Mobile Oracle-basierte Applikationen für verschiedene Endgeräte

Torsten Rieger
Horus software GmbH
Ettlingen

Schlüsselworte

Oracle XML DB, Sencha Touch 2, Mobile Applikationen, HTML5

Einleitung

Der Einsatz mobiler Endgeräte wird zunehmend wichtiger. Dies betrifft nicht nur den mobilen Zugriff auf datenorientierte Business-Applikationen wie bspw. CRM- oder ERP-Systeme, sondern auch Applikationen mit hohen Anforderungen bei der grafischen Benutzerschnittstelle, wie dies beispielsweise bei einem Geschäftsprozessmodellierungswerkzeug der Fall ist. Aufbauend auf einem Oracle-basierten Repository wird in diesem Beitrag gezeigt, wie solche mobile Applikationen elegant für unterschiedliche mobile Plattformen realisiert werden können. Unter Verwendung von Oracle- und Java-Technologien und eines entsprechenden Frameworks (Sencha Touch) wurden im Rahmen der Erweiterung eines Geschäftsprozessmodellierungswerkzeugs anspruchsvolle grafische Oberflächen implementiert, die ein Navigieren und Ändern in komplexen hierarchisch aufgebauten Geschäftsprozessmodellen ermöglichen.

Projektrahmen: Horus Methode™ und Werkzeuge

Der Begriff Horus steht stellvertretend sowohl für eine Methode zur Modellierung von Geschäftsprozessen, als auch für eine Sammlung von Werkzeugen, die den Anwender nicht nur bei der Modellierung, sondern während des gesamten Geschäftsprozessmanagements unterstützt. Eine Kernfunktion nimmt dabei der Horus Business Modeler ein, dabei handelt es sich um ein besonders leicht und intuitiv zu bedienendes Tool zur grafischen Modellierung und Simulation von Geschäftsprozessen. Auf Basis wissenschaftlich fundierter Konzepte werden aber auch weitere einfach zu bedienende Werkzeuge geboten, die in Verbindung mit der Horus Methode™ ein durchgängiges agiles Modellierungsverfahren unterstützen. Eingebunden in eine Web 2.0 Collaboration-Plattform unterstützt Horus Enterprise das kooperative Arbeiten in Business Communitys (Social BPM), auch über Unternehmensgrenzen hinweg.

Horus Knowledge Explorer

Ein wichtiger Aspekt von Social BPM im Rahmen der Horus Methode™ ist die Einbeziehung möglichst aller relevanten Betroffenen innerhalb eines Projektes. Doch im Gegensatz zu (angehenden) Geschäftsprozess-Experten, wollen – und sollen – sich die Betroffene nicht mit der Modellierung selbst und den unterstützenden Werkzeugen auseinandersetzen müssen. Beispielsweise bei Review von Prozessen mit den Fachabteilungen soll den Fachexperten eine einfache Möglichkeit geboten werden, die betreffenden Prozesse zu kontrollieren, Korrekturvorschläge einzubringen und Kommentare zu verfassen. Um dies zu Realisieren, wird eine einfach zu bedienende Wiki-Software eingesetzt, deren Datenbestand auf Initiative des Geschäftsprozess-Experten mit einem zentralen datenbankbasierten Prozess-Repository synchronisiert werden kann. Somit kann mit geringem Einarbeitungsaufwand ein Wissenstransfer mit den Fachabteilungen hergestellt werden.

Doch gerade solche ereignisgetriebenen Prozesse wie ein Review, müssen nicht zwingenderweise am eigenen Arbeitsplatz stattfinden. Oft werden solche Aufgaben unterwegs, z.B. in Zügen oder an den Projektstandorten selbst, ausgeführt. Dazu sollte eine plattformübergreifende Lösung für mobile Endgeräte entwickelt werden, die unabhängig vom Standort Zugriffe und Einflussnahme auf Geschäftsprozessmodelle und zugehörige Artefakte ermöglicht. Die Lösung sollte einfach zu nutzen sein, besonders im Hinblick auf berührungsempfindliche Bildschirme, kurz: Ein Horus Knowledge Explorer für mobile Endgeräte.

