AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki
Katedra Informatyki

PROJEKT INŻYNIERSKI

PLATFORMA DO AUTOMATYCZNYCH AKTUALIZACJI OPROGRAMOWANIA NA URZĄDZENIACH ZDALNYCH. - DOKUMENTACJA UŻYTKOWNIKA

PRZEMYSŁAW DĄBEK, ROMAN JANUSZ
TOMASZ KOWAL, MAŁGORZATA WIELGUS

OPIEKUN:
dr inż. Wojciech Turek

Kraków 2012

OŚWIADCZENIE AUTORA PRACY

OŚWIADCZAM, ŚWIADOMY(-A) ODPOWIEDZIALNOŚCI KARNEJ ZA POŚWIADCZENIE NIEPRAWDY, ŻE NINIEJSZY PROJEKT WYKONAŁEM(-AM) OSOBIŚCIE I SAMODZIELNIE W ZAKRESIE OPISANYM W DALSZEJ CZĘŚCI DOKUMENTU I ŻE NIE KORZYSTAŁEM(-AM) ZE ŹRÓDEŁ INNYCH NIŻ WYMIENIONE W DALSZEJ CZĘŚCI DOKUMENTU.

. . . . . .

PODPIS

1. Instalacja erlangowego węzła wbudowanego na urządzeniu zdalnym

1.1. Przygotowanie środowiska docelowego

Przed zainstalowaniem węzła erlangowego na systemie docelowym, konieczna jest jego konfiguracja. Obejmuje to kilka założeń i czynności:

- Docelowym systemem musi być system Linux wyposażony w menedżer pakietów dpkg. Zalecane jest wykorzystanie bardziej wysokopoziomowych narzędzi, np. apt'a. Konieczna jest konfiguracja repozytoriów.
- Węzeł erlangowy uruchamiany jest przez skrypty zarządzające podczas instalacji lub aktualizacji odpowiednich pakietów .deb. Stąd, jest on uruchamiany jako użytkownik root.
- Konieczna jest konfiguracja aplikacji klienckiej platformy do automatycznych aktualizacji w celu jej poprawnego działania.

1.2. Konfiguracja apt'a

Konfiguracja menedżera pakietów sprowadza się do dodania repozytorium platformy do automatycznych aktualizacji. W tym celu można skorzystać z dostępnych w systemie graficznych konfiguratorów lub wprowadzić nowe repozytorium bezpośrednio do pliku konfiguracyjnego /etc/apt/sources.list. W tym celu należy w nim dopisać linię:

```
deb http://adresplatformy/repository/binary /
```

1.3. Instalacja węzła

Po skonfigurowaniu menedżera pakietów i umieszczeniu zestawu pakietów dla węzła erlangowego w repozytorium platformy, jest on już gotowy do instalacji. Za pomocą apt'a można zrobić to przy uzyciu komend (z prawami root'a):

```
apt-get update
apt-get install nazwa_pakietu_glownego_wezla
```

Podczas instalacji węzeł powinien zostać automatycznie uruchomiony. Może jednak nie działać poprawnie, jeśli nie został uprzednio (tj. przed zbudowaniem paczek .deb) skonfigurowany.

1.4. Uruchamianie i zatrzymywanie węzła

Do wykonywania podstawowych operacji na węźle służy skrypt wygenerowany przez narzędzie rebar o nazwie takiej samej jak nazwa węzła, znajdujący się w katalogu bin węzła. Może on być wywoływany z następującymi komendami:

- `start` - uruchamia węzeł w tle (bez powłoki)

- `stop` - zatrzymuje węzeł
- `restart` - restartuje węzeł (wszystkie aplikacje), bez restartowania samego procesu
- `reboot` - całkowicie restartuje węzeł (razem z procesem)
- `ping` - sprawdza, czy węzeł działa
- `console` - uruchamia węzeł w trybie konsoli (z powłoką)
- `console_clean` - uruchamia węzeł z powłoką używając skryptu bootującego `start_clean` (najczęściej oznacza to uruchomienie tylko aplikacji `kernel` i `stdlib`)
- `attach` - podłącza konsolę do działającego węzła

