Entwicklung von Komponenten für Webcenter Content mit Eclipse und Maven

Thorsten Wussow

Slix GmbH

Au i. d. Hallertau

Schlüsselworte

Webcenter Content, UCM, Komponenten, Java, Maven, Eclipse

Einleitung

Oracle Webcenter Content ist nicht nur ein leistungsfähiges Produkt sondern es beinhaltet auch ein Framework für nahezu beliebige Erweiterungen. Eigene Services, SQL-Abfragen, Datenbankobjekte, Javaklassen und eigene IDOC-Scripte können über Komponenten in den Content Server integriert werden. Die von Oracle bereitgestellten Tools wie z.B. der ComponentWizard dienen zur grundlegenden Erstellung einfacher Komponenten. Werden die Komponenten komplexer, stoßen diese Tools wegen des immer noch notwendigen manuellen Aufwands schnell an ihre Grenzen

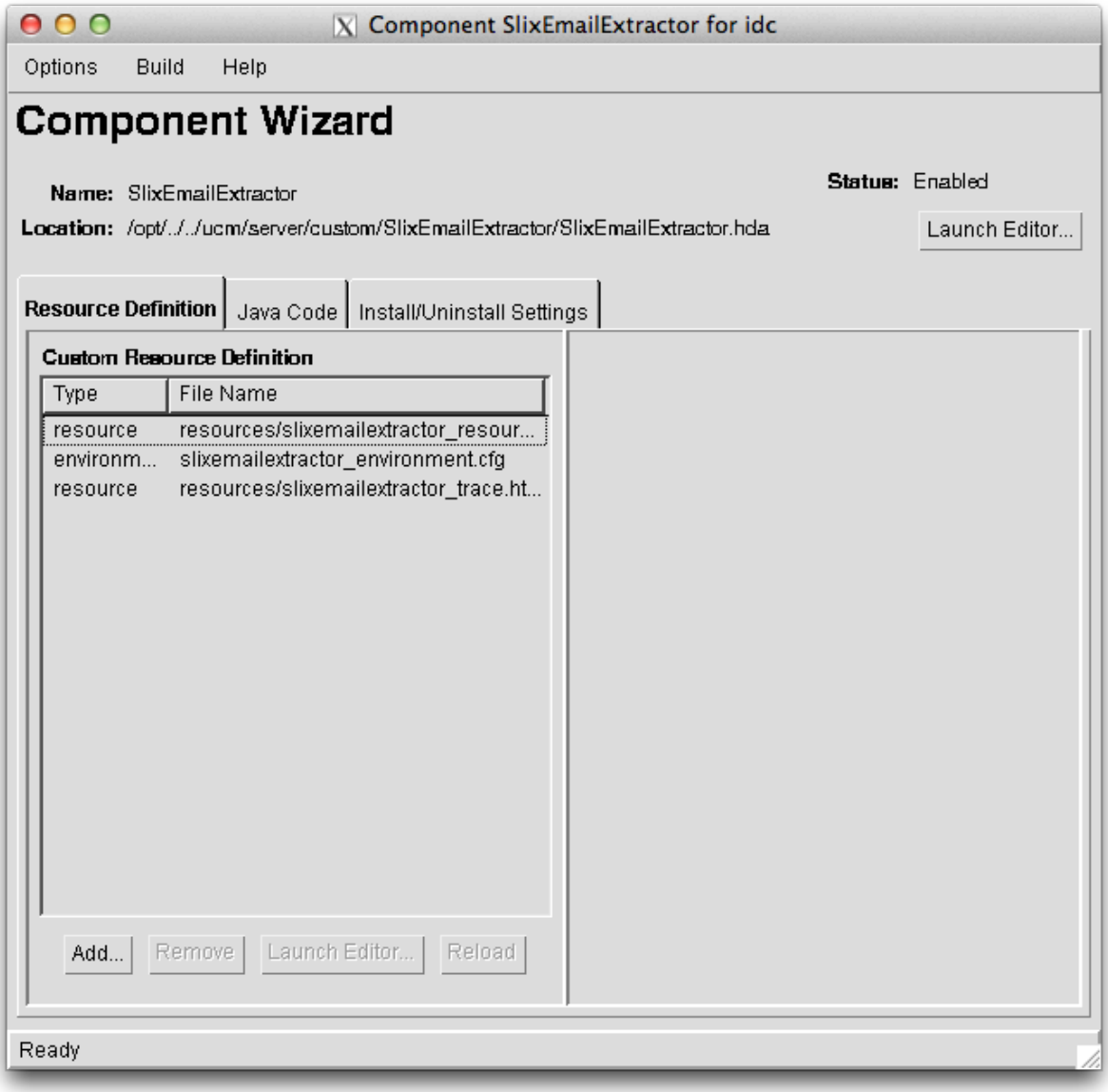
Der Vortrag demonstriert wie man den kompletten Prozess beginnend mit der Einrichtung der Entwicklungsumgebung über die Entwicklung bis zur Installation von Komponenten im ContentServer mit Eclipse und Maven integriert und Routineaufgaben automatisiert werden können.