Technische Aspekte der Datenquelle

Wie bereits erwähnt ist der Horus Business Modeler eine Kernkomponente der Horus Werkzeug-Palette, damit werden von Geschäftsprozess-Experten die Prozessmodelle modelliert. Der Horus Business Modeler in der Enterprise-Edition nutzt zur Persistierung ein Repository, dass auf der Oracle XML DB basiert. Dazu werden die Prozessmodelle in XML, konkret in der Petri Net Markup Language (PNML) und im XML Metadata Interchange (XMI), serialisiert. Anschließend werden die XML-Daten über eine SOAP-Schnittstelle an den Horus Repository Server übergeben und dort in einer einfachen Tabellenstruktur der Oracle XML DB abgelegt. Hierfür wurden, wie in Listing 1 exemplarische gezeigt, innerhalb der XML DB Tabellen mit Spalten vom Typ XMLTYPE angelegt.

```
CREATE TABLE HORUS_OBJECTS
 (ID NUMBER(32) NOT NULL
 ,PETRI_XML XMLTYPE
 ,AOM_XML XMLTYPE
 ,SHM_XML XMLTYPE
 ,CREATED DATE NOT NULL
 ,CREATED_BY VARCHAR2(30) NOT NULL
 ,UPDATED_BY VARCHAR2(30)
 ,UPDATED DATE
 )
 TABLESPACE HORUS_OWNER_DATA
 XMLTYPE COLUMN PETRI_XML
  store as clob (tablespace HORUS_OWNER_XML)
 XMLTYPE COLUMN AOM_XML
  store as clob (tablespace HORUS_OWNER_XML)
 XMLTYPE COLUMN SHM_XML
  store as clob (tablespace HORUS_OWNER_XML)
/
```

Durch das Ablegen der Modelle in der Oracle XML DB wird es möglich mit Querys auf die Modelldaten innerhalb der XML Dateien zuzugreifen. So können wie in Listing 2 gezeigt, auch relationale Views auf Grundlage dieser Daten angelegt werden.

```
CREATE OR REPLACE VIEW HORUS_OWNER.HORUS_ACTIVITIES AS
select
  obj.id diag_id
  extractvalue(Value(tra), '/transition/@id') act_id
, extractvalue(Value(tra), '/transition/viewName/text/text()') short_name
, to_number( extract( Value(tra), '/transition/position/@x' ),
'9999999999999999' ) x
, to_number( extract( Value(tra), '/transition/position/@y' ),
'9999999999999999' ) y
, to_number( extract( Value(tra), '/transition/dimension/@x' ),
'9999999999999999' ) width
, to_number( extract( Value(tra), '/transition/dimension/@y' ),
'9999999999999999' ) height
```

```
from
  HORUS_OWNER.horus_objects obj
, table(XMLSequence(extract(obj.petri_xml, '/pnml/net/page/transition')))
tra
/
```

