Nicht schön, aber praktisch

Das Google Web Toolkit

Steffen Schäfer, Papick G. Taboada

Das Google Web Toolkit (GWT) hat im vergangenen Jahr viel Aufmerksamkeit erhalten, denn mit Google AdWords und Google Wave sind die ersten großen GWT-basierten Anwendungen von Google erschienen. Für weiteres Aufsehen haben die Neuerungen in der aktuellen Version 2.0 von GWT selbst gesorgt. Was aber ist GWT? Mit einem optimierenden Compiler und pfiffigen Codegeneratoren werden maßgeschneiderte JavaScript-Anwendungen erstellt, deren Entwicklung findet allerdings in Java statt. Im folgenden Artikel werden die grundlegenden GWT-Konzepte vorgestellt, gefolgt von einem kurzen Überblick über die Neuigkeiten der Version 2.0 und über die Architektur-„Best Practices".

Web-2.0-Softwareentwicklung

Die Webentwicklung heute bringt sowohl Entscheider als auch Entwickler in eine schwierige Lage. Aufgrund einer technologisch kaum definierbaren „Mit Web 2.0 geht alles"-Erwartungshaltung der Auftraggeber werden sehr hohe Ziele gesteckt, andererseits möchte man das Web von morgen (s. gleichnamigen Kasten) heute im schlimmsten Fall auf Basis einer vom Microsoft Internet Explorer, Version 6 dominierten Landschaft von gestern betreiben. In diesem Rahmen sollen dann nachhaltige Entscheidungen getroffen werden. Und nicht selten mutiert die zu treffende Wahl eines Web-Frameworks unkontrolliert zur betriebspolitischen Forderung nach einer langfristigen Client-Strategie.

So trägt die Strategie der Zukunft nicht seltenen den Titel „RIA". Unter dem Überbegriff „Rich Internet Application" (RIA) wurden verschiedene Ansätze vorgestellt, wie man das ursprüngliche HTML-Terminal (Browser) in eine Anwendungsplattform verwandeln kann. Gesetztes Ziel ist es, die Vorteile beider Client-Modelle (Web- und Rich Client) in einer Lösung zu vereinen:

- Unkomplizierte Softwareauslieferung: Webanwendungen müssen nicht installiert und aktualisiert werden, die Anwendung ist immer aktuell. Änderungen und Fehlerbehebungen können im Vergleich zu Rich Clients schnell eingespielt und ausgerollt werden.
- Geringe Latenz in der Bedienung: Rich Clients verarbeiten die Benutzerinteraktion prinzipiell lokal (von Fehlimplementierungen abgesehen ...) und können die Server-Kommunikation in den Hintergrund verlagern.
- Ansprechende Benutzeroberfläche: Rich Clients bieten nicht selten eine größere Vielfalt an GUI-Komponenten und dadurch eine höhere Flexibilität in der Gestaltung von (komplexen) Benutzungsoberflächen.

Eine RIA-Strategie bleibt nicht ohne Konsequenz. Bei dem Ansatz wird der gestandene Webanwendungsentwickler mit einem Paradigmenwechsel konfrontiert: Bekannte und erprobte Architekturmuster, Idiome und Werkzeuge aus der klassischen Webanwendungsenwicklung verlieren ihre Gültigkeit, ihren Sinn und/oder Nutzen. Während Entwurfsmuster aus der Rich-Client-Entwicklung hier eine Renaissance erleben, werden Entwicklungswerkzeuge vor ganz neue Anforderungen gestellt. Auch bisher eingesetzte Bibliotheken können größtenteils nicht mehr eingesetzt werden, erarbeitete Lösungsansätze funktionieren in der neuen Architektur nicht mehr.

Durch den massiven Einsatz von JavaScript sind RIAs heute auch ohne den Einsatz von proprietären Lösungen (Plug-ins) möglich. Allerdings findet man sich plötzlich in einer reinen JavaScript-Entwicklung wieder. Das führt zu neuen Problemen:

- Ein guter Java-Entwickler ist nicht zwingend ein guter JavaScript-Entwickler.
- Während die Implementierung der JavaScript-Sprache inzwischen in den Browsern ausreichend kompatibel ausfällt, gestaltet sich die mit DHTML notwendige DOM-Manipulation in den Browsern sehr unterschiedlich.

