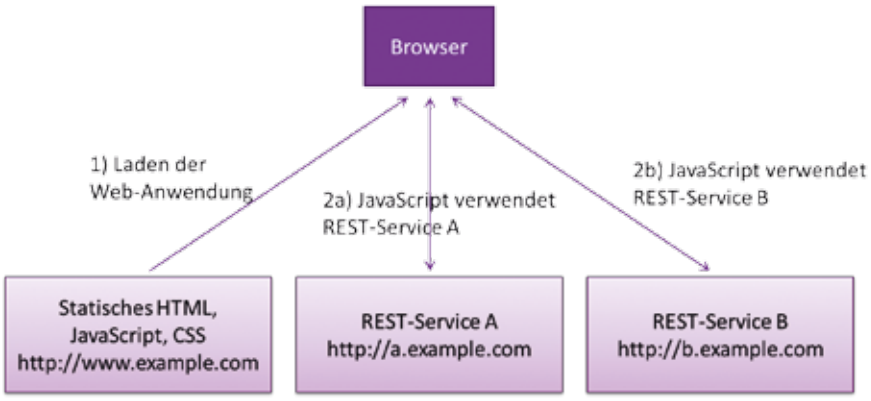


Abbildung 1: Architektur-Beispiel

```
OPTIONS /rest/login HTTP/1.1
Access-Control-Request-Method: POST
Origin: http://www.example.com

HTTP/1.1 200 OK
Access-Control-Allow-Origin: http://www.example.com
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET, POST, OPTIONS
Access-Control-Allow-Headers: Content-Type
```

Listing 1

```
POST /rest/login HTTP/1.1

HTTP/1.1 200 OK
Access-Control-Allow-Origin: http://www.example.com
Access-Control-Allow-Credentials: true
Set-Cookie: JSESSIONID=e3c3e1f965fce0cdb0b794cfbd02
```

Listing 2

```
$.ajax({
    url: "http://a.example.com/rest/login",
    type: 'POST',
    xhrFields: { withCredentials: true },
    success: function (data, textStatus, jqXHR) {
        // …
    }
    error: function (jqXHR, textStatus, errorThrown) {
        //…
});
```

Listing 3

```
@Override
public void doFilter(…) {
    HttpServletResponse resp = (HttpServletResponse) response;
    resp.setHeader(
        "Access-Control-Allow-Origin",
        "http://www.example.com");
    resp.setHeader(
        "Access-Control-Allow-Credentials",
        "true");
    chain.doFilter(req, resp);
}
```

Listing 4

trauenswürdig akzeptiert. Erst dann kann die JavaScript-Anwendung auf den REST-Service zugreifen. Neben den bekannten HTTP-Requests „GET", „POST", „PUT" und „DELETE" gibt es im HTTP-Protokoll den OPTIONS-Request. Dieser wird beim Cross-Origin-Resource-Sharing genutzt, um den Preflight-Request zu implementieren.

Listing 1 zeigt ein Beispiel für einen Preflight-Request. Der Browser sendet den OPTIONS-Request an den REST-Service „http://a.example.com/rest/login", um zu klären, ob POST-Requests von der Ursprungs-Domain „http://www.example.com" zulässig sind. Der Server bestätigt die Zulässigkeit mit dem Status „200 OK" und den angegebenen Access-Control-Allow-Headern.

Nach dem Preflight-Request findet im zweiten Schritt der eigentliche POST-Request statt. Wie in Listing 2 dargestellt, beinhaltet die Antwort des Servers die zusätzlichen Access-Control-Allow-Header, damit der Browser die Antwort akzeptiert. Nur wenn diese Header enthalten sind, kann das JavaScript-Programm die Antwort des Servers lesen.

Ein wichtiges Feature von Cross-Origin-Resource-Sharing ist, dass der Browser zu jedem Server eine eigenständige, Cookie-basierte HTTP-Session aufbauen kann. Um dies zuzulassen, muss der Server, wie in Listing 2 zu sehen, den Access-Control-Allow-Credentials-Header setzen.

Implementierung

Browserseitig ist die Implementierung fast ohne Mehraufwand möglich, da der Ajax-Aufruf in jQuery bereits Cross-Origin-Resource-Sharing unterstützt. Wie in Listing 3 gezeigt, muss lediglich die zusätzliche Option „withCredentials: true" angegeben werden, um Cookie-basierte HTTP-Sessions zu nutzen.

Serverseitig muss dafür gesorgt sein, dass in jeder Antwort die Access-Control-Allow-Header enthalten sind. Im Servlet-Container kann dies zum Beispiel erreicht werden, indem ein passender Servlet-Filter implementiert und in der „web.xml" angegeben wird. Listing 4 zeigt die entsprechende „doFilter()"-Methode.

