Tablet ahoi!

App für den Hamburger Hafen

Holger Breitling, Martin Berghoff

In diesem Beitrag schippert der Hamburger Hafen mit der Kombination „Tablet + Hafenbarkasse" lässig an der endlos bemühten bayrischen Renommier-Formel von „Laptop und Lederhose" vorbei! Doch lassen wir die regionalen Frotzeleien einmal beiseite: Wenn die Mitarbeiter der Hamburg Port Authority (HPA) auf ihren Schiffen im Hafen der Hansestadt unterwegs sind und die Lage dort mit Hilfe eines Tablets überprüfen und übermitteln, dann ist das nicht nur modern und cool – dann verbirgt sich dahinter auch eine App, die auf Basis von Java (GWT) multiplattformfähig ist und außerdem eine tief greifende Integration in das Backend der Hafenbehörde realisiert.

Abb. 1: Mit dem Tablet auf der Barkasse (© HPA 2013)

Der Mobile Port Monitor

Seit Herbst 2012 setzt die Hamburg Port Authority (HPA) in ihrer Nautischen Zentrale das innovative Leitstandsystem *Port Monitor* ein. Die Nautische Zentrale ist zuständig für alles, was sich auf den Wasserstraßen des Hamburger Hafens und auf der Unterelbe bewegt, und der Port Monitor liefert die Ereignisse und Informationen, die für einen störungsfreien Verkehrsfluss der Schifffahrt wichtig sind – in Echtzeit und auf Basis georeferenzierter Daten. Im Frühling 2013 stach der „kleine Bruder" des Port Monitors, der *Mobile Port Monitor* mit den Besatzungen der Hafenbarkassen der HPA in See.

Das Ziel bei der Implementierung des *Mobile Port Monitors* war, die in der Desktop-Anwendung verfügbaren Informationen auf einem mobilen Gerät bereitzustellen, welches auf Schiffen der HPA eingesetzt werden kann.

Mitarbeiter der HPA fahren mit Barkassen regelmäßig den Hamburger Hafen ab, um bekannte Baustellen und Hindernisse zu kontrollieren und nicht bekannte Hindernisse und Störungen zu erkennen und zu melden. Bisher wurden bei derartigen Kontrollfahrten die Informationen nur notiert; gegebenenfalls wurden vor Ort Fotos erstellt. Erst nach der Überwachungstour konnten die neuen Informationen dann in die Systeme eingepflegt werden. Der Mobile Port Monitor ermöglicht es, von der Barkasse aus mit einem „Klick" aktuelle Statusinformationen sofort auf allen angeschlossenen Computern

Abb. 2: Der Port Monitor (Desktop)

sichtbar zu machen – eben auch auf dem Monitor des Nautikers vom Dienst in der Nautischen Zentrale.

Zu Projektbeginn gab es keine eindeutige strategische Vorgabe zur mobilen Zielplattform für die *Mobile Port Monitor App*. Die HPA hielt sich alle Optionen für die Zukunft offen. Zwar würde die Software zunächst konkret auf dem iPad zum Einsatz kommen, die Entwicklungstechnologie aber sollte möglichst plattformunabhängig sein, damit die App später problemlos auf unterschiedlichen Geräten bereitgestellt werden könnte. Die HPA hat inzwischen entschieden, ihre mobile IT-Strategie auf Windows (Windows Phone, Windows RT) auszurichten.

Funktionalität

Der Mobile Port Monitor bietet einen Ausschnitt der Funktionalität der Desktop-Variante. Im Zentrum steht die Darstellung der Hafenkarte, die von der Kartografie der HPA bereitgestellt wird – Google Maps & Co. reichen hier wegen mangelnder Genauigkeit nicht aus!

Der Benutzer kann mit der Karte wie gewohnt interagieren (Zoom, Pinch, Move usw.), aktuelle Baustellen, Maßnahmen und Hindernisse erkennen und die Eigenschaften dieser Kartenelemente (Kontextinformationen) einblenden. Um die Kartenelemente zu bearbeiten, kann er von der Kartenansicht aus für sie einen Editor öffnen.

