Die Siegessäulen – Der Weg zur erfolgreichen ADF-Architektur

Timo Veeders
IKB Deutsche Industriebank AG
Düsseldorf

Schlüsselworte
ADF, Architektur, Design Patterns, Projektbericht, Forms, Migration

Einleitung

Im Zuge einer Plattform-Modernisierung von Forms zu ADF ist es notwendig, eine Ziel-Architektur zu wählen und zu implementieren. Dieser Erfahrungsbericht zeigt neben einer möglichen Strukturierung innerhalb einer Pillar-Architektur auch Stolpersteine und Hürden auf, die Entwickler und Architekten begegnen. Praxisnahe Lösungsvorschläge helfen bei der Implementierung.

Von Forms über eine Sum-of-the-Parts-Architektur zum Pillar

Die Kreditplattform, welche auf ADF umgestellt wird zählt etwa 500 Forms-Module und 300 Reports. Ohne Erfahrung und Kenntnis von Architekturen wurde nach der Sum-of-the-Parts-Architektur begonnen, einzelne Module der alten Applikation nach ADF zu migrieren. Charakteristisch für eine Sum-of-the-Parts-Architektur ist, dass ein EAR erzeugt und auf den Server deployed wird. Dazu wird eine Master-Applikation mit einem Unbounded-Taskflow gebaut, die weitere Applikationen als ADF Library (oder Shared Library) einbindet und aufruft. Diese Architektur bietet eine hohe Wiederverwendbarkeit durch „Loose-Coupling“. Jeder Bounded-Taskflow kann eine für sich abgeschlossene Einheit bilden. Nachteile dieser Architektur werden, je größer die Applikation wird, immer deutlicher; Die Architektur, sowie das Build- und Dependency-Management zunehmend komplexer. Das EAR-File wird, sofern ADF-Librarys genutzt werden, groß und die Ressourcenanforderungen auf lange Sicht zu hoch.

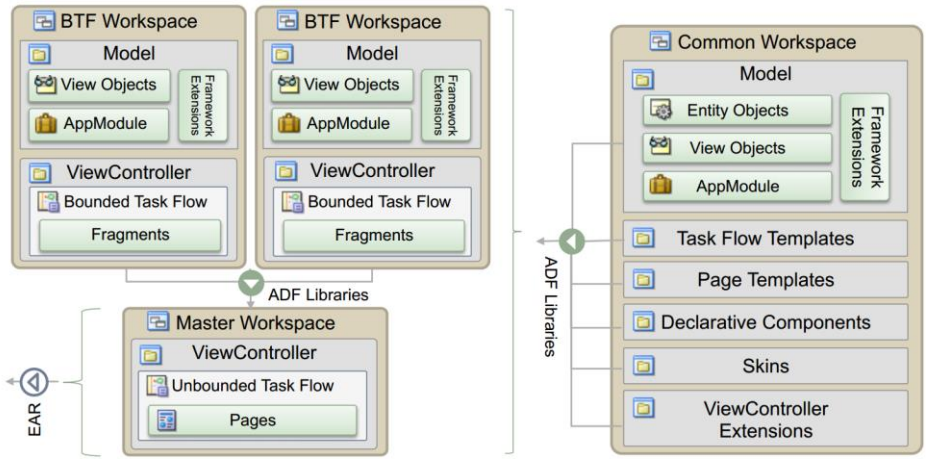


Abb. 1: Sum-of-the-Parts Architecture[1]

[1] Quelle: ORACLE, 2013: Real World ADF Design & Architecture Principles – ADF Architectural Patterns S. 30

Eine alternative Architektur ist die Pillar-Architektur. Bei dieser ist jede Applikation ein eigenes EAR-File. Im Fall der IKB bleibt die „Master"-Applikation mit einem Unbounded-Taskflow bestehen. Die einzelnen Pillar-Applikationen bieten einen Bounded-Taskflow als Einstieg an.

