Apex und JasperReports – ein starkes Team mit dem JR PrintServer

Michael Schmid, Trivadis GmbH

Gerade für die Entwicklung Daten-basierter Web-Anwendungen stellt Apex ein ausgereiftes Entwickler-Framework dar und hat sich entsprechend auf dem Markt etabliert. In vielen Apex-Projekten besteht die Anforderung, qualitativ hochwertige Druck- beziehungsweise Seiten-orientiere Berichte zu erstellen. Dieser Artikel zeigt, wie das Zusammenspiel von Apex und der JasperReports Library diese Aufgabe elegant und effizient löst.

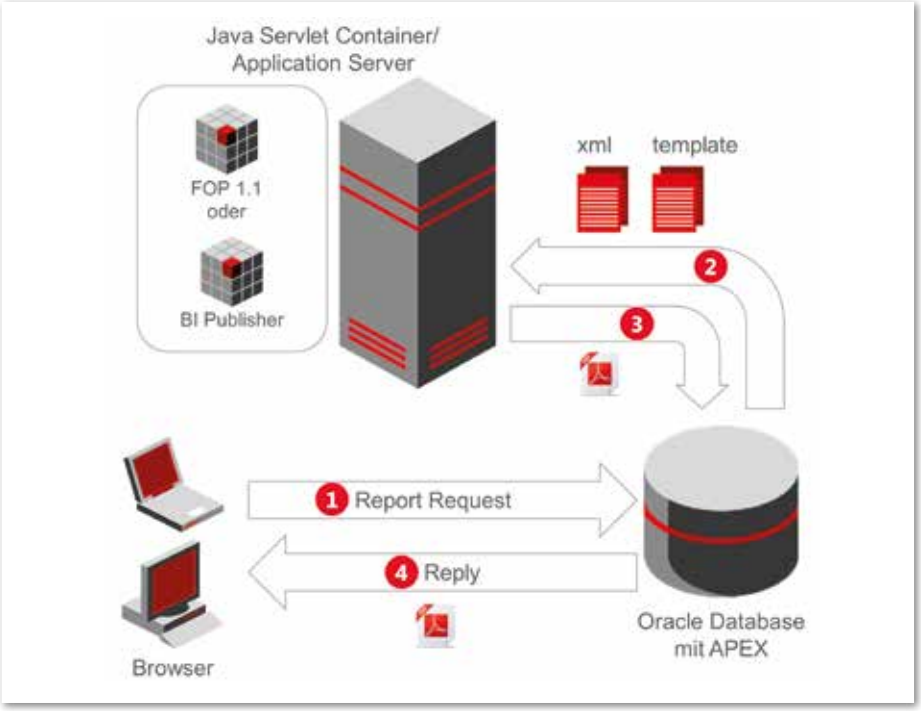

Abbildung 1: Wie Apex mit einem Print-Server kommuniziert

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCUMENT>
  …
  <REGION ID="0">
      <ROWSET1>
          <ROWSET1_ROW>
                  <DEPTNO>10</DEPTNO>
                  <DNAME>ACCOUNTING</DNAME>
              <LOC>NEW YORK</LOC>
          </ROWSET1_ROW>
          <ROWSET1_ROW>
              <DEPTNO>20</DEPTNO>
              <DNAME>RESEARCH</DNAME>
              <LOC>DALLAS</LOC>
          </ROWSET1_ROW>
```

Listing 1: XML-Fragment des HTTP-Post-Parameters „xml" mit den Reportdaten

Apex enthält selbst keinen Mechanismus zum Erstellen Seiten-orientierter Berichte, stattdessen bedient es sich der Hilfe externer Print-Server, die diese Aufgabe übernehmen. Der grobe Ablauf gestaltet sich dabei so, dass Apex zunächst die Daten, die im Bericht erscheinen sollen, aus der Datenbank abfragt und zu XML aufbereitet. Dies wird im Falle von klassischen beziehungsweise interaktiven Reports durch die Bericht-Abfrage oder durch eine entsprechende „Report Query" (unter „Shared Components") geregelt.

