FORMATION JAVA FRAMWORK

Industrialisation d'une application Web (Maven)

Plan

- Le POM (Project Object Model)
- Présentation du modèle POM maven et notion de coordonnées
- Détails et sections du fichier pom.xml de maven
- Le « super POM » et les mécanismes d'héritage de maven
- Exploration de la structure des projets Maven
- Les types de projets Maven
- Notion de propriétés et de filtre des ressources dans maven
- **Repository Maven et coordinations**
- Repository local de maven : .m2/repository
- Mécanisme de localisation d'une librairie dans maven Notion de repositories distants
- Outillage pour la gestion du cache et de la sécurité avec Nexus
- Mise en place de Maven

Qu’est ce que MAVEN ?

- Pour développeur/concepteur : outil de build
- Pour Chef de projet : outil de gestion de projet
- Maven est un outil de gestion projet caractérisé par
  - le modèle POM (Project Object Model)
  - Un ensemble de standards
  - Un cycle de vie de projet
  - Un système de gestion des dépendences
  - Une stratégie d'exécution d'objectifs (goals) standard à chaque phase d'un projet
  - Un ensemble de plugins implémentant des services

Installation

- Télécharger la distribution binaire
  - http://maven.apache.org
- Dézipper dans un répertoire
- Positionner les variables d'environnement
  - set JAVA_HOME=c:\j2sdk1.x.y
  - set MAVEN_HOME=c:\maven-3.z.w
  - set PATH=%JAVA_HOME%\bin;%MAVEN_HOME%\bin
  - mvn –version
  - mvn --help
- (éventuellement) Configurer ~/.m2/settings.xml
  - repositories, plugins repositories, proxies, …
- Intégration à votre IDE (Eclipse, NetBeans, IDEA, …)
  - http://m2eclipse.codehaus.org/, http://mevenide.codehaus.org…

mvn --help

usage: mvn [options] [<goal(s)>] [<phase(s)>]

Options:

-q,--quiet Quiet output - only show errors

-C,--strict-checksums Fail the build if checksums don't match

-c,--lax-checksums Warn if checksums don't match

-P,--activate-profiles Comma-delimited list of profiles to activate

-ff,--fail-fast Stop at first failure in reactorized builds

-fae,--fail-at-end Only fail the build afterwards; allow all non-impacted builds to continue

-B,--batch-mode Run in non-interactive (batch) mode

-fn,--fail-never NEVER fail the build, regardless of project result

-up,--update-plugins Synonym for cpu

-N,--non-recursive Do not recurse into sub-projects

-npr,--no-plugin-registry Don't use ~/.m2/plugin-registry.xml for plugin versions

-U,--update-snapshots Forces a check for updated releases and snapshots on remote repositories

-cpu,--check-plugin-updates Force upToDate check for any relevantregistered plugins

-npu,--no-plugin-updates Suppress upToDate check for any relevant registered plugins

-D,--define Define a system property

-X,--debug Produce execution debug output

-e,--errors Produce execution error messages

-f,--file Force the use of an alternate POM file.

-h,--help Display help information

-o,--offline Work offline

-r,--reactor Execute goals for project found in the reactor

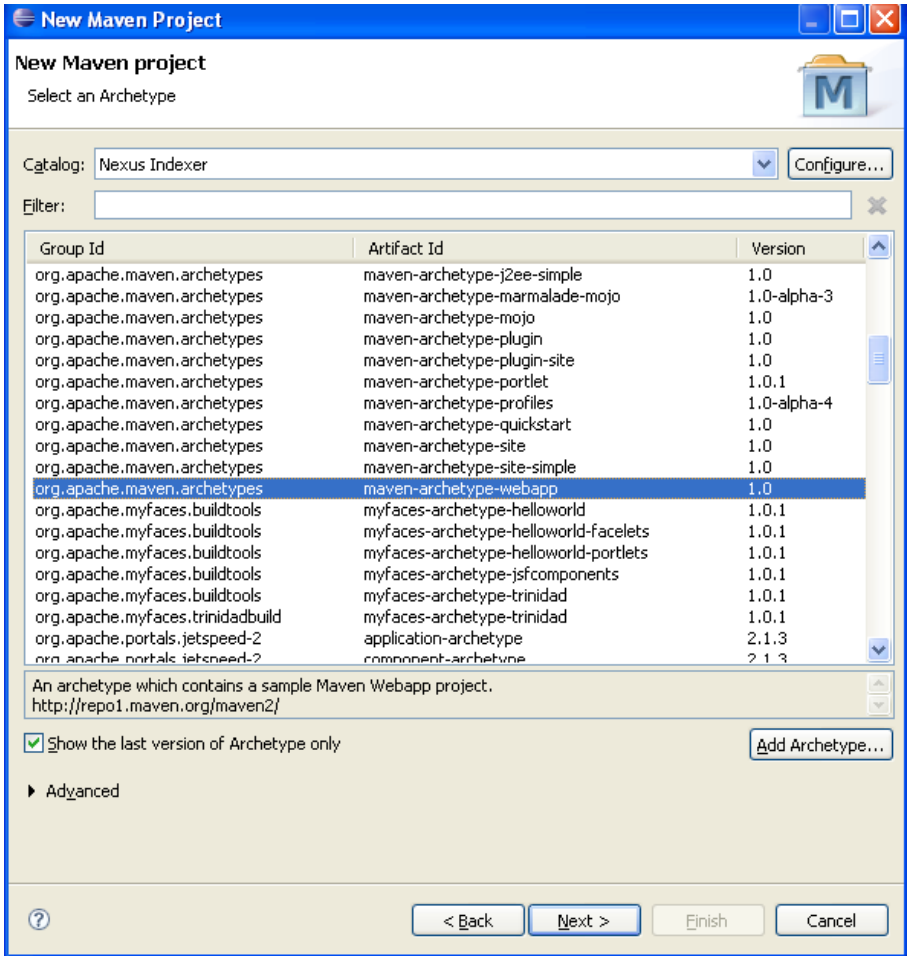
-s,--settings Alternate path for the user settings file

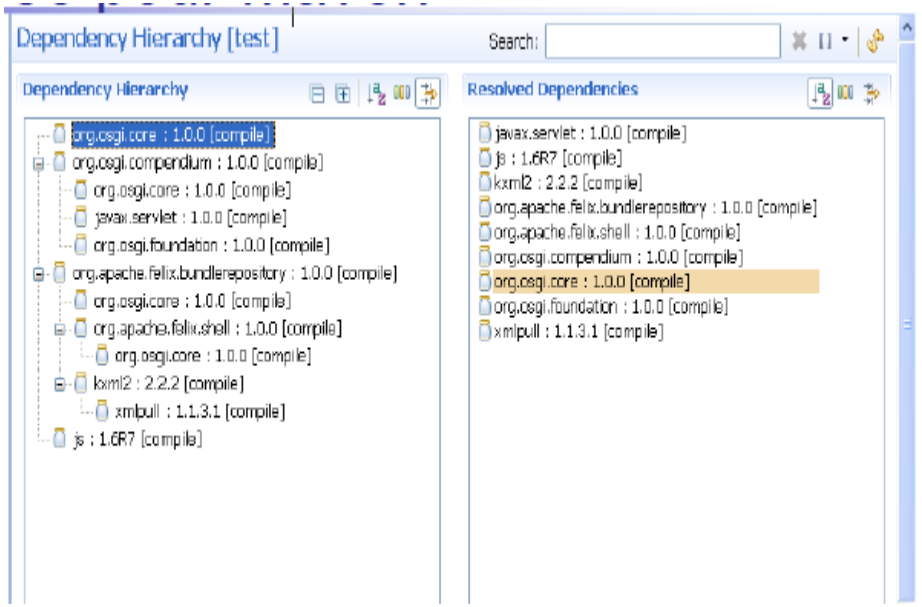
-v,--version Display version information

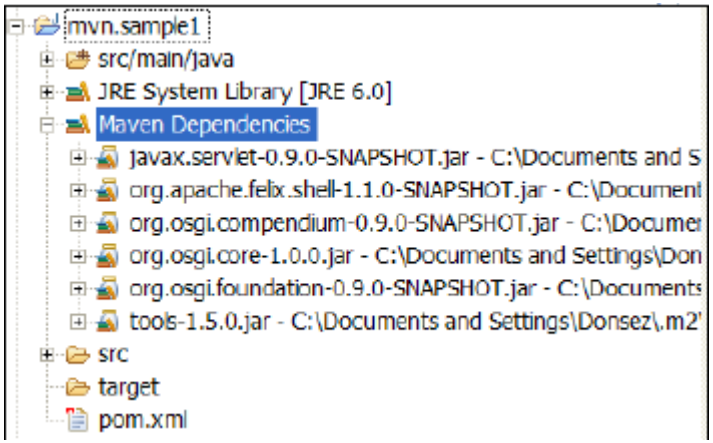
M2Eclipse
Plugin Eclipse pour Maven

- Création de projets
  - Wizard, Archetypes
- Edition du POM
- Affichage graphique
- Recherche de dépendances
  - Depuis les dépôts local et distants
- Ajout des dépendances du POM au .classpath
  - org.maven.ide.eclipse.MAVEN2_CLASSPATH_CONTAINER
- Exécution des principales phases : clean, test, install, …
- Livre en ligne
  - http://www.sonatype.com/m2eclipse/documentation/download-book?file=books/m2eclipse-book.pdf

M2Eclipse
Plugin Eclipse pour Maven







Convention over Configuration

Item	Default
source code	${basedir}/src/main/java
Resources	${basedir}/src/main/resources
Tests	${basedir}/src/test
Complied byte code	${basedir}/target
distributable JAR	${basedir}/target/classes

```
simple/❶
simple/pom.xml❷
/src/
/src/main/❸
/main/java
/src/test/❹
/test/java
```

Le modèle de projet (POM pour Project Object Model)

Description d'un projet indépendante des actions à accomplir

- Orienté objet → héritage du modèle

Exemple

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  ...
</project>
```

Identifiant (unique) du projet : Identifiant de l'artifact produit

type du projet: pom, jar, war, ear, bundle, …

dépendances du projet envers d'autres projets (artifact)
constitue le $CLASSPATH

id d'une dépendance
version peut être un intervalle

portée de la dépendance par rapport au cycle de vie (compile, provided, runtime, test)

la suite bientôt …

POM Example

