Ein Überblick über die Chancen und Möglichkeiten von GWT

The Web 2.0 Toolkit

Das Google Web Toolkit ist eine Sammlung von Werkzeugen und Bibliotheken für die Entwicklung von JavaScript-Anwendungen. Das Toolkit wird sowohl intern von Google als auch weltweit in vielen Projekten eingesetzt. Das Besondere an GWT ist die Vorgehensweise: Ein Compiler liest Java-Quellen ein und generiert JavaScript. Somit können Java-Entwickler auf bestehendes Wissen, Erfahrungen und Werkzeuge aufsetzen. Allerdings hat GWT nicht den Versuch unternommen, das gesamte JDK und Swing im Browser zu unterstützen. Die JDK-Unterstützung ist fragmentarisch (nicht alle Klassen können verwendet werden), und GWT liefert ein eigenes, an HTML orientiertes UI-Komponentenmodell.

von Papick Taboada und Benjamin Barth

Das Google Web Toolkit hat nicht das Ziel, das Aussehen von Desktopanwendungen im Browser nachzuahmen. Der Fokus der GWT-Entwicklung liegt im Software-Engineering: Einerseits ermöglicht die Entwicklung in Java per Definition den Einsatz von Entwurfsmustern, qualitätssichernden Vorgehensweisen und Tools. Sogar das Debuggen der im Browser laufenden Webanwendung aus dem Java-Code heraus ist nicht nur möglich, sondern sogar sehr einfach. Aber was ist eine GWT-Anwendung? Prinzipiell ist eine mit GWT erzeugte Anwendung ein Script, das vom Browser ausgeführt wird. An dieser Stelle ist die Analogie zu Java und Swing angemessen: Java ist die Programmiersprache, Swing liefert das UI-Komponentenmodell. Zwar bieten Swing und GWT verschiedene Komponentenmodelle, dennoch ist das Programmiermodell vergleichbar. Daher ist eine GWT-Anwendung aus Architektursicht eher einem Rich-Client als einer Webanwendung gleichzusetzen: Die Anwendung wird heruntergeladen, im Browser ausgeführt und kommuniziert mit dem Server. Dabei wird die ursprünglich angesteuerte Webseite nie verlassen („Single Page"-

Prinzip), die Anwendung bleibt im Cache des Browsers und muss bei erneutem Besuch nicht noch einmal geladen werden. Moderne Browser stellen die notwendigen Schnittstellen zur Verfügung, um lokal Daten zu speichern, sodass sogar ein Arbeiten im Offlinemodus umsetzbar ist.

In Bezug auf den ökonomischen Umgang mit den Ressourcen sieht die Bilanz außerordentlich gut aus. Die Anwendung kann im Cache behalten werden und sogar in Scheiben (Split Points) ausgeliefert werden. Das Ausliefern der Anwendung findet also nur einmal statt, es folgt lediglich die Kommunikation mit dem Server für den Datenaustausch beziehungsweise das Ausführen von Befehlen auf dem Server. Das Rendern der Webseiten, das früher zu den Aufgaben des Webservers gehörte, wird jetzt effizient auf die Clients (Browser) verlagert. Die Kommunikation mit dem Server wird auf ein Minimum reduziert. Intelligentes Zwischenspeichern (Cachen) und die Einführung von lokal gespeicherten Daten können die Bilanz noch besser aussehen lassen. GWT bietet neben der üblichen Kommunikation mittels Austausch von mit JSON und XML formatierten Daten auch ein eigenes Verfahren namens GWT RPC, das das Umwandeln ganzer serialisierbarer Java-Objektbäume beherrscht. In Bezug auf GWT RPC gilt die Analogie zu RMI mit einem wesentlichen Unterschied: Während RMI eine generische (und somit nicht schlanke) Laufzeitumgebung für die Interprozesskommunikation umsetzt, wird bei GWT RPC im Kompiliervorgang auf die Kommunikationsschnittstelle optimierter JavaScript-Code generiert.