Anschließend generiert Apex beim klassischen und interaktiven Report automatisch eine Umwandlungsvorschrift beziehungsweise verwendet eine vom Entwickler vordefinierte Transformations-Spezifikation („Report Layouts" unter „Shared Components"). Diese Information wird ebenfalls als XML zum Print-Server geschickt.

Als letzten Parameter des Aufrufs teilt Apex dem Print-Server das gewünschte Berichtsformat (beispielsweise PDF oder RTF) mit. *Abbildung 1* zeigt die wesentlichen Komponenten und den grundsätzlichen Ablauf.

Out-of-the-Box: Oracle BI Publisher und Apache FOP

Standardmäßig unterstützt Apex den Oracle BI Publisher und Apache FOP. Beim BI Publisher handelt es sich um eine leistungsfähige Reporting-Engine, für die auch Oracle komfortable Entwicklungswerkzeuge zur Verfügung stellt. Allerdings handelt es sich dabei um ein extra zu lizenzierendes Produkt, das im Gegensatz zu

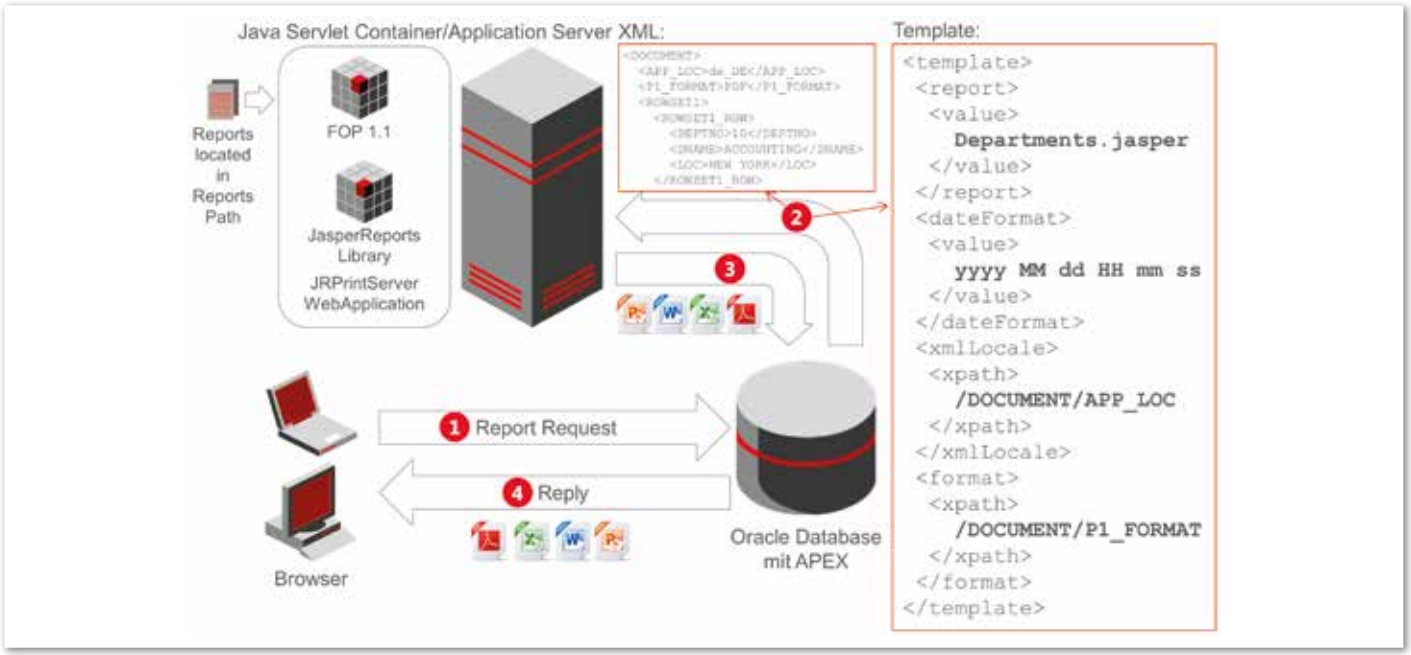

Abbildung 2: Schematischer Aufbau einer Apex-Infrastruktur mit JR PrintServer

Apex selbst nicht in der Datenbank-Lizenz enthalten ist, sodass diese Lösungsoption aufgrund wirtschaftlicher Erwägungen bei kleineren und mittelgroßen Apex-Projekten kaum zum Einsatz kommt.

