Java 9: Neuer, toller, bunter? S. 18

Angular 2: Moderne Frontends entwickeln S. 65

Cloud: Enabler der digitalen Transformation S. 84

JAVA Mag

Java magazin™

Java | Architektur | Software-Innovation

W-JAX
Programminfos ab Seite 26!

DIE NEUE GENERATION BIG DATA

SMACK

© S&S Media

Ausgabe 7.2016

Deutschland € 9,80
Österreich € 10,80
Schweiz sFr 19,50
Luxemburg € 11,15

www.javamagazin.de

Sanfte Migration von Swing ins Web

Die goldene Brücke

Das Web als neue Run-anywhere-Plattform ist gesetzt. Viele Unternehmen haben jedoch in der Vergangenheit in integrierte Anwendungen mit Desktoptechnologien investiert, wie Java Swing oder Microsoft WPF. In diesem Artikel wird ein Architekturansatz vorgestellt, der eine schrittweise Migration mit parallelem Betrieb und nahtloser Integration von neuen und alten Bestandteilen ermöglicht.

von Thomas Kruse

Java-Desktopanwendungen wurden vor dem Aufkommen von JavaFX klassischerweise mit Swing implementiert. Swing bietet ein betriebssystemunabhängiges API und eine reichhaltige Komponentenbibliothek, mit der sich Geschäftsanwendungen gut umsetzen lassen. In **Abbildung 1** ist eine typische Java-Swing-Desktopanwendung zu sehen. Sie besteht aus einer Navigation und einem Formularbereich. Die Navigation erlaubt den Zugriff auf alle Anwendungsfälle, z. B. alles rund um Verträge. Statt einer Schnellnavigation könnte der Einstieg auch über ein Menü erfolgen. Im Formularbereich wird der jeweilige Anwendungsfall zur Verfügung gestellt. Im Anschluss an den Formularbereich sind unten verknüpfte Anwendungsfälle zu sehen.

Die Vorteile dieser Integration zeigen sich neben der starken Vernetzung der Daten und Anwendungsfälle auch an Punkten, die auf den ersten Blick weniger offensichtlich sind. Es ist lediglich eine Anmeldung erforderlich, und die Bedienkonzepte sind einheitlich. Außerdem gibt es einen gemeinsamen Kontext, in dem aggregierte Daten angezeigt werden können, z. B. auf der Kundenseite, wenn ein Mahnverfahren läuft.

Das User Interface im Swing Client ist sowohl für die grafische Aufbereitung als auch für die View-Logik ver-

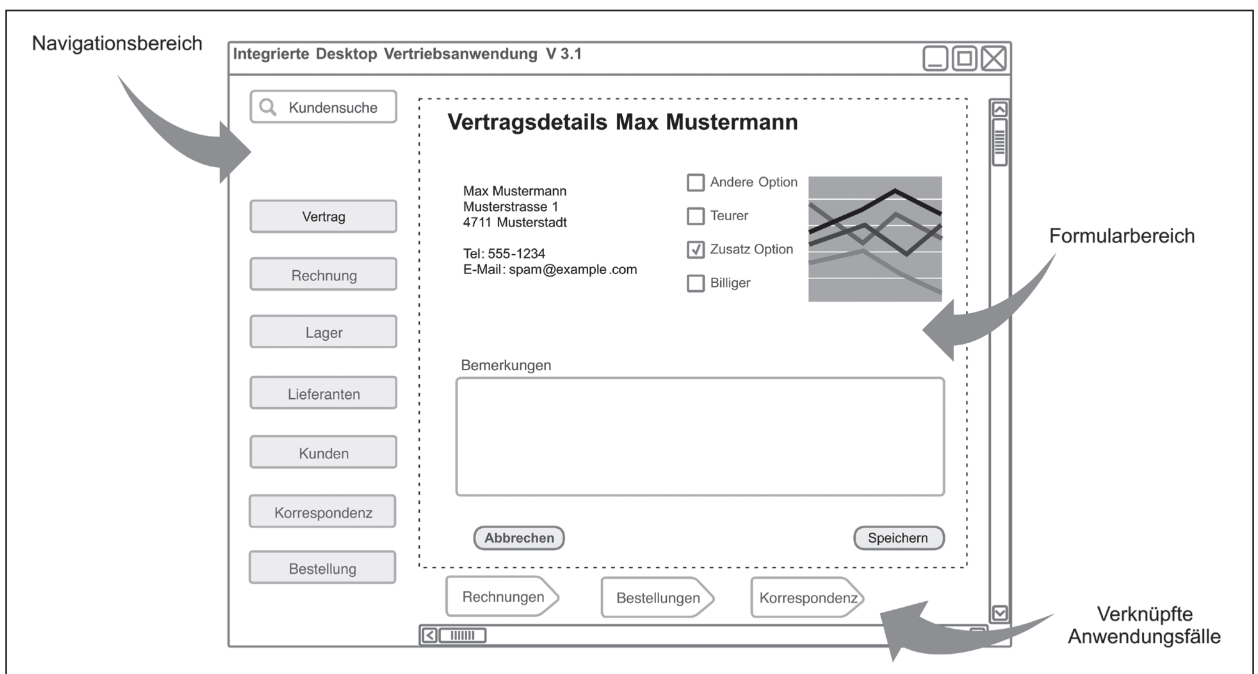


Abb. 1: Eine integrierte Desktopanwendung mit Swing bietet eine starke Vernetzung der Daten und Anwendungsfälle