Die HTTP-Spezifikation schränkt die Anzahl der offenen persistenten Verbindungen, die ein Browser zu ein- und demselben Server aufbauen kann, auf zwei ein. Da JavaScript der Same Origin Police unterliegt, darf man von einem prinzipbedingten Flaschenhals sprechen. Auch wenn sich Browser nicht an die Spezifikation halten, ist die Anzahl der Verbindungen dennoch beschränkt, meistens auf sechs oder acht Verbindungen. Wie wenig das ist, wird erst dann deutlich, wenn man Webanwendungen unter die Lupe nimmt und untersucht, wie viele Requests für die Anzeige einer Seite durchgeführt werden. Zum Beispiel hat das Laden der Einstiegsseite bei Amazon.de zu beindruckenden 90 Requests geführt (JavaScript, Bilder und CSS-Dateien, die geladen werden). Da nicht alle Anfragen parallel ausgeführt werden können, entstehen beachtliche Latenzen/Wartezeiten, bis alle Ressourcen geladen worden sind. Einerseits liegt dieses Ladeverhalten in der Natur von Webanwendungen, andererseits sind Lösungsansätze, die dieses Problem reduzieren, bekannt. Die Anzahl der möglichen Verbindungen zu erhöhen, würde nur gleichzeitig auch die Last auf dem Server beachtlich erhöhen, ohne das Problem wirklich zu lösen. Außerdem müssten die Browser entsprechend konfiguriert werden, was nicht immer einfach oder gar möglich ist. Die Lösung des Problems liegt darin, die Anzahl der notwendigen Verbindungen prinzipiell zu reduzieren. Das Single-Page-Prinzip sorgt schon mal dafür, dass die Anwendung lediglich einmal geladen wird, und somit führt nicht jede Benutzeraktion auf der Webseite zu einem kompletten Neuladen und Rendern der Seite. Des Weiteren können sowohl CSS als auch JavaScript in die geladene HTML-Host-Seite eingebettet und statische Bilder zu einem Mosaik zusammengefasst (und clientseitig per CSS Clipping angezeigt) werden. Zudem besteht zusätzlich die Möglichkeit, Bilder Base64-kodiert ebenfalls in die HTML-Seite einzubetten.

Das Toolkit liefert einen Java-zu-JavaScript-Compiler, der mehr als die eigentliche Übersetzung durchführen kann. Neben der eigentlichen Transformation ist GWT zudem in der Lage, unter anderem die oben genannten Optimierungen vorzunehmen. Der generierte JavaScript-Code kann außerdem je nach Konfiguration zum Beispiel in einer knappen Form (obfuscated) erzeugt werden. Es werden mehrere Permutationen der Anwendung generiert, für jede Sprache und jeden Browser, damit dem Anwender das Laden und Ausführen von unnötigem Anwendungscode erspart bleibt. Dem Toolkit merkt man sehr schnell an, dass es nicht nur Produktionsreife hat, sondern auch tatsächlich produktiv (von Google) eingesetzt wird. So werden zum Beispiel die generierten Dateien mit (sehr) eindeutigen Namen versehen. Dank des „cache“ oder „nocache“ im Namen können Filterregeln im Webserver erstellt werden, durch die entsprechende HTTP Header für das richtige Cacheverhalten im Browser sorgen. Die Art und Weise (Umfang und Konsequenz) mit der Optimierungen am Compiler, Internationalisierungskonzepte, Compilerberichte und vieles mehr umgesetzt werden zeigen, dass GWT für den produktiven Einsatz entwickelt wird.

Das Google Web Toolkit definiert ein eigenes Modulkonzept. Module können andere Module erben und können fachlich sowohl Teile einer Anwendung als auch einzelne Komponenten beinhalten. Die Flexibilität in der Gestaltung der Granularität der GWT-Module ermöglicht die Konzeption und Realisierung eigener, anwendungsspezifischer Bedürfnisse. Allerdings handelt es sich um ein Modularisierungskonzept, das allein zur Compilezeit zum Tragen kommt. Das Toolkit selber besteht aus Modulen und Modulvererbungen.

