Oracle Enterprise Scheduler (ESS) Unleashed

Carsten Wiesbaum
esentri AG
Ettlingen

Schlüsselworte
Automatisierung, Betrieb, Middleware

Einleitung

Der Oracle Fusion Middleware Stack beinhaltet eine leistungsstarke und vollkommen integrierte Enterprise Scheduler Lösung, Oracle Enterprise Scheduler (ESS). Sie ermöglicht es, auf einfache Weise, eine Vielzahl an unterschiedlichen Job Typen (Java, PL/SQL, Webservices, uvm.) zu definieren und zu vorher bestimmten Zeiten auszuführen. Eine oft vergessene Nutzung von ESS ist jedoch die Integration der Funktionen in eigene Java Anwendungen. Diese ESS Custom Applications ermöglichen die Definition sowie Einplanung eigener Jobs innerhalb der ESS Infrastruktur. Innerhalb dieses Vortrags wird zunächst die grundlegende Architektur des Oracle ESS erläutert. Anschließend werden an einem Beispiel die benötigten minimalen Entwicklungsschritte für die Integration der ESS Funktionalitäten in die eigene Anwendung beschrieben.

Oracle Enterprise Scheduler (ESS)

Der Oracle Fusion Middleware Stack beinhaltet eine leistungsstarke und vollkommen integrierte Enterprise Scheduler Lösung, Oracle Enterprise Scheduler (ESS). Es ist ein bekanntes 11g Produkt der Oracle Fusion Applications und ermöglicht, auf einfache Weise, die Definition einer Vielzahl an unterschiedlichen Job Typen (Java, PL/SQL, Webservices, uvm.) und deren Einplanung zu vorher bestimmten Zeiten. Seit Version 12c ist das Produkt auch in der Oracle SOA Suite verfügbar.

Die grundlegende Architektur von ESS basiert auf der eigentlichen Oracle Enteprise Scheduler Komponente, sogenannten Client Applikationen und deren Metadaten (siehe Abbildung 1). Eine Client Applikation beinhaltet dabei den ausführbaren Code sowie beschreibende Metadaten. Über eine geeignete Oberfläche und die Nutzung von entsprechenden APIs kann die Ausführung des Codes an die ESS Komponente delegiert werden. Die Scheduler Komponente beinhaltet ein Runtime Modul, einen Request Dispatcher und einen Request Processor. Client Applikationen kommunizieren direkt mit dem Runtime Modul, welches die im MDS abgelegten Metadaten der Client Applikation lädt und deren Jobs einer Wait Queue hinzufügt. Diese Wait Queue wird vom Request Dispatcher überwacht. Er verschiebt die Jobs zum richtigen Zeitpunkt in die Ready Queue. Die Jobs aus der Ready Queue werden nacheinander von dem Request Processor aufgegriffen und die Ausführung des entsprechenden Codes in der Client Applikation über eine Endpoint MDB angestoßen.

