



JavaScript-Code mit dem Google Web Toolkit erzeugen

Make the Web a Better Place

Quellcode
auf CD

>> PAPICK G. TABOADA



Mit dem Google Web Toolkit (GWT) soll ein Java-Entwickler schnell in die Lage versetzt werden, JavaScript/Ajax-Anwendungen zu schreiben. Die Nachfrage ist groß, und immer mehr Projekte sollen mit JavaScript im Browser dem Trend des Web 2.0 folgen: von intelligenter Validierung über Drag-and-Drop bis hin zu Mashups mit Google Maps. Der Browser hat sich zur beliebtesten Anwendungsplattform entwickelt, die User Experience ist hier das Schlüsselwort.

Seitdem das Google Web Toolkit [1] auf der JavaOne 2006 vorgestellt wurde, erfreut sich das Projekt immer größerer Beliebtheit. Was aber ist GWT? Kurz: eine Technologie bzw. ein Framework, mit dem JavaScript-Webanwendungen erstellt werden können. Das Besondere am GWT-Entwicklungsprozess ist nicht das Ergebnis, sondern die Vorgehenswei-

se: Der Entwickler schreibt die gesamte Webanwendung in Java (kein HTML, kein JavaScript). Der Trick besteht darin, den Java-Quelltext nach JavaScript zu übersetzen, sodass die Vorgehensweise das Nutzen aller Fähigkeiten der IDE wie das Refactoring und Debuggen im clientseitigen Code ermöglicht. Das GWT Framework bietet ein grafisches

UI-Komponentenmodell, ein Modularisierungskonzept, eine fragmentarische Java-Runtime-Emulation, ein API zur Manipulation der Browser-History, eine eigene RPC Implementierung, Internationalisierung und noch einiges mehr. Hier Googles Leitspruch für GWT: „GWT’s mission is to radically improve the web experience for users by enabling developers to use existing Java tools to build no-compromise AJAX for any modern browser“ [2]. Frei übersetzt bedeutet das eine radikale Verbesserung bei der Verwendung von Webanwendungen (für den Endanwender). Bestehende Java-Werkzeuge sollen eingesetzt werden, um ohne Einschränkungen AJAX-Anwendungen auf beliebigen Browsern zu entwickeln.

Es ist schon beeindruckend, was sich hinter dieser neuen Technologie verbirgt. Aber es ist von Google ... Schon seit einiger Zeit hat Google das Image des saube-



ren kleinen Start-ups verloren. Google ist groß, Ziele und Mittel teilweise umstritten. An dieser Stelle erst einmal eine Entwarnung – auch wenn Google draufsteht, ist kein Google drin. Durch die Verwendung von GWT werden keinerlei Dienste von Google (außer GWT selbst) verwendet. Google hat natürlich nichts dagegen, wenn sich immer mehr Produkte bei anderen Google-Technologien bedienen – zum Beispiel Google Search oder Google Maps. Wenn die Hemmschwelle hier bislang bei den nötigen JavaScript-Kenntnissen lag, so ist diese mit GWT letztendlich behoben worden. Auch für JSF gibt es bereits Komponenten, die das Verwenden von Google JavaScript APIs erleichtern: „Naturally, GWT is also a great way to easily take advantage of the latest-and-greatest Google APIs and browser enhancements, such as Google Gears” [3].

GWT ist heute einsatzbereit: Es ist seit Version 1.3 unter der Apache Lizenz 2.0 veröffentlicht und seit Version 1.4 trägt GWT nicht mehr den Zusatz „Beta“ im Namen. Die aktuelle Versionsnummer lautet 1.4.61 und kann von der GWT-Homepage in Google-Code heruntergeladen werden. Das GWT-Team arbeitet gerade an der neuen Version 1.5. Wichtigste Neuerung, an der aktuell gearbeitet wird, ist die Unterstützung der bei Java 5.0 neu eingeführten Sprachelemente wie Annotations und Generics, denn aktuell wird lediglich der Sprachumfang von Java 1.4.2 unterstützt. Die unterstützten Browserplattformen des Projekts sind im Augenblick die aktuellen Versionen des Microsoft Internet Explorers, Firefox, Opera und Safari.

Die Technologie

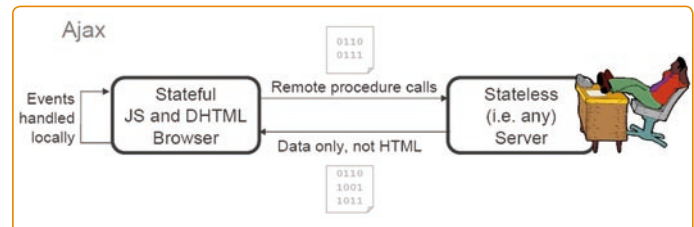
GWT muss einerseits als Technologie und andererseits als Framework betrachtet werden. In der technologischen Betrachtung wird die Vorgehensweise erläutert, anhand derer mit GWT Webanwendungen entwickelt werden.

Eine dynamische Java-Webanwendung – wie wir sie ohne AJAX kennen – ist prinzipiell eine Ansammlung von einzelnen Seiten, die miteinander verlinkt sind. Eine Anwendung entsteht letztendlich dadurch, dass diese Seiten auf gemeinsame Sitzungsdaten zurückgreifen (Session State). Die einzelnen Seiten werden serverseitig erzeugt und per HTTP an den Browser geschickt. Was sich einfach anhört, wird schnell zum Wartungsälptraum. Die bun-

Abb. 1: User Interfaces mit GWT entwickeln [7]



Abb. 2: User Interfaces mit GWT entwickeln [7]



te Vielfalt an Frameworks aus dem Java-Umfeld versucht Ordnung in die eine oder andere Facette der Entwicklung von Webanwendungen zu bringen, und zwar mit dem Ziel, eine höhere Wartbarkeit und Änderbarkeit zu erreichen. Von MVC über Templating, XSL-Transformationen und aufwändigen serverseitigen GUI-Komponentenmodellen werden dem gemeinen Java-Entwickler im Enterprise-Bereich viele Ansätze angeboten.

