Laboratorium Inżynierii Oprogramowania

Automatyzacja procesu budowy oprogramowania

Apache Ant

Przygotował: dr inż. Radosław Adamus

Historia zmian

Data	*Wersja*	*Autor*	*Opis zmian*
6.11.2009	0.1	Radosław Adamus	Utworzenie dokumentu i edycja wprowadzenia
8.11.2009	0.2	Radosław Adamus	Opis Ant'a
9.11.2009	0.3	Radosław Adamus	Przygotowanie zadań laboratoryjnych
11.11.2009	0.4	Radosław Adamus	Przygotowanie zadań laboratoryjnych
12.11.2009	1.0	Radosław Adamus	Dodanie linków zewnętrznych, końcowa edycja i korekta dokumentu
13.11.2009	1.01	Radosław Adamus	Cel i rezultat laboratorium
09.12.2009	1.02	Radosław Adamus	Korekcja linków
11.12.2009	1.03	Radosław Adamus	Poprawki edycyjne
20.122009	1.04	Radosław Adamus	Poprawki edycyjne

1. Wprowadzenie

Budowanie oprogramowania oznacza tworzenie plików wykonywalnych na podstawie źródeł. Dla prostych aplikacji jest zazwyczaj prostą kompilacją. Dla bardziej skomplikowanych programów (czyli w większości rzeczywistych przypadków) proces budowy oprogramowania jest dużo bardziej złożony. Kod źródłowy jest przechowywany w wielu (np. tysiącach) plików. Proces kompilacji oraz generowania postaci wykonywalnej może być zróżnicowany w zależności od platformy docelowej, wersji, klienta, itp. Kod źródłowy może występować w wielu językach (kompilowalnych, skryptowych). W trakcie pisania kodu wykorzystywane było wiele gotowych bibliotek, które muszą być uwzględnione w procesie budowy oraz uruchamiania oprogramowania.
Automatyzacja procesu budowy oprogramowania dokonywana jest przez dedykowane narzędzia (ang. build tools). Są to aplikacje, bez których koordynacja i zarządzanie procesem budowy skomplikowanego oprogramowania byłoby praktycznie niemożliwe. Przykładami tego typu aplikacji są: GNU make[1], Apache Ant[2], Apache Maven[3] czy Scons[4]. Ze względu na to iż proces budowy może być długotrwały, ważną cechą tych aplikacji jest skracanie czasu potrzebnego na zbudowanie oprogramowania. W najprostszym wypadku jest to osiągane poprzez kompilacje tylko tych elementów systemu, które z punktu widzenia wprowadzonych zmian muszą być ponownie zbudowane. Z technicznego punktu widzenia automatyzacja procesu budowy oprogramowania oznacza najczęściej utworzenie skryptu, który automatyzuje zadania , które wytwórcy oprogramowania wykonują, w ramach swojej pracy, w sposób powtarzalny, a których celem jest zbudowanie oprogramowania. Należą do nich:

- kompilacja kodu źródłowego do postaci wykonywalnej (binarnej).
- Pakowanie kodu binarnego
- uruchamianie testów
- wdrażanie w środowisku docelowym (testowym, akceptacyjnym, produkcyjnym)
- tworzenie dokumentacji i informacji nt. wydania (ang. release notes)

2. Apache Ant

Apache Ant jest narzędziem służącym do automatyzacji procesu budowania

1 http://www.tutorialspoint.com/makefile/index.htm
2 http://ant.apache.org/
3 http://maven.apache.org/
4 http://www.scons.org/

oprogramowania. Koncepcja działania jest podobna to tej znanej z programu GNU Make, będącego aplikacją powłoki systemu Unix. Ant pozwala na automatyzację procesu budowy aplikacji napisanych w języku Java (sam również został zaimplementowany w tym języku). Do konfiguracji procesu wykorzystywane są pliki skryptowe w formacie XML.

2.1 Instalacja

Do działania programu Ant wymagane jest posiadanie przynajmniej środowiska wykonawczego Java (JRE – Java Runtime Environment[5]). Jednak jeżeli zamierzamy programować w tym języku potrzebna nam będzie wersja dla programistów (JDK – Java Development Kit[6]).
Program w wersji binarnej jest dostępny na stronie projektu Apache Ant. Po pobraniu archiwum należy je rozpakować (wraz ze strukturą folderów) w wybranym folderze (w laboratorium jest to katalog c:\tools). Po rozpakowaniu należy ustawić/uzupełnić następujące zmienne środowiskowe (dla systemu Windows):

1. ANT_HOME – zmienna wskazująca na katalog w którym znajduje się instalacja programu Ant (na laboratorium będzie to c:\tools\apache-ant-$$$, gdzie $$$ to numer wersji aplikacji)
2. PATH – należy dodać ścieżkę do katalogu bin znajdującą się wewnątrz katalogu na który wskazuje zmienna ANT_HOME.
3. Sprawdzić czy w zmiennych środowiskowych istnieje zmienna JAVA_HOME i czy poprawnie wskazuje na katalog w którym zainstalowane jest JDK Java.