antwortlich. Oft fällt es schwer, die Grenzen zwischen Geschäftslogik und View-Logik scharf zu trennen, da beide ineinander greifen. Bei einer sauberen Trennung von Verantwortlichkeiten befindet sich auf dem Swing Client lediglich der Teil der Geschäftslogik, der eng mit der Darstellung verbunden ist. Im Beispiel könnte dies eine farbige Markierung von Feldern sein, die Daten mit Validierungsfehlern enthalten. Je nach Art der Validierung ist diese als Teil der reinen Geschäftslogik auf dem Server zu finden. Um die User Experience zu verbessern, kann auch ein Teil im Client redundant implementiert sein.

Zur Umsetzung des User Interface mit View-Logik kommt in der Regel das Entwurfsmuster Model View Controller (MVC) zum Einsatz: Swing-Komponenten selbst sind intern bereits nach dem MVC-Muster aufgebaut, aber auch bei dem übergreifenden Design von grafischen Anwendungen hat sich der Einsatz des MVC-Musters bewährt. Es gilt daher zwischen MVC auf Anwendungsebene und auf Komponentenebene zu differenzieren.

Bei Swing erfolgt die Integration verschiedener Anwendungsteile durch entsprechende Implementierung der Dialoge und Verknüpfung innerhalb der View-Logik. Da der Client einen eigenen Zustand und eine eigene Ablauflogik hat, lassen sich damit Anforderungen wie Datenübergabe zwischen Dialogen und abwechselnde Bearbeitung verschiedener Anwendungsfälle abbilden. Letzteres ist wichtig, wenn ein Bearbeiter in einem Vorgang durch ein synchrones Ereignis, beispielsweise einen Telefonanruf, unterbrochen wird. Dann soll es möglich sein, den anrufenden Kunden zu bedienen und danach an der Stelle weiterzumachen, an der der Bearbeiter vor dem Telefonanruf war.

Diese hohe Integration hat ihren Preis: Im Vergleich zu mehreren isolierten kleineren Anwendungen ist der langfristige Aufwand höher. Neben einer umfangreicheren Architektur für das reibungslose Zusammenspiel der Anwendungsteile ist die Entwicklung neuer Features aufwendiger. Auch bei der Wartung eines solchen Frontend-Monolithen entstehen allein durch Abstimmung und Test des insgesamt komplexeren Systems höhere Aufwände. Dem höheren Entwicklungsaufwand stehen die gute Bedienbarkeit, Zeitersparnis bei der Nutzung und höherer Zufriedenheit der Anwender gegenüber. Bei einem Wechsel der Plattform von Swing-Desktop zu Webanwendungen gibt es daher oft die Anforderung, weiterhin die gute Benutzbarkeit eines integrierten Frontends bereitzustellen.

Moderne Webarchitektur am Beispiel AngularJS

Als Konsequenz von clientseitigem JavaScript gepaart mit Ajax und leistungsfähigen Browsern entstehen heute moderne Webanwendungen, die Desktopanwendungen sehr nahe kommen. Häufig auch als Single-Page-App bezeichnet, kommt bei solchen Anwendungen kein Neuladen der Seite mehr vor. Dabei stellt die Serverseite eine Schnittstelle mit HTTP als Kommunikationskanal bereit. Als Datenformat kommt häufig JSON oder XML

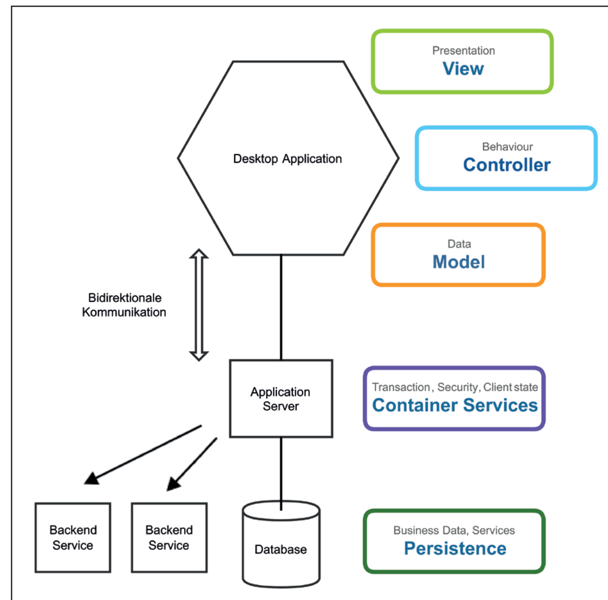


Abb. 2: Aufbau einer Swing-basierten mehrschichtigen Architektur mit einem Java-EE-Server im Backend

zur Anwendung. Optional lässt sich dies mit Verfahren kombinieren, die Daten aktiv zum Client pushen können.

