Write once, render anywhere

Mobile Apps

Harald Pehl

Mit dem Slogan „Write once, run anywhere" startete für Java in den 90er Jahren ein unvergleichlicher Siegeszug. Dabei lag der Fokus zunächst auf dem Client. Im Laufe der Zeit verlagerte sich die Programmierung allerdings immer mehr hin zum Server. Mit dem Aufkommen der unterschiedlichen Smartphones und Tablets wird es für den Entwickler wieder interessant, eine Codebasis für die verschiedenen Plattformen zu haben. Mithilfe des Google Web Toolkits (GWT) und dem Einsatz des Model-View-Presenter (MVP)-Musters rückt dieses Ziel in greifbare Nähe.

Will man heutzutage Anwendungen entwickeln, die auf möglichst vielen unterschiedlichen Plattformen laufen, sieht man sich mit einer Vielzahl von Programmiersprachen und Entwicklungswerkzeugen konfrontiert. Apples iOS erfordert eine kostenpflichtige Registrierung und das Erlernen einer für Java-Entwickler eher fremdartigen Sprache. Android und Blackberry bieten zwar die Möglichkeit, Anwendungen in Java zu erstellen, die beiden Systeme unterscheiden sich aber grundlegend im Aufbau.

Einen Ausweg aus diesem Dilemma stellen Webanwendungen dar, die an die jeweilige Plattform angepasst sind und sich wie native Applikationen bedienen lassen. Laut einer Studie von Developer Economics [DE11] liegen Webanwendungen mit 56 % in der Gunst der Entwickler bereits an dritter Stelle hinter Android (67 %) und iOS (59 %). Das liegt auch daran, dass mittlerweile alle aktuellen Smartphones und Tablets moderne Browser an Bord haben. Dadurch können bei der Entwicklung Features wie lokale Datenspeicherung, Geolocation oder Canvas genutzt werden. Die eigentliche Implementierung der Anwendung erfolgt in der Regel in JavaScript.

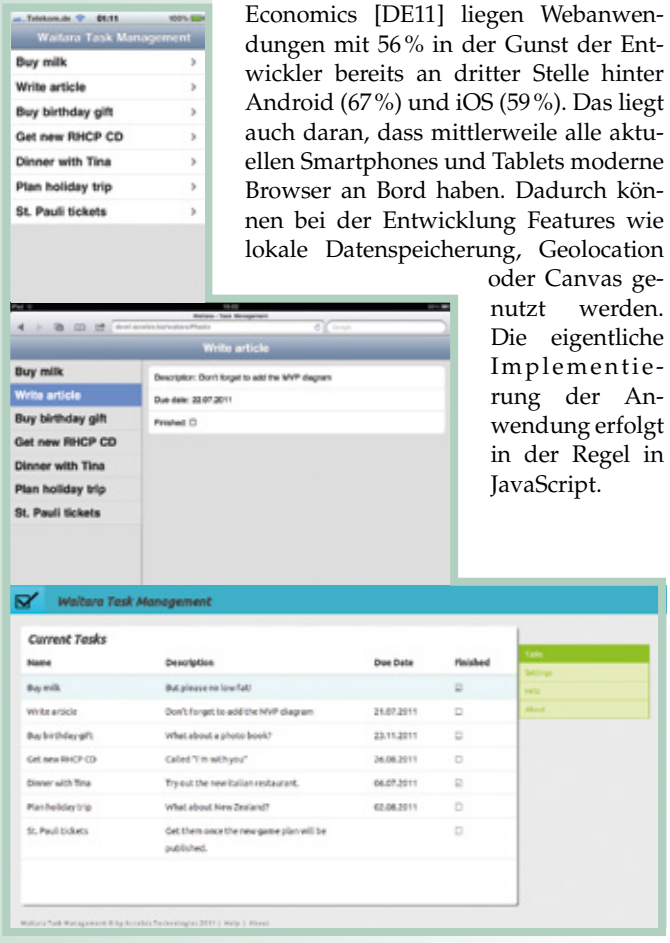

Abb. 1: Render anywhere: iPhone, iPad, Desktop

Nun ist die Programmierung in JavaScript nicht jedermanns Sache. Deshalb soll im Folgenden gezeigt werden, wie Webanwendungen auch mithilfe von Java erstellt werden können. Ziel dabei ist es, so viel gemeinsamen Code wie möglich und so wenig unterschiedlichen Code wie nötig für die jeweilige Plattform zu schreiben.

