Jenseits des Tellerrands, Teil 8: Java-Bashing

Ist die Java-Plattform zu komplex geworden?

VON EBERHARD WOLFF, HENNING WOLF UND STEFAN ROOCK

Softwareentwicklung ist weit mehr als Programmierung und Programmierung ist weit mehr als das bloße Beherrschen von Programmiersprachen und Werkzeugen. Die Artikel dieser Serie greifen wichtige Themen, aktuelle Fragen und auch grundsätzliche Probleme auf, beleuchten diese und stellen „die gängigen Antworten" immer wieder in Frage. Subjektivität ist dabei keineswegs verpönt, sondern das notwendige Salz in der technischen Einheitssuppe.

Manchmal beschleicht mich das Gefühl, trotz der jahrelangen Beschäftigung mit Java immer weniger über die Plattform zu wissen. Kaum ein Tag vergeht, ohne dass ich nicht über ein neues API oder einen neuen JSR stolpere, dessen Verwendung (und manchmal auch dessen Sinn) mir ein völliges Rätsel ist. Fand ich nicht damals Java gerade deswegen gut, weil es viel einfacher zu beherrschen schien als C++? Ein Streitgespräch zwischen den beiden fiktiven, aber hochgradig renommierten Java-Kennern J. Avanie und S.P. Ring geht der Frage nach, ob Java nicht im Begriff ist, die Rollen von C++ und Cobol als übermäßig komplizierte Sprache bzw. überalterte Legacy-Plattform zu übernehmen.

Johannes Link
(john.link@gmx.net)

J. Avanie: Warum ich heute so kritisch auf Java blicke? Java ist als Programmiersprache und Plattform viel zu kompliziert geworden. Wir brauchen eine neue Sprache und eine neue Plattform. Die Java-Systeme von heute sind nicht wirklich besser wartbar als die alten Cobol-Systeme. Haben wir mit Java nicht heute schon massenhaft Legacy-Systeme, die in Relation zu den Cobol-Anwendungen noch total jung sind?

Meiner Meinung nach schon. Das liegt daran, dass Java so kompliziert ist, sodass die Entwickler viel Energie in die Technologie stecken müssen und kaum noch Kraft für Fachlichkeit und vernünftige Architektur übrig bleibt.

S.P. Ring: Dann sollten wir wohl wieder Cobol machen? Ist das die Lösung? Gibt es wirklich einen Vorteil bei Cobol? Wie kann es sein, dass man mit Cobol anscheinend oder auch nur angeblich besser zurechtkommt? Und was kann man daraus lernen? Meiner Meinung nach sind die Vorteile von Cobol seine Beschränktheit und sein Alter. Es gibt für Cobol erprobte Best Practices, die nicht alle paar Monate komplett in Frage gestellt werden. Es gibt nur wenige Technologien, die man beherrschen muss. Dafür muss man mehr Code tippen, wenn man keinen Generator verwendet, und kann auch nur eine Art von Anwendungen damit bauen – wenn auch eine sehr verbreitete Art: zentralistische Struktur mit nur einem Server, Daten schubsen, ein wenig Fachlogik, die nur auf dem Server liegt, und ein schwach interaktives UI. Diese Beschränkungen wollte man ja gerade loswerden.

Ich denke, bei vielen Projekten wird zu viel auf einmal gemacht. Unternehmen, die jahrzehntelang nur Cobol gemacht haben, machen auf einen Schlag alles anders: neue objektorientierte Denkweise, neue Programmiersprache, neue Plattform und neues Vorgehensmodell. Da ist doch sonnenklar, dass dies nicht zu schaffen ist; auch nicht mit vielen neuen Ent-