Dass dabei die XML Daten unstrukturiert (als CLOB) persistiert werden ermöglicht, dass die Daten ohne zu registrierendes Schema abgelegt werden können. Jedoch entfällt damit auch die automatische Typisierung von Elementtexten und Attributwerten, weshalb die XML DB vom Typ VARCHAR2 ausgeht, was die Nutzung entsprechender Funktionen zur Typumwandlung in Querys (und damit auch in den Views) nach sich zieht. Die CLOB-Persistierung hat auch Auswirkungen auf Update-Operationen, denn bedingt durch die mangelnde Kenntnis der XML DB über die Struktur der XML-Dokuments, müssen bei UPDATE-Operationen die gesamten XML Daten der Datei ersetzt werden. Konkret werden, wie in Listing 3 zu sehen, die aktuellen XML Daten gelesen, die Manipulationen ausgeführt und dann die geänderten XML Daten zurückgeschrieben. In Listing 3 wird die Spalte „short_name“ der zuvor erstellten und auf XML Daten basierenden relationalen View für modellierte Aktivitäten, mithilfe eines Triggers, um Update-Fähigkeit erweitert.

```
CREATE OR REPLACE TRIGGER HORUS_OWNER.UPDATE_ACT_SHORT_NAME_TRG
INSTEAD OF UPDATE ON HORUS_OWNER.HORUS_ACTIVITIES
REFERENCING OLD AS oldAct NEW AS newAct
BEGIN
 UPDATE horus_objects
 SET petri_xml = updateXML(
  petri_xml,
  '/pnml/net/page/transition[@id="' ||
    :oldAct.act_id ||
    '"]/viewName/text/text()',
  DBMS_XMLGEN.CONVERT(:newAct.short_name,0)
 )
 WHERE id = :oldAct.diag_id;
END;
```

Damit ist die XML basierte Datenquelle mit Unterstützung durch die Oracle XML DB um erste grundlegende Lese- und Editierfunktionalitäten erweitert worden.

Realisierungskonzepte für mobile Endgeräte

Auf Seite der mobilen Endgeräte gibt es verschiedene zielführende Konzepte, wie eine mobile Applikation realisiert werden kann. Es werden drei Grundkonzepte unterschieden:

- **Native Applikationen** müssen für jede mobile Plattform separat entwickelt werden. Dazu wird in der jeweils plattformspezifischen Programmiersprache und unter Einsatz der, von den Plattformanbietern angebotenen, Entwicklungs-Frameworks entwickelt. Das schließt natürlich auch die clientseitige Geschäftslogik mit ein, so dass bei n Zielplattformen auch der n-fache Entwicklungsaufwand aufgebracht werden muss. Dem gegenüber steht jedoch die Möglichkeit, die jeweils angebotenen plattformspezifischen Funktionalitäten der Endgeräte vollständig nutzen zu können, das schließt neben Sensoren vor allem auch die plattformnahen Optimierungsmöglichkeiten, z.B. für anspruchsvolle Benutzerschnittstellen oder die Ausgabe von 3D-Inhalten auf dem Bildschirm, mit ein. So eignet sich dieses Konzept primär für grafisch anspruchsvolle Applikationen wie Computerspiele oder Anwendungen, die ihren Fokus auf eine hohe Ausführungsgeschwindigkeit legen.

- **Webapplikationen** werden vollständig mit Web-Technologien, d.h. HTML, CSS und JavaScript, entwickelt. Durch den breiten Einsatz moderner Browser- und Web-Technologien (Webkit-Rendering-Engine und HTML5) auf mobilen Endgeräten, können solche Applikationen, basierend auf einer Implementierung und mit nur wenigen Anpassungen, auf vielen Plattformen ausgeführt werden. Jedoch sind hierbei nur Zugriffe auf plattformspezifische Funktionalitäten möglich, die durch die Webbrowser zur Verfügung gestellt werden.
  Die angebotenen Funktionalitäten können jedoch durch plattformspezifische Wrapper (PhoneGap, Cordova) erweitert werden. Hierzu werden durch den Wrapper plattformnahe Schnittstellen gekapselt und als JavaScript-Methoden in die Laufzeitumgebung der Webapplikation integriert. Zusätzlich können optionale Erweiterungen eingebunden werden, die eine native Ausführung rechenintensiver Arbeitsschritte erlauben (bspw. Barcode-Scanner). Diese Wrapper ermöglichen auch die Deployment-Prozesse für native Anwendungen zu nutzen, womit die Webapplikationen auch über die plattformspezifischen Vertriebskanäle verteilt werden können.
- **Hybride Applikationen** stellen ein Mischkonzept dar, denn bei hybriden Applikationen wird in einer einheitlichen Programmiersprache (meist JavaScript) entwickelt, plattformspezifische Funktionalitäten werden dabei über Schnittstellen – vergleichbar mit den Wrappern bei Webapplikationen – in die Laufzeitumgebung integriert. Ganz im Gegensatz zu Webapplikationen, werden jedoch bei diesem Konzept auch native Oberflächenkomponenten gekapselt, so dass die Oberfläche den Eindruck und das Bedienerlebnis einer vollständig nativen Applikation bieten kann. Jedoch werden hierzu die angebotenen Oberflächenkomponenten zumeist auf den kleinsten gemeinsamen Nenner reduziert, um eine möglichst plattformunabhängige Entwicklung anzubieten. Eine weit verbreitetes Entwicklungs-Framework, dass das Konzept der hybriden Applikationen unterstützt, bietet das Unternehmen Appcelerator mit seinem Produkt Titanium Mobile.
  Jedoch können die unterschiedlichen Speichermanagement-Konzepte von JavaScript und der nativ implementierten Wrapper für Oberflächenkomponenten dazu führen, dass ungenutzter Speicher nicht mehr freigegeben wird, was gerade komplexe Applikationen mit langer Laufzeit langsam werden lässt und zu nicht nachvollziehbaren Abstürzen führen kann.