Eine solche Architektur hat den Vorteil gegenüber klassischen Webarchitekturen mit serverseitigem Benutzerinterface, dass eine deutlich stärkere Trennung zwischen Client, Server, Daten und Präsentation besteht und die Kommunikation sowohl von der Plattform als auch vom Framework unabhängig ist. Mit diesem Ansatz lassen sich leicht verschiedene Clients entwickeln, die dieselbe Schnittstelle verwenden. Diese Architektur ist damit näher an Desktoparchitekturen, die auch Datenschnittstellen nutzen. Beispiele für den sinnvollen Einsatz unterschiedlicher Clients sind separate mobile Anwendungen oder auch das Bereitstellen von Teilfunktionalität für Endkunden in einer separaten Anwendung.

Für die Entwicklung von integrierten und auf Webstandards basierenden Browseranwendungen hat sich neben den Platzhirschen AngularJS [1] und ReactJS ein reichhaltiges Ökosystem an Frameworks entwickelt. Erfahrungsgemäß kommen Java-Entwickler gut mit den Konzepten von AngularJS zurecht. Wie in **Abbildung 3** zu sehen ist, hat diese Architektur noch einen weiteren Vorteil: Die Integration verschiedener Dienste kann im Client erfolgen. Das erlaubt Optimierungen (z. B. Caching oder Lazy Loading) und verringert die Last im Backend.

Leise Schritte statt lauter Knall

Ablöseprojekte und Neuentwicklungen im Big-Bang-Verfahren tendieren dazu, über Zeit- und Kostenrahmen hinaus zu laufen. Erschwerend kommt hinzu, dass dabei das neue System erst spät in Produktion geht. Fehler und Verbesserungspotenzial werden somit ebenso spät erkannt, notwendige neue Features müssen eventuell doppelt neu entwickelt werden. Das steht im Gegen-

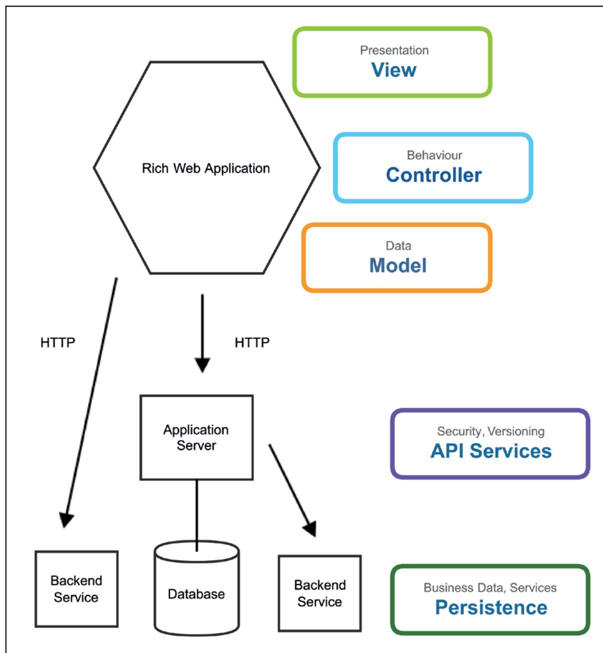


Abb. 3: Aufbau einer modernen Webarchitektur auf Basis eines clientseitigen JavaScript-Frameworks wie AngularJS

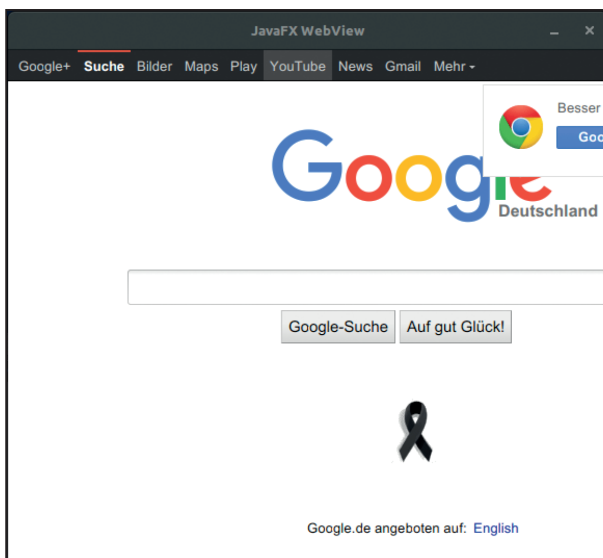


Abb. 4: JavaFX WebView mit Google in einer Swing-Anwendung

satz zu dem Wunsch, früh Feedback von echten Nutzern einzuholen und das System kontinuierlich weiterentwickeln und verbessern zu können.

Aus diesen Gründen ist eine schrittweise Ablösung wünschenswert – am besten ohne Medienbruch, sondern als nahtlos integrierte Anwendung. Es gilt also einen Weg zu finden, ein Frontend mit Webtechnologie mit dem Swing-basierten Frontend zu integrieren. Die Herausforderung besteht dabei nicht nur darin, dass eine Webseite innerhalb der Anwendung angezeigt werden muss. Es gilt auch Interaktionen zwischen dem Desktopteil und dem Webteil zu realisieren, denn verknüpfte Anwendungsfälle müssen in beide Richtungen realisiert werden. Ein möglicher Lösungsweg geht dabei über JavaFX.