Jenseits des Tellerrands

Was bisher geschah und Sie zukünftig erwartet

- Teil 1: Das nächste Java?
- Teil 2: Typisierung in Java und darüber hinaus
- Teil 3: Warum gibt es eigentlich Softwareentwicklungsprojekte?
- Teil 4: Softwarearchitektur: eine kritische Bestandsaufnahme
- Teil 5: Textuelle domänenspezifische Sprachen
- Teil 6: Moderne System- und Abnahmetests
- Teil 7: Warum nutzen wir die Mittel zur Softwareverbesserung nicht, obwohl wir sie kennen? Eine Diskussion
- **Teil 8: Java-Bashing: Unsere liebste Plattform ist zu kompliziert geworden**
- Teil 9: Rich Client vs. Web Client
- Teil 10: Wie viel Abstraktion brauchen wir?
- Teil 11: Berufsethos, Qualität, Rollenverständnis
- Teil 12: Time's Arrow: ein Projekt von hinten

wicklern. Außerdem stehen die Projekte unter massivem Zeitdruck, sodass sich gute Architekturen, die bei Cobol zehn Jahre oder mehr gekostet haben, gar nicht bilden können.

J. Avanie: Klar, die neuen Möglichkeiten von Java sind im Gegensatz zu Cobol phantastisch. Aber sowohl Unternehmen als auch Entwickler werden durch die schier unendlichen Möglichkeiten von Java zu Allmachtsphantasien verleitet. Zudem sind die Systeme, die wir gerne bauen würden, viel komplexer als die Systeme, die wir bauen können. Wir haben in den letzten Jahrzehnten unsere Software-Engineering-Fähigkeiten um den Faktor 10 oder vielleicht 100 steigern können, aber unsere Wünsche sind um mindestens eine Größenordnung stärker gewachsen. Wir sehen unsere eigenen Beschränkungen nicht.

Es wird eben immer von irgendjemandem das gemacht, was gerade machbar ist. Das versuchen dann viele Leute zu kopieren – und das Unheil nimmt seinen Lauf. Kaum ein Projekt meint, noch ohne Web Service, Rich Clients, Web Interface, lineare Skalierbarkeit, schnelle Antwortzeiten und so weiter auskommen zu können. Natürlich brauchen sie dafür alle möglichen Technologien und öffnen damit die Büchse der Pandora. Von Cobol erwartet so was niemand. Häufig wurden in den Legacy-Systemen nicht mal eine richtige Datenbank eingesetzt, sondern einfach indexsequenzielle Dateien.

Wenn wir uns wieder auf die Programmiersprachen und Plattformen beschränken würden, die einen spezifischen Einsatzkontext hätten, wäre einiges zu gewinnen. Ruby on Rails (RoR) [1] macht das gerade vor. Damit kann man eigentlich nur einen Typ von Webanwendungen gut schreiben – aber für diesen Anwendungstyp klappt das eben ganz hervorragend. Für diesen Typ von Webanwendungen braucht man weder Verteilung à la Web Services oder EJBs noch verteilte Transaktionen oder hochtrabende Security-Konzepte.

S. P. Ring: Na ja, aber es ist ja niemand gezwungen, all diese Features der Java-Plattform zu nutzen. Man kann sich auch bei Java auf eine einfache Teilmenge des Machbaren konzentrieren und damit ganz einfach Software schreiben. Das konnte man eigentlich bereits mit dem JDK 1.1 recht gut, ohne das ganze Java EE-Zeug. Und auf diese Teilmenge kann man sich immer noch zurückziehen. Ich sehe das Problem daher nicht in Sprache oder Plattform, sondern in der Psychologie. Entwickler glauben immer, dass sie Technologie XYZ unbedingt auch noch einsetzen müssen, um cool zu sein.