Sencha Touch

Ein Framework, welches das Konzept der Webapplikationen vertritt, ist Sencha Touch 2. Dabei handelt es sich um ein ausgewachsenes Model-View-Controller (MVC) Framework, dass JavaScript um ein eigenes Klassensystem und gängige Bibliotheksfunktionen erweitert. Neben den wichtigsten Adaptern zur Anbindung externer Datenquellen, bietet es vor allem zahlreiche plattformunabhängige Oberflächenkomponenten. Die primäre Orientierung an dem erfolgreichen Bedienerlebnis der iOS-Plattform ist zwar unbestreitbar, jedoch sind zunehmende Tendenzen zu alternativen Plattformen in der Entwicklung zu erkennen. Und gerade durch den hohen Freiheitsgrad von HTML5 und CSS sind die Standardoberflächenkomponenten von Sencha Touch nicht als Einschränkung zu verstehen, sondern demonstrieren vielmehr die gebotenen Möglichkeiten. Der größte Vorteil gegenüber vielen anderen Frameworks (z.B. jQuery Mobile) liegt darin, dass Sencha Touch schon konzeptionell vorsieht, die gesamte Geschäfts- und Darstellungslogik clientseitig auszuführen. Somit entfällt in den meisten Fällen die Wartezeit für Antworten von einer serverseitigen Komponente während der Benutzerinteraktion mit der Applikation. Und sollte doch ein Datenaustausch mit einem entfernten System notwendig sein, so müssen nur die nötigsten Nutzdaten übertragen werden.

Schnittstellenkonzept

Da es Ziel war lediglich eine ergänzende mobile Variante des Horus Knowledge Explorer zu realisieren, sollten bereits vorhandene Schnittstellen genutzt werden. Der Horus Business Modeler nutzt zur Kommunikation mit dem Horus Repository Server eine SOAP-Schnittstelle, welche eine Operation für serverseitige funktionale Erweiterungen vorsieht. In Listing 4 ist die zugehörige Webservice-Definition zu sehen.

```
<wsdl:operation name="callCustomExtension">
  <wsdl:input message="ns1:callCustomExtension"
    name="callCustomExtension"></wsdl:input>
  <wsdl:output … ></wsdl:output>
  <wsdl:fault … ></wsdl:fault>
</wsdl:operation>
```

Dabei werden Anfrageparameter erwartet, welche in Listing 5 gezeigt werden, der Rückgabewert ist vom einfachen Typ String.

```
<wsdl:message name="callCustomExtension">
  <wsdl:part name="nodeType" type="xsd:int"></wsdl:part>
  <wsdl:part name="nodeId" type="xsd:long"></wsdl:part>
  <wsdl:part name="extensionId" type="xsd:string"></wsdl:part>
  <wsdl:part name="parameters" type="xsd:string"></wsdl:part>
</wsdl:message>
```