1.5. Konfiguracja aplikacji klienckiej sup_beagle

Konfiguracja aplikacji klienckiej zawarta jest w pliku `priv/sup_beagle.config` w katalogu aplikacji klienckiej, tj. `lib/sup_beagle-X.X`. Najlepiej dokonać konfiguracji przed zbudowaniem pakietu. Można to zrobić również po instalacji węzła na urządzeniu zdalnym, jednak może to wymagać restartu węzła. Dostępne są następujące opcje konfiguracyjne:

- `management_host` - adres serwera platformy automatycznych aktualizacji
- `management_port` - port serwera platformy automatycznych aktualizacji (domyślnie 5678)
- `periodic_notify_interval` - odstęp (w sekundach) pomiędzy periodycznymi powiadomieniami z urządzenia zdalnego do serwera (inicjacje sesji)
- `upgrade_command` - komenda wywoływana podczas aktualizacji węzła. Przykładowo, dla apt'a komendą tą może być:

```
apt-get update && apt-get -y --force-yes install \
nazwa_pakietu_glownego_wezla
```

2. Interfejs użytkownika

2.1. Instalacja i uruchomienie

Jedynym wymaganiem jest zainstalowana dystrybucja Erlanga. Zalecana R14B03 lub nowsza.

Należy pobrać kod z repozytorium, a następnie przejść do katalogu `server` należy wykonać polecenie `make all`, które pobierze wszystkie zależności i skompiluje moduły erlangowe wchodzące w skład aplikacji. Po pomyślnym zakończeniu procesu kompilacji należy uruchomić aplikację - służy do tego skrypt `./start-dev.sh`. Domyślna konfiguracja uruchamia serwer www (*mochiweb*) na porcie 8080.

Po poprawnym uruchomieniu, pod adresem `http://localhost:8080/` powinna pojawić się poniższa strona:

2.2. Zarządzanie urządzeniami i oprogramowaniem

2.2.1. Urządzenia

Podstrona *Devices* jest odpowiedzialna za zarządzanie urządzeniami zapisanymi w bazie danych. *Filter devices by category* filtruje urządzenia wyłącznie wg danej kategorii; 'pusta' kategoria oznacza wszystkie urządzenia. Poniżej znajduje się lista wszystkich urządzeń zgodnie z ustawionym filtrem kategorii. Dla każdego urządzenia tabela zawiera informację o nazwie, czasie ostatniego połączenia serwera z nim i adresie IP. Po lewej stronie każdego urządzenia w liście jest widoczny checkbox, który jest wykorzystywany przez formularz w sekcji *Enqueue jobs* służący do zlecania aktualizacji oprogramowania. Przycisk *View* po prawej stronie przenosi do podstrony ze szczegółowymi informacjami o danym urządzeniu.

W szczegółowym widoku o urządzeniu można:

- sprawdzić informację o wersji *release'u*
- usunąć je z platformy,
- przypisać do wybranych kategorii,
- dowiedzieć się o zainstalowanych aplikacjach w release,
- zlecić zadania do wykonania,
- obejrzeć historię zakończonych zadań.

Devices | Categories | Repository

Device info

Identity	00:22:15:70:84:B6-beagle
Last contact	30-12-2011 21:07:54
IP	127.0.0.1
Releases	beagle, 2

Delete

Categories

☑ My Devices

Update

Applications

Name	Description	Version
sup_beagle	Software Update Platform client	1.0
sasl	SASL CXC 138 11	2.1.9.4
inets	INETS CXC 138 49	5.6
sampleapp	Sample application for Software Update Platform	2.0
stdlib	ERTS CXC 138 10	1.17.4
kernel	ERTS CXC 138 10	2.14.4

Pending jobs

Message	Module	Function	Extra	Status
				Submit

Upgrade to release

Release Name	
	Submit

Finished jobs

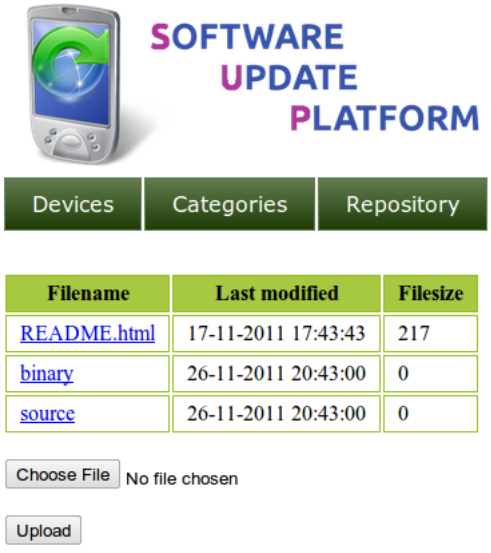
Message	Module	Function	Extra	Status

2.2.2. Kategorie

Podstrona *Categories* jest odpowiedzialna za dodawanie i usuwanie kategorii, do których można przypisać urządzenia. Dla każdej kategorii tabela zawiera informacje o nazwie i ilości urządzeń do niej przyporządkowanych.

