Geschäftstüchtig

Webarchitekturen mit Apache Wicket

Alexander Elsholz, Elena Stoll

Apache Wicket wird zum Mainstream – und immer mehr Firmen springen auf den Zug auf. Die Gründe hierfür sind offensichtlich: Statusbehaftete, komponentenorientierte und ereignisgesteuerte Webprogrammierung ist mit erprobten UI-Mustern in Java möglich und setzt kein Expertenwissen in HTML, CSS oder JavaScript voraus. Der Artikel zeigt aus der Praxis, wie testgetriebene Entwicklung von modernen Weboberflächen mit Wicket möglich ist. Dabei gehen wir auf die Wicket-Kernprinzipien ein und stellen Chancen und Herausforderungen dar, die sich bei der Entwicklung mit dem Webframework – insbesondere im Umfeld von Geschäftsanwendungen – ergeben.

Einleitung

Java-Webframeworks gibt es mittlerweile viele. Dabei etabliert sich Apache Wicket zunehmend als Alternative zu den Platzhirschen wie JSP, JSF, Struts oder Spring MVC. Die Vorteile liegen auf der Hand: Mit Wicket lassen sich in Java ansprechende, leistungsfähige Benutzungsoberflächen umsetzen, die sich gut in die Unternehmensarchitektur integrieren. Ereignisbehandlung, testgetriebene Entwicklung, Komponentenorientierung und Kapselung von technischen sowie fachlichen Aspekten – all dies kann der Entwickler mit Java-Bordmitteln umsetzen.

Das schlanke Webframework Wicket basiert auf einem einfachen LifeCycle-Management und der Manipulation von HTML im Java-Code. Der Entwickler programmiert seine Oberfläche ähnlich wie bei einer Rich-Client-Technologie (Swing, VB, Delphi). Dabei werden UI-Controls zu einer Oberfläche orchestriert und EventHandler für deren Interaktion definiert.

Auf der Basis einer umfangreichen Sammlung an Controls und Frameworks im Apache-Wicket-Umfang sowie der vielen verfügbaren Erweiterungen (Wicket-Stuff-Projekte, bei Google-Code gehostete Erweiterungen usw.) lassen sich so ohne große Kenntnisse in CSS und JavaScript moderne, Ajax-basierte Web 2.0-Anwendungen entwickeln. Dabei behält der Entwickler volle Kontrolle über seinen erstellten Code. Es kann auf Quellcode-Ebene debugged, getestet und refaktorisiert werden. Aufbauend auf die vielen fertigen Controls und APIs lassen sich einfach individuelle Erweiterungen erstellen, um technische Aspekte vom Fachentwickler zu kapseln. Dafür werden keine Taglibs oder ähnliche Konzepte verwendet. Die Erweiterungen werden viel mehr im bekannten Java-Umfeld implementiert.

Auch die Entwicklung von Widgets mit klar definierter Java-Schnittstelle wird vom Framework gut unterstützt. Dies ermöglicht eine komponentenorientierte Entwicklung der Präsentationsschicht. UI-Blackbox-Komponenten werden erstellt, beschrieben und können von anderen Entwicklern in die eigene Oberfläche integriert werden.

Wicket – behind the scenes

Bevor wir auf die Vorteile der Programmierung von modernen, Web 2.0-basierten Geschäftsanwendungen auf Basis von Wicket eingehen, werden im Folgenden die Ziele des Webframeworks und seine grundsätzliche Funktionsweise dargestellt.

Wicket-Ziele

Nachstehende Ziele liegen der Entwicklung des Webframeworks zugrunde [Wicket]:

- *Easy:* Das Framework soll einfach, klar und konsistent sein. Der Entwickler muss nach dem Verständnis der Konzepte in der Lage sein, die Aktionen des Webframeworks nachvollziehen zu können. Dank dieser Transparenz ist es vergleichsweise einfach, das Framework um eigene Abstraktionen zu ergänzen. So können beispielsweise unterschiedlichste JavaScript- oder CSS-Frameworks sehr einfach integriert werden. Auch Wickets Lifecycle ist gut anpassbar. Dies wird zum Beispiel durch das integrierte Testframework deutlich, welches den Standard-Lifecycle ersetzt. Demnach ist die Webapplikation gut in die eigene IT-Architektur integrierbar.
- *Reusable:* Das Framework soll die Wiederverwendung unterstützen. Dabei sollen die Komponenten als Standard-JAR-Dateien verwendet werden können. Die Komponentisierung von fachlichen Frontend-Komponenten ist beispielsweise auf Basis von Maven oder OSGi [WicketOsgi] möglich. Auch eine Anbindung an JSR-286 Portal oder den Widget-Container Shindig ist verfügbar. Die Integration folgt dabei dem Ziel der Einfachheit. Die wiederverwendbaren Komponenten werden über Komposition oder Vererbung in Java genutzt.
- *Non-intrusive:* Das Ziel hier ist, Layout und Logik nicht miteinander zu vermischen. Das Layout soll so wenig wie möglich spezifische Tags enthalten. So ist es möglich, das Layout durch Webdesigner erstellen zu lassen oder durch Tools bzw. Layoutmanager zu generieren. Die Logik soll im Java-Code gekapselt sein. So ist die Webanwendung besser zu warten und es können die gängigen Softwareentwicklungsmethoden verwendet werden: Test Driven Development, Refactoring und Debugging.
- *Safe:* Wicket unterstützt das Entwickeln von sicheren Webanwendungen. Dazu gehört neben der Typsicherheit in der Implementierung (Java) auch die Beschränkung des direkten Zugriffes auf die Wicket-Seiten. So können nur extra gekennzeichnete Seiten (sogenannte BookmarkablePages) direkt über URL angesprochen werden. Auch beim Aspekt Security steht wieder das Ziel der Einfachheit im Fokus. Eine sehr einfache Schnittstelle für die Autorisierung ermöglicht die Integration in Standard-Security-Frameworks wie JAAS oder Acegi bzw. die Adaption in individuelle Autorisierungslösungen. Auch hier existiert eine Reihe von verfügbaren Implementierungen, die direkt nutzbar sind bzw. als Basis für individuelle Lösungen dienen.
- *Efficient/Scalable:* Skalierbarkeit und Performance sollen bei allen Designentscheidungen berücksichtigt, aber nicht auf Kosten der anderen Ziele überproportional gewichtet werden. Interessant beim Aspekt Performance ist die mögliche Nutzung von Standard-Entwicklungstools. Es ist ohne Weiteres möglich, aussagekräftige Lasttests zu implementieren bzw. Profiler für die Analyse von Performance- oder Speicherproblemen zu nutzen. Allgemein können Wicket-Applikationen sehr performant und mit wenig Speicherverbrauch implementiert werden. Komponentenbasierte Frameworks sind aufgrund des am Server vorgehaltenen Komponentenmodells speicherintensiver als klassische actionbasierte Alternativen. Im Vergleich zu JSF schneidet Wicket in Benchmarks deutlich besser ab [WicketPerformance].
- *Complete:* "The Wicket team is committed to deliver a feature complete, ready-to-use framework for developing Java web

applications.“ Ein gutes Beispiel hierfür ist jtrac [JTrac], eine Java-Version des bekannten Projektmanagementwerkzeuges Trac [Trac]. jtrac wurde von JSP auf Wicket umgestellt. Von den zwölf verschiedenen Frameworks, angefangen mit Ajax über Taglibs, Security bis hin zur Page-Inheritance, ist nach der Umstellung nur Wicket übrig geblieben. Dem Fachentwickler stellt sich Wicket als homogenes „ready-to-use“-Framework dar.

Wicket-Programmierung

Wie wird Wicket den hoch gesteckten Zielen gerecht? Wicket wurde nach dem Grundsatz „just java, just html and meaningful abstractions“ entwickelt. Die klare Trennung zwischen Layout und Logik, ergänzt um einfach anzuwendende transparente Abstraktionen, ermöglicht es dem Fachentwickler, sich auf seine Kernaufgabe zu konzentrieren. Er kann die optimale Umsetzung der Kundenanforderungen gewährleisten. Projektindividuelle Querschnittsaufgaben kann der Architekt bzw. Framework-Entwickler gut in Java kapseln.