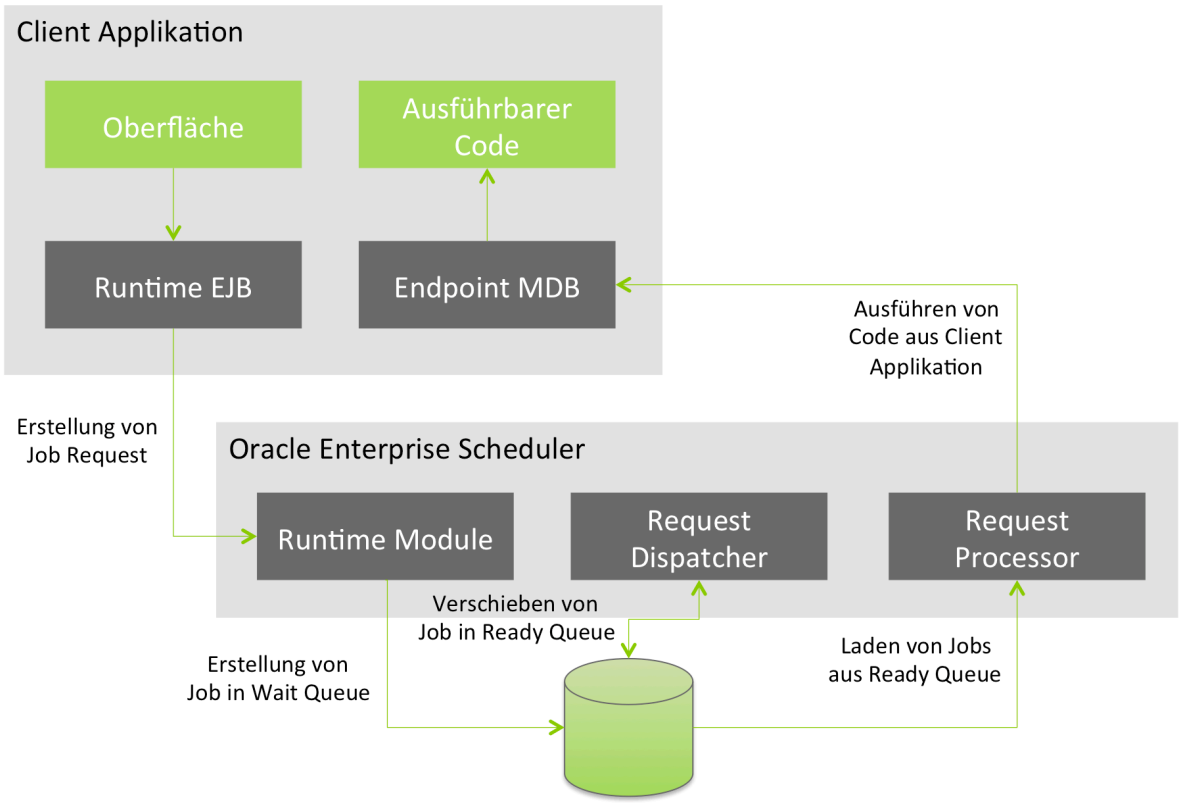

Abb. 1: Oracle Enterprise Scheduler Architektur

Um einen Job in ESS einzuplanen müssen zur Design Time zunächst entsprechende Metadaten definiert werden. Folgende Metadaten können definiert werden:

- Job Type
- Job Definition
- Job Set
- Schedule
- Incompatibility

Ein *Job Type* definiert was ausgeführt werden soll. Mögliche Job-Typen sind unter anderem Java, PL/SQL oder der Aufruf von Webservices. Je nach Job Type müssen unterschiedliche Metadaten konfiguriert werden, für einen Java Job zum Beispiel die auszuführende Java Klasse sowie deren Parameter. Nach der Definition eines Job Typen muss definiert werden wie er konkret aufgerufen wird. Dies geschieht über eine *Job Definition*. In der Job Definition wird der entsprechende Job Type ausgewählt, die konkrete Belegung der Parameter sowie weitere spezifische Eigenschaften konfiguriert. Mehrere Job Definitionen können nun noch über ein *Job Set* zusammengefasst und so als eine Einheit parallel oder sequentiell abgearbeitet werden. Des Weiteren können noch *Incompatibility* Metadaten definiert werden. Diese beschreiben welche Job Definitions und Job Sets auf keinen Fall gleichzeitig ausgeführt werden dürfen. Zuletzt werden natürlich noch Metadaten benötigt die beschreiben wann ein bestimmter Job ausgeführt werden soll. Dies wird über *Schedules* definiert. Sie beschreiben bestimmte Zeitpunkte oder auch wiederkehrende Intervalle. Um einen Job Request an ESS zu senden müssen mindestens Job Type, Job Definition und ein Schedule definiert sein.