Als Beispiel dient eine einfache Taskverwaltung, die je nach Plattform eine andere Oberfläche bietet. Abbildung 1 zeigt die Anwendung auf dem iPhone, dem iPad und auf dem Desktop. Das Beispiel ist unter `http://devel.accelsis.biz/waitara/` erreichbar, der Quellcode kann unter [AccWaitara] heruntergeladen werden.

Architektur & Aufbau

Die Beispielanwendung wurde mithilfe von GWT und weiteren Frameworks erstellt (siehe Kasten „Tools & Frameworks"). Weiterhin kommt das Model-View-Presenter (MVP)-Muster zum Einsatz. MVP ist aus dem Model-View-Controller-Muster entstanden, mit dem Ziel, das Modell vollständig von der Ansicht zu trennen.

Der Controller wird hier Presenter genannt und übernimmt die komplette Steuerung der Ansicht und die Aktualisierung des Modells. Dabei sollte der Presenter nach Möglichkeit keinen UI-Code enthalten. Das ist insbesondere für GWT-Anwendungen wichtig, damit der Presenter einfach Unit-Tests unterzogen werden kann. Die Ansicht wiederum sollte nur UI-

Tools & Frameworks

- *GWT* wurde von Google erstmals im Mai 2006 veröffentlicht und liegt aktuell in der Version 2.4 vor. Das Herzstück von GWT ist ein Java-zu-JavaScript-Compiler. Dieser sorgt dafür, dass Anwendungen komplett in Java erstellt werden können und zu JavaScript übersetzt werden. Neben dem Compiler bietet GWT eine umfangreiche Bibliothek mit Klassen zur Oberflächengestaltung und Client-Server-Kommunikation. Beginnend mit der Version 2.0 wurden weitere Bereiche abgedeckt. Dazu zählen die Unterstützung von loser Kopplung durch einen Event-Bus, die Verwendung von MVP und die Trennung von Layout und Code per `UiBinder`.
- *GWTP* ist ein Open-Source-Framework für GWT, das die Erstellung MVP-basierter Anwendungen stark vereinfacht. Dazu verwendet GWTP eine Reihe von Features aus GWT und setzt sie auf eine solide Basis. Vor allem bei der Entwicklung größerer GWT-Projekte zahlt sich der Einsatz von GWTP aus. So kann z. B. durch den Einsatz einer einfachen Annotation sichergestellt werden, dass Teile der Anwendung erst bei deren Verwendung vom Server nachgeladen werden. Die Homepage von GWTP ist unter [GWTP] erreichbar.
- *GIN* (*GWT IN*jection) ist ein Dependency-Injection-Framework für GWT, das sich an Google Guice orientiert. So können mithilfe von GIN Abhängigkeiten mit der aus CDI bekannten Annotation `@Inject` injiziert werden. Weitere Informationen zu GIN sind unter [GIN] erreichbar.

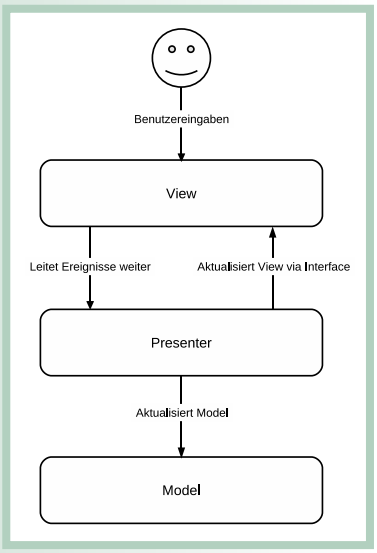

Abb. 2: Model-View-Presenter-Muster

Code und keine Steuerungslogik enthalten. Abbildung 2 zeigt die Verantwortlichkeiten und Bestandteile in einer MVP-Architektur.

Durch den Einsatz von MVP kann möglichst viel Code für alle Plattformen wiederverwendet werden. So sind in der Taskverwaltung das Modell und der Presenter für alle Plattformen identisch. Lediglich die unterschiedlichen Ansichten müssen pro Plattform implementiert werden. Allerdings kann hier eine gemeinsame Oberklasse verwendet werden. Das eigentliche Layout wird dann durch ein **UiBinder**-Template umgesetzt, das in der konkreten View-Klasse referenziert wird.