Doch wie läuft dabei eine Wicket-Anfrage ab? Ruft der Nutzer eine von Wicket verwaltete URL auf, wird der Request vom WicketServletFilter behandelt. Zunächst löst Wicket die URL auf und bestimmt die gesuchte Ressource. Wicket extrahiert die Request-Parameter und gibt diese nach erfolgreicher Typkonvertierung sowie Validierung an das zugehörige Modell. Anschließend werden die EventListener der Controls aufgerufen. Die EventHandler werden üblicherweise durch anonyme innere Klassen (analog zu Swing oder Eclipse-RCP) definiert. Die EventHandler führen die notwendige Geschäftslogik aus und entscheiden, welche Seite bzw. welches Seitenfragment (Ajax) zum Client übertragen werden soll. Das zugehörige WebPage-Objekt der Seite erzeugt für die enthaltenen Controls die Ausgabe. Dabei wird die zugehörige Layoutdefinition in der statischen HTML-Datei als Template genutzt und alle HTML-Tags mit einer Wicket-ID werden durch das generierte Markup des zugehörigen Wicket-Controls (dem Controller) ersetzt.

Man kann Wicket in diesem Bezug durchaus als Template-Engine bezeichnen. Der Entwickler schreibt sein statisches Layout (Template) auf der einen Seite und entwickelt die Dynamik objektorientiert auf der anderen (s. Abb. 1). Wicket übernimmt dabei die Zusammenführung beider Artefakte. Alle Wicket-Controlklassen im Komponentenbaum generieren die dynamischen Inhalte (HTML-Content), die Attribute (HTML oder CSS Attribute) und die Interaktion (JavaScript). Dieser sehr zentrale Aspekt im Wicket-Framework ermöglicht die Kapselung der komplexen Aspekte der Webentwicklung vom Fachanwender.

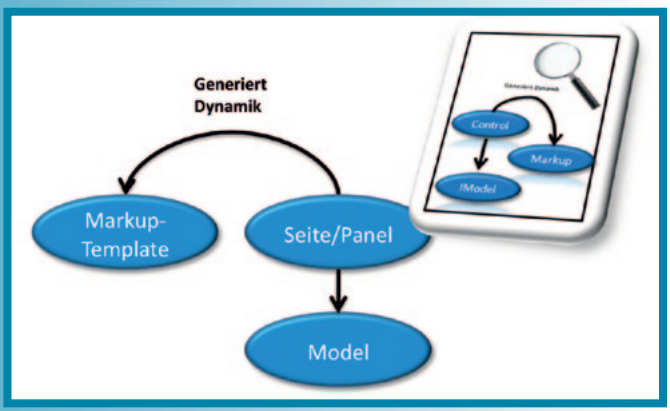

Abb. 1: Komponenten in Wicket

Wicket – Vorteile für Geschäftsanwendungen

Wicket eignet sich sehr gut für die Entwicklung von Web 2.0-basierten Geschäftsanwendungen. Neben der intuitiven, einfach zu bedienenden Oberfläche hat eine Geschäftsanwendung weitere zentrale Anforderungen:

- komponentenorientierte Entwicklung, um parallele Entwicklung und Wartbarkeit zu unterstützen,
- gute Testbarkeit der entwickelten Komponenten, um eine hohe Testabdeckung durch automatisierte Unit-Tests zu ermöglichen,
- Erstellung eigener Abstraktionen, um Wiederverwendung von technischem und fachlichem Code zu erreichen sowie eine einheitliche Darstellung und Funktionsweise der Gesamtanwendung zu forcieren,
- Unterstützung der projektindividuellen Muster zur Erstellung von Frontends, der Integration anderer Frontends und deren Integration in die Gesamtarchitektur.

Auf diese Ansprüche hat Apache Wicket die passenden Antworten, die wir nachfolgend darstellen.

Komponentenorientierte Entwicklung des Frontends

Wicket unterstützt die komponentenorientierte Entwicklung von Webfrontends. Fachlich zusammenhängende Controls werden zu einer UI-Komponente zusammengefasst. Dabei können alle Ressourcen in Form eines Jar-Artefaktes gebündelt werden. Neben dem Controller, der HTML und den Properties-Dateien für die Internationalisierung wird eine fachliche Schnittstelle definiert, die die von anderen Komponenten nutzbaren Funktionalitäten beschreibt. Diese fachlich abgeschlossenen Komponenten (Widgets) werden zu Seiten oder höherwertigen Komponenten orchestriert. Dabei findet die Kommunikation nur über die definierte öffentliche Java-Schnittstelle statt.