Was zu einer, wie in Listing 6 zu sehen, sehr einfachen Java-basierten Server-Komponente als Erweiterung des Horus Repository Server führt.

```
public class HorusMobileServerCustomExtension implements ICustomExtension {
  private Gson gson = new GsonBuilder().setDateFormat(
      "yyyy-MM-dd'T'HH:mm:ssZ").create();

  @Override
  public String callCustomExtensionMethod(IHorusContainer base,
      HorusSecurityManager sm, String parameters)
      throws SecurityManagerException {
    JSONCall call = gson.fromJson(parameters, JSONCall.class);
    String result = null;

    if (call.getMethod().equals(Constants.GET_CHILDREN))
      List<JSONResource> res =
        ResUtil.getJSONChildren(call.getData(), base, sm);
      result = gson.toJson();
    } else ...

    return result;
  }
}
```

Dabei wird vorerst auf die Automatismen der Java Persistence API (JPA) verzichtet und stattdessen Google Gson zum (de-)serialisieren in die JavaScript Object Notation (JSON) verwendet. Gson benötigt dabei keine Annotationen in den zu serialisierenden Klassen, so können auch Klassen genutzt werden, die nicht geändert werden dürfen oder deren Quellcode nicht verfügbar ist.

In Abb. 1 ist eine Gesamtübersicht der einzelnen Komponenten, ihrer Zusammenhänge und Kommunikationsmitte und -wege dargestellt.