Listing 1 zeigt das Template für den Desktop, Listing 2 die Variante für das Smartphone. Das **UiBinder**-Template ist die zentrale Stelle, in der alle plattformspezifischen Elemente enthalten sind. Es übernimmt die folgenden Aufgaben:

- Beschreiben der HTML-Struktur,
- Referenzieren weiterer Views und Widgets,
- Einbinden plattformspezifischer Stylesheets.

Wie im Beispiel zu sehen, sind die Templates sehr unterschiedlich umgesetzt. So wird für den Desktop ein Bereich für die Navigation und die Statusleiste erzeugt, der in der Smartphone-Variante komplett fehlt. Ob und wenn ja welche weiteren Ansichten eingebunden werden, ist also von der Plattform abhängig. Die Stylesheets sind in separaten Dateien ausgelagert und werden im jeweiligen **UiBinder**-Template verwendet und gegebenenfalls verfeinert.

```
<!DOCTYPE ui:UiBinder SYSTEM "http://dl.google.com/gwt/DTD/xhtml.ent">
<ui:UiBinder xmlns:ui="urn:ui:com.google.gwt.uibinder"
  xmlns:g="urn:import:com.google.gwt.user.client.ui">

  <ui:with field="nameTokens"
          type="biz.accelsis.waitara.client.NameTokens" />
  <ui:style>...</ui:style>
  <g:DockLayoutPanel unit="PX">
    <g:north size="48">
      <g:HTMLPanel>
        <header>
          <h1>Waitara Task Management</h1>
        </header>
      </g:HTMLPanel>
    </g:north>
    <g:south size="20">
      <g:HTMLPanel>
        <footer>
          Waitara Task Management &copy; by
          <a href="http://www.accelsis.biz">Accelsis Technologies</a>
          2011 | 
          <g:InlineHyperlink targetHistoryToken=
            "{nameTokens.help}">Help</g:InlineHyperlink>
           | 
          <g:InlineHyperlink targetHistoryToken=
            "{nameTokens.about}">About</g:InlineHyperlink>
        </footer>
```

➔

```
      </g:HTMLPanel>
    </g:south>
    <g:east size="250">
      <g:HTMLPanel>
        <nav>
          <g:InlineHyperlink ui:field="tasks" targetHistoryToken=
            "{nameTokens.tasks}">Tasks</g:InlineHyperlink>
          <g:InlineHyperlink ui:field="settings" targetHistoryToken=
            "{nameTokens.settings}">Settings</g:InlineHyperlink>
          <g:InlineHyperlink ui:field="help" targetHistoryToken=
            "{nameTokens.help}">Help</g:InlineHyperlink>
          <g:InlineHyperlink ui:field="about" targetHistoryToken=
            "{nameTokens.about}"
            addStyleNames="{style.noBorder}">About</g:InlineHyperlink>
        </nav>
      </g:HTMLPanel>
    </g:east>
    <g:center>
      <g:ScrollPanel ui:field="mainPanel"
        addStyleNames="{style.mainPanel}"></g:ScrollPanel>
    </g:center>
  </g:DockLayoutPanel>
</ui:UiBinder>
```

Listing 1: UiBinder-Template, Desktop

```
<!DOCTYPE ui:UiBinder SYSTEM "http://dl.google.com/gwt/DTD/xhtml.ent">
<ui:UiBinder xmlns:ui="urn:ui:com.google.gwt.uibinder"
  xmlns:g="urn:import:com.google.gwt.user.client.ui">

  <ui:with field="nameTokens"
    type="biz.accelsis.waitara.client.NameTokens" />
  <ui:image field="headerBackground"
    repeatStyle="Horizontal" src="../../resources/header.png" />
  <ui:style>...</ui:style>
  <g:DockLayoutPanel unit="PX">
    <g:north size="41">
      <g:HTMLPanel>
        <header>
          <h1 ui:field="header">Waitara Task Management</h1>
          <g:InlineHyperlink ui:field="backToTaskList"
            targetHistoryToken="{nameTokens.tasks}"
            addStyleNames=
              "{style.backToTaskList}">Tasks</g:InlineHyperlink>
        </header>
      </g:HTMLPanel>
    </g:north>
    <g:center>
      <g:SimpleLayoutPanel ui:field="mainPanel"></g:SimpleLayoutPanel>
    </g:center>
  </g:DockLayoutPanel>
</ui:UiBinder>
```

Listing 2: UiBinder-Template, Smartphone

Deferred Binding

Woran erkennt die Anwendung nun, welche Ansicht und damit welches **UiBinder**-Template zu verwenden ist? Oder anders formuliert: Woher weiß die Webanwendung, auf welcher Plattform sie läuft? Der Schlüssel liegt im sogenannten Deferred Binding. Dies ist ein sehr mächtiges Konstrukt in GWT, das u. a. für die folgenden Bereiche eingesetzt werden kann:

- als Code-Generator für die Erzeugung von Implementierungen bestimmter Interfaces,
- als Reflection-Ersatz zur Übersetzungszeit,
- zur Auswahl bestimmter Klassen an Hand von bestimmten Eigenschaften.

