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 - PRZEWODNIK

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. Cel prac i wizja produktu

1.1. Opis problemu

Problem: Duża ilość zdalnych urządzeń rozmieszczonych w terenie. Urządzenia komunikują się przez zawodną sieć. Na różnych urządzeniach działają różne wersje aplikacji. Zachodzi potrzeba aktualizacji oprogramowania na grupie urządzeń. Rozwiązanie: Platforma umożliwi monitorowanie oraz aktualizację wersji aplikacji na grupach urządzeń.

1.2. Opis użytkownika i zewnętrznych podsystemów

Użytkownikiem systemu jest osoba odpowiedzialna za oprogramowanie w systemie składającym się z urządzeń zdalnych. System składa się z dużej liczby tych urządzeń, które mogą mieć mocno ograniczone zasoby sprzętowe (np. *Beagleboard*). Urządzenia mogą mieć różne architektury. Urządzenia będą łączyć się z serwerem głównym, do którego przesyłają zebrane dane. Komunikacja odbywa się za pomocą zawodnego połączenia (np. *GSM*).

1.3. Opis produktu

Produkt składa się z serwera zawierającego repozytorium oprogramowania. Serwer ma za zadanie monitorować jakie wersje aplikacji znajdują się na konkretnych urządzeniach i grupach urządzeń. Udostępniono interfejs webowy, dzięki któremu można wybrać urządzenia (lub ich grupy), na których chcemy przeprowadzić aktualizację i sprawdzić jakie aplikacje są zainstalowane. Ponieważ nie zawsze możliwe jest połączenie z wybranym urządzeniem, serwer przechowuje jego stan. W momencie uzyskania połączenia z urządzeniem serwer wykonuje zaplanowane aktualizacje.

2. Zakres funkcfonalności

2.1. Wymagania funkcjonalne

Platforma:

- monitoruje, czy dane urządzenie jest dostępne
- monitoruje zainstalowane aplikacje oraz ich wersje na urządzeniach
- rejestruje urządzenia
- pozwala na definiowanie grup urządzeń
- umożliwia aktualizację i instalację aplikacji na pojedynczych urządzeniach
- umożliwia aktualizację i instalację aplikacji na grupach urządzeń
- umożliwia aktualizację i instalację za pomocą systemowych narzędzi takich jak apt

- webowy interfejs użytkownika
- umożliwia tworzenie paczek aplikacji dedykowanych dla platformy wraz ze skryptami instalacyjnymi

2.2. Wymagania niefunkcjonalne

- Obsługa między 1000 - 10000 urządzeń
- Obsługa różnych architektur i systemów operacyjnych
- Prawidłowe działanie w przypadku zrywających się połączeń
- Wymagania stawiane dokumentacji:
  - Podręcznik użytkownika
  - Podręcznik instalacji i konfiguracji.
  - Dokumentacja techniczna – dokumentacja kodu, opis testów.

3. Wybrane aspekty realizacji

Właściwie wszystkie podsystemy zostały wykonane w Erlangu, aby zachować homogeniczne środowisko:

- baza danych - *mnesia*
- serwer http - *mochiweb*
- backend
- aplikacja kliencka

Jedynym elementem, który używa innych technologii (*ErlyDTL*, *JavaScript*, *jQuery*, *CSS*) jest webowy interfejs użytkownika.

Komunikacja między urządzeniem klienckim i serwerem odbywa się przez własny protokół nad TCP. Jest on w łatwy sposób rozszerzalny, gdyby trzeba było dodać nowe funkcjonalności oraz zapewnia możliwość komunikacji z urządzeniami znajdującymi się za NATem.

4. Organizacja pracy

W trakcie prac podzieliliśmy się na dwa zespoły:

- zespół zajmujący się backendem w składzie Roman Janusz oraz Tomasz Kowal
- zespół zajmujący się frontendem oraz bazą danych w składzie Przemysław Dąbek oraz Małgorzata Wielgus

Prace zostały podzielone na dwa etapy. W pierwszym dokonaliśmy przeglądu potrzebnych technologii oraz zbudowaliśmy prototypy. W drugim zaimplementowaliśmy właściwą funkcjonalność. Pierwszy etap zakończył się w czerwcu 2011 roku. Pracę nad drugim zaczęliśmy w trakcie wakacji 2011 roku.

Prace wykonane przez poszczególne osoby

- Roman Janusz
  - Dokładne zapoznanie się z wzorcami OTP tworzenia oprogramowania w Erlangu - w szczególności *application* i *release*
  - Zaprojektowanie ogólnej struktury oprogramowania przeznaczonego na urządzenia zdalne. Opracowanie modelu rozwoju oprogramowania z użyciem narzęcia *rebar* i dodatkowych skryptów.
  - Dokładne zapoznanie się ze strukturą pakietów *deb*, działaniem menedżera pakietów *dpkg*/*apt* oraz metodami tworzenia pakietów i instalacji repozytorium. Rozpoznanie możliwości paