Die eigene ADF-Anwendung auf Wolke 7 bringen

Markus Klenke, TEAM

Viele Unternehmen haben in der Vergangenheit mithilfe von visuellen und deklarativen Oracle Frameworks geschäftskritische Anwendungen gebaut, welche über viele Jahre weiterentwickelt wurden. Parallel haben sich Leitarchitekturen und Technologien allerdings ebenfalls weiterentwickelt oder betreffen Konzepte, die zu Beginn der Entwicklung der Geschäftsanwendungen noch gar nicht im Sichtfeld der Entwicklung lagen. Insbesondere die Anwendungsentwicklung und -Bereitstellung in der Cloud nimmt heutzutage immer höhere Stellenwerte ein. Und obwohl das Oracle Application Development Framework zu Zeiten weit vor dem Cloudhype konzipiert wurde, stellen viele Komponenten eine sehr gute Grundlage dar, um die eigene Anwendung „Cloud-ready" zu machen.

Die grundlegende Idee des Application Development Frameworks (ADF) von Oracle ist, dass mithilfe eines Frameworks der gesamte Stack einer Webanwendung mit Datenzugriff im MVC Pattern via Java realisiert werden kann, ohne dass sich Entwickler mit einer Vielzahl von einzelnen Tools, Libraries oder verschiedenen Sprachen auseinandersetzen müssen. Diese Anwendung wird dann als Fat-WAR / EAR auf einem Java EE-Server bereitgestellt. Innerhalb der Anwendung wird allerdings schon zur Designzeit vollständig auf abstrakte Interfaces und nur schwache Klassenreferenzen gesetzt (vollqualifizierte Namen), was zwar den Einstieg in die Entwicklung etwas verkompliziert, dem erfahrenen Entwickler allerdings ein enormes Flexibilitätskonzept bietet.

Wie in *Abbildung 1* zu sehen, bietet ADF eine Implementierung für jede der Schichten in der MVC-Architektur, es ist allerdings auch möglich, mit ADF eine Anwendung zu bauen, welche nur in den datenverarbeitenden Schichten ADF- Komponenten verwendet und die visuellen Schichten durch alternative Implementierungen bereitstellt. Genau diese Flexibilität der Schichten innerhalb der Applikation erlaubt es uns, die Anwendungen auf unterschiedliche Arten und Weisen effizient in die Cloud zu bringen. Dieser Artikel zeigt mehrere Lösungswege auf, um unterschiedlichen Anforderungen an den Übergang in die Cloud gerecht zu werden.

Oracle Java Cloud-Service – Die „native" Lösung

Oracle bietet mit dem Java Cloud-Service eine Möglichkeit, einen vollwertigen Weblogic-Server in der Cloud zu betreiben – also den kanonischen Weg, die ADF-Anwendung als Java EE-Applikation in die Cloud zu bringen. Während der Aufwand eines Deployments in die Cloud über diesen Service nahezu keine Komplexität mit sich bringt, bietet dieser Ansatz allerdings auch nur wenige native Vorteile, wie die Cloud sinnvoll eingesetzt werden kann, da sowohl der Weblogic-Server als auch die Full-Stack-Anwendung allein durch ihre Größe bedingt meist unpraktisch sind (Beispiel: Startgeschwindigkeit eines Docker-Containers mit Oracle Fusion Middleware). Hier obliegen Themen wie Ausfallsicherheit und Elastizität eher dem Cluster-Konzept des Weblogic-Servers (beziehungsweise dem Oracle JCS) als der Topologie selbst.

Dennoch ist dieser Ansatz ein sehr valider, insbesondere wenn die Cloud-Strategie ist, die selfhosted Server abzuschaffen. Die Administratoren der Weblogic-Domäne finden sich in ihrem üblichen Metier sofort zurecht und müssen nicht in neue Strukturen angelernt, für den Build/Deployment-Prozess müssen nur Hostnamen angepasst werden; die Anwendung ist also grundsätzlich Cloud-fähig. Um einen wirklichen Mehrwert der Cloud zu erhalten, bietet es sich allerdings an, dennoch Restrukturierungen der Anwendung durchzuführen. Ein möglicher Ansatz der Umformung der Applikation kann durch das Konzept der Remote Regions *[2]* erreicht werden. Hiermit kann beispielsweise eine große monolithische Anwendung in viele kleine für sich selbst lauffähige Anwendungsteile zerlegt werden, ohne die Wiederverwendung durch bekannte ADF-Strukturen zu verlieren *(siehe Abbildung 2)*. Regions werden nicht mehr durch zur Design-Zeit festgelegte interne Library- Referenzen gefüllt, sondern über ein spezielles Java-Servlet, welches die Inhalte der Region und dessen Sicherheitskonzept über ein Servlet einer anderen ADF- Anwendung lädt. Das ermöglicht deutlich einfachere Ansätze für Ausfallsicherheit, Skalierbarkeit und Austauschbarkeit.