Grundlagen für den Vortrag

Für den Vortrag sollten das Komponentenmodell und Grundlagen von Webcenter Content bekannt sein. Ebenso sollten Erfahrungen in Java Eclipse einschließlich der grundlegenden Verwendung von Maven vorliegen.

Komponentenentwicklung vorher

Oracle liefert zur Entwicklung von Komponenten für Webcenter einen Componenten Wizard.



Dieses Werkzeug ist für einzelne Komponenten und für erste Schritte ganz hilfreich, stößt jedoch schnell an seine Grenzen.

Typisch dafür ist folgender Ablauf:

- Erstellen der Komponente im ComponentWizard

- Edititeren der .hda-Dateien und Resourcedateien mit Notepad++ (auf Windows) inkl. eines Plugins zum Syntaxhighlighting (wenn ein UCM-Server verwendet wird, der nicht lokal auf dem Entwicklungsrechner liegt, dann müssen die Dateien zunächst kopiert werden)

- Erstellen des Javacodes in JDeveloper für Filter oder neue ServiceHandler

- Kopieren der erstellten Classdateien und resource und hda-Dateien auf den UCM-Server

- Neustart von UCM und testen des Codes

- Wenn alles fertig ist dann das Java-Projekt und die Sourcen in ein SCM einchecken und über den ComponentWizard die fertige Komponente erstellen lassen

Diese Art der Entwicklung hat einige zeitaufwendige Schritte und eine Teamarbeit ist nur schwer möglich. Meist wird auf dem Entwicklungsrechner eine lokale UCM-Instanz installiert um das kopieren der Dateien etwas einfacher zu machen. Aber seit UCM11g kommt ja der Weblogic-Server mit und da steigt der Resourcenverbrauch auf dem Entwicklungsrechner.

Das beschriebene Vorgehen lässt sich nicht in den Entwicklungsprozess integrieren. Spätestens, wenn man mit der Anforderung zur Entwicklung von Komponenten für verschiedene Content Server Versionen konfrontiert wird, reift der Wunsch nach besseren Alternativen.

So geht's besser

Mit einem guten Template , einigen wenigen Konventionen, Maven, Plugins und ein paar Skripten kann weitestgehend in Eclipse entwickelt werden. Die Projektstruktur kann mit Hilfe eines Templateprojektes angelegt werden, welches nach der DOAG auf unserer Homepage zum Download bereitgestellt wird. Der ComponentWizard wird nur noch zum initialen Erstellen der Komponente benötigt (wenn man die Inhalte der manifest.hda und des Komponentendeskriptors kennt, kann man auch darauf verzichten). Die initial erstellten Resourcendateien werden dann in die Projektstruktur kopiert. Alles weitere, das Erstellen der Komponente, das Deployment der Komponente zum Testen auf den ContentServer wird auch aus Eclipse heraus gestartet und durch Maven ausgeführt.

Dieses Verfahren hat den Vorteil, dass die Projektstruktur komplett in einem SCM (bei uns GIT) abgelegt werden kann und somit auch eine Teamarbeit oder auch die Anbindung an ein Continous Integration (CI) System wie z.B. Hudson oder Bamboo möglich ist.

In dem Vortrag wird das Erstellen einer Komponente nach diesem Verfahren gezeigt.