Die Vorteile einer komponentenorientierten Frontend-Entwicklung sind:

- Einfach entwickelbare und gut wartbare Komponenten, da überschaubare, fachlich abgeschlossene, voneinander entkoppelte Einheiten entwickelt werden.
- Eine parallele, unabhängige Entwicklung gegen Mock-Implementierungen ist möglich. Dabei kann gegen die öffentliche Schnittstelle der genutzten Komponenten gearbeitet werden.
- Wiederverwendbarkeit der Komponenten.

Wiederverwendbarkeit wird dabei nicht als zentrales Argument für die UI-Komponentisierung gesehen, sondern für die Entkopplung und damit einhergehend bessere Wartbarkeit. Gerade Geschäftsanwendungen unterliegen in ihrem Lebenszyklus vielen Anforderungen. In einem solchen dynamischen Umfeld kann Komponentisierung – ob Backend-Geschäftskomponenten oder Frontend-Widgets – ihre Vorteile voll zur Geltung bringen.

Wicket-Testunterstützung

Wicket ist eines der wenigen Webframeworks, das eine integrierte Unterstützung für Unit-Tests gewährleistet. Die größten Probleme mit existierenden Web-Unit-Test-Frameworks wie HttpUnit oder Selenium sind:

- Initialaufwand für die Integration des Testframeworks in den Entwicklungsprozess.
- Performance, d. h. ein Unit-Test, der die Laufzeit einer Sekunde übersteigt, führt zu langen Iterationen und wird zu Recht vom Entwickler nicht akzeptiert.
- Refactoring/Änderungen, denn bei Anpassungen am Layout bzw. an der Interaktionslogik ist es sehr mühsam, die

Unit-Tests nachzuziehen. Gerade in stressigen Projektphasen führt diese zum Überspringen der Tests.

Ein in Wicket integriertes Testframework kann genutzt werden, um über Java-Unit-Tests die Browserinteraktion zu simulieren. Hierbei wird der RequestCycle, der die Abarbeitung der Webrequests kapselt, durch eine testspezifische Implementierung gemockt. Zunächst wird mittels der testspezifischen Befüllung des Modells oder mit den Eingabe simulierenden Operationen die grafische Benutzungsoberfläche in den initialen Testzustand gebracht. Anschließend wird die zu testende Aktion ausgeführt. Dafür stehen Anweisungen zum Submitten der Form, zum Ausführen von AjaxEvent oder zum Betätigen der Links zur Verfügung. Ein umfangreiches Potpourri von assert-Funktionen – von der Überprüfung des Model-Status einzelner Controls bis hin zur Überprüfung der Anzahl von Warnhinweisen oder konkreten Meldungen – ermöglicht das Verifizieren des Testergebnisses. Natürlich sind über Mocks auch die Aufrufe von Servicekomponenten auswertbar.

Bereits ein einfacher Unit-Test, der die Seite bzw. die Komponente rendert, führt zu einer deutlich effektiveren Webentwicklung. Einfache Fehler, wie der Mismatch zwischen Java- und HTML-Komponentenbaum oder fehlende Lokalisierungen, können direkt mit Ausführen des Unit-Tests erkannt werden. Das Redeployment und die manuelle Navigation zur zu testenden Seite sowie deren Initialisierung mit entsprechenden Testdaten entfallen. Neben der in Wicket integrierten Lösung zum Testen der Webanwendung existieren eine Reihe weiterer Frameworks mit unterschiedlichen Schwerpunkten.

Auf Basis von JDave [JDave] können an Behavior Driven Development angelehnte Tests entwickelt werden. Die Erweiterung Webdriver ist für konkrete Browsertests nutzbar, um zum Beispiel Browserinkompatibilitäten abzufangen. Des Weiteren gibt es auch Adapter für gängige Testframeworks wie Selenium. Die Erweiterung des Testframeworks, gerade für individuell erstellte Abstraktionen, ist ebenfalls sinnvoll und einfach möglich.