Apache FOP ist hingegen frei und kostenlos verfügbar; zudem befindet sich eine entsprechende kleine Web-Applikation (fop.war) schon im Apex-Lieferumfang. Leider lässt jedoch die Leistungsfähigkeit der wenigen verfügbaren Entwicklungswerkzeuge für Apache FOP zu wünschen übrig und darüber hinaus werden nur wenige Ausgabeformate (PDF beziehungsweise RTF) unterstützt. Es besteht offensichtlich Verbesserungspotenzial, um auch im Apex-Umfeld Berichte effizient und komfortabel zu erstellen.

JasperReports Library

Bei der JasperReports Library handelt es sich um eine der leistungsfähigsten und am weitesten verbreiteten Java-Reporting-Engines. Die Bibliothek ist seit September 2001 frei und kostenlos verfügbar und kann als stabil und ausgereift angesehen werden. Die bekannten Merkmale einer Reporting-Engine wie Berechnungen, Filterungen, Gruppierungen, Diagramme, Barcodes etc. werden nativ unterstützt. Zudem eröffnet das Feature, Berichte mit Sub-Reports ineinander schachteln zu können, mächtige Möglichkeiten zur Wiederverwendung von Entwicklungsresultaten. Positiv fällt auch auf, dass eine Vielzahl von Ausgabeformaten unterstützt wird wie PDF, RTF, CSV und Text, aber auch die Office-Formate DOCX, XLS, XLSX sowie PTTX und andere. Zu guter Letzt existieren mit dem „Jaspersoft Studio“ und „iReport“ auch noch zwei freie Entwicklungswerkzeuge, um bequem und produktiv entsprechende Berichte zu entwerfen.

Jedoch unterstützt Apex JasperReports zurzeit nicht und auch auf der Roadmap

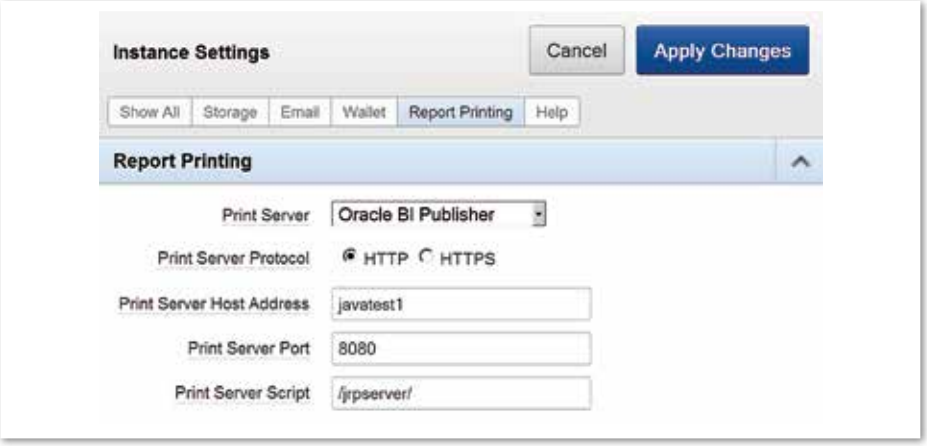

Abbildung 3: Die Print Settings einer Apex-Instanz für den JR PrintServer

```
<?xml version = '1.0' encoding = 'utf-8'?>
<xsl:stylesheet version="2.0"
       xmlns:xsl=http://www.w3.org/1999/XSL/Transform
       xmlns:fo=http://www.w3.org/1999/XSL/Format
       xmlns:xlink="http://www.w3.org/1999/xlink">
    <xsl:variable name="_XDOFOPOS" select="''"/>
    <xsl:variable name="_XDOFOPOS2" select="number(1)"/>
…
```

Listing 2: XML-Fragment eines XSL-FO-Dokuments

```
<template>
  <report>
      <value>myFirstJasperReport.jasper</value>
  </report>
  <dateFormat>
      <value>yyyy MM dd HH mm ss SSS</value>
  </dateFormat>
  <xmlLocale>
      <xpath>/DOCUMENT/APP_LOC</xpath>
  <xmlLocale>
  <xmlTimeZone>
      <value>+09:00</value>
  </xmlTimeZone>
</template>
```

Listing 3: Report Layout für den JR PrintServer

für zukünftige Releases sucht man eine entsprechende Anbindung vergebens – genau hier setzt JR PrintServer an, um diese Lücke zu schließen und den Brückenschlag von Apex zu den Features der JapserReports Library zu ermöglichen.