Verständlicherweise scheuen viele diesen Weg, da das notwendige JavaScript- und Browser-Know-how selten vorhanden ist. In diesem Umfeld ist das Google Web Toolkit angetreten, um die Probleme in der Team-Skalierung und des fehlenden Software Engineerings in der Webanwendungsentwicklung zu lösen.

Das Web von morgen?

Was ist aber die Webentwicklung von morgen? Ein Blick in die Vergangenheit zeigt einen Browser, der sich über viele Jahre hinweg – trotz technologischer und sicherheitstechnischer Mängel – als Marktführer zum De-facto-Standard etablieren konnte. Auch in Bezug auf Standards (HTML, CSS, JavaScript, HTTP) hat man sich nicht wirklich weiterentwickelt. So gesehen haben Web-Frameworks seit vielen Jahren ein stabiles Ziel – und trotzdem konnte sich hier kein Standard etablieren: Ein Phänomen, wie wir es mit dem Struts-Projekt erlebten, hat sich in der modernen GUI-Komponenten-basierten Webentwicklung nicht wiederholt.

Ein Blick in die aktuellen Geschehnisse zeigt erdbebenartige Fortschritte. Mit HTML5 kommt Bewegung in die festgefahrenen Web-Standards: Flash und Applets haben – unter anderem – Lücken in den Fähigkeiten/Möglichkeiten der reinen DHTML-Entwicklung gestopft. Diese Lücken werden jetzt zum Teil durch HTML5 geschlossen, mit dem Ergebnis, dass in Zukunft noch mehr mit JavaScript gelöst werden kann. Web-2.0-Anwendungen zeichnen sich heute schon teilweise durch den Verzicht auf DHTML-fremde Technologien (Flash, Applets) aus. Also wissen wir heute schon, dass es morgen nicht weniger JavaScript wird.

Bleibt noch der gewagte Blick in die Zukunft: Hinter HTML5 haben sich zwei große Unternehmen gestellt – namentlich Google und Apple. Bis auf den Internet Explorer verstehen alle modernen Browser größtenteils die unfertige HTML5-Spezifikation. Microsoft hat HTML5-Unterstützung prinzipiell angekündigt. Dadurch steht aber jetzt schon fest: HTML5 wird tragischerweise mittelfristig noch keine Rolle spielen – hier muss noch auf den Internet Explorer gewartet werden. In den letzten Jahren haben sich Unternehmen nicht besonders migrationsfreudig gezeigt. Sollte sich dieses Verhalten nicht ändern, dann wird HTML5 auch langfristig keine Rolle spielen.

Das Google Web Toolkit 2.0

Das Google Web Toolkit (GWT, oft „Gwit" ausgesprochen) ist vermutlich eines der am häufigsten missverstandenen Technologien im Java-Umfeld:

- GWT ist kein JavaScript-Toolkit wie z. B. Dojo,
- GWT ist nicht die Übertragung/Nachahmung des Swing-Komponentenmodells in den Browser,
- GWT nutzt keine Dienste von Google,
- GWT wird inzwischen auch von Google [Schu09] verwendet: Google AdWords und Wave werden mit GWT entwickelt.

Technisch gesehen ist GWT ein Java-nach-JavaScript-Compiler und ermöglicht dem Entwickler, eine Anwendung in Java zu schreiben und diese dann in JavaScript zu kompilieren. Dabei liest der GWT-Compiler den Java-Quelltext (Java-5-Syntax) und wandelt diesen in JavaScript um. Das durch GWT bereitgestellte GUI-Komponentenmodell ist in erster Instanz eine Abbildung der HTML-Komponenten. Zusätzlich wird eine Handvoll komplexer GUI-Komponenten wie beispielsweise Tab-Panels, Dialoge und Trees angeboten. GWT liefert auch ganz spezielle Lösungsansätze für Internationalisierung, RPC-Kommunikation, Browser-History-Management und viele mehr. Aus dieser Perspektive darf GWT auch als Framework betrachtet werden.