Innerhalb von GWT selbst wird Deferred Binding intensiv verwendet, um u. a. die Unterschiede der verschiedenen Desktop-Browser hinter einer einheitlichen Programmierschnittstelle zu

kapseln. So werden z. B. im GWT-Modul **com.google.gwt.user.Popup** zwei Implementierungen für Popups definiert:

```
<replace-with class="com.google.gwt...PopupImplMozilla">
  <when-type-is class="com.google.gwt...PopupImpl"/>
  <when-property-is name="user.agent" value="gecko1_8"/>
</replace-with>
<replace-with class="com.google.gwt...PopupImplIE6">
  <when-type-is class="com.google.gwt...PopupImpl"/>
  <when-property-is name="user.agent" value="ie6"/>
</replace-with>
```

Diese Konfiguration ist folgendermaßen zu lesen: „Wann immer die Klasse **PopupImpl** referenziert wird und die Anwendung läuft in Mozilla, verwende die Klasse **PopupImplMozilla**. Falls die Anwendung im IE6 läuft, verwende stattdessen die Klasse **PopupImplIE6**." Damit das Ganze funktioniert, darf die Klasse **PopupImpl** nicht per Konstruktor, sondern muss mithilfe von **GWT.create** erzeugt werden:

```
// Liefert die richtige Instanz abhängig vom verwendeten Browser
PopupImpl impl = GWT.create(PopupImpl.class);
```

Die Methode **GWT.create** ist der Hinweis an den GWT-Compiler, nach Möglichkeit Deferred Binding zu verwenden. Falls keine entsprechenden Regeln definiert wurden, wird einfach auf den Default-Konstruktor zurückgegriffen.

Diesen Umstand machen wir uns auch in unserer Beispielanwendung zunutze. Listing 3 zeigt das GWT-Modul **biz.accelsis.waitara.FormFactor**. In diesem Modul definieren wir die Eigenschaft **formfactor** und deren mögliche Werte. Innerhalb des Elements **<property-provider>** können wir dazu mithilfe von JavaScript den Wert berechnen. Um die verschiedenen Plattformen auch vom Desktop aus testen zu können, werten wir als Erstes einen bestimmten Request-Parameter aus. Nur wenn dieser Parameter nicht gesetzt wurde, greifen wir auf die Eigenschaft **UserAgent** zurück.

Für Android unterscheiden wir anhand der Displaygröße, ob es sich um ein Smartphone oder ein Tablet handelt. Weitere Plattformen wie Blackberry oder WebOS werden im Beispiel nicht explizit ausgewertet, lassen sich aber leicht integrieren.

Das GWT-Modul der eigentlichen Anwendung benutzt nun das Modul **biz.accelsis.waitara.FormFactor**, um die entsprechenden Ansichten auszuwählen (s. Listing 4).

```
<?xml version="1.0" encoding="UTF-8"?>
<module>
  <define-property name="formfactor" values="desktop,tablet,mobile"/>
  <collapse-property name="formfactor" values="*"/>

  <property-provider name="formfactor">
  <![CDATA[
    // Look for the formfactor as a url argument.
    var args = location.href;
    var start = args.indexOf("formfactor");
    if (start >= 0) {
      var value = args.substring(start);
      var begin = value.indexOf("=") + 1;
      var end = value.indexOf("&");
      if (end == -1) {
        end = value.indexOf("#");
        if (end == -1) {
          end = value.length;
        }
      }
      return value.substring(begin, end);
    }
    // Detect form factor from user agent.
    var ua = navigator.userAgent.toLowerCase();
    if (ua.indexOf("iphone") != -1 || ua.indexOf("ipod") != -1) {
      return "mobile";
    } else if (ua.indexOf("ipad") != -1) {
      return "tablet";
    } else if (ua.indexOf("android") != -1 ||
              ua.indexOf("mobile") != -1) {
      var dpi = 160;
      var width = $wnd.screen.width / dpi;
      var height = $wnd.screen.height / dpi;
      var size = Math.sqrt(width*width + height*height);
      return (size < 6) ? "mobile" : "tablet";
    }
    return "desktop";
  ]]>
  </property-provider>
</module>
```