Aufbau des JR PrintServers

Die Schnittstelle zwischen Apex und einem Print-Server besitzt einen einfachen Aufbau. Der Aufruf besteht aus einem HTTP-Post-Request mit den folgenden drei Parametern:

- *xml*
  Enthält die Daten des Berichts als XML-Dokument und wird automatisch oder aufgrund einer Report Query erstellt. *Listing 1* zeigt das von Apex generierte XML.
- *template*
  Enthält die Transformationsvorschrift zur Erstellung des Berichts und wird automatisch oder mithilfe eines Report Layouts generiert. Sowohl beim BI Publisher als auch bei FOP handelt es sich dabei um ein XSL-FO-Dokument. *Listing 2* stellt beispielhaft einen kurzen Auszug aus einem solchen Dokument dar.
- *_xf*
  Enthält das gewünschte Format im Klartext, also „PDF“, „RTF“ etc.

Der JR PrintServer implementiert nun diese Schnittstelle als Print-Server. Er nimmt also die Daten aus dem xml-Parameter entgegen, so wie auch ein BI Publisher, verwendet jedoch als Template-Parameter kein XSL-FO-Dokument, sondern ein spezifisch aufgebautes XML-Dokument, um die Art und Weise, wie der Bericht mithilfe der JasperReports Library erstellt werden soll, zu spezifizieren.

Abbildung 2 zeigt den Aufbau einer Apex-Infrastruktur mit dem JR PrintServer schematisch. Man erkennt, dass es sich beim JR PrintServer um eine übliche Java-Web-Anwendung handelt, die in einen Java-Servlet-Container wie Apache Tomcat oder Oracle WebLogic geladen wird. Die Web-Applikation enthält neben der JasperReports Library und eigenem Code, der das Apex-Print-Server-Protokoll realisiert, auch Apache FOP, um entsprechende Report Layouts mit XSL-FO zu unterstützen. Dadurch sind die von Apex für klassische und interaktive Reports automatisch erzeugten und auch die selbsterstellten XSL-FO-Report-Layouts mit dem JR PrintServer weiterhin lauffähig.

Um eine Apex-Instanz auf den JR PrintServer als Print-Server einzustellen, müssen weiter nur in der Instanz-Verwaltung die sogenannten „Print Settings“ angepasst werden *(siehe Abbildung 3)*. Die mithilfe von Jaspersoft Studio erstellten und kompilierten JasperReports werden dem JR PrintServer in einem oder mehreren konfigurierbaren Verzeichnissen („ReportsPath“) zum Laden zur Verfügung gestellt.

Report Layout für den JR PrintServer

Wichtig ist vor allem die Struktur des Template-Parameters. Sie wurde im Rahmen der Entwicklung des JR PrintServers spezifiziert:

- *<template>*
  XML-Root-Knoten: Die folgenden Tags können maximal einmal innerhalb des Templates vorkommen

- *<report>*
  Obligatorischer Dateiname des Reports (.jasper-Datei), der ausgeführt werden soll und in einem ReportsPath-Verzeichnis abgelegt sein muss
- *<numberFormat>*
  Optionale Java-Formatmaske für String -> Zahl
- *<dateFormat>*
  Optionale Java-Formatmaske für String -> Datum
- *<xmlLocale>*
  Optionales Java-Locale der XML-Daten (etwa für Dezimalzeichen)
- *<xmlTimeZone>*
  Optionale Java-TimeZone der XML-Daten
- *<reportLocale>*
  Optionales Java-Locale für den Report (z.B. für Dezimalzeichen, Anpassung von Texten)
- *<reportTimeZone>*
  Optionale Java-TimeZone für den Report
- *<format>*
  Optionales Dokumentformat des Reports (z.B. PDF, RTF, DOCX, XLS etc.); übersteuert gegebenenfalls den „_xf"-Parameter

Innerhalb aller XML-Tags kann die Angabe entweder direkt oder über einen XPath-Verweis auf das XML-Datendokument erfolgen:

- *<value>*
  Direkte Angabe des entsprechenden Werts als Zeichenkette
- *<xpath>*
  XPath-Ausdruck, angewandt auf das XML-Datendokument zur Ermittlung des Werts

Listing 3 stellt einen typischen Aufbau eines entsprechenden Report Layouts vor, wobei die Lokalisierung nicht direkt angegeben wurde, sondern als XPath-Verweis auf das Daten-XML.

