COMP0104 Software Development Practice:
ANT – A framework for program construction

Jens Krinke

Centre for Research on Evolution, Search & Testing
Software Systems Engineering Group
Department of Computer Science
University College London

Weak points of MAKE

- MAKE directly links commands and targets:
  To create a target, it is mandatory to execute specific commands.
- Life would be easier if the construction tool itself knew how to perform a specific task.

ANT (another neat tool)

- Rather than specifying commands, an ANT user specifies **tasks** that realise a specific **target**.
- Each task knows which tools and commands to use to realise the target.
- ANT comes with hundreds of predefined tasks.

System model: Buildfile

- A **Buildfile** has exactly one project.
  Every project has a **name** and a **default target**.
- Every project has one or more **targets**.
  Rather than files to be constructed,
  an ANT target refers to some general activity.
- Every target has a name and optional **dependencies**.
  The dependencies list targets
  that must be realised before the actual target.
- Every target is realised by a number of **tasks**,
  activities to be conducted to realise the target.

Example

```
<project name="SimpleProject" default="dist">
  <target name="compile">
    <mkdir dir="classes"/>
    <javac srcdir="." destdir="classes"/>
  </target>
  <target name="dist" depends="compile">
    <mkdir dir="lib"/>
    <jar jarfile="lib/simple.jar" basedir="classes"/>
  </target>
  <target name="clean">
    <delete dir="classes"/>
    <delete dir="lib"/>
  </target>
</project>
```

Example invocation of ANT

- The default target of `build.xml` is `dist`.
- The target `dist` depends on `compile`,
  so `compile` must be realised first.
- The `compile` target is realised by the
  two tasks `mkdir` and `javac`, which must be executed first.
- Now the tasks of `dist` can follow:
  `mkdir` and `jar`.
- All targets are realised – the built was successful.

Example invocation of ANT: output

```
$ ant
Buildfile: build.xml

compile:
    [mkdir] Created dir: /src/moore/classes
    [javac] Compiling 3 source files
            to /src/moore/classes
dist:
    [mkdir] Created dir: /src/moore/lib
      [jar] Building jar: /src/moore/lib/simple.jar

BUILD SUCCESSFUL

Total time: 3 seconds
$ _
```

Example with properties

```
<project name="SimpleProject" default="dist">
  <property name="dist" value="lib"/>
  <target name="compile">
    <mkdir dir="classes"/>
    <javac srcdir="." destdir="classes"/>
  </target>
  <target name="dist" depends="compile">
    <mkdir dir="${dist}"/>
    <jar jarfile="${dist}/simple.jar"/>
  </target>
  <target name="clean">
    <delete dir="classes"/>
    <delete dir="${dist}"/>
  </target>
</project>
```

Incremental builds

- With every new build, only those targets are realized whose dependencies have changed.
- Incrementality is not built into the tool, but must be implemented by the individual task.
- ANT is thus just a framework to organise the individual tasks; the intelligence of (incremental) construction is in the tasks.
- Example: the `javac` task determines the dependencies automatically.

Extending ANT

- ANT comes with a JAVA class `Task` that can be subclassed to realise new tasks.
- Existing tasks also come as Task subclasses, they can be further subclassed to create new extensions or adaptations.
- ANT can also be configured at run time.

Concepts

- The build model of ANT specifies **tasks** that realise a specific **target**.
- Rather than files to be constructed, an ANT target refers to some general activity.
- Dependencies list targets that must be realised before the actual target.
- Each task knows which tools and commands to use to realise the target.
- Incrementality is not built into the tool, but must be implemented by the individual task.