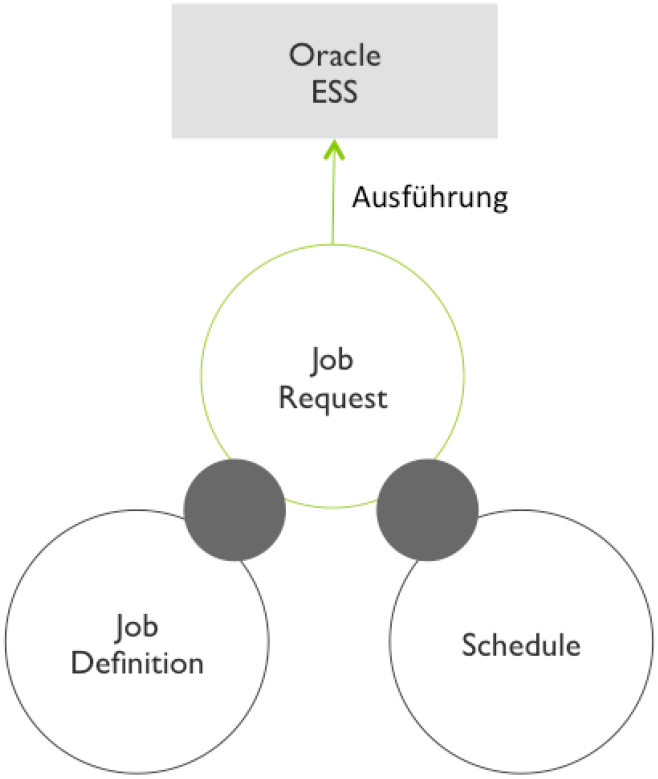

Abb. 2: ESS Job Request Definition

ESS Custom Applications

Von Haus aus liefert ESS eine Standardapplikation zum einplanen von Jobs mit. Diese Anwendung ist in den Oracle Enterprise Manager integriert und kann zum einplanen einfacher Jobs verwendet werden (siehe Abbildung 3). Es kann und gibt jedoch häufig die Anforderung eine Funktionalität zur Einplanung von Jobs direkt in die eigene Anwendung zu integrieren. Um diesen Anforderungen gerecht zu werden bietet ESS die Möglichkeit der *Custom Applications*. Hierbei werden einige Bibliotheken in die eigene Anwendung integriert und so die Funktionalitäten von ESS genutzt. Beim Design einer Custom Application kann sowohl eine *Single-* als auch ein *Split-Application* umgesetzt werden. Bei einer Single-Application befinden sich sowohl die Komponenten zur Einplanung von Jobs als auch die auszuführenden Code Fragmente in einer Anwendung. Beides kann jedoch auch in zwei oder mehr kleinere Anwendung aufgeteilt werden. So ist es möglich man ein Design verfolgen in dem es eine Planungsanwendung und viele kleinere Anwendung, die einzelne Jobs implementieren, gibt.

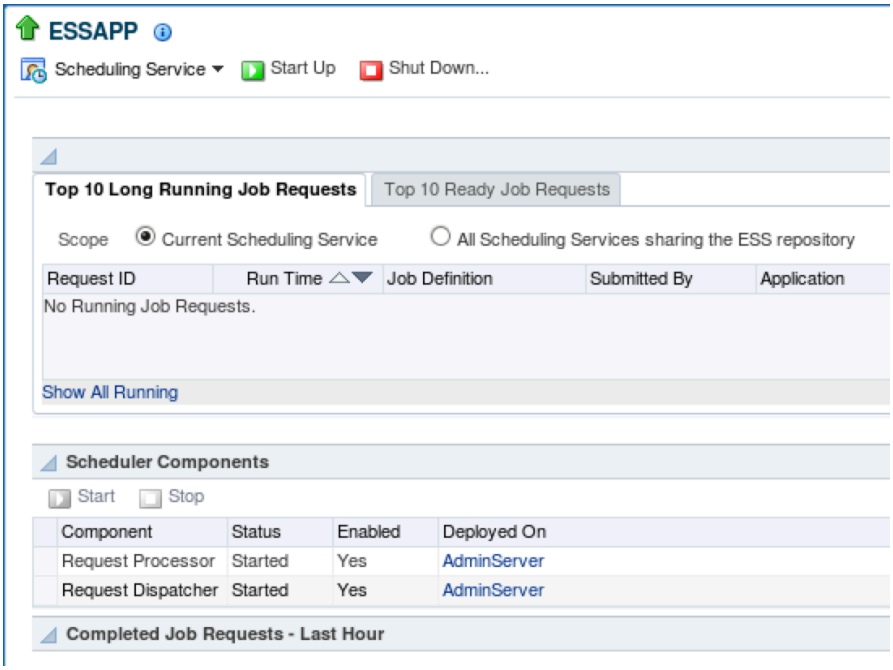

Abb. 3: ESS Anwendung im Oracle Enterprise Manager

Unabhängig von dem Anwendungsdesign müssen folgende Schritte zur Entwicklung einer ESS Custom Application durchgeführt werden:

- Konfiguration der Entwicklungsumgebung
- Erstellung der Java Anwendung(en) zum Abschicken (submitting) und Ausführen (hosting) von Jobs
- Definition der Oracle ESS Metadaten
- Umsetzung der Job-Logik in Java
- Verpacken und Auslieferung der Anwendung
- Ausführung der Anwendung