Seit eine Webanwendung mit AJAX asynchrone Aufrufe zum Server ausführen kann, ist die Notwendigkeit des „Seitenhüpfens“ nicht mehr gegeben. Allerdings müssen die HTML-Seiten so gestaltet werden, dass Teile ohne einen Seitenaufruf neu erstellt bzw. ausgetauscht werden können. Hier kommen schließlich CSS, JavaScript und HTML ins Spiel. Die bösen Drei, auch DHTML genannt, sind – wie schon Ajax – definitiv nichts Neues. Allerdings sind die Unterschiede zwischen den modernen Browsern und sogar zwischen einzelnen Versionen eines Browsers so groß, dass es praktisch unmöglich ist, DHTML effizient in Projekten einzusetzen.

Dank AJAX und DHTML entfällt der Gedanke des Request-Response-Zyklus dynamischer Webanwendungen, wie wir es von JSP und JSF kennen. Die in JavaScript geschriebene Anwendung im Browser ist in der Lage, selbstständig nach Daten zu fragen und die Anzeige zu aktualisieren. Diese Technik wird schon oft in bestehenden Anwendungen eingesetzt, um beispielsweise Suchergebnisse einzublenden, ohne dass die Seite neu geladen werden muss. Eine GWT-Anwendung ist die Weiterführung die-

ses Gedankens: Nicht nur Teile werden „ajaxifiziert“, sondern gleich die gesamte Anwendung. In diesem Sinne ist eine GWT-Anwendung eine statische Webanwendung, die ähnlich wie Flash-Anwendungen oder auch Applets lediglich einmal geladen wird. Ein Applet wird den Server nur aufrufen, um neue Daten zu erfragen oder um eine Benutzereingabe an den Server zu schicken. Ein Applet, das keine Dienste des Servers benötigt, wird nur einmal geladen und ist autonom (oder auch offlinefähig). Das gleiche Prinzip gilt auch für GWT-Anwendungen. Sie laufen autonom im Browser und werden nur dann einen Server benötigen, wenn dessen Dienste gefragt sind. So hat der erste Aufruf der Anwendung prinzipbedingt eine größere Antwortzeit als die einer klassischen Webanwendung.

Java to JavaScript

GWT liefert den GWT Compiler, der Java nach JavaScript übersetzt. Dieser führt nicht nur eine einfache lexikalische Übersetzung durch, sondern ist zudem in der Lage, Optimierungen vorzunehmen. So wird zum Beispiel unbenutzter Code nicht übersetzt und das Ergebnis auf Wunsch komprimiert.

Die Implementierung von DHTML ist in modernen Browsern leider alles andere als kompatibel. Aus diesem Grund erzeugt der GWT Compiler für jeden Browser und für jede Sprache die entsprechenden JavaScript-Dateien. So wird in jedem Browser nur das JavaScript ausgeführt, das auch wirklich vom Browser verstanden wird. Dadurch wird die tatsächliche Menge an JavaScript-Quelltext, der zur Laufzeit im Browser benötigt wird, reduziert.



Aus dem in Java vorhandenen 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, welche das Google Maps API kapseln.

Die Vorgehensweise

Da GWT die Entwicklung von Webanwendungen in der Programmiersprache Java ermöglicht, sind keine neuen Entwicklungsumgebungen nötig. Der Entwickler kann die bisher eingesetzte Java IDE weiterhin verwenden. GWT ist kein Framework mit Effektivitätsbremse: Gerade im Bereich Refactoring, Code Completion, Debugging und weiteren Nettigkeiten aus der Java-Welt ist das ein großer Fortschritt.

Ein GWT-Projekt hat ein teilweise vorgeschriebenes Layout. Für die ersten Gehversuche mit GWT lautet die Empfehlung, sich auch tatsächlich an dieses zu halten. GWT liefert ein paar Skripte aus, mit denen Beispielprojekte angelegt werden können.

In den Beispielen und Anleitungen, die mit GWT geliefert werden, wird die Eclipse IDE verwendet. Prinzipiell werden keine Plug-ins benötigt, dennoch existieren bereits GWT Plug-ins, welche die Arbeit an verschiedenen Stellen vereinfachen. Für Eclipse sind der GWT Designer [4] (kommerziell) und Cypal Studio [5] (Open Source, Version RC4) bekannte Plug-ins. Cypal Studio scheint allerdings Probleme mit der aktuellen Eclipse-Version 3.3 zu haben, sodass zu hoffen bleibt, dass ein Update schnell be-

reitgestellt werden kann. Für andere IDEs gibt es auch bereits Erweiterungen [6] für GWT. Beide Eclipse Plug-ins sind bei der Erstellung eines GWT-Projekts (Modul) behilflich, sodass die GWT-Skripte nicht unbedingt eingesetzt werden müssen. Für die Beispiele in diesem Artikel wurde der GWT Designer verwendet.

Eine GWT-Anwendung wird aus der Entwicklungsumgebung heraus im so genannten Hosted Modus ausgeführt. Im Hosted Modus wird ein Tomcat als Webserver und ein spezieller Browser gestartet. Der Java-Bytecode der Anwendung wird ausgeführt – es findet also keine Übersetzung in JavaScript statt: Änderungen im Quelltext (sofern diese dann automatisch von der IDE übersetzt werden) werden sofort in der laufenden Anwendung sichtbar. Über diesen Mechanismus wird das clientseitige Debuggen der Webanwendung im Java-Code ermöglicht. Erst für ein Deployment wird die Anwendung in JavaScript übersetzt.