In der Entwicklung unterscheidet GWT zwischen zwei Modi:

- Im *Development-Modus* wird die Anwendung in der in GWT mitgelieferten „Development-Shell" gestartet. Diese startet intern einen Jetty-Server als Web-Container. Es findet kein aufwendiges „packaging & deployment" statt. Seit der GWT-Version 2.0 kann die Anwendung dann mit verschiedenen Browsern getestet werden. Da zu diesem Zeitpunkt über einen speziellen Mechanismus – ganz ohne JavaScript-Erzeugung – der Bytecode in der Entwicklungsumgebung ausgeführt wird und kein JavaScript erzeugt wurde, ist eine Code-Änderung in der Entwicklungsumgebung durch ein Aktualisieren im Browser sofort sichtbar. Auch das Setzen von Breakpoints im Java-Quelltext und selbst das Debuggen der Anwendung im Java-Quelltext werden durch diesen speziellen Mechanismus ermöglicht.
- Die Anwendung wird dann schließlich mithilfe des GWT-Compilers in JavaScript umgewandelt. Hier spricht man vom *Production-Modus*, die Anwendung wird nicht mehr über den speziellen Mechanismus, sondern als reine JavaScript-Anwendung ausgeführt.

Durch den Einsatz pfiffiger Codegeneratoren während des Compiliervorgangs wird nicht bloß eine einzige Webanwendung, sondern es werden Varianten für alle unterstützten Browser und deklarierten Sprachen (i18n) erzeugt. So erzeugt z. B. der GWT-Compiler für eine Anwendung, die in drei Sprachen internationalisiert wurde, achtzehn verschiedene Webanwendungen. Dadurch wird sichergestellt, dass jeder Browser nur den notwendigen, für ihn geeigneten JavaScript-Code und die verwendete Sprache laden muss.

GWT kann das generierte JavaScript komprimieren (GWT nennt es Obfuscation), Bilder und andere Ressourcen zu einer Datei packen, schlanken RPC-Code erzeugen und noch einige weitere Optimierungen vornehmen. Seit der GWT-Version 2.0 kann der Entwickler sogenannte Split-Punkte definieren: Der Compiler untersucht diese Punkte auf Querabhängigkeiten und versucht mittels der gekennzeichneten Bereiche nachladbare Fragmente zu generieren. Dadurch wird der initiale Download der JavaScript-Anwendung etwas kleiner, da unabhängige Teile bei Bedarf nachgeladen werden können.

Üblicherweise wird eine GWT-Anwendung (die Erzeugnisse des Compiliervorgangs) als WAR-Archiv ausgeliefert. Die Notwendigkeit eines Java-EE-Web-Containers ist aber erst dann gegeben, wenn die Anwendung den GWT-RPC-Mechanismus nutzt. Dieser basiert nämlich auf der Servlet-Spezifikation und benötigt eine entsprechende Laufzeitumgebung.

Die moderne Webanwendung folgt dem „Single Page"-Prinzip

Dank Ajax und DHTML findet ein Wandel in der Architektur von Webanwendungen statt. Der Browser wird plötzlich nicht mehr als dummes HTML-Terminal, sondern als Anwendungsplattform verwendet.

An dieser Stelle spricht man vom „Single Page"-Prinzip: Eine Webanwendung wird einmalig in den Browser geladen, es findet kein Page-Hopping mehr statt. Die durch den Request-Render-Response-Zyklus erzeugte Latenz entfällt in diesem Ansatz komplett. Dafür muss der Anwender einen größeren Initial-Download der Anwendung (der ersten HTML-Seite) in Kauf nehmen, der dann aber im Cache des Browsers verbleiben kann.

Im Gegensatz zu der klassischen Vorgehensweise führt nicht jede Benutzerinteraktion zu einem Neuladen der Seite. Alle Benutzerinteraktionen werden im Browser abgearbeitet. Erst wenn Daten vom Server benötigt werden, findet eine Kommunikation statt. Es werden dann lediglich die benötigten Daten angefordert und nicht noch mal die ganze Seite, denn die Daten werden dann von der Webanwendung verarbeitet und angezeigt.

In einer Ajax-ifizierten Architektur wird die Server-Kommunikation auf ein Minimum reduziert. Das spart nicht nur serverseitige Ressourcen, sondern reduziert ganz nebenbei auch noch die Latenz in der Anwendung.