Entwicklung einer ESS Custom Application

Im folgenden Abschnitt wird die Entwicklung einer ESS Custom Application mit Oracle JDeveloper beschrieben. Um JDevelopers Funktionen zur Entwicklung einer solchen Applikation zu nutzen muss vor dem Start zunächst die *MW_HOME* Systemvariable definiert werden. Listing 1 zeigt die korrekte Konfiguration der Entwicklungsumgebung unter Windows. Im Beispiel wurde JDeveloper im Verzeichnis *c:\Oracle\Middleware\jdeveloper* installiert. Nach dem Start von JDeveloper sind die Funktionen zur Entwicklung einer ESS Custom Application nutzbar.

```
> cd c:\Oracle\Middleware\jdeveloper
> set MW_HOME=c:\Oracle\Middleware
> jdeveloper
```
Listing 1: Konfiguration der Entwicklungsumgebung

Die Funktionalität der Applikation in diesem Artikel wird in zwei Projekte aufgeteilt (siehe Abbildung 4). Im ersten Projekt wird eine einfache Oberfläche entwickelt, die zur Einplanung der Jobs verwendet wird (ESS Client Support). Hierzu muss bei der Erstellung des Projektes das Project Feature *ESS Client Support* zum Projekt hinzugefügt werden. Das zweite Projekt beinhaltet den eigentliche Code für den Java Job (ESS Host Support). Damit die Ausführung des Codes von ESS angestoßen werden kann müssen die Project Features *ESS Host Support* und *ESS Job Support* ausgewählt werden.

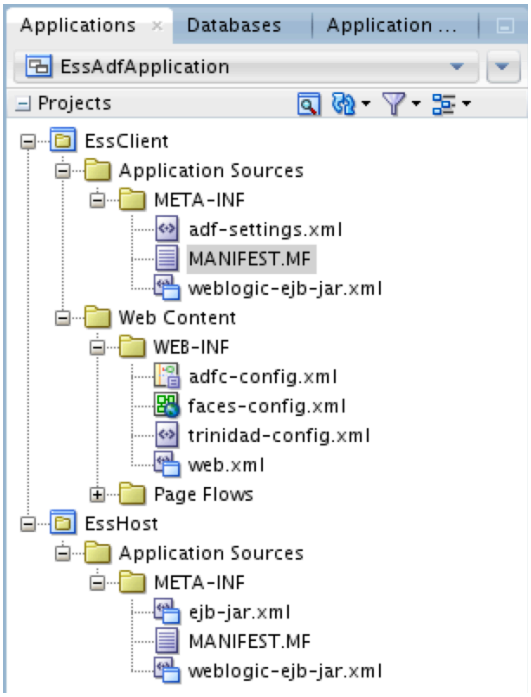

Abb. 4: ESS Custom Application in Oracle JDeveloper

Nach dem Aufsetzen der Applikationsstruktur kann eine neue Job Definition inklusive der dazugehörigen Java Implementation in der *EssHost* Applikation erstellt werden. Den entsprechenden Wizard öffnet man über die *New Gallery* unter *Business Tier -> Enterprise Scheduler Metadata -> Job Definition* (siehe Abbildung 5). Im Wizard Dialog müssen Informationen wie Name, Package und Job Type angegeben werden. Wählt man den *JavaJobType* aus kann man außerdem eine entsprechende Java Klasse vom Wizard erstellen lassen. Für diesen Artikel wird eine Klasse mit dem Namen *HelloAdfImpl* erstellt. Nach Bestätigung des Wizard Dialogs findet man in der EssHost Applikation die generierte Java Klasse und die dazugehörigen Job Definition Metadaten.