Erstellung technischer und fachlicher Abstraktionen

Eigene Abstraktionen können in Wicket mit Standardmustern in Java umgesetzt werden. Fachentwickler wollen sich nicht mit der technischen Integration von Technologien beschäftigen. Sie wollen klare Anwendungsmuster mit den notwendigen Freiheitsgraden zur Verfügung gestellt bekommen, um optimal und effektiv die Anforderungen des Fachbereichs umzusetzen.

In Wicket ist die Abstraktion von technischem Code mittels Vererbung, Control übergreifendem Verhalten und Komposition möglich. Durch *Vererbung* können bestehende Wicket-Standardkomponenten um eigene Konzepte erweitert werden. Denkbar wären beispielsweise:

- transparente Internationalisierung,
- verbesserte Unterstützung der MVC-Programmierung durch die Erweiterung der Controls um aus Swing bekannte Event-Listener,
- Implementierung einer FocusPolicy,
- Erweiterung der Tabelle um Selectionhandling, Renderer oder Kontextmenü
- und vieles mehr.

Aber auch die Vererbung von Markup ist mit Wicket möglich. So kann zum Beispiel eine einheitliche Seitengestaltung forciert werden. Analog zur Vererbung der Seiten wird auch der HTML-Code vererbt. Spezielle Wicket-Markup-Tags definieren dabei die Position, wo „Sub-Markups“ integriert werden sollen.

Neben der Vererbung unterstützt Wicket auch die Abstraktion von Control-unabhängigem Verhalten. *Behaviors* können Control-übergreifend implementiert werden. Beispiele für Behaviors sind:

- Darstellung von Pflichtfeldern oder nicht validen Controls,
- Integration von JavaScript-Bibliotheken für Effekte, Kontextmenüs oder anderen Control-unspezifischen Features.

Behaviors können zu fachlichem Code hinzugefügt oder in technischen Erweiterungen von Basiscontrols zentral definiert werden.

Ein weiteres sehr mächtiges Konzept in Wicket ist die *Komposition* von Komponenten, die Erstellung sogenannter CompoundComponents. Mehrere Elemente stellen sich dem Fachentwickler als eine komplexe Komponente dar, welche mit Java-Mitteln integriert werden kann. Technische, aber auch fachliche Kompositionen von Controls können so zentral entwickelt und in vielen fachlichen Komponenten wiederverwendet werden. Grundlage für Komposition sind Panels. Diese Container bieten eine fast identische Schnittstelle wie Webseiten, können aber individuell orchestriert und in vielen Webseiten oder Komponenten wiederverwendet werden.

Mögliche Anwendungsszenarien sind eine Sprachauswahl, ein Währungstextfeld einschließlich Anbindung an die eigenen Datentypen oder ein Layoutmanager für die gitternetzbasierte Gestaltung von Formularen ohne die Implementierung eines HTML-Layouts. CompoundComponents bilden auch die Grundlage für die erwähnte komponentenorientierte Entwicklung der Webapplikationen.

Wicket-Herausforderungen

Die Entwicklung von Geschäftsanwendungen auf Basis von Apache Wicket birgt viele Potenziale. Um diese optimal ausschöpfen zu können, müssen sich die Entwickler mit dem Framework näher auseinandersetzen. Wicket bietet viele einfache Schnittstellen, oft mit mehreren Lösungsalternativen für ein Problem. Projektabhängig sollten diese Freiheitsgrade in individuellen Abstraktionen oder Mustern für die Fachentwickler sinnvoll eingegrenzt werden. Gerade in größeren Projekten ist ein auf das Unternehmen (evtl. auch projektübergreifendes) abgestimmtes Framework zu empfehlen. Wicket bietet die Möglichkeit, schnell und mit wenig Aufwand Anwendungen zu entwickeln. Unsere Erfahrung zeigt, dass mit einem projekt- oder unternehmensspezifischem, auf Wicket aufbauenden Framework effektiv sehr gut wartbare Web 2.0-Anwendungen entwickelt werden können.

Oft wird die mangelnde Dokumentation von Wicket kritisiert. Dies kann mehr und mehr widerlegt werden. Viele Blogs, die hilfsbereite Community und auch einige verfügbare Bücher (aktuell gibt es zwei deutschsprachige Bücher auf dem Markt) ermöglichen hier einen guten Einstieg. Als Framework-Entwickler kommt man allerdings ab und zu am Lesen des Quellcodes nicht vorbei.

Wicket ist kein Standard. Aktuell arbeiten achtzehn Entwickler an dem Wicket-Core-Framework. Dazu kommen die vielen meist Open-Source-Projekte, die Erweiterungen zur Verfügung stellen. Wie das Beispiel von Cocoon zeigt, können auch bei Apache gehostete Webframeworks schnell eingestellt werden. Doch Wicket wird zum Mainstream. In Deutschland setzen bereits viele große Firmen verschiedener Branchen bei ihrer Webentwicklung auf Wicket. Das ist zwar keine Garantie für eine sichere Zukunft, doch diese hat man auch bei „Standards“ nicht. Hinzu kommt, dass die genannten Entwickler fast ausschließlich in ihrer Freizeit an Wicket arbeiten bzw. mit Wicket-Beratung ihr Geld verdienen. So ist die Entwicklung von neuen Features nicht immer zeitnah möglich. Trotzdem stehen die meisten von ihnen in der Usergroup den Fragen der Community zur Verfügung. Oft finden sich in Diskussio-

nen auch Interessenverbände, die in einem Wicketstuff-Projekt die individuellen Erweiterungen umsetzen. Wie die aktuelle Version 1.4 zeigt, werden aber weiterhin große, aufwendige Anpassungen, wie die Erweiterung der Controls und Models um Generics zur typsicheren UI-Entwicklung, angegangen und umgesetzt.

Fazit

Wicket etabliert sich zunehmend als Framework zur Entwicklung von Web 2.0-Geschäftsanwendungen. Dank der klaren, einfachen und transparenten Architektur sowie der strikten Trennung zwischen Layout und Logik ist es möglich, ansprechende, komponentenorientierte, gut wart- und testbare Komponenten zu entwickeln. Zwar ist ein gewisser Initialaufwand für das Erlernen der Konzepte und die Einschränkung der Freiheitsgrade notwendig, um das volle Potenzial des Frameworks auszuschöpfen. Dafür lässt sich Wicket sehr gut um eigene Abstraktionen erweitern, um den Fachentwickler ideal bei der Entwicklung seiner Anforderungen zu unterstützen. Er kann sich aus dem Baukasten von Wicket-Standard- und individuellen Komponenten bedienen und diese auf Basis von bewährten UI-Mustern zu fachlichen Anwendungen orchestrieren. Findet man das richtige Maß an Abstraktion, kann sich der Anwendungsentwickler voll auf seine Hauptaufgabe konzentrieren, auf die optimale Umsetzung der fachlichen Anforderungen. Die Auseinandersetzung mit verschiedensten Technologien, Frameworks und deren Inkompatibilitäten bleibt ihm somit erspart. Kurzgefasst: die Entwicklung von Webanwendungen mit Apache Wicket macht einfach Spaß.

Links

[JDave] `http://www.jdave.org/modules.html`
[JTrac] `http://www.jtrac.info/`
[Trac] `http://trac.edgewall.org/`
[Wicket] Apache Wicket – Introduction, `http://wicket.apache.org/introduction.html`
[WicketOsgi] Antilia, `http://code.google.com/p/antilia/`
[WicketPerformance] Seam/JSF vs Wicket, `http://ptrthomas.wordpress.com/2009/01/14/seam-jsf-vs-wicket-performance-comparison/`

Alexander Elsholz arbeitet als Senior IT-Architekt bei der WidasConcepts GmbH, einer IT-Unternehmensberatung mit dem Fokus auf den IT-Architekturen und Anwendungsentwicklung. Zu seinen Schwerpunkten gehören die Entwicklung von serviceorientierten JEE-Architekturen sowie die Konzeption und die Realisierung von Standardsoftware.
E-Mai: alexander.elsholz@widas.de.

Elena Stoll ist Diplommathematikerin und arbeitet als Anwendungsentwicklerin bei der WidasConcepts GmbH. Sie beschäftigt sich in erster Linie mit der Konzeption und Entwicklung von modernen Weboberflächen mit Ajax sowie dem Design und der Umsetzung von web-basierten Java-EE-Anwendungen.
E-Mail: elena.stoll@widas.de.