

Nils Hartmann ♦ Gerd Wütherich

Build my bundle!

oder: Es muss nicht immer PDE sein...

Inhalt

» Theorie

- » Bauen von Software
- » Projekte, Projektbeschreibungen und Definition von Abhängigkeiten
- » Integration bestehender Bibliotheken
- » Repositories

» Praxis

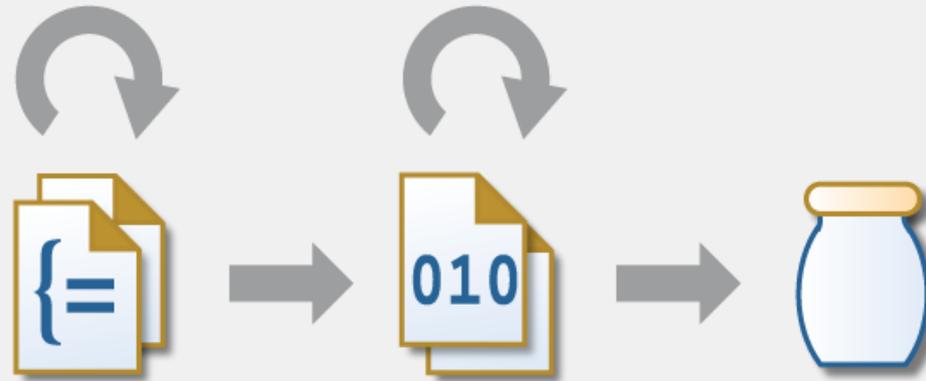
- » Manifest First
 - » Eclipse PDE (PDE Build, Pluginbuilder, Ant4Eclipse, Tycho)
 - » SpringSource dm Server Tools
- » Generate Manifest
 - » BND (MAVEN + FELIX Tools)
 - » Bundlor

» Fazit

Bauen von Software

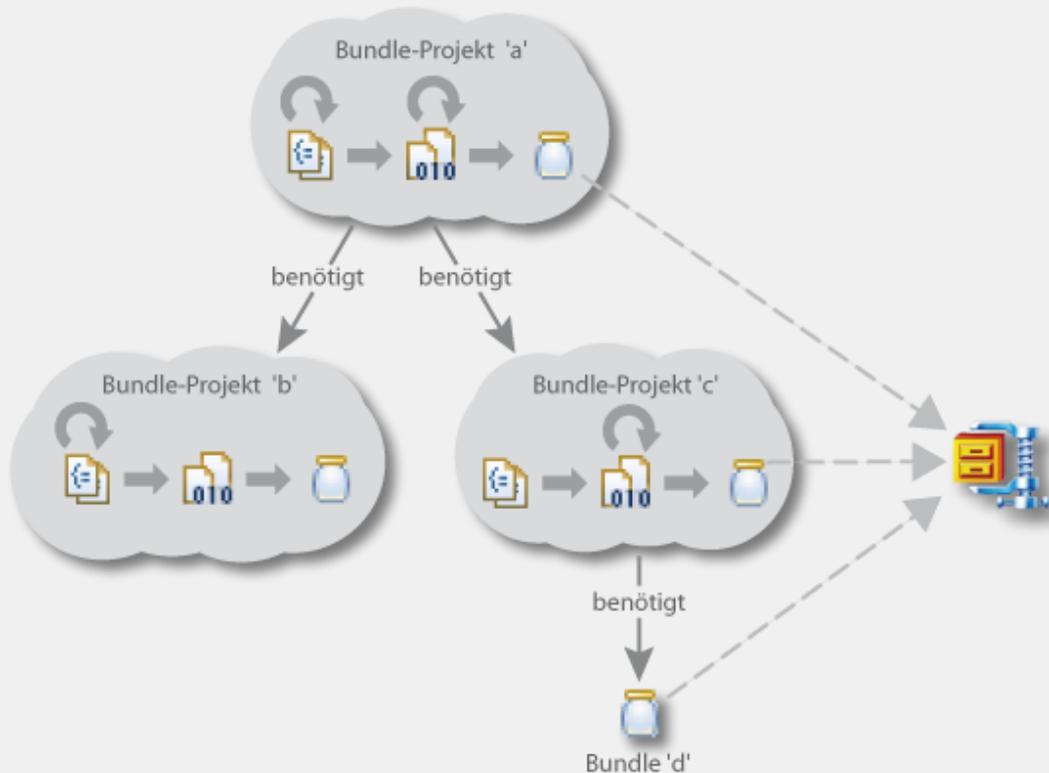
- » Das Build-System ...
 - » ...ist einer der größten Produktivitätsvervielfältiger innerhalb einer Entwicklungsorganisation
- » Studie von 2002:
„In komplexen Projekten werden schätzungsweise 10-30% der Entwicklungszeit für
 - » Auseinandersetzungen mit dem Build-Tool,
 - » das Warten auf langsame Builds oder
 - » das Suchen nach Phantom Bugs aufgrund inkonsistenter Builds verwendet.“

Was bedeutet „Bauen von Software“?



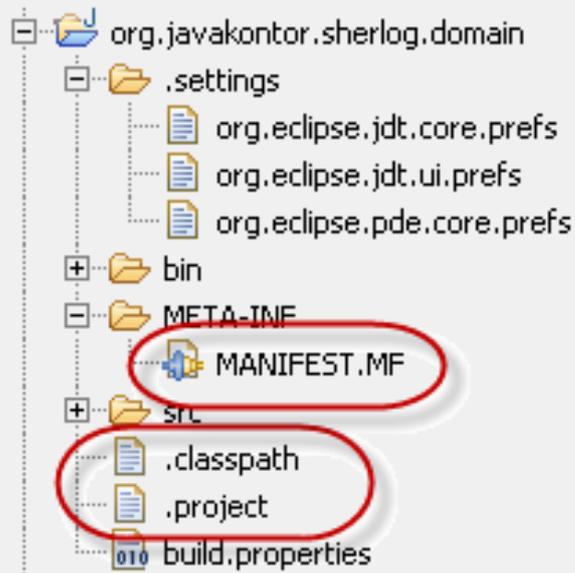
- » Entwickler schreiben Sourcecode
- » Der Build erzeugt aus den Quelldateien ...
 - » ... ein ausführbares Programm *oder*
 - » ... ein beliebiges Artefakt, das in anderen Programmen genutzt werden kann.
- » Der Build muss Abhängigkeiten zwischen Quelldateien auflösen können

Projekte



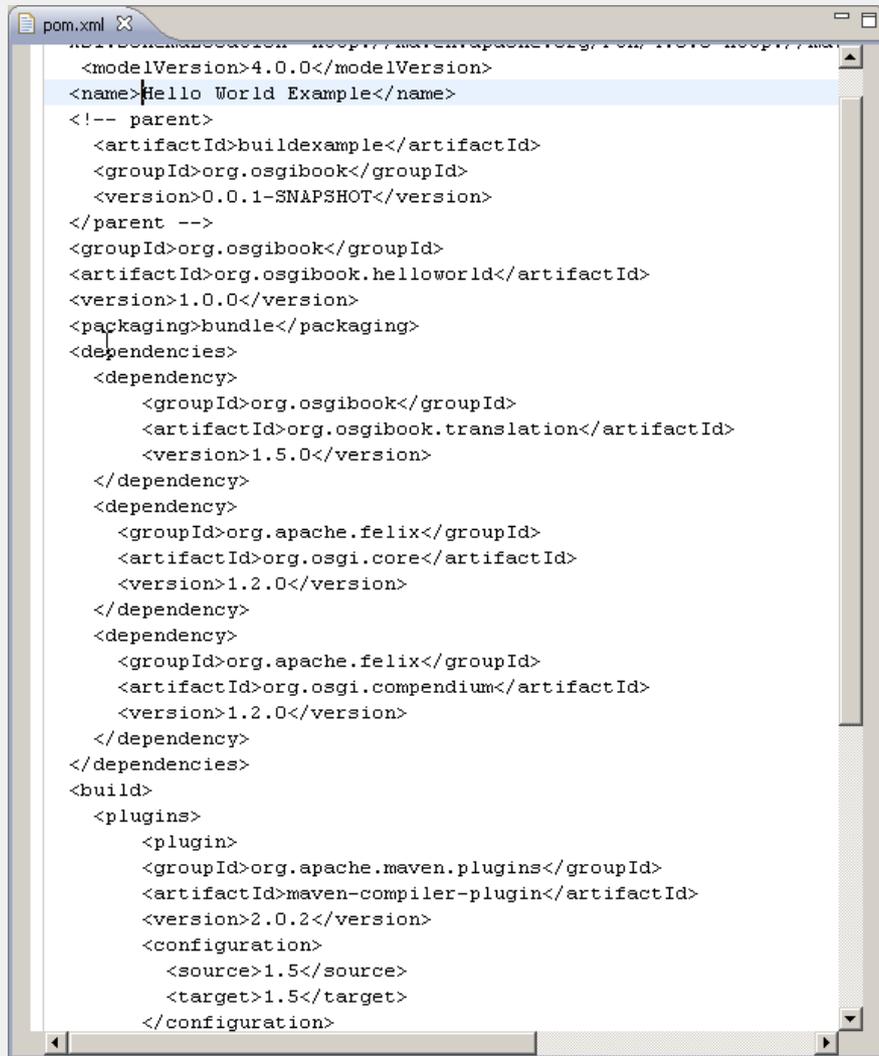
- » **Projekte:**
Quelldateien sind in Projekten organisiert
- » **Projektbeschreibung:**
Metainformationen zu einem Projekt
- » **Abhängigkeiten:**
Projektbeschreibung definiert Abhängigkeiten zu anderen Projekten, JAR-Files und/oder Bundles

Beispiel: Eclipse-Projekte



- » Projekte befinden sich innerhalb des Workspace
- » Projektinformationen in den Dateien `.project` und `.classpath`
- » Zusätzliche Informationen unterhalb des Ordners `.settings`

Beispiel: Maven-Projekte



```
pom.xml
<modelVersion>4.0.0</modelVersion>
<name>Hello World Example</name>
<!-- parent -->
<artifactId>buildexample</artifactId>
<groupId>org.osgibook</groupId>
<version>0.0.1-SNAPSHOT</version>
</parent -->
<groupId>org.osgibook</groupId>
<artifactId>org.osgibook.helloworld</artifactId>
<version>1.0.0</version>
<packaging>bundle</packaging>
<dependencies>
<dependency>
<groupId>org.osgibook</groupId>
<artifactId>org.osgibook.translation</artifactId>
<version>1.5.0</version>
</dependency>
<dependency>
<groupId>org.apache.felix</groupId>
<artifactId>org.osgi.core</artifactId>
<version>1.2.0</version>
</dependency>
<dependency>
<groupId>org.apache.felix</groupId>
<artifactId>org.osgi.compendium</artifactId>
<version>1.2.0</version>
</dependency>
</dependencies>
<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>2.0.2</version>
<configuration>
<source>1.5</source>
<target>1.5</target>
</configuration>
</plugin>
</plugins>
</build>
```