Brückenschlag dank JavaFX und WebKit

JavaFX gehört zum Standardlieferumfang der Oracle-Java-Laufzeitumgebung und steht somit auf allen aktuellen Plattformen zur Verfügung. Außerdem hat Oracle JavaFX als Open Source bereitgestellt, sodass keine Bedenken bezüglich der Zukunftssicherheit aufkommen. JavaFX bietet zwei wichtige Funktionalitäten: Zum einen lassen sich JavaFX-Komponenten gemischt mit Swing-Komponenten einsetzen. Zum anderen bietet JavaFX eine auf der WebKit Engine basierende Browserkomponente, die WebView [2]. Mit der WebView lässt sich die Darstellung von Webinhalten direkt in eine Swing-Anwendung einbetten. Es werden lediglich wenige Zeilen Java benötigt, um Google innerhalb einer Swing- oder JavaFX-Anwendung einzubetten. Das Ergebnis ist in **Abbildung 4** zu sehen:

```
final WebView webView = new WebView();
final WebEngine webEngine = webView.getEngine();
webEngine.load("https://www.google.com/");
```

Natürlich wird eine Rahmenanwendung benötigt, die die Swing-Komponenten initialisiert und aus der die WebView eingebunden wird. Innerhalb der WebView ist bereits eine normale Verwendung der Webanwendung möglich: Links lassen sich anklicken, Formulare ausfüllen und absenden.

Was jetzt noch fehlt, ist eine Brücke zwischen Webinhalten und der Rahmenanwendung. Das Ziel bei der Architektur der Verbindung sollte sein, dass die Integration in Swing nicht zwingend ist, sondern die Webanwendung auch losgelöst betrieben und genutzt werden kann.

Dazu bietet sich JavaScript an: Zum einen kann von Java heraus JavaScript in der jeweiligen Seite ausgeführt werden, zum anderen können Java-Methoden als JavaScript publiziert und dann von der Seite heraus aufgerufen werden. Das API erlaubt auch weitergehende Eingriffe, wie Modifikationen der zu ladenden CSS- und JavaScript-Dateien sowie Modifikationen des DOM der Seite selbst. Damit lässt sich die Webseite für die eingebettete Nutzung optimieren und das Styling anpassen oder zusätzliche Interaktionselemente hinzufügen.

Im einfachsten Fall publizieren wir ein Java-API. Dabei ist darauf zu achten, dass die Seite geladen ist, damit die Registrierung fehlerfrei funktioniert. An dieser Stelle wird auch deutlich, dass eine Single-Page-App sich besonders gut eignet, da diese lediglich einmal geladen wird. Der zugehörige Quellcode zur Veranschaulichung enthält die Schritte:

1. Registrierung eines Callbacks, um den Ladestatus der Seite zu ermitteln
2. Reaktion des Callbacks, wenn der Ladestatus ergibt, dass die Seite geladen ist
3. Aus dem Callback erfolgt die Publizierung der Java-Klasse *JavaApp* als JavaScript-*app*-Objekt

Die Java-Klasse ist einfach gehalten und zeigt lediglich eine Swing Alertbox an. Die grundsätzliche Verwendung sieht wie folgt aus:

```
public static class JavaApp
{
  public void trigger()
  {
    JOptionPane.showMessageDialog(null, "Java Dialog triggered from the
                                                              web!");
  }
}
```

Das Beispiel für die Registrierung der Java-Klasse verwendet Java-8-Lambda-Ausdrücke, um den Listener zu implementieren:

```
webEngine.getLoadWorker().stateProperty().addListener(
  (ObservableValue<? extends State> ov, State oldState, State newState) ->
  {
    if (newState == State.SUCCEEDED)
    {
      JSObject win = (JSObject) webEngine.executeScript("window");
      win.setMember("app", new JavaApp());
    }
});
```

In **Abbildung 5** ist der geöffnete Swing-Dialog als Reaktion auf den Knopfdruck in der WebView zu sehen.

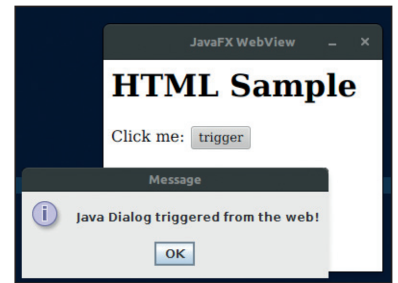


Abb. 5: Nachdem der Knopf im Web gedrückt wurde, erscheint ein Java-Dialog

Der Sprung ins Web mit APIs

Die Integration selbst ist keine große technische Herausforderung. Wünschenswert wäre jedoch ein Vorgehen, bei dem die notwendigen Änderungen beim schlussendlichen Sprung aus der Swing-Desktopanwendung ins Web minimiert werden. Die Lösung dazu könnte wie folgt aussehen: Es wird ein API zur Integration designt, die sich sowohl mit der Swing-Rahmenanwendung als auch mit einer (späteren) Webanwendung abbilden lässt. Der Umfang des API wird dabei gering und das Abstraktionsniveau hoch gehalten, schließlich soll hier keine zweite Controller-, sondern eine Integrationsschicht entstehen. Die Java-Anwendung ist dabei auch in der Lage Legacy-APIs als lokale HTTP-Dienste anzubieten, um auch an dieser Stelle eine schrittweise Migration zu ermöglichen. Mit dieser Architektur ist ein paralleler Betrieb von Swing- und JavaScript-Rahmenanwendung möglich. Das erlaubt frühes Testen, schnelles Feedback