Der Mobile Port Monitor ist idealerweise über das mobile Internet mit den Systemen der HPA verbunden. Ist diese Verbindung unterbrochen, funktioniert er jedoch auch offline. Dazu speichert er die entsprechenden Informationen, unter anderem die Hafenkarte, lokal und kann Änderungen zu einem späteren Zeitpunkt synchronisieren (z. B. Upload ausstehender Änderungsaufträge).

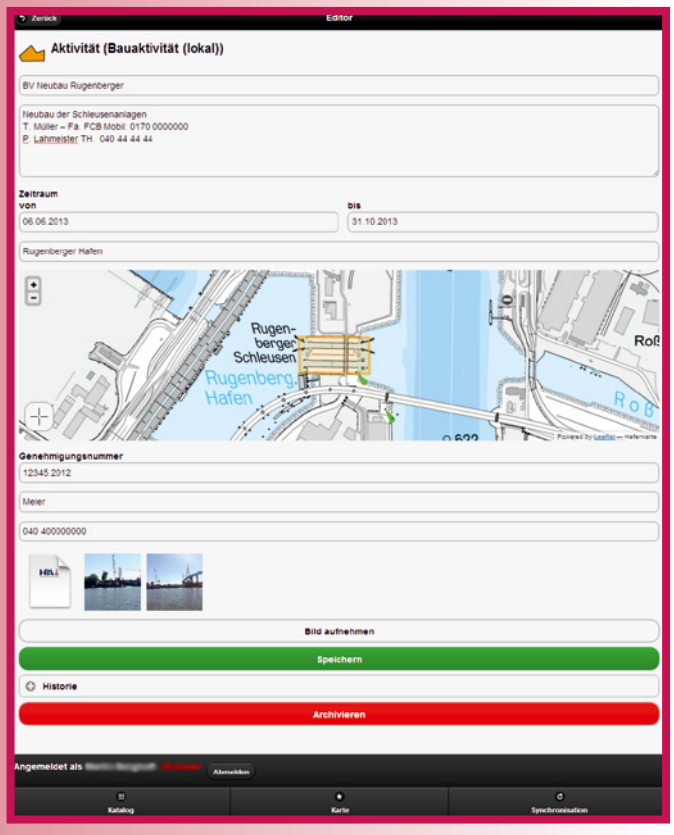

Abb. 3: Der Mobile Port Monitor (Detailansicht Aktivität)

Weitere Features sind:

- personalisierter Zugang (Authentifizierung gegen bestehendes Active Directory der HPA),
- Listenansicht aktueller Baustellen, Maßnahmen und Hindernisse, sortiert nach Entfernung zum eigenen Standort,
- Vorschau auf anhängende Dokumente und Bilder (lesender Zugriff auf den Sharepoint-Server der HPA) und direkter Zugriff auf die Dateien im Sharepoint (Öffnen in nativen Apps),
- Anhängen und Hochladen eines Kamerabildes (schreibender Zugriff auf den Sharepoint-Server der HPA).

Architektur

Die Vorgabe war, den Mobile Port Monitor plattformunabhängig und offline-fähig umzusetzen. Um die Plattformunabhängigkeit zu gewährleisten, entschieden wir uns grundsätzlich für einen Web-Stack als Basis der App (JavaScript, HTML 5, CSS). Wir entwickelten sie unter Einsatz von mehreren Toolkits und Bibliotheken:

- *Google Web Toolkit [GWT]* für das Grundgerüst der Anwendung, die Interaktionssteuerung und Fachlogik,
- die JavaScript-Bibliothek *jQuery Mobile [jQM]* für die Implementierung der Benutzungsschnittstelle,
- die JavaScript-Bibliothek *Leaflet* zur Darstellung interaktiver Karten,
- das Toolkit *PhoneGap*, um auf native Funktionen des Mobilgeräts zuzugreifen (z. B. Kamera) und unsere Anwendung als App zu „verpacken".

Im Folgenden gehen wir auf die einzelnen Bestandteile und ihr Zusammenspiel genauer ein.