2.2.3. Repozytorium

Repozytorium służy do przechowywania wcześniej przygotowanych paczek debianowych. Folder główny, *binary* oraz *source* są dostępnymi lokalizacjami, w których można umieszczać paczki.



3. Model rozwoju aplikacji zarządzanej przez platformę

Platforma do automatycznych aktualizacji zakłada określony wzorzec tworzenia oprogramowania zarządzanego przez platformę. Jest to konieczne ze względu na zależność mechanizmów używanych w systemie (np. paczkowania w pakiety *.deb*).

3.1. Podstawowe założenia

Podstawowymi założeniami i ograniczeniami przyjętymi dla modelu rozwoju aplikacji zarządzanej przez platformę są:

- W systemie używanym do tworzenia oprogramowania musi być obecny Erlang. Od jego wersji i architektury zależy wersja runtime'u wchodzącego w skład wynikowych paczek na urządzenia zdalne. Erlang jest również używany przez większość skryptów dostarczanych wraz z platformą. Dodatkowo, skrypty generujące paczki debiana korzystają z silnika szablonów erlydtl `https://github.com/evanmiller/erlydtl`, który musi być zainstalowany w systemie.

- W celu tworzenia pakietów .deb konieczna jest obecność w systemie menedżera pakietów dpkg.

- Fragmentem oprogramowania zarządzanym przez platformę jest erlangowy release `http://www.erlang.org/doc/design_principles/release_structure.html` tworzony zgodnie z wzorcami Erlang OTP `http://www.erlang.org/doc/design_principles/users_guide.html`. Zakłada się, że użytkownik platformy tworzący oprogramowanie przez nią zarządzane jest zaznajomiony z wzorcami tworzenia oprogramowania w Erlangu (OTP).

- Oprogramowanie wdrażane jest na urządzenia zdalne w postaci tzw. węzła wbudowanego (erlang embedded node). Oznacza to wykorzystanie konkretnych mechanizmów instalacji, aktualizacji i uruchamiania węzła. Przez większość czasu są one niewidoczne dla użytkownika, jednak należy o nich pamiętać podczas tworzenia np. paczek aktualizacyjnych.

- Oprogramowanie w Erlangu tworzone jest z wykorzystaniem narzędzia rebar `https://github.com/basho/rebar`. Zakłada się znajomość tego narzędzia przez twórcę oprogramowania zarządzanego przez platformę. Ze względu na niedociągnięcia w standardowej wersji, użyta została wersja z poprawkami `https://github.com/smarkets/rebar`.

Szczegóły nt. wyżej wymienionych założeń, ograniczeń i narzędzi opisane są w dalszych częściach dokumentacji.

3.2. Tworzenie środowiska rozwoju oprogramowania zarządzanego przez platformę

Platforma udostępnia jako plik archiwum przykładową strukturę katalogów wraz z wykorzystywanymi narzędziami, która może służyć do tworzenia release'u erlangowego zarządzanego przez platformę.

Zawartość środowiska startowego:

```
devenv/
    rebar
    rebar.config
    apps/
        sup_beagle/
            ...
    rel/
        genrelup
    debian/
        debian.config
        templates/
            ...
        makebasedeb
        makeappdeb
        makereldeb
```

3.2.1. rebar

Plik wykonywalny zawierający narzędzie rebar. W celu łatwiejszego jego użycia można umieścić katalog ze środowiskiem w zmiennej PATH lub stworzyć alias w shellu odwołujący się do tego skryptu.