Da ein Browser laut HTTP-Spezifikation nur zwei Verbindungen zu einem Server aufbauen kann, bietet GWT Mechanismen an, um die Anzahl der zu ladenden Ressourcen zu reduzieren. In GWT spricht man hier von sogenannten „ClientBundles", in denen zum Beispiel Bilder als Mosaik und CSS-Ressourcen zusammengefasst werden. Bilder werden dann beispielsweise über CSS-Clipping im Browser wieder richtig angezeigt.

Der Compiliervorgang wird somit nicht zu einer Schwäche, sondern zu einer Stärke von GWT. Dadurch wird eine Trennung zwischen Entwicklungs- bzw. Projekt- und Deployment-Layout möglich. In der Entwicklung werden aus wartungstechnischen Gründen Ressourcen pragmatischerweise in mehreren Fragmenten einzeln verwaltet. Durch den Compiliervorgang werden die Ressourcen dann so umgepackt, dass HTTP-Connection-Engpässe und Caching-Aspekte berücksichtigt werden können.

Für die Entwicklung in der Eclipse IDE liefert das Google-Team ein Plug-in. Für andere Entwicklungsumgebungen gibt es verschiedene Lösungen von Drittanbietern oder aus der Open-Source-Gemeinde. Inzwischen sammelt sich um GWT herum eine sehr große Entwickler-Gemeinde, was sich in der Anzahl der Bücher, Projekte und Blogs rund um das Thema GWT verdeutlicht.

Im direkten Vergleich zu anderen JavaScript-Toolkits schneiden die durch GWT bereitgestellten Komponenten sehr schlecht ab. Das GWT-Team hat bisher das eigene Komponentenmodell als reine Abbildung der HTML-Komponenten betrachtet, doch gerade das bescheidene Aussehen und das Fehlen komplexer GUI-Komponenten wie beispielsweise eine Datentabelle werden oft als großes Manko angesehen. Auch das Fehlen eines Databindings, wie man es von anderen Web-Frameworks kennt, wird oft bemängelt. Diese Lücken haben inzwischen verschiedene Open-Source- und kommerzielle Projekte gefüllt.

Deswegen steht GWT auch nicht für „Eye Candy", sondern viel mehr für Software Engineering in der Webanwendungs-

entwicklung. Diskussionen, inoffizielle Ankündigungen in der offiziellen GWT-Mailingliste und die aktuellen Commits im SVN-Trunk des Projektes deuten stark darauf hin, dass man für das nächste Release das Problem der Datenhaltungskomponenten und des Databindings gerade in Angriff nimmt. Zum Thema „Eye Candy" ist die Botschaft klar und deutlich: GWT-Anwendungen können und sollen über CSS angepasst werden. Toolkits mit extrem anspruchsvollen GUI-Komponenten haben alle eines gemeinsam: Die Komponenten sind schwer anpassbar, da das mitgelieferte CSS sehr umfangreich (und unübersichtlich) ist. Manchmal ist weniger doch mehr ...

GWT und Software Engineering

Die Entwicklung einer JavaScript-basierten RIA redefiniert die Webanwendungsentwicklung. Bekannte Ansätze verlieren ihre Gültigkeit, alte Entwurfsmuster werden wieder eingesetzt. In der klassischen Request-Response-basierten Webanwendungsentwicklung sind verschiedene Lösungsansätze für das Problem der Seiten-Navigation entwickelt worden, ein Problem, das RIAs so nicht kennen. Andererseits dürfen sich RIA-Anwendung ebenso wie Rich Clients mit dem Problem des Datentransfers zwischen Client und Server auseinandersetzen, ein Problem, das wiederum Technologien, die serverseitig arbeiten, durch ORM-Tools und Lazy-Loading und eine Handvoll „Best Practices" geschickt gelöst bekommen haben.

Wie bei jeder Softwareentwicklung spielen die Erweiterungs- und Wartungskosten eine wesentliche Rolle im Lebenszyklus einer Webanwendung. Gesucht sind Methoden und Vorgehensweisen, die im Vorfeld zu besserer Softwarequalität führen, und Architekturmuster, die eine nachhaltige Wartung und Pflege der Webanwendungen begünstigen.