Entwicklungsumgebung für die Demonstration

Hier wird die Entwicklungsumgebung dargestellt welche für die Demo während des Vortrages verwendet wird.

Der Demorechner ist ein Laptop mit 6GB RAM

Als Webcenter Content Server wird eine Pre-built Developer VirtualBox-VM für SOA und BPM verwendet. Diese enthält Webcenter Content. Diese VM ist so zu konfigurieren, dass nur der ContentServer und der AdminServer laufen. Alles andere wird hier nicht benötigt.

Es wird Eclipse Juno (aktuellste Version) mit Maven und Git-Integration verwendet.

Als zentrales Repository der Maven Artifacts wird Artifactory verwendet (ist aber optional und wird nur gezeigt wenn noch Zeit bleibt).

Struktur Templateprojekt

Die Verzeichnisstruktur des Templateprojektes basiert im wesentlichen auf der empfohlenen Verzeichnisstruktur eines Maven-Projektes. und wurde um diverse Verzeichnisse für die Komponentenentwicklung erweitert. Das ist die Grundlage für den Build. Entwickler brauchten sich um die eigentlichen Strukturen einer Komponente nicht weiter zu kümmern.

Struktur	**Beschreibung**
Hauptverzeichnis/	Verzeichnis welches alle Daten enthält
| – buildnumber.properties	Diese Datei enthält die aktuelle Buildnumber. Wird von dem Maven-Versions Plugin verwendet (optional)
| – pom.xml	Maven Steuerdatei
| – releasenotes.txt	Die Releasenotes oder readme der Komponente. Wird durch Maven in die Komponente kopiert
| – settings.xml	Enthält die benötigten Einstellungen der lokalen Maveninstanz als Beispiel. Diese Einstellungen können in die lokale settings.xml übertragen werden, oder es kann diese Datei direkt verwendet werden.
| – src/	Hier sind alle Quelldateien der Komponente enthalten
| | – main/	
| | | – assembly/	Enthält assembly-descriptoren
| | | | – component-assembly.xml	assembly-descriptor zum erstellen der Komponenten Zip-Datei
| | | – componentresources/	Enthält hda- und htm-Dateien und sonstige resourcendateien für die Komponente
| | | | – additional/	zusätzliche had-Dateien (optional)
| | | | – additionalcomponents/	zusätzliche Komponenten die mit der Komponente installiert werden sollen (kann auch über ein zentrales Repository erfolgen, dann kann das Verzeichnis entfernt werden)
| | | | – componentdescriptor/	Enthält den Descriptor der Komponente. Die Datei sollte dem artifactId der pom.xml entsprechen
| | | | – environment/	Umgebungsdefinitionsdateien für die Komponente
| | | | – manifest/	Enthält die Manifestdatei der Komponente

| | | – publish/	Dateien die im Publish-Zyklus veröffentlicht werden sollen (Layout und Skins)
| | | – resources/	Die Resourcedateien der Komponente
| | | – templates/	Die Templatedateien der Komponente
| | | – webresources/	Enthält Verzeichnisse die so in das Weblayout des Contentservers kopiert werden (Bilder, Hilfe, usw.)
| | – doc/	Enthält die Dokumentation zur Komponente (optional)
| | – hda/	Enthält zusätzliche hda-Dateien (Workflowdefinitionen, Rules, usw., optional)
| | – help/	Enthält die Onlinehilfe zur Komponente (optional)
| | – lib/	Enthält Librarydateien (jar-Files) für die Komponente, wie z.B. server.zip und classes.jar dateien anderer Komponenten um die Java-Klassen zu kompilieren
| | – sql/	Enthält SQL-Skripte (optional)
| | – wlst/	Enthält die Start-/Stop-Skripte für Webcenter Content 11g
| – site/	Enthält eine site für das Maven-Projekt (optional)
| – test/	Enthält die Testsourcen für JUnit-Tests

Libraries

Die benötigten Artifacts und deren Abhängigkeiten werden als Dependencies in der pom.xml geflegt, und müssen im Repository (lokal oder remote) vorhanden sein. Die Erstellung dieser Einträge kann durch die IDE automatisiert werden. Für das Kompilieren einer Komponente mit Javaklassen wird mindestens die idcserver.jar von WebCenter Content benötigt. Werden Java-Klassen von weiteren Komponenten von WebCenter Content angesprochen, wie z.B. Folders_g dann werden auch die entsprechenden .jar-Dateien benötigt. Diese kopiert man am besten aus einer laufenden WebCenter Content-Installation.