Zur Aktivierung der Remote-Region-Funktionalität müssen die jeweils einzelnen Projekte nur als Remote Region Producer resp. Consumer definiert werden, wodurch die entsprechenden Servlets in die web.xml eingetragen werden *(siehe Listing 1)*.

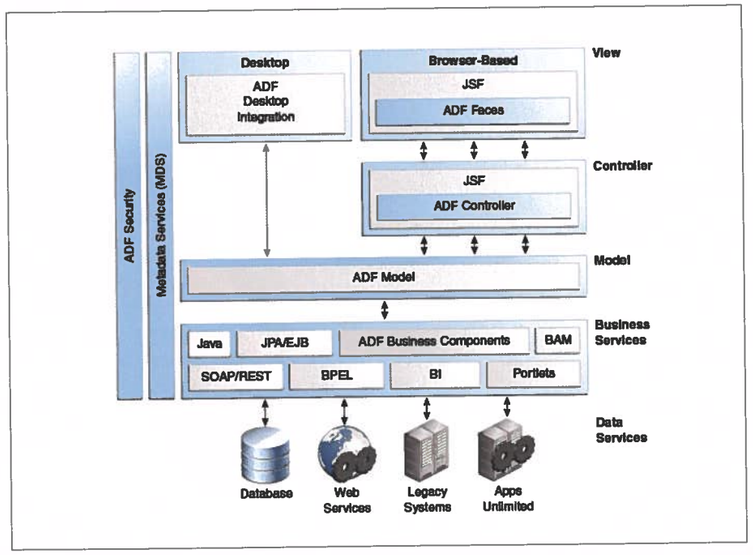

Abbildung 1: ADF Architektur (Quelle © Oracle, [1])

Zusätzlich müssen noch einige Parameter am Task-Flow selbst eingestellt werden, beispielsweise das Transaktionsmanagement.

Das Feature Remote-Regions benötigt allerdings eine Single Sign On-Lösung für die Applikationen und ist zusätzlich in der kostenfreien Variante von ADF (ADF Essentials) nicht enthalten ist. Daher kann dieses Feature nur auf einem Oracle Weblogic-Server verwendet werden, was wiederum nicht notwendigerweise gewollt sein muss, da man sich so an eine fixe Instanz eines Java EE-Servers bindet.

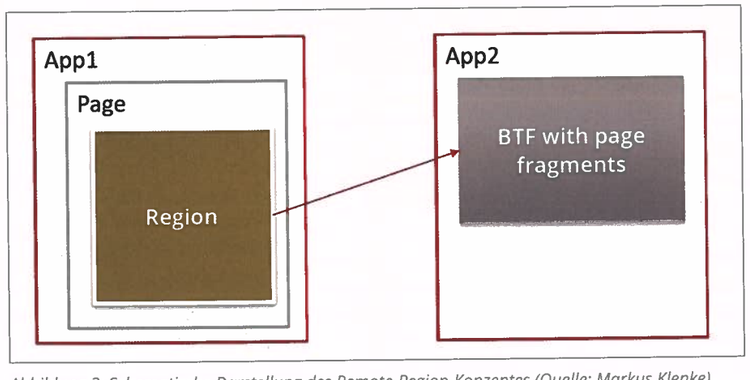

Abbildung 2: Schematische Darstellung des Remote-Region-Konzeptes (Quelle: Markus Klenke)

Separation der Schichten – ADF BC Rest Services

Wie in *Abbildung 1* gezeigt, sollte jede ADF-Webanwendung nach dem MVC Pattern entwickelt worden sein. Es besteht durch die Trennung dieser Schichten und dem ADF Model Layer als Abstraktionsschicht der Daten ohnehin schon nur eine schwache Abhängigkeit zwischen der Daten-/Businessschicht und der „Frontend"- Schicht . Nimmt man nun noch die Tatsache hinzu, dass nach Best-Practice der Großteil der fachlichen Komplexität und Business-Logik in der Business Service-Schicht implementiert sein sollte, so befindet sich der Kern der meisten ADF- Anwendungen in den Business Components. Für diese besitzt

```
<servlet>
    <servlet-name>rtfqueryServlet</servlet-name>
    <servlet-class>oracle.adfinternal.controller.rtfquery.RemoteTaskFlowQueryServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/rr/*</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>rtfqueryServlet</servlet-name>
    <url-pattern>/rtfquery</url-pattern>
  </servlet-mapping>
```