Anzeige

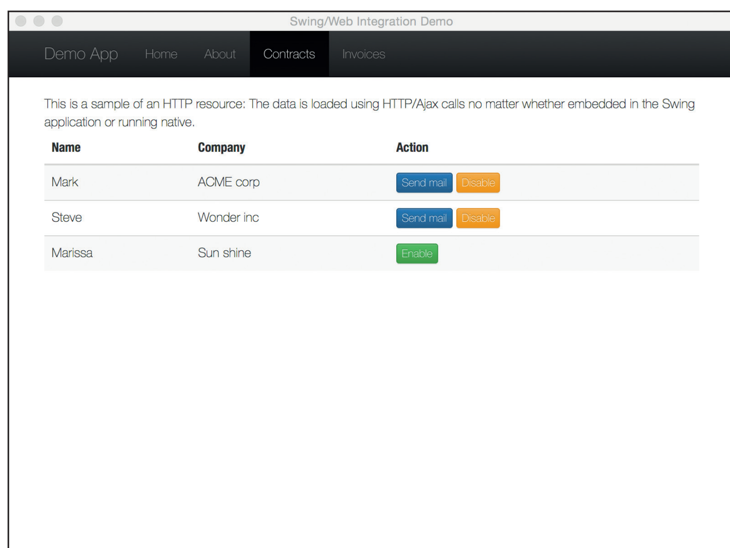


Abb. 6: Beispiel einer Desktopanwendung mit integrierter WebView

und je nach Anwendung auch eine stufenweise Migration unterschiedlicher Benutzergruppen.

Bei der Rahmenanwendung im Web lässt sich dabei frei zwischen den verfügbaren JavaScript-Frameworks und den zu nutzenden Verfahren zur Gestaltung der Datenflüsse wählen. In die Auswahl sollte dabei miteinbezogen werden, dass sich manche Frameworks, wie AngularJS, aufgrund der verwandten Konzepte für Java-Entwickler eher anbieten, als solche mit ungewohnten Mustern. Das folgende Beispielprojekt ist mit AngularJS 1.4 umgesetzt und lässt sich sowohl per Swing als auch nur mit einem Webbrowser nutzen.

Ein Maven-Projekt als Beispiel

Zur Veranschaulichung gibt es eine beispielhafte Umsetzung als Maven-Projekt unter [3]. Die Klasse *Swing-Integration* liefert dabei alles, was man braucht: Einen lokalen Webserver zur Simulation des Backends, eine Swing-basierte Rahmenanwendung und die JavaScript-Anwendung zur Integration auf dem Desktop. Im Beispiel kommen AngularJS und Bootstrap zum Einsatz. Diese werden in diesem Fall aus dem Internet nachgeladen. Nachdem die Klasse gestartet wurde, kann parallel unter dem folgenden URL per Browser die pure JavaScript-Anwendung genutzt werden: *http://localhost:8081/sample.html*.

Wenn man den Zugriff über Browser und über die Swing-Anwendung vergleicht, bemerkt man unterschiedliche Einträge unter dem Punkt *Invoices* – dies demonstriert die Implementierung des Datenzugriffs über Swing bzw. JavaScript. Im Java-Code ist dies die Methode *getInvoices()*, die für die WebView exportiert wird. In der nativen Browserversion werden die Daten durch den *InvoiceService* direkt bereitgestellt.

Der Zugriff auf die *Contracts* erfolgt durch Ajax-HTTP-Zugriffe, sowohl im Browser als auch in der Swing-Anwendung. Unter Contracts wird der Rückweg aus dem Browser zum Swing demonstriert, durch den Knopf SEND EMAIL wird auf dem Desktop ein Swing-Dialog geöffnet, im Browser ein HTML-Dialog.

Listing 1: Zugriff auf die Invoices im AngularJS-Service

```
app.factory('InvoiceService', function () {
  var api = {
    getInvoices: function () {
      var invoices = [];
      var invoice = {number: "acme-2", date: "2015-11-01", companyId: 1, paid: false };
      invoices.push(invoice);
      return invoices;
    }
  };
  if (window.appFrame) {
    api.getInvoices = function () {
      var data = window.appFrame.getInvoices();
      return JSON.parse(data);
    };
  }
  return api;
});
```

Wege zur Migration von Enterprise-Anwendungen

Mit den gezeigten Mitteln ist damit eine schrittweise Migration von typischen Enterprise-Anwendungen möglich. Die konkrete Umsetzung ist im Wesentlichen auf zwei Wegen denkbar:

- Möglichst schnelle Umsetzung von (teil-)autonomer Funktionalität als Webanwendung. Diese kann dann auf jedem webfähigen Endgerät genutzt werden und erlaubt unter Umständen schon ganzen Nutzergruppen, auf die Desktopanwendung zu verzichten.
- Langfristiger Einsatz einer Desktoprahmenanwendung. Hier ist zwar das Ziel, schon kurzfristig von den Möglichkeiten einer Webanwendung zu profitieren und eine langfristige Migration ins Web zu erzielen, jedoch sind die Webanteile zwingend auf die Rahmenanwendung angewiesen und nicht separat lauffähig.