Die Last mit der Last

Webanwendungen müssen skalierbar sein. Technologien sollten dies 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 muss 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 Skalierungsprobleme schließlich mit weiteren einzusetzenden Technologien auf. So können bereits die Datenbankverbindungen zum Engpass werden oder grobe Program-

mierfehler (zu viele Objekte, zu viele Exceptions) den Server in die Knie zwingen. Aufwändige serverseitige Verarbeitung ist in Bezug auf Performance und Skalierbarkeit kritisch zu betrachten, da der Ressourcenverbrauch entsprechend mit jeder neuen Sitzung steigt.

Die meisten Java-Web-Frameworks haben eines gemeinsam: Sie sind in Java geschrieben und werden serverseitig ausgeführt. Webseiten werden serverseitig erstellt und schließlich dem Browser übertragen.

GWT geht hier einen anderen Weg: die Webanwendung wird einmalig übertragen. Das „Rendern“ der Anwendung findet ausschließlich im Browser statt und spart Ressourcen auf dem Server. Die Kommunikation auf dem Server wird auf ein Minimum reduziert, es werden lediglich Daten hin und her geschickt.

Das Framework

Das mit GWT gelieferte Framework ist recht umfangreich. Hier ein kurzer Überblick über die wichtigsten Eigenschaften.

Das Komponentenmodell

GWT führt ein eigenes UI-Komponentenmodell ein. Neben Swing, SWT und JSF hat der Java-Entwickler jetzt ein neues Komponentenmodell erhalten:

- Widgets sind grafische Komponenten, wie z.B. eine Textbox, ein Button oder eine Checkbox
- Panels sind Widgets, die ganz nach dem Entwurfsmuster des Composite Patterns wiederum Widgets aufnehmen können

Neben den Standardkomponenten, die wir von HTML kennen, finden sich hier auch komplexere UI-Elemente wie etwa Tree, TabBar, SplitPanel und DialogBox. GWT versucht dem Web bzw. den HTML-Komponenten treu zu bleiben. Hierzu ein Zitat aus einer GWT-Präsentation während der Google Developer Days: „Wir wollen unseren Eltern das Web nicht neu beibringen müssen. Es soll nicht anders, sondern nur besser werden.“ Hier grenzt sich GWT von jenen Frameworks ab, die einen Rich Client nachahmen oder ganz neuartige Benutzerführungen anstreben, wie z.B. RAP oder OpenLaszlo.

Das Komponentenmodell von GWT ist erweiterbar. Dank Vererbung können sehr einfach neue eigene Komponenten

MyGWT

MyGWT ist eine Open-Source-Java-Bibliothek für das Google Web Toolkit. Sie stellt optisch ansprechende Widgets zur Verfügung, sodass man sich die Arbeit erspart, eigenes CSS zu erstellen. Die Palette der Widgets reicht dabei von einfachen Buttons bis hin zu Listen, Bäumen und Tabellen. Ganz im Sinne von GWT ist MyGWT vollständig in Java implementiert und verwendet keine externen JavaScript-Bibliotheken. Das API basiert auf Ideen aus dem Eclipse JFace API und erleichtert besonders Eclipse-RCP-Entwicklern den Umstieg auf GWT.

So existiert beispielsweise ein eigener Content Provider und Viewer für Tree oder Table. Die aktuelle Version ist momentan 0.4.x. Mutige können aber auch schon einen ersten Einblick in die Neuerungen der Version 0.5.x bekommen, die als Alpha-Versionen bereitstehen. Die Tatsache, dass die MyGWT-Widgets aktuell noch nicht mit dem GWT Designer Plug-in verwendbar sind (die UI-Komponenten besitzen keinen Default Constructor), bildet im Moment den einzigen Wermutstropfen. Jedoch ist dieses Feature für die Zukunft in der Planung.



Abb. 3: GWT Widgets: MenuBar, TabBar, Tree, Table

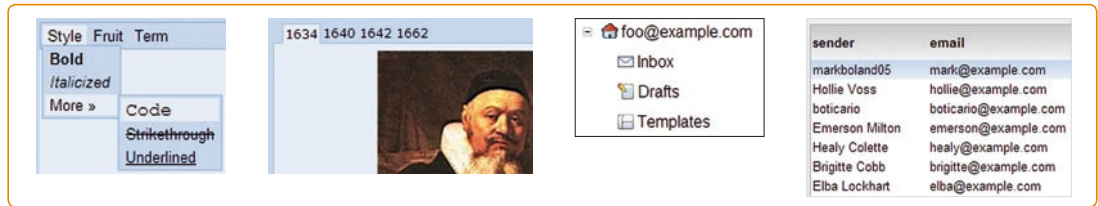
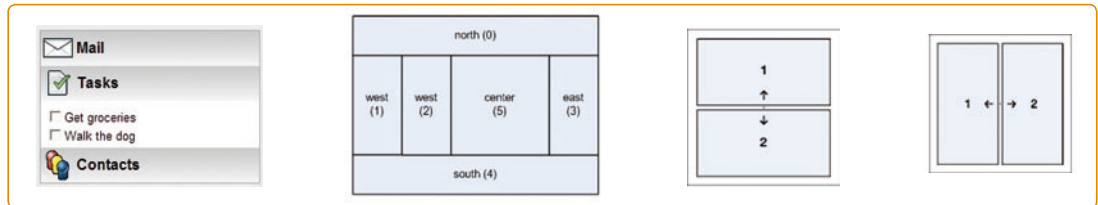


Abb. 4: GWT Panels: StackPanel, DockPanel, VerticalSplitPanel und HorizontalSplitPanel



aus den vorhandenen Komponenten gebildet werden. Es können auch neue Komponenten gebildet werden, die von Grund auf neu sind oder sogar fremdes, natives JavaScript einsetzen. Abbildung 3 zeigt einige Beispiele [8] für Widgets, wie sie in GWT angeboten werden. Abbildung 4 zeigt Beispiele für Panels.

Die Modularisierung

Ein Projekt in GWT wird auch Modul genannt, das über eine Beschreibungsdatei im XML-Format verfügt. In dieser Moduldefinition können verschiedene Einstellungen vorgenommen und unter anderem können auch andere Module importiert werden. Somit können eigene Komponenten in Module ausgelagert und wiederverwendet werden. Gepackt werden diese im üblichen JAR-Format: Java-Quelltext und Bytecode müssen in der JAR-Datei vorhanden sein, da der GWT Compiler den Quelltext und nicht den Bytecode übersetzt.

Das GWT Framework selbst wird durch diesen Mechanismus in ein Modul importiert. Es können beliebig viele solcher Module importiert werden. An dieser Stelle sei noch einmal darauf hingewiesen, dass der GWT Compiler lediglich die Teile der Module übernimmt und übersetzt, die tatsächlich verwendet werden.

Hallo Server!

Da eine GWT-Anwendung prinzipiell eine clientseitige Anwendung ist, bleibt noch zu klären, wie die Kommunikation mit dem Server stattfinden kann. Diese wurde über ein proprietäres – also GWT-spezifisches – Protokoll implementiert. Andere AJAX-Anwendungen verwenden JSON oder XML-RPC für die Kommunikation zwi-

schen Browser und Server. Diese können ebenfalls verwendet werden: Die mitgelieferten Beispiele zeigen, wie man JSON-Aufrufe mit JSNI realisieren kann, und es gibt bereits Open-Source-Bibliotheken, die XML-RPC für GWT implementieren.

Das Besondere am GWT-eigenen Protokoll ist, dass es nahtlos mit den eigenen Java-Objekten verwendet werden kann, sofern diese bestimmten Serialisierungsregeln entsprechen (Implementierung von *java.io.Serializable* usw.).

Nennenswert an dieser Stelle ist, dass alle RPC-Aufrufe asynchron ausgeführt werden. Das Ergebnis wird dann einem dem Aufruf übergebenen Callback-Objekt weitergereicht.

Internationalisierung

Auch GWT kennt ein Konzept für die Internationalisierung einer Webanwendung (Listing 1, 2 und 3). Ähnlich wie in Java

wurde hier auf das Substitutionsprinzip gesetzt. Wie bereits in Java, müssen Texte in separaten Dateien pro Sprache abgelegt werden. Allerdings wurde dies in GWT ein wenig anders realisiert, als es in Java üblich ist. Der Entwickler muss für die in den Dateien abgelegten Textbausteine eine Methode in einem Interface hinterlegen. Textbausteine, die parametrisiert werden können, bekommen Methoden mit der entsprechenden Anzahl von Parametern. GWT sorgt in der Anwendungsentwicklung auch bei der Internationalisierung von Texten für Typsicherheit. Der GWT Compiler überprüft die Textdateien nach ihrer Entsprechung mit den Definitionen des Interfaces. Mit diesem Mechanismus werden Fehler in den Textdateien bereits beim nächsten Kompilierungsvorgang angezeigt.

Eine weitere besondere Eigenschaft der Internationalisierung in GWT ist die

Listing 1

Internationalisierung in GWT, MyMessages.java

```
package de.pgt.gwt.client;

import com.google.gwt.i18n.client.Messages;

public interface MyMessages extends Messages {

    String sayHello();

    String sayHelloTo(String name);

    String accessDenied(int errorCode, String userName,
                        String resource);
}
```

Listing 2

Internationalisierung in GWT, MyMessages.properties

```
sayHello=Hello World!
sayHelloTo=Hello {0}
accessDenied=Error {0}: User {1} does not have
permission to access {2}
```

Listing 3

Internationalisierung in GWT

```
MyMessages messages = (MyMessages)
    GWT.create(MyMessages.class);

Window.alert(messages.sayHelloTo("Papick"));
```

Abb. 5:
Die Projektstruktur

Tatsache, dass diese Sprachdateien aus Gründen der Performanz nicht zur Laufzeit abgefragt werden, sondern dass der GWT Compiler eine Version der Anwendung für jede definierte Sprache erzeugt.

Styling

Nicht alles, was praktisch und durchdacht ist, sieht auch automatisch gut aus. Und hier kommt auch die Stelle, an der Java-Entwickler sich mit GWT schwer tun könnten. An diesem Missstand wird gerade gearbeitet: Mit der nächsten Version 1.5 des Toolkits soll ein besserer CSS-Grundstock ausgeliefert werden,

damit GWT-Anwendungen von Haus aus etwas ansprechender aussehen. Das Aussehen von GWT-Komponenten wird über CSS-Eigenschaften definiert. Wer schöne GWT-Anwendungen schreiben will, muss in der Lage sein, entsprechendes CSS zu zaubern. Auch wenn der Entwickler die JavaScript-Besonderheiten von modernen Browsern nicht mehr wirklich mitbekommt, so wird man dank CSS wieder in die hässliche Realität zurückgeholt.

Wer keinen Dokortitel in CSS-Feintuning besitzt, sollte sich ein gutes Buch zum Thema CSS besorgen oder gleich bei

anderen Projekten abkupfern gehen. Auf der Suche nach Vorlagen oder Support für CSS lohnt sich ebenfalls der Blick auf die Open-Source-Projektlandschaft. Ein starker Kandidat hier ist MyGWT [9], ein vielversprechendes GWT-Projekt mit vielen sinnvollen Erweiterungen, das obendrein auch noch sehr schönes CSS für GWT mitbringt.

Browsergedächtnis

Eines der Probleme mit AJAX/DHTML-Anwendungen ist es, dass es sich in der Browser-History nicht niederschlägt, wenn sich Teile der Anzeige geändert haben. So bleiben die Knöpfe VOR und ZURÜCK im Browser wirkungslos oder im schlimmsten Fall steuern Sie direkt aus der gesamten Anwendung heraus.

GWT bietet eine Abstraktion über die Browser-History. Somit kann der Entwickler diese programmatisch verändern und auch entsprechend reagieren, wenn der Anwender die Knöpfe im Browser bedient. Als Nebeneffekt wird die Anwendung auch noch „bookmarkable“: Der Endanwender kann ein Lesezeichen setzen und die Anwendung wird dann entsprechend bei Aufruf des Lesezeichens reagieren können. Zum Beispiel kann der Entwickler darauf reagieren, dass eine Lasche in einem *TabPanel* selektiert wurde und einen Eintrag in die Browser-History machen. Wenn der Nutzer dann mehrere Laschen anklickt, werden diese in die Browser-History übernommen. Der Nutzer kann dann den BACK-Button bedienen und entsprechend in der Anwendung zurücknavigieren.

Hier ist wichtig, dass der Entwickler entscheidet, bei welchen Aktionen die Browser-History um einen Eintrag erweitert wird und welche Bedeutung es für die Anwendung hat, wenn dieser Eintrag direkt angesteuert wird. Das Reagieren auf die Browsernavigation wird dem Entwickler überlassen, wie in dem Beispiel „HelloHistory“ (Listing 4) gezeigt.

Testen mit JUnit

Die aktuelle Version von GWT liefert eine JUnit-Integration. Diese basiert auf der Version 3.X von JUnit, da GWT aktuell noch keine Java-5-Sprachfeatures unterstützt. Die *TestCase*-Klasse von GWT startet einen unsichtbaren Hosted Mode Browser und bietet spezielle Methoden, die das Testen von asynchronen Aufrufen erleichtern. Da die eigene Anwendung

Listing 4

Internationalisierung in GWT, HelloHistory.java

```
package de.pgt.gwt.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.core.client.GWT;
import com.google.gwt.user.client.History;
import com.google.gwt.user.client.HistoryListener;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.SourcesTabEvents;
import com.google.gwt.user.client.ui.TabListener;
import com.google.gwt.user.client.ui.TabPanel;

/**
 * Entry point classes define <code>onModuleLoad()
 * </code>.
 */
public class HelloHistory implements EntryPoint,
    HistoryListener {

    private TabPanel mainTab;

    public void onModuleLoad() {
        RootPanel rootPanel = RootPanel.get();

        mainTab = new TabPanel();
        mainTab.add(new Label("Panel 1"), "Panel 1");
        mainTab.add(new Label("Panel 2"), "Panel 2");
        mainTab.add(new Label("Panel 3"), "Panel 3");

        mainTab.addTabListener(new TabListener() {
            public boolean onBeforeTabSelected(SourcesTabEvents
                sender,
                int tabIndex) {
                return true;
            }
        });

        public void onTabSelected(SourcesTabEvents sender,
            int tabIndex) {
            GWT.log("Added history token: " + "mainTab" +
                tabIndex, null);
            History.newItem("mainTab" + tabIndex);
        }
    };

    rootPanel.add(mainTab);

    // process initial state. Bookmarks will get handled here.
    History.addHistoryListener(this);

    String initToken = History.getToken();
    if (initToken.length() == 0) {
        initToken = "mainTab0";
    }
    onHistoryChanged(initToken);

    public void onHistoryChanged(String historyToken) {
        GWT.log("History token: " + historyToken, null);
        if (historyToken.startsWith("mainTab")) {
            char charAt = historyToken.charAt(7);
            mainTab.selectTab(Integer.valueOf
                (""+ charAt).intValue());
        }
    }
}
```



in Java geschrieben wird, können diese Klassen dadurch getestet werden.

Ein Browser sagt „Hallo“

Anhand des bekanntesten Anwendungsfalls überhaupt soll kurz gezeigt werden, wie eine GWT-Anwendung aufgebaut wird. Die Standardprojektstruktur einer GWT-Anwendung besteht aus einem Basisverzeichnis und folgenden Unterverzeichnissen bzw. Packages:

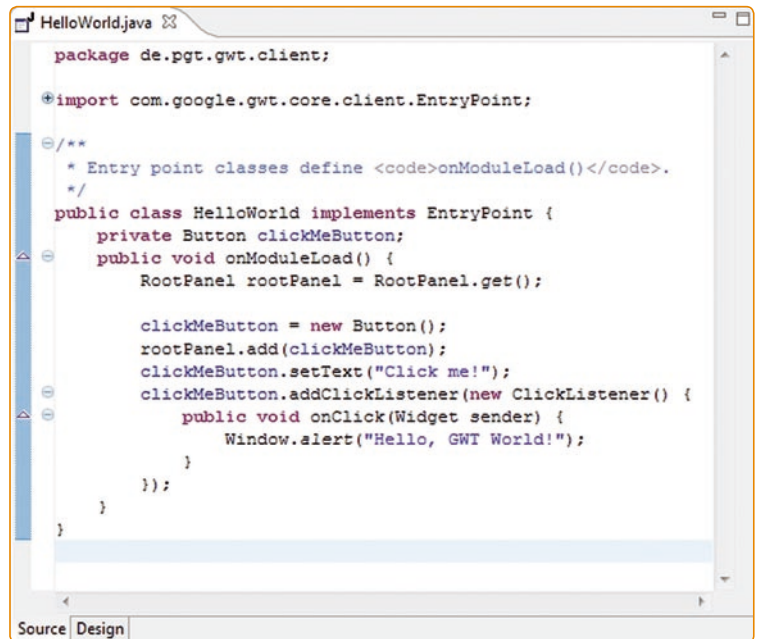
- **CLIENT:** Java-Klassen werden unterhalb dieses Package vom GWT Compiler in JavaScript umgewandelt.
- **SERVER:** Java-Klassen stellen unterhalb dieses Package die serverseitige Logik zur Verfügung (Services) und werden nicht nach JavaScript umgewandelt.
- **PUBLIC:** Alle Dateien, die in PUBLIC abgelegt werden, gehören zu den Ressourcen, die der Webanwendung (also clientseitig) zur Verfügung stehen werden.

