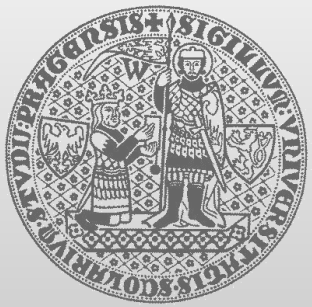
Software Building
(Sestavování aplikací)

http://d3s.mff.cuni.cz

Pavel Parízek

parizek@d3s.mff.cuni.cz



CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

What is software building

- Transforming source code (tree of files) into executable binary code
  - C/C++, C# → Win32 exe, Linux elf
  - Java, Scala → class files (bytecode)

- More generally
  - LaTeX source files → PDF documents

Requirements

- Automation
  - Minimal input from the developer
- Portability
  - Support for multiple platforms
- Efficiency
  - Process each source code file once
  - Reuse previously built fragments
- Robustness
  - Try building as much as possible
- Generality
  - Not only for a particular application
- Easy to use
  - How to easily define the build script

Challenges

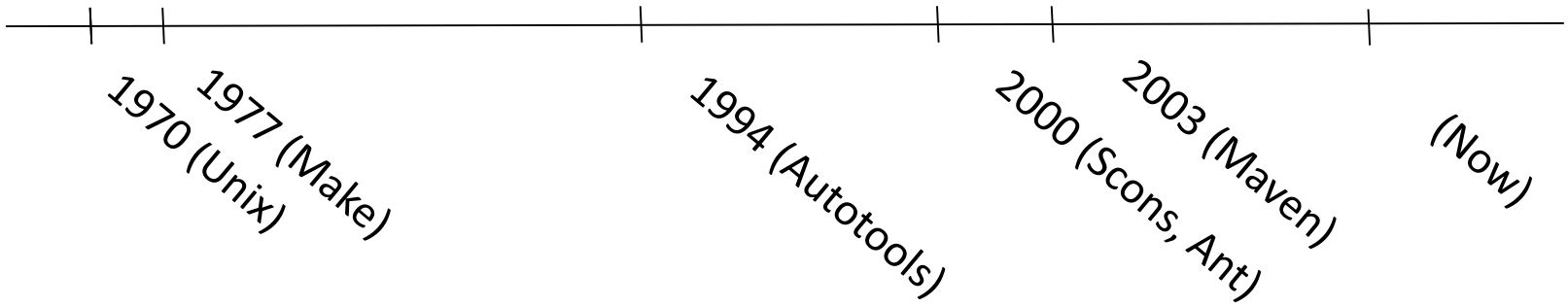
- Dependencies
  - Building files in the correct order
    - first binary object files (.o) and then executable
  - Recompile after modification
    - header file (.h) ➔ source code file (.c)
  - How to identify them properly
    - Pre-processor directives (“#include” in C)
    - Source code analysis (bytecode for Java)
    - Metadata and debug symbols in binaries
- Correct order
  - Logical dependencies between sources (.c, .java) and intermediate results (.o)

Other problems solved by build tools

- Packaging (distribution)
  - generated binary files, metadata, documentation
- Running selected unit tests
- Generating documentation

Build tools

- Java world: Ant, Maven, Gradle
- Unix (C/C++): Make, Autotools
- Windows (.NET): MSBuild, (GUI)



Ant

Ant

- Build tool mostly for Java projects
  - Wide support of tools and frameworks common in the Java world (JUnit, JSP/Servlets, EJB, ...)
- Web: http://ant.apache.org/
- Highly extensible
  - Plug-ins written in Java
- Very portable scripting
- Scripts written in XML
  - Default file name: `build.xml`

Build file structure

```
<project name="MyProject" default="dist" basedir=".">
  <property name="src.dir" value="./src"/>
  <property name="build.dir" value="./build"/>

  <target name="init">
     <mkdir dir="${build.dir}"/>
  </target>

  <target name="compile" depends="init">
      <javac srcdir="${src.dir}" destdir="${build.dir}"/>
  </target>

  <target name="clean">
     <delete dir="${build.dir}"/>
  </target>
</project>
```

Department of Distributed and Dependable Systems

Terminology

- Task
  - Specific action to be performed during the build process
    - execute the Java compiler, create new directory
- Target
  - Goal required for building (compilation, packaging, running tests, generating documentation)
  - Set of tasks that must be executed to fulfill the goal
  - May have dependencies on other targets
- Project
  - Set of targets relevant for the application
- Property
  - name-value pair (strings)
  - usage: `${prop.name}`

Dependencies between targets

- Build script

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

- Execution order

E ➔ D,C,A,E
D,C,A,E ➔ A,C,D,C,A,E
A,C,D,C,A,E ➔ A,B,C,D,C,A,E
A,B,C,D,C,A,E ➔ A,A,B,C,D,C,A,E
A,A,B,C,D,C,A,E ➔ A,A,B,C,D,A,B,C,A,E
A,A,B,C,D,A,B,C,A,E ➔ A,B,C,D,E

Department of Distributed and Dependable Systems

Basic tasks

- Compilation of Java source files

```
<javac srcdir="${src.dir}" destdir="./build"
    debug="on" deprecation="on"/>
```

- Running an external Java program

```
<java classname="myapp.Main" fork="true">
  <arg value="nswi154"/>
  <jvmarg value="-Xmx512m"/>
</java>
```

- Packaging class files in JAR archive

```
<jar destfile="myapp.jar" basedir="./build">
  <manifest>
    <attribute name="Main-Class" value="..."/>
  </manifest>
</jar>
```

Department of Distributed and Dependable Systems

Online documentation

- http://ant.apache.org/manual/index.html
  - Important concepts (properties, filesets, paths)
  - List of core tasks (javac, java, file management)