Dateien für Webcenter Content 11g

Library	**Pfad im ContentServer**
idcserver.jar	<CS-HOME>/Oracle_ECM1/ucm/idc/jlib
classes-Folders_g.jar	<CS-HOME>/Oracle_ECM1/ucm/idc/components/Folders_g

Dateien im Templateprojekt

Die im Template erforderlichen (und als Beispiel vorhandenen) Dateien werden hier mit den notwendigen Anpassungen beschrieben

pom.xml

Die in dem Template enthaltene pom.xml enthält exemplarisch den Aufbau einer Komponente. Das ist auch der Platz für Erweiterungen und Anpassungen nach eigenen Bedürfnissen. Hier sind zum Beispiel Developmentprofile definiert, eines für die Verwendung mit Webcenter Content 11g, eines für Webcenter Content 10g und eines für eine Bereitstellung auf einem Webserver zum Download. Die Einstellungen auf verschiedene Umgebungen ist durch Properties realisiert. Bei einer anderen Umgebung müssen also nur die entsprechenden Properties angepasst werden. In der folgenden Tabelle sind die Properties und ihre Bedeutung aufgeführt

Property	**Beschreibung**
groupId	Entsprechend der eigenen Umgebung anpassen
artifactId	Entspricht dem Namen der zu erstellenden Komponente, auf Groß-Klein-Schreibung achten
version	Die aktuelle Version des Projektes, kann an die Komponente angehängt werden
name	Ausführlicher Name der Komponente (optional)
organization	Daten zur Organisation (optional)
description	Eine Beschreibung der Komponente (optional)
wcc10ghome	Pfad zum Homeverzeichnis von WCC 10g
wcc11gdomain	Pfad zum Domänenverzeichnis von WCC 11g
wccweblogic	Pfad zum WebLogic-Server von WCC 11g
maven.build.timestamp.format	Format des Timestamps für das Build-Datum
componentVersion	Version der Komponente, wird an den Komponentennamen angehängt
componentDate	Maven Build-Datum als aktuelles Komponentendatum
testhost	IP oder Domainname des WCC Testservers für die Integrationstests
testuser	Benutzername für den Testserver
testpass	Passwort für den testuser

webhost	IP oder Domainname des Webservers auf den die Site deployed werden kann (optional)
webroot	Pfad zur Webserver Documentroot, wo die Site hin kopiert werden soll (optional)
repositoryLib	Library im Repository für Releases (optional)
repositorySnapLib	Library im Repository für Snapshots (optional)
repositoryUrl	URL für den Zugriff auf ein Repository um die Komponenten bereitzustellen oder zu andere zu Benutzen (optional)
uploadDirectory	Das Verzeichnis auf dem Webserver, wo die Daten für Kunden zum Download angeboten werden (optional, für Production-Profil)
componentName	Der Aufbau des Namens der Komponente kann hier festgelegt werden. Es wird empfohlen hier nichts zu ändern.

settings.xml

In dieser Datei sind die Vorlagen enthalten zur Definition der Credentials für den Testserver, den Webhost und ein Remoterepository. Die Credentials sind entsprechend in die lokale settings.xml von Maven zu übertragen und anzupassen.

component-assembly.xml

Die component-assembly definiert die Komponentenstruktur für Maven. Über die Assembly wird die Komponente zusammengesetzt. Werden neue Verzeichnisse innerhalb der Komponente benötigt, so muss diese Datei entsprechend angepasst werden.

WLST-Skripte für Webcenter Content 11g

Zum Starten und Stoppen wird im Templatearchiv ein Satz WLST-Skripte zur Verfügung gestellt, die von Maven verwendet werden. Dazu ist die wlst_params.properties an die Umgebung anzupassen. Die Skripte werden in das /tmp-Verzeichnis auf den Testserver kopiert und von dort ausgeführt. Dies setzt voraus, dass der NodeManager auf dem Entwicklungsserver läuft.

manifest.hda

Eine manifest.hda ist als Beispiel in dem Templateprojekt enthalten. Diese sollte entsprechend der zu erstellenden Komponente angepasst werden, z.B. wenn weitere WebResources verwendet werden oder eigene Logos, usw. Dadurch wird der Ablageort innerhalb des ContentServers definiert.

