Be dynamic! RichClient-Funktionalitäten mit dem Google Web Toolkit

Julian Gärtner
ORDIX AG
Paderborn

Schlüsselworte:

GWT, AJAX, moderne Web-Applikationen, Remote Procedure Calls, Model View Controller, JavaScript, HTML, CSS

Einleitung

In modernen Web-Anwendungen wird die Nachfrage nach RichClient-Funktionalitäten zur Umsetzung spezieller Anforderungen immer größer. Diese Funktionalitäten sollen dem Nutzer jedoch ohne die Installation von Browser Plugins zur Verfügung stehen. Das Google Web Toolkit ermöglicht die Erstellung von hochdynamischen Web-Applikationen und kann durch eine Vielzahl von Erweiterungen spezielle Anforderungen besser abdecken als andere Frameworks. Dennoch ist gerade bei unterschiedlichen Anforderungsprofilen die Kombination und Integration mit anderen Frameworks von Vorteil. Der Vortrag zeigt anhand eines Beispiels aus der Praxis die erfolgreiche Integration einer RichClient-Applikation basierend auf dem Google Web Toolkit in einer klassischen Web-Anwendung.

Was ist das Google Web Toolkit (GWT)?

Mit dem Google Web Toolkit lassen sich moderne, hochdynamische Webseiten mit asynchroner Kommunikation (AJAX) erstellen. Dabei geht der Entwickler bei der Erstellung von Benutzeroberflächen ähnlich vor, wie er es von Desktop-Applikationen (z. B. Swing) kennt. Er benutzt dabei fast ausschließlich Java als Sprache und benötigt nur ein Minimum an Know-how in den Bereichen HTML, CSS und JavaScript. Der GWT-Compiler übernimmt die Übersetzung von Java in JavaScript/HTML und generiert den für jeden Browser spezialisierten und optimierten Code.

Jede GWT-Applikation besteht lediglich aus einer einzelnen HTML-Seite. Der erzeugte JavaScript-Code sorgt dafür, dass die Ergebnisse aus Benutzerinteraktion und Serverkommunikation durch Manipulation des aktuellen Document Object Models (DOM) im Browser zur gewünschten Darstellung führen.

GWT wird von Google entwickelt und gepflegt, ist eine Open Source Software und unter der Apache 2.0 Lizenz frei verfügbar.

Anforderungen an moderne Web-Applikationen

Oft besteht in der Praxis der Bedarf nach Anwendungen, die unter anderem aus Gründen der Wartungsfreundlichkeit und Kostenersparnis, nicht auf vielen und meistens örtlich verteilten Rechnern

installiert werden können. Das ist das typische Szenario für eine zentral gehostete Server-Anwendung, die über einen allerorts vorhanden Webbrowser aufgerufen werden kann.

Gleichzeitig steigen aber die Anforderungen an die Leistungsfähigkeit solcher Applikationen. Dies betrifft insbesondere die Punkte Benutzerfreundlichkeit und Performance. Web-Applikationen werden hier zunehmend mit nativen Desktop-Anwendungen verglichen und stoßen mit ihrem klassischen Response/Request-Verhalten an technische Grenzen.
Mit der Einführung von asynchronen, nicht an den Response/Request-Zyklus gebundenen, Server-Aufrufen (AJAX), steht eine Technologie zur Verfügung, die das Antwortverhalten von Web-Anwendungen maßgeblich verbessert. Durch Hinzunahme von weiteren JavaScript-Funktionalitäten lassen sich Applikationen erstellen, die sich, bis hin zu Features wie Drag&Drop, Desktop-Applikationen fast vollkommen angenähert haben.

Das Google Web Toolkit ist für die Implementierung solcher Web-Anwendungen entwickelt worden und vereint neben den aufgezählten Features viele weitere in einem einheitlichen Framework.

Von Java zu JavaScript

Um in einem Browser eine Applikation ausführen zu können, muss man sich zwangsläufig JavaScript bedienen. Leider ist die Entwicklung größerer Programme in JavaScript keine triviale Aufgabe. Das liegt sowohl an den leicht inkompatibelen Implementierungen der verschiedenen Browser als auch an deren unterschiedlichen Datenmodellen und Darstellungen.

Zum Glück verbirgt das GWT diese Problematik für den Entwickler nahezu komplett. Er implementiert die gesamte Anwendung in reinem Java. Dazu steht ihm ein Subset des Standard-JDK-Sprachumfangs zur Verfügung. Eine Teilmenge deshalb, weil bestimmte Funktionalitäten nicht auf die JavaScript-Laufzeitumgebung im Browser abzubilden sind (z. B. Threads u. ä.).

Der GWT-Compiler übersetzt dann den Java-Code in JavaScript. Er generiert dabei für jeden unterstützen Browser eine hochoptimierte JavaScript Quelltext-Datei. Beim Start der Applikation, wird dann zunächst ein universelles Stück JavaScript-Code geladen, das den Browser analysiert und identifiziert, um anschließend den passenden, optimierten, browser-spezifischen Code nachzuladen.

Asynchrone Kommunikation

Ein zentraler Aspekt bei der Entwicklung von Applikationen mit dem GWT ist die ausschließlich asynchrone Kommunikation mit dem Server. Die notwendigen Klassen zur Kommunikation liefert die GWT-API für Client und Server mit.

Benötigt ein GWT-Client Daten von einem Server, so fordert er diese mittels eines Remote Procedure Calls (RPC) an. Serverseitig wird die Funktionalität als Servlet implementiert, was auch die Integration anderer Frameworks (JSF/Seam) erleichtert. Die zu übertragenden Daten müssen serialisierbar sein, neben primitiven Datentypen können auch eine ausreichende Menge anderer Strukturen (darunter Arrays) übermittelt werden. Hierbei handelt es sich um ein proprietäres Dateiformat, alternativ kann JSON verwendet werden.

Eine Server-Anfrage erfolgt, wie bereits erwähnt, asynchron. Das impliziert, dass es einen Callback-Mechanismus geben muss, der die angeforderten Daten beim Eintreffen am Client weiterverarbeitet.

Hierbei entstehen die größten Herausforderungen für den Entwickler. Ohne einen strukturierten Plan zur Behandlung der Kommunikationsabläufe droht schnell ein nicht beherrschbares Chaos.

Doch nicht nur die externe Kommunikation stellt hohe Anforderungen an die Architektur (siehe auch Best Practices) einer GWT-Applikation. Auch die internen Ereignisse wie Mausklicks, Tastatureingaben oder Timer-Signale werden asynchron geliefert und müssen über entsprechende Handler verarbeitet werden. Das Observer-Pattern spielt in diesem Zusammenhang in jeder GWT-Anwendung eine zentrale Rolle.

Model-View-Controller

Bei der Entwicklung von Benutzeroberflächen hat sich das Model-View-Controller Paradigma bewährt. Entwickelt für Desktop-Applikationen im Umfeld der Sprache Smalltalk, ist es immer noch aktuell und findet in teils leicht abgewandelten Formen auch in Web-Applikationen und Frameworks wie GWT Verwendung.

Ziel ist die Trennung von Haltung und Darstellung der Daten. Man möchte Abhängigkeiten reduzieren, einen hohen Wiederverwendungsgrad erreichen und eine gute Erweiterbarkeit sicherstellen. Das Modell repräsentiert dabei den Zustand der Daten, während die View für deren Darstellung zuständig ist. Der Controller ist das Bindeglied zwischen beiden. Er benachrichtigt die View, sobald die Daten an der Oberfläche aktualisiert werden müssen und reagiert z. B. auf Tastatureingaben, die eine Veränderung des Datenmodells auslösen.

Innerhalb einer GWT-Applikation funktioniert das sehr ähnlich zu den Mechanismen, wie man sie beispielsweise aus Swing kennt. Die Verwendung des Observer-Patterns steht hier ebenfalls im Mittelpunkt. Für die korrekte Synchronität ausgelagerter Daten auf Server-Ebene mit der Benutzeroberfläche sind naturgemäß noch weitere Maßnahmen erforderlich.

Entwicklungsumgebung

Das GWT wird in Form eines SDKs zur Verfügung gestellt, das alle APIs, einige Tools und den GWT-Compiler enthält. Es ist standalone verwendbar. Zusätzlich existiert ein Eclipse-Plugin, welches das GWT nahtlos in die IDE integriert. Dieses verfügt über Assistenten, die initiale GWT-Projekte anlegen oder eine Entwickler-Shell für den Test von GWT-Applikationen konfigurieren.

In der Entwicklungsphase werden die Applikationen üblicherweise im Hosted-Mode ausgeführt. Es wird dazu ein modifizierter Browser verwendet, der ohne Hilfsmittel ein direktes Debugging innerhalb der Entwicklungsumgebung ermöglicht. Nachteil dieses Vorgehens ist jedoch der Verzicht auf diverse Tools, die ein Entwickler zur Unterstützung im Browser gerne einsetzt (z. B. Firebug, DeveloperToolBar, ...).

Alternativ existieren für diverse Browser Plugins, die die Ausführung von Applikationen im sog. Development-Mode ermöglichen. Damit ist auch das transparente Debugging im jeweiligen Browser kein Problem.

Debugging bedeutet in diesem Zusammenhang tatsächlich das Debuggen des Java-Quelltextes einer JavaScript-Applikation, wie man es von nativen Java-Anwendungen gewohnt ist.

Sonstige Features

Neben den bereits aufgezählten Eigenschaften verfügt das GWT noch über eine Vielzahl anderer Features.

Anwendungen lassen sich mittels Resourcen-Bundles leicht internationalisieren. Die Code-Optimierung reicht soweit, dass bei der Verwendung einer bestimmten Sprache die Teile einer nicht benötigten Sprache erst gar nicht geladen werden.

Benutzeroberflächenkomponenten sind in anderen Projekten wiederverwendbar. Die Erstellung von sog. Widgets garantiert eine gekapselte Code-Erzeugung und damit eine korrekte, einheitliche Darstellung.

GWT-Code bis hin zu den Remote Procedure Calls kann mittels JUnit getestet werden.

Serverseitig ist man nicht auf Java festgelegt. Die eingesetzten Techniken und Protokolle erlauben die Verwendung anderer Sprachen wie z. B. PHP.

Die Integration zusätzlicher JavaScript-Bibliotheken ist möglich. Mittels des JSNI (JavaScript Native Interface) lässt sich ein eigener JavaScript-Code einbinden.

Die Unterstützung des Zurück-Buttons und der Historie im Browser ist durch eine transparente Zustandsverwaltung möglich.

Aufgabenstellung im konkreten Projekt

Bei der im Folgenden darzustellenden GWT-Anwendung handelt es sich um ein Teilprojekt aus einer größeren medizin-technischen Applikation, deren Hauptaufgabe die Überwachung und Visualisierung von Infusionspumpen in einem Krankenhaus ist.

Ein Krankenhausbett kann vereinfacht beschrieben mit einer variablen Anzahl von Infusions- und Spritzenpumpen ausgestattet sein, die einen Patienten mit Medikamenten versorgen. Diese Pumpen liefern an einen zentralen Server kontinuierlich Laufzeitdaten über Medikamente, Fördermengen, Förderzeiten und Alarmzustände.

Die Applikation soll nun verschiedene Grundrisse mit eingezeichneten Zimmern, Betten und Pumpen visualisieren und so dem Krankenhauspersonal einen schnellen Überblick über den Zustand der überwachten Betten ermöglichen.

Zusätzlich soll die Software einen Administrationsteil erhalten, mit dem die Grundrisse inkl. Zimmer und Betten (unter Verwendung von Drag&Drop-Funktionalitäten) konfiguriert und bearbeitet werden können.

Als übergeordnete Anforderung gilt: Auf den Client-Rechnern existiert außer einem Firefox oder Internet Explorer keine weitere Software, die verwendet werden kann. Zudem darf keinerlei zusätzliche Software zum Betrieb der Applikation installiert werden (auch keine Browser-Plugins).