Besonders interessant sind auch die Erweiterungsmöglichkeiten, die angeboten werden. Da das UI-Komponentenmodell das Composite Pattern umsetzt, ist die Erstellung komplexer, wiederverwendbarer fachlicher Komponenten sehr einfach. Die Implementierung projektspezifischer Widgets ist auch sehr einfach. Der vom Compiler eingesetzte Mechanismus, Klassen spezifisch für die einzelnen Permutationen auszutauschen beziehungsweise die Möglichkeit, Codegeneratoren einzubinden, kann auch in eigenen Projekten verwendet werden. Die Anbindung von bereits existierendem JavaScript-Code in die eigene Anwendung ist sehr elegant umgesetzt worden: Aus dem in Java vorhan-

denen Konzept des Java Native Interface (JNI) wurde JavaScript Native Interface (JSNI), ein einfacher Weg, nativen JavaScript direkt in der Java-Klasse einzubinden. So werden beispielsweise Aufrufe von Fremdbibliotheken ermöglicht. Prominentes Beispiel an dieser Stelle ist die Brücke von GWT zu Google Maps: Es werden Java-Klassen bereitgestellt, die das Google Maps API kapseln.

Architektur?

Die Entwicklung einer JavaScript-basierten RIA stellt die Webanwendungsentwicklung vor neue Anforderungen. Bekannte Ansätze verlieren ihre Gültigkeit, alte Entwurfsmuster werden wieder eingesetzt. In der klassischen Request-Response-basierten Webanwendungsentwicklung sind verschiedene Lösungsansätze für das Problem der Seitennavigation entwickelt worden, ein Problem, das RIAs so nicht kennen. Andererseits dürfen sich RIA-Anwendungen ebenso wie Rich-Clients mit dem Problem des Datentransfers zwischen Client und Server auseinandersetzen, ein Problem, das wiederum Technologien, die serverseitig arbeiten, geschickt gelöst haben: Mit ORM-Tools, verschiedenen Frameworks, Lazy Loading, einer handvoll Best Practices, Tooling und Industrieunterstützung haben sich sehenswerte Lösungsansätze etabliert. Die meisten Ansätze betreiben allerdings keine Ursachenbekämpfung; die Komplexität des Gesamtsystems steigt bei jedem hinzugefügten Framework. Der Betrieb, die Fehlersuche und die Wartung von Webanwendungen sind keine trivialen Tätigkeiten. Gerade die starken Technologiebrüche in der Laufzeitumgebung und in der Entwicklung sind in Bezug auf Effizienz und qualitätssichernde Maßnahmen aus dem Kapitel Software-Engineering absolut kontraproduktiv. Auch das Versprechen, das Problem durch noch bessere Tools in den Griff zu bekommen, ist nicht eingelöst geworden.

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. Webanwendungen haben nicht selten die Anforderung, mit steigender Last entsprechend skalieren zu müssen. Technologien sollten das nicht verhindern. Sobald es sich nicht um eine Prototyp- oder eine Intranetanwendung für eine überschaubare Menge an Endanwendern handelt, sollte man sich kritisch mit dem Problem der Last und der Skalierbarkeit auseinandersetzen. Dabei müsste berücksichtigt werden, wie hoch der serverseitige Aufwand pro Sitzung ist. Während JSP und Servlets eine leichtgewichtige Laufzeitumgebung mit speicherpersistenten Servlets zur Verfügung stellen, tauchen die ersten Skalierungsprobleme schließlich mit weiteren einzusetzenden Technologien beziehungsweise Frameworks auf. Einerseits ist die Einführung von Komponentenmodellen in der Webentwicklung erstrebenswert, andererseits fordert die Transformation des Request-Response-Zyklus in ein ereignisbasiertes Modell auf dem Server ihren Tribut: einen mit der Anzahl der Teilnehmer steil steigenden Ressourcenverbrauch und eine relativ hohe Latenz. Modulare und komponentenbasierte Entwicklung hat in der Vergangenheit auch dazu geführt, dass im Browser benötigte CSS- und JavaScript-Ressourcen entsprechend der Komponentenstrukturierung auf viele kleine Dateien verteilt wurden. So bringt nicht selten jede UI-Komponente ihre eigenen CSS- und JavaScript-Ressourcen mit ins Boot. Das Ergebnis ist eine HTML-Seite, die viele Requests durch den HTTP Connection Bottleneck schicken muss.

Durch den RIA-bedingten Architekturwechsel und den pfiffigen Compileransatz kann GWT diese Probleme während der Transformation lösen. Das Ergebnis: Latenzen werden nicht mehr in Sekunden, sondern in Millisekunden gemessen. Natürlich vorausgesetzt, der Server kann die RPC Calls entsprechend schnell ausliefern – die Webarchitektur steht einer skalierbaren, schnell reagierenden Anwendung nicht im Weg.

> Latenzen werden nicht mehr in Sekunden, sondern in Millisekunden gemessen.

Der Ansatz

Die Webentwicklung mit GWT unterscheidet sich beachtlich von den bisher üblichen Vorgehensweisen. Im Java-Umfeld haben sich zwei Arten der Entwicklung durchgesetzt: entweder IDE-basiert, zum Beispiel mit Eclipse WTP und Plug-ins, oder Build-Tool-basiert, zum Beispiel mit Maven. Gemeinsam haben die Vorgehensweisen den Fokus auf das Deployment und Redeployment der Anwendung, denn dynamische Webanwendungen in Java dürfen erst einmal paketiert und anschließend „deployt“ werden. Je nach Toolkette kann an der einen oder anderen Stelle optimiert beziehungsweise abgekürzt werden, das Grundprinzip bleibt aber gleich.

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: Da GWT die Entwicklung der Webanwendung in der Programmiersprache Java ermöglicht, sind keine neuen Entwicklungsumgebungen nötig. Der Entwickler kann die bisher eingesetzte Java IDE weiterhin verwenden. Etablierte Konzepte, Vorgehensweisen und Werkzeu-

ge können weiterhin beziehungsweise zum ersten Mal auch für die Webanwendungsentwicklung eingesetzt werden. PMD, FindBugs, Refactoring, etablierte Idiome und Entwurfsmuster, der GWT-Ansatz macht es möglich.

Allerdings geht das Google-Team in Bezug auf den Entwicklungszyklus einen etwas anderen Weg. Da es sich bei GWT nicht um eine dynamische Java EE wie aus dem Lehrbuch handelt, sind auch Eclipse-Plug-ins aus dem Webumfeld kaum nützlich. So hat das Team ein eigenes Plug-in für Eclipse entwickelt, das eine nahtlose Integration der GWT-Entwicklung in Eclipse ermöglicht. Das Google Plug-in for Eclipse, kurz GPE [1], nutzt neben GWT auch das Performance Tool Speed Tracer und unterstützt das Deployment via Google App Engine. Der GWT-Ansatz muss aus zwei Blickwinkeln betrachtet werden. Einerseits gibt es Komponenten, die serverseitig ausgeführt werden, andererseits wird die eigentliche Clientanwendung im Browser ausgeführt. Zunächst die Serverseite: Je nach Anwendung soll der Client mit dem Server über HTTP-Aufrufe oder per GWT RPC kommunizieren. Im Java-Umfeld tun wir das anhand einer Auswahl aus den unzähligen Frameworks, die allesamt früher oder später auf die Servlet-Spezifikation aufsetzen. GWT RPC ist auch nicht anders und basiert ebenfalls auf Servlets. Demnach benötigen wir für den serverseitigen Anteil einer GWT-Anwendung einen Webcontainer.