Listing 1: Servlet Properties für den Remote Region Producer

Oracle eine deklarative Möglichkeit, diese Services als REST- (oder SOAP) Schnittstelle nach außen verfügbar zu machen, wodurch nicht nur Datensätze, sondern auch Metadaten wie mehrsprachige Labels, Auswahllisten, Validatoren und selbst fachliche Funktionen propagiert werden können. Wird diese Schnittstelle genutzt, so können beliebige Frontend Frameworks (sofern sie WebService-fähig sind) mit diesen angereicherten Informationen beladen werden.

In *Abbildung 3* ist schematisch zu sehen, dass statt der JSF-Architektur für das Frontend ebenso eine JavaScript-basierte Architektur (hier Oracle JET) gewählt werden kann, um anwenderspezifische Frontend-Anforderungen an die Geschäftsprozesse abzubilden. Weiterhin kann nach diesem Muster die Anwendung selbst nach und nach von einem Monolithen in separate Anwendungsbereiche (oder auch nur Funktionen) aufgespalten werden.

Ein valider Ansatz ist hier, die bestehende Anwendung so umzubauen, dass jedes Application Module dupliziert und entsprechend eines der beiden auf die Nutzung für REST Webservices restrukturiert wird. Insbesondere folgende Punkte sind dabei zu beachten:

- Alle Operationen auf Application Module-Ebene (Client Interface des Application Modules) sollten so umgebaut werden, dass sie innerhalb des Client Interfaces eines View-Objekts definiert sind, damit sie über eine REST-Ressource zugreifbar gemacht werden können.
- Methoden auf View-Objekten im Client Interface sind per Definition immer über das REST-Interface erreichbar. Sollen Methoden nicht via REST verfügbar gemacht werden, so muss dies explizit gesetzt werden *(siehe Listing 2)*.
- Zur Verwendung von ViewAccessors sollten diese ebenfalls im DataModel des Application Modules sowie als ReadOnly Ressource zur Verfügung gestellt werden. So können deklarativ individuelle Voreinstellungen bezüglich der Datenakquise erstellt werden.

Ist die Migration des Modules abgeschlossen, kann die Umstrukturierung der Anwendung beginnen. Nach und nach werden nun View Controller-Projekte / Page Definitions so umgestellt, dass diese nicht mehr wie üblich auf den ADF BC Data Con-

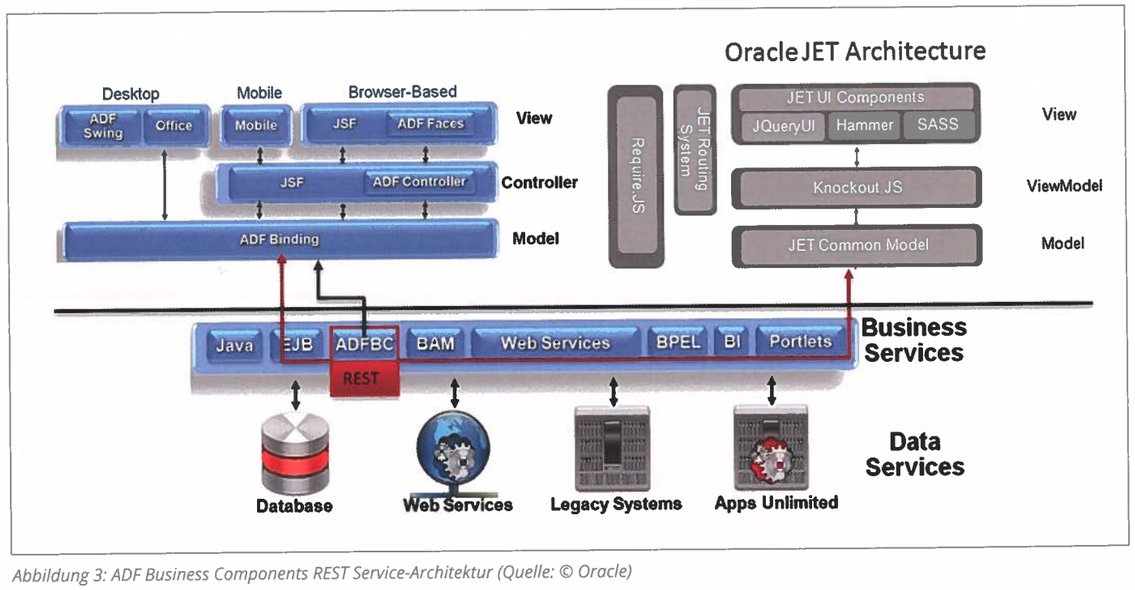

Abbildung 3: ADF Business Components REST Service-Architektur (Quelle: © Oracle)