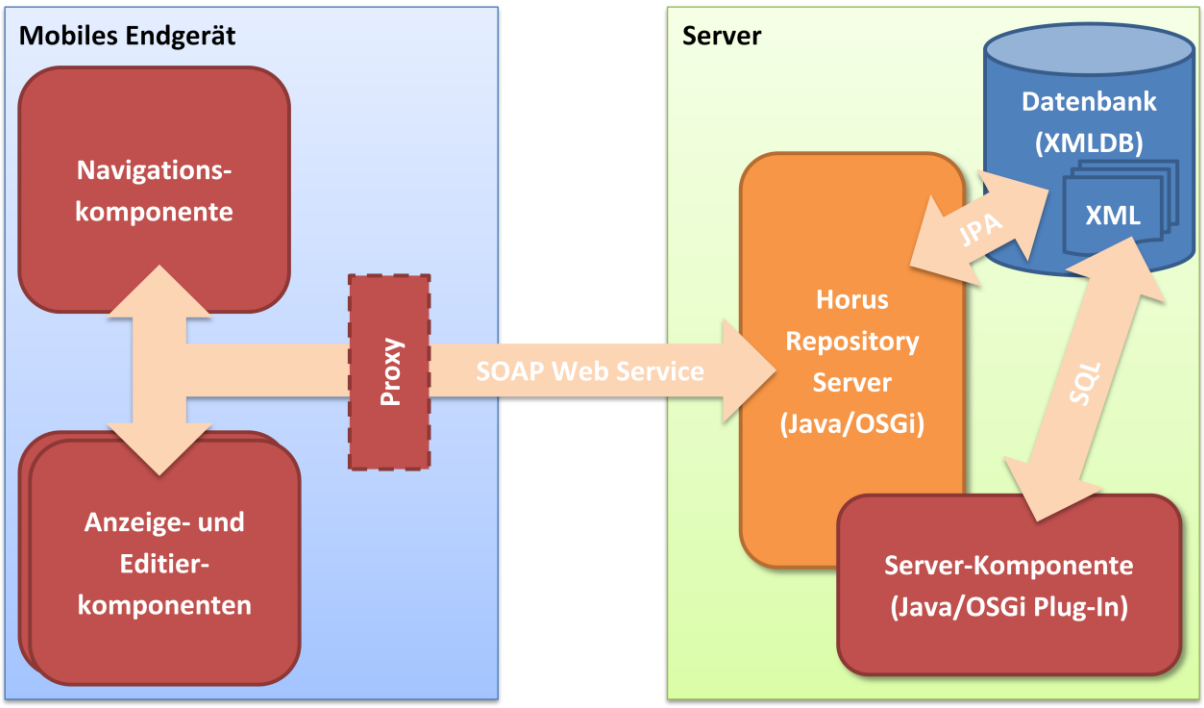

Abb. 1: Übersicht der Komponenten des Horus Knowledge Explorer für mobile Endgeräte und der serverseitigen Komponenten

Mobile Applikationen mit Sencha Touch

Zur Realisierung der Applikation für mobile Endgeräte wurde Sencha Touch eingesetzt, welches die Entwicklung von plattformunabhängigen komplexen Webapplikationen mit einer hochwertigen Softwarearchitektur und einem guten Bedienerlebnis für den Endanwender ermöglicht. Zur Entwicklung wurde die Entwicklungsumgebung Sencha Architect 2 eingesetzt, da diese die Framework-spezifischen Eigenschaften und damit den Entwickler bei seiner täglichen Arbeit mit dem Framework unterstützt.

Mit Sencha Touch realisierte Webapplikationen bestehen im Wesentlichen aus einer Vielzahl von Komponenten:

- **Models** beschreiben Typen von Datenobjekten, zumeist Geschäftsobjekte, im Allgemeinen durch die Menge ihrer Attribute. Dabei können sogar komplexe Modelhierarchien definiert werden.
- **Views** sind Anzeige- oder Editiermasken und können weitere Oberflächenkomponenten enthalten. Sencha Touch liefert bereits eine Auswahl wichtiger Views, z.B. Panel, Form Panel, Overlay Panel, Navigation View, Action Sheet, Carousel, List, Nested List, Data View und weiterer Komponenten, z.B. Button, Textfield, Map, Video, Date Picker, Picker, usw., mit, wie sie auf mobilen Endgeräten üblich sind.
- **Controllers** werden genutzt um die Geschäftslogik zu implementieren. Sie referenzieren zugehörige Views bzw. innere Komponenten und ermöglichen durch Event Binding mit Hilfe von einfachen Selektoren auf geschäftsrelevante Ereignisse von einzelnen oder einer Menge von Komponenten, zu reagieren.

- **Stores** stellen einen lokalen Datenspeicher da und können eine Menge von Elementen eines konfigurierten Models, also Datentyps, aufnehmen. Werden sie an Views gebunden, die auf einer Data View basieren, so wird deren Inhalt automatisch synchronisiert. Einem Store kann neben einem Model auch ein Proxy zugewiesen werden, welcher die Anbindung externer Datenquellen ermöglicht. Sencha Touch liefert die geläufigsten Proxys mit, konkret sind dies: Ajax, JSON-P, LocalStorage, SessionStorage, Memory, Rest, SQL.
  Leider wird der recht neue SOAP Proxy des Schwester-Frameworks für Desktopumgebungen noch nicht für Sencha Touch angeboten.
- **Profiles** werden endgeräteabhängig aktiviert und bieten die Möglichkeit zur Laufzeit jeweils spezialisierte Klassen (Views, Controller…) entsprechend dem Endgerätetyp zu laden.

Die Besonderheit des Sencha Touch Frameworks, dass es von anderen vergleichbaren Frameworks abgrenzt, wird klar: Die Beschreibung der Oberfläche mit HTML entfällt fast vollkommen, alle Komponenten der Applikation basieren auf JavaScript und werden zur Laufzeit dynamisch generiert.