Google Web Toolkit

Mit dem Google Web Toolkit (GWT) nutzen wir eine Sammlung von Werkzeugen zur Entwicklung von Webanwendungen, die die Programmierung in Java erlauben und den Code dann in eine JavaScript-Ajax-Anwendung übersetzen, die in jedem JavaScript-fähigen Browser lauffähig ist. Dies ist ein wesentlicher Baustein, um die Unabhängigkeit von der konkreten mobilen Plattform zu erreichen.

Die App ist offline-fähig, da Ajax-Seiten wie eigene Anwendungen im Browser laufen. Einmal geladen, ist die Anwendung vollständig auf dem Client verfügbar und kommuniziert mit dem Server nur, wenn es nötig und der Server erreichbar ist. GWT bietet für diese Kommunikation einen eigenen, optimierten Remote-Procedure-Call-Mechanismus [GWTRPC].

Die zu GWT gehörenden Widgets nutzen wir nicht; hierfür binden wir Widgets von jQuery Mobile ein – bzw. für die Karte die Bibliothek Leaflet. Das Ergebnis ist optisch attraktiver und näher an einem „mobilen Look & Feel", als es mit GWT-Bordmitteln der Fall wäre, zudem liefern die Bibliotheken die benötigte Unterstützung von Touch-Gesten, die GWT selbst fehlt.

Um jQM und Leaflet mit GWT zu verbinden, nutzen wir GWTs sogenanntes JavaScript Native Interface (JSNI), um beim Codieren in Java den zur Zusammenarbeit mit den Bibliotheken benötigten JavaScript-Code so einzubinden, dass der GWT-Compiler entsprechende Aufrufe im Kompilat (dort: von JavaScript nach JavaScript) erzeugt. Für die Widgets erstellen wir so „GWT-Wrapper".

Die beschriebene Technologiekombination mit GWT als Basis erlaubt es uns, Java als primäre Programmiersprache einzusetzen und damit auf eine ausgereifte und statisch typisierte Sprache mit vielfältigen Entwicklungswerkzeugen zurückzugreifen. Dass auch die Desktop-Version des Port Monitors eine Java-Anwendung ist, macht diesen Ansatz für unser Entwicklungsteam noch sinnvoller.

jQuery Mobile

jQuery ist eine weit verbreitete JavaScript-Bibliothek für Animation, DOM-Manipulation, Event-Verarbeitung und Ajax. jQuery Mobile (jQM) baut darauf auf und enthält ein für Mobilgeräte geeignetes „User Interface System“, das wegen des konsequenten Einsatzes von HTML 5 auf allen relevanten Plattformen und Gerätegrößen (Smartphone, Tablet, Desktop) funktioniert. Die bereitgestellten Widgets behandeln in angemessener Weise sowohl Touch-Gesten als auch Maus- und Tastaturereignisse und sind für unterschiedliche Auflösungen und Formate verfügbar.

Wie bereits im Abschnitt zu GWT dargestellt, verknüpfen wir GWT mit jQM, indem wir „GWT-Wrapper“ verwenden, die auf Basis von JSNI erstellt sind und die jQM-Komponenten von GWT aus in ähnlicher Weise benutzbar machen wie die fest zu GWT gehörenden Widgets.

Eine besondere Herausforderung war, das Page-Flow-Konzept von jQM in unser Anwendungskonzept zu integrieren. Eine jQM-Anwendung ist über Pages organisiert, zwischen denen es (potenziell animierte) Übergänge gibt. GWT bietet bei der Strukturierung einer Anwendung mit Activities und Places einen davon unabhängigen Ansatz. Diese beiden Welten miteinander zu verbinden, ist nicht trivial. Wir haben uns dazu entschlossen, dicht am jQM-Modell zu bleiben, da es für das mobile Umfeld gut passt und der Page-Flow tief in jQM verankert ist. Eine jQM-Page interpretieren wir als die Benutzungsoberfläche eines Softwarewerkzeugs. Jedes dieser Werkzeuge repräsentieren wir in GWT wiederum durch eine Tool-Klasse für Interaktion und Logik sowie eine UI-Klasse, die die Widgets verwaltet und layoutet. Die Wechsel zwischen Werkzeugen implementieren wir als jQM-Page-Wechsel. Die Steuerung übernimmt ein GWT-Applikations-Controller, der vor jedem Wechsel involviert wird und das angesprochene Werkzeug veranlasst, die jQM-Page (die im HTML einem DOM-Knoten entspricht) vorzubereiten. Da im GWT-Anteil der Anwendung Fachlogik und Interaktionssteuerung implementiert sind, prüft dieser zum Beispiel auch, ob die Voraussetzungen (Authentifizierung und Autorisierung) gegeben sind, das Werkzeug aufzurufen.

