Génie Logiciel

Philippe Collet

Master 1 IFI
2012-2013
http://deptinfo.unice.fr/twiki/bin/view/Minfo/GenieLog1213

Génie logiciel : organisation générale

Lignes de produits logiciels

Génie Logiciel (réflexivité, test OO, profilage, généricité, héritage, patrons de conception, chargement dynamique, composants…)

M1

Conception OO (UML+OCL)

Prog. système

Projet de devt (C, PHP ou Java)

L3

Objectifs

- Maîtriser des techniques de génie logiciel, en se focalisant sur les approches par objets et par composants
- Principes et outils de construction et d'exécution automatiques
- Tests
- Introspection, réflexivité et chargement dynamique
- Choix et limites des mécanismes d'héritage, de composition et de généricité contrainte
- Micro-architectures et patrons de conception
- Premier pas en architecture logicielle

Plan

- Construction (outil ant)
- Introspection, réflexivité
- V&V objet, test unitaire OO
- Généricité
- Héritage
- Patrons de conception
- Chargement dynamique
- (Injection de dépendances)

Evaluation

- 1 note de contrôle continu en TD
  - (présence, travail, participation)
  - (25%)
- 1 projet par équipe
  - (35%)
- 1 contrôle terminal
  - (40%)

Automatisation de construction en Java : ANT

- D'après le cours sur Ant de Richard Grin
- http://ant.apache.org/

Introduction

- Syntaxe et options très fournies
- Dans ce cours
  - Version 1.5
  - Pas de syntaxe complète des tâches
  - Ant est toujours distribué avec son manuel
- Projet Open source (fondation Apache)
- La référence pour la construction automatique et **portable** d'applications Java
- Ecrit lui-même en Java

Principes

- Modèle de la commande make
  - un projet
  - des cibles (compile, jar, javadoc,…)
  - La description des cibles et les dépendances entre les cibles sont décrites dans un fichier
  - *Fichier XML, nommé par défaut build.xml*
- Extensible : on peut ajouter ses propres commandes/tâches

build.xml : exemple

```
<project name="hello" default="compile">

  <target name="prepare">
    <mkdir dir="./classes" />
  </target>

  <target name="compile" depends="prepare">
    <javac srcdir="./src"
           destdir="./classes" />
  </target>
</project>
```

Script de construction : structure

- une en-tête XML (avec l'indication optionnelle d'une DTD)
- une entrée **`project`** qui contient
  - optionnellement, des entrées **`property`**
  - optionnellement, des entrées path ou classpath
  - une ou plusieurs entrées **`target`**
  - optionnellement, une entrée **`description`**
    - Description informelle du projet
    - ```
      <description>
      Ce projet permet de . . .
      . . .
      </description>
      ```
      ```

Entrée `project`

- Chaque fichier de construction contient une et une seule entrée project
- Cette entrée peut avoir 3 attributs
  - **`name`** le nom du projet
  - **`default`** la cible par défaut (requis)
  - **`basedir`** le répertoire de base pour les chemins relatifs
    - peut être écrasé par la propriété **`basedir`**
    - par défaut le répertoire où se trouve le fichier de construction

Les cibles

- Une cible (**`target`**)
  - correspond à une action qui est décrite dans le fichier
  - peut dépendre d'autres cibles (attribut **`depends`**)
- Chaque type de cible peut avoir ses propres attributs
- Les attributs communs à toutes les cibles :
  - **`name`** : le nom de la cible (obligatoire)
  - **`description`** : si elle apparaît, permet de lister une description de la cible avec l'option **`-projecthelp`** de l'appel de **`ant`**
  - **`depends`** : permet d'indiquer les autres cibles dont dépend une cible

Dépendances de cibles

- On peut indiquer plusieurs cibles dont une cible dépend (**`depends A,B,C`** par exemple)
  - les cibles seront exécutées dans l'ordre du **`depends`** (de gauche à droite)
- Dans la gestion des dépendances, les tâches ne sont exécutées qu'une fois :

```
<target name="A"/>
<target name="B" depends="A"/>
<target name="C" depends="A,B"/>
```

A ne sera exécuté qu'une seule fois

Comportement sur erreur

- Le plus souvent, une erreur dans une tâche arrête la construction de la cible correspondante
  - Une classe ne compile pas, la cible qui construit le jar s'arrête

- Certaines tâches ne provoque pas d'arrêt
  - On peut leur ajouter un attribut « **`failonerror`** » à **`true`** pour forcer l'arrêt
  - Exemple : la tâche « **`java`** »

Cible d'initialisation

- Il est recommandé d'avoir une cible d'initialisation nommé **`init`** qui contient au moins la tâche **`tstamp`** :
  -
```
<target name="init">
  <tstamp/>
</target>
```
  - tstamp récupère le temps système et initialise les propriétés **`DSTAMP`** (aaaammjj), **`TSTAMP`** (hhmm), et **`TODAY`** (mois jour année)
- Toutes les cibles liées à la construction de l'application devront dépendre de la cible **`init`** (directement ou non)

Tâches

- Une tâche est une unité d'exécution « élémentaire » pour réaliser une cible
- Attributs possibles :
  - id donne un identificateur unique à la tâche ; cet identificateur peut être utilisé dans le reste du fichier pour désigner la tâche
  - taskname donne un autre nom à la tâche ; ce nom sera utilisé dans les rapports d'exécution
  - description décrit la tâche (texte non formaté)
- Les tâches optionnelles
  - nécessitent une bibliothèque supplémentaire pour être exécutées (fichier .jar à installer)

Tâches (java)

- Ant fournit des tags XML pour les tâches les plus communes en Java :
  - javac, java, rmic, javadoc, jar, unjar, war, unwar, ear

```
<javadoc
   packagenames=« com.bigmoney.pack.*"
  sourcepath="${src}"
   destdir="${doc}/api"
   use="true"/>
```

La tâche **javac**

- Compilateur utilisé : propriété **build.compiler**
  - par défaut, JDK qui exécute Ant
- Compiler récursivement tous les fichiers java du répertoire des sources
  - Utilisation des dates de dernière modification pour savoir si une classe a besoin d'être recompilée
- Très grand nombre d'attributs : **srcdir** (requis), **classpath**, **debug**, **optimize**, **source**, **fork**…

```
<javac srcdir="${src}“ destdir="${build}"
  classpath="xyz.jar" debug="on" />
```

La tâche **java**

- Lance l'exécution d'un programme java
- Attributs :
  - **classname** ou **jar** pour indiquer la classe à exécuter
  - **classpath**, **fork**, **failonerror**, **output**, **append**…

```
<java jar="dist/test.jar" fork="true"
        failonerror="true" maxmemory="128m">
    <arg value="-h"/>
    <classpath>
      <pathelement location="dist/test.jar"/>
      <pathelement path="${java.class.path}"/>
    </classpath>
</java>
```

D'autres tâches

- Système :
  - mkdir, delete, copy, move, chmod, touch, get, zip, unzip, tar, untar, gzip, gunzip
- Propriétés :
  - property donne la valeur d'une propriété

```
<property name="jaxp.jar"
          value="./lib/jaxp11/jaxp.jar"/>
```

  - available initialise une propriété si une ressource est disponible (fichier, répertoire, ressource de la JVM)

```
<available classname="fr.unice.Classe"
           property="Class.present"/>
```

D'autres tâches

- Programmation :
  - fail stoppe le processus de construction
  - ant exécute un autre fichier ant (utile s'il y a des sous-projets)
  - antcall appelle une autre cible du fichier de configuration
  - apply, exec exécute des shellscripts et des programmes externes
  - echo affiche un message sur System.out
  - mail envoie un courrier électronique
  - sql exécute une requête SQL en utilisant une source JDBC
  - ftp établit un client FTP pour transmettre des fichiers
  - junit ajoute des tâches liées à l'outil de tests JUnit (optionnel)
  - cvs exécute une commande CVS (optionnel)

Exécution de Ant

- « **`ant`** » lance Ant en utilisant
  - le fichier **`build.xml`** du répertoire courant
  - la cible par défaut
    - on peut donner une autre cible en argument
- Options
  - **`-buildfile`** pour utiliser un autre fichier que **`build.xml`**
  - **`-D`*`propriété=valeur`*** pour donner la valeur d'une propriété
  - **`-help`** affiche les options disponibles
  - **`-projecthelp`** affiche une description du projet, avec toutes les cibles (*targets*) qui ont une description

Principaux types de données

- **`property`** : pour paramétrer la construction
- **`filelist`** : liste de fichiers, sans jokers
- **`dirset`** : idem **`fileset`** pour des répertoires
- **`fileset`** : permet plus de possibilités que **`filelist`**, en particulier les **`patternset`**
- **`patternset`** : utilisent des jokers ; inclus dans **`fileset`** ou **`dirset`**
- **`filterset`** : pour remplacer des token par des valeurs
- **`path`**, **`classpath`** : pour donner des chemins tels que PATH ou CLASSPATH

Propriétés

- Chaque projet peut avoir un ensemble de propriétés qui sont utilisées comme des variables dans les attributs des tâches
- **`${prop}`** représente la valeur de la propriété ***`prop`***
- Il peut y avoir des propriétés locales ou globales (en dehors de toute cible)
- Le nom d'une propriété est de la forme **`project.name`** ou **`build.dir`**, sur le modèle des noms de propriétés Java
- Il est sensible à la casse des lettres
- 3 façons de valuer une propriété :
  - tâche **`property`**
  - tâche **`available`**
  - au lancement de Ant avec l'option –D :
    `ant -D`*`propriété`***`=`***`valeur`**`…`

Tâche **`property`**

- Plusieurs façons de donner la valeur d'une ou plusieurs propriétés
- Pour une seule propriété :
  - **`name`** et **`value`**
  - **`name`** et **`refid`**
  - **`name`** et **`location`**
- Pour plusieurs propriétés :
  - **`file`** (donne le nom d'un fichier au format des propriétés Java)
  - **`resource`** (idem **`file`** mais recherche dans le *classpath*)

Exemples

- ```
  <available classname="fr.unice.Classe"
      property="Class.present"/>
  ```

- ```
  <property name="jaxp.jar"
      value="./lib/jaxp11/jaxp.jar"/>
  <available file="${jaxp.jar}"
      property="jaxp.jar.present"/>
  ```

- ```
  <available file="/usr/local/lib" type="dir"
      property="local.lib.present"/>
  ```

- ```
  <property file="build.properties"/>
  ```

Propriétés de base

- On peut utiliser toutes les propriétés système Java données par **`System.getProperties()`** et aussi des propriétés internes à **`ant`** :
  - **`basedir`** : le chemin absolu de la racine du projet (mis par l'attribut « **`basedir`** » du tag « **`project`** »
  - **`ant.file`** : le chemin absolu du fichier de construction
  - **`ant.version`** : version de Ant
  - **`ant.java.version`** : la version de la JVM

`path` et `classpath`

- Des entrées spéciales **`path`** et **`classpath`** sont réservées aux noms ou listes de noms de fichiers
- Ces 2 entrées ont la même syntaxe
- Elles peuvent être
  - incluses dans une définition de cible
  - ou au même niveau que les propriétés globales
    - on leur donne un identificateur et on peut les utiliser dans plusieurs cibles

Classpath (ou path)

- Permet d'indiquer le *classpath* :

```
<classpath>
   <pathelement path="${classpath}"/>
   <pathelement location="lib/helper.jar"/>
</classpath>
```

peut contenir plusieurs entrées

ne peut contenir qu'une entrée

- Les éléments sont indiqués par des entrées **`pathelement`** ou **`fileset`**

Comment faire mieux ? Maven

- Regroupe des outils open source sous un chapeau commun pour gérer des projets, par exemple :
  - Ant pour la construction
  - JUnit pour le test unitaire (cf. cours suivants…)
  - Jalopy pour formater le code source
  - Checkstyle pour valider le code Java envers des standards de codage
  - Javadoc pour la doc Java
- Gère des tâches comme des rapports, des dépendances, des configurations, des releases, des distributions, etc.
- http://maven.apache.org/

Maven : principes

- Création d'un projet

```
mvn archetype:create -DgroupId=com.mycompany.app -DartifactId=my-app
```

- Structure par défaut :

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- mycompany
    |               `-- app
    |                   `-- App.java
    `-- test
        `-- java
            `-- com
                `-- mycompany
                    `-- app
                        `-- AppTest.java
```

Maven : pom.xml

- Le fichier central de toute configuration
  - Contient la majorité des informations sur le projet
  - Devient très long et très complexe (généré et modifié par interfaces graphiques)

```
<project xmlns="http://maven.apache.org/POM/4.0.0" ... >
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 ...">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Maven : phases

```
mvn archetype:create -DgroupId=com.mycompany.app -DartifactId=my-app
```

- archetype:create
  - archetype : nom du plugin => organisation en plugin avec dépendances
  - create : but (goal), similaire aux tâches ant

```
$ mvn package
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2 seconds
[INFO] Finished at: Thu Oct 05 21:16:04 CDT 2006
[INFO] Final Memory: 3M/6M
[INFO] ------------------------------------------------------------------------
```

package

- Est une phase : une étape du cycle de construction
- Le cycle de construction est une suite ordonnée de phases
- L'exécution de la phase exécute toute les phases précédentes dans l'ordre

Maven : phases (suite)

```
mvn compile
```

- Exécute les phases suivantes
  1. validate
  2. generate-sources
  3. process-sources
  4. generate-resources
  5. process-resources
  6. compile
- Phases par défaut :
  - **validate**: validation du projet et de toutes les informations nécessaires (dépendances)
  - **compile**: compilation du code source
  - **test**: test du source compilé avec un framework de test (déclaré). Ces tests ne doivent pas nécessiter que le code soit packagé ou déployé
  - **package**: création d'un package distribuable (Jar par ex.) à partir du code compilé
  - **integration-test**: déploiement le package si nécessaire dans un environnement où des tests d'intégration sont exécutés
  - **verify**: exécution de vérifications sur la validité du package ou des critères de qualité
  - **install**: installation du package dans le repository local, pour être utilisable en dépendances d'autres projets locaux
  - **deploy**: copie du package dans un repository distant pour partage
- Autres phases très utiles :
  - **clean**: nettoyage !
  - **site**: génération du site web de documentation du projet

A faire

- Préparer le TD 1 :
  - http://deptinfo.unice.fr/twiki/bin/view/Minfo/GlTD1
  - Paramétrage et lancement d'Eclipse
  - Tutoriaux de prise en main d'Eclipse