Ob man sich für den Weg entscheidet, einzelne Teile bereits als reine Webanwendung autonom betreiben zu können, oder bis zum Schluss eine Rahmenanwendung beibehält, hängt stark davon ab, ob ein Teilbetrieb im Web sinnvoll ist und auch in welchem Zeitraum die Migration durchgeführt werden soll. Lassen sich bereits einzelne Anwendungsfälle mit der reinen Webtechnologie nutzen, hat man die Chance auf frühes Nutzerfeedback: Stimmen z. B. Performance, Bedienkonzepte und die Nutzung auf Geräten mit unterschiedlichen Bildschirmformaten und -auflösungen? Auf jeden Fall sollte man einplanen, dass nach mehreren Jahren Nutzung der Desktopanwendung die Umstellung ins Web für den einen oder anderen Anwender ein kleiner Kultur-

schock ist. Umso wichtiger ist die Möglichkeit durch einen frühzeitigen und engen Dialog mit den Anwendern nachsteuern zu können. Die **Abbildungen 7** und 8 zeigen exemplarisch die Phasen einer Enterprise-Java-Anwendung bei der Migration auf dem Weg von einer reinen Swing-Anwendung hin zu einer Webanwendung.

Das unterschiedliche Styling, durch das ein optischer Bruch entsteht, ist für dieses Beispiel bewusst gewählt worden. In der Praxis können hier Anpassungen über CSS erfolgen, die ein harmonisiertes Erscheinungsbild ermöglichen.

Die Anwendung ist nach dem typischen Model-View-Controller-Muster aufgebaut. Die bidirektionale Integration zwischen Swing und Web wird zum einen deutlich, wenn zwischen den verschiedenen Funktionen der Anwendung umgeschaltet wird. Zum anderen reagiert die Statusleiste im unteren Bereich der Anwendung auf Aktivitäten in der WebView. Auch der Dialog zum E-Mail-Versand, der als Swing-Formular angezeigt wird, kann nahtlos aus einer Aktion im Webanteil aufgerufen werden.

Umparken im Kopf

Vorteil der sanften Migration ist, dass auch das Unternehmen Erfahrungen mit Webtechnologie schrittweise sammeln kann. Der Zwang im ersten Anlauf gleich alles richtig zu machen, entfällt damit. Neben der hohen Entwicklungsgeschwindigkeit von JavaScript-Frameworks spielt auch der Faktor Mensch eine wichtige Rolle und profitiert von entsprechenden zeitlichen Puffern. Gerade im Java-Enterprise-Umfeld ist Webentwicklung – allem voran JavaScript – oft mit gewissen Vorbehalten belastet: JavaScript sei keine ernstzunehmende Sprache, es fehle ein statisches Typensystem, es gäbe keine professionelle Werkzeugunterstützung und testgetriebene Entwicklung mit Refactoring sei nicht machbar. Mit dem Google Web Toolkit ist jedoch die Entwicklung mit den gewohnten Werkzeugen in der gewohnten Sprache möglich.

Die Herausforderungen beschränken sich jedoch gerade im Enterprise-Umfeld nicht nur auf bestehende Anwendungen und große Codebasen, die weiter entwickelt werden wollen, sondern es gibt auch bestehende Mitarbeiter mit entsprechenden Skills. In manchen Firmen haben Mitarbeiter nicht nur Vorbehalte, sondern echte Schwierigkeiten damit, außerhalb der gewohnten Java-Welt zu entwickeln. Die erlebte Herausforderung, sich mit JavaScript, HTML und CSS zu befassen, ist sicherlich individuell unterschiedlich stark ausgeprägt. Eine dauerhaft tragfähige Lösung ist die wie auch immer geartete Abstraktion oder gar Isolation der Webwelt von Java-Entwicklern jedoch nicht. Auch dafür lässt sich der in diesem Artikel vorgestellte Ansatz optimal nutzen: Während Entwickler, die Webtechnologien als spannende Chance sehen, sich weiter zu Pionieren entwickeln werden, können sich eher konservative Entwickler um Wartung und Weiterentwicklung des in Java entwickelten Anwendungsteils kümmern.