J. Avanie: Auch das Finden des passenden Subsets kostet viel Zeit und Nerven. Es wäre deutlich einfacher für uns alle, wenn sich Leute bei Sun oder wo auch immer mit Sinn und Verstand hinsetzen und Subsets der Technologien definieren würden, die für bestimmte Anwendungstypen sinnvoll sind. Bisher müssen die Projekte das selber leisten und ich habe schon einige Projekte gesehen, die Jahre darauf verwendet haben, die passenden Java-Technologien zu definieren, ohne auch nur eine Zeile sinnvollen Code zu schreiben. Sie wurden nämlich bei ihrer Evaluation immer wieder von der Technologieentwicklung überholt. Kaum waren sie der Meinung, dass Hibernate der geeignete OR-Mapper sei, wurde JDO bekannter und sie sahen sich genötigt, auch noch JDO zu evaluieren. Dieses Problem wird dadurch verstärkt, dass es unzählige Open-Source-Werkzeuge gibt; und in einigen Bereichen, wie zum Beispiel bei Web-Frameworks, weiß keiner mehr, was alles existiert und was man wann verwenden sollte.

S. P. Ring: Ja, das ist ein Problem, das durch die Java-Standardisierung nicht

Die Diskutanten

J. Avanie promovierte an der Universität von Maui und hat jahrelange Erfahrung mit (zu) großen Java-Projekten. Dort lernte er, wie EJB aus gestandenen Entwicklern ein wimmerndes Häufchen Elend machte. In Beratungssituationen erlebte er immer wieder, dass Entwicklerteams millionenschwere Entwicklungsbudgets für Technologieevaluationen verbrauchten, ohne auch nur eine Zeile fachlichen Code geschrieben zu haben.

S.P. Ring studierte an der Universität von Java und ist einer der führenden Java-Reformatoren. Er setzt sich seit Jahren für grundlegende Änderungen in der Java-Plattform ein und hat bei der Konzeption und Entwicklung neuerer Java-Technologien mitgewirkt.

unbedingt verbessert wird, weil viele Standards auch nicht unbedingt die beste Lösung sind, sondern lediglich ein Kompromiss der beteiligten Hersteller und Lobbyisten. Daher gefallen mir Ansätze wie Spring, bei denen auch gleich ein Set von anderen Frameworks mitgeliefert und integriert wird. Spring [2] ist eben auch eine Technologie-Toolbox für die wichtigsten Programmierprobleme. Diese Idee steckt auch hinter AppFuse [3] und Equinox [4] und ist sicherlich einer der Erfolgsfaktoren von Ruby on Rails.

J. Avanie: Wer oder was sind denn AppFuse und Equinox?

S. P. Ring: Das sind Projektskelette für Webanwendungen, in die man verschiedene Technologien integrieren kann. Ähnliche Ansätze verfolgen auch die Java EE BluePrints oder struts-blank aus dem Struts-Projekt.

J. Avanie: Dennoch: Man könnte fast vermuten, dass, sobald jemand herausfindet, wozu Java gut ist und wie man es richtig benutzt, es durch etwas noch Merkwürdigeres und Komplexeres ersetzt wird. Einige Leute behaupten, das sei bereits geschehen – und zwar mit Java 5.

S. P. Ring: Gilt das nicht für jede Technologie? Java hat außerdem eine wahnsinnige Verbreitung erreicht. Vor allem ist es das erste Mal, dass sich Firmen wie IBM, Sun, SAP, BEA, Oracle usw. auf einen Standard geeinigt haben und ihn unterstützen. Auf dieser Ebene gibt es schlicht keine Alternative. Java 5 ist gerade als Reaktion auf Schwächen in Java entwickelt worden. Generics zum Beispiel lösen real existierende Probleme.

J. Avanie: Aber Java löst seine Probleme, indem weitere Komplexität hinzugefügt wird wie z.B. durch Generics und Annotations. In Groovy oder Ruby sind die Probleme anders gelöst – dafür war keine zusätzliche Technologie notwendig. Zudem sind Generics und Annotations aus meiner Sicht Nice-to-Have-Features, die das eigentliche Problem nicht lösen: die Komplexität der Sprache und der Java-Plattform. Ich kenne einen Fall in einer Versicherung, in der seit Jahrzehnten eine Cobol-Anwendung für die Kfz-Sparte von einem einzigen Entwickler erfolgreich betreut wird. Jetzt ist die Versicherung seit fünf Jahren mit 30 Mann dabei, die gleiche Anwendung mit Java neu zu implementieren und ein echtes Ende ist immer noch nicht in Sicht. Aus meiner Beratungspraxis kann ich berichten, dass das kein Einzelfall ist. Ich führe das auf die Komplexität zurück.

