Migration, Maven, Möglichkeiten

Stephan La Rocca
PITSS GmbH
Bielefeld

Schlüsselworte
Maven, JDeveloper, ADF, Migration,12.2.1

Einleitung

Projekte, die mit der Verfügbarkeit von Maven im JDeveloper 12.1.3 voller Enthusiasmus in die Welt der POM-Files eingestiegen sind, haben recht schnell an Begeisterung verloren, da der Weg doch recht steinig war. Alleine hier die Fettnäpfe und deren Vermeidung zum Beispiel bei der Verwendung von ADFLibraries zu besprechen, würde in einem Erfahrungsaustausch wertvolles zu Tage bringen.

Mit der aktuellen Version in 12.2.1 ist Maven wesentlich stabiler und wertvoller geworden, aber die Migration vorhandener Projekte birgt weiterhin ihre Tücken. Notwendige Work-arounds gilt es neu zu bewerten und effizient zu entfernen.

Grundkonzept Maven

Bevor auf die Besonderheiten von Maven innerhalb des JDevelopers hingewiesen wird, sollen zunächst kurz die Grundkonzepte von Maven erläutert werden.
Maven ist ein Framework der Apache Software Foundation, welches innerhalb der Software-Entwicklung die Aufgaben für Abhängigkeitsverwaltung, Software-Strukturen und alle Phasen des Buildprozesses vereinfachen soll.
Um diese Aufgaben vereinheitlichen zu können, wird eine Verzeichnis-Struktur für die Software-Artefakte empfohlen.
Zentrales Steuerelement für alle Aufgaben ist die POM (Project Object Model) Datei, die in einer XML-Notation alle Metadaten, die für das Erstellen der Applikation notwendig sind, zusammenfasst.
Der Build-Prozess aus Sicht von Maven wird in einem sogenannte Lifecycle beschrieben, der alle notwendigen Phasen zum Erstellen einer Applikation durchläuft. Phasen sind dabei einzelne Prozess-Schritte von der Validierung der Sourcen, über das Kompilieren und Testen bis hin zum Deployen der zusammengefassten Applikation. In jeder dieser Phasen kann Maven durch PlugIns an die eigenen Gegebenheiten angepasst werden.

Neben der Verwaltung der Phasen ist die Pflege der Abhängigkeiten der Software-Artefakte untereinander ein wichtiger Bestandteil von Maven. Innerhalb der POM.XML wird beschrieben, von welchem Artefakt das zu bauende Objekt abhängig ist. Dabei wird das Artefakt durch die Koordinaten GroupID, ArtefactID, Version und Typ beschrieben, z.B.
GroupID: com.pitss.common.seme
ArtefactID: toolset
Version: 4.2.4
Typ: jar.

Maven im JDeveloper 12.1.x

Mit der Version 12.1.x war es das erste Mal möglich, den Funktionsumfang von Maven auch vollumfänglich innerhalb des JDevelopers zu nutzen. Die Integration bestand darin, dass zum einen

die Projektabhängigkeiten nicht mehr ausschließlich in dem JPR-File des Projektes, sondern auch in dem zugeordneten POM-File gepflegt werden konnten. Diese beiden Stellen wurden synchron gehalten (meistens). Zum anderen war es auch vorgesehen, die einzelnen Phasen des Lifecycles innerhalb des JDevelopers zu starten.

Größte Hürde innerhalb dieser Unterstützung war allerdings, wie Maven mit der Verwendung von ADF-Libraries umging.
Etabliertes Verfahren innerhalb größerer ADF-Projekte ist die Modularisierung mit eigenen ADF-Bibliotheken, die dann einer Master-App als ADF-Libraries wieder eingebunden wurden. Diese Verwendung wird vom JDeveloper allerdings „anonym" über ein ArtefactID „ADF Library" und kaskadierend als „ADF Library Dependency" in das POM-File unter der GroupID com.oracle.adf eingetragen. Diese Verwendung ist innerhalb des Maven Repositories so nicht zu nutzen und es entstand stets ein zusätzlicher Aufwand, das POM-File zum JPR-File synchron zu halten.
Am Ende zeigte sich das beste Ergebnis dadurch aus, das JPR-File und das POM-File eigenständig zu pflegen.