Listing 3: FormFactor-Modul

```
<?xml version="1.0" encoding="UTF-8"?>
<module rename-to="waitara">
  ...
  <inherits name="biz.accelsis.waitara.FormFactor" />
  <entry-point class="biz.accelsis.waitara.client.Waitara" />

  <replace-with class=
"biz.accelsis.waitara.client.application.view.ApplicationViewDesktop">
    <when-type-is class=
    "biz.accelsis.waitara.client.application.view.ApplicationView" />
  </replace-with>
  <replace-with class=
"biz.accelsis.waitara.client.application.view.ApplicationViewTablet">
    <when-type-is class=
     "biz.accelsis.waitara.client.application.view.ApplicationView" />
    <when-property-is name="formfactor" value="tablet" />
  </replace-with>
  <replace-with class=
"biz.accelsis.waitara.client.application.view.ApplicationViewMobile">
    <when-type-is class=
    "biz.accelsis.waitara.client.application.view.ApplicationView" />
    <when-property-is name="formfactor" value="mobile" />
  </replace-with>
  ...
</module>
```

Listing 4: Anwendungsmodul

Startseite

Wir können nun bestimmen, welche Ansichten für welche Plattformen verwendet werden. Was noch fehlt, ist die optimale Integration der Anwendung in die jeweilige Plattform. Dafür gibt es eine Reihe von Meta-Tags und anderen Elementen, die in die Startseite der Anwendung notiert werden sollten. Listing 5 zeigt den entsprechenden Code in Aktion.

```
<!DOCTYPE html>
<html>
<head>
<!-- Icons -->
<link rel="icon" href="task_fav.png" type="image/png" />
<link rel="apple-touch-icon" href="task_57x57.png" />
<link rel="apple-touch-icon" sizes="72x72" href="task_72x72.png" />
<link rel="apple-touch-icon" sizes="114x114" href="task_114x114.png" />
<link rel="apple-touch-startup-image"
      href="task_splash.png"> <!--  320x460 -->

<!-- Metadaten -->
<meta charset="utf-8" />
<meta http-equiv="X-UA-Compatible" content="chrome=1" />
<meta name="HandheldFriendly" content="true" />
<meta name="apple-mobile-web-app-capable" content="yes" />
<meta name="viewport" content="width=device-width, user-scalable=no,
```

```
      minimum-scale=1, maximum-scale=1" />

<!-- Web Fonts -->
<link
  href=
'http://fonts.googleapis.com/css?family=Ubuntu:400,400italic,500,500italic&
subset=latin,latin-ext&v2'
  rel='stylesheet' type='text/css'>

<!-- GWT Bootstrap -->
<script type="application/x-javascript"
        src="waitara/waitara.nocache.js"></script>
<title>Waitara - Task Management</title>
</head>

<body>
  <iframe src="javascript:''" id="__gwt_historyFrame" tabIndex='-1'
          style="position: absolute; width: 0; height: 0; border: 0">
  </iframe>
</body>
</html>
```

Listing 5: Startseite der Anwendung

Die **<link>**-Elemente definieren unterschiedliche Icons, die unter iOS verwenden werden. Durch **apple-touch-startup-image** kann eine Grafik angegeben werden, die als Splash-Screen während des Ladens der Seite dargestellt wird. Diese Grafik muss im PNG-Format vorliegen und eine Größe von 320 x 460 Pixeln haben. Das Meta-Tag **HandheldFriendly** gibt an, dass der Content für mobile Geräte ausgelegt ist und nach Möglichkeit nicht skaliert werden soll. Das Flag wurde ursprünglich für den Avant-Go-Browser von Palm entworfen, wird aber heute von den meisten mobilen Browsern ausgewertet. Das Meta-Tag **apple-mobile-web-app-capable** schaltet den Safari-Browser in den Vollbildmodus und blendet die Adressleiste aus. Möchte man auch die Statuszeile ausblenden, muss man zusätzlich noch das Tag **apple-mobile-web-app-status-bar-style** angeben.