3.2.2. rebar.config

Plik konfiguracyjny rebara. Początkowa zawartość tego pliku jest następująca:

```
{sub_dirs, [
            "rel",
            "apps/sup_beagle"
           ]
}.
```

Jak widać, konfiguracja rebara sprowadza się do wylistowania podkatalogów, które mają być przez niego widoczne. Plik ten wymaga edycji podczas dodawania lub usuwania aplikacji `http://www.erlang.org/doc/design_principles/applications.html` do release'u erlangowego przez programistę. Przykładowo, dodanie erlangowej aplikacji o nazwie `sampleapp` wymaga dodania do listy `sub_dirs` elementu `apps/sampleapp"`.

3.2.3. apps

Katalog ten zawiera źródła wszystkich aplikacji erlangowych tworzonych przez programistę, które mają znaleźć się w wynikowym release. Domyślnie katalog ten zawiera źródła aplikacji `sup_beagle` będącej klientem platformy automatycznych aktualizacji. Aplikacja ta musi znaleźć się w wynikowym release. Źródła wszystkich innych tworzonych aplikacji należy umieścić w katalogu `apps`, w podobny sposób jak aplikację `sup_beagle` (tj. źródła w podkatalogu src, pliki nagłówkowe w katalogu `include`, itp.). Dla każdej aplikacji należy również dodać wpis w pliku konfiguracyjnym rebara (patrz wyżej).

3.2.4. rel

Ten katalog służy do generacji wynikowych węzłów erlangowych za pomocą rebara (komenda generate - szczegóły w odpowiedniej sekcji dokumentacji). Wygenerowany przez rebara węzeł jest obrazem wynikowego środowiska docelowego zawierającego erlangowy runtime z zestawem skryptów i plików konfiguracyjnych oraz skompilowanym release'm. Środowisko takie stanowi podstawę do generacji paczek debiana, gotowych do instalacji na urządzeniu zdalnym.

Skrypt `genrelup` służy do tworzenia w węźle docelowym pliku relup `http://www.erlang.org/doc/man/relup.html`. Szczegóły jego użycia opisane są w sekcji dotyczącej generowania paczek debiana.

3.2.5. debian

Katalog ten zawiera zestaw skryptów oraz plik konfiguracyjny używane do generacji paczek *.deb*. Jest to również katalog, w którym umieszczane będą wygenerowane paczki. Szczegóły nt. użycia skryptów i konfiguracji opisane są w sekcji dotyczącej generacji paczek.

3.3. Inicjalizacja węzła erlangowego

Po stworzeniu bazowego środowiska należy zainicjalizować węzeł erlangowy za pomocą rebara. W tym celu należy wejść do katalogu `rel` i wykonać następującą komendę:

```
../rebar create-node nodeid=id_wezla
```

`id_węzła` jest dowolnym identyfikatorem docelowego węzła erlangowego. Służy on również m.in. jako nazwa release'u i (domyślnie) stanowi bazową nazwę dla paczek debiana.

Efektem wykonania komendy jest wygenerowanie przez rebara w katalogu `rel` plików `reltool.config`, `sys.config` oraz podkatalogu `files`.

3.3.1. reltool.config

Plik konfigurujący zachowanie komendy `rebar generate`. Ściślej: konfiguruje zachowanie funkcji z erlangowego pakietu `reltool` używanego wewnętrznie przez rebara do generacji docelowych release'ów i węzłów.

Wymagana jest edycja tego pliku i dodanie do niego następujących wpisów:

- wpisanie katalogu z źródłami aplikacji (`"../apps"`) do listy `lib_dirs`

- wylistowanie wszystkich aplikacji wchodzących w skład release'u o nazwie odpowiadającej id węzła (obowiązkowo należy wylistować aplikację `sup_beagle`)
- nadanie release'owi pożądanego numeru wersji

Uwaga: dla poprawnego działania skryptów zarządzających w paczkach *.deb* wymagane jest pozostawienie pustego release'u `start_clean` oraz opcji `{excl_archive_filters, [".*"]}`.