Example 1

- Download the APISign package
  - http://d3s.mff.cuni.cz/teaching/software_development_tools/files/apisign.tgz
  - Contains two simple tools for generating and verifying file signatures
  - Taken from http://docs.oracle.com/javase/tutorial/security/apisign/
- Write the build script for APISign
  - Compilation
    - All source code files written in Java
  - Packaging
    - Create an archive `sign.jar` with the main class `GenSig`
  - Execution
    - Runs the archive `sign.jar` on a file name given as a command-line argument
  - Use properties where it makes sense, typical directory layout (`./src`, `./build`), and standard targets (compile, build, init, clean, dist)
  - Specify reasonable dependencies between targets
- Alternative (preferred): use your own programs instead of APISign
- Running Ant
  - Command-line: `ant <target name>`

Department of Distributed and Dependable Systems

File management

- Tasks
  - `<mkdir>`
  - `<delete>`
  - `<copy>`
  - `<move>`

Path-like structures

```
<path id="myapp.classpath">
  <pathelement path="${classpath}"/>
  <fileset dir="lib">
    <include name="**/*.jar"/>
  </fileset>
  <pathelement location="classes"/>
  <dirset dir="${build.dir}">
    <include name="apps/**/classes"/>
    <exclude name="apps/**/*Test*"/>
  </dirset>
  <pathelement location="third_party/util.jar"/>
</path>

<javac ...>
  <classpath refid="myapp.classpath"/>
</javac>
```

This is a modified version of an example from the Apache Ant documentation

Properties defined externally

- Create the file `build.properties`

```
src.dir=./src
build.dir=./build
lib.dir=./lib
```

- … and include the file in `build.xml`

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

Example 2

- Download the CoCoME package
  - http://d3s.mff.cuni.cz/teaching/software_development_tools/files/cocome.tgz
  - It is a model of a trading system developed for research purposes
  - The package contains also few external libraries
- Alternative: use your own programs instead of CoCoME
- Write similar `build.xml` as in the first example
  - compilation, packaging, directory layout, target “clean”, properties, dependencies
- Improve the build script
  - Define classpath properly and use in tasks like <javac>
  - Use an external file to define properties

Department of Distributed and Dependable Systems D3S

Dependencies between source files

- Recompile everything from scratch
  - We can probably recommend this approach
- Use task `<depend>`
  - Deletes all obsolete .class files (modified sources)
  - Re-use of some previously compiled class files
  - Limitation: cannot discover some dependencies
  - Example

```
<depend srcdir="./src" destdir="${build.dir}"/>
```

Department of Distributed and Dependable Systems

Conditional processing of targets

- Property value is set/unset

```
<property name="my-cond" value="..."/>
<target name="..." if="my-cond">
<target name="..." unless="my-cond">
```

- File availability

```
<available file="./lib" property="have_lib"/>
<target name="copy-libs" if="have_lib">
  <copy ...>
</target>
```

Department of Distributed and Dependable Systems D3S

Boolean conditions

```
<condition property="config_debug">
  <and>
    <available file="./config"/>
    <or>
      <istrue value="debug_mode"/>
      <istrue value="testing"/>
    </or>
  </and>
</condition>
```

Explicit processing of targets

```
<target name="dist" depends="build-main,build-test">
  <antcall name="prepare-dist">
    <param name="release" value="2.1-rc3"/>
    <param name="website" value="http://..."/>
  </antcall>
</target>
```

Example 3

- Play with advanced features
  - Conditional processing of targets based on property values
  - Boolean conditions (setting properties)
  - Explicitly trigger processing of some target

Scripting

```
<parallel>
   task 1
   <sequential>
      task 2
      task 3
   </sequential>
   task 4
</parallel>
```

Scripting

- Loops
  - http://d3s.mff.cuni.cz/teaching/software_development_tools/files/ant-contrib-1.0b3.jar
  - Taken from: Ant-Contrib Tasks (http://ant-contrib.sourceforge.net)

- Example

```
<taskdef resource="net/sf/antcontrib/antlib.xml">
  <classpath>
    <pathelement location="ant-contrib-1.0b3.jar"/>
  </classpath>
</taskdef>

<for list="1,2,5,10" param="count">
  <sequential>
    <java classname="MyApp">
      <arg value="@{count}"/>
    </java>
  </sequential>
</for>
```

Department of Distributed and Dependable Systems

Other useful tasks

- Executing arbitrary system commands

```
<exec executable="cmd.exe"
      timeout="1000" output="log.txt"
      errorproperty="error.msg">
  <arg value="some_data"/>
</exec>
```

- Creating archives: `<zip>`, `<tar>`
- Setting properties that contain the current date and time

```
<tstamp/>
```

- Printing messages

```
<echo message="Error: ${error.msg}">
```

Department of Distributed and Dependable Systems

MSBuild

- XML syntax of build scripts (“Makefiles”)
- Used internally by Visual Studio 20xx-15
- Syntax evolving (non-trivial differences)
- Familiar concepts: task, target, property

- Homepage
  - https://docs.microsoft.com/cs-cz/visualstudio/msbuild/msbuild?view=vs-2015

Support for C# in Ant

- Plugin for Ant
  - http://ant.apache.org/antlibs/dotnet/index.html
- NAnt
  - https://sourceforge.net/projects/nant/
  - http://nant.sourceforge.net/

Homework

- Assignment
  - http://d3s.mff.cuni.cz/~parizek/teaching/sdt/
- Deadline
  - 29.10.2018 / 30.10.2018

- Homework assumes usage of Ant by default
  - Option: use MSBuild on FTP server
    - Find how to invoke the Java compiler

- Alternative: create `build.xml` script for your C# program of reasonable size (“zápočťák”)

Department of Distributed and Dependable Systems D3S