Der clientseitige Teil einer GWT-Anwendung, also die Webanwendung an und für sich, entspricht eigentlich einer JavaScript-Anwendung, die im Browser über DOM-Manipulation die HTML-Anwendung entstehen lässt. Während der Entwicklung will man die aufwendige Java-zu-JavaScript-Transformation vermeiden. Daher bietet das GWT-Team für verschiedene Browser bestimmte Browser-Plug-ins für die Entwicklung an. Sie ermöglichen ein ferngesteuertes Manipulieren des DOMs im Browser. Dadurch entfällt die Notwendigkeit der Transformation, die Manipulation kann ferngesteuert aus der IDE heraus durchgeführt werden. Diese Vorgehensweise wird dann Developer-Modus genannt. Entweder per Build-Tool (Swing-Anwendung) oder über ein Eclipse-Plug-in wird die Anwendung im Developer-Modus gestartet. Sie setzt die benötigte GWT-interne Laufzeitumgebung für die Fernsteuerung der DOM-Manipulation und einen Webcontainer (ein Jetty) in Gang. Bei diesem Ansatz startet folgerichtig sowohl der Client als auch der Server in ein- und derselben JVM, das ermöglicht das nahtlose Debuggen von Frontend bis Backend der Anwendung in Java, die dennoch praktisch im Browser ausgeführt wird. Neben dem Developer-Modus bietet

das Eclipse-Plug-in noch das IDE-basierte Kompilieren der GWT-Anwendung, verschiedene GWT-spezifische Wizards, Editoren und Projektvalidatoren. Seit dem Ankauf des Unternehmens Instantiations durch Google wurden auch Teile des Produkts GWT Designer in das Google-Eclipse-Plug-in integriert.

Fazit

Googles Erfahrung mit Web-2.0-Anwendungen ist unumstritten, Google Maps und Google Mail haben das Unmögliche möglich gemacht. Dank GWT können Java-Entwickler teilweise an den Browsereigenarten vorbei sauberen JavaScript-Code erzeugen. Manchmal bleibt aber auch der GWT-Entwickler entsetzt staunend vor der eigenen Anwendung im Browser erstarrt und muss feststellen, das der IE9 Probleme hat, einen DIV zu zentrieren. Die durch GWT eingeführte Vorgehensweise in der Entwicklung ist effizient und ähnelt der Vorgehensweise bei der Entwicklung von Rich-Clients mit Java. Der Einstieg in die Technologie wird dadurch sehr erleichtert. Wenn Web 2.0 sich als Integrationsplattform durchsetzt und der Browser die am häufigsten verbreitete Anwendungsplattform bleibt, so wurde mit GWT sicherlich ein Meilenstein in der Geschichte moderner Webanwendungen geschaffen. Dank GWT kann Software Engineering endlich den Weg in die Webanwendungsentwicklung finden. Auch wenn die Komponenten keinen Schönheitswettbewerb gewinnen, sind es doch die inneren Werte, die GWT so attraktiv machen. Das Toolkit hat schon lange den gefühlten Status des „unbekannten Frameworks“ verlassen und ist heute „Production ready“. Als Open-Source-Projekt genießt GWT den 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), sondern inzwischen entwickelt auch Google selbst wichtige Produkte mit diesem Werkzeug. Dennoch mangelt es immer noch an Erfahrung in der Entwicklung komplexer GWT-Anwendungen. Dem wurde dank des Vortrags „Architectur Best Practices“ von Ray Ryan auf der Google IO 2009 Abhilfe geschaffen. Inzwischen sind einige Punkte aus dem Vortrag als Rahmenwerk in GWT eingeflossen, andere sind als Open-Source-Projekte außerhalb der GWT-Entwicklung umgesetzt worden. Insgesamt bietet das Google Web Toolkit eine beeindruckende Sammlung von Werkzeugen, Bibliotheken und Ideen. Die von Google erbrachte Ingenieurleistung in dem Projekt ist erstaunlich.

Wohin geht die Reise?