Ein entsprechendes Report Layout kann bei den Shared Components unter „Report Layouts" angelegt werden. Zu beachten ist dabei, dass als Typ am besten „Generic Columns (XSL-FO)" verwendet werden sollte; im Feld „Page Template" wird das eigentliche Template für den JR PrintServer abgelegt. Die restlichen Felder erhalten Dummy-Einträge *(siehe Abbildung 4)*.

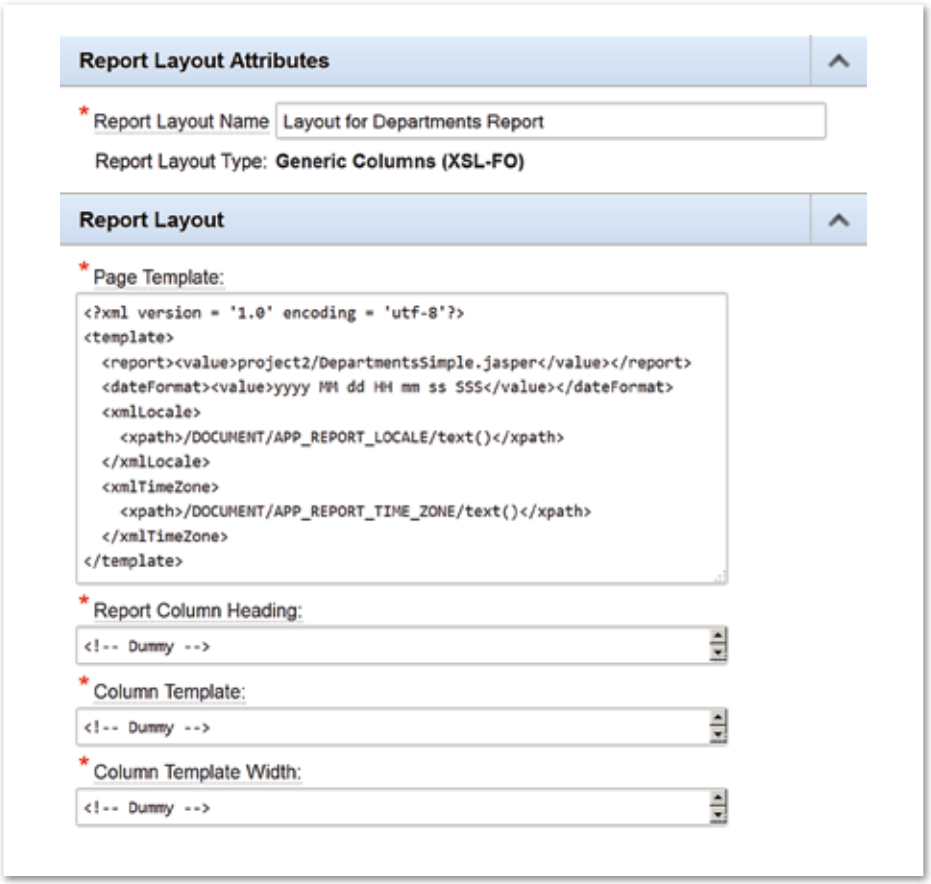

Abbildung 4: Anlegen eines Report Layouts für den JR PrintServer im Apex Application Builder

Abbildung 5: Report Query für den Master-Detail-Report

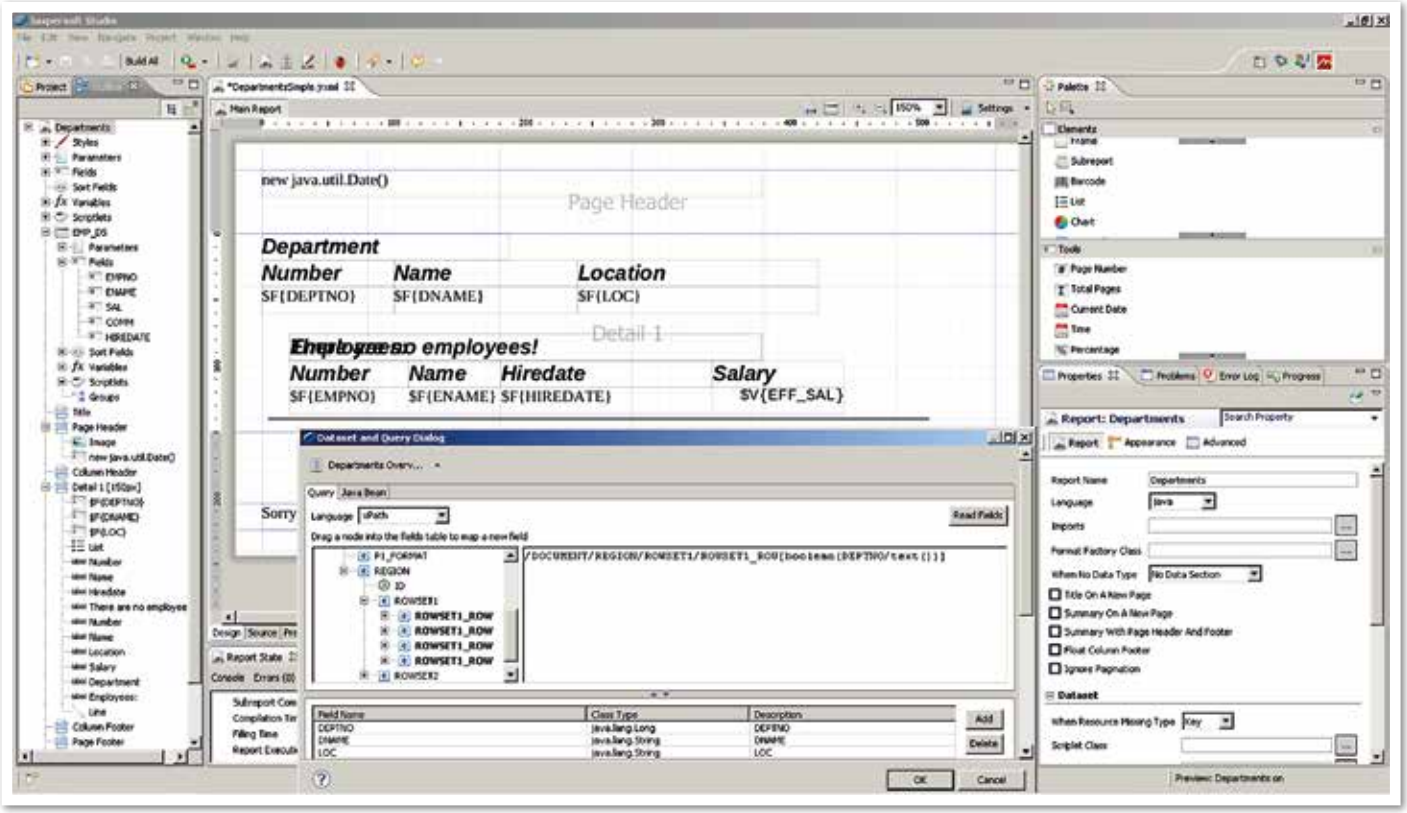

Abbildung 6: Erstellen des Berichts mit Jaspersoft Studio

```
<template>
   <report>
      <value>myFirstJasperReport.jasper</value>
   </report>
   <dateFormat>
      <value>yyyy MM dd HH mm ss SSS</value>
   </dateFormat>
   <xmlLocale>
      <xpath>/DOCUMENT/APP_LOC</xpath>
   <xmlLocale>
   <xmlTimeZone>
      <value>+09:00</value>
   </xmlTimeZone>
</template>
```

Listing 3: Report Layout für den JR PrintServer

```
/DOCUMENT/REGION/ROWSET1
   /ROWSET1_ROW[boolean(DEPTNO/text())]
```

Listing 4: XPath-Abfrage zur Ermittlung der Abteilungen

```
/DOCUMENT/REGION/ROWSET2
   /ROWSET2_ROW
      [number(DEPTNO) = number($P{DEPTNO})]
```

Listing 5: XPath-Abfrage zur Ermittlung der Angestellten einer bestimmten Abteilung

Ein Master-Detail-Bericht

Als Beispiel für die Benutzung des JR PrintServers soll hier ein einfacher Master-Detail-Bericht auf den wohlbekannten EMP- und DEPT-Tabellen des SCOTT-Schemas dienen. Zunächst ist dazu ein Report Layout zu erstellen; das in *Abbildung 4* dargestellte, wäre für unsere Zwecke geeignet. Anschließend sind die Datenquellen, also die Abfragen für den Bericht, zu definieren. Dies erfolgt durch das Anlegen einer Report Query *(siehe Abbildung 5)*.