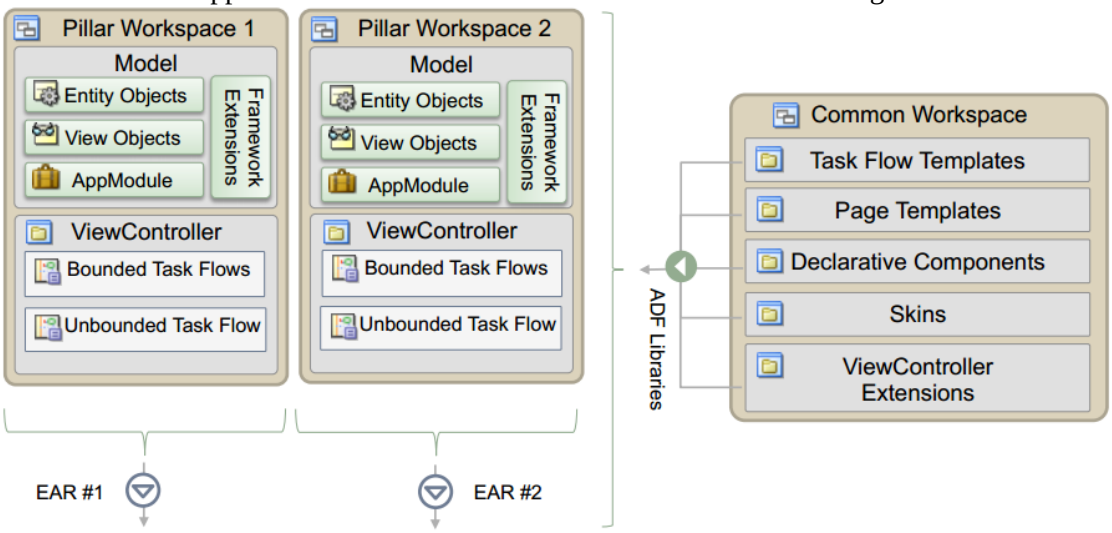


Abb. 2: Pillar Architecture[2]

In der Pillar-Architektur werden mehrere EAR-Files gebaut, wodurch es möglich ist, diese auf unterschiedliche Weblogic-Server zu verteilen. Im Hinblick auf Performance und Ressourcen ist dies von Vorteil. Des Weiteren ist ein Flexibleres Deployment möglich; Einzelne Applikationen können gebaut und deployed werden, ohne das die gesamte Anwendung nicht zur Verfügung steht.

Nachteile einer Pillar-Architektur sind unter anderem, dass die Wiederverwendbarkeit eingeschränkt ist. Funktionen, die über mehrere Pillar hinweg genutzt werden, müssen in einen Common-Workspace ausgelagert werden. Das State-Sharing zwischen einzelnen Applikationen ist komplex und mitunter nicht ohne weitere Software, wie beispielsweise Confluence oder einem Datenbank-Framework, möglich.

Dimensionierung einer Pillar-Applikation

Schon vor der Implementierung einer oder mehrerer Pillar-Applikationen ist es von Vorteil, wenn der Funktionsumfang der Applikation grob spezifiziert wird. Eine Dimensionierung kann anhand vorhandener Datenmodelle und/oder Namensräumen in der Datenbank erfolgen. Ist dies nicht möglich, kann man mit dem Kunden oder den Fachbereichen Funktionsgruppen und Prozesse herausarbeiten. Diese Vorarbeit erspart aufwändiges Refactoring und sorgt für Ordnung in Package-Strukturen.

Deckt die Pillar-Applikation einen großen Funktionsbereich ab, ist die Wiederverwendung von Taskflows einfacher und die Kommunikation zwischen Applikationen wird reduziert. Dabei besteht die Gefahr, dass ein monolithischer Pillar entsteht, dessen Ressourcenanforderungen sehr hoch sind. Das Build-Management wird kompliziert und zyklische Abhängigkeiten lassen sich nur schwer vermeiden. Im Gegensatz dazu stehen minimalistische Pillar mit einem sehr geringen Funktionsumfang. Die Funktionsgruppen können auf mehrere Server verteilt werden und das

[2] Quelle: ORACLE, 2013: Real World ADF Design & Architecture Principles – ADF Architectural Patterns S. 46

Deployment einzelner Funktionsgruppen ist möglich. Jedoch muss der Anwender oft zwischen Pillar-Applikationen wechseln, wodurch der Kommunikationsaufwand mit anderen Pillar-Applikationen stark anwächst. Zudem sind sehr viele Sessions auf den Servern aktiv und die Wiederverwendbarkeit der Funktionen sehr beschränkt. Die Dimensionierung eines Pillar sollte also im Vorhinein evaluiert werden.

Single-Sign-On und Kommunikation zwischen Pillar-Applikationen

Der Anwender hat den Anspruch sich nur einmal authentifizieren zu müssen, egal in welcher Architektur er sich bewegt. Daher sollte ein Single-Sign-On Konzept implementiert werden.