S. P. Ring: Na ja, die erste Frage ist ja, ob nicht auch hier wieder die Gründe für den Fehlschlag des Projektes nicht eher im nichttechnischen Bereich zu suchen sind. Technologien können zwar auch zum Scheitern eines Projekts führen, aber oft sind es andere Gründe wie unklare Anforderungen oder firmeninterne Streitigkeiten. Darüber hinaus gibt es Ansätze zur Reduzierung der Komplexität: Spring zum Beispiel geht auch hier in die richtige Richtung; es reduziert die Komplexität, indem es die Java-Plattform so einkapselt, dass die prinzipielle Funktionalität angeboten wird, aber hinter handhabbaren Schnittstellen.

Bei Generics denke ich sogar, dass eine Programmiersprache mit statischer Typisierung ohne ein solches Konzept keinen Sinn macht, weil man sonst doch überall bei Collections usw. Typumwandlungen mit potenziellen Fehlern hat. Annotations haben schon in der .NET-Welt gezeigt, dass sie die Entwicklung vereinfachen. Ansätze wie Java EE 5 oder Spring zeigen, was man mit diesen Möglichkeiten alles erreichen kann.

J. Avanie: Dennoch wird Java als Ansatz immer komplexer. Die Anzahl der Klassen im JDK wächst kontinuierlich, die Anzahl der APIs und Open-Source-Projekte steigt rasant – und dieser Trend wird sich auch nicht umkehren, da man die alten Sachen nicht einfach wegwerfen kann. Auch eine Abstraktion wie Spring kann man oft erst verstehen, wenn man die Basis vorher verstanden hat.

S.P. Ring: Ja, das stimmt. Allerdings steigt durch die Anzahl der APIs auch der mögliche Einsatzbereich von Java …

J. Avanie: … aber mehr als 80 Prozent der Java-Projekte benötigen weniger als 20 Prozent der Java-Features …

S.P. Ring: … und diese Projekte sollten dann unter der Komplexität nicht leiden, oder?

J. Avanie: Genau. Aber die Projekte leiden häufig gerade unter der Komplexität, die sie eigentlich gar nicht benötigen. So müssen die allerwenigsten Projekte irgendwas im Bereich Class Loading machen. Dennoch brauchen Entwickler ziemlich genaues Wissen über Class Loading, wenn sie z.B. mit EARs arbeiten. Ansonsten sind für sie die Fehlermeldungen bei falsch zusammengepackten EARs völlig unverständlich.

Enterprise JavaBeans

J. Avanie: Neben dem höheren Einarbeitungsaufwand muss man die Komplexität ständig beherrschen. Dazu gab es mal im „Informatik Spektrum" zu Zeiten der Java-Anfänge eine Studie über die Performance von Java im Vergleich zu C und C++. Die schnellsten C- und C++-Programme waren ein ganzes Stück schneller als die Java-Programme. Allerdings waren die Java-Programme im Schnitt genauso schnell wie die C++-Programme. Das wurde darauf zurückgeführt, dass die Entwickler wegen der Komplexität von C++ die Programme so „vollmüllen", dass sie dann doch langsamer sind, als hätten dieselben Leute sie mit Java geschrieben.

Man denke nur mal an Entity Beans vor EJB 3 und die ganzen Artefakte, die man für EJBs entwickeln muss; hinzu kommt die einfach unzureichende Unterstützung für Sicherheit und so weiter. Frei nach Douglas Adams könnte man sagen: „EJB wurde nicht entworfen, sondern tiefgefroren. Es ist schon Hässlicheres erblickt worden, aber nicht von verlässlichen Augenzeugen."