Zusätzlich muss der Preflight-Request beantwortet werden. Wenn der REST-Service mit JAX-RS implementiert ist, kann man dazu einfach einen zusätzlichen

Handler für den OPTIONS-Request implementieren (siehe Listing 5).

```
@OPTIONS
public Response handleCorsPreflight(…) {
    return Response.ok()
        .header("Access-Control-Allow-Methods",
                "GET, POST, OPTIONS")
        .header("Access-Control-Allow-Headers",
                "Content-Type")
        .build();
}
```

Listing 5

Fazit

Mit Cross-Origin-Resource-Sharing hat das W3C einen Mechanismus vorgeschlagen, der die browserseitige Integration von REST-Services ermöglicht. Sowohl aufseiten des Browsers als auch serverseitig ist die Implementierung von Cross-Origin-Resource-Sharing mit wenig Aufwand möglich. Im Gegensatz zu bisher möglichen Workarounds wie JSON-P ermöglicht Cross-Origin-Resource-Sharing echte REST-Aufrufe, wobei auch Cookie-basierte HTTP-Sessions möglich werden.

Cross-Origin-Resource-Sharing ist eine neue Technologie. Obwohl aktuelle Browser die Technologie unterstützen, sollte sie im Moment noch vorsichtig eingesetzt werden, da die Gefahr besteht, User mit älteren Browsern auszuschließen. Wer heute Cross-Origin-Resource-Sharing in einer Anwendung nutzen will, sollte einen Fallback-Mechanismus bereithalten für User, bei denen das Protokoll noch nicht unterstützt wird. In der Praxis wird Cross-Origin-Resource-Sharing zum Beispiel in den NorCom-Produkten NCPower und NCSpace erfolgreich eingesetzt.

Weitere Informationen

http://www.w3.org/TR/cors/

Dr. Fabian Stäber
fst@norcom.de

Dr. Fabian Stäber ist Software-Entwickler im Produktbereich der NorCom Information Technology AG. Sein Schwerpunkt sind JEE-Serveranwendungen.

„Java ist so etwas wie ein guter Kollege, auf den ich mich (zumeist) verlassen kann …“

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

Wie bist du zur JUG Ostfalen gekommen?
Uwe Sauerbrei: Nachdem ich ein paar Veranstaltungen der JUG HH (Hamburg) besucht hatte, kam mir die Idee, dass so etwas auch in Braunschweig/Wolfsburg möglich sein sollte. Gesagt, getan, also streckte ich meine Fühler aus, sammelte Kontakte, suchte Gleichgesinnte und gründete Anfang 2010 die JUG Ostfalen.

Wie ist die JUG Ostfalen organisiert?
Uwe Sauerbrei: Wir haben keine feste Organisations-Struktur. Es gibt eine Gruppe bei Xing, über Google+ und Twitter sind wir ebenfalls zu erreichen. Unsere Webseite informiert über die aktuellen Planungen und berichtet über durchgeführte Veranstaltungen. Wir sind Mitglied des iJUG und ich vertrete unsere Interessen beim International Software Architecture Qualification Board (iSAQB).

Was zeichnet die JUG Ostfalen aus?
Uwe Sauerbrei: Das Thema „Java“ ist nur der Aufhänger für unsere Veranstaltungen, die sich auch um die „Qualitäten“ eines Entwicklers drehen, einen Blick auf Android und Big Data erlauben und uns gelegentlich in das Wohnzimmer von Arno Haase führen. Bei uns treffen sich Studenten, Professoren und Experten aus der Industrie zu Vorträgen, Coding Dojos und Workshops. Man kennt sich inzwischen und das betrachte ich als unseren größten Erfolg.

Wie viele Veranstaltungen gibt es pro Jahr?
Uwe Sauerbrei: Unser Ziel ist es, monatlich mindestens eine Veranstaltung durchzuführen. Hier kooperieren wir gerne mit den mittelständischen Unternehmen der Region oder finden uns im Haus der Wissenschaft in Braunschweig zusammen.

Zur Person: Uwe Sauerbrei

Nach dem Studium der Geophysik an der Bergakademie Freiberg entwickelte er gleich im ersten Projekt eine Software für die Erdöl-Exploration. Lagen die Anfänge noch in C/C++ und Unix, wechselte der Schwerpunkt später in den Bereich „(Enterprise) Java". Er arbeitete als Berater/Entwickler in den Branchen Telekommunikation, Automotive, Logistik und Luftfahrt. 2006 gründete er die „emmert CONSULTING + partner".