Gotowy plik `reltool.config` dla release'u o nazwie `beagle` w wersji 2 zawierającego aplikację `sampleapp` może wyglądać następująco:

```
{sys, [
       {lib_dirs, ["../apps"]},
       {rel, "beagle", "2",
        [
         kernel,
         stdlib,
         sup_beagle,
         sampleapp
        ]},
       {rel, "start_clean", "",
        [
         kernel,
         stdlib
        ]},
       {boot_rel, "beagle"},
       {profile, embedded},
       {incl_cond, exclude},
       {excl_sys_filters, ["^bin/.*",
                           "^erts.*/bin/(dialyzer|typer)"]},
       {excl_archive_filters, [".*"]},
       {app, sup_beagle, [{incl_cond, include}]},
       {app, sampleapp, [{incl_cond, include}]},
       {app, kernel, [{incl_cond, derived}]},
       {app, stdlib, [{incl_cond, derived}]},
       {app, sasl, [{incl_cond, derived}]},
       {app, inets, [{incl_cond, derived}]}
      ]}.

{target_dir, "beagle"}.

{overlay, [
           {mkdir, "log/sasl"},
           {copy, "files/erl", "\{\{erts_vsn\}\}/bin/erl"},
           {copy, "files/nodetool", "\{\{erts_vsn\}\}/bin/nodetool"},
```

```
 {copy, "files/beagle", "bin/beagle"},
 {copy, "files/beagle.cmd", "bin/beagle.cmd"},
 {copy, "files/start_erl.cmd", "bin/start_erl.cmd"},
 {copy, "files/app.config", "etc/app.config"},
 {copy, "files/vm.args", "etc/vm.args"}
]}.
```

3.3.2. sys.config

Plik zawierający domyślną konfigurację release'u, tj. zbiorczą konfigurację aplikacji erlangowych wchodzących w jego skład.

3.3.3. files

Katalog zawierający pliki używane przez rebara do generacji węzła wynikowego.

3.4. Kompilacja i generacja węzła docelowego

Aby skompilować aplikacje zawarte w docelowym release erlangowym, w głównym katalogu środowiska należy wywołać komendę texttt./rebar compile

Po kompilacji można wygenerować węzeł docelowy za pomocą komendy `./rebar generate` która w katalogu `rel` stworzy katalog o nazwie takiej jak id węzła zawierający docelowy węzeł erlangowy. Nadaje się on do testowego uruchamiania. Służy do tego skrypt o nazwie takiej, jak id węzła w podkatalogu katalogu `bin` w wygenerowanym węźle.

3.4.1. Zachowywanie starych wersji

W celu generacji plików appup i `relup` służących do wykonywania upgrade'ów ze starszych wersji release'u, konieczne jest zachowywanie starszych wersji wygenerowanych węzłów docelowych. W tym celu wystarczy zachować katalog z wygenerowanym węzłem w danej wersji pod inną nazwą.

3.5. Generacja paczek debiana

Do tworzenia pakietów .deb służy zestaw skryptów w katalogu debian. Przed ich użyciem należy je skonfigurować, używając do tego pliku `debian.config`. Zawiera on następujące opcje:

- `nodename` - nazwa węzła - musi być taka sama jak id węzła użyty podczas inicjalizacji węzła (komenda `rebar create-node`)
- `source_dir` - katalog z wygenerowanym węzłem erlangowym służący jako źródło plików umieszczanych w paczkach debiana.
- `target_root` - katalog systemu docelowego na urządzeniu zdalnym, w którym zainstalowany zostanie wygenerowany węzeł erlangowy

- `maintainer` - dane osoby zarządzającej paczkami *.deb* w postaci *Imię Nazwisko* <*email*>. Dane te zostaną umieszczone w plikach `control` pakietów debiana (opcja `Maintainer`).
- `description` - opis umieszczany w pliku `control` pakietu debiana generowanego przez skrypt `makereldeb` (opcja `Description`)

Przykładowa zawartość pliku debian.config dla węzła o nazwie beagle może wyglądać następująco:

```
{nodename, "beagle"}.
{source_dir, "../rel/beagle"}.
{target_root, "/opt/beagle"}.
{maintainer, "Jan Kowalski <jankowalski@gmail.com>"}.
{description, "Sample SUP-managed node for Beagleboard"}.
```

3.5.1. Zarys modelu generacji paczek debiana

Węzeł erlangowy wygenerowany za pomocą rebara dzielony jest na następujący zestaw pakietów debiana:

- pakiet bazowy zawierający erlangowy runtime oraz zestaw skryptów stworzonych przez rebara do uruchamiania węzła. Pakiet ten jest zależny od architektury, a jego wersja odpowiada wersji ERTS (Erlang Run-Time System) węzła. Nazwa pakietu bazowego to `nazwa_węzła-base`.
- osobny pakiet dla każdej aplikacji erlangowej wchodzącej w skład release'u. Pakiety te są niezależne od architektury (opcja `Architecture` w pliku `control` ma wartość /textttall). Wersja każdego pakietu odpowiada wersji aplikacji. Nazwa pakietu z aplikacją to `nazwa_węzła-nazwa_aplikacji`
- pakiet główny, zawierający pliki release'u samego w sobie. Pakiet ten jest zależny od pakietu bazowego i pakietów aplikacji. Zależność jest ścisła, tj. wyspecyfikowane są konkretne wersje pakietów zależnych i tylko te mogą być akceptowane podczas instalacji. Nazwa pakietu głównego jest taka sama jak nazwa węzła.