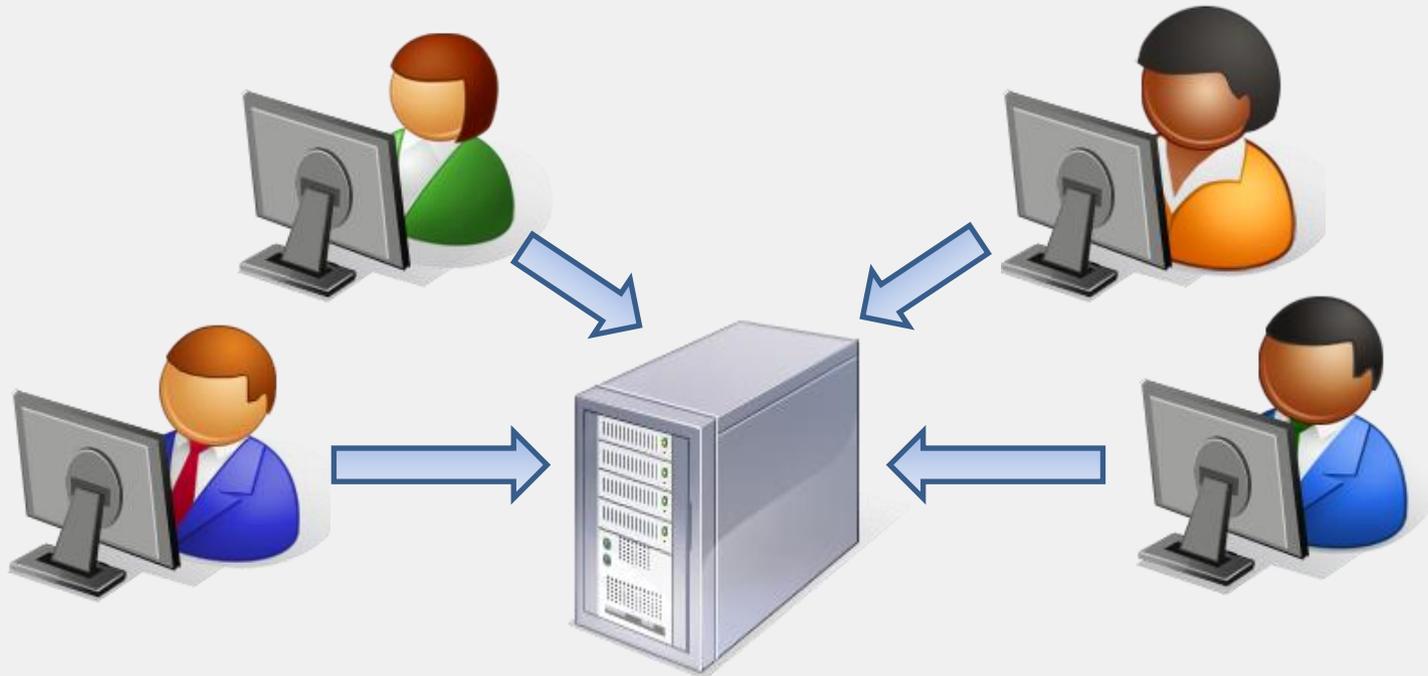
- » Maven-spezifisches Projektlayout
- » Projekt-Metainformationen werden innerhalb der Datei pom.xml beschrieben
- » Abhängigkeiten werden gegen das Maven-Repository aufgelöst

Bauen von Software in der IDE



- » Produktivität des Entwicklers steht im Vordergrund
 - » Geringe Turn-Around-Zyklen (Inkrementeller Build etc.)
 - » Komfortables Arbeiten (Code Completion, Wizards etc.)

Automatik-Build



Anforderungen an den Automatik-Build:

- » Complete, Repeatable, Informative, Schedulable, Portable

IDE vs. Automatik-Build

- » Projektbeschreibungen sind innerhalb der IDE und des Automatik-Build häufig verschieden
- » *Resultat:*
 - » Inkonsistenzen zwischen IDE und Automatik-Build
 - » Fehler im Automatik-Build sind (für den Entwickler) schwer auffindbar
- » *Forderung:*
 - » Keine Redundanzen in den Projektbeschreibungen innerhalb der IDE und des Automatik-Builds

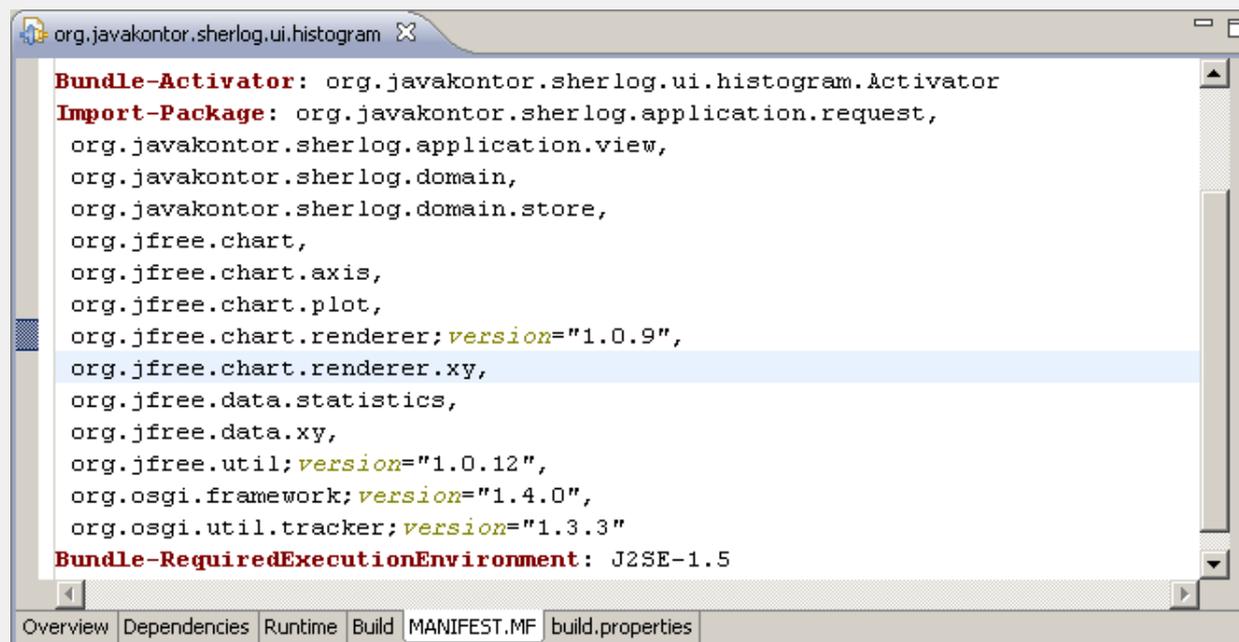
Definition der Abhängigkeiten

- » Bundles haben Abhängigkeiten *zur Laufzeit*
- » Projekte haben Abhängigkeiten *zur Compile-Zeit*

- » Frage:
 - » Wie werden die Abhängigkeiten zur Laufzeit und zur Compile-Zeit beschrieben?
 - » Wie lassen sich Redundanzen in den Beschreibungen vermeiden?

Laufzeit-Abhängigkeiten in OSGi

- » Die OSGi Service Platform erlaubt das Management von Abhängigkeiten *zur Laufzeit*
- » Relevante Einträge im Bundle-Manifest:
 - » Export-Package, Import-Package, Require-Bundle, Fragment-Host



```
org.javakontor.sherlog.ui.histogram
Bundle-Activator: org.javakontor.sherlog.ui.histogram.Activator
Import-Package: org.javakontor.sherlog.application.request,
org.javakontor.sherlog.application.view,
org.javakontor.sherlog.domain,
org.javakontor.sherlog.domain.store,
org.jfree.chart,
org.jfree.chart.axis,
org.jfree.chart.plot,
org.jfree.chart.renderer;version="1.0.9",
org.jfree.chart.renderer.xy,
org.jfree.data.statistics,
org.jfree.data.xy,
org.jfree.util;version="1.0.12",
org.osgi.framework;version="1.4.0",
org.osgi.util.tracker;version="1.3.3"
Bundle-RequiredExecutionEnvironment: J2SE-1.5
```

Overview Dependencies Runtime Build MANIFEST.MF build.properties

Compile-Zeit-Abhängigkeiten

- » Definition von Abhängigkeiten *zur Compile-Zeit* innerhalb der jeweiligen Projektbeschreibung:
 - » Abhängigkeiten zu anderen Projekten,
 - » JAR-Files oder
 - » Bundles
- » Auswirkungen auf den Build-Prozess:
 - » Sichtbarkeit von Bibliotheken/Packages während des Kompilierens (,Classpath‘)
 - » Build-Reihenfolge der Projekte

Mögliche Ansätze

- » **Möglichkeit 1 - Two are better than one:**
 - » Abhängigkeiten in Projektbeschreibung
 - » Manuell erstelltes Manifest
- » **Möglichkeit 2 - Generate-Manifest:**
 - » Abhängigkeiten in Projektbeschreibung
 - » Generiertes Manifest
- » **Möglichkeit 3 - Manifest-First:**
 - » Abhängigkeiten im Manifest
 - » Manifest treibt den Buildlauf

Generate-Manifest vs. Manifest-First

Manifest-First

- » Vorteile:
 - » Standardisierte Beschreibung der Abhängigkeiten
 - » Direktes Feedback bzgl. der OSGi-Package-Abhängigkeiten in der IDE
- » Nachteile:
 - » Fehlerhafte Interpretation der OSGi-Package-Abhängigkeiten führen zu fehlerhaften/unnötigen Manifest-Einträgen

Generate-Manifest

- » Vorteile:
 - » Bestehender Build-Prozeß kann i.d.R. weiterverwendet (und erweitert) werden
- » Nachteile:
 - » Bestehende Abhängigkeitsbeschreibungen (POM, Eclipse .classpath) u.U. nicht ausreichend (Versionen, Versionsbereiche etc.)

Weitere Anforderung: Integration bestehender Artefakte

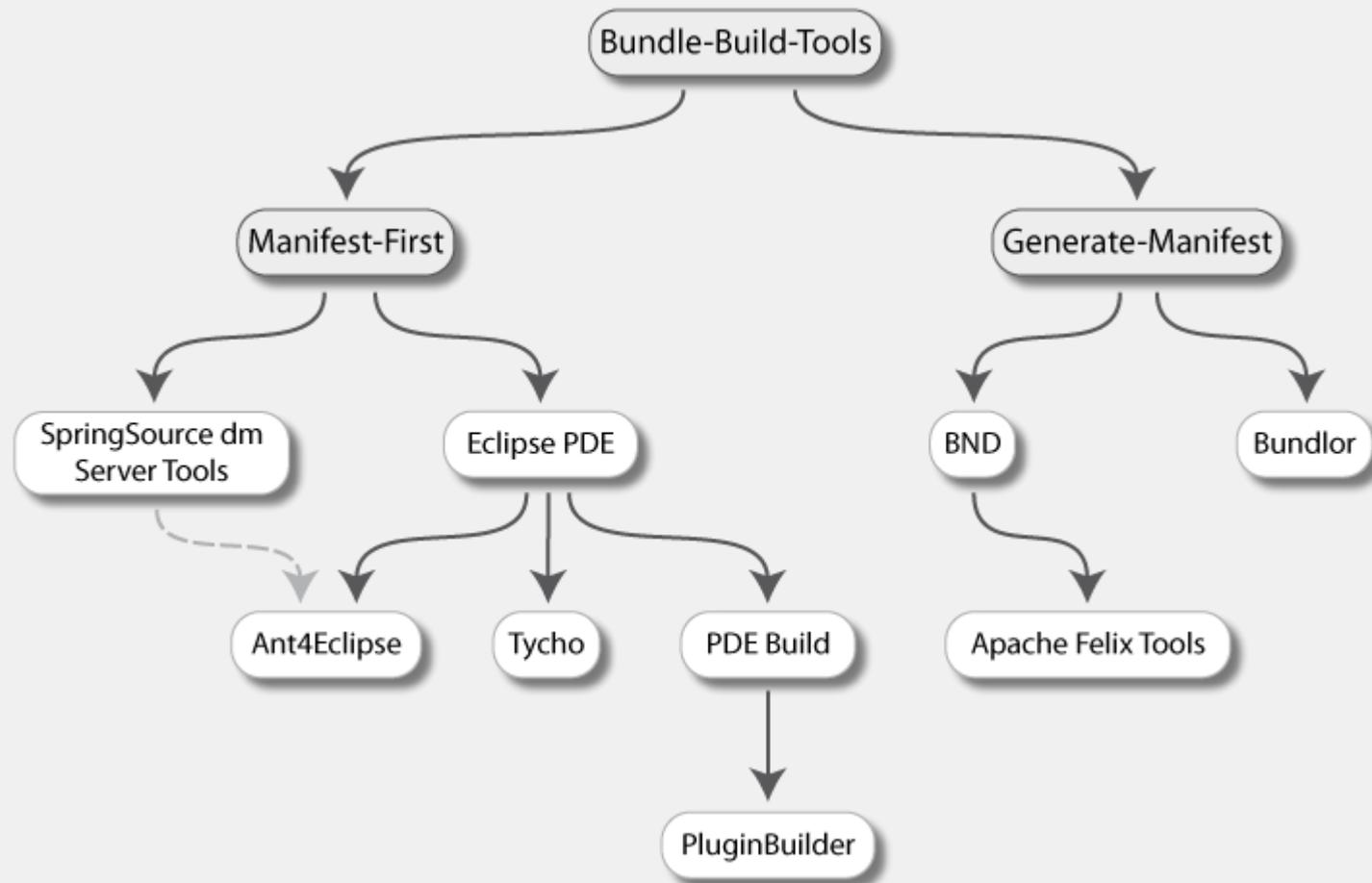
- » Viele Artefakte sind (noch) keine Bundles
- » Einbindung bestehender Bibliotheken
 - » Bundle-Manifest muss erstellt werden

Weitere Anforderung: Einbindung von Repositories

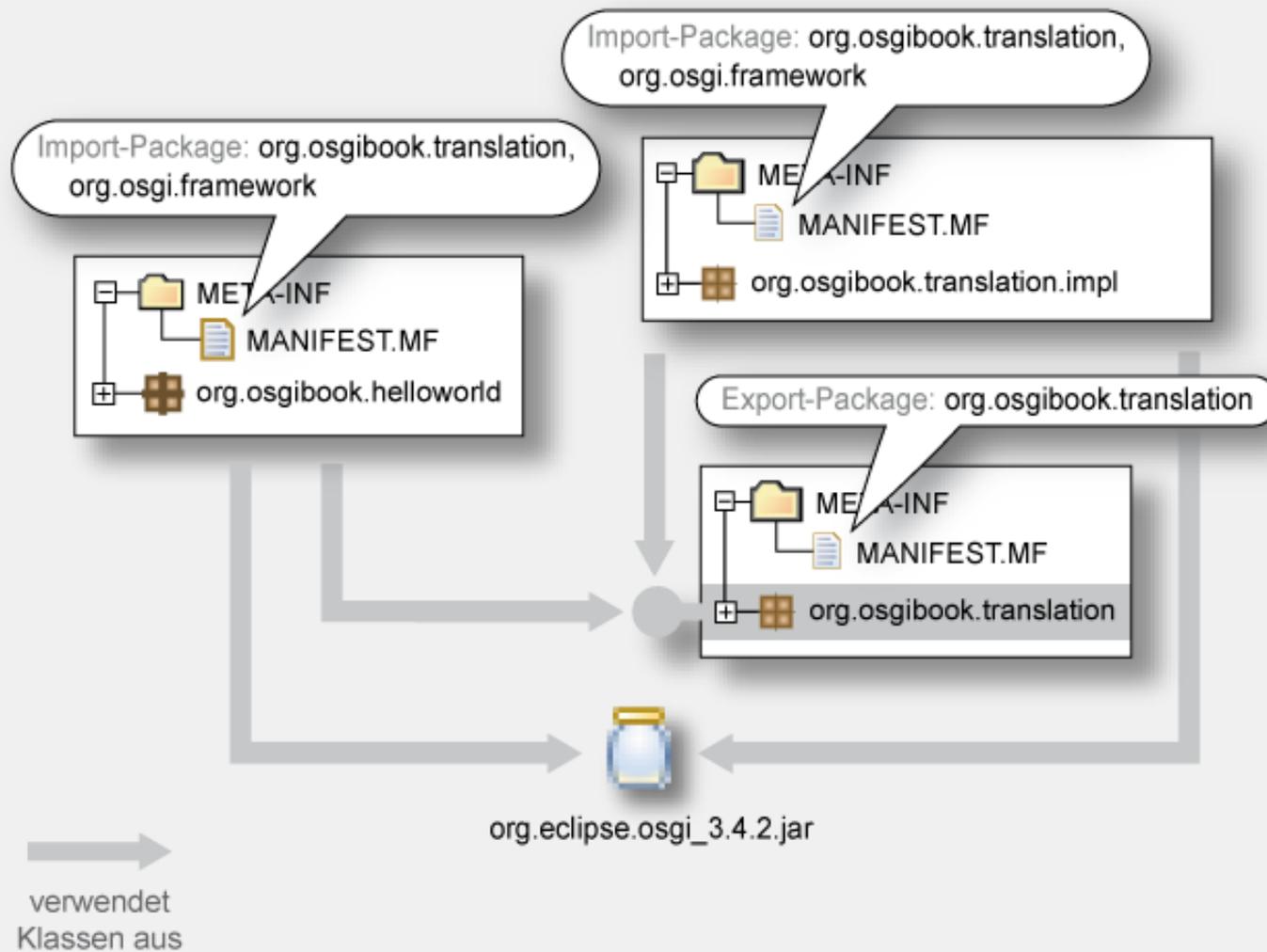
- » Maven Repository
- » OBR
- » EBR
- » Equinox P2

- » (Eclipse Target Platform)

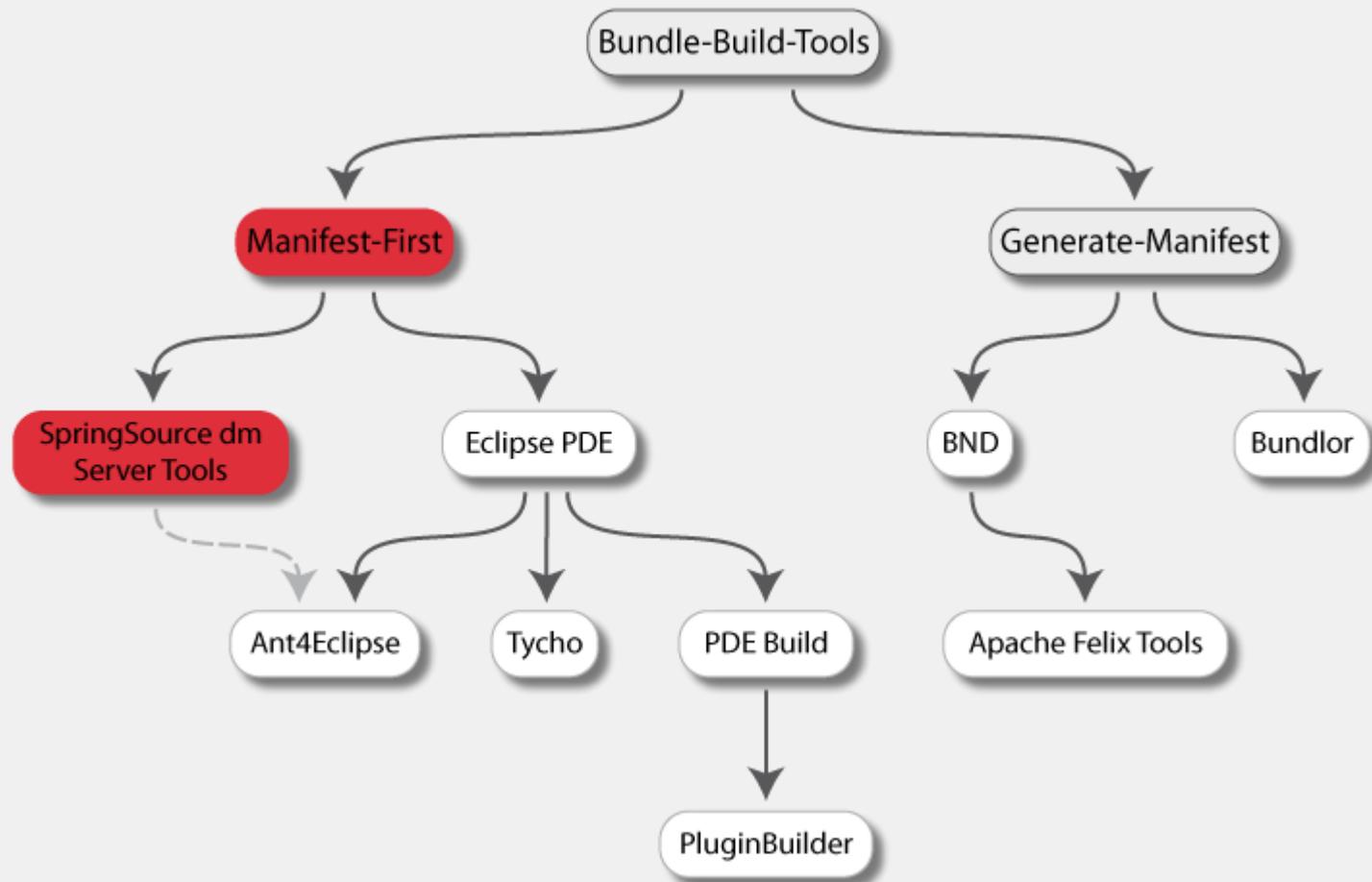
Toolübersicht



Praxis: „Hello World!“



Tools: DM Server Tools



Spring dm Server Tools

The screenshot displays the Eclipse IDE interface with the following components:

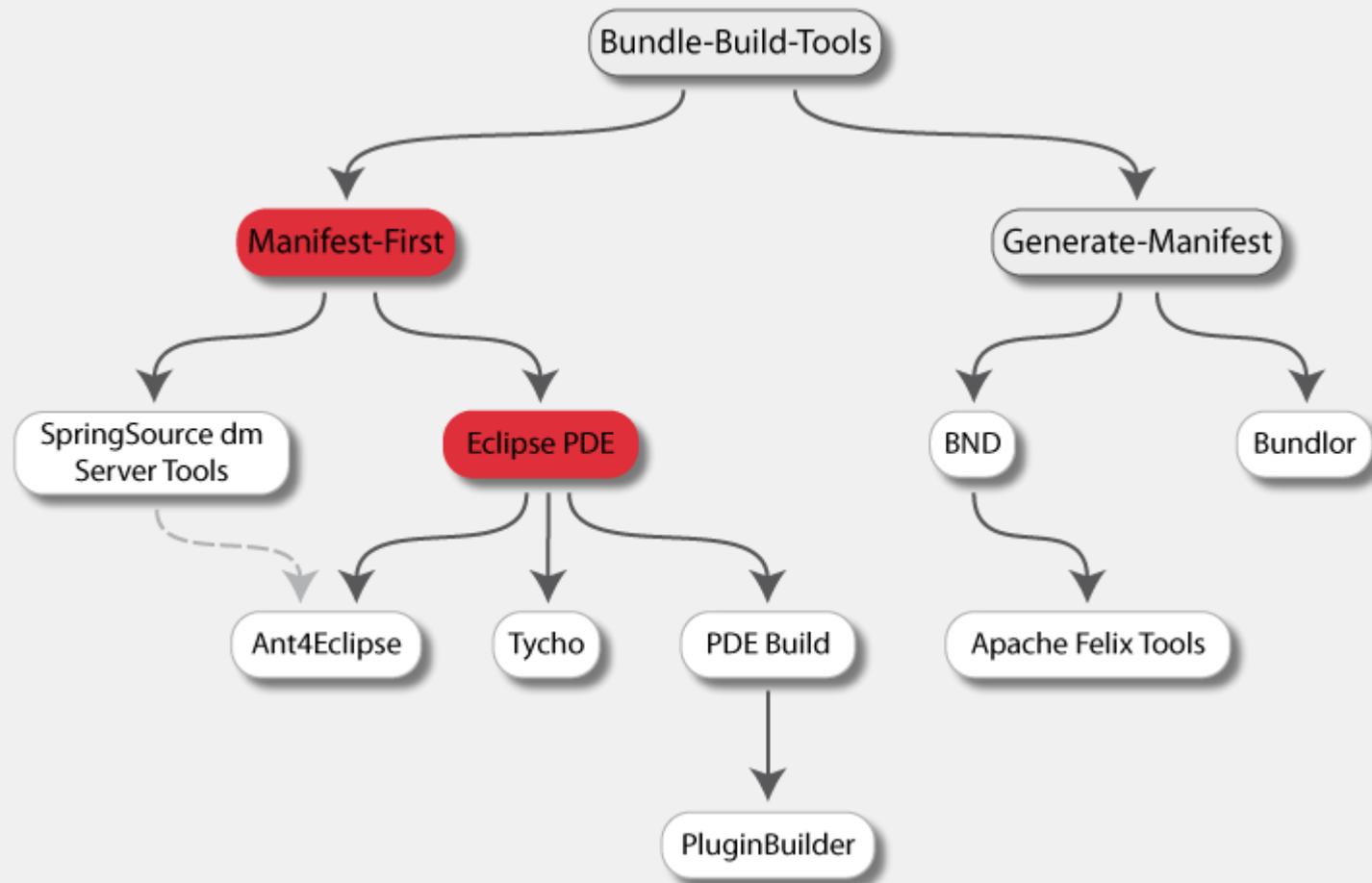
- Project Explorer:** Shows a project structure for `org.osgibook.translation.impl` with sub-projects like `org.osgibook.helloworld`, `org.osgibook.translation`, and `org.osgibook.translation.impl`. It also shows the `SpringSource dm Server v1.0 at localhost` server configuration.
- Dependencies:** A panel for managing dependencies. It includes sections for **Import Package** (listing `org.osgi.framework` and `org.osgibook.translation`) and **Import Bundle** (empty). Buttons for `Add...` and `Download...` are present.
- Bundle Repository Browser:** A panel for searching and installing bundles. It features a search bar with the text `*hibernate*`, a list of bundles, and buttons for `Search`, `Select All`, `Deselect All`, `Analyse`, `View License`, and `Download`. A checkbox for `Download source jars from repository` is checked. The repository URL is `http://www.springsource.com/repository`.
- Servers:** A panel showing the `SpringSource dm Server v1.0 at localhost` server, which is currently `Stopped`. It lists the bundles installed on the server: `org.osgibook.helloworld`, `org.osgibook.translation`, and `org.osgibook.translation.impl`.