Einmal authentifiziert, kann der Anwender sich zwischen den Applikationen bewegen. Um die Kommunikation sicherzustellen, können Remote Taskflow Calls verwendet werden. Diese erzeugen automatisch eine URL mit definierten Parametern. In der Praxis ist es oft nicht möglich, bereits zur Design-Time die Remote Application URL zu bestimmen. Hinterlegen Sie daher besser eine EL-Expression, die die URL über eine Bean ermittelt, abhängig davon, auf welchem Server Sie sich befinden und wo der Pillar deployed ist. Der Taskflow der per Remote Taskflow Call aufgerufen werden soll, sollte die Property URL Invoke auf <url-invoke-allowed/> gesetzt haben. Definieren Sie die Input- und Output-Parameter außerdem ausschließlich als java.lang.String, da die URL-Parameter nicht Typsicher übergeben werden können. Ein entsprechendes Error-Handling in den set-Methoden sorgt dann für die nötige Sicherheit.

Session-Handling

Mit steigender Pillar-Anzahl steigen auch die offenen Sessions auf dem Server und der Datenbank. Die Ressourcen auf Server und Datenbank sind begrenzt. Daher ist es in einer Pillar-Architektur noch wichtiger hier frühzeitig einzugreifen und Gedanken an Session-Handling und Session-Timeout zu verschwenden. Navigiert ein Anwender zu einer anderen Pillar-Applikation wird eine neue Session geöffnet und die Session der anderen Pillar-Applikation soll ihre Ressourcen so schnell wie möglich wieder freigeben und beendet werden. Für viele Applikationen reicht es, den Session-Timeout in der web.xml auf einen geringen Minutenwert zu setzen (+-10 Minuten). In Business-Anwendungen reicht dies oft nicht aus. Nach längeren Telefonaten oder Pausen sind eingegebene Daten verloren. Eine Lösungsvariante ist eine JavaScript-Funktion, die beim Laden der jspx-Seite ausgeführt wird und kurz vor Ablauf der Session einen Request auf ein Servlet absetzt. Im Beispielcode wird zusätzlich JQuery[3] benötigt.

Binden Sie die JQuery-Bibliothek in ein jspx-Template ein:

```
<af:resource type="javascript" source="RELATIVER_PFAD "/>
```

Folgendes JavaScript Snippet liest den SessionTimeout aus und sendet einen Request 3 Minuten vor Ablauf der Session:

```
<af:resource type="javascript">
    $(document).ready(function() {
        setInterval(function() {
$.get("${pageContext.request.contextPath}/poll");
        }, ${(pageContext.session.maxInactiveInterval - 180) * 1000});
  });
</af:resource>
```

[3] http://jquery.com/

Das Poll-Servlet ruft die Session auf und sendet eine leere Response:

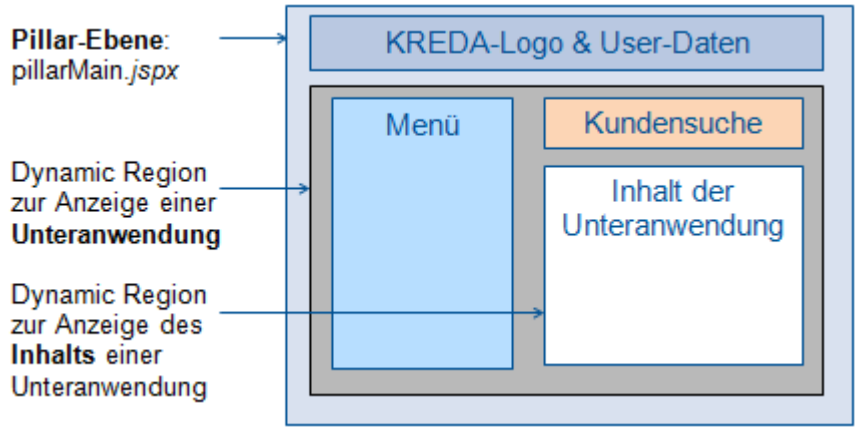
```
public class PollServlet extends HttpServlet {
    public PollServlet() {
        super();
    }
    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response) throws
ServletException,
IOException {
        request.getSession();
        response.setStatus(response.SC_NO_CONTENT);
    }
}
```

Registrieren Sie das Servlet in der web.xml des Pillar und unterdrücken Sie das Expiration-Warning Popup:

```
<servlet>
    <servlet-name>poll</servlet-name>
    <servlet-class>de.ikb.adf.basis.ui.umgebung.PollServlet</servlet-class>
</servlet>
…
<context-param>
    <description>Unterdrueckt das Expiration Warning Popup</description>
    <param-
name>oracle.adf.view.rich.sessionHandling.WARNING_BEFORE_TIMEOUT</param-
name>
    <param-value>0</param-value>
  </context-param>
```

User Interface

Über Page-Templates kann ein einheitliches UI-Konzept realisiert werden. Sie bieten die Möglichkeit Standards, wie beispielsweise das JavaScript zum Session-Polling (siehe vorheriger Abschnitt), zu implementieren.