Podział na wiele pakietów służy zmniejszeniu ilości danych pobieranych podczas aktualizacji oprogramowania. Nie należy wykonywać żadnych operacji bezpośrednio na pakietach innych niż pakiet główny.

3.5.2. Obsługa aktualizacji

Budując pakiety dla kolejnych release'ów należy pamiętać o kilku czynnościach, które należy wykonać przed wygenerowaniem węzła z nową wersją release'u bez których automatyczna aktualizacja może nie działać:

- należy zawsze zmienić wersję release'u na wyższą w pliku `reltool.config`. Oczywiście należy również pamiętać o zmienianiu wersji samych aplikacji, które się zmieniły, w ich plikach *.app*.

- konieczne jest zachowywanie wygenerowanych komendą `rebar generate` węzłów dla starych wersji release'u (tj. dla tych wersji, z których może być wykonywana aktualizacja).

- w katalogach źródłowych aplikacji, których wersja się zmieniła należy stworzyć pliki appup `http://www.erlang.org/doc/man/appup.html` definiujące sposób aktualizacji pojedynczej aplikacji. W tym celu można posłużyć się komendą `rebar generate-appupps`, jednak należy zważyć na jej ograniczoną funkcjonalność (możliwe jest wyspecyfikowanie tylko jednej starej wersji release'u). Pliki `appup` są konieczne do wygenerowania pliku `relup` za pomocą skryptu `genrelup`

3.5.3. genrelup

Generując zestaw pakietów dla nowej wersji release'u, w celu obsługi aktualizacji należy wygenerować plik `relup` definiujący sposób aktualizacji release'u na urządzeniu zdalnym, ńa żywo"(hot upgrade). Służy do tego skrypt `genrelup` o następującej składni:

```
genrelup release target_rel_dir:vsn upfrom_rel_dir:vsn,... \
downto_rel_dir:vsn,...
```

- `release` jest nazwą release'u (identyczna z id węzła)

- `target_rel_dir:vsn` definiuje katalog z docelowym węzłem wraz z wersją release'u

- `upfrom_rel_dir:vsn,...` to oddzielana przecinkami lista katalogów z wygenerowanym węzłem oraz wersją releasu, z których może zostać wykonana aktualizacja

- `downto_rel_dir:vsn,...` to oddzielana przecinkami lista katalogów z wygenerowanym węzłem oraz wersją releasu, do których może zostać wykonany downgrade (downgrade nie jest obsługiwany przez platformę, lecz potencjalnie może być wykonywany ręcznie)

  Przykładowe wywołanie skryptu generujące plik relup dla releasu o nazwie `beagle` w wersji 3 w węźle w katalogu `beagle-new` dla starych wersji release'u 1 i 2 zawartych w węźle wygenerowanym w katalogu `beagle-old`:

  ```
  genrelup beagle beagle-new:3 beagle-old:1,beagle-old:2
  ```

3.5.4. Tworzenie plików *.deb*

Do generacji finalnych paczek służą skrypty `makebasedeb`,`makeappdeb` oraz `makereldeb`. Skrypty te zapisują wygenerowane pakiety w katalogu `debian`.

3.5.5. makebasedeb

Skrypt tworzący pakiet bazowy. Nie wymaga żadnych parametrów.

3.5.6. makeappdeb

Skrypt tworzący pakiet dla danej aplikacji erlangowej. Składnia:

```
makeappdeb nazwa_aplikacji [wersja_aplikacji]
```

Jeśli węzeł zawiera tylko jedną wersję danej aplikacji, specyfikowanie wersji nie jest konieczne.

3.5.7. makereldeb

Skrypt służący do generacji pakietu głównego dla release'u. Składnia:

```
makereldeb wersja_releasu
```