```
<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/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <groupId>com.companyname.project-group</groupId>
      <artifactId>project</artifactId>
      <version>1.0</version>
</project>
```

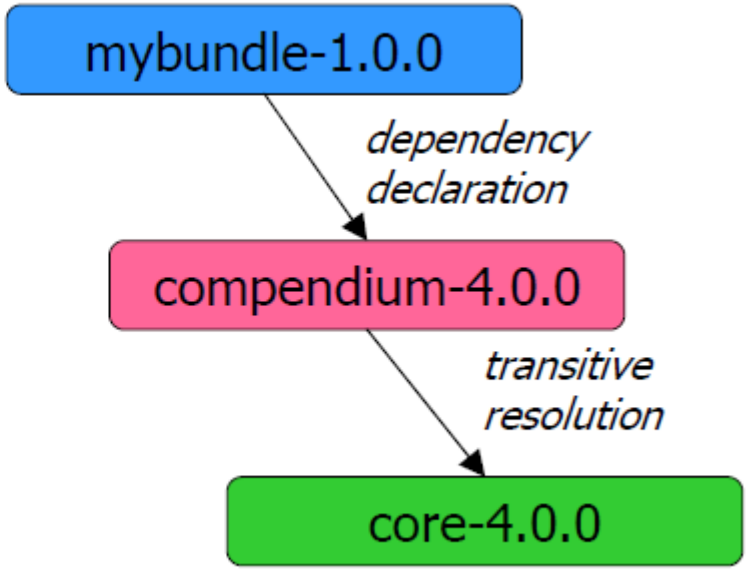
La configuration

- **Dans fichier pom.xml, vous décrivez votre projet :**
  - Quelle licence ? (<licence>)
  - Quels développeurs ? (<developpers>)
  - Quelles dépendences ? (<dependencies>)
  - Quel outil gestion des sources ? (<scm>)
  - Quel site web ? (<site>)
  - Quels plugins (<plugins>)…
- **Vous définissez de façon unique votre projet, en lui associant des 'coordonées' = ensemble d'identifiants :**
  - GroupId : groupe ds lequel le projet se trouve
  - ArtifactId : nom du projet
  - Version : version du projet

Dependances

- Concerne les artifacts comme les plugins
- Résolution transitive

```
<project>  ...
  <groupId>com.mycompany.app</groupId>
  <artifactId>mybundle</artifactId>
  <version>1.0.0</version> ...
  <dependencies>
    <dependency>
      <groupId>org.osgi</groupId>
      <artifactId>compendium</artifactId>
      <version>4.0.0</version>
</dependency>
 </dependencies>
... </project>
```



- Sert à constituer le CLASSPATH
  - Pour la compilation, pour les tests, pour l'exécution

Portée des dépendances

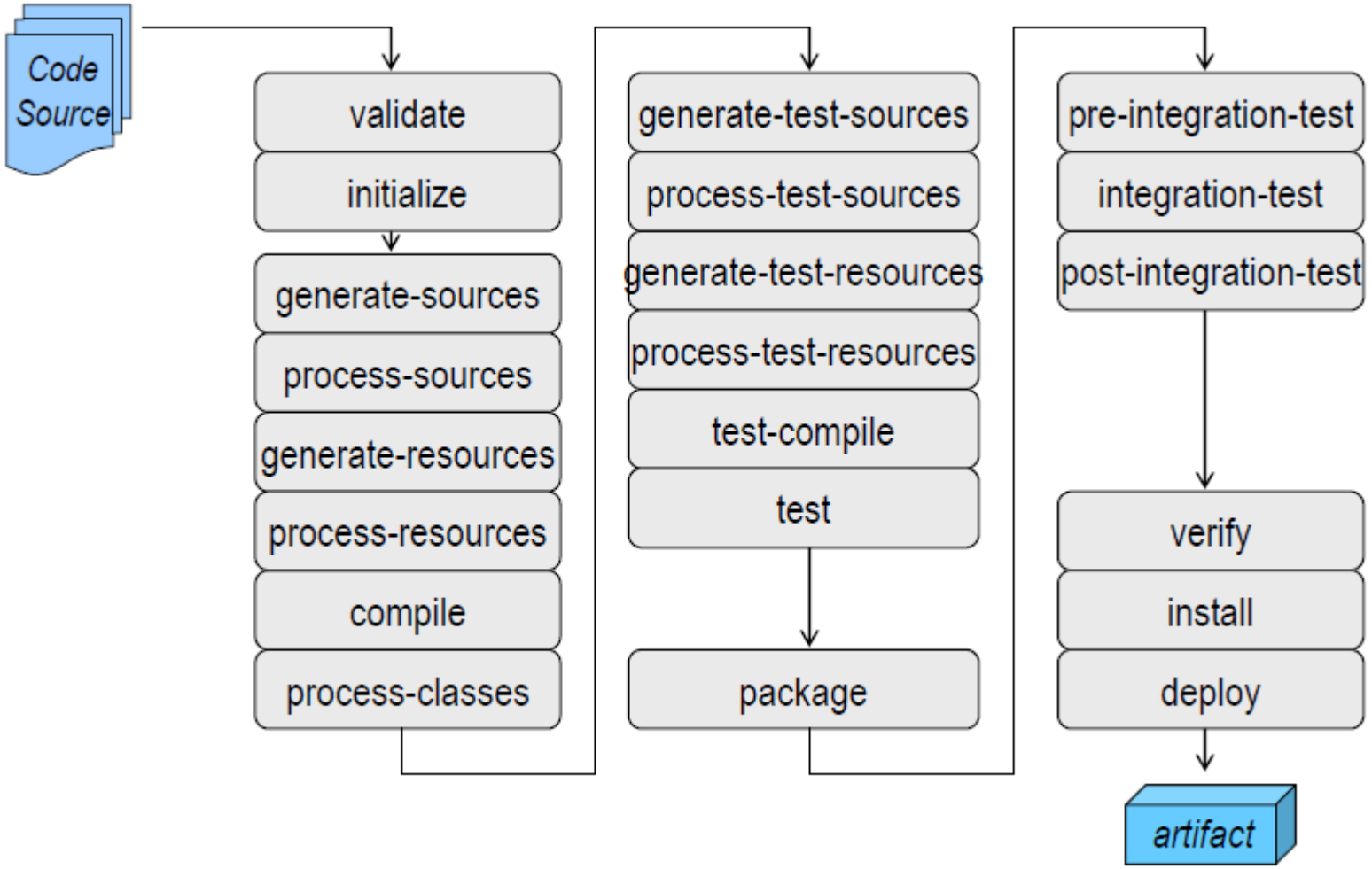
- **5 portées possibles par rapport aux classpaths du projet**
  - **compile (défaut)**
    - Disponible dans tous les classpaths
    - Transistive vers les projets dépendants
  - **provided**
    - compilation and test classpaths
    - Not transitive.
  - **runtime**
    - runtime and test classpaths.
  - **test**
    - test compilation and execution phases.
  - **system**
    - similar to provided but the artifact is always available and is not looked up in a repository.
  - **import**
    - only used on a dependency of type pom in the <dependencyManagement> section.

Cibles Maven

- **# mvn compile**
  - Compilation dans target/classes
- **# mvn package**
  - Création du Jar
- **# mvn install**
  - Création du Jar et install.
  - Repository local $USER_HOME/.m2/repository
- **# mvn package**
  Création du Jar
- **# mvn clean**
  Suppression du répertoire target

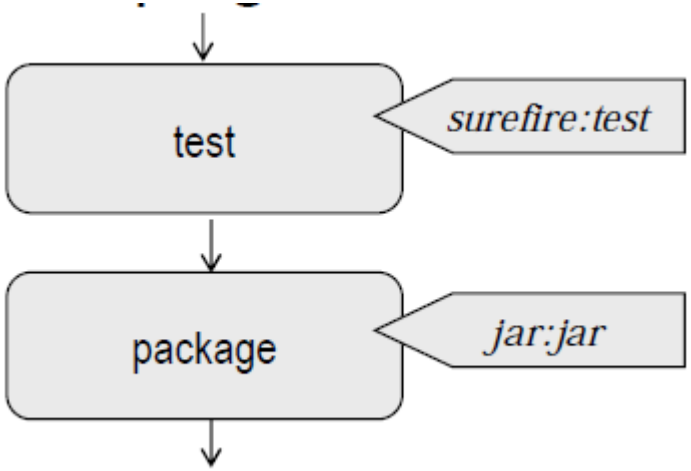
Cycle de vie (par défaut) d'un projet

- Séquence de 21 phases



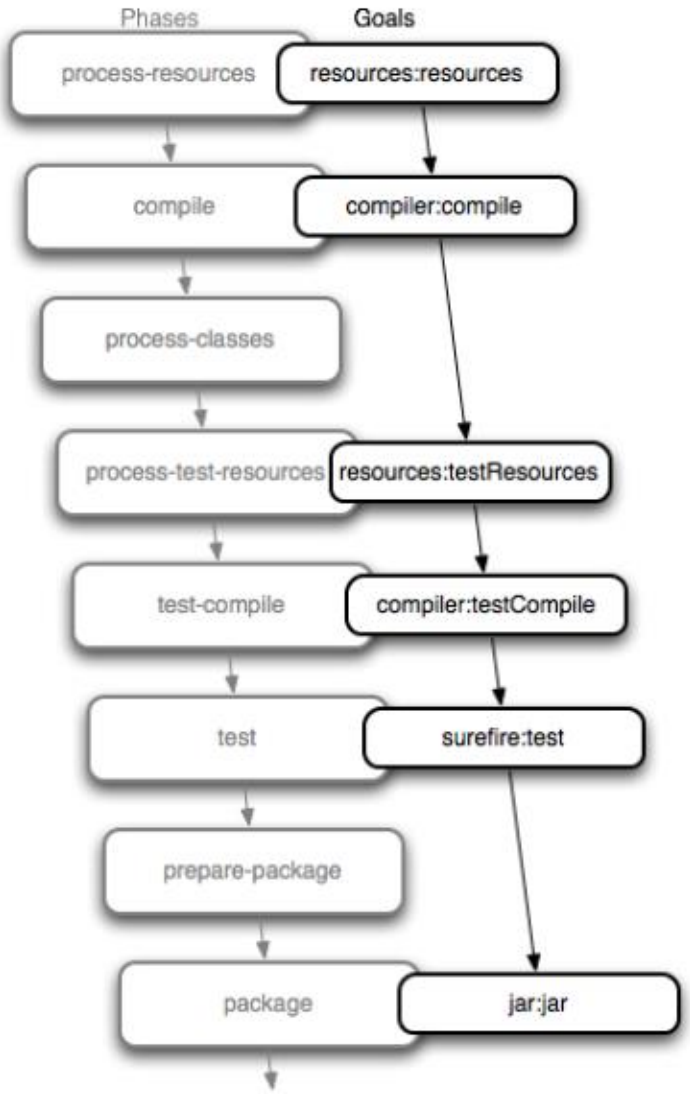
Phases et Buts (goals)

- A chaque phase est associé un ou plusieurs buts d'un ou de plusieurs plugins

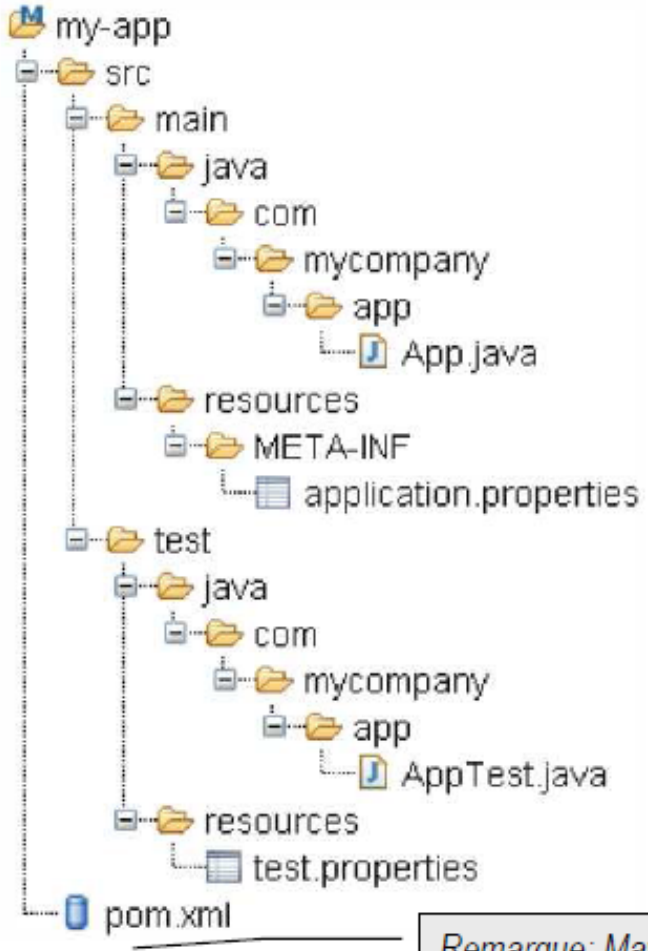


- Remarque
  - *mvn resources:resources compiler:compile resources:testResources compiler:testCompile surefire:test jar:jar est équivalent à mvn package*
- D'autre cycles de vie ont été définis
  - clean = pre-clean clean post-clean
  - site = pre-site site post-site site-deploy
  - …

Phases Maven

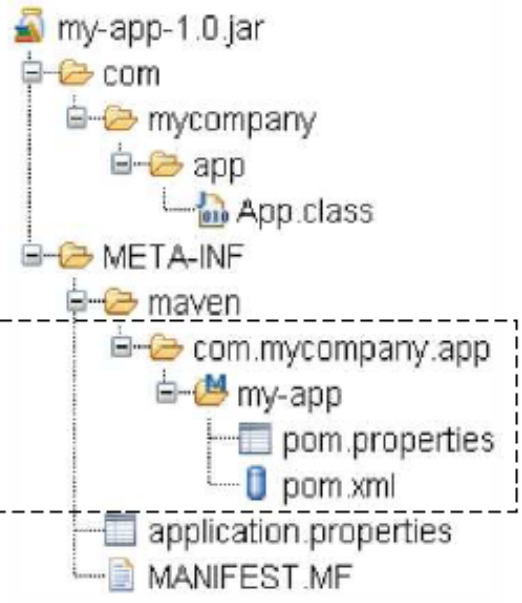


Structure « standard » d'un projet



Remarque: Maven crée un répertoire de travail ${basedir}/target commun à tous les plugins

mvn package



Remarque: Maven ajoute le POM à l'artifact packagé

Numérotation des versions

- Schéma
  - **<major>.<mini>[.<micro>][-<qualifier>[-<buildnumber>]]**
- Incrément
  - Major : changement majeur
    - pas de retro-compatibilité (descendante) garantie
  - Mini : ajouts fonctionnels
    - retro-compatibilité garantie
  - Micro : maintenance corrective (*bug fix)*
- Qualificateurs
  - SNAPSHOT (Maven) : version en évolution
  - alpha1 : version alpha (très instable et incomplète)
  - beta1, b1, b2 : version beta (instable)
  - rc1, rc2 : release candidate
  - m1, m2 : milestone
  - ea : early access
  - 20081014123459001 : date du build
  - jdk5 : dépendance avec une arch, un os, un langage
- Ordre sur les versions
    - Différent de l'ordre lexicographique
  - 1.1.1 < 1.1.2 < 1.2.2
  - 1.1.1-SNAPSHOT < 1.1.1
  - 1.1.1-alpha1 < 1.1.1-alpha2 < 1.1.1-b1 < 1.1.1-rc1 < 1.1.1-rc2 < 1.1.1
- Remarque (parfois)
  - <mini> pair : release stable
  - <mini> impair : release instable

Versionnement

- Snapshot
  - *A **snapshot in Maven is an artifact which has been prepared using** the **most recent sources available. … Specifying a snapshot version for a dependency means that Maven will look for new** versions of that dependency without you having to manually specify a new version.*
  - mvn -U command line option to force the search for updates.
- Dépendances
  - Spécification d'intervalles de versions