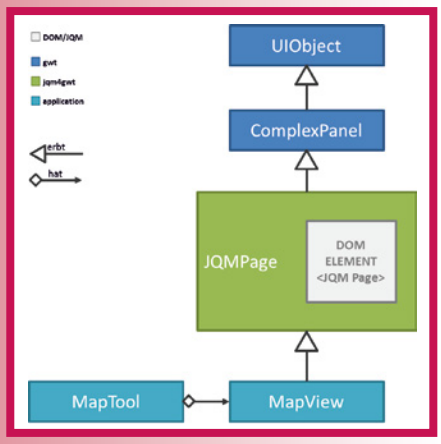

Abb. 4: Integration GWT/jQM im Mobile Port Monitor (statisch)

Leaflet

Leaflet ist eine für Mobilgeräte optimierte JavaScript-Bibliothek für interaktive Karten. Sie kann die Hafenkarte, die vom Server des Geoinformationssystems (GIS) der HPA über den Schnittstellenstandard Web Map Service (WMS) zur Verfügung gestellt wird, auslesen und bietet touch-basiert die bekannten Interaktionsmöglichkeiten eines Kartenwerkzeugs. Sie unterstützt verschiedene Projektionen und Georeferenzsysteme und kann auf der Karte Symbole, Polygone und Linien über der Hintergrundkarte einblenden. Die technische Einbettung über JSNI wurde in den Abschnitten zu GWT und jQM bereits erläutert.

PhoneGap

PhoneGap ist das Toolkit, das es uns ermöglicht, unsere auf einem plattformunabhängigen Web-Stack aufgebaute Anwendung wie eine native App auf den unterschiedlichen mobilen Plattformen zu installieren und zu nutzen:

- PhoneGap stellt ein JavaScript-API bereit, über das Anwendungen auf native Geräteeigenschaften zugreifen können, z. B. Kamera, Kompass, GPS, Eigenschaften der Datenverbindung, das Dateisystem.
- PhoneGap „verpackt“ die HTML-Anwendung in einen nativen Container, der für das mobile Betriebssystem die eigentliche App darstellt. Dieser Container bringt eine HTML-Rendering-Engine (einen Web-Browser-View) für die Webanwendung mit. Die gesamte Webanwendung kann als fester Bestandteil der App (einschließlich benötigter Ressourcen) auf dem Gerät installiert werden und muss nicht von einem Server geladen werden. Dies macht unsere App erst vollständig offline-fähig.

Serverzugriff und Caching

GWT bringt für die Kommunikation zwischen Client und Server das GWT-RPC-Framework mit. Die meisten fachlichen Services, die unser mobiler Client benötigt, waren für die Desktop-Variante des Port Monitors bereits vorhanden. Für diese Services haben wir GWT-RPC-Adapter erstellt. Die Service-Definition erfolgt dabei direkt als Java-Interface, welches von der GWT-Klasse `RemoteService` erbt. Der serverseitige Teil wird über ein Servlet bereitgestellt. Das GWT-Interface `AsyncCallback` bietet eine gute Möglichkeit, im Clientcode asynchron mit Serviceaufrufen zu arbeiten und so geeignet mit der Unzuverlässigkeit mobiler Netzverbindungen umzugehen.