Auch die Architekten müssen sich etwas umgewöhnen. Anders als in der Enterprise-Java-Welt gilt es vorsichtig damit umzugehen, eigene Frameworks zu

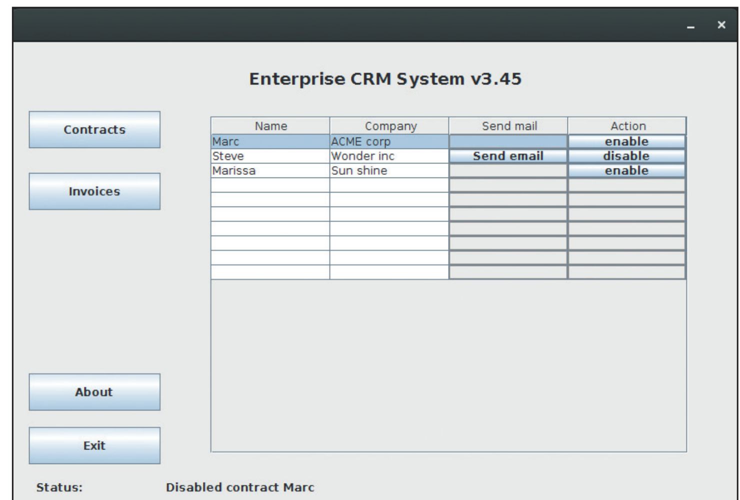


Abb. 7: Eine Swing-Desktopanwendung mit typischen Elementen von formularbasierten Geschäftsanwendungen

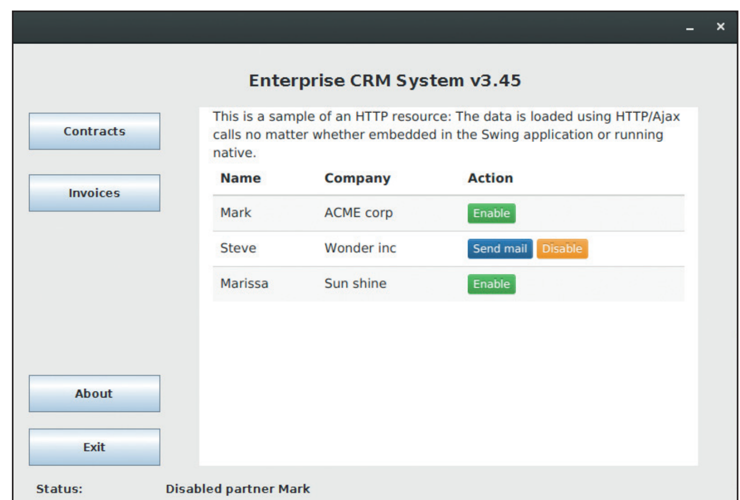


Abb. 8: Nach der teilweisen Migration: Ein Formular ist durch die WebView abgelöst; die Funktionalität lässt sich aus dem Webbrowser oder in der Swing-Anwendung nutzen

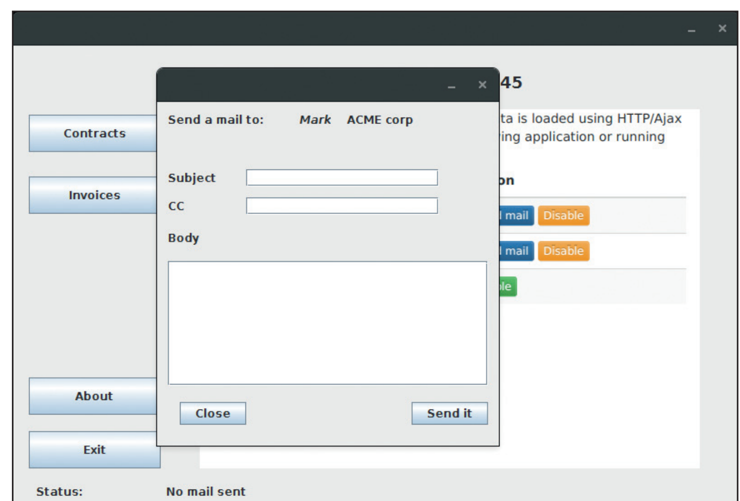


Abb. 9: Ein nativer Swing-Dialog wird als Resultat auf einen Knopfdruck in der WebView angezeigt

entwickeln. War man in der Java-Welt eigentlich erst dann als echtes Enterprise-Projekt ernst zu nehmen, wenn man ein eigenes Persistenz- und Logging-Framework entwickelt hatte, kann sowas in der Webwelt dazu führen, dass keine erfolgreiche Evolution von Architektur und Technologie möglich ist. Entsprechend vorsichtig sollte hier vorgegangen werden. Optimaler Weise ist eine möglichst lose Kopplung von gewähltem Framework zur Frontend- und Geschäftslogik im Frontend vorgesehen. Anders als das langfristig stabile Swing ist hier von einer anderen Änderungsfrequenz auszugehen. Auch wenn heute eine Eigenentwicklung in Qualität und Funktionsumfang vorhandene Lösungen übertrifft, kann nächste Woche schon ein neuer Open-Source-Stern aufgehen. Dann sollte man in der Lage sein, sich dem anzupassen – nicht zuletzt auch aus wirtschaftlichen Gesichtspunkten.