Pełna instrukcja pobrania wersji binarnej oraz instalacji w systemach Windows oraz Unix dostępna jest pod adresem http://ant.apache.org/manual/install.html[7].

2.2 Skrypt konfiguracyjny (ang. buildfile)

Aby wykorzystać aplikację Apache Ant do automatyzacji procesu budowy oprogramowania należy utworzyć plik skryptu o nazwie *build.xml*. Plik ten będzie zawierał informacje o projekcie, takie jak: nazwa projektu, domyślny *target* oraz bazowy katalog projektu (ang. basedir). Sam opis projekt w pliku konfiguracyjnym składa się z tzw. targets (czyli celów), które zawierają opis zadań (ang. tasks), podlegających automatyzacji.

2.2.1 Targets

Pojedynczy *target* definiuje zadania, które mają zostać wykonane w danym kroku budowy aplikacji (np. kompilacji, testowaniu, tworzeniu pliku wykonywalnego, tworzeniu dokumentacji, etc.). Dodatkowo może on być zależny (ang. depend) od innych elementów *targets*. Zależność oznacza, iż zanim wykonane zostaną zadania z danego elementu typu *target,* najpierw wykonane muszą zostać zadania zdefiniowane na poczet tych, od których jest uzależniony. Poniższy przykład pokazuje podstawową strukturę projektu w pliku build.xml.

```
<project name="nazwa_projektu" default="nazwa1" basedir=".">

    <target name="nazwa1" depends=”nazwa2”>

        ...

    </target>


    <target name="nazwa2">
```

5 http://java.sun.com/javase/downloads/index.jsp
6 http://java.sun.com/javase/downloads/index.jsp
7 W systemie Ubuntu wystaczy wydać polecenie *sudo apt-get install ant*

```
        ...
    </target>
    ...
</project>
```

Hierarchiczna budowa treści (czyli: zagnieżdżania elementów/tagów XML) pliku w formacie XML odpowiada powyższemu opisowi struktury. Podstawowy element <project> reprezentuje projekt (wraz z odpowiednimi atrybutami) i składa się z elementów <target> zawierających (nie pokazane w przykładzie) zadania (elementy <task>). Każdy *target* posiada unikatową nazwę (atrybut *name*) oraz może być zależny od innych *targets* (atrybut *depends*).

2.2.2 Uruchomienie skryptu:

Uruchomienie skryptu z linii poleceń odbywa się poprzez wywołanie polecenia *ant* w katalogu, w którym znajduje się plik build.xml:

```
user@system:~/temp$ ant
```

Domyślne działanie powoduje uruchomienie zadań znajdujących się w domyślnym *target* zawartym w pliku build.xml bieżącego katalogu (jeżeli domyślny *target* nie został zdefiniowany, poszukiwany jest *target* o ustalonej nazwie 'run'). Wszelkie zmiany domyślnego działania możliwe są poprzez dodanie parametrów. Pełna ich lista wyświetlona zostanie po wywołaniu polecenia z parametrem '-h':

```
user@system:~/temp$ ant -h
```

2.2.3 *Tasks* czyli zadania

Zadanie (ang. task) jest kawałkiem kodu, który może zostać wykonany. Ant posiada wiele wbudowanych zadań (tzw. podstawowych oraz opcjonalnych). Istnieje również możliwość napisania własnego. Każde zadanie może posiadać zestaw atrybutów będących argumentami zadania, Struktura składni XML zadania jest następująca:

```
<taskname attribute1="value1" attribute2="value2" ... />
```

Pełna lista wbudowanych zadań dostępna jest w dokumentacji Ant'a dostępnej pod adresem: http://ant.apache.org/manual/tasklist.html

2.2.4 *Properties* czyli własności

Projekt może definiować listę właściwości (ang. properties), które umożliwiają zdefiniowanie zmiennych konfiguracji na zasadzie klucz-wartość. Ant udostępnia zestaw wbudowanych właściwości oraz umożliwia definiowanie własnych. Każda własność definiująca zmienna posiada atrybut *name* definiujący nazwę zmiennej oraz jej wartość (jako *value*, *location* lub *refid*).