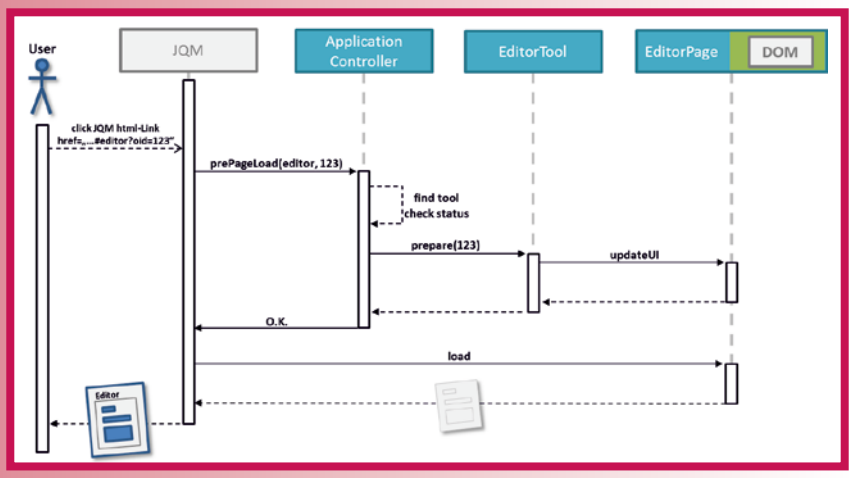

Abb. 5: Page-Flow des Mobile Port Monitors (Sequenzdiagramm)

Der GWT-Clientcode wird von uns als Teil der PhoneGap-Applikation auf dem Gerät installiert. Dadurch entsteht die für GWT unübliche Situation, dass die Webseite von einer lokalen Adresse geladen wird, der Server aber eine abweichende Adresse hat. Damit der Serviceaufruf dennoch funktioniert, muss dem ServiceProxy nachträglich die zu verwendende Serviceadresse bekannt gemacht werden.

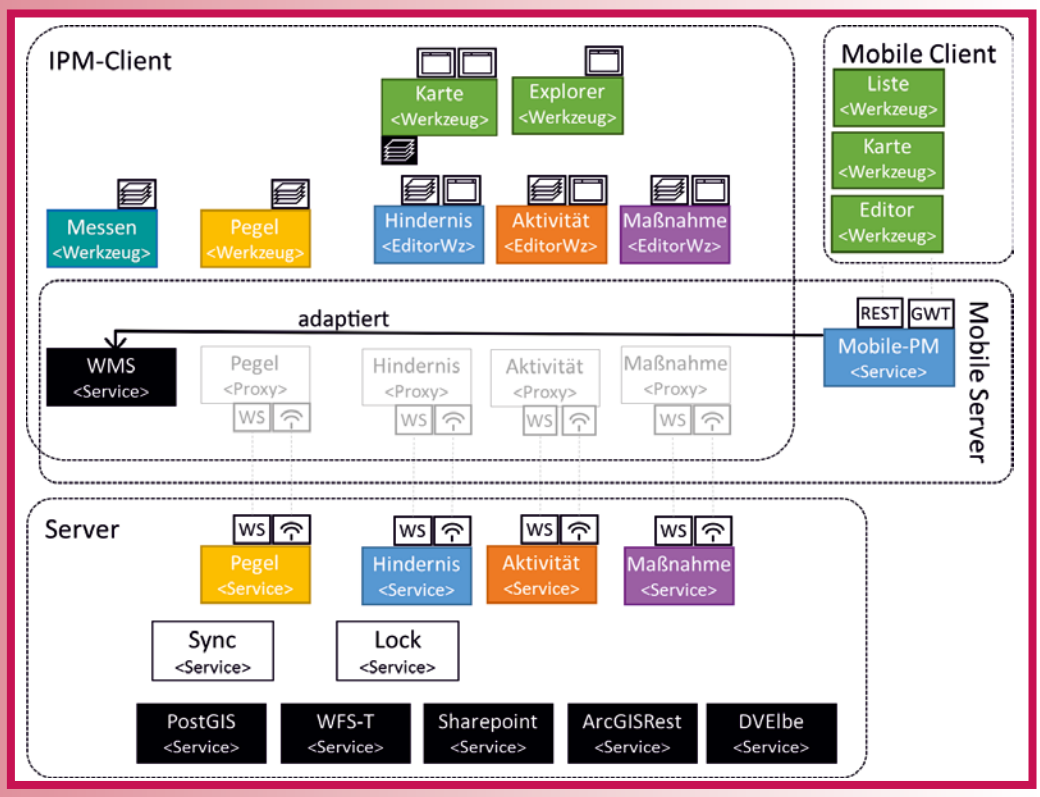

Abb. 6: Einordnung des Mobile Port Monitors in die Umgebung (Architekturbild)