Realisierung der mobilen Applikation

Die Horus Knowledge Explorer Webapplikation besteht im Wesentlichen aus zwei Komponenten: Einer Navigationskomponente und einer Anzeige- bzw. Editierkomponente. Diese beiden Komponenten werden, wie in Abb. 2 zu sehen, in einer umrahmenden Ansicht in zwei nebeneinander angeordneten Bereichen dargestellt.

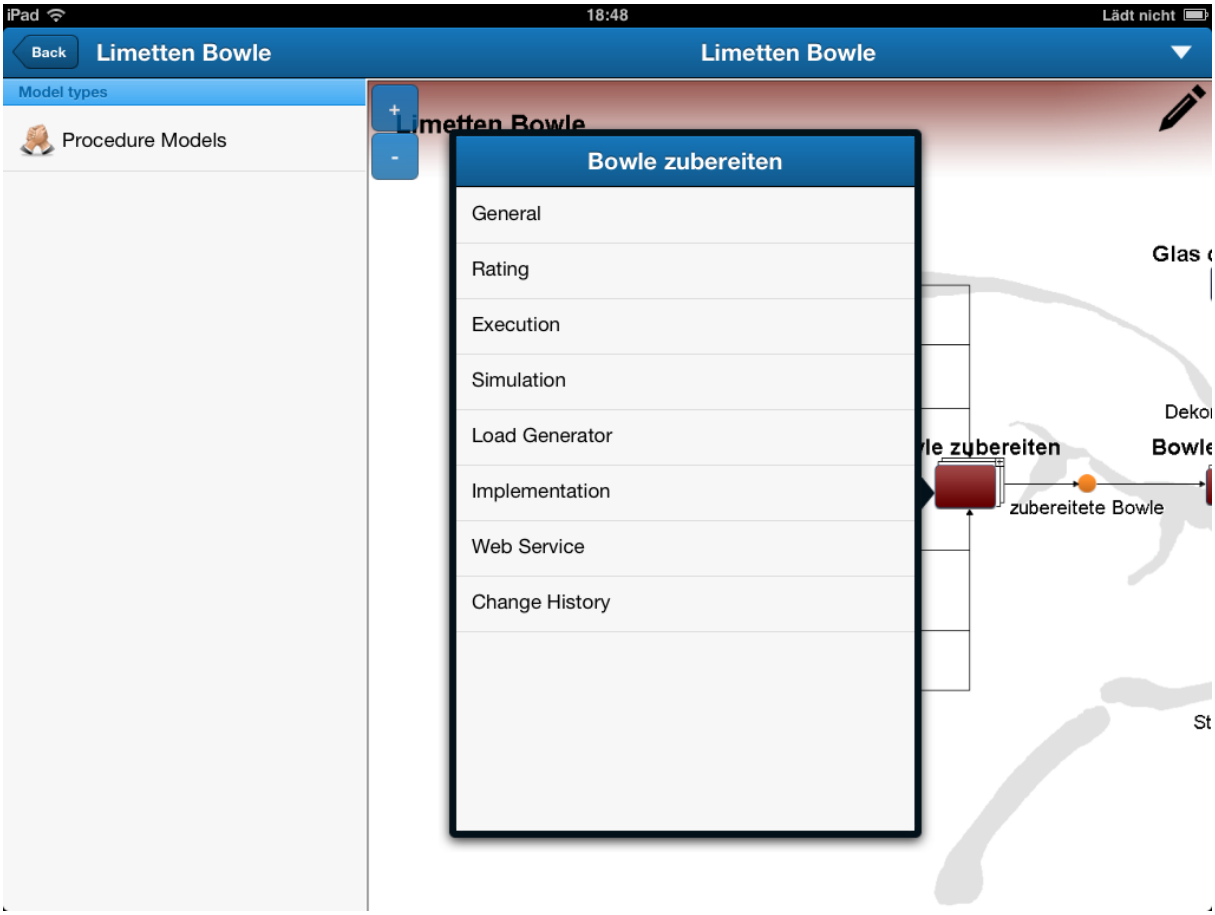

Abb. 2: Screenshot des Horus Knowledge Explorer für mobile Endgeräte

Die Navigationskomponente (Abb. 2, links) basiert auf der Standardkomponente Navigation-View, welche einen Stapel von Ansichten verwalten kann und einzig das Zurücknavigieren (Zurück-Schaltfläche) unterstützt. Innerhalb der Navigation-View werden Listen angezeigt, zuerst lokal gespeicherte Verbindungsinformationen zu Horus Repository Servern, bei jeder Auswahl durch Antippen wird eine neue Liste auf den Stapel der Navigation-View gelegt und mit einer Animation eingeblendet. In Listing 7 wird gezeigt, wie einfach eine neue Liste (hier: WorkspaceExplorerListView, eine unwesentlich vorkonfigurierte Standardliste) erstellt, initial konfiguriert und anschließend auf den Stapel gelegt werden kann.

```
var newList = Ext.create('HKE.view.WorkspaceExplorerListView', {
  title: parent.get('name'),
  itemId: parent.get('id'),
  store: { // Store konfigurieren und setzen
    model: 'HKE.model.Resource',
    proxy: {
      type: 'HKE.proxy.HorusSOAPProxy', // Horus spez. SOAP Proxy
      method: 'GET_CHILDREN', // s. Methode in Listing 6
      connection: connection, // SOAP (URL & Benutzer & Passwort)
      base: parent // Parent-Workspace, s. Base in Listing 6
    }
  }
});
navigationView.push(newList);
```