Glücklicherweise können wir dank des „Java to JavaScript"-Ansatzes genau diejenigen Werkzeuge und Methoden auch für die Webentwicklung einsetzen, die wir seit Jahren für die Qualitätssicherung während der Java-Softwareentwicklung eingesetzt haben. Stellvertretend seien hier Refactoring als Methode sowie Findbugs, PMD und Checkstyle als Werkzeuge genannt.

Unglücklicherweise stehen uns aufgrund der fragmentarischen Java-Unterstützung zur Laufzeit weder Reflection noch Dynamic Proxies zur Verfügung, sodass sich einige der etablierten Frameworks und Techniken aus der Rich-Client-Entwicklung nicht so einfach in die GWT-Entwicklung übertragen lassen. Diese Einschränkung ist dennoch sinnvoll, denn das Einbinden aller benötigten Metadaten und das Bereitstellen einer zu Java ähnlichen Laufzeitumgebung würde die JavaScript-Anwendung in eine nicht vertretbare Größe anschwellen lassen.

Architektur-„Best Practices"

Die hier vorgestellten „Best Practices" für die Architektur von GWT-Anwendungen stammt aus einem Vortrag von Ray Ryan auf der Google I/O 2009 [Ryan09,pgt09]. Diese Architekturmuster wurden aus der Neuentwicklung der Anwendung AdWords von Google gewonnen und in dem Vortrag vorgestellt:

- EventBus: Kommunikation einzelner Komponenten in der Anwendung über einen zentralen Event-Bus.
- Command-Pattern: Client-Server-Kommunikation über Aktion/Result-Objektpaare.
- Model-View-Presenter: Aufbau und Verhalten der UI-Komponenten. In diesem Ansatz findet kein direkter Zugriff zwischen Model und View statt, das gesamte Verhalten der Benutzungsoberfläche ist im Presenter gekapselt und Unit-Testbar.
- Places: Über eine „Place"-Abstraktion und die Verwendung der History API von GWT werden die Vorwärts- und Rückwärts-Navigation sowie das Anspringen der Anwendung über einen Bookmark geregelt.

Im Rahmen dieses Artikels werden aus platztechnischen Gründen nur EventBus und Command-Pattern näher vorgestellt.

EventBus

Ein für Entwickler und Architekten sehr bekanntes Phänomen in der Softwareentwicklung ist der Wildwuchs von Referenzen zwischen Objekten, wodurch Veränderungen nur noch schwer durchführbar sind.

Damit einzelne Komponenten (die Festlegung der Granularität einer Komponente bleibt letztendlich dem Entwickler und/oder Softwarearchitekten überlassen) nicht über Referenzen aneinander gekoppelt werden, wird ein EventBus eingeführt. Da GWT die Erweiterung des Event-Modells zulässt, kann an dieser Stelle auf bestehende Infrastruktur zurückgegriffen werden. Die Typisierung wird dadurch gewährleistet, dass eigene Event-Klassen eingeführt werden.

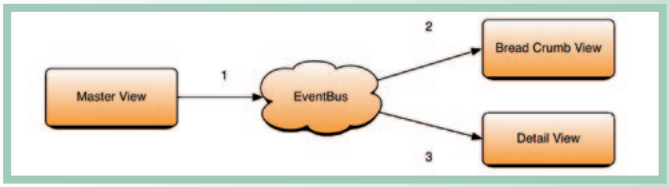

Abb. 1: Master-Detail-Ansicht mithilfe des EventBus

Als Beispiel soll eine einfache Master-Detail-Ansicht vorgestellt werden. Statt beide Komponenten (MasterView und DetailView) über eine Referenz miteinander zu verdrahten, kann auch der gefühlte Umweg über den EventBus gewählt werden (s. Abb. 1). Die Vorgehensweise im Detail:

- Es werden eine Event-Klasse und die entsprechende Event-Handler-Schnittstelle definiert.
- Die Detailansicht meldet sich beim EventBus für eine bestimmte Event-Klasse an.
- Die MasterView feuert über den zentralen EventBus Events, wenn Datensätze selektiert wurden.
- Die DetailView wird vom EventBus über die Datensatz-Selektion benachrichtigt und zeigt die Daten an.