Ich habe da meine eigene Theorie. Und zwar glaube ich, dass die Superschurken mit Weltbeherrschungsambitionen aus den frühen James-Bond-Filmen inzwischen in irgendwelchen Gremien sitzen und sich so etwas wie EJB 2 ausdenken. Aber es gab einen deutlichen Wandel von EJB 2 zu EJB 3: Die größten EJB-Probleme werden mit EJB 3 behoben – abgesehen vielleicht vom Sicherheitsaspekt. Der Übergang von EJB 2 zu EJB 3 ist auch insofern interessant, als dass sich die Java-Plattform tatsächlich mal in Richtung Einfachheit weiterentwickelt. Und natürlich kann man Java und Java EE gut auch ohne EJB verwenden.

S.P. Ring: Na, na, das ist doch reichlich polemisch. Die Leute haben sich bei EJB schon was gedacht. EJB ist beispielswei-

se ein deutlicher Fortschritt gegenüber CORBA. Ich glaube, dass die Probleme auf einer anderen Ebene liegen: Die Standardisierung dauert zu lange: Seit zwei Jahren gibt es EJB 3-Entwürfe, aber immer noch keinen Standard. Die Standards werden erst mal komplett fertig gestellt, bevor man Erfahrungen mit ihnen sammeln darf. Das Ergebnis sind langsame Iterationen und spätes Feedback, während uns die agile Softwareentwicklung lehrt, dass das keine guten Voraussetzungen für einen schnellen Erfolg sind. Dennoch gibt es sehr wertvolle Standards wie Servlets, die es erlauben, auf jedem Webserver jedes Web-Framework zu verwenden. An anderen Stellen – wie z.B. EJB 1 oder EJB 2 – werden dann suboptimale Lösungen hervorgebracht. Aber man darf nicht vergessen: EJB ist eben nur ein Standard. Es gibt genügend andere Standards – auch in Java EE –, die einfach und elegant sind.

J. Avanie: Aber es sind auch viele Dinge kompliziert, die einfach sein sollten: Doppelklick in Swing-Listen, Scroll Bars um Listen, Text Areas, Streams, Einladen von Bildern oder Web Services vor JSR 181. JDBC ist sogar kompletter Müll, weil man damit viel zu leicht Fehler machen kann. Das zieht sich bis in Bereiche, die trivial sein sollten. Warum gibt es zwei Date-Klassen und dann auch noch Calendar?

S.P. Ring: Das heißt, die Funktionalität der Java-Plattform ist im Prinzip nützlich – vielleicht einmal abgesehen von ein paar Ausnahmen wie Entity Beans. Nur mit der Benutzbarkeit hapert es. Spring hilft da für die Java EE APIs. Vielleicht braucht man mehr Kapseln um das Gesamt-API, die jeweils zugeschnitten sind auf spezielle Einsatzbereiche. Dann wären die Java-APIs eine Art Assembler-Sprache, mit der man sich normalerweise gar nicht mehr auseinander setzt. Stattdessen benutzt man darauf aufsetzende APIs im Sinne von Hochsprachen.

Was tun?

J. Avanie: Aber was kann man Managern und Entwicklern raten, um mit den genannten Problemen umzugehen? Zurück zu Cobol will ich eigentlich nicht, weil Cobol in seinen Sprachkonstrukten und seiner Plattform fast allem unterlegen ist, das es sonst noch so am Markt gibt. Sollten die Projekte auf andere Sprachen bzw. Plattformen ausweichen, z.B. Webanwendungen nur noch mit Ruby on Rails oder Grails [5] und der Java-Plattform bauen?

S.P. Ring: Das wirft für mich die Frage auf, ob die Komplexität von Java schon derartig groß ist, dass man mit einem komplett neuen Ansatz besser fährt. Oder kann man mit der Komplexität von Java heute noch sinnvoll umgehen?