Gut zu erkennen ist auch, dass unterschiedliche Komponenten schon bei der Instanziierung einer Klasse als Startkonfiguration übergeben werden können, auch über mehrere Ebenen hinweg, wie an Store und Model oder Store und Proxy zu sehen, wobei im ersten Fall auf eine Klasse verwiesen wird und im zweiten Fall zusätzlich eine Parametrisierung mit einer Startkonfiguration vorgenommen wird. Dies funktioniert sogar für in die View eingebettete Oberflächenkomponenten – in diesem einfach gehaltenen Beispiel nicht gezeigt.

Desweiteren wurde ein proprietärer Proxy zur Anbindung eines Horus Repository Server (Listing 7 i.V.m. Abb. 1) implementiert, der die in JSON serialisierten Geschäftsobjekte in XML-basierte SOAP-Anfragen einbettet und aus den XML-Antworten zurückgelieferte Geschäftsobjekte extrahiert.

Die Anzeige- und Editierkomponente (Abb. 2, rechts) wurde im Wesentlichen als eine erbende Erweiterung zu einer Komponente zur Bildanzeige realisiert. Wobei zusätzliche Funktionalitäten, wie z.B. das Auswählen von Aktivitäten, Drag & Drop, Drill-down per Pinch-to-Zoom usw., primär als View-Logik innerhalb der Spezialisierung der reinen Anzeigekomponente implementiert wurden, womit eine wiederverwendbare Modelleditorkomponente entstand.

Fazit

Es wurde gezeigt, dass selbst komplexe XML-Dokumente schwergewichtiger Desktopanwendungen mit Standardmitteln fit für leichtgewichtige mobile Applikationen gemacht werden können. Außerdem war zu sehen, dass Webtechnologien, trotz ihrer Stärken und Schwächen, eine Alternative für Applikationen auf mobilen Endgeräten darstellen und auch für komplexe Applikationen sinnvoll eingesetzt werden können.

Links

http://www.oracle.com/technetwork/database-features/xmldb

http://www.oracle.com/technetwork/articles/quinlan-xml-095823.html

http://www.sencha.com

Kontaktadresse:

Torsten Rieger

Horus software GmbH

Pforzheimer Str. 160

D- 76275 Ettlingen

Telefon:	+49 (0) 7243-2179 0
Fax:	+49 (0) 7243-2179 99
E-Mail	torsten.rieger@horus.biz
Internet:	www.horus.biz