Hier bietet der GWT-Wrapper für PhoneGap bereits eine fertige Lösung:

```
MyServiceAsync myRemoteService = GWT.create(MyService.class);
PhoneGapUtil.prepareService((ServiceDefTarget) myRemoteService,
  baseURL, "myService");
```

Im Client werden die Datenobjekte (Materialien und Anwendungskonfiguration) in einem lokalen persistenten Cache (HTML5 Web Storage) gehalten und zur Laufzeit in den Anwendungsspeicher geladen. Ein Synchronisationsmechanismus prüft periodisch die Verfügbarkeit der mobilen Datenverbindung zum Server und gleicht den lokalen Datenbestand mit dem Serverstand ab. Dabei werden lokale Änderungen an den Server übertragen, neue Fotos zum Sharepoint hochgeladen und Änderungen vom Server zum Client übertragen. Hierbei geht es vor allem um die Möglichkeit, Hindernisse und schifffahrtspolizeiliche Maßnahmen direkt vor Ort erfassen und bearbeiten zu können, ohne auf eine permanente Datenverbindung angewiesen zu sein. Außerdem wird die Hafenkarte vollständig auf dem Mobilgerät vorgehalten, sodass weiter navigiert werden kann (auch mit GPS), wenn einmal kein Netzempfang möglich ist.

Erfahrungen aus der Entwicklungsarbeit

GWT-basierten Code haben wir in der Eclipse-IDE mit dem Google-Plug-in entwickelt. Dieses ist ausgereift und gut integriert. Die Übersetzung von Java- in JavaScript-Code funktioniert zuverlässig und macht keinerlei Probleme. Ein Generator unterstützt die Erstellung der Serviceschnittstellen. GWT RPC ist ein einfach zu beherrschender Mechanismus, der die Objekte transparent serialisiert und deserialisiert.

Das Zusammenspiel der verschiedenen Bibliotheken hat jedoch das Debugging stark verkompliziert; im sogenannten GWT Development Mode, der schnelles Testen und Debugging im Java-Quelltext erlaubt, sind die jQM-Komponenten nicht verfügbar. Das Debuggen der fertigen Anwendung (im Container mit Zugriff auf native Geräteeigenschaften) ist für iOS ebenfalls nur im generierten JavaScript möglich.

Die Dokumentation der von uns eingesetzten Toolkits und Bibliotheken (GWT, jQuery, jQM, Leaflet und PhoneGap) ist gut. Die Communities um diese Projekte sind aktiv. Es gibt zahlreiche Beispiele und gute API-Dokumentation. Eine Herausforderung stellt nur die spezielle Kombination der unterschiedlichen Bibliotheken dar, zu der es naturgemäß weniger frei verfügbare Informationen gibt.

Die GWT-Wrapper für Java-Script-Bibliotheken, die wir im Internet finden konnten, sind vom Entwicklungsstand her durchwachsen. Sie eigneten sich nur als Ausgangspunkt für unsere Eigenentwicklung mit JNSI.

Trotz der beachtlichen Leistungsfähigkeit des Web-Stacks muss man sich darüber im Klaren sein, dass beim aktuellen Stand der Technik Performanceeinbußen gegenüber einer nativen Implementierung nicht vermieden werden können, da diese für ihre Plattform optimiert ist und direkten Zugriff auf die Systemressourcen hat. Dies betrifft besonders die Benutzungsoberfläche bei Animationen, Überblendungen und die Reaktionsgeschwindigkeit nach Benutzereingaben. Wir haben dennoch eine User Experience (UX) realisieren können, bei der vor allem die aufgabenangemessene Unterstützung der Anwender, die umfangreiche Funktionalität und ein gelungenes Design im Vordergrund stehen und die genannten Performanceschwächen darum kaum wahrgenommen werden.