```
<dependency>
<groupId>org.codehaus.plexus</groupId>
<artifactId>plexus-utils</artifactId>
<version>[1.1,)</version>
</dependency>
```

Range	Meaning
(,1.0]	Less than or equal to 1.0
[1.2,1.3]	Between 1.2 and 1.3 (inclusive)
[1.0,2.0)	Greater than or equal to 1.0, but less than 2.0
[1.5,)	Greater than or equal to 1.5
(,1.1),(1.1,)	Any version, except 1.1

Quelques plugins usuels

- Core
  - clean, compiler, deploy, install, resources, site, surefire, verifier
- Packaging
  - ear, ejb, jar, rar, war, *bundle (OSGi)*
- Reporting
  - changelog, changes, checkstyle, clover, doap, docck, javadoc, jxr, pmd, project-info-reports, surefire-report
- Tools
  - ant, antrun, archetype, assembly, dependency, enforcer, gpg, help, invoker, one (interop Maven 1), patch, plugin, release, remoteresource, repository, scm
- IDEs
  - eclipse, netbeans, idea
- Autres
  - exec, jdepend, castor, cargo, jetty, native, sql, taglist, javacc, obr …

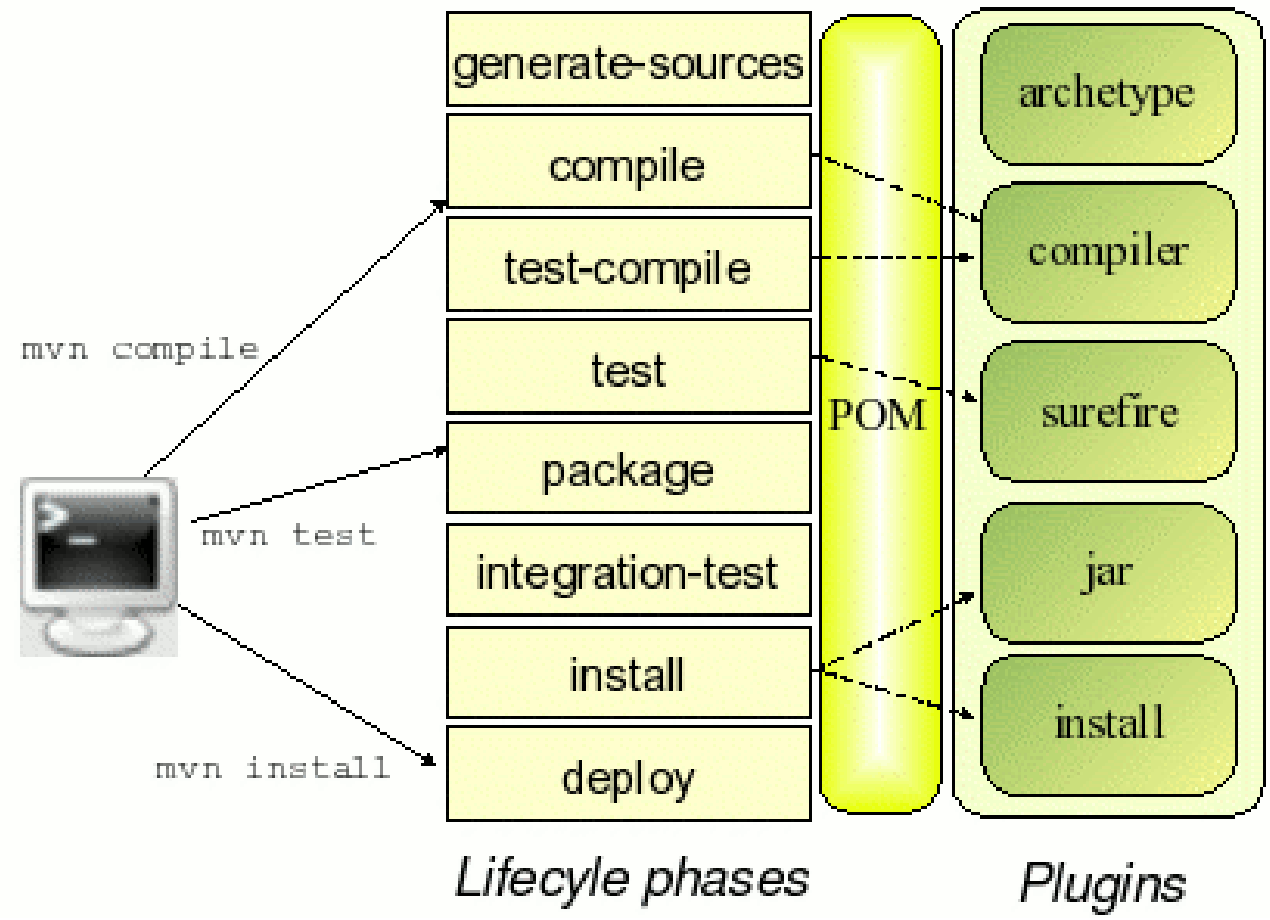
- http://maven.apache.org/plugins/,http://mojo.codehaus.org/plugins.html, …

Configuration des plugins

- Passage de paramètres autre que ceux définis par défaut
- Exemple

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
                <source>1.5</source>
                <target>1.5</target>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-jar-plugin</artifactId>
            <configuration>
                <archive>
                    <manifest>
                        <mainClass>${artifactId}.Main</mainClass>
                        <addClasspath>true</addClasspath>
                    </manifest>
                </archive>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Cycle de vie et relation Phases / Plugins

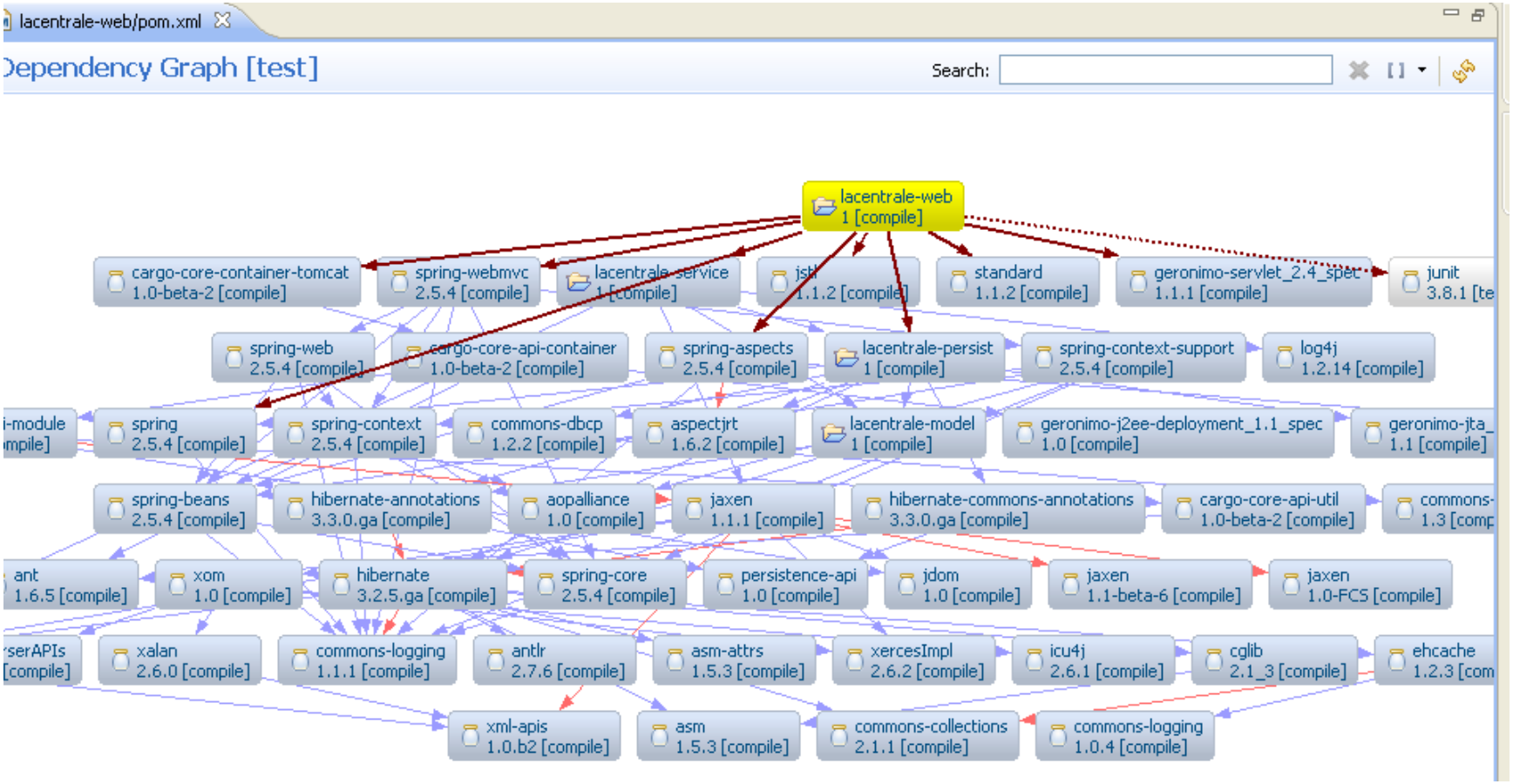


Plugin dependency

- mvn dependency:resolve (lister dépendances)
- mvn dependency:tree (arbre des dépendances)
- mvn dependency:tree -X (arbre : mode debug)
- Utile pour résolution de pb
- Equivalent graphique via plugin m2eclipse

```
C:\formationintegcont\tpmaven\tax-calculator-maven>mvn dependency:tree
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'dependency'.
[INFO] ------------------------------------------------------------------------
[INFO] Building Tax Calculator
[INFO]    task-segment: [dependency:tree]
[INFO] ------------------------------------------------------------------------
[INFO] [dependency:tree {execution: default-cli}]
[INFO] com.javapowertools.taxcalculator:taxcalculator:jar:1.0
[INFO] +- commons-logging:commons-logging:jar:1.1:compile
[INFO] |  +- logkit:logkit:jar:1.0.1:compile
[INFO] |  +- avalon-framework:avalon-framework:jar:4.1.3:compile
[INFO] |  \- javax.servlet:servlet-api:jar:2.3:compile
[INFO] +- log4j:log4j:jar:1.2.14:compile
[INFO] \- junit:junit:jar:4.4:test
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
```

Exemple graphe dépendances



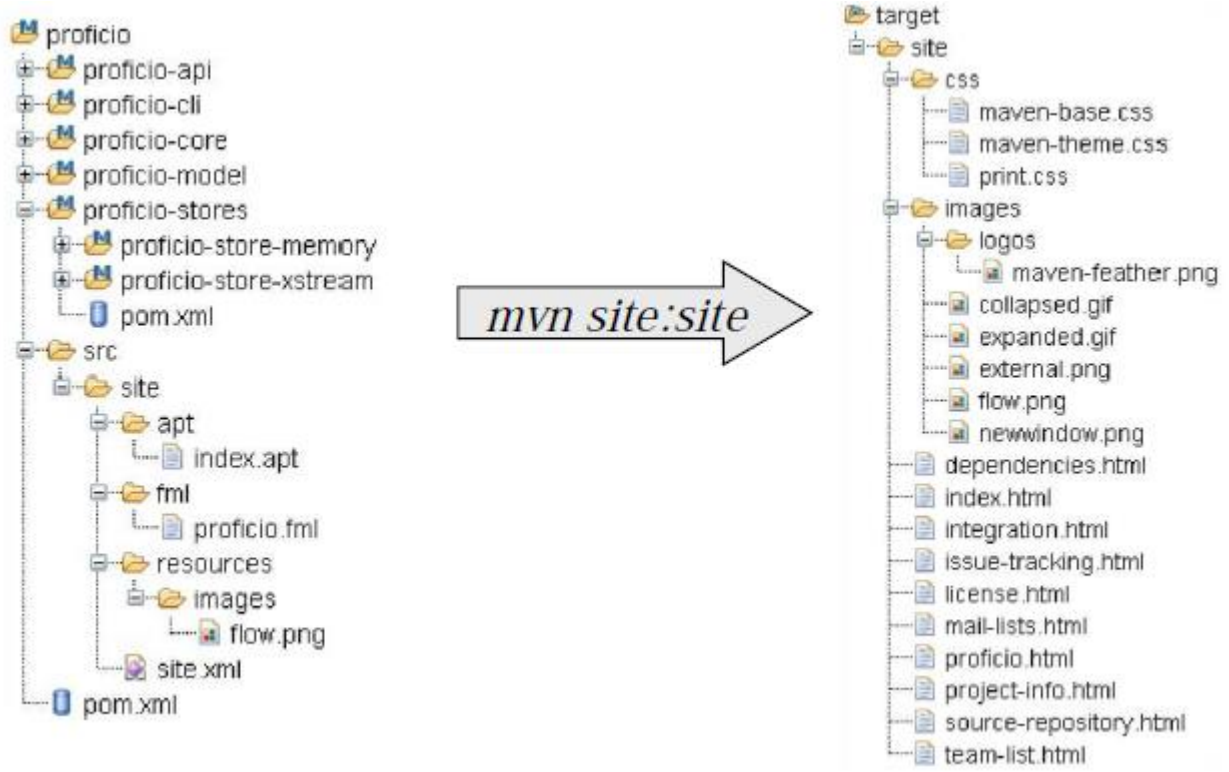
Profils

- Motivation
  - Améliorer la portabilité des projets par rapport aux environnements
    - Différents JVM, versions de Java, serveurs JEE, SGBD, développement versus production

  ➔Créer des variations (=profils) de projets
- Elément <profile> du build
  - Contient les variations de plugins et entre les plugins
- Activation du profil
  - Profil par défaut
  - En fonction des propriétés (systèmes, version JDK, …)
  - Par son identifiant
    - mvn --activate-profiles felix,equinox clean install

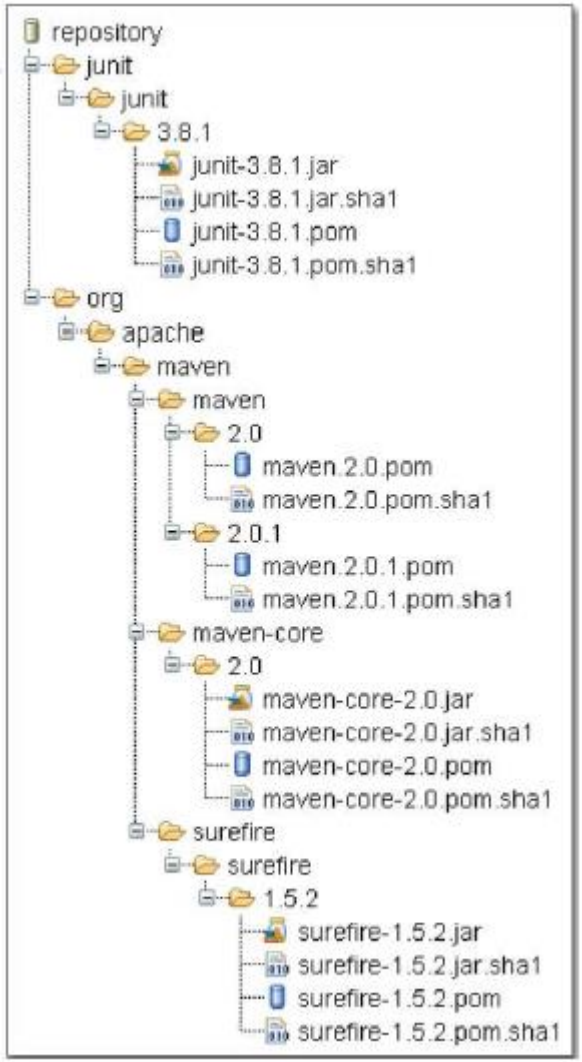
Documentation Web d'un projet

- Transforme plusieurs formats de documentation
  - XDOC, APT (Almost Plain Text), FML (FAQ ML), DocBook Simple, Twiki, Confluence
- La documentation source peut contenir des variables du projet ($project.name, $reports, …)



Dépôts de projets

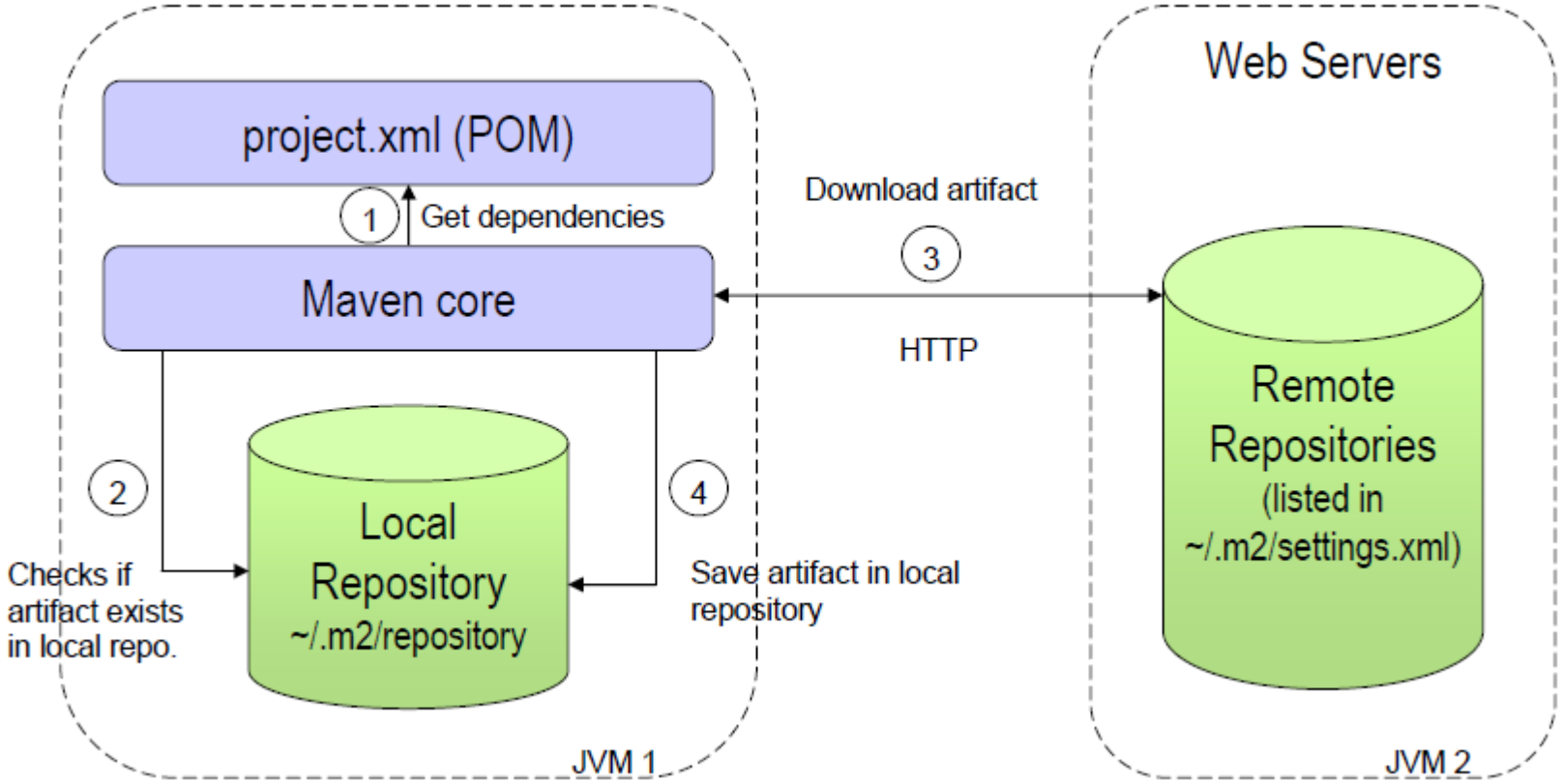
- Local ~/.m2/repository
  - Projets (dont artifacts) installés
  - localement
    - mvn install
    - mvn install:install-file
  - Caches des projets (artifacts) téléchargés depuis les dépots distants
    - Listés dans les POM et settings.xml
- Distants
  - Dépôts d'entreprise
  - Cache de dépôts
  - Dépôts publiques
- Structure
  - Nommage hiérarchique
  - ${groupId}.replace('.','/') / ${articfactId} / ${version}



Dépôts publiques

- Les principaux
  - Apache Maven Central
    - http://repo1.maven.org/maven2/
    - Plus de 20000 artifacts décrits *(en 2007)*
    - *Tous en licence ASL v2*
- CodeHaus
  - http://www.codehaus.org
  - Dependance vers d'autres licences (BSD, …)

Recherche des dépendances



R1: La mise à jour du dépôt local est journalière (sauf si `mvn -U`)
R2: Les plugins sont recherchés et mis à jour de la même façon

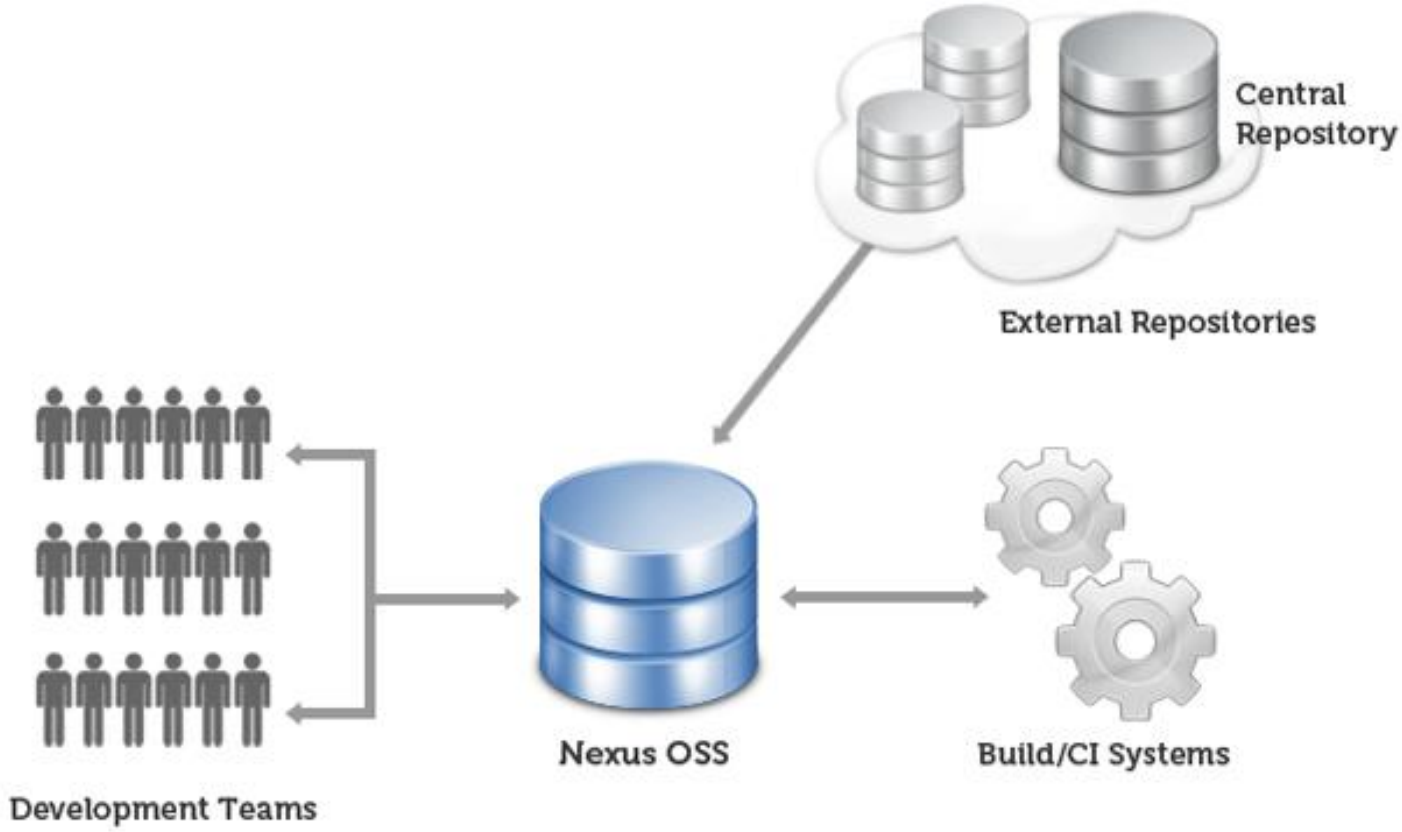
Settings.xml

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                      https://maven.apache.org/xsd/settings-1.0.0.xsd">
  <localRepository/>
  <interactiveMode/>
  <usePluginRegistry/>
  <offline/>
  <pluginGroups/>
  <servers/>
  <mirrors/>
  <proxies/>
  <profiles/>
  <activeProfiles/>
</settings>
```

Gestionnaire repository Maven

- Offre du marché :
  - Archiva : http://archiva.apache.org
  - Artifactory : http://jfrog.org
  - Nexus : http://nexus.sonatype.org
    - par équipe Maven, + moderne, nombreux plugins
- Critères d'évaluation
  - Cycle de livraison
  - Modèle de déploiement (war)
  - Proxy/Cache de repositories (maven central, codehaus..)
  - Règles d'inclusion / exclusion
  - Conversion à la volée de Maven2/Maven1
  - integration eclipse (Uniquement index Nexus)
  - Recherche/indexation
- Matrice de comparaison :
  - https://binary-repositories-comparison.github.io/

Nexus


Central
Repository
External Repositories
Development Teams
Nexus OSS
Build/CI Systems

Substitution de variables à la construction

Motivations

- Instancier les valeurs des ressources lors de la phase *process-resources*

Exemple de POM

```
... <build>
 <filters>
  <filter>src/main/filters/filter.properties</filter>
 </filters>
 <resources>
  <resource>
   <directory>src/main/resources</directory>
   <filtering>true</filtering>
  </resource>
 </resources>
</build>
```

```
# src/main/filters/filter.properties
my.filter.value=Hello !
```

```
# src/main/resources/application.properties
message=${my.filter.value}
application.name=${project.name}
application.version=${project.version}
```

Archetype

- Construction initial d'un projet Maven
  - En fonction d'un type de projet T
    - T= quickstart, archetype, bundles, j2ee-simple, marmalademojo, mojo, plugin, plugin-site, portlet, profiles, simple, site, site-simple, webapp, …
- Exemple
  - mvn archetype:create