Ist man sich dessen bewusst, dass es im Enterprise-JavaScript-Umfeld zu schnellen Entwicklungen kommt, und plant dies entsprechend ein, kann eine passende Abstraktionsschicht zwischen Fremdbibliotheken und eigener Anwendung sehr wohl sinnvoll sein: Änderungen unter der Haube, z. B. der Austausch des Build-Systems, haben optimaler Weise keinen Einfluss auf Entwickler mit dem Fokus auf die Anwendungsentwicklung. Durch bewährte Muster wie Komposition statt Vererbung, Dependency Injection und Vorgehensweisen wie Test-driven Development ist auch in der JavaScript-Welt eine langfristig wartbare und erweiterbare Anwendung möglich.

Alternativen zu JavaFX

Die vorgestellte Architektur ist auf andere Plattformen, wie .NET oder native Anwendungen, analog übertragbar. In den Fällen wird dann zwar eine andere technische Implementierung benutzt, die Konzepte und auch APIs sind jedoch ähnlich. Ein Beispiel für eine andere Implementierung ist das Chromium Embedded Framework, kurz CEF [4]. Von CEF gibt es eine Java-Portierung namens JCEF, die mittels JNI arbeitet und keine JavaFX-Abhängigkeit besitzt. Der Integrationsansatz ist dabei nicht nur auf Migrationsprojekte beschränkt: Es gibt viele Nutzer von CEF, die damit Anwendungen entwickeln, die von den Möglichkeiten beider Welten profitieren. In der Liste der CEF-Nutzer finden sich namhafte Beispiele wie Spotify, Amazon Music und der Stream-Client [5].

Fazit und Ausblick

Der vorgestellte Ansatz erlaubt eine sanfte Migration hin zu einer API-gestützten Architektur, die nicht nur Desktop und Browser als Clients unterstützt. Durch HTTP-Endpunkte ist es leicht möglich, Dienste für externe Partner bereitzustellen oder mobile Clients mit nativen Anwendungen zu unterstützen. Ebenso bietet eine Architektur mit definierten HTTP-APIs die Möglichkeit, unterschiedliche Anwendungen für verschiedene Nutzergruppen bereitzustellen: Während der Mitarbeiter im Backoffice ein Expertensystem nutzt, steht ein Wizard-System für Endkunden im Internet bereit.

Die Herausforderungen bei der Migration sollen jedoch nicht verschwiegen werden: Um eine nahtlose Integration zu realisieren, sind Punkte wie eine gemeinsame Anmeldung und Abmeldung für alle Bestandteile der Anwendung zu lösen. Bei einem angenommenen Zeithorizont von Jahren ist der damit einhergehende Aufwand für redundante Implementierung von Funktionalität und Querschnittsfunktionalität einzuplanen.

Insgesamt stellt sich die Frage, in welcher Reihenfolge die Funktionalitäten der Anwendung migriert werden sollen. Als Anhaltspunkte können da verschiedene Kriterien dienen: Welche Teile werden in der nächsten Zeit weitere Entwicklung erfahren und können dann sinnvoll in die neue Welt verlegt werden? Soll eine autonome Nutzung im Browser frühzeitig möglich sein, und welche Grundfunktionen sind dafür erforderlich?

Auch lassen sich Bedienkonzepte von sehr reichhaltigen Desktopanwendungen oft nicht sinnvoll eins zu eins auf eine Webanwendung übertragen. Sollen diese beiden Welten nahtlos miteinander verbunden werden, kann es erforderlich sein, die bestehenden Bedienkonzepte zu hinterfragen und für eine Nutzung im Web zu harmonisieren.

Im Test ist aufgefallen, dass die Anwendung unter Windows und OS X optisch einwandfrei war, unter Linux hingegen Artefakte zu sehen waren. Hier bleibt zu hoffen, dass Oracle mit zukünftigen JavaFX-Versionen Abhilfe schafft.

Thomas Kruse hat Informatik studiert und ist als Architekt, Berater und Coach für die trion development GmbH tätig. In seiner Freizeit engagiert er sich im Open-Source-Umfeld und leitet die Java Usergroup in Münster.

Links & Literatur

[1] AngularJS: https://angularjs.org/

[2] Java FX WebView: http://docs.oracle.com/javafx/2/webview/jfxpub-webview.htm

[3] Repository zum Artikel auf GitHub: https://github.com/trion-development/java-web-bridge

[4] CEF: https://bitbucket.org/chromiumembedded/cef

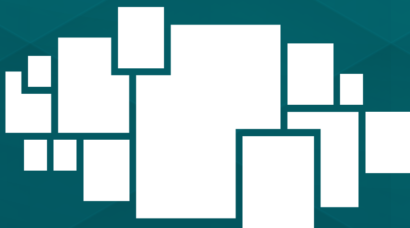
[5] CEF-Anwendungen: https://en.wikipedia.org/wiki/Chromium_Embedded_Framework#Applications_using_CEF

Javamagazin³

Jetzt abonnieren und 3 TOP-VORTEILE sichern!

1 Alle Printausgaben frei Haus erhalten

2 Im entwickler.kiosk immer und überall online lesen – am Desktop und mobil



3 Mit vergünstigtem Upgrade auf das gesamte Angebot im entwickler.kiosk zugreifen

Java-Magazin-Abonnement abschließen auf www.entwickler.de

S&S Media Group www.entwickler.de entwickler.kiosk