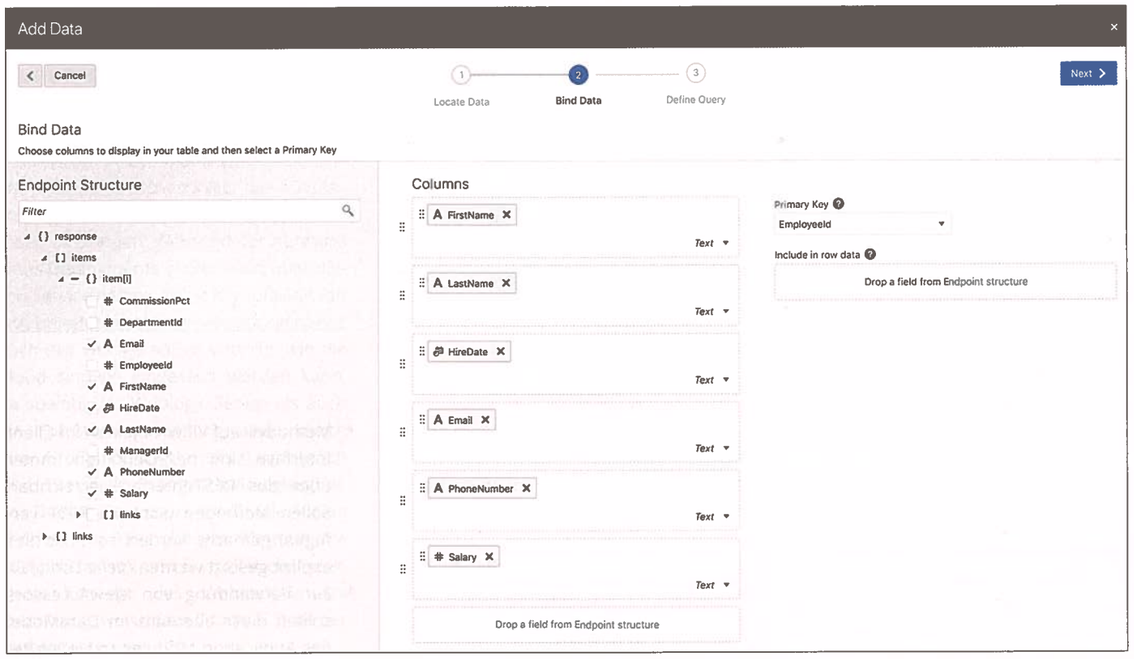

Abbildung 4: Visual Builder Data Wizard auf ADF BC Rest (Quelle: © Oracle)

trol zugreifen, sondern via ADF REST Data Control auf die neuen Module, welche separat auf dem Server als REST Web Application deployt werden. Auf Grund der Abstraktionsschichten und der angereicherten REST Responses ist dieser Schritt sowohl für den Endanwender kaum ersichtlich (Insbesondere, wenn Fetch-Sizes entsprechend angepasst werden) als auch der Implementierungsaufwand für die Entwickler minimal. Wichtig ist hier natürlich auch wieder, dass bei der Implementierung der Java Beans darauf geachtet wird, dass nicht an den Interfaces vorbeiprogrammiert wurde (Beispiel: Direktzugriff auf oracle.jbo extended-Klassen im Controller), da sonst die Komplexität einer Umstellung um ein Vielfaches erhöht wird.

Oracle VBCS – ADF-like JavaScript Development

Eine der größten Herausforderungen für die Frontendentwicklung mit ADF ist die serverbasierte Architektur von JSF. Diese bietet durch den komponentenbasierten Fokus zwar enorm viele Vorteile in Sachen Uniformisierung, Abstraktion und Komplexitätsreduktion, hat durch die Querkommunikation zwischen Client und Server aber durchaus Schwierigkeiten in puncto Geschwindigkeit und User Experience, wenn es um Vergleiche mit Javascript Frameworks geht. Der Fokus auf Client-Side Web Applications und Progressive Web Apps erfordert von den Entwicklern fast schon eine Umstellung auf JavaScript. Die Einarbeitung in eine neue, sehr technische und Code-affine Programmiersprache stellt allerdings für viele Entwickler mit 4GL-Hintergrund eine große Herausforderung dar, welche im Tagesgeschäft häufig nicht gemeistert werden kann.

```
<Method Name="hello">
      <Return
        Type="java.lang.String"/>
      <Properties>
        <CustomProperties>
          <Property
            Name="PAYLOADHINT"
            Value="Hide"/>
        </CustomProperties>
      </Properties>
    </Method>
```