    mode interactif
  - mvn archetype:create

    -DgroupId=demo.maven

    -DartifactId=hello

    -Dversion=0.1.0-SNAPSHOT

    -DarchetypeGroupId=org.apache.maven.archetypes

    -DarchetypeArtifactId=maven-archetype-quickstart

Archetypes personnalisés

- Possibilité de créer ses propres archetypes

  ➔ de zero

  mvn archetype:create

  -DarchetypeGroupId=org.apache.maven.archetypes

  -DarchetypeArtifactId=maven-archetype-archetype

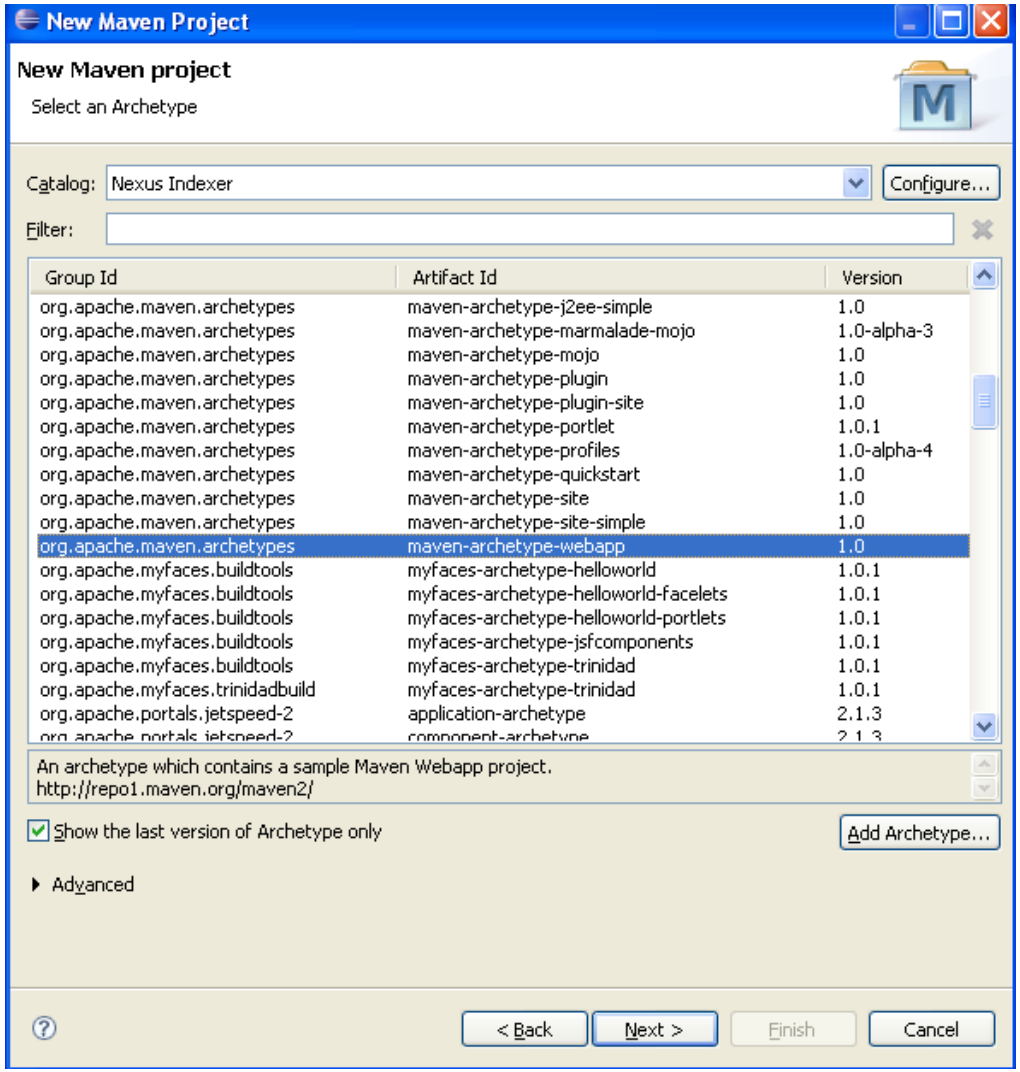
  -DgroupId=com.mycompany

  -DartifactId=my-archetype

  ➔ depuis un archetype existant
- Développement

  Basé sur des templates Velocity (http://velocity.apache.org/)

Eclipse Archetypes

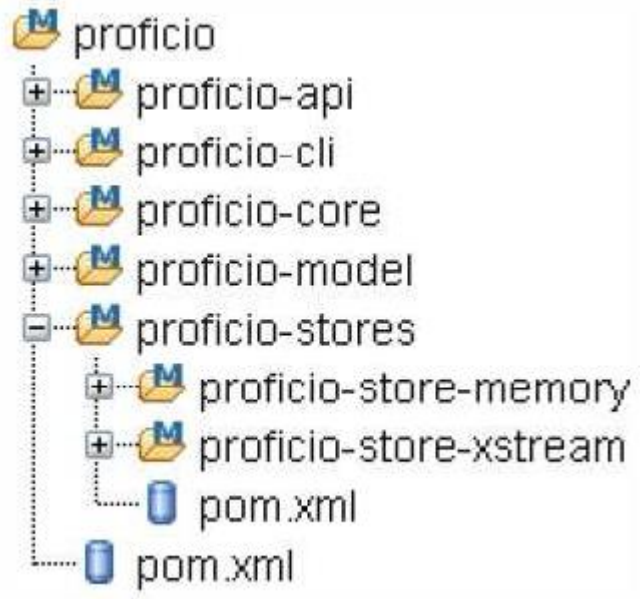


Good & Best Practices

- Beginners
  - KISS (Keep It Simple, Stupid)
  - Start from scractch
    - No Copy/Paste
  - Use only what you need
    - Filtering, Modules, Profiles, …
- Bad practices
  - Ignore maven conventions
  - Different versions in sub modules
  - Too many inheritance levels
  - AntRun (OK for integration test)
  - Plugins without versions
  - …

Organisation hiérarchique de projets

- Motivations
  - Organiser le développement en sous-projets
    - Avec N niveaux (N>=1)
- Méthode
  - Création d'un super POM (de type pom) par niveau
    - Regroupe les plugins/goals communs du même niveau
  - Les sous-projets (appelé modules) héritent de ce super pom
- Exemple

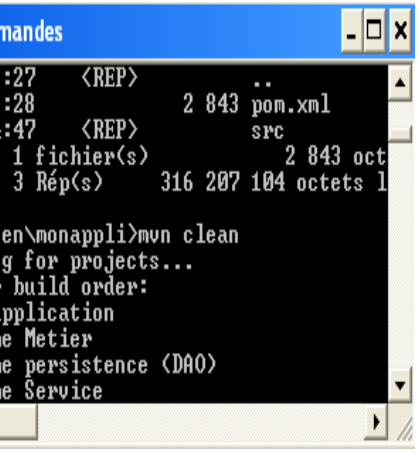


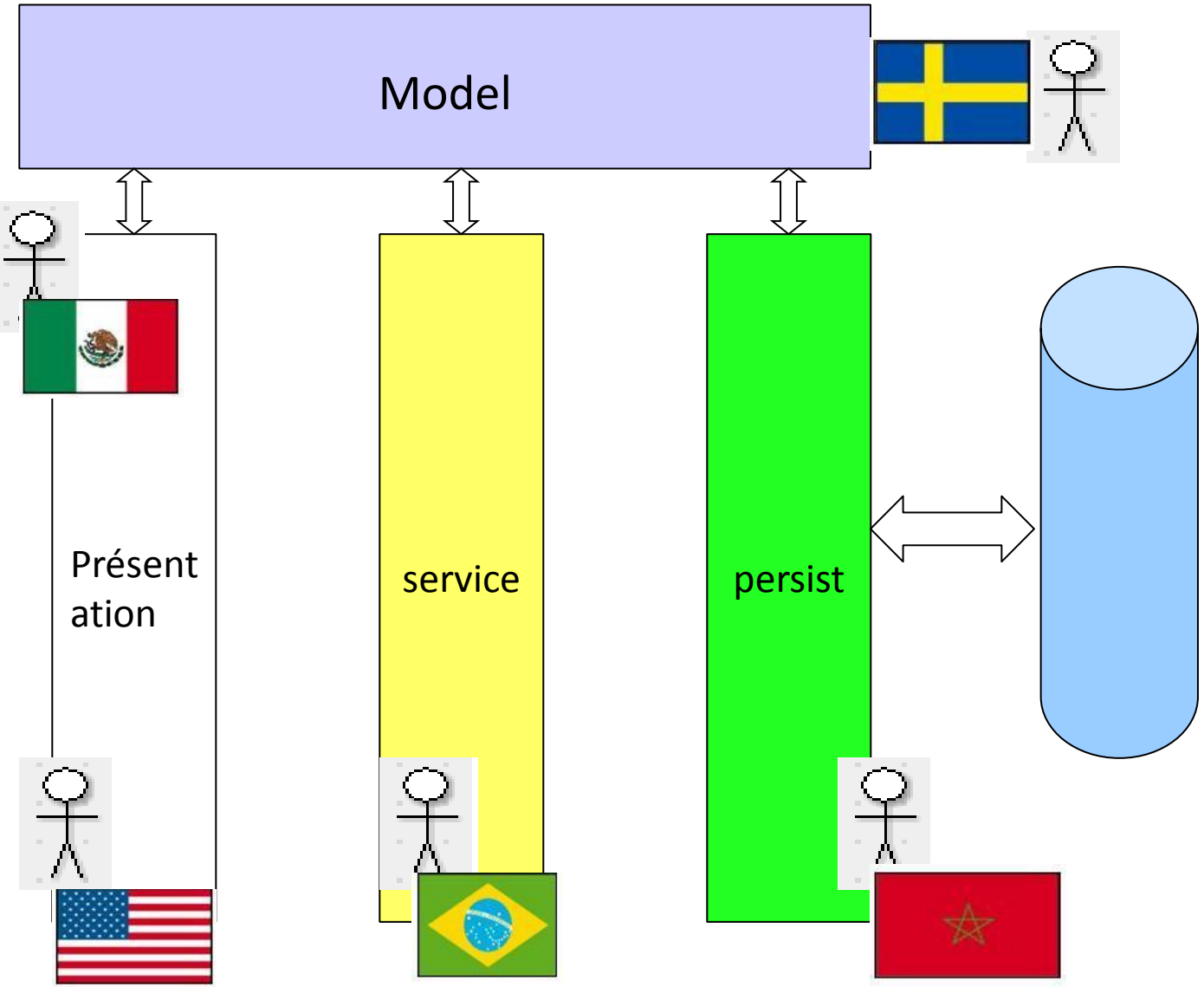
- Commande
  - mvn --reactor clean install
    - Pour la construction globale

Exemple de projet multi modules

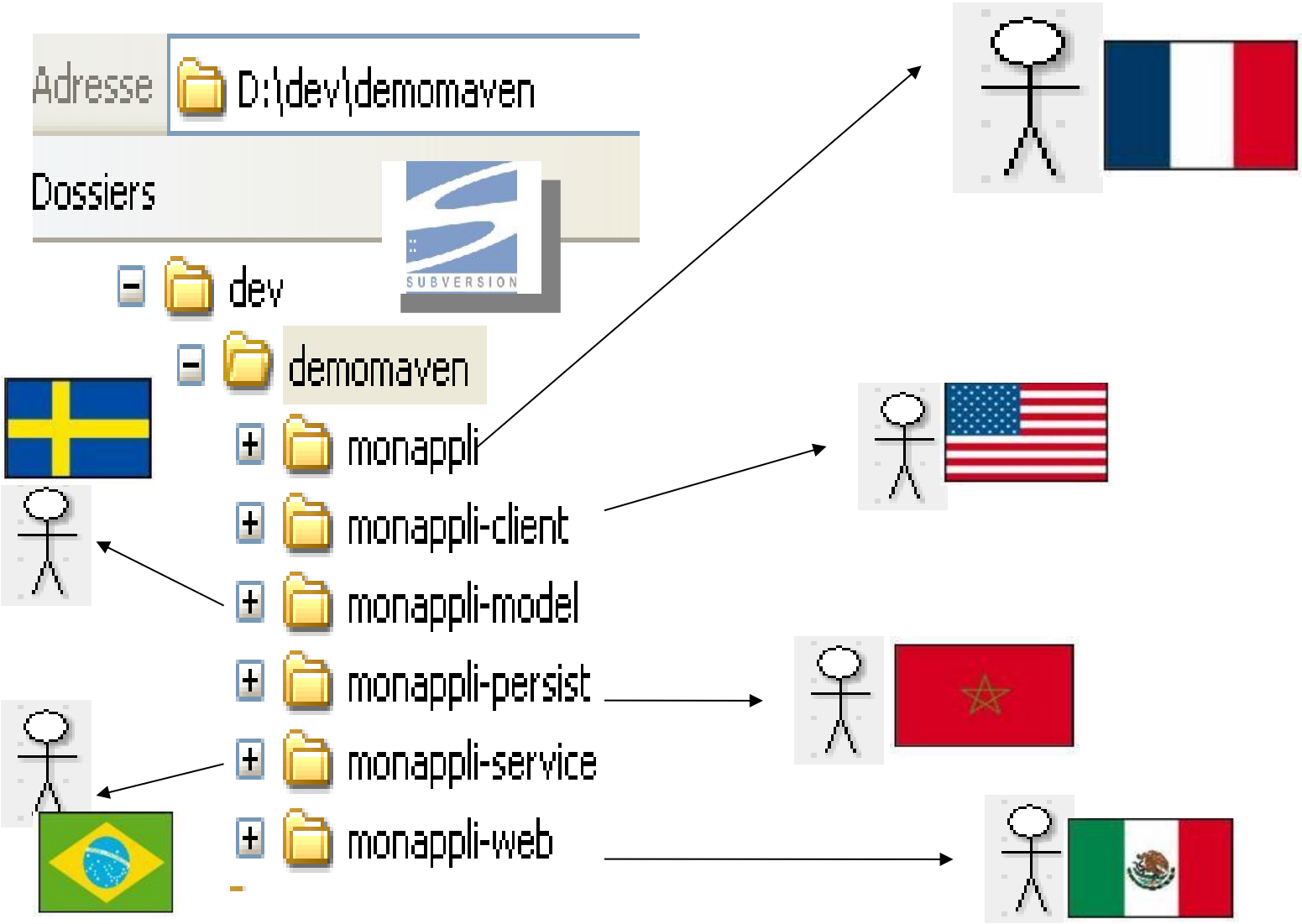
- Plusieurs équipes de développements
  - France, Suède, Maroc, Brésil, Italie, USA
- Chef de projets basé en France
  - Besoin de suivi avancement travaux (Quantité + qualité)
- Architecture en couche
- Technologies : Struts, Spring, Hibernate