Die von Apex generierte „Print URL" kann als Ziel eines HTML-Links oder eines Apex-Branch dienen, wodurch der Vorgang zur Berichtserstellung ausgelöst und der daraufhin erzeugte Bericht dem Anwender zum Herunterladen angeboten wird. Insbesondere nützlich ist auch die Download-Funktion *(siehe Button in Abbildung 5)*. Damit lässt sich das von Apex erzeugte XML-Datendokument abrufen, abspeichern und in Jaspersoft Studio als XML-Datenquelle registrieren. Das weitere Design des Berichts erfolgt offline in Jaspersoft Studio, wobei dann in dem Bericht nicht etwa SQL als Abfragesprache dient, sondern XPath, da ja unsere Daten-Grundlage als XML vorliegt *(siehe Abbildung 6)*.

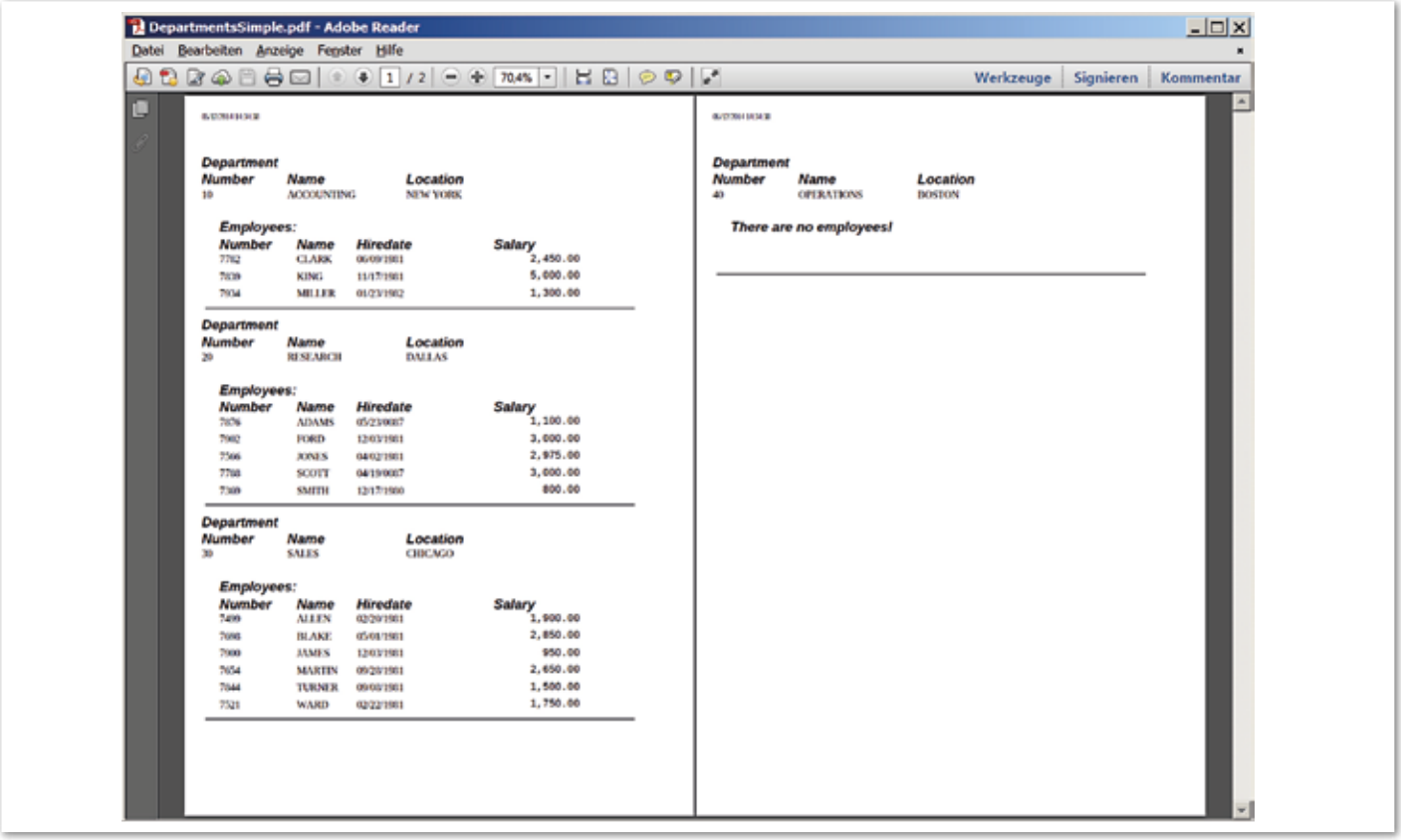

Abbildung 7: Mit JR PrintServer erstellter Bericht als PDF-Dokument im Adobe Reader

Eine Darstellung der Mittel und Möglichkeiten von Jaspersoft Studio würde den Rahmen dieses Artikels sprengen. Ein Punkt soll aber dennoch kurz gezeigt werden, die XPath-Abfragen des Master-(„DEPT“) und auch des Detail(„EMP“)-Datasets (siehe Listings 4 und 5). Im generierten XML-Datendokument finden wir die Abteilungssätze als „ROWSET1_ROW“-Knoten wieder. Diese werden folglich selektiert und zusätzlich wird geprüft, ob der jeweilige DEPTNO-Knoten darunter eine Abteilungsnummer enthält *(siehe Listing 4)*.

Um die Angestellten einer bestimmten Abteilung zu ermitteln, werden zunächst alle „ROWSET2_ROW“-Knoten ausgewählt und anschließend nach der entsprechenden Abteilungsnummer gefiltert. Der dabei verwendete Ausdruck „$P{DEPTNO}“ verweist als Parameter auf die jeweils abgearbeitete Abteilung.