Eines der wichtigsten Meta-Tags ist **viewport**. Dadurch kann die per Default eingestellte Skalierung deaktiviert werden. Durch Angabe von **viewport="width=device-width, user-scalable=no, minimum-scale=1, maximum-scale=1"** wird erreicht, dass die volle Breite des Geräts für die Darstellung der Seite verwendet wird. Weiterhin wird das Skalieren deaktiviert. Das funktioniert sowohl für Smartphones als auch für Tablets.

Ausgerüstet mit diesen Elementen steht einer Verwendung unserer Anwendung nichts mehr im Wege. Sinn und Zweck all dieser Einstellungen ist es, dass sich unsere Taskverwaltung weniger wie eine Webseite, sondern mehr wie eine native Applikation anfühlt.

Ausblick

Momentan ist die Erzeugung eines Widgets in GWT fest mit einem oder mehreren DOM-Elementen verbunden. Diese Elemente sind wiederum mit bestimmten CSS-Klassen verknüpft. Möchte man das Aussehen eines Widgets in GWT ändern, kann man zwar die CSS-Klassen anpassen, auf die Struktur und den Aufbau des Widgets hat man allerdings keinen Einfluss. Diese Architektur erschwert das Anpassen und Austauschen des Look & Feels für unterschiedliche Browser, Betriebssysteme und Plattformen.

Deshalb gibt es derzeit den Ansatz, die Widgets mithilfe des sogenannten Appearance-Musters neu aufzusetzen. Dieses Muster verfolgt dabei die folgenden Ziele:

- Trennung des Widgets von der DOM-Struktur,
- Austausch der Styles eines Widgets,
- Austausch der DOM-Struktur eines Widgets,
- Kapselung der Stylesheets pro Widget.

Die DOM-Struktur und die Styles werden dazu in einer Appearance-Klasse gekapselt. Jedes Widget nutzt dann zur Darstellung eine bestimmte Appearance-Klasse. Der Austausch einer Appearance kann programmatisch oder per Deferred Binding abhängig von Browser und Plattform geschehen. Auf diese Art und Weise bekommen die Widgets auf dem iPhone ein typisches iPhone Look & Feel, während die Widgets auf dem Desktop in gewohnter Manier gerendert werden.

Das Appearance-Framework ist momentan in Entwicklung und wird aller Voraussicht nach in einer der nächsten GWT-Versionen enthalten sein. Weitere Informationen zum Appearance-Muster finden sich unter [CellBackedWidgets] und [Mey11].

Fazit

Durch Verwendung von GWT und Model-View-Presenter-Muster lassen sich Webanwendungen entwickeln, die auf allen gängigen Plattformen benutzt werden können. Dabei können große Teile der Anwendung für alle Plattformen gemeinsam verwendet werden. Insbesondere die Logik und die Steuerung im Presenter müssen nur einmal implementiert werden. Plattformabhängiger Code kann in den Views und den **UiBinder**-Templates hinterlegt werden. Der geschickte Einsatz von CSS und Meta-Tags ermöglicht dabei, dass sich die Webapplikationen wie native Anwendungen anfühlen.

Die Beispielapplikation einer einfachen Taskverwaltung besteht insgesamt aus ca. 1600 Zeilen Code. Davon werden etwa 1000 Zeilen für alle Plattformen gemeinsam verwendet. Der Rest verteilt sich auf die verschiedenen Plattformen.

Links

[AccWaitara] **http://devel.accelsis.biz/waitara/waitara.zip**
[CellBackedWidgets]
http://code.google.com/p/google-web-toolkit/wiki/CellBackedWIdgets
[DE11] VisionMobile: Developer Economics, 2011,
http://www.developereconomics.com
[GIN] GWT Injection, **http://code.google.com/p/google-gin/**
[GWTP] GWT Platform, **http://code.google.com/p/gwt-platform/**
[Mey11] D. Meyer, Ext GWT 3.0 Appearance Design, 28.6.2011, **http://www.sencha.com/blog/ext-gwt-3-appearance-design/**

Harald Pehl arbeitet als Senior Softwareentwickler, Enterprise Application Architect, IT-Berater und Dozent für die Accelsis Technologies GmbH. Er ist Informatiker mit über zehn Jahren Berufserfahrung in der Entwicklung von Enterprise-Java-Lösungen, Webapplikationen/-Portalen, Middleware und modernen Benutzeroberflächen. Aktuell liegt sein Schwerpunkt vor allem im Bereich GWT und RESTful Softwarearchitekturen. Harald Pehl ist Initiator und Committer von Piriti, einem JSON/XML-Mapper für GWT.
E-Mail: harald.pehl@accelsis.biz