Architecture

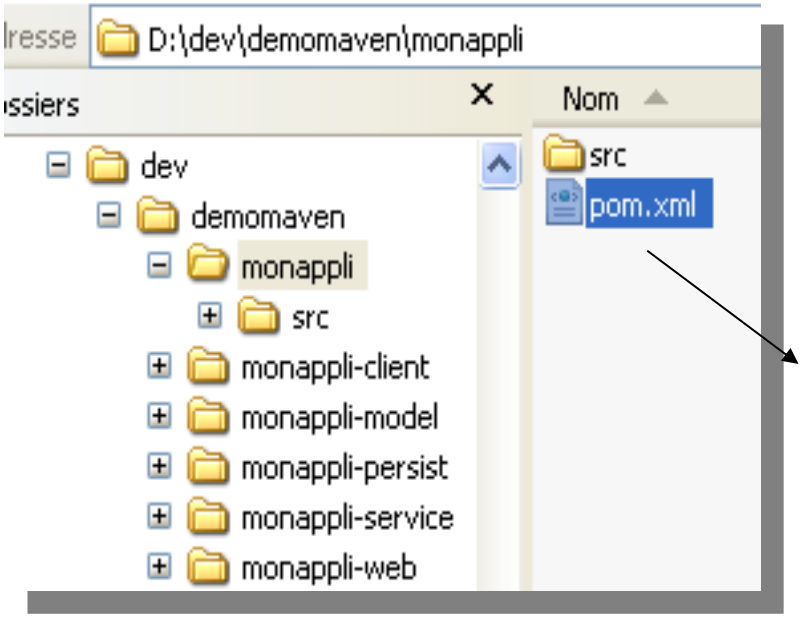




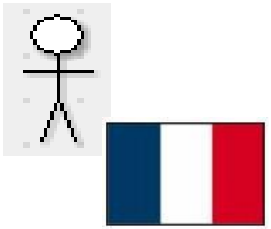
Projet Maven multi-modules



Projet Maven multi-modules




```
<!-- DESCRIPTION PROJET 'Mon appli' -->

<modelVersion>4.0.0</modelVersion>
<groupId>com.objis.demomaven</groupId>
<artifactId>monappli-parent</artifactId>
<version>1</version>
<packaging>pom</packaging>
<name>Mon application</name>

<!-- MODULES PROJET 'Mon appli' -->

<modules>
  <module>../monappli-model</module>
  <module>../monappli-persist</module>
  <module>../monappli-service</module>
  <module>../monappli-web</module>
  <module>../monappli-client</module>
</modules>
```

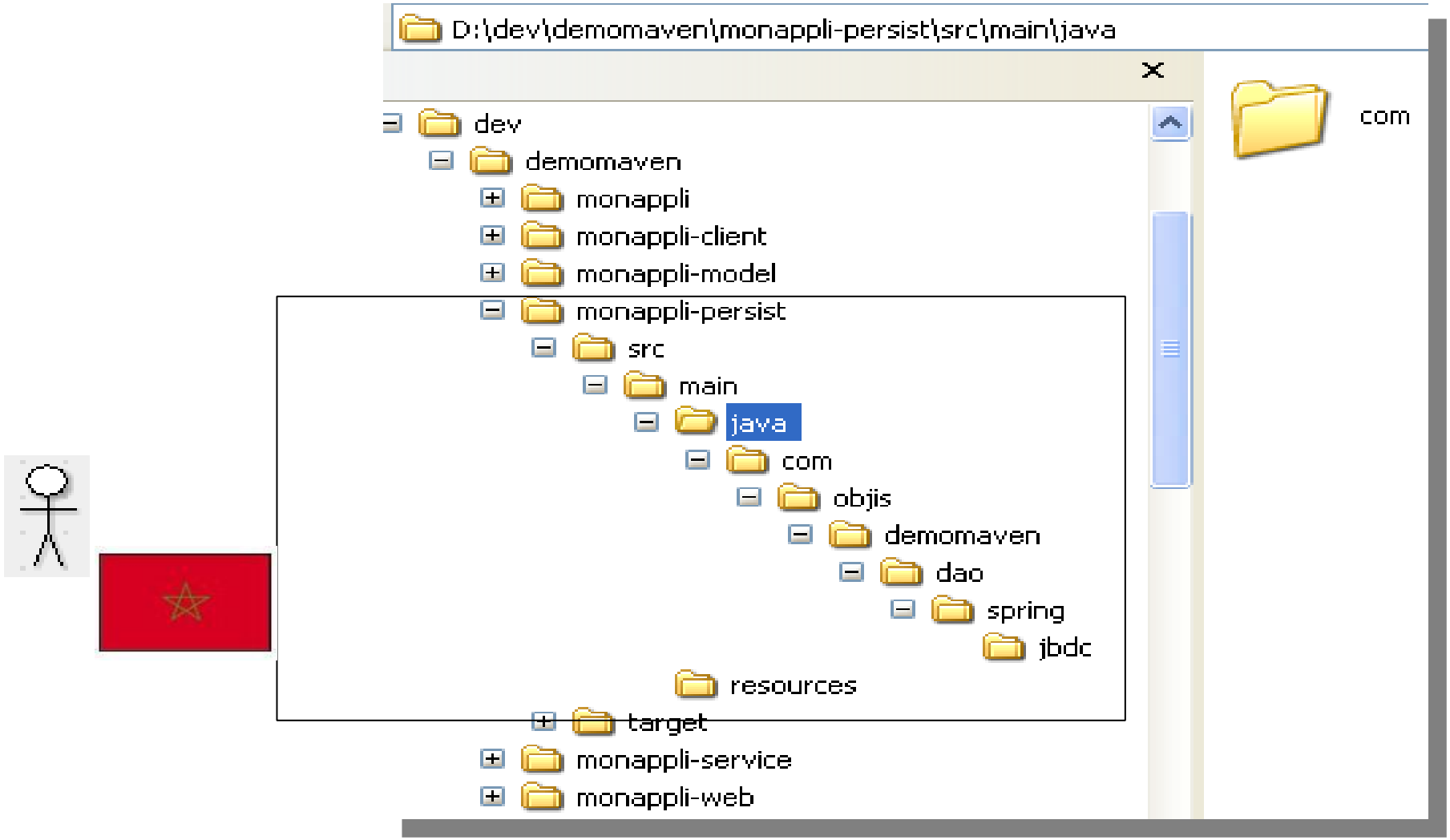


(Composition)

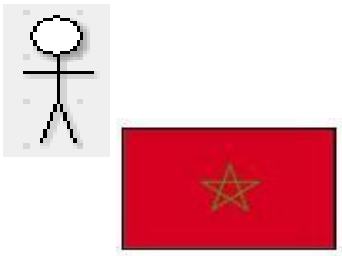
Mvn install sur le parent entraîne aussi