Das komplette Projekt, sollte ohne Lizenzkosten für Tools, APIs, etc. auskommen.

Entscheidungskriterien für GWT

Aufgrund der Anforderungen (Drag&Drop, Zeichnen der Grundrisse mit Browser-Bordmitteln) war schnell klar, dass nur ein JavaScript-basiertes Framework in Frage kam. Da serverseitig ein J2EE-Umfeld mit JSF/Seam unter JBoss 4.x anzubinden war, wurden folgende Punkte als wichtigste Kriterien definiert:

- problemlose Integration in JSF/Seam/JBoss (insbesondere Kommunikationswege)
- ausreichend komfortable Drag&Drop-Funktionalität
- möglichst geringer Aufwand zur Einarbeitung
- möglichst keine großen Technologiebrüche

Die erste Überlegung, auch diese Applikation mit JSF/RichFaces zu implementieren, wurde wegen der ungenügenden Drag&Drop-Unterstützung fallen gelassen. Der Gedanke eine zusätzliche, reine JavaScript-Library einzusetzen erzeugte Unbehagen, da JavaScript-Know-how nur rudimentär vorhanden war und auch keine Erfahrungen mit bestimmten Libraries existierten.

Aber auch den Einsatz von GWT wurde nicht ohne Vorbehalte betrachtet. Zwar empfand man es als großen Vorteil nahezu komplett mit Java arbeiten zu können, allerdings fragte man sich, wie ausgereift (zu diesem Zeitpunkt war GWT 1.3 aktuell) die ganze Sache war.

Um eine Entscheidung herbeiführen zu können, entschied man sich einen Prototypen zu entwickeln, der speziell die o. g. Punkte überprüfen sollte. Überraschender Weise stellten sich die beiden ersten Punkte (Anbindung und Drag&Drop) als relativ problemlos heraus. Einzig die kleinen Inkompatibilitäten bei der Darstellung im Internet Explorer und Firefox hinterließen einige Skepsis.

Die Einarbeitung in GWT und die API blieb natürlich. Das Problem konnte aber auch dank der relativ guten Dokumentation in Grenzen gehalten werden. Bei Verwendung einer JavaScript-Library hätte man ebenfalls eine gewisse Lernkurve bewältigen müssen, jedoch wäre der Technologiebruch hier wahrscheinlich noch größer gewesen.

Best Practices

Kommunikation

Nachdem die Entscheidung für GWT gefallen war und die Anwendung eine gewisse Größe angenommen hatte wurde schnell klar, dass durch die asynchrone Arbeitsweise, die Menge und das Zusammenspiel der diversen Listener und Handler nicht mehr überschaubar war.

Ebenso wurde die Menge der unterschiedlichen RPCs unterschätzt, die ja immer ein Gegenstück auf der Server-Seite verlangte.

In einem ersten Refactoring-Schritt wurde also für die RPCs ein Command-Pattern und für die asynchrone Kommunikation ein Event-Bus eingeführt.

Das Command-Pattern bewirkt eine einheitliche, generische Schnittstelle zum Server. Über diese Schnittstelle wird ein Datensatz gesendet, der die eigentliche Aktion und deren Parameter enthält. Der Datensatz wir auf Serverseite analysiert und an die passende Methode delegiert. Das Ergebnis wird ebenfalls in einem einheitlichen Datensatz mit spezifischem Antworttyp gefüllt und auf Empfängerseite wiederum analysiert und entsprechend zurück delegiert.

Der Event-Bus verhindert ein wildes Durcheinander von Beziehungen zwischen Listenern und Handlern. Er verwaltet alle Beziehungen zentral und ist die einzige Anlaufstation für alle Komponenten. Dazu wird für jedes vorkommende Ereignis ein Typ definiert. Jede Komponente, die sich für ein Ereignis interessiert registriert sich am Event-Bus. Jede Komponente, die ein Ereignis auslöst, signalisiert dieses Ereignis ausschließlich dem Event-Bus. Dieser wird sowohl für die clientseitigen, internen Ereignisse (Mausklicks, Tastureingaben, Timer, ...) benutzt, als auch für die asynchron eintreffenden Ergebnisse der RPCs (Callbacks).
Der Nebeneffekt dieser Architekturverbesserung ist, dass man jetzt ein Rezept aufstellen kann, um bestimmte Funktionalitäten zu implementieren, denn alle Serveraufrufe hatten jetzt eine einheitliche Schnittstelle und wurden auf beiden Seiten jeweils nach dem gleichen Schema behandelt. Neu ins Team gekommene Kollegen konnten sich schnell der Mechanismen bedienen.