Abb. 7: Alles im Blick (© HPA 2013)

Benutzerfeedback

Nach der produktiven Einführung wurde der Mobile Port Monitor (MPM) durch die Anwender vorwiegend positiv bewertet. Die App wird bei der täglichen Arbeit durchgängig eingesetzt und als sinnvolle Ergänzung des Desk-

top-basierten Port Monitors angesehen, als „verlängerter Arm des Desktops“. Die im Vergleich zu einer nativen Anwendung unmerklich weniger flüssige Interaktion wird fast nicht registriert. Die vorherrschende Wahrnehmung fasst der folgende O-Ton zusammen: „Mit dem MPM können neue Informationen einfach und direkt erfasst werden und stehen anderen Mitarbeitern viel schneller als bisher zur Verfügung.“

Fazit

Die Kombination von GWT, jQuery Mobile, Leaflet und PhoneGap haben wir als Antwort auf Kundenanforderungen gewählt; den Mobile Port Monitor haben wir mit diesem Stack plattformunabhängig erstellt und für iPad/iOS als konkrete Plattform optimiert, getestet und in Einsatz gebracht. Das erlaubte dem Entwicklungsteam, Java als primäre Entwicklungssprache einzusetzen und dabei eine hohe Entwicklungsgeschwindigkeit und gute Qualität zu erreichen.

Zwar ergaben sich konzeptionelle Herausforderungen bei der Verknüpfung von GWT und jQM sowie im Debugging, insgesamt hat sich die Technologiekombination aber als gut beherrschbar und tragfähig erwiesen. Die HPA hat sich strategisch inzwischen für Windows als mobile Plattform entschieden, und wir konnten die Multiplattformfähigkeit des erstellten Codes erfolgreich für Windows Phone erproben und nachweisen. Da mit dieser strategischen Entscheidung auf mittlere Sicht eine andere Ablaufumgebung für den Mobile Port Monitor ansteht, sind wir davon überzeugt, dass der gewählte Implementierungsansatz genau richtig war. Außerdem gibt es Überlegungen, den Mobile Port Monitor in einem anderen Kontext möglicherweise auch als „normale“ Webanwendung zur Verfügung zu stellen, die dann auf Desktops außerhalb der HPA genutzt werden könnte.

Die wichtigste Bestätigung erfährt die App von den Anwendern, die sie täglich benutzen und ihre Zufriedenheit unter anderem auch dadurch zeigen, dass sie zahlreiche Vorschläge für funktionale Erweiterungen und die Ausweitung ihres Einsatzes machen.

Wir sind mit dem Mobile Port Monitor hoffentlich erst am Anfang einer längeren Schiffsreise und freuen uns jedes Mal, wenn wir eine neue Version „zu Wasser lassen“ können: „Leinen los!“

Links

[GWT] Google Web Toolkit,
`http://de.wikipedia.org/wiki/Google_Web_Toolkit`
[GWTPhonegap] `https://code.google.com/p/gwt-phonegap/`
[GWTRPC] GWT Project, RPC, `http://www.gwtproject.org/doc/latest/DevGuideServerCommunication.html#DevGuideRemoteProcedureCalls`
[HPA12] Klarmachen zum Klicken,
Seiten 24 bis 27 aus dem Geschäftsbericht 2012 der HPA,
`http://www.hamburg-port-authority.de/de/presse/broschueren-und-publikationen/Documents/Geschäftsbericht%202012%20Imageteil.pdf`
[jQM] `http://jquerymobile.com/`
[jQuery] `http://jquery.com/`
[Leaflet] `http://leafletjs.com/`
[PhoneGap] `http://phonegap.com/`

Holger Breitling ist Senior Softwarearchitekt bei der C1 WPS GmbH und Mitglied der Geschäftsleitung. Er beschäftigt sich mit Integrations- und Transformationsarchitekturen und Softwareentwicklungsvorgehen.
E-Mail: holger.breitling@c1-wps.de

Martin Berghoff ist seit 2008 als Berater und Softwarearchitekt bei der C1 WPS GmbH tätig. Dabei liegen seine Schwerpunkte auf der objektorientierten Anwendungsentwicklung sowie der Gestaltung und Entwicklung von Benutzungsmodellen für Benutzungsoberflächen.
E-Mail: martin.berghoff@c1-wps.de