Uwe Sauerbrei, Vorsitzender der Java User Group (JUG) Ostfalen

Was motiviert dich besonders, als Vorstand die JUG Ostfalen zu führen?

Uwe Sauerbrei: Es sind die vielen neuen Kontakte und die Möglichkeit, aus berufenem Munde über spannende Themen hautnah informiert zu werden. Es ist schön zu erleben, wie sich eine Vortragsidee in einer Veranstaltung manifestiert und Leute bereit sind, ihre Freizeit zu investieren.

Was bedeutet Java für dich?

Uwe Sauerbrei: Java begleitet mich seit den neunziger Jahren und ist so etwas wie ein guter Kollege, auf den ich mich (zumeist) verlassen kann.

Welchen Stellenwert besitzt die Java-Community für dich?

Uwe Sauerbrei: Es ist schön, wenn ich bei Konferenzen, Workshops oder Vorträgen immer öfter bekannte Gesichter sehe. Viele meiner Referenten habe ich auf diese Weise kennengelernt.

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

Uwe Sauerbrei: Sun und Oracle sind zwei völlig gegensätzliche Unternehmen. Ich war gespannt und skeptisch zugleich, wie das funktionieren sollte. Zu Beginn sah ich mich in meinem Zweifel bestätigt, aber inzwischen sehe ich auch viele Vorteile, die diese Übernahme mit sich gebracht hat.

Wie sollte sich Java weiterentwickeln?

Uwe Sauerbrei: In einer Zeit, in der JavaScript massiv an Bedeutung gewinnt und Web-Technologien sich immer schneller entwickeln, muss auch Java flexibler werden. Die Anwendungen werden immer mobiler, die Daten wandern in die Cloud und hier gewinnt, wer die richtigen Werkzeuge anbietet. Das Ziel ist also klar.

Wie sollte Oracle mit Java umgehen?

Uwe Sauerbrei: Java ist nicht nur eine Programmiersprache. Es ist eine Kultur und das hat Oracle, nach anfänglichen Schwierigkeiten, nun wohl verstanden. Es ist ein Gemeinschaftsprojekt vieler Entwickler und Organisationen, einer einzigartigen Community und der Industrie. Oracle ist gut beraten, hier in erster Linie zu koordinieren, zu moderieren und den „Papierkram" zu erledigen.

Wie sollte sich die Community gegenüber Oracle verhalten?

Uwe Sauerbrei: Oracle ist kein Abstraktum. Es sind Entwickler und Manager, die sich mit ganz konkreten Plänen und Zielen beschäftigen. Man kann mit ihnen sprechen, sich austauschen und zusammenarbeiten. Dafür gibt es viele Kanäle und diese sollte die Community nutzen. Eine bessere Möglichkeit, an der Zukunft von Java mitzuarbeiten, gibt es nicht.

Uwe Sauerbrei
info@jug-ostfalen.de

Oracle gegen Google: APIs sind nicht urheberrechtlich geschützt

Für Oracle ist es eine schmerzhafte Niederlage, für die IT-Branche könnte sich der Prozess als wichtiger Präzedenzfall erweisen: Richter William Alsup hat entschieden, dass APIs nicht urheberrechtlich geschützt sind.

Oracle hatte Google unter anderem vorgeworfen, mit dem Betriebssystem Android die Urheberrechte an 37 APIs verletzt zu haben. Nun sieht es der Richter anders. Für Alsup seien die Software-Schnittstellen von Java Funktionselemente, die das Zusammenspiel von Programmen ermöglichen. Aus diesem Grund sei es nicht möglich, diese urheberrechtlich zu schützen. Einziger Verletzungsfall, den die Geschworenen während des Prozesses anerkannt haben: Neun Zeilen Softwarecode, den Google bereits aus aktuellen Android-Version entfernt hat. Der Softwareriese, der eine Schadenersatzsumme von einer Milliarde Dollar gefordert hatte, geht leer aus. Oracle hat jedoch angekündigt, in Berufung gehen zu wollen.

www.ijug.eu

JETZT ABO BESTELLEN

Sichern Sie sich 4 Ausgaben für 18 EUR

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

FAXEN SIE DAS AUSGEFÜLLTE FORMULAR AN
0700 11 36 24 39

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

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

Java aktuell

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

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

☐ **Ja**, ich bestelle den kostenfreien Newsletter: Java aktuell – der iJUG-Newsletter

ANSCHRIFT

Name, Vorname

Firma

Abteilung

Straße, Hausnummer

PLZ, Ort

GGF. RECHNUNGSANSCHRIFT

Straße, Hausnummer

PLZ, Ort

E-Mail

Telefonnummer

X

Die allgemeinen Geschäftsbedingungen* erkenne ich an, Datum, Unterschrift

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