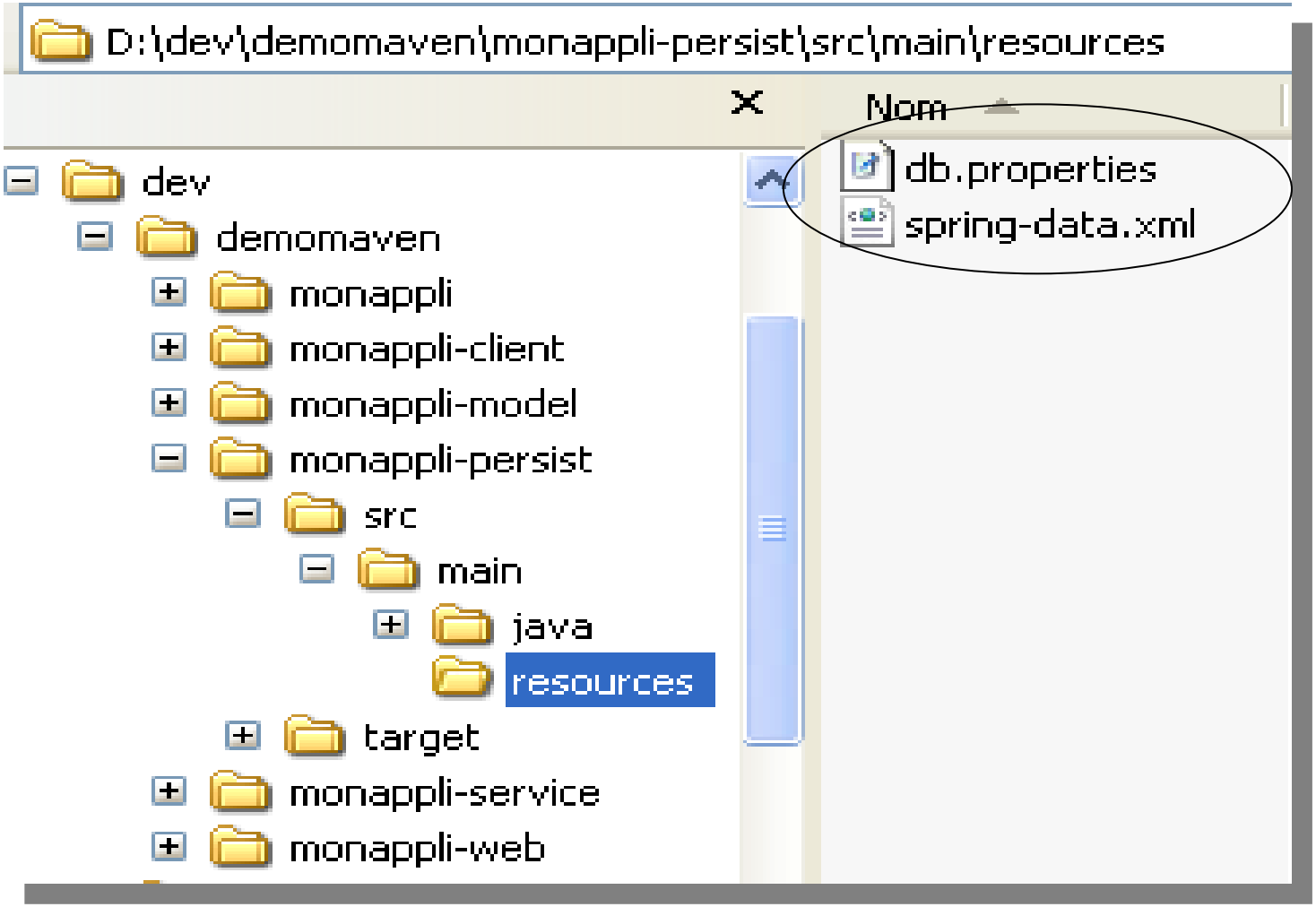
mvn install sur modules enfant

Focus couche persistence : les sources

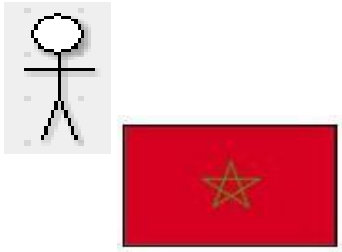


Focus couche persistence : les fichiers de propriétés





Focus couche persistence :



```
<!-- DEPENDENCES DU MODULE d'accès aux donées (persistence) -->

<dependencies>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.0.5</version>
    </dependency>
  <dependency>
    <groupId>com.objis.demomaven</groupId>
    <artifactId>monappli-model</artifactId>
    <version>1</version>
  </dependency>
  <dependency>
    <groupId>commons-dbcp</groupId>
    <artifactId>commons-dbcp</artifactId>
    <version>1.2.2</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring</artifactId>
    <version>2.5.4</version>
  </dependency>
  <dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>1.6.2</version>
  </dependency>
</dependencies>
```

Focus couche persistence : relation avec parent

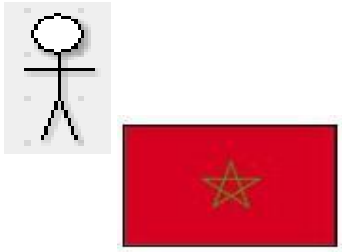
Le module hérite de certaines propriétés du parent (ex : éépendences) .

(Héritage)

Cela allège le pom.xml du module

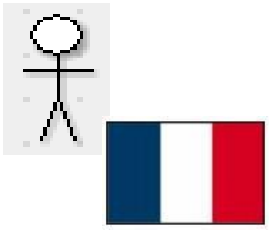
```
    <!-- DESCRIPTION MODULE d'accès aux donées (persistence) -->

<modelVersion>4.0.0</modelVersion>
<groupId>com.objis.demomaven</groupId>
<artifactId>monappli-persist</artifactId>
<version>1</version>
<packaging>jar</packaging>
<name>Couche persistence (DAO)</name>

  <!-- DECLARATION PROJET PARENT  -->

<parent>
  <groupId>com.objis.demomaven</groupId>
  <artifactId>monappli-parent</artifactId>
  <version>1</version>
  <relativePath>../monappli/pom.xml</relativePath>
</parent>
```

Focus couche persistence : les rapports



```
    <!-- RAPPORTS liées au module d'accès aux données (persistence)-->
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-checkstyle-plugin</artifactId>
    </plugin>
  </plugins>
</reporting>
```

Projet Maven : compilation globale



> mvn clean install

```
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO] ------------------------------------------------------------------------
[INFO] Mon application ..................................... SUCCESS [19.797s]
[INFO] Couche Metier ....................................... SUCCESS [24.140s]
[INFO] Couche persistence .................................. SUCCESS [39.313s]
[INFO] Couche Service ...................................... SUCCESS [5.375s]
[INFO] Couche Presentation ................................. SUCCESS [12.922s]
[INFO] Couche Client console ............................... SUCCESS [38.156s]
[INFO] ------------------------------------------------------------------------
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2 minutes 20 seconds
[INFO] Finished at: Wed Apr 15 16:00:38 CEST 2009
[INFO] Final Memory: 39M/254M
[INFO] ------------------------------------------------------------------------

D:\dev\demomaven\monappli>
```

DependencyManagement

- Si le projet ’monappli-parent’ utilise pour définir une dépendance à junit:junit:4.4 , alors les POMs héritant du parent peuvent définir leur dépendance en fournissant uniquement groupId=junit et artifactId=junit (pas la version). Maven trouvera remplir la version en utilisant la version du parent.
- — Bénéfice 1 : les détails de dépendance peuvent être centralisé dans un endroit qui sera propagé dans les POMs hérités.
- — Bénéfice 2 : la version et le scope des artifacts impliqués dans les dépendances transitives peuvent aussi être controlés en les spécifiant dans la section

Optimisation : Module web

```xml
<!--OPTIMISATION DEPENDANCES-->

<dependencyManagement>
<dependencies>
  <dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>monappli-domaine</artifactId>
    <version>${project.version}</version>
  </dependency>
    <dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>monappli-dao</artifactId>
    <version>${project.version}</version>
  </dependency>
    <dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>monappli-service</artifactId>
    <version>${project.version}</version>
  </dependency>
    <dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>monappli-web</artifactId>
    <version>${project.version}</version>
  </dependency>
</dependencies>
</dependencyManagement>
```

```xml
  <dependencies>
    <dependency>
      <groupId>${project.groupId}</groupId>
      <artifactId>monappli-service</artifactId>
      <!--<version>${project.version}</version>-->
    </dependency>
    <dependency>
      <groupId>${project.groupId}</groupId>
      <artifactId>monappli-domaine</artifactId>
      <!--<version>${project.version}</version>-->
    </dependency>
    <dependency>
    <groupId>org.apache.geronimo.specs</groupId>
    <artifactId>geronimo-servlet_2.5_spec</artifactId>
    <version>1.2</version>
</dependency>
```

Comparaison ANT / MAVEN

- MAVEN a des **conventions**. Il sait déja où sont les sources, les tests, les fichiers de config...
  - Classes créées dans target/classes
  - Création de Jars.

- MAVEN est **déclaratif**. Tout ce que vous avez à faire est de créer un fichier pom.xml et mettre vos sources dans le répertoire par défaut.

- Un projet MAVEN possède **un cycle vie**, que vous invoquez lorsque vous lancez mvn install.
  - Maven execute un ensemble d'instructions associées au cycle de vie du projet

Comparaison ANT / MAVEN

Cibles ANT

```xml
<project name="my-project" default="dist" basedir=".">
    <description>
        Exemple de fichier build
    </description>
  <!-- propriétés globales -->
  <property name="src" location="src/main/java"/>
  <property name="build" location="target/classes"/>
  <property name="dist" location="target"/>
  <target name="init">
    <!-- Dates -->
    <tstamp/>
    <!-- creation repertoire de build -->
    <mkdir dir="${build}"/>
  </target>
  <target name="compile" depends="init"
        description="compile the source " >
    <!-- Compilation de ${src} vers ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>
  <target name="dist" depends="compile"
        description="generate the distribution" >
    <!-- Creation repertoire de livraison -->
    <mkdir dir="${dist}/lib"/>
    <!-- Copie de ${build} vers fichier Jar MyProject-${DSTAMP}.jar  -->
    <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
  </target>
  <target name="clean"
        description="clean up" >
    <!-- Supprimer arborescence ${build} et ${dist}  -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
</project>
```

Appel de tâches ANT dans un projet Maven

Motivations

- récupération de projets existants avant connversion
- Exécution de taches patrimoniales n'ayant pas de plugins équivalents
  - Remarque: pensez à utiliser la définition de Macro ANT !

Exemple avec le plugin org.apache.maven.plugins:maven-antrun-plugin

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-antrun-plugin</artifactId>
  <executions>
    <execution>
      <phase>generate-sources</phase>
      <configuration>
        <tasks unless="maven.test.skip">
          <!-- Place any ant task here. You can add anything
          you can add between <target> and </target> in a build.xml.-->
            <echo message="To skip me, just call mvn -Dmaven.test.skip=true"/>
            <exec dir="${basedir}"
                  executable="${basedir}/src/main/sh/do-something.sh" failonerror="true">
              <arg line="arg1 arg2 arg3 arg4" />
            </exec>
        </tasks>
      </configuration>
      <goals>
        <goal>run</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Conversion d'un projet ANT en projet Maven

- 2 possibilités pour la structure du projet
  - Réorganiser (manuellement, projet ANT si plusieurs projets)
    - src ➔ src/main/java, src/test/java, doc ➔ src/site
    - classes ➔ target/classes, build ➔ target, …
  - Configurer les paramètres par défaut du POM en fonction de la structure du projet ANT
- Définir les dépendances
  - en fonction du <classpath …>

Antlib for Maven

Taches Maven pour projet ANT

- Manipulation d' artifacts depuis un projet Ant
  - Gestion (transitive) des dépendances
    - scope recognition and SNAPSHOT handling
- Déploiment des artifacts vers un dépot Maven
- Analyse d'un pom.xml

Exemple

```
<artifact:dependencies pathId="dependency.classpath">
   <dependency groupId="javax.servlet" artifactId="servlet-api"
         version="2.4" scope="provided" />
   ...
</artifact:dependencies>
<javac ...>
   <classpath refid="dependency.classpath" />
   ...
</javac>
```

Développement de plugins

- Plugin = { <goal,MOJO> }
- MOJO = Maven POJO
  - Annotations XDocLet
- Langages
  - Java et Groovy (pour le scripting)
  - D'autres possibles …
- Déploiement
  - Artifact Maven
  - Utilise les mécanismes de déploiement (version, dépendances, …)
  - Dépôts de plugins
    - http://maven.apache.org/plugins/, http://repository.codehaus.org/

Développement de plugins
Exemple (i)

```
package sample.plugin;
import org.apache.maven.plugin.AbstractMojo;
import org.apache.maven.plugin.MojoExecutionException;
/**
 * Says "Hi" to the user.
 * @goal sayhi
 * @phase compile
 */
public class GreetingMojo extends AbstractMojo {
    /** The greeting to display.
     * @parameter alias="message" expression="Hello, world (from ${project.groupId}:${project.artifactId})" */
    private String greeting;

/** The classpath.
  * @parameter expression="${project.compileClasspathElements}"
  * @required
  * @readonly */
  private List classpathElements;

    public void execute() throws MojoExecutionException {
      getLog().info(greeting);
      getLog().info("Project classpath: " + classpathElements().toString().replace( ',', ';' ));
    } }
```

phase et but durant laquelle execute() est appelé

paramètre renseigné dans <configuration>

Integer, …, String, List, Properties, Map, Object, File, URL, …

paramètre issue du pom

Développement de plugins
Exemple (ii)

Dans le POM

```
<build>
  <plugins>
    <plugin>
      <groupId>sample.plugin</groupId>
      <artifactId>maven-hello-plugin</artifactId>
      <configuration>
          <message>Welcome</message>
       </configuration>
    </plugin>
  </plugins>
</build>
```

Exécution

```
mvn sample.plugin:maven-hello-plugin:sayhi
```

Plugins et Cycles de vie

- MOJO attaché à une phase du cycle de vie
  - @nnotations doclet
- Cycles de vie personnalisés
  - Surcharge de META-INF/plexus/components.xml,