Spring dm Server Tools

- » Anbieter: SpringSource
 - » <http://www.springsource.org/dmservertools>
 - » Eclipse-basierte Entwicklungsplattform für SpringSource dm Server
- » Grundlage: SpringSource Bundle Projekte
- » Repository: EBR, lokale dm Server-Installation

- » Grundidee:
 - » Eigene ‚Bundle‘-Projekt-Nature
 - » Abhängige Bundles werden direkt aus EBR geladen
 - » Projekte werden direkt in einen dm Server installiert
 - » Export von Bundles aus der IDE

Tools: Eclipse PDE



Tools: PDE

The screenshot displays the Eclipse PDE interface for configuring a plug-in. The main window is titled "Plug-in Development - org.osgibook.translation.impl/META-INF/MANIFEST.MF - Eclipse SDK - R:\osgi_workshop\workspace".

Package Explorer: Shows the project structure for "org.osgibook.translation.impl", including "JRE System Library [J2SE-1.5]", "Plug-in Dependencies", "src", "META-INF", "MANIFEST.MF", "build.properties", and "build.xml".

Plug-in Dependencies: Shows the "Current PDE State Environment" with a tree view of dependencies, including "org.osgibook" and its sub-packages.

Overview: The main configuration panel for the plug-in, divided into several sections:

- General Information:** Describes general information about the plug-in. Fields include ID (org.osgibook.translation.impl), Version (1.0.0), Name (Translation Implementation Bundle), Provider, Platform Filter, and Activator (org.osgibook.translation.impl.Activat...).
- Plug-in Content:** Describes the content of the plug-in, including Dependencies and Runtime.
- Extension / Extension Point Content:** Describes the plug-in's extensions and extension points, including Extensions and Extension Points.
- Testing:** Provides options to test the plug-in by launching an OSGi framework, including "Launch the framework" and "Launch the framework in Debug mode".
- Exporting:** Provides options to package and export the plug-in.

Execution Environments: A list of execution environments required to run the plug-in, currently showing "J2SE-1.5".

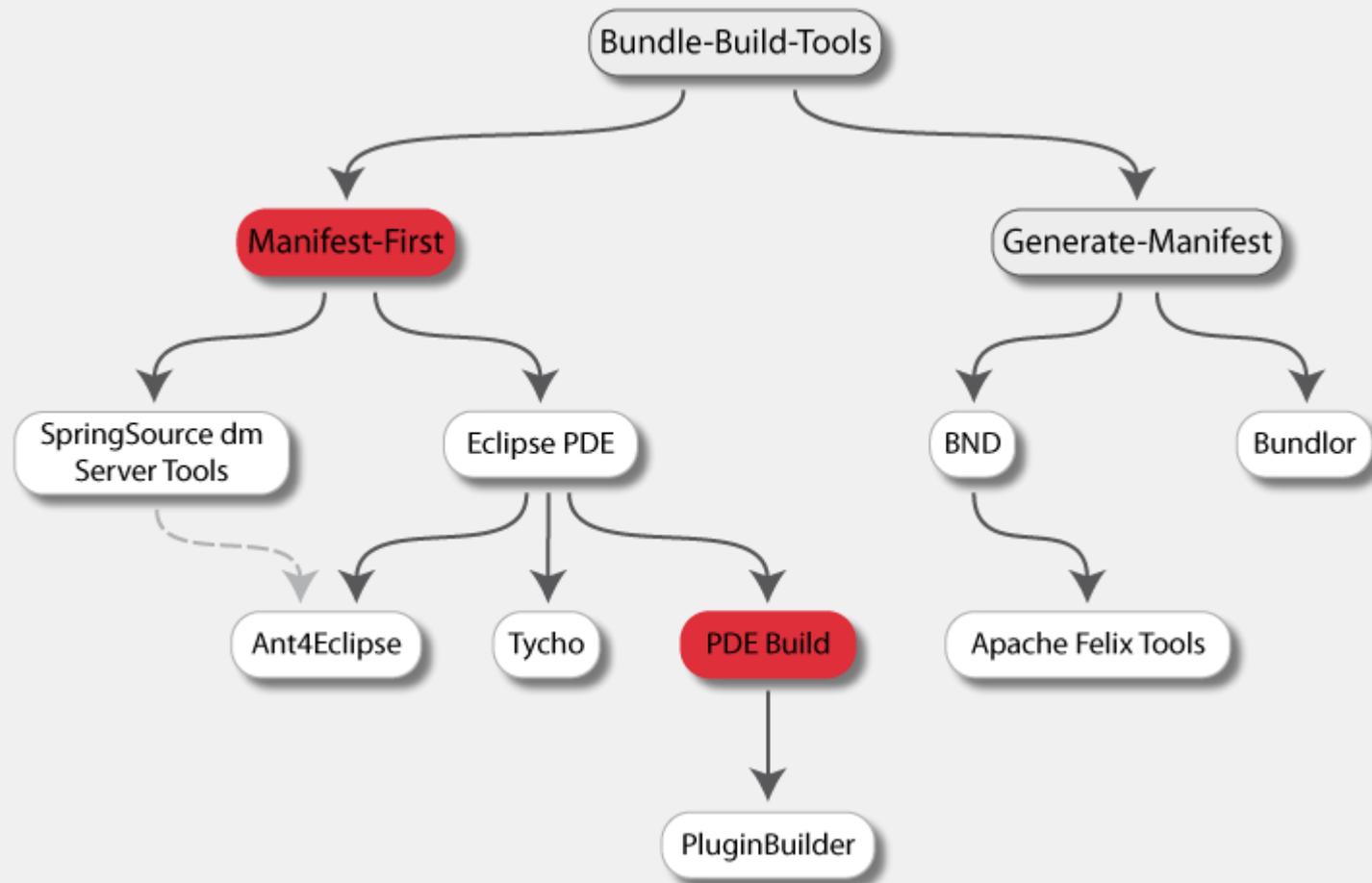
Plug-in Registry: A search results view showing a filter matched 52 of 169 plug-ins, with a tree view of results including "org.eclipse.ui" and its prerequisites.

Eclipse PDE

- » Bestandteil der Eclipse „Classic“ Distribution
 - » Eclipse-basierte Entwicklungsplattform für Plug-ins/Bundles
- » Grundlage: Eclipse Plug-in Projekte
- » Repository: Target Platform

- » Grundidee:
 - » Grafische Wizards und Editoren für OSGi-Artefakte
 - » Code-Completion und Syntax-Check für Manifest-Dateien
 - » Plug-in-Projekte lassen sich direkt in Eclipse Equinox ausführen
 - » Export-Wizard für Bundles/Plug-ins

Tools: PDE Build



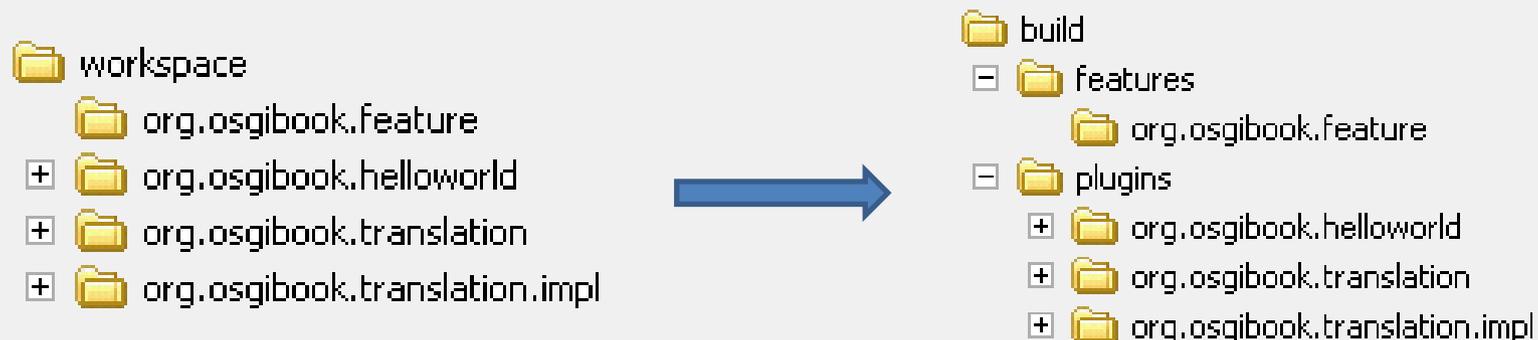
PDE-Build

- » Kategorie: Manifest-driven
- » Anbieter: Eclipse
 - » Bestandteil von Eclipse
 - » Die ‚offizielle‘ Eclipse Lösung für Plug-ins, Features, Update Sites, ...
- » Grundlage: Eclipse (Plug-in-)Projekte
- » Repository: Target Platform

- » Grundidee:
 - » Generiert Ant-Scripte, die mit Eclipse „antRunner“ ausgeführt werden
 - » Verwendung der Eclipse/OSGi-Infrastruktur in Ant-Tasks
 - » Projekt-Abhängigkeiten werden analog zur IDE aufgelöst
 - » Package-sichtbarkeiten werden OSGi-konform eingehalten

PDE-Build: Konfiguration

- » Bundles müssen in einem „Feature“ zusammengefasst sein
- » Bundles und Plug-ins in getrennten Ordnern



» Konfigurationsdateien

- » build.properties: Verzeichnisse, Java-Versionen, ...
- » customTargets.xml: „Callbacks“, um den Build zu modifizieren (opt.)
- » map-File: Source-Code-Repositories (opt.)
- » Vorlagen im Plug-in „org.eclipse.pde.build_<version>“