J. Avanie: Na ja, der Wechsel der Programmiersprache oder Plattform liegt meist nicht im Ermessensspielraum der Entwickler, sondern ist strategisch vorgegeben. Ein kompletter Umstieg mag eine reizvolle Utopie sein, den meisten Projekten ist mit dieser Utopie leider nicht gedient. Sie brauchen eine Handreiche, um mit der aktuellen Situation in konkreten Projekten umzugehen.

S.P. Ring: Nun ja, da gibt es Ansätze. Wenn man Spring anschaut, geht es schon recht weit darin, über viele APIs ein abstrakteres Programmiermodell zu legen. Dadurch ist also die Komplexität und Problematik auf der API-Ebene prinzipiell beherrschbar. Darüber hinaus gibt es andere Sprachen neben Java, die auch auf der JVM laufen.

J. Avanie: Die konnten sich jedoch bisher nie behaupten. Gute Chancen räume ich Groovy [6] ein, weil Groovy sehr gut mit Java integriert und ein standardisierter Bestandteil der Java-Plattform ist. Das könnte ein Weg sein, wie Projekte schrittweise oder in Teilbereichen Groovy einsetzen, um sich das Leben zu erleichtern.

S.P. Ring: Das ist ein wichtiger Punkt. Die Sprache Java an sich ist ersetzbar. Ein wesentlicher Vorteil von Java ist, dass es eine breite Auswahl an hoch optimierten JVMs gibt, die in vielen Unternehmen auch schon etabliert sind und darauf aufbauend Lösungen wie Application Server, die eine standardisierte Ablaufumgebung definieren und Monitoring und Administration entscheidend vereinfachen.

Das fehlt anderen Sprachen. Daher kann die JVM in Zukunft eine attraktive Basis für Skriptsprachen sein. Das bedeutet, dass Java als Sprache und auch die APIs austauschbar sind und das wesentliche Element die JVM übrig bleibt.

Abschluss

J. Avanie: Ich möchte den Java-Projekten Folgendes mit auf den Weg geben:

- Ihr müsst nicht alles können. Beschränkt euch bewusst auf einen definierten Satz von Technologien – euer Handwerkszeug. Es ist meist wichtiger, sich für eine Lösung zu entscheiden, als die „beste" Lösung zu finden.
- Sucht kontinuierlich nach Möglichkeiten, die Programmierung zu *vereinfachen*. Nicht jedes Problem, das eine Technologie löst, muss man wirklich in seinem Projekt lösen.
- Kümmert euch zunächst um die fachlichen Aspekte wie funktionale Anforderungen, Domänenentwurf und die fachliche Gesamtarchitektur. Sucht ausgehend davon nach den einfachsten Technologien, die bei der Realisierung der Anforderungen helfen.

S.P. Ring: Da möchte ich ergänzen:

- Hinterfragt jede neue Technologie auch und vor allem, wenn sie ein Java-Standard ist. Seid euch immer klar darüber, welches Problem eine Technologie löst. Wenn das nicht klar ist, lasst die Finger von der Technologie.
- Ansätze wie Groovy, Ruby on Rails oder Spring können interessant sein. Aber: Auch das sind lediglich Werkzeuge, um Probleme zu lösen.
- Trennt zwischen der Ablaufumgebung eines Java EE Application Server und dem Programmiermodell. Man kann sehr wohl auf einer standardisierten Java-Enterprise-Plattform andere Modelle – wie zum Beispiel Spring – verwenden.

Danksagung

Die Diskutanten bedanken sich bei Dierk König, Johannes Link, Stefan Roock, Eberhard Wolff und Henning Wolf für interessante und provokative Denkanstöße und für die Aufzeichnung des Gesprächs.

Links & Literatur

[1] rubyonrails.org
[2] www.springframework.org
[3] appfuse.dev.java.net
[4] equinox.dev.java.net
[5] grails.codehaus.org
[6] groovy.codehaus.org