Im Basisverzeichnis des GWT-Moduls ist die Modulbeschreibungsdfile abgelegt. Da es sich hier um eine sehr einfache Anwendung handelt, benötigen wir lediglich einen Verweis auf das GWT-User-Modul (die Bibliothek *gwt-user.jar* muss im Klassenpfad liegen) und die Definition des Entry Points. Im GWT-User-Modul sind alle GWT-Klassen abgelegt, die das GWT Framework ausmachen (Abb. 5).

Ein Entry Point besteht aus einer Java-Klasse, die ein bestimmtes Interface implementiert. Diese Klasse wird schließlich beim Start der Anwendung im Browser aufgerufen. In unserem einfachen Beispiel soll ein *Control* vom Typ *Button* angezeigt werden. Wird dieses vom Benutzer betätigt, so soll eine JavaScript-Dialogbox mit einer Begrüßungsnachricht angezeigt werden (Abb. 6).

Letztes Puzzlestück in dieser Minimalanwendung ist die HTML-Datei, die als Eintrittsstelle dienen soll. Die generierte JavaScript-Anwendung muss schließlich aus einer HTML-Seite heraus gestartet werden. Aus diesem Grund werden in dieser HTML-Seite sämtliche CSS-Dateien und das entsprechende Startskript für die GWT-Anwendung eingetragen. Wird die Anwendung gestartet, so werden zwei Fenster angezeigt: Die GWT Development Shell, die den mitgelieferten Tomcat startet, und

Abb. 6: Entry Point



ein GWT-Webbrowser, in der die Webanwendung gestartet wird (Abb. 7).

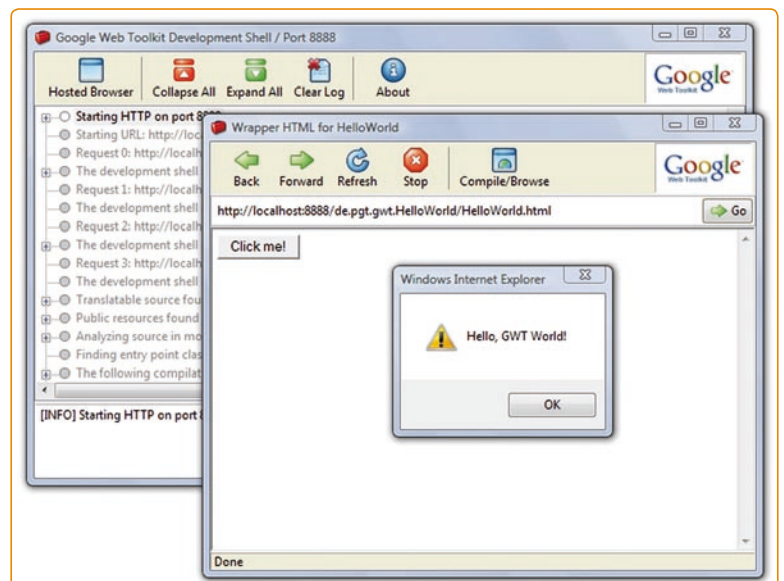
Das Starten der Anwendung wurde direkt von Eclipse aus dem Plug-in GWT Designer ausgelöst. Die mitgelieferten Skripte in der GWT-Distribution erlauben das Erzeugen von Startskripten und Launch-Konfigurationen für Eclipse, auch ohne ein Plug-in. Die genaue Verwendung der Skripte wird auf der Dokumentationsseite des Projekts unter „Getting Started“ [10] Schritt für Schritt erläutert.

Ein Browser ruft an

GWT hat einen eigenen Mechanismus definiert, mit dem der Browser mit dem

Server kommunizieren kann. Im Gegensatz zu den üblichen Protokollen im AJAX-Umfeld ist GWT RPC in der Lage, Java-Objekte zu serialisieren. Dadurch wird die Entwicklung der Client-Server-Kommunikation deutlich vereinfacht. Als Einführungsbeispiel soll ein „Hello-World-“Service angelegt werden. Die Anwendung im Browser soll eine Methode auf dem Server aufrufen und das Ergebnis anzeigen. Der Einfachheit halber wird lediglich ein String übertragen. Für die Implementierung einer Client-Server-Kommunikation mit GWT-RPC werden folgende drei Java-Klassen benötigt:

Abb. 7: „Hello World“ in Action



```

package de.pgt.gwt.client;

import com.google.gwt.user.client.rpc.RemoteService;

public interface SayHelloService extends RemoteService {

    String sayHello();

}

```

Abb. 8: Die Service-schnittstelle

```

package de.pgt.gwt.client;

import com.google.gwt.user.client.rpc.AsyncCallback;

public interface SayHelloServiceAsync {

    void sayHello(AsyncCallback callback);

}

```

Abb. 9: Die asynchrone Service-Schnittstelle

- *SayHelloService* (Schnittstelle)
- *SayHelloServiceAsync* (Schnittstelle)
- *SayHelloServiceImpl* (Klasse)

Die serverseitige Implementierung ist trivial. Ein GWT-Service muss von der Klasse *RemoteServiceServlet* ableiten und die Serviceschnittstelle implementieren. Interessant wird es bei den Schnittstellen. Da die RPC-Aufrufe asynchron ausgeführt werden, muss zusätzlich zur eigentlichen Serviceschnittstelle noch eine weitere, die für den asynchronen Aufruf ausgelegt ist, implementiert werden. An dieser Stelle ist die Erleichterung durch den GWT Designer groß, da das Plug-in die leidige Aufgabe übernimmt, diese Interfaces konsistent zu halten.

