

## Tworzenie aplikacji mobilnych pod Linuxem

# W ruchu – Wireless Toolkit

TOMASZ RYBICKI

Urządzenia mobilne, a w szczególności telefony komórkowe, cieszą się coraz większą popularnością. Producenci oferują całą gamę narzędzi do tworzenia aplikacji na ich urządzenia, jednakże są to przeważnie programy działające pod Windows. W niniejszym artykule opiszę krok po kroku instalację i konfigurację środowiska do tworzenia aplikacji mobilnych pod Linuxem (Mandrake 9.2) opartego na SUN Wireless Toolkit i edytorze Eclipse.

Aby tworzyć aplikacje mobilne wystarczy zainstalować Wireless Toolkit (WTK) – środowisko do uruchamiania i testowania programów. WTK, oferując wyrafinowane sposoby wszechstronnego testowania aplikacji, nie wspiera jednak procesu tworzenia – uprzedni napisany program należy skopiować do odpowiedniego podkatalogu WTK. Dopiero wtedy możliwe jest jego kompilowanie, testowanie i uruchamianie w jednym z „wbudowanych” standardowych emulatorów. Ponieważ proces tworzenia programu często wymaga wielokrotnej kompilacji i nie jest przeważnie wolny od błędów, proces kopiowania kolejnych, poprawionych czy też rozszerzonych wersji kodu, staje się niezmiernie uciążliwy. Kilka prostych czynności pozwala skonfigurować dowolne środowisko programistyczne (pokażemy to na przykładzie Eclipse) do pracy z J2ME, co ułatwi i przyspieszy pisanie oraz skróci proces debugowania programu i uczyni te czynności możliwie bezbolesnymi.

Nasze środowisko opierać się będzie na dwóch filarach: wspomnianym już wcześniej Wireless Toolkit oraz cieszącym się coraz większą popularnością Eclipse. Obydwa programy wymagają obecności w systemie interpretera języka Java.

## Krok pierwszy: instalujemy Javę

Ze strony <http://java.sun.com> ściągamy pakiet instalacyjny Java. W naszym przypadku jest to `j2sdk-1_4_2_02_linux-i586.rpm`. Wystarczy

wpisać (jako root oczywiście):

```
rpm -i j2sdk-1_4_2_02_02_
linux-i586.rpm
```

i po chwili w `/usr/java/j2sdk1.4.2_02` mamy zainstalowaną Javę. Kolejnym krokiem jest dodanie ścieżki do polecenia java w pliku `.bash_profile` (Listing 1).

Następnie należy się wylogować i zalogować ponownie (plik `.bash_profile` wczytywany jest przy logowaniu) oraz sprawdzić, czy nie pomyliliśmy się w ścieżce wpisując w linii poleceń

```
java
```

Efekt powinien być podobny do tego na Rysunku 1.

## Krok drugi: instalujemy Eclipse

Pakiet Eclipse można ściągnąć z <http://www.eclipse.org>. – wykorzystamy wersję 3.0M2. Instalacja polega na rozpakowaniu ściągniętego archiwum. Po wykonaniu polecenia

```
unzip eclipse-SDK-3.0M2-2
linux-gtk.zip -d /usr
```

w katalogu `/usr` zostanie utworzony katalog `eclipse` zawierający program. Przy pierwszym uruchomieniu Eclipse tworzy sobie katalog `workspace`, gdzie będą przechowywane źródła tworzonych aplikacji. Katalog ten zostanie utworzony w katalogu bieżącym, tzn. jeżeli uruchomimy Eclipse będąc w katalogu `/usr/eclipse`, to program spróbuje utworzyć katalog `workspace` właśnie tam (`/usr/eclipse/workspace`), co może zaowo-



Rysunek 1: Wynik wywołania polecenia java na konsoli.

wać przerażającym i nie do końca zrozumiałym komunikatem o błędzie (Rysunek 2) – oznaczającym tak naprawdę tylko tyle, że nie można utworzyć katalogu `workspace` w zadanej lokacji (na przykład z powodu braku uprawnień). Dopiero komenda

```
/usr/eclipse/eclipse
```

wywołana z katalogu domowego (w moim przypadku `/home/tomek`) powoduje poprawne uruchomienie programu. Przy pierwszym uruchomieniu wyświetlana jest informacja o konfigurowaniu środowiska (Rysunek 3), po czym uruchomi się Eclipse (Rysunek 4). Jeżeli zamiast tego ujrzymy komunikat z Rysunku 5,

### Listing 1: Plik `.bash_profile` z dodaną ścieżką do interpretera Java

```
# .bash_profile

# User specific environment and startup programs

PATH=$PATH:$HOME/bin:/usr/java/j2sdk1.4.2_02/bin

export PATH
unset USERNAME
```



Rysunek 2: Wynik wywołania eclipse z katalogu /usr/eclipse/

oznacza to, że źle podaliśmy ścieżkę do javy w .bash\_profile.