Im Prinzip handelt es sich um das altbekannte Observer-Muster. Im Falle einer Erweiterung (eine andere Komponente, zum Beispiel eine im Nachhinein eingeführte Bread-Crumb-Navigationskomponente) kann man sich auch am EventBus für das gleiche Event anmelden und bei einem Selektions-Ereignis entsprechend die Navigations-Anzeige aktualisieren.

Nicht nur die Erweiterung gestaltet sich einfach, auch ein Entfernen der eingeführten Bread-Crumb-Navigation wird ohne Nebeneffekte stattfinden können: Es reicht, die Komponente aus dem System zu nehmen. Dadurch meldet sich die Komponente nicht mehr am EventBus an und wird auch nicht mehr benachrichtigt.

Command-Pattern

Mit dem Command-Pattern wird unter anderem das Problem des Wildwuchses in Service-Schnittstellen angegangen. In der Client-Kommunikation in Client-Server-Architekturen über RPC-Schnittstellen findet man nicht selten eine Vervielfachung von Datentransferobjekten, die fachlich ein und demselben Do-

mänenobjekt zuzuordnen sind. Diese Vervielfachung entsteht aus dem technischen Ansatz, das Problem der Teildatenübertragung für unterschiedliche Masken zu lösen. In der folgenden Schnittstelle wird dieser Sachverhalt am Domänenobjekt „Kunde“ exemplarisch vorgeführt:

```
DTOKundeMitAllem          getKundeMitAllenDaten(String kundeId);
DTOKundeFlach             getKundeFlach(String kundeId);
DTOKundeFuerTelefonListe  getKundeFuerTelListe(String kundeId);
```

Diese Schnittstelle wird dann von allen Komponenten in der Software verwendet, die in irgendeiner Form Kundendaten anzeigen wollen. Die Einschränkung der zu übertragenden Daten kann sowohl technisch sinnvoll als auch fachlich notwendig sein. Die Definition mehrerer Transferklassen sorgt für die Typsicherheit in den unterschiedlichen Anwendungsfällen. Solche Schnittstellen führen schnell zu schwer wartbarer Software, da sie einerseits für eine Explosion im Klassenmodell sorgen und andererseits Bindungen über die verschiedenen Komponenten hinweg erzeugen.

Der von Ryan vorgeschlagene Lösungsansatz mit dem Command-Pattern ersetzt den schnittstellenorientierten RPC-Ansatz durch einen ereignisorientierten Ansatz: Die Kommunikation zum Server wird über Action/Result- und Handlerklassen (s. Abb. 2) modelliert – die Schnittstelle als solche entfällt, ohne dass die Typsicherheit aufgegeben werden muss. Die Granularität solcher Aktion/Result-Paare kann in der Anwendung variieren. Denkbar sind grobgranulare Aktionen im Sinne von „Gib alles, was die Maske für die initiale Anzeige braucht“ bis hin zu recht feingranularen Aktionen wie die einzelne Validierung einer Eingabe.

Ähnlich wie bei dem EventBus sind die Auswirkungen in der Wartung der Software bemerkenswert. Durch einfaches Hinzufügen des Tripels (Action/Result/Handler) bzw. durch Löschen eines Tripels wird das Kommunikationssystem erweitert bzw. geschrumpft. Die Result-Klassen sind an die Action-Klassen gekoppelt und werden lediglich in deren Kontext verwendet, sodass keine Explosion oder Wildwuchs im Objektmodell entsteht. Sogar dem Problem des Betreibens mehrerer Versionen eines Dienstes wird durch das Wegoptimieren des Schnittstellengedankens Rechnung getragen.