Im *SayHelloService*-Beispiel ist nur eine parameterlose Methode definiert. Als Rückgabebetyp ist *String* gesetzt (Abb. 8).

In der asynchronen Schnittstelle hat die entsprechende Methode keinen Rückgabebetyp mehr. Die serverseitige Antwort wird dem *Callback*-Objekt des

Typs *AsyncCallback* zu einem späteren Zeitpunkt übergeben (Abb. 9)

Die Schnittstelle *AsyncCallback* definiert zwei Methoden: *onFailure()* und *onSuccess()*. Übergeben werden entsprechend die ausgelöste Exception bzw. das Ergebnis des serverseitigen Aufrufs. In unserem Beispiel wird zuerst eine Referenz auf den Service erzeugt und konfiguriert. Dem Aufruf wird dann eine Instanz eines *AsyncCallback*-Objekts übergeben (Abb. 9)

Damit der mitgelieferte Tomcat die serverseitige Implementierung auch ausführen kann, muss die Modulbeschreibung noch um einen Servlet-Eintrag erweitert werden. Die Anwendung sieht dann aus wie in Abbildung 10.

Here to stay?

Wie kommerzielle Projekte müssen sich Open-Source-Projekte behaupten und den Marktdruck überleben. Um ein Gefühl für die Chancen eines Projekts zu bekommen, kann man die Konkurrenz in dem bestimmten Segment und die

Community betrachten, die sich um ein Projekt gebildet hat. Im speziellen Fall des Google Web Toolkits spielt sicherlich das „G“ für Google eine Rolle. Und dass GWT den Namenszusatz „Beta“ im Projektamen verloren hat, ist der Sache sicher auch zuträglich.

Viele verschiedene Web Application Frameworks im Java-Umfeld konkurrieren miteinander um die Gunst der Entscheider und Entwickler. So verschieden die Anforderungen auch sind, so unterschiedlich sind auch die Frameworks, die aktuell angeboten werden. AJAX Frameworks sind recht neu, da sie erst jetzt langsam aus dem Prototypstadium in die Produktion wechseln.

Struts2 und Spring MVC sind beides etablierte Frameworks, die ihre Stärken in der Übertragung des Model-View-Controller-Gedankens in die Architektur von Webanwendungen bewiesen haben. Beide Frameworks machen wenig Aussage über die einzusetzende „Rendering Engine“, die Technologie, die letzten Endes das HTML erzeugen wird.

JSF ist wohl das bekannteste komponentenbasierte Framework zur Erstellung von Webanwendungen. Als Standard von Sun in die Java-EE-Spezifikation aufgenommen, profitiert JSF immer mehr durch die so genannte „Akzeptanz durch die Industrie“. Inzwischen ist ein Markt für höherwertige JSF-Komponentenbibliotheken entstanden, die es dem Entwickler ermöglichen, ansprechende Webanwendungen zu gestalten. Technologien und Tools wie Facelets, Spring WebFlow oder das All-In-One-Paket Seam haben Lücken in der Spezifikation erfolgreich gefüllt und machen JSF zu einem mächtigen Framework.

Mit RAP und Echo2 haben wir zwei prominente AJAX/DHTML Frameworks nach dem Konzept Half Object Plus Protocol: Die Anwendung läuft auf dem Server und dem Browser wird dann über AJAX bzw. DHTML der aktuelle Zustand der Anwendung geschickt. Aktionen des Benutzers werden mittels Ajax-Techniken an den Server geschickt, der dann entsprechend die Oberfläche auf dem Browser aktualisiert.

Mit GWT bekommt der Java-Entwickler Zutritt zu einer neuen Art von Webanwendung, die bisher eine Domäne der JavaScript-Entwickler war. Yahoo User Interface Library (YUI), Dojo Toolkit und ExtJS sind bekannte JavaScript-

```

private void justDoIt() {
    final long start = System.currentTimeMillis();
    // get a reference to the asynchronous service
    SayHelloServiceAsync service = (SayHelloServiceAsync) GWT.create(SayHelloService.class);
    // configure service entry point
    ServiceDefTarget target = (ServiceDefTarget) service;
    target.setServiceEntryPoint(GWT.getModuleBaseUrl() + "SayHelloService");
    // make the remote procedure call
    service.sayHello(new AsyncCallback() {
        public void onFailure(Throwable caught) {
            GWT.log("Server rpc error", caught);
        }
        public void onSuccess(Object result) {
            long stop = System.currentTimeMillis();
            rootTable.setWidget(1, 1, new Label("" + (stop - start)));
            rootTable.setWidget(2, 1, new Label(result.toString()));
        }
    });
}

```

Abb. 10: Der asynchrone Aufruf



Bibliotheken, die es dem Entwickler einfacher machen, sehr anspruchsvolle Webanwendungen zu implementieren. Diese Bibliotheken haben das bisherige Konzept der Architektur von Webanwendungen auf den Kopf gestellt: Der Browser wird zur Anwendungslaufzeitumgebung. Der Server ist nicht mehr für das Erzeugen der HTML-Dateien zuständig, da dieser Vorgang nun in JavaScript im Browser stattfindet bzw. ganz entfällt.