Listing 2: Explizites Deaktivieren einer Methode des Client Interfaces für REST

Glücklicherweise bietet Oracle mit dem Oracle Visual Builder Cloud-Service eine Möglichkeit, den Übergang von Java zur JavaScript-Entwicklung für ADF-Entwickler so einfach wie möglich zu machen. VBCS ist eine deklarative Entwicklungsumgebung, in der Oracle JET-Anwendungen nicht nur via WYSYWIG- Editor visuell gebaut werden können, sondern insbesondere der JavaScript Lifecycle deklarativ erstellt werden kann (ähnlich wie Task-Flows in ADF für Java). Somit wird es für viele Entwickler deutlich leichter, sich mit Themen wie Callbacks, dynamischen Seitenwechseln und Fehlerbehandlung auseinanderzusetzen. Da es sich beim VBCS grundsätzlich aber auch um einen Service zur Entwicklung von Frontend handelt, benötigt man noch einiges an Arbeit, um die Datenaufbereitung durchzuführen - es sei denn, der datengebende Service ist ein ADF REST-Service, dann werden vom VBCS zusätzlich die Metadaten des REST-Service interpretiert. Insbesondere Labels und Auswahllisten können so direkt verwendet werden, ohne diese mit großem Aufwand neu definieren zu müssen. (Wizard gesteuert, *siehe Abbildung 4*)

Sind die ADF REST-Services entsprechend gut vorbereitet, können einfache Oberflächen oder spezielle Services so sehr einfach als JavaScript-Anwendung bereitgestellt werden. Über diesen Weg können Teile der ADF-Anwendung beispielsweise sehr effizient ohne großen Aufwand als PWA für mobile Endgeräte entwickelt werden oder Teile der Anwendung je nach Kontext sehr effizient skaliert werden, um beispielsweise die Performance der Anwendung zu verbessern.

Die Zukunft von ADF ist heiter in den Wolken

Die beiden beschriebenen Ansätze können fast auf jede ADF-Anwendung, welche nach bekannten ADF-Architekturmustern umgesetzt ist, angewandt werden. Ob nun eine Anwendung nach Pillar-Architektur oder eine monolithische Anwendung; je nach Anwendungsfall ergeben sich unterschiedlichste Begründungen, überhaupt in die Cloud zu gehen. Den größten Vorteil erreicht man allerdings, wenn die Applikation nicht nur in die Cloud migriert, sondern diese mit sinnvollen Anpassungen so umstrukturiert wird, dass sie eine möglichst flexible Anbindung an andere Systeme erlaubt. Der Kern der Anwendung mit seiner Geschäftslogik bleibt weitestgehend unverändert, Schnittstellen für Benutzerinteraktion oder Einbindung in Service- Prozesse werden allerdings deutlich vereinfacht.

Somit bietet ADF (eventuell mit Unterstützung durch Oracle VBCS) auch 2020 noch effizienteste deklarative Datenbankanwendungsentwicklung mit modernsten Techniken, und als Entwickler bleibt man somit für die Zukunft auch ohne große Umschulungen bestens für Web-Anwendungsentwicklung gerüstet.

Quellen

[1] Oracle ADF Documentation Library : *https://docs.oracle.com/en/middleware/developer-tools/adf/12.2.1.4/concepts/overview-oracle-adf.html#GUID-D3AFB48B-94D7-471D-A3A9-B7DC9E09C2E1*

[2] Oracle ADF Remote Regions : *https://docs.oracle.com/middleware/1221/adf/develop/GUID-D7FDF023-1373-47CB-B349-DC24826441B0.htm#ADFFD54447*

Über den Autor

Markus Klenke unterstützt als Oracle Consultant unterschiedliche Kunden bei der Entwicklung von Business-Software und der Erstellung und Analyse von Softwarearchitektur. Dabei liegt der Fokus auf modellgetriebenen Architekturen und der Entwicklung sowie der Oracle Fusion Middleware, wobei insbesondere die deklarativen Frameworks Oracle ADF und Oracle VBCS hier die zentralen Interessensgebiete darstellen.

Markus Klenke
mke@team-pb.de