Da GWT keine offizielle Roadmap hat, musste man bisher auf Quellen wie Logs der Repository Commits, Diskussionen in der Google User Group oder Issues und deren Kommentare zurückgreifen, um geplante Neuerungen vorherzusehen. Seit Kurzem nutzt das GWT-Team Git, sodass die Arbeiten an neuen Features nicht öffentlich werden. Früher konnte man in den einzelnen Feature-Branches der Entwickler im SVN nachvollziehen, woran am eifrigsten gearbeitet wurde. Auch haben einige wichtige Entwickler das Team verlassen. Google hat in den vergangenen Monaten einige Dienste, teilweise nicht überraschend, beendet und eine Phase der „Fokussierung“ eingeleitet. In diesem Kontext und seit der Vorstellung von Dart wurden Sorgen um das Projekt in der Webcommunity laut. So laut, das einige GWT sogar lauthals für tot erklärt haben. Ist GWT wirklich tot? Nein!

Was sagt das GWT-Team selbst dazu?

Eine erste Stellungnahme hat das GWT-Team im offiziellen Blog [2] abgegeben, die Kernaussage: Dart und GWT verfolgen ähnliche Ziele, man sei über das Potenzial von Dart sehr optimistisch. Sollte Dart „ready for primetime“ werden, werde man eine enge Zusammenarbeit mit der GWT-Community anstreben, um Dart näher zu erforschen. In verschiedenen Blogs (bzw. in den Kommentaren) und auf Google+ wird man noch weitere Aussagen finden, die im OIO-Blog [3] zusammengeführt wurden. Eric Clayberg, seit der Übernahme von Instantiations neuer GWT-Teammanager bei Google, versichert, dass GWT alles andere als tot sei und sowohl jetzt als auch zukünftig eine ausgezeichnete Wahl für das Entwickeln von Webanwendungen darstelle. Interessant sind auch die erstmals in der Geschichte von GWT getroffenen Aussagen zu den in der kommenden Version geplanten Neuerungen. Der Fokus des Releases 2.5 liegt demnach in der darunter liegenden Infrastruktur, und es stelle eines der umfangreichsten Releases bisher dar, so Ray Cromwell in seinen Kommentaren. Somit steht offiziell erst einmal fest, dass das Team an GWT festhält. Die Tatsache, dass noch so intensiv an GWT gearbeitet und auch kommuniziert wird, ist auch ein sehr positives Zeichen an die Community. GWT ist und bleibt ein Toolkit für die Webanwendungsentwicklung von heute. Was das Web von morgen mit sich bringt, kann niemand sagen.

Dipl.-Wirtsch.-ing. (TH) Papick Taboada ist Entwickler, Berater, Softwarearchitekt, Technology Scout. Schwerpunkte seiner Arbeit liegen in der Konzeption und Erstellung von Softwarearchitekturen im Java-EE-Umfeld mit Open-Source-Technologien. Mit dem Google Web Toolkit setzt er sich bereits seit 2007 auseinander und hat praktische Erfahrungen in Projekten gesammelt, die er auch als Coach und Trainer weitergibt. In den vergangenen Jahren hielt er verschiedene Vorträge auf namhaften Konferenzen und veröffentlichte Artikel in anerkannten Fachzeitschriften.

Benjamin Barth ist Bachelor of Science in der Fachrichtung Software Engineering und arbeitet vorwiegend an der Planung und Umsetzung von Softwareprojekten und Webanwendungen im Java-EE-Umfeld, wobei nicht selten Open-Source-Technologien zum Einsatz kommen. Mit dem Google Web Toolkit beschäftigt er sich seit mehreren Jahren in diversen Projekten und als Seminartrainer.

Links & Literatur

[1] Google Plug-in for Eclipse: http://code.google.com/intl/de-DE/eclipse/

[2] GWT and Dart, Official-GWT-Blog: http://bit.ly/t8OfcO

[3] Future of GWT and GWT 2.5, OIO-Blog: http://bit.ly/tCNFvI