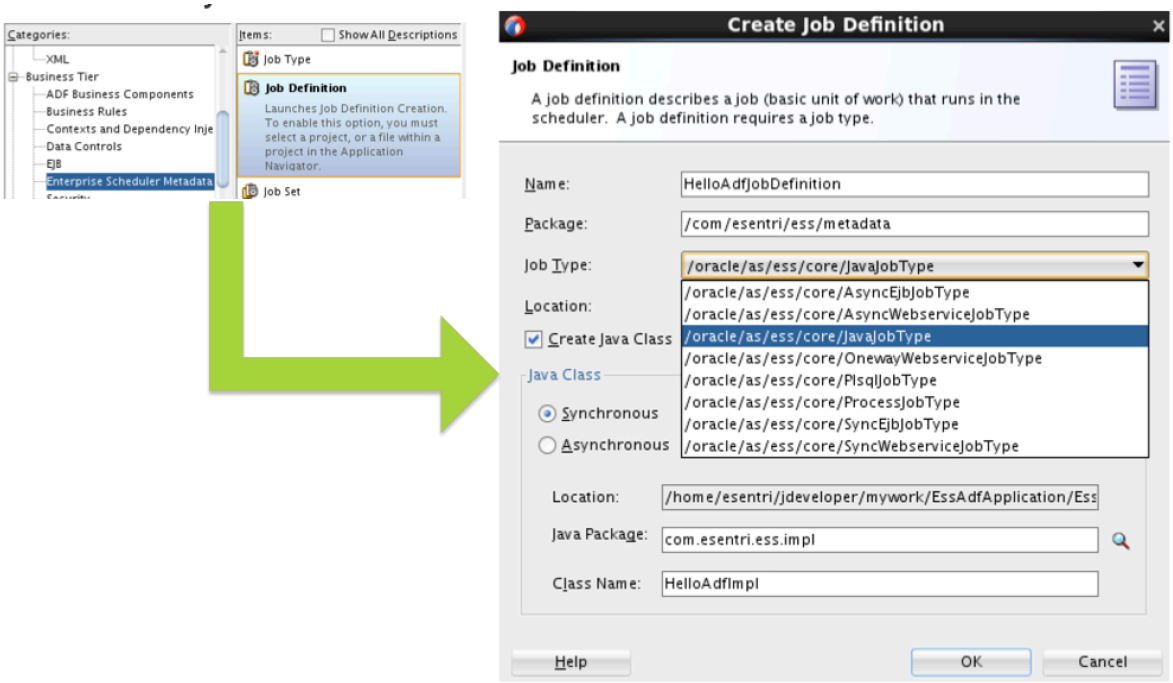

Abb. 5: Erstellung der Job Definition

Die generierte Klasse nutzt die beiden Interfaces *Executable* und *Cancellable*, wodurch die Methoden *execute()* und *cancel()* implementiert werden müssen. Die execute() Methode wird vom ESS Request Processor angestoßen wenn er einen entsprechenden Job Request in der Ready Queue findet. Die cancel() Methode ermöglicht es ESS einen eingeplanten Job Request abzubrechen. Im Listing 2 ist die Implementation der Klasse zu sehen. Der Job in diesem Beispiel schreibt lediglich eine Log-Nachricht über das Java Logging Framework. Damit ist die Umsetzung des EssHost Projektes bereits abgeschlossen.

```
public class HelloAdfImpl implements Executable, Cancellable
{
  public void execute(RequestExecutionContext ctx, RequestParameters
params)
    throws ExecutionErrorException, ExecutionWarningException,
           ExecutionCancelledException, ExecutionPausedException
  {
      _logger.info("----> Hello ADF! Request ID: " + ctx.getRequestId());
}

  public void cancel()
  {
  }
}
```

Listing 2: HelloAdfImpl Klasse

Für das EssClient Projekt soll eine einfache ADF Seite entwickelt werden. Auf dieser Seite wird der implementierte Java-Job über einen Button eingeplant. Zur Umsetzung dieser Funktionalität kann eine einfache *Managed Bean* verwendet werden. Diese muss zunächst Instanzen des ESS *RuntimeService* und *MetadataService* per Lookup beziehen. Anschließend können über den MetadataService die Job Definitionen der Custom Application durchsucht werden (siehe Listing 3).

```
Enumeration<MetadataObjectId> jobDefs = null;

try {
    MetadataServiceHandle handle = mds.open();
   jobDefs = mds.queryJobDefinitions(handle, null, QueryField.NAME, true);
} catch (MetadataServiceException e) {
   ...
}
```
Listing 3: Abfrage der Custom Application Metadaten

Das Ergebnis der MetadataService Abfrage ist eine Liste von passenden Metadaten. Nachdem die gewünschte Job Definition gefunden wurde, kann ein entsprechender Request für den Job über den RuntimeService abgeschickt werden. Hierbei können vorher definierte Schedules verwendet, oder im Code zur Laufzeit ein bestimmter Zeitpunkt zur Ausführung definiert werden (siehe Listing 4).

```
try {
   RuntimeServiceHandle handle = rts.open();

   LOGGER.info("Scheduling job: " + jobDef.getNamePart());

   Calendar cal = new GregorianCalendar();
   cal.add(Calendar.SECOND, 20);

   rts.submitRequest(handle, "AdfEssClient Job", jobDef, cal, new
RequestParameters());
} catch (RuntimeServiceException e) {
   ...
}
```
Listing 4: Einstellen eines Job Request per ESS RuntimeService

Die gezeigten Code Fragmente werden in der Managed Bean in einer Methode implementiert. Diese kann nun als *ActionListener* des Buttons genutzt werden und stellt so den Job Request in die ESS Infrastruktur ein. Damit sind die benötigten Implementationsschritte zur Entwicklung einer ESS Custom Application abgeschlossen.