2.3 Przykładowy plik konfiguracyjny Ant [8]

```
<project name="MyProject" default="dist" basedir=".">
    <description>
        przykładowy plik automatyzacji budowy
    </description>
  <!-- ustawienie globalnych właściwości -->
  <property name="src" location="src"/>
  <property name="build" location="build"/>
  <property name="dist"  location="dist"/>

  <target name="init">
    <!-- utworzenie stempla czasowego -->
    <tstamp/>
    <!-- utworzenie struktury katalogowej dla przechowywania rezultatów procesu
      kompilacji-->
    <mkdir dir="${build}"/>
  </target>

  <target name="compile" depends="init"
        description="kompilacja źródeł" >
    <!-- skompilowanie kodu Java z ${src} do ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>

  <target name="dist" depends="compile"
        description="generacja dystrybucji" >
    <!-- utworzenie katalogu dla przechowywania dystrybucji -->
    <mkdir dir="${dist}/lib"/>

    <!-- Wstawienie całej zawartości ${build} do pliku archiwum o nazwie
MyProject-${DSTAMP}.jar  -->
    <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
  </target>

  <target name="clean"
        description="sprzątanie" >
    <!-- usuwa drzewa katalogów ${build} oraz ${dist} -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
</project>
```

Na uwagę zasługuje szczególnie wykorzytanie atrybutu depends umożliwiajace kaskadowe wykonanie poszczególnych targets i tym samym wywołanie całego procesu pracy (ang. workflow) powodującego zbudowanie aplikacji. Skrypt wykorzystuje również zmienne definiowane z wykorzystaniem tag'u *<property>*. Każda zmienna ma swoją unikatową nazwę. Wartosciami zmiennych są nazwy podfolderów. Składnia użycia zmiennej (znajdująca swoje zastosowanie w powyższym przykładzie) jest następująca: *${nazwa_zmienej}*.

Pełna dokumentacja Ant'a znajduje się na stronie: http://ant.apache.org/manual/index.html

3. „Automatyzacja automatyzacji”

Powyższy opis bardzo skrótowo przybliża podstawy automatyzacji za pomocą przykładowego narzędzia. Należy jednak pamiętać, że rzeczywiste i praktyczne wyzwania są dużo bardzie skomplikowanie. Każdy projekt aplikacji korzysta przecież z komponentów ponownego użycia (np. w postaci bibliotek), które należy zlokalizować, pobrać i umieścić w strukturze projektu oraz

8Na podstawie dokumentacji Ant'a: http://ant.apache.org/manual/using.html#example

samym skrypcie. Może to się dla skomplikowanych aplikacji stać prawdziwym wyzwaniem. Dlatego sama automatyzacja zależnościami oprogramowania jest również potrzebna. Ant posiada sub-projekt o nazwie Ivy[9], który na taką automatyzację pozwala. Inny projekt organizacji Apache o nazwie Maven[10] udostępnia zunifikowane repozytoria elementów ponownego użycia oraz skryptową formę opisu zależności i automatyzację procesu pobierania (i jest również młodszym odpowiednikiem Ant'a).
Należy również zwrócić uwagę, iż pomimo że Ant czy inne tego typu narzędzia automatyzują proces budowy oprogramowania to samo ich uruchomienie musi być harmonogramowane niezależnie. Polityka organizacji może zakładać częstotliwość tworzenia build'ów oprogramowania a harmonogramowaniem mogą zajmować się wyspecjalizowane narzędzia takie jak Cruise Control[11], Apache Continuum[12] czy Hudson-ci[13].

4. Laboratorium:

4.1 Cel i rezultat

Celem laboratorium jest:

- zapoznanie się z podstawową strukturą skryptu programu Ant.
- zapoznanie się ze sposobem uruchamiania skryptów programu Ant
- zdobycie umiejętności samodzielnego pisanie prostych skryptów programu Ant

Po ukończeniu laboratorium student:

- potrafi sprawdzić czy program jest poprawnie zainstalowany w systemie
- potrafi opisać podstawową strukturę pliku skryptowego programu Ant
- potrafi uruchomić skrypt programu Ant i określić poprawność jego wykonania
- sprawnie posługuje się pomocą programu dostępną online
- potrafi wprowadzać zmiany w skrypcie poprzez dodanie nowych zadań budowy.

4.2 Zadania

1. Sprawdź czy Ant jest poprawnie zainstalowany. Jeżeli nie to wykonaj czynności zgodnie z opisem w rozdziale 2.1 Instalacja.
2. Pobierz strukturę przykładowego projektu o nazwie hello-ant z repozytorium znajdującym się pod adresem https://dev.kis.p.lodz.pl:8443/svn/samples/trunk/pio/hello-ant
3. Zapoznaj się z zawartością pliku build.xml znajdującego się w głównym folderze projektu oraz z fizyczną strukturą katalogów projektu.
4. Z poziomu linii komend systemu przejdź do głównego katalogu projektu i wywołaj polecenie *ant.* Poprawny rezultat powinien wyglądać mniej więcej tak jak poniżej:

9 http://ant.apache.org/ivy/http://ant.apache.org/ivy/
10 http://maven.apache.org/
11 http://cruisecontrol.sourceforge.net/
12 http://continuum.apache.org/
13 http://hudson-ci.org/

```
Buildfile: build.xml

init:
    [mkdir] Created dir: /home/radek/workspace/hello-ant/build

compile:
    [javac] Compiling 1 source file to /home/radek/workspace/hello-ant/build

run:
     [java] message : hello ant !

BUILD SUCCESSFUL
Total time: 1 second
```

5. Wywołaj polecenie *ant* a argumentem *clean.* Zaobserwuj zmiany w strukturze katalogów projektu.
6. Zmodyfikuj skrypt build.xml dodając jeszcze jeden krok w procesie budowy oprogramowania – tworzenie dystrybucji w postaci biblioteki. Jego celem będzie utworzenie w nowym katalogu o nazwie *dist* (który ma się znajdować bezpośrednio w głównym katalogu projektu) archiwum jar (akronim: Java archive), którego zawartością będzie skompilowany kod projektu oraz nadanie mu nazwy według wzorca [nazwa_projektu]-[stempel_czasowy].jar. Dodaj do niego manifest informujący o klasie, które będzie punktem startowym aplikacji (wymagane dla wykonania zadania 7).

   WSKAZÓWKI: wykorzystaj *ant task* o nazwie <*jar*> (http://ant.apache.org/manual/Tasks/jar.html). Manifest potrzebny do utworzenia archiwum znajduje się w katalogu *src* projektu a jego nazwa dostępna jest w skrypcie początkowym za pomocą zdefiniowanej zmiennej o nazwie *manifest.file*. Przyjrzyj się jeszcze raz przykładowi: 2.3 Przykładowy plik konfiguracyjny Ant 8. Nazwa obecnie przetwarzanego projektu znajduje się w zmiennej wbudowanej o nazwie *ant.project.name.*

7. Zmodyfikuj target *run,* tak aby uruchamiał aplikacji z wykorzystaniem archiwum zamiast tak jak dotychczas skompilowanego kodu w folderze *build.*

   WSKAZÓWKA: wykorzystaj task <java> z atrybutami jar i fork.

Rezultat poprawnie wykonanych zadań 6 i 7 powinien dać wynik zbliżony do poniższego.

```
Buildfile: build.xml

init:
    [mkdir] Created dir: /home/radek/workspace/hello-ant/build
    [mkdir] Created dir: /home/radek/workspace/hello-ant/dist

compile:
    [javac] Compiling 1 source file to /home/radek/workspace/hello-ant/build

dist:
      [jar] Building jar: /home/radek/workspace/hello-ant/dist/hello-ant-20091109.jar

run:
     [java] message : hello ant !

BUILD SUCCESSFUL
Total time: 1 second
```

8. ZAAWANSOWANE Pobierz z projekt hello-lib-ant z repozytorium znajdującym się pod

adresem https://dev.kis.p.lodz.pl:8443/svn/samples/trunk/pio/hello-lib-ant. Jest to projektu, który do poprawnej kompilacji oraz wykonania potrzebuje dodatkowych bibliotek (istniejący skrypt build.xml nie wykona się poprawnie). Biblioteki są już przygotowane i znajdują się w katalogu *lib* projektu. Zmodyfikuj skrypt w taki sposób aby poprawnie wykonana została kompilacja oraz uruchomienie programu.

WSKAZÓWKI: Zapoznaj się z *task*'ami: *classpath* oraz *fileset* (oraz opcjonalnie *path* i *classpathref*).

9. ZAAWANSOWANE Podobnie jak w zadaniu 6. zmodyfikuj skrypt poprzez dodanie kroku budującego dystrybucję (pamiętaj, że dystrybucja powinna posiadać swoje kopie wymaganych do uruchomienia bibliotek).

Poprawne wykonanie zadanie 8 i 9 powinno dać wynik zbliżony do poniższego:

```
Buildfile: build.xml

init:
    [mkdir] Created dir: /home/radek/workspace/hello-lib-ant/build
    [mkdir] Created dir: /home/radek/workspace/hello-lib-ant/dist
    [mkdir] Created dir: /home/radek/workspace/hello-lib-ant/dist/lib

compile:
    [javac] Compiling 1 source file to /home/radek/workspace/hello-lib-ant/build

dist:
      [jar] Building jar: /home/radek/workspace/hello-lib-ant/dist/hello-lib-ant-
20091111.jar
     [copy] Copying 3 files to /home/radek/workspace/hello-lib-ant/dist/lib

run:
     [java] standard message : hello lib ant!
     [java] capitalized by org.apache.commons.lang.WordUtils : Hello Lib Ant!

BUILD SUCCESSFUL
Total time: 1 second
```