PDE-Build: build.properties

Erforderliche Angaben:

- » Id des Top-Level-Elements (Feature)
 - » `topLevelElementId=org.osgibook.feature`
- » Eclipse-Installation, gegen die gebaut werden soll
 - » `base=t:/Software/eclipse-SDK-3.4.2-win32`
- » Verzeichnis, in dem sich die Source-Projekte befinden
 - » `buildDirectory=t:/pde/build`
- » Compiler-Version
 - » `javacSource=1.5`
 - » `javacTarget=1.5`
- » (optional) Zugriff auf Versionsverwaltung
 - » `mapsRepo=svn://localhost/jax/pde/`
 - » `mapsRoot=org.osgibook.maps`
 - » `mapsCheckoutTag=trunk`

PDE-Build: customTargets.xml

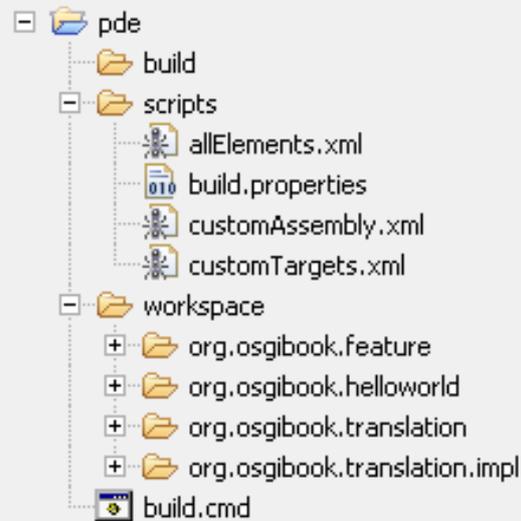
- » Callback-Targets, um Standard-Build-Prozess zu modifizieren
 - » Vor und nach Auschecken, Compilieren, Build-File-Generierung, ...

```
<!-- ===== -->
<!-- Steps to do before setup -->
<!-- ===== -->
<target name="preSetup">
    <!-- Copy files from workspace to build directory -->

    <copy todir="${buildDirectory}/plugins">
        <fileset dir="${workspaceDirectory}" includes="**/*"
            excludes="org.osgibook.feature/**"/>
    </copy>

    <copy todir="${buildDirectory}/features">
        <fileset dir="${workspaceDirectory}"
            includes="org.osgibook.feature/**"/>
    </copy>
</target>
```

PDE-Build: Demonstration



» Drei Bundles und ein Feature-Projekt

» Aufruf:

```
java -jar
```

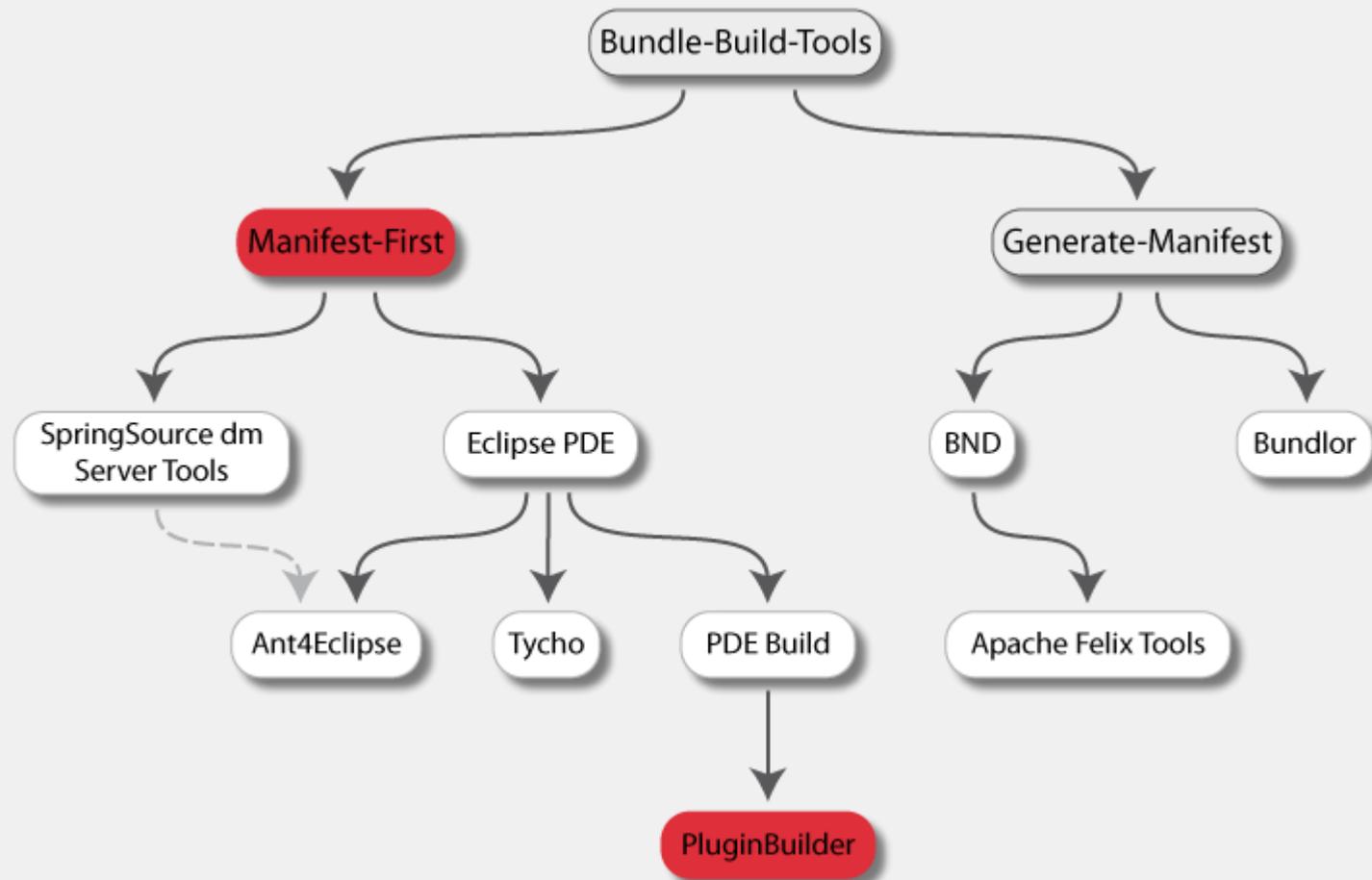
```
%ECLIPSE%\plugins\org.eclipse.equinox.launcher_<version>.jar
```

```
-application org.eclipse.ant.core.antRunner
```

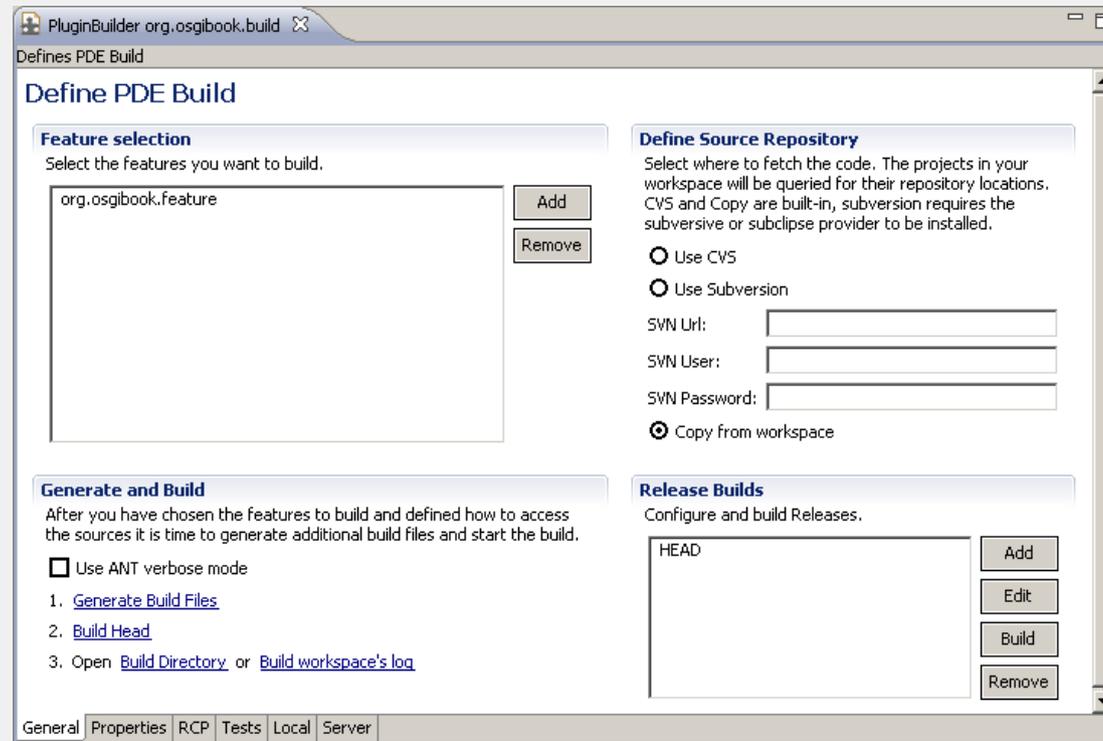
```
-f %ECLIPSE%\plugins\org.eclipse.pde.build_<version>\scripts\build.xml
```

```
-Dbuilder=t:\pde\scripts
```

Tools: Plug-in-Builder

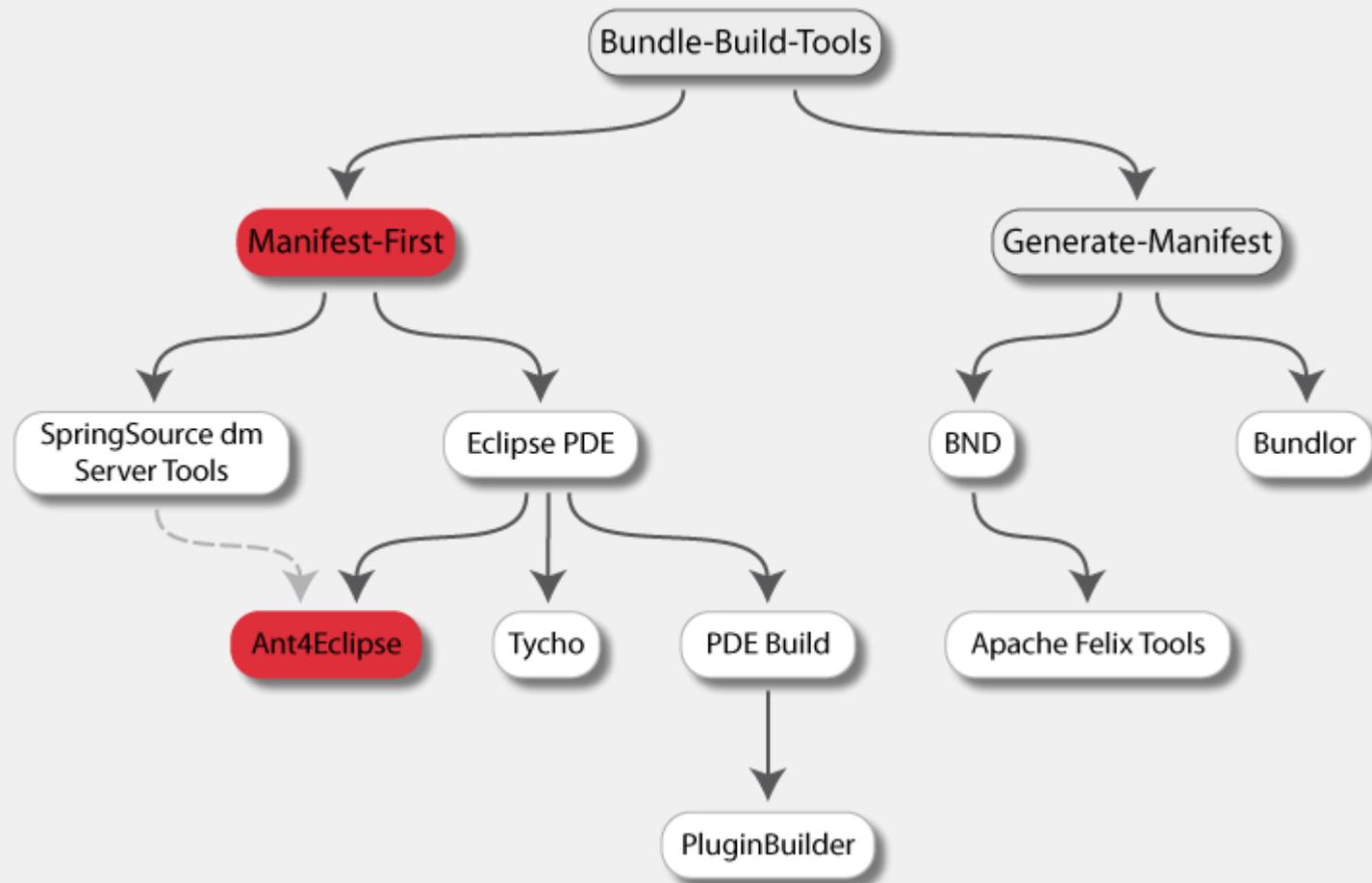


Plugin-Builder



- » <http://www.pluginbuilder.org>
 - » Open-Source-Projekt (EPL)
- » Grundidee:
 - » Grafische Wizards und Editoren zur Konfiguration des PDE-Builds
 - » Erlaubt Ausführung von Tests nach dem Build in eigener Eclipse-Instanz