Aufgaben in der Migration
Mit der Version 12.2.1 ist die Maven-Integration deutlich verbessert worden, so dass diese offensichtlichen Hürden nicht mehr bestehen. Für Projekte in einer Zeit vor 12.2.1 stellt sich dann aber die Frage, wie eine Migration aussehen kann.
Restriktiv und sinnvoll ist es nicht mehr möglich, gleichzeitig das POM-File und das JPR-File zu bearbeiten. So ist die erste Entscheidung vor der Migration die Wahl der validen Source aus 12.1.x. Etabliert hat sich das Verfahren, aus dem JPR-File ein neues POM-File zu erstellen und anschließend eventuell fehlende PlugIns nachträglich wieder zu konfigurieren.

In der Version 12.2.1 kann dann eine Verwendung einer eigenen ADF-Library in dem Wizzard des JDevelopers direkt aus dem Maven-Repository erfolgen, so dass die eigenen Koordinaten (group-id, artifact-id, version, typ) durchgängig genutzt werden.

Neben den Lifecycle-Phasen können auch weiterhin die JDeveloper-eigenen Prozesse zum Kompilieren, Generieren und lokalen Testen genutzt werden.
Zu berücksichtigen ist dabei eine klitzekleine Hürde; vor dem ersten Test (RUN) aus dem JDeveloper muss einmalig das Maven-Deploy aufgerufen werden, damit alle abhängigen JARs an die passende Stelle kopiert werden.

Maven im Gesamtkonzept
Mit seiner Mächtigkeit ist Maven mehr als nur ein Hilfsmittel um Sourcen zu Kompilieren und Artefakte zu bauen. In einem Gesamtkonstrukt, in dem mehrere Entwickler in unterschiedlichen Teams an mehreren Produkten arbeiten, ist es eine ideale Plattform, um den Überblick zu behalten.

Zentrales Element ist dabei der Einsatz von Software-Repositories. Maven stellt ein zentrales Software-Repository zur Verfügung (https://repo.maven.apache.org/maven2/.index) in dem alle frei verfügbaren Sourcen gepflegt werden.
Bei jeder Verwendung von Maven auf einem Rechner wird ein Abbild der genutzten Komponenten in einem lokalen Repository (in der Regel im User-Home unter .m2) kopiert.
Für größere Software-Projekte bietet es sich an, darüber hinaus ein unternehmensweites Repository aufzubauen. In diesem werden die erzeugten Artefakte aus den eigenen Teams, gemeinsam genutzte Frameworks oder Beistellungen von anderen Entwicklungs-Partnern eingepflegt. Dieses Repository dient dem Maven-Client als erste „Anlaufstelle" bei der Suche nach neuen oder aktualisierten Artefakten.

Mögliche Tools, die hier eine Erstellung und Administration eines solchen Repository erlauben, sind z.B. Archiva.
In Verbindung mit Continuous Integration Plattformen, wie z.B. Hudson oder Jenkins, die beide eine sehr enge Maven-Integration aufweisen, kann Maven auch den nächtlichen Build-Prozess oder das Bauen eines spezifischen Releases aufgreifen.
Maven kennt den Prozess der Snapshots in den Releases, die eine aktuelle Arbeitskopie der Applikation darstellen. In dem Vortrag wird darauf eingegangen, wie effizient diese Möglichkeit genutzt werden kann.

Abgerundet wird der Vortrag, in dem aufgezeigt wird, wie mit dem JDeveloper ein kompletter Lebenszyklus, von der ersten Arbeitsversion der Software, bis hin zum Installieren der fertigen Artefakte in das unternehmensweite Repository gesteuert werden kann.

Kontaktadresse:
Stephan La Rocca
PITSS GmbH
Otto-Brenner-Str. 209
D-33604 Bielefeld

Telefon: +49 (0) 521 5467 9507
Fax: +49 (0) 521 5467 9501
E-Mail slarocca@pitss.com
Internet: www.pitss.de