Mamy już zainstalowany i skonfigurowany edytor Eclipse i można pisać oraz uruchamiać programy w „standardowej” edycji Javy (J2SE). Teraz czeka nas

### Krok trzeci: instalacja Wireless Toolkit

Wireless Toolkit jest darmowym programem do tworzenia aplikacji mobilnych. Jest dostępny do ściągnięcia ze strony <http://java.sun.com>. Ściągamy j2me\_wireless\_toolkit-2\_0\_01-linux-i386.bin i uruchamiamy program. Wyświetli się pytanie o akceptację warunków licencji, następnie program instalacyjny zapyta o:

- ścieżkę do interpretera java (patrz krok pierwszy),
- katalog, do którego mają zostać skopiowane pliki.

Po potwierdzeniu ustawień rozpocznie się instalacja:

#### Extracting the installation



Rysunek 5: Wynik wywołania eclipse przy niewłaściwie skonfigurowanej Javie, np. źle podanej ścieżce do interpretera java.



Rysunek 3: Komunikat towarzyszący pierwszemu uruchomieniu Eclipse, kiedy to tworzony jest workspace.

```
files...
Documentation for the J2ME
Wireless Toolkit version 2.0_01
is in the file
/home/tomek/WTK2.0/index.html
In order to start using the J2ME
Wireless Toolkit, please run
/home/tomek/WTK2.0/bin/ktoolbar
```

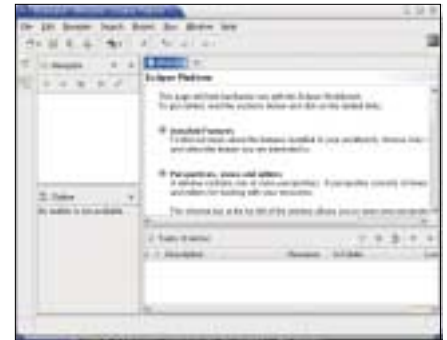
Po uruchomieniu programu naszym oczom ukaże się znany z Windows interfejs Wireless Toolkit.

Wireless Toolkit organizuje pliki w projekty. Projekty zapisywane są w katalogu apps, który znajduje się w katalogu z zainstalowanym WTK. Struktura każdego projektu jest identyczna i składa się z następujących katalogów:

- bin – zawiera wygenerowane przez WTK pliki .jar i .jad, a także plik symulujący schowek (persistent storage)
- classes – tutaj zapisywane są pliki .class
- lib – tu należy umieścić wszelkie biblioteki zewnętrzne wykorzystywane przez naszą aplikację
- res – katalog na wszelkiego rodzaju „zasoby” aplikacji: obrazki, dźwięki, itp.
- src – zawiera źródła aplikacji (pliki .java)
- tmp – katalog na dane tymczasowe
- tmpclasses – katalog na dane tymczasowe

Praca z WTK wymaga uprzedniego przekopiowania odpowiednich plików we właściwe miejsca (do odpowiednich katalogów). Kompilacja, przeweryfikacja, obfuskacja, tworzenie plików .jar i jad odbywa się (pół)automatycznie.

Mamy już zainstalowane podstawowe komponenty systemu. Działa zarówno Eclipse, jak i Wireless Toolkit. Teraz sprawimy, że Eclipse „zauważy” J2ME, co pozwoli wykorzystywać jego mechanizmy (znajdowanie błędów pod-



Rysunek 4: Ekran powitalny Eclipse.

czas pisania, podpowiadanie nazw funkcji, itp) także przy tworzeniu MIDletów.

### Krok czwarty: konfiguracja projektu w Eclipse

Tworzymy projekt – Aplikacje\_Komorkowe (to ważne, żeby w nazwie nie było znaków specjalnych, spacji ani polskich znaków – w przeciwnym wypadku skrypty Ant, które będziemy wykorzystywać, mogą mieć problemy ze ścieżkami), a następnie we właściwościach projektu dopisujemy plik midpapi.zip do ścieżki kompilacji (Project->Properties->Build Path->Libraries->Add External JARs). Plik midpapi.zip zawiera bibliotekę J2ME i jest częścią WTK. Znajduje się w katalogu lib, będącym podkatalogiem katalogu, w którym znajduje się Wireless Toolkit (w moim przypadku jest to /home/tomek/WTK2.0/lib). Rysunek 6 pokazuje dodawanie midpapi.zip do ścieżki kompilacji w oknie właściwości projektu Aplikacje\_Komorkowe.



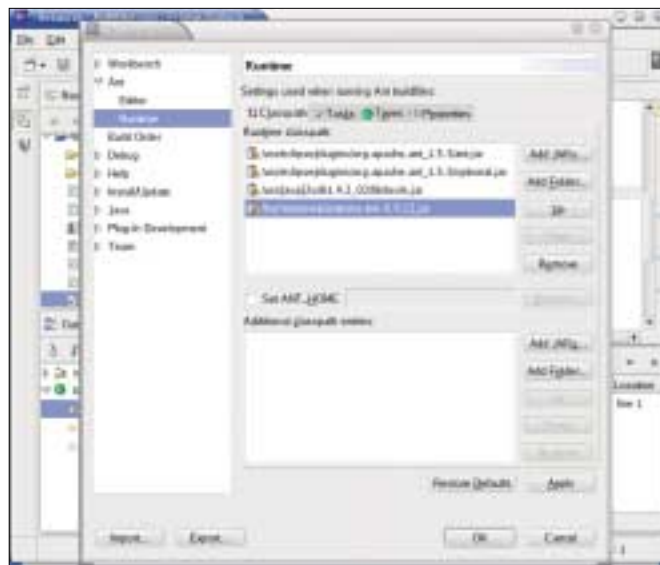
Rysunek 6: Dodawanie biblioteki MIDP (midpapi.zip) do ścieżki kompilacji we właściwościach projektu środowiska Eclipse.

AUTOR

Tomasz Rybicki jest studentem piątego roku Wydziału Elektroniki i Techniki Informatycznych Politechniki Warszawskiej, zajmuje się wykorzystaniem telefonów komórkowych do obliczeń rozproszonych. Od roku prowadzi warsztaty z J2ME. Kontakt – trybicki@autograf.pl



Rysunek 8: Uruchomienie MIDletu w standardowym emulatorze WTK z poziomu platformy Eclipse.



Rysunek 7: Konfigurowanie Ant w celu umożliwienia korzystania z zadań Ant w skryptach Ant platformy Eclipse.

Jest już możliwe wygodne pisanie programów, lecz w dalszym ciągu do ich uruchamiania konieczne jest wykorzystanie WTK. Aby tego uniknąć, można wykorzystać skrypty Ant – pliki XML będące jawowym odpowiednikiem znanego z C i C++ makefile. Skrypty Ant wykorzystywane w Eclipse mają postać plików build.xml, znajdujących się w katalogach projektów (prosty skrypt, zawierający zadanie kopiowania plików projektu do określonego katalogu WTK, znajduje się na Listingu 2).

Rozwiązanie takie upraszcza nieco proces tworzenia aplikacji, ale w dalszym ciągu trzeba korzystać jednocześnie z dwóch programów – piszemy aplikację w Eclipse, po czym kompilujemy ją, wykonujemy skrypt Ant, który kopiuje za nas pliki, i uruchamiamy pod Wireless Toolkit. Ideałem byłoby, gdybyśmy wszystko, od napisania kodu, poprzez kompilację, aż do jego uruchomienia w emulatorze mogli zrealizować pod Eclipse, a najlepiej, żeby odbywało się to po naciśnięciu jednego przycisku.

Takie możliwości daje Antenna. Jest to zestaw predefiniowanych zadań umieszczanych w skryptach Ant, działających w oparciu o Wireless Toolkit.

## Krok 5: Antenna

Pakiet Antenna można pobrać z <http://antenna.sourceforge.net>. Dostępny jest zarówno pakiet zawierający źródła, jak i wersja binarna. Nas będzie interesować to drugie. Ściągnięty plik JAR (antenna-bin-0.9.12.jar) dodajemy do 'ścieżki kompilacji' modułu Ant platformy Eclipse (menu Window->Prefe-

rences->Ant->Runtime->Classpath->Add JAR's) – ilustruje to Rysunek 7.

Antenna zawiera 7 predefiniowanych zadań, które można umieszczać w skryptach Ant platformy Eclipse. Są to następujące zadania:

- kompilacja
- wstępna weryfikacja
- obfuskacja
- tworzenie deskryptora (plik .jad)
- pakowanie aplikacji w archiwum (plik .jar)
- uruchomienie jednym z emulatorów dostępnych w Wireless Toolkit
- przekonwertowanie archiwum (pliku .jar) do formatu PalmOS (plik .prc)

Listing 3 zawiera przykładowy skrypt Ant zawierający zadania Anteny. Skrypt wy-

maga, aby bieżący projekt zawierał katalog res, w którym umieszczone będą zasoby aplikacji (obrazki, dźwięki, itp). Wyniki kompilacji (pliki .class) zostaną zapisane w podkatalogu classes (jeżeli go nie ma, to zostanie utworzony).

Po skonfigurowaniu Anteny możliwe jest de facto uruchamianie WTK z poziomu Eclipse. W ten sposób do napisania, zdebugowania, przetestowania i uruchomienia naszej aplikacji komórkowej w emulatorze możemy wykorzystać Eclipse (Rysunek 8). Nie ma potrzeby uruchamiania żadnych dodatkowych programów, wszystko jest wykonywane z poziomu naszego IDE. I o to chodziło. ■

## Listing 2: Prosty skrypt Ant kopiujący pliki z katalogu dir.src do dir.dest z pominięciem plików \*.class i \*.xml

```
<project name='Aplikacje_Komorkowe' default='copy'>

  <property name='dir.src' value='/home/tomek/Aplikacje_Komorkowe/' />
  <property name='dir.dest' value='/home/tomek/WTK2.0/apps/Aplikacje_
Komorkowe/src' />

  <target name='copy' >
    <copy todir='${dir.dest}'>
      <fileset dir='${dir.src}'>

        </fileset>
      </copy>
    </target>

  </project>
```

## Listing 3: Skrypt Ant wykorzystujący zadania Antenny

```

<?xml version='1.0'?>
<project name='Aplikacje_Komorkowe' default='build'
  basedir='.'>
  <property name='wtk.home' value='/home/tomek/WTK2.0'/>
  <property name='smtk.home' value='/usr/java/
java1.4.2_02'/>

  <!-- Nazwa MidleSuite i katalog w ktorym sie znajduje -->

  <property name='midlet.name' value='MyMIDlet'/>
  <property name='midlet.home' value='.'/>

  <!-- WAŻNE! DEKLARACJE ZADAŃ! -->

  <taskdef name='wtkjad' classname='de.pleumann.
antenna.WtkJad'/>
  <taskdef name='wtkbuild' classname='de.pleumann.
antenna.WtkBuild'/>
  <taskdef name='wtkpackage' classname='de.pleumann.
antenna.WtkPackage'/>
  <taskdef name='wtkmakeprc' classname='de.pleumann.
antenna.WtkMakePrc'/>
  <taskdef name='wtkrun' classname='de.pleumann.
antenna.WtkRun'/>
  <taskdef name='wtkpreverify' classname='de.pleumann.
antenna.WtkPreverify'/>
  <taskdef name='wtkobfuscate' classname='de.pleumann.
antenna.WtkObfuscate'/>

  <target name='clean'>
  <delete failonerror='false' dir='classes'/>
  <delete failonerror='false' dir='tmpclasses'/>
  <delete failonerror='false'>
  <fileset dir='.'>
  <exclude name='build.xml'/>
  </fileset>
  </delete>
  </target>

  <target name='build'>

  <!-- Tworzenie pliku deskryptora, jego parametry
(pola) określone są przez atrybuty tagu wtkjad -->
  <wtkjad jadfile='${midlet.name}.jad'
  name='Sun Samples - Demos'
  vendor='Sun Microsystems'
  version='1.0.3'>

  <midlet name='MyMIDlet'
  class='MyMIDlet'/>

  </wtkjad>

  <mkdir dir='classes'/>

  <!-- Kompilacja źródeł: pliki .class skopiowane
zostaną do utworzonego w poprzednim kroku katalogu
classes-->

  <wtkbuild srcdir='${midlet.home}'
  destdir='classes'
  preverify='false'/>

  <!-- Tworzenie pliku archiwum (.jar). Wykorzystywany
jest stworzony uprzednio plik deskryptora. Dodatkowo
archiwum jest wstępnie weryfikowane. Aby wykorzystać
obfuscatora, należy nadać wartość true odpowiedniemu
atrybutowi – UWAGA – wymagany jest obfuscator
Atrybut autoversion jest funkcją autoinkrementującą
atrybut MIDlet-Version
-->
  <wtkpackage jarfile='${midlet.name}.jar'
  jadfile='${midlet.name}.jad'
  obfuscate='false'
  autoversion='true'>

  <!-- Wskazanie, które katalogi mają zostać dodane do
pliku .jar – zostanie przeszukany katalog classes
i res bieżącego projektu
-->
  <fileset dir='classes'/>
  <fileset dir='${midlet.home}/res'/>

  </wtkpackage>

  <!-- Brak zainstalowanego obfuscatora, dlatego
opcja zakomentowana -->

  <!--wtkobfuscate jarfile='${midlet.name}.jar'
jadfile='${midlet.name}.jad'-->

  <!-- Preweryfikacja -->

  <wtkpreverify jarfile='${midlet.name}.jar'
jadfile='${midlet.name}.jad'/>

  <!-- Konwertowanie pliku .jar do pliku .prc (PalmOs) -->

  <!--wtkmakeprc jadfile='${midlet.name}.jad'
prcfile='${midlet.name}.prc'-->

  <!-- Uruchomienie w domyślnym emulatorze Wireless
Toolkit -->

  <wtkrun jadfile='${midlet.name}.jad' device=
'DefaultColorPhone' wait='true'/>

  </target>

</project>

```