Nun muss die Anwendung lediglich noch verpackt und auf einem Server ausgeliefert werden. Gerade dieser Teil, der in Anbetracht der vorhandenen und fortgeschrittenen Build-Werkzeugen eigentlich keine große Herausforderung mehr sein sollte, stellt sich bei der Entwicklung einer ESS Custom Application jedoch als umständlich und fehleranfällig heraus. Der gesamte Vorgang ist in der Oracle Dokumentation beschrieben (7.3.5 Assembling the EssDemoApp Application) und umfasst folgende Punkte:

- Konfiguration der Oracle Enterprise Scheduler Properties
- EJB-JAR Deployment Profile konfigurieren
- WAR Deployment Profile konfigurieren
- MAR Deployment Profile konfigurieren
- EAR Deployment Profile konfigurieren
- ADF Security konfigurieren
- Manuelles entfernen von library-ref Tags aus weblogic-application.xml
- Manuelles ersetzen von der *Manifest.inf* Datei im Projekt EssHost
- Ersetzen der adf-config.xml (Zusammenführen trifft es besser)

Gerade die manuelle Ersetzung von Dateien führte bei der Umsetzung häufig zu Fehlern. Wird der Prozess korrekt ausgeführt ist es mit der Applikation jedoch möglich den Java-Job über die ESS Infrastruktur einzuplanen und ausführen zu lassen (siehe Abbildung 6).

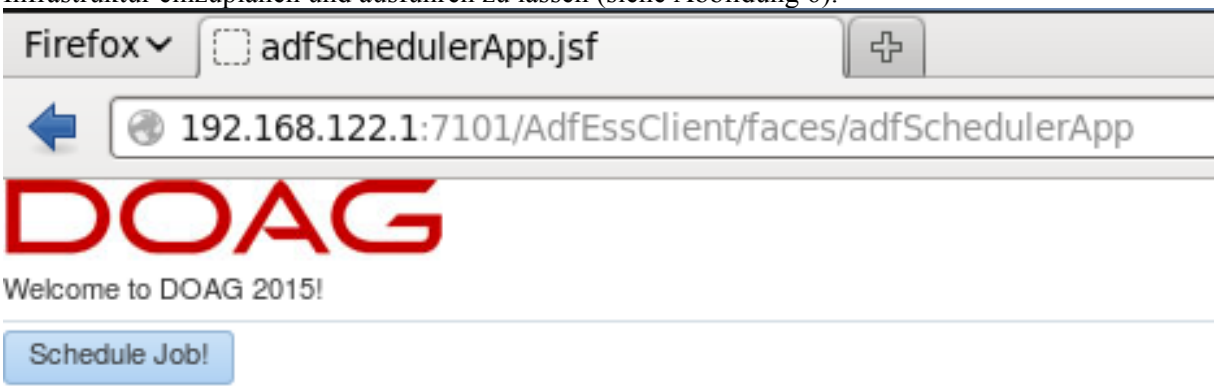

Abb. 6: Laufende ESS Custom Application

Fazit

Die Entwicklung einer ESS Custom Application ist machbar. Im gezeigten Beispiel gestaltete sich die reine Entwicklung einer entsprechenden Anwendung mit ADF sehr einfach. An vielen Stellen bietet JDeveloper komfortable Funktionen um die Entwicklung zu vereinfachen. Hierbei werden allerdings auch viele Schritte vom Werkzeug im Hintergrund durchgeführt und es ist zum Teil unklar was alles passiert. Außerdem wurde während der Umsetzung zum Teil invalides XML generiert.
Die eigentliche Herausforderung bei der Entwicklung einer solchen Applikation scheint jedoch die umständliche Paketierung und Auslieferung auf einem Server zu sein. Dieser Prozess umfasst viele manuelle Schritte die in der Dokumentation beschrieben sind. Hierbei kann es häufig zu kleineren Fehlern kommen die eine erfolgreiche Ausführung der Anwendung verhindern. An dieser Stelle muss die Unterstützung des Werkzeuges definitiv noch verbessert werden.

Kontaktadresse:

Carsten Wiesbaum
esentri AG
Pforzheimer Str. 132
D-76275 Ettlingen

Telefon: +49 (0) 170 – 560 146 1
E-Mail carsten.wiesbaum@esentri.com
Internet: www.esentri.com