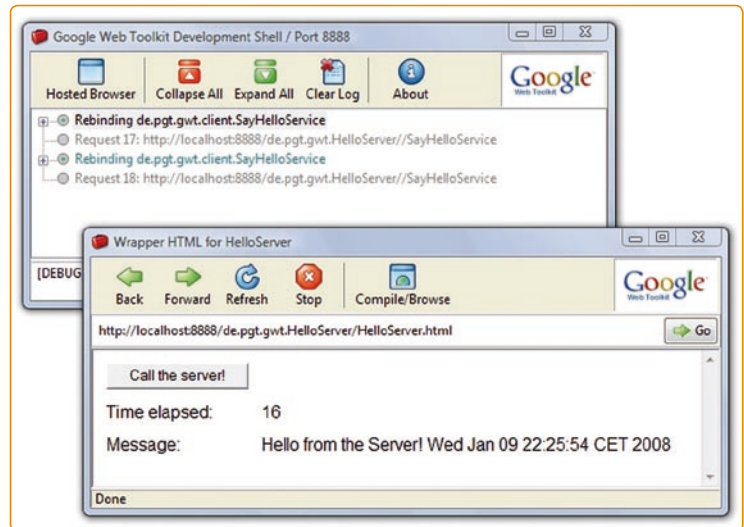
Es bleibt nun folgende Frage offen: Warum sollten dann nicht gleich Applets oder Flash verwendet werden? Das größte Problem mit Applets ist die Startdauer. Nicht nur muss die eigentliche Anwendung übertragen und gestartet werden (wie auch bei allen anderen Technologien, die auf JavaScript oder Flash aufsetzen), hinzu kommt noch der Startvorgang der Java VM. Mit neuen Ideen wird in dieser Ecke gefeilt, kleinere und schneller startende Java VMs werden gerade bei Sun entwickelt und können sogar schon ausprobiert werden. Bei Flash gab es in der Vergangenheit Performanceprobleme mit dem Flash-Player. Für die Java-Gemeinde dürfte aber das neue Programmiermodell bzw. die neue Programmiersprache die größte Einstiegshürde darstellen.

Ein weiterer Grund für JavaScript-Anwendungen ist, dass diese Skriptsprache sich zur Integrationssprache im Internet gemausert hat. Das Buzzword hier heißt „Mashup“ und wird oft im Zusammenhang mit Web 2.0 genannt. Google Maps und viele andere Tools können Dank JavaScript APIs in die eigene Webanwendung integriert werden. Der Browser wird damit zur Integrationsplattform beim Endanwender. Die Infrastruktur der Diensteanbieter wird verwendet, der eigene Server wird entlastet. Das ist das neue Web.

Die Community und Referenzprojekte

GWT selbst bietet zwar eine reichhaltige Palette an Komponenten an, dennoch bleiben immer noch Wünsche offen. Es sind inzwischen unzählige Open-Source-Komponentenbibliotheken entstanden, welche die unterschiedlichsten Erweiterungen für GWT anbieten. Wenn man die Anzahl der Downloads, der Forenbeiträge und der Entwickler beobachtet, wird man ein rasantes Wachstum in der Verbreitung der Technologie und der

Abb. 11:
„HelloWorld“
mit GWT



Gemeinde feststellen. Auch im Bereich der Dokumentation wird zu GWT einiges geboten. Nicht nur ist die GWT-Dokumentation recht umfangreich, es gibt inzwischen ein halbes Dutzend guter Bücher, viele Tutorials, Vorträge und Artikel zu diesem Thema. Google hat zudem einige Vorträge zum Thema GWT in YouTube [11] eingestellt.

Inzwischen gibt es einige Projekte, die erfolgreich mit GWT an den Start gegangen sind. Natürlich verwendet Google selbst die eigene Technologie, wie bei Google Base und Google Checkout. Auf der GWT-Homepage findet sich eine Referenzliste von Projekten. Zum Beispiel findet man dort den Verweis auf ein CRM-System [12] das mit GWT entwickelt wurde.

Fazit

Unumstritten ist Googles Erfahrung mit Web-2.0-Anwendungen – Google Maps und Google Mail haben das Unmögliche möglich gemacht. Dank GWT können jetzt Java-Entwickler an den Browsereigenarten vorbei sauberen JavaScript-Code erzeugen. 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.



Papick G. Taboada ist freiberuflicher Softwarearchitekt und Technology Scout. Schwerpunkte seiner Arbeit liegen in der Konzeption und Erstellung von Softwarearchitekturen im Java-EE-Umfeld mit Open-Source-Technologien. Mit GWT setzt er sich bereits seit 2007 auseinander und hat praktische GWT-Erfahrung in Projekten gesammelt. Seine Erfahrungen gibt er auch als Coach und Trainer weiter. In den vergangenen Jahren hielt er verschiedene Vorträge auf namhaften Konferenzen und veröffentlichte Artikel in anerkannten Fachzeitschriften. Kontakt: pgt@pgt.de.

>> Links & Literatur

- [1] www.code.google.com/webtoolkit
- [2] www.code.google.com/webtoolkit/makinggwtbetter.html
- [3] www.googleblog.blogspot.com/2007/08/google-web-toolkit-towards-better-web.html
- [4] www.instantiations.com/gwt/designer/index.html
- [5] www.cypal.in/studio
- [6] www.code.google.com/webtoolkit/tools.html
- [7] Fast, Easy, Beautiful: Pick Three Building User Interfaces with Google Web Toolkit: <http://code.google.com/webtoolkit/presentations.html#GDD2007>
- [8] www.code.google.com/webtoolkit/documentation/com.google.gwt.doc.DeveloperGuide.UserInterface.html
- [9] www.mygwt.net
- [10] www.code.google.com/webtoolkit/gettingstarted.html
- [11] GWT Conference auf YouTube: www.youtube.com/view_play_list?p=24044DF77EB53015
- [12] www.queplich.com/solutions/queweb-overview