Es gibt folgende Typen:

Entry Type	Beschreibung	Standard Zielpfad
Classes	Java class-Dateien, wenn nicht mit einer .jar gearbeitet wird	DomainHome/ucm/short-product-id/classes
Common	Standarddateien zur anzeige im Weblayout	DomainHome/ucm/short-product-id/weblayout/common
Component	Die Komponenten-hda-Datei	DomainHome/ucm/short-product-id/custom
ComponentExtra	Zusätzliche Dateien wie eine readme oder diverse jar-Dateien oder zusätzliche Komponenten	DomainHome/ucm/short-product-id/custom
Help	Online Hilfe Dateien (deprecated in 11g)	DomainHome/ucm/short-product-id/weblayout/help
Images	Zusätzliche Bilder für die Komponente (deprecated in 11g)	DomainHome/ucm/short-product-id/weblayout/images
Jsp	zusätzliche JavaServer Pages	DomainHome/ucm/short-product-id/weblayout/jsp
weblayout	Webresources wie zusätzliche Javascripts	DomainHome/ucm/short-product-id/weblayout

Komponentendeskriptor

Hier werden die Definitionen und Zuordnungen der in der Komponente benötigten Resourcen gepflegt. Die Datei muss zumindest ein *ResourceDefinition* ResultSet enthalten. In diesem ResultSet wird der Typ, der Dateiname und die Ladereihenfolge der Resourcendateien definiert. Weiterhin können in der Datei ein *MergeRules* und ein *Filters* ResultSet enthalten sein. Eine detaillierte Beschreibung dazu kann der Dokumentation zu WebCenter Content entnommen werden. Die Datei hat noch einige besondere Felder im Header, welche z.B. den Classpath und zusätzliche Komponenten definieren. Die folgende Tabelle enthält einige dieser Felder:

Feld	Beschreibung
classpath	z.B. $COMPONENT_DIR/classes.jar Hier werden die .jar Dateien welche die Komponente benötigt angegeben
classpathorder	Gibt an wo im Classpath des Systems die Komponente angehängt wird, 1 bedeutet ganz vorne.
libpath	z.B. $COMPONENT_DIR/lib Pfad zu den zusätzlich benötigten Libraries (-Dlibrary-path)
libpathorder	Gibt an wo im libpath des Systems die Komponentenlibraries angehängt werden, 1 bedeutet ganz vorne
version	Hier steht die Version der Komponente. Im unten stehenden Beispiel wird die Version durch eine Mavenvariable gesetzt
additionalComponents	Eine Liste der zusätzlichen Komponenten. Diese müssen als ComponentExtra in der manifest.hda definiert sein Name der Komponente:Name der zip-Datei,...
disableZipFileBackup	Gibt an ob ein Backup der in der Komponente evtl. enthaltenen zusätzlichen Komponenten erstellt wird.

Vorführung

Im Vortrag werde ich die Entwicklung einer Komponente Schritt für Schritt vorführen und folgende Schritte live zeigen.

- Importieren des Templateprojektes in Eclipse und Anpassen der notwendigen Einstellungen

- Initiales Erstellen der Komponente und kopieren der notwendigen Dateien in die Projektstruktur

- Anlegen eines ServiceHandlers

- Erstellen der Komponente und Deployment auf den Server mit Maven

- Testen der neuen Komponente

- Erweitern der Komponente

- erneutes Erstellen und Deployment auf den Server mit Maven

Falls noch Zeit bleibt:

- Verwenden von jUnit

- Verwenden eines zentralen Repositories (auch für Abhängige Komponenten)

Kontaktadresse:
Thorsten Wussow
Slix GmbH
Nandlstädter Weg 6
D-84072 Au i. d. Hallertau

Telefon:	+49 (0) 3212-1273698
Mobil:	+49 (0) 173-3208013
Fax:	+49 (0) 3212-1273698
E-Mail	thorsten.wussow@slix.de
Internet:	http://www.slix.de