Überlegen Sie sich, welche verschiedenen Grundtypen an jspx und jsff-Seiten Sie in Ihrer Anwendung verwenden und implementieren Sie jeweils ein Template. Für das Navigationskonzept wurde folgender Schematischer Aufbau für die Rahmenseite eines Pillar gewählt und implementiert:

Abb. 3: Schematischer Aufbau eines einheitlichen User Interface

Deep-Linking zwischen Pillar-Applikationen

Die verschiedenen Pillar-Applikationen sowie die Menüeinträge werden in der Datenbank konfiguriert und gespeichert. Durch die Konfiguration in der Datenbank können auch Metadaten zu den jeweiligen Pillar-Applikationen gepflegt und von anderen ausgelesen werden.

Ein weiterer Vorteil ist, dass die Anwender Menüpunkte verschiedener Pillar-Applikationen als Favorit hinterlegen können. Durch einen URL-Aufruf und einem Deep-Linking-Konzept wird dann der jeweilige Pillar mit dem entsprechendem Taskflow gestartet und nötige, definierte Kontext-Parameter übergeben. Bestandteil der URL ist unter anderem eine Menü-ID in Form eines Kürzels, sodass die Dynamic-Region der Unteranwendung und die des Inhalts korrekte Taskflows darstellen.

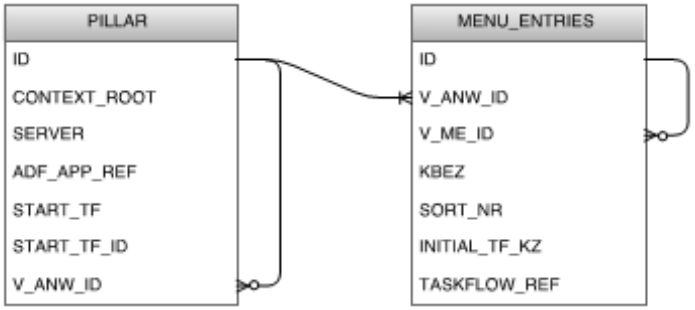


Abb. 4 Datenmodell eines Deep-Linking-Konzeptes

```
http://[SERVER]:8001/[PILLAR_CONTEXT_ROOT]/faces/adf.task-
flow?adf.tfDoc=[PILLAR_START_TF]&adf.tfId=[PILLAR_START_TF_ID]&menuKuerzel=
[MENU_KBEZ]
```

Dynamische Bestandteile der URL können aus der Datenbank ermittelt werden; Server, Context Root, Start TF und Start-TF-Id werden an der Applikation hinterlegt; Menu_Kbez (Unique Key) verweist auf einen Taskflow in der Tabelle MENU_ENTRIES. Durch die Übergabe eines Unique Keys in der URL kann dann per DynamicRegion direkt auf einen Menüeintrag navigiert werden. Weitere URL-Parameter wie beispielsweise eine Kundennummer können dazu genutzt werden, einen Kontext zwischen den Pillarn zu übergeben.

Fazit

Die Herausforderungen einer Pillar-Architektur sind keine besonderen und treten überall dort auf, wo eine Gesamt-Applikation auf mehrere EARs und Server verteilt werden kann. Über ein zuverlässiges Single-Sign-On Konzept und Deep-Linking kann zwischen den EARs navigiert werden, ohne dass der Anwender sich mehrfach authentifizieren muss. Das Session-Handling kann über Session-Timeout und entsprechendes JavaScript ressourcenschonend implementiert werden und Templates bieten eine effiziente Möglichkeit, UI-Patterns und Styleguides durchzusetzen und zu implementieren.

Kontaktadresse:
Timo Veeders
IKB Deutsche Industriebank AG
Wilhelm-Bötzkes-Straße 1
D-40474 Düsseldorf

Telefon: +49 (0) 211-8221 4338
E-Mail: timo.veeders@ikb.de
Internet: www.ikb.de