Nachdem das Erarbeiten und Testen des Berichts abgeschlossen ist, kann er kompiliert werden. Die entsprechende „.jasper“-Datei wird zum Server transferiert, auf dem der JR PrintServer läuft, und in einem der ReportsPath-Verzeichnisse abgelegt. Anschließend kann die Berichterstellung von Apex aus initiiert werden. Bei kleinen Berichten (unter 10 Seiten) liegt die benötigte Ausführungszeit typischerweise unter einer Sekunde. *Abbildung 7* zeigt das Ergebnis für den einfachen DEPT-EMP-Bericht.

Fazit

Der JR PrintServer erschließt den Apex-Applikationen die vielfältigen und leistungsfähigen Features der JasperReports Library. Die Berichte können mithilfe der freien Entwicklungswerkzeuge Jaspersoft Studio und iReport einfach und schnell entworfen sowie durch die Endanwender gepflegt werden. Durch die Kombination von JasperReports und der Oracle-Datenbank lassen sich ansprechende Seiten- und Druck-orientierte Berichte für Apex-Anwendungen auf komfortable und performante Weise erstellen. Der JR PrintServer steht auf Sourceforge zum kostenlosen Download bereit.

Weitere Informationen

1. JasperReport Library auf der Jaspersoft-Website: http://community.jaspersoft.com/project/jasperreports-library
2. Jaspersoft Studio auf der Jaspersoft-Website: http://community.jaspersoft.com/project/jaspersoft-studio
3. JR-PrintServer-Projekt auf Sourceforge: http://sourceforge.net/projects/jrprintserver

Michael Schmid
michael.schmid@trivadis.com