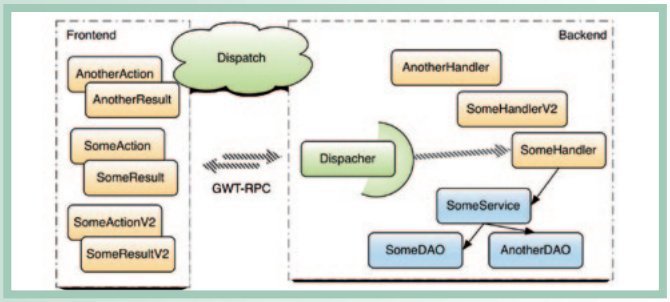

Abb. 2: Aufbau einer Dispatcher-Infrastruktur

Herzstück des Command-Patterns ist die Dispatcher-Infrastruktur. Hier gibt es bereits in der Open-Source-Gemeinde den einen oder anderen fertigen Ansatz. Dank der zentralen Natur des Dispatcher-Dienstes kann an dieser Stelle Fehlerbehandlung, Caching, Retry-Algorithmen, Sicherheitsinformationen usw. zentral implementiert werden.

Fazit

Dank GWT kann Software Engineering endlich den Weg in die Webanwendungsentwicklung finden. Auch wenn die GWT-Komponenten keinen Schönheitswettbewerb gewinnen, sind es doch die inneren Werte, die GWT so attraktiv machen. GWT hat spätestens mit der Version 2.0 den gefühlten Status des „early adopters“ verlassen und ist heute „production ready“.

GWT genießt den für ein Open-Source-Projekt ungewöhnlichen Status, sich keine Sorgen um die Finanzierung machen zu müssen. Google hat nicht nur ein starkes Interesse daran, dass immer mehr Entwickler JavaScript-Anwendungen schreiben (und somit eventuell durch die Erstellung von Mashups indirekt für Kundenbindung sorgen), inzwischen entwickelt auch Google selbst wichtige Produkte mit diesem Werkzeug.

Dennoch mangelt es immer noch an Erfahrung in der Entwicklung komplexer GWT-Anwendungen. Der Mangel wurde dank der Architektur-„Best Practices“ von Ray Ryan beseitigt. Die klare Vorgabe lautet: Entkopplung. Die Open-Source-Gemeinde hat auch bereits reagiert und im Sinne dieser Erfahrungsberichte einige Projekte auf den Weg gebracht, sodass der Einstieg und eine praktische Umsetzung dieser Entwurfsmuster erleichtert werden.

Literatur und Links

[Eckl09] R. Eckl, B. Gleich, A. MacWilliams, Embedded Java-RIA-Frameworks in: JavaSPEKTRUM, 6/09

[GWT] Google Web Toolkit, `http://code.google.com/intl/de/webtoolkit/`

[GoogleAdWords] Google AdWords, `http://adwords.google.com`

[HTML] HTML5, A vocabulary and associated APIs for HTML and XHTML, W3C Working Draft 4 March 2010, `http://www.w3.org/TR/2010/WD-html5-20100304/`

[pgt09] Best Practices For Architecting Your GWT App, 2009, `http://pgt.de/2009/09/18/best-practices-for-architecting-your-gwt-app/`

[Ryan09] R. Ryan, Google Web Toolkit Architecture: Best Practices For Architecting Your GWT App, Google I/O 2009, `http://code.google.com/intl/de-DE/events/io/2009/sessions/GoogleWebToolkitBestPractices.html`

[Schu09] A. Schuck, Google Wave: Powered by GWT, Google I/O 2009, `http://code.google.com/intl/de/events/io/2009/sessions/GoogleWavePoweredByGWT.html`

Steffen Schäfer ist bei der Orientation in Objects GmbH in Mannheim als Entwickler, Trainer und Berater tätig. Er beschäftigt sich mit den Themen Java und XML. Seine Schwerpunkte liegen hierbei in den Bereichen GWT, Eclipse RCP und Java mit Open Office. Kontakt: http://oio.de.

Papick G. Taboada wechselte Ende 1991 für sein Wirtschaftsingenieurstudium an der Universität Karlsruhe (TH) von Brasilien nach Deutschland. Seit 1997 ist er freiberuflicher Softwarearchitekt und Technology Scout. Schwerpunkte seiner Arbeit liegen in der Konzeption und Erstellung von Softwarearchitekturen im Java-EE-Umfeld mit Open-Source-Technologien. Seine gesammelten Erfahrungen gibt er auch als Coach und Trainer weiter. In den vergangenen Jahren hielt er verschiedene Vorträge auf namhaften Konferenzen und veröffentlichte mehrere Artikel in anerkannten Fachzeitschriften.
Kontakt: http://pgt.de.