Tools: Ant4Eclipse



Ant4Eclipse

- » Kategorie: Manifest-driven
- » Anbieter: wir 😊
 - » Open-Source-Projekt (<http://www.ant4eclipse.org>)
 - » Ant-Tasks zum Auslesen von Eclipse Projekt-Informationen
 - » Vorgestellte Version noch in Arbeit
- » Grundlage: Eclipse Plug-in-Projekte
- » Repository: Target-Plattform
- » Grundidee:
 - » ‚Reine‘ Ant-Lösung, kein Starten der Eclipse Infrastruktur
 - » (Low-Level-)Tasks arbeiten direkt auf Eclipse Projekt-Artefakten
 - » Fertige Ant-Makros für komplexe Aufgaben (buildPlugin, ...)
 - » Verwendung der Eclipse API zum Auflösen der OSGi-Abhängigkeiten

Ant4Eclipse: Installation

» Zusätzliche Bibliotheken:

- » Eclipse Compiler (`ejc-3.4.jar`)
- » Equinox OSGi Implementierung (`org.eclipse.osgi._<version>.jar`)
- » Ant-contrib (`ant-contrib.jar`)

» ant4eclipse-Namespace definieren:

```
<project name="..." xmlns:ant4eclipse="antlib:org.ant4eclipse">  
</project>
```

» Importieren der PDE-Macros:

```
<import file="ant4eclipse/pde-macros.xml" />
```

» Starten von Ant mit Verweis auf lib-Verzeichnis:

```
ant.bat -lib <ant4eclipse-dir>/libs
```

Ant4Eclipse: JRE und TargetPlatform

» JRE-Container definieren

- » Gibt an, gegen welche JREs gebaut werden soll

```
<ant4eclipse:jreContainer>  
    <jre id="jdk15" location="t:/software/jdk_15" />  
    <jre id="jdk16" location="t:/software/jdk_16" />  
</ant4eclipse:jreContainer>
```

» TargetPlatform spezifizieren

- » Definiert, gegen welche Bundles gebaut werden soll

```
<ant4eclipse:targetPlatform id="target.platform">  
    <location dir="t:/target.platform/equinox-3.4/plugins" />  
    <location dir="t:/target.platform/ibr" />  
    <location dir="t:/target.platform/codehaus" />  
    <location dir="t:/target.platform/javakontor" />  
</ant4eclipse:targetPlatform>
```

Ant4Eclipse: buildPlugin-Makro

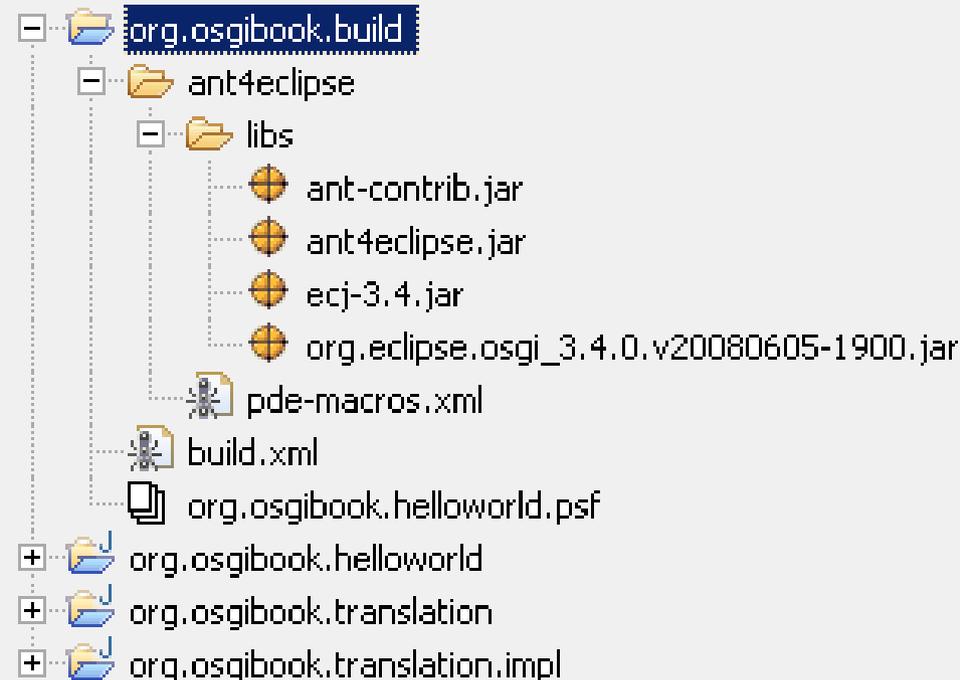
» Einzelnes Bundle bauen:

```
<target name="build">  
    <!-- Bundle bauen -->  
    <buildPlugin workspace="t:/workspace"  
        projectname="org.osgibook.translationervice"  
        targetPlatformId="target.platform"  
        destination="${destination}" />  
</target>
```

» Tasks zum iterieren über diverse Projekt-Artefakte vorhanden

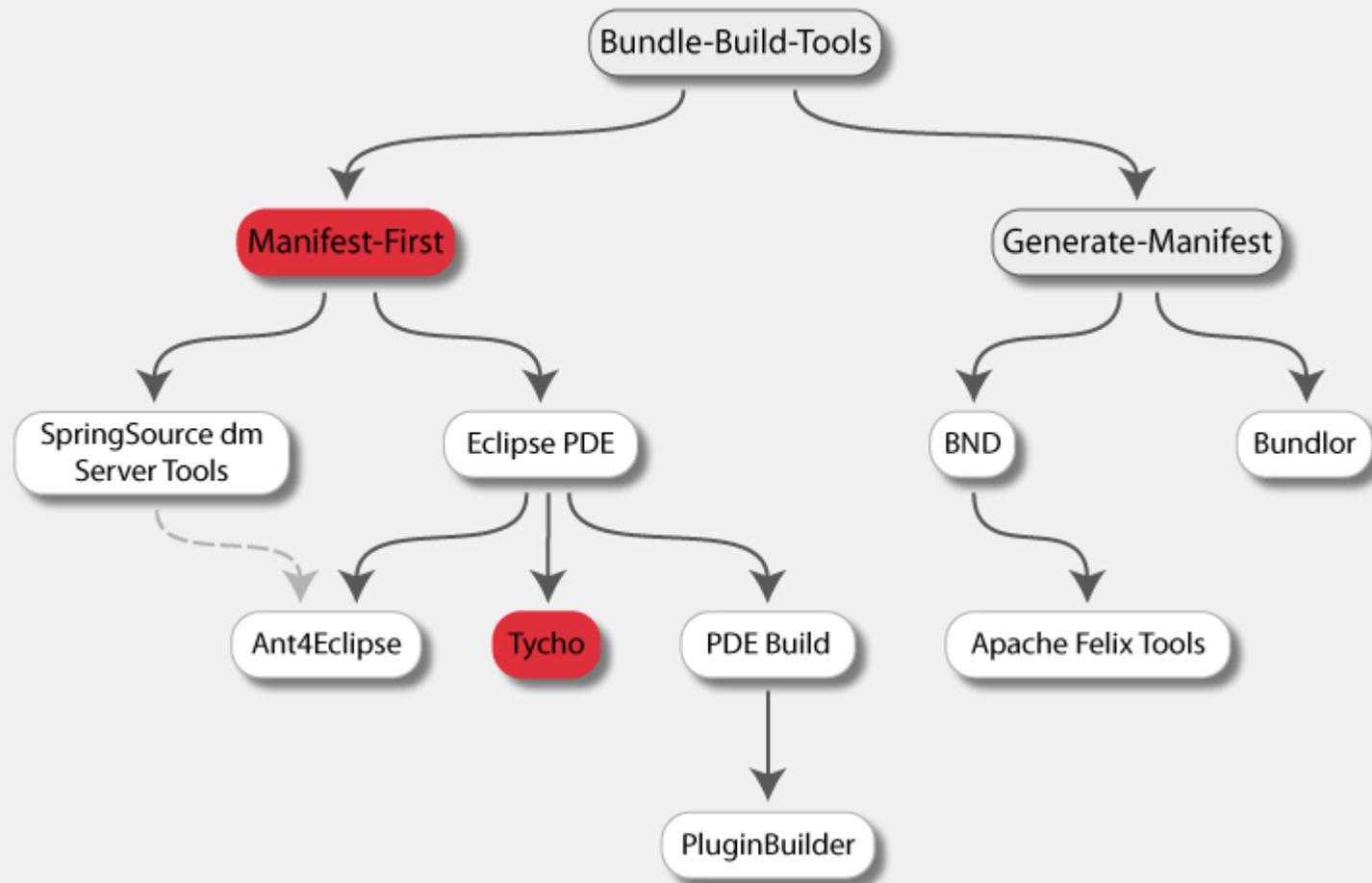
» `forEachProject`, `forEachSourceDirectory`, `forEachOutputDirectory`, ...

Ant4Eclipse: Demonstration



- » Drei Plug-in-Projekte liegen im Workspace vor
- » Team-Project-Set-Datei enthält Namen der Projekte
 - » Build-Reihenfolge wird ermittelt
 - » buildPlugin wird für jedes Projekt aufgerufen

Tools: Tycho



Tycho Maven Plug-in

- » Kategorie: Manifest-driven
- » Anbieter: Sonatype (Open Source)
 - » Teil von M2Eclipse (Maven Integration for Eclipse)
 - » *<http://docs.codehaus.org/display/M2ECLIPSE/Tycho+project+overview>*
- » Grundlage: Maven Projekte
- » Repository: Target Platform (0.3), P2, Maven (ab 0.4 - in Arbeit)
- » Grundidee:
 - » Generiert aus Eclipse Plug-in Projekten POMs
 - » Abhängigkeiten im Manifest werden aus Repository geladen ab (0.4)
 - » Eclipse Compiler für OSGi-Sichtbarkeiten
 - » Bundles werden in Maven Repository deployed

Tycho Maven Plug-in: Verwendung

» Erfordert Maven 3.0!

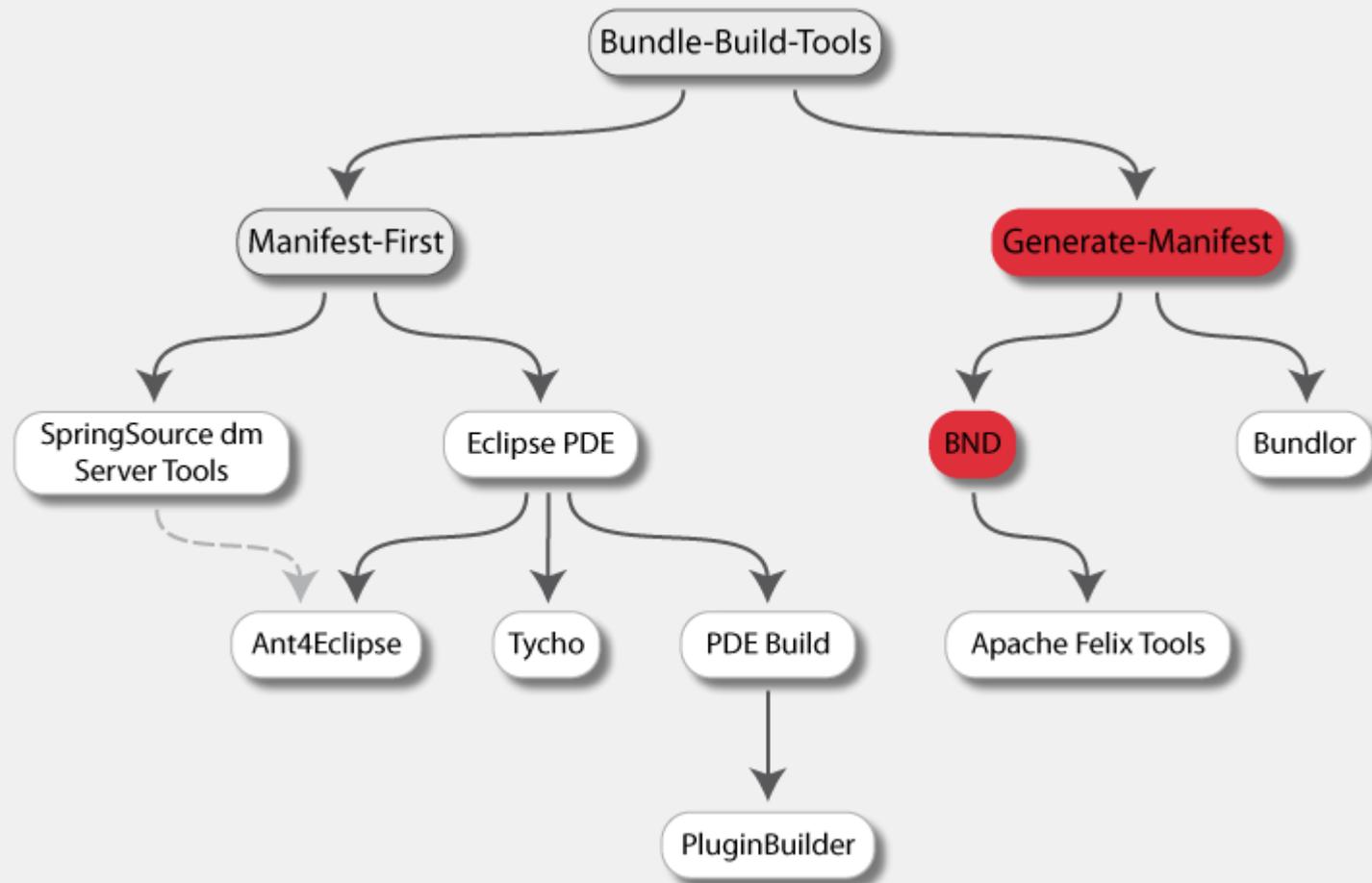
» POMs generieren:

```
mvn org.codehaus.tycho:maven-tycho-plugin:generate-poms  
-DgroupId=org.osgibook  
-Dtycho.targetPlatform=T:\Software\Eclipse-3.4.2
```

» Bundles erzeugen und deployen:

```
mvn install  
-Dtycho.targetPlatform=T:\Software\Eclipse-3.4.2
```

Tools: BND

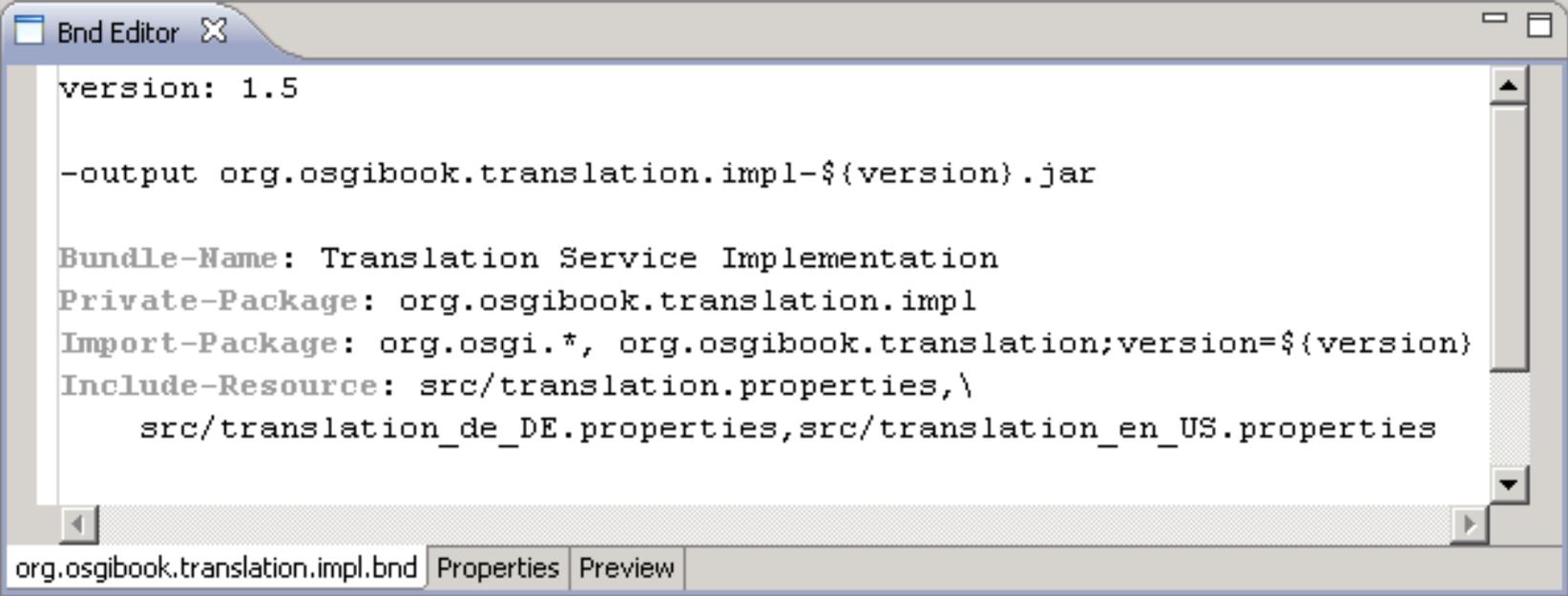


bnd

- » Kategorie: Generate-Manifest
- » Anbieter: Peter Kriens (aQute, OSGi)
 - » <http://www.aqute.biz/Code/Bnd>
 - » Verwendbar in Kommandozeile, Ant, Maven, Eclipse Plug-in
- » Grundlage: Klassen, bnd-Konfigurationsdatei
- » Grundidee:
 - » Generiert Bundles aus einer Menge von Klassen
 - » Erstellt Manifest für bestehende nicht-OSGi JAR-Files
 - » Abhängigkeiten werden durch Class-File-Analyse ermittelt
 - » Manifest kann mittels .bnd-Datei beeinflusst werden
 - » Erzeugt Konfiguration für Declarative Services

bnd: bnd-Konfigurationsdatei

- » Manifest-ähnliche Konfigurationsdatei
 - » Manifest-Header, BND-Direktiven, Variablen
 - » Insb. `Export-Package`, `Private-Package`, `Include-Resource`
- » Header dürfen Wildcards enthalten
- » `Private-Package-Header` verhindert den Export eines Packages



The screenshot shows a window titled "Bnd Editor" with a close button. The main area contains the following text:

```
version: 1.5

-output org.osgibook.translation.impl-${version}.jar

Bundle-Name: Translation Service Implementation
Private-Package: org.osgibook.translation.impl
Import-Package: org.osgi.*, org.osgibook.translation;version=${version}
Include-Resource: src/translation.properties,\
    src/translation_de_DE.properties,src/translation_en_US.properties
```

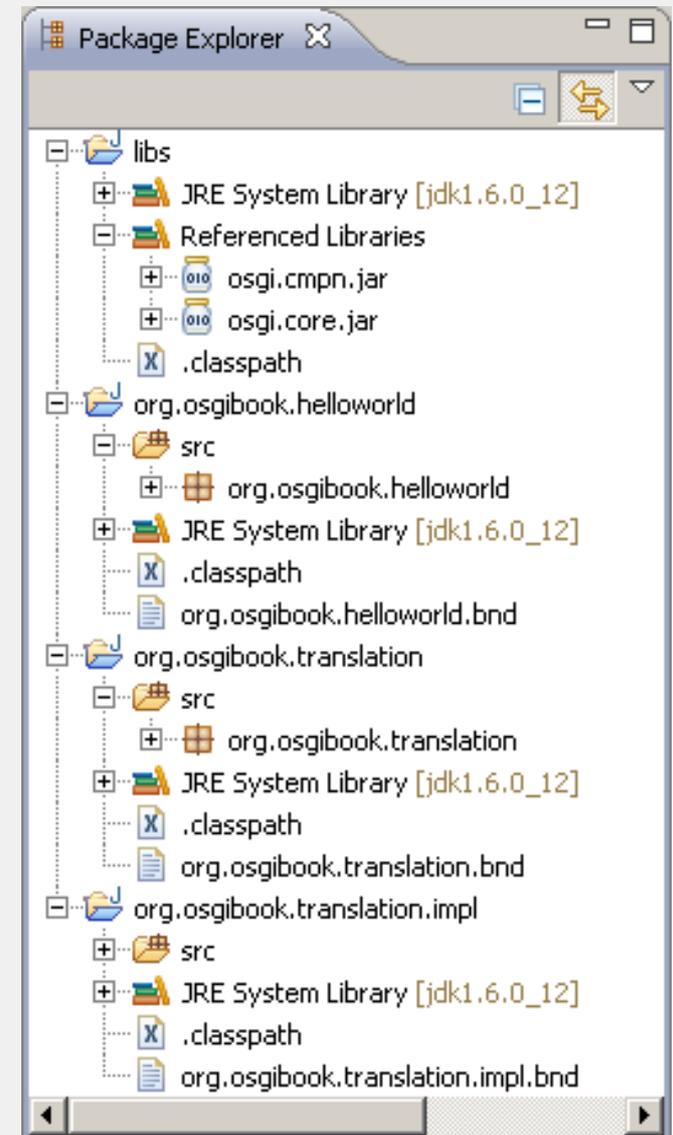
At the bottom, there are three tabs: "org.osgibook.translation.impl.bnd", "Properties", and "Preview".