Anbindung JSF/Seam

Für die Standard-Anwendungen waren keine besonderen Maßnahmen notwendig. So funktionierte das normale Sessionhandling wie gewohnt. Für die Behandlung von Sonderfällen (z. B. Übergabe von zusätzlichen Daten) lässt sich ein relativ einfacher Mechanismus implementieren.

Ziel ist es Daten in den JavaScript-Client zu bekommen. Dazu wird die GWT-Seite (daraus besteht die ganze Applikation) in den JSF-Kontext eingebunden („eine xhtml-Seite draus machen“). Über eine JSF-Expression wird jetzt ein Stück JavaScript-Code generiert, das eine globale (JavaScript-)Variable mit den gewünschten Werten definiert. Diese globale Variable kann jetzt von der GWT-Applikation verwendet werden.

Browser

Obwohl nahezu alle Unterschiede der Browser von dem generierten JavaScript-Code korrekt behandelt werden, ergeben sich doch manchmal Situationen, die es erfordern, dass man eigene CSS-Styles einbinden muss. Dabei kommt es zwangsläufig zu kleineren Darstellungsfehlern.

Hier hat sich der Einsatz der DeveloperToolBar (IE) und des Firebug-Plugins (Firfox) bewährt. Ohne diese Hilfsmittel ist eine effektive Fehlersuche quasi unmöglich. Leider muss man sich an dieser Stelle doch noch mit CSS/HTML-Themen auseinandersetzen.

Performance

Die Ablaufgeschwindigkeit im Client selbst, d. h. die Leistung des JavaScript-Interpreters, war ausreichend. Es konnten keine Limitierungen festgestellt werden. Ein anderer Aspekt gibt aber durchaus Anlass, dieses Thema genauer zu betrachten.

Durch die asynchrone Kommunikation mit dem Server können je nach Situation relativ viele RPCs in sehr kurzen Zeitabständen auftreten. Hierbei können Timeouts und Racing Conditions auftreten, was den Eindruck einer blockierten oder langsamen Anwendung erzeugen kann. Ja nach Anwendungsfall müssen hier optimierte Methoden zur Nachrichtenverarbeitung eingesetzt werden. Für eine Untersuchung des Antwortverhaltens hat sich das Firefox Plugin Firebug bewährt.

Fazit

Die Implementierung hat gezeigt, dass das Google Web Toolkit eine effiziente Entwicklung von modernen Web-Anwendungen ermöglicht. Der größte Nutzen liegt in der Kapselung der eigentlichen Web-Technologien HTML/CSS/JavaScript und dem Verbergen der damit verbundenen Komplexität. Für Java Entwickler, die aus dem Desktop-Bereich kommen, ist die Umstellung zur Entwicklung von Web-Anwendungen relativ leicht. Auch die Integration anderer Frameworks fällt leicht.

Der asynchronen Kommunikation bleibt geschuldet, dass Anwendungen noch strukturierter mit einer sauberen Architektur entworfen werden müssen. Hier ist sicherlich mit einem zusätzlichen Aufwand zu rechnen. Ebenso bleibt die Fehlersuche teils schwierig.

Alles in allem überwiegen aber die Vorteile und das Gesamturteil fällt deutlich positiv aus.

Kontaktadresse:

Julian Gärtner
ORDIX AG
Westernmauer 12 - 16
D-33098 Paderborn

Telefon: +49 (0) 5251-10630
Fax: +49 (0) 180-1673490
E-Mail: info@ordix.de
Internet: www.ordix.de