bnd: Eclipse Plug-in

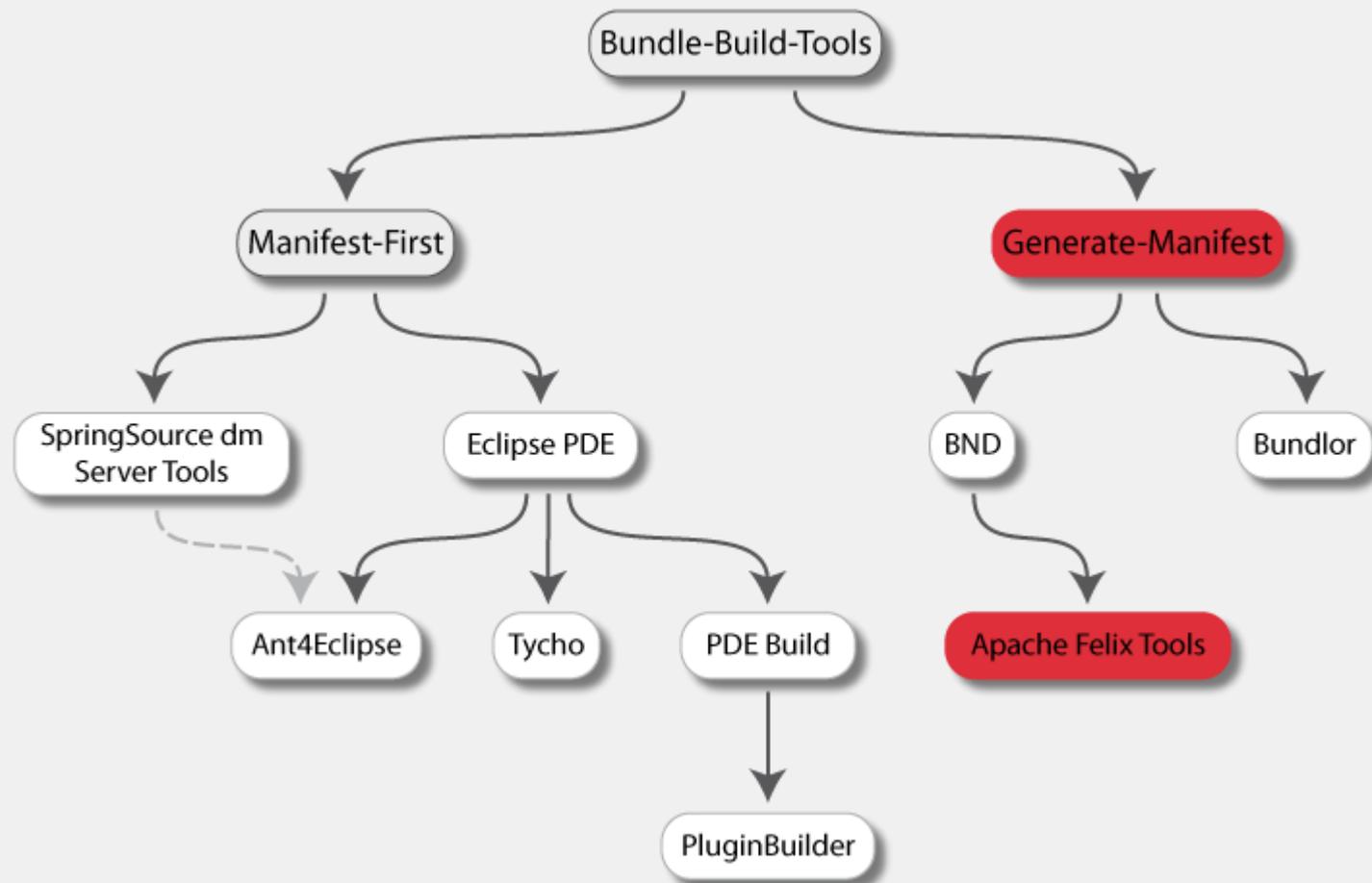
- » Kontext-Menüs
 - » Erstellen von Bundles
 - » Wrappen von JARs
- » Eclipse Klassenpfad wird verwendet
- » Editor für .bnd-Dateien
- » View für Bundle-JARs
 - » Zeigt Inhalt und Manifest eines JARs

bnd: Demonstration

- » Bundles as Java Projekte in Eclipse
 - » .bnd-Datei für jedes Projekt
 - » OSGi API als Jar im Klassenpfad
- » bnd-Aufruf
 - » Kontext-Menü



Tools: Apache Felix Tools



maven-bundle-plugin

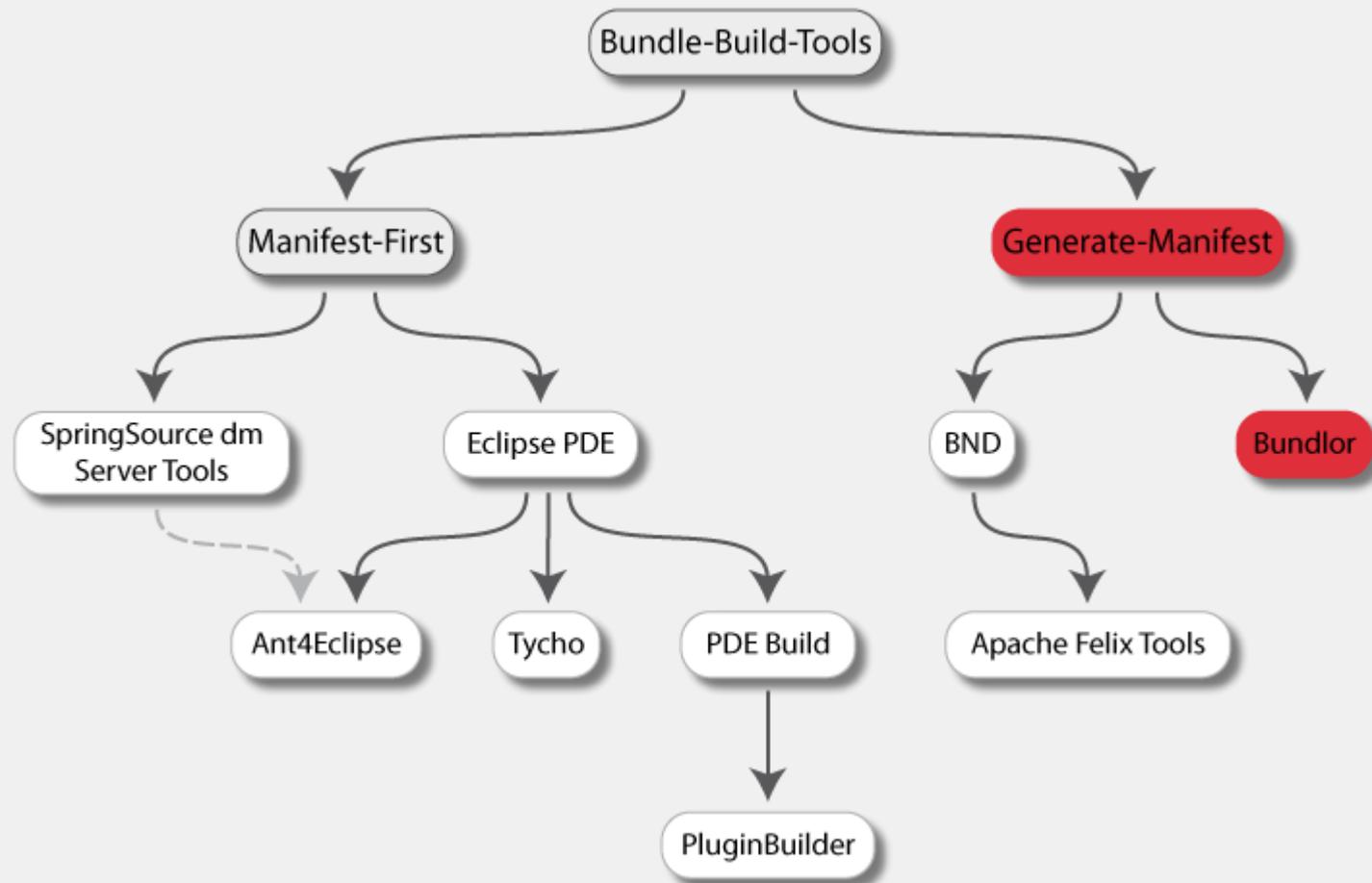
- » Kategorie: Generate-Manifest
- » Anbieter: Apache Foundation
 - » Bestandteil von Apache Felix
 - » <http://felix.apache.org/site/apache-felix-maven-bundle-plugin-bnd.html>
 - » Ersetzt „alte“ Felix Maven-Plug-ins
- » Grundlage: Maven Projekte
- » Repository: Maven

- » Grundidee:
 - » Maven-Wrapper um bnd-Tool
 - » Konfiguration von bnd wird direkt im POM vorgenommen
 - » Definiert Maven-Packaging Type „bundle“
 - » Deployed Bundles in Maven-Repository und OBR

maven-bundle-plugin: Verwendung

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <name>Translation Service Default Implementation</name>
  <groupId>org.osgibook</groupId>
  <artifactId>org.osgibook.translation.impl</artifactId>
  <version>1.0.0</version>
  <packaging>bundle</packaging>
  <dependencies>
    <dependency>
      <groupId>org.osgibook</groupId>
      <artifactId>org.osgibook.translation</artifactId>
      <version>1.5.0</version>
    </dependency>
    <dependency>
      <groupId>org.apache.felix</groupId>
      <artifactId>org.osgi.core</artifactId>
      <version>1.0.0</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <extensions>true</extensions>
        <configuration>
          <instructions>
            <Bundle-SymbolicName>${pom.artifactId}</Bundle-SymbolicName>
            <Export-Package>org.osgibook.translation</Export-Package>
          </instructions>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

Tools: SpringSource Bundlor



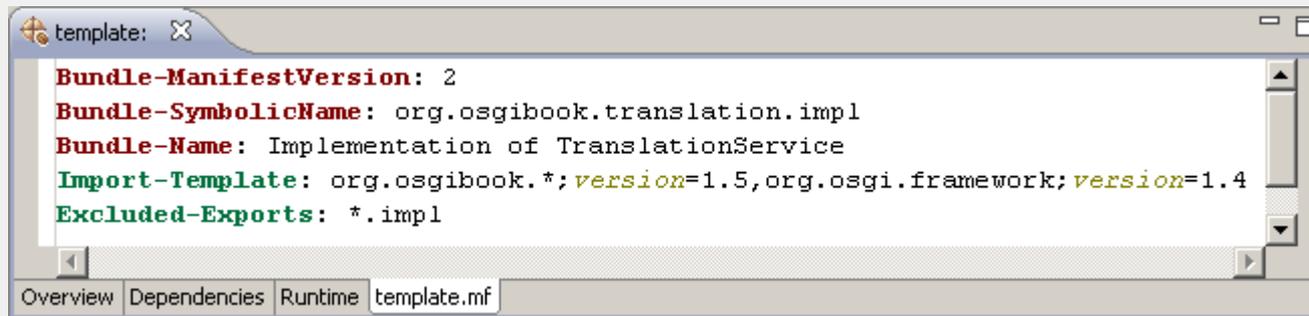
bundler

- » Kategorie: Generate-Manifest-Ansatz
- » Anbieter: SpringSource (APL 2.0)
 - » <http://www.springframework.org/bundler>
 - » Kommandozeile, Ant, Maven
- » Grundlage: ‚reguläre‘ JAR-Datei, Manifest-Template
- » Repository: Maven, Ivy

- » Grundidee:
 - » Ermittelt die Abhängigkeiten einer JAR-Datei (Klassen, Spring, JPA, etc)
 - » Generiert eine Manifest-Datei oder ganze Bundle JAR
 - » Manifest-Erzeugung kann mit „Template“ gesteuert werden

- » Inkrementeller Manifest-Builder für Eclipse (dm Server Tools)

bundlor: Verwendung



» Manifest-Template

- » ‚Echte‘ Manifest-Header werden ins Manifest übernommen
- » ‚Template‘-Header erlauben Wildcards zur Konfiguration

Export-Template: org.osgibook.*;version=1.5

Import-Template: org.log4j.*;version=1.3;resolution:=„optional“

- » Header werden mit ggf. bestehenden Manifest-Headern gemerged

» Aufruf

```
bundlor transform -b jars\org.osgibook.translation.jar
-m jars\org.osgibook.translation-template.mf
-o output\org.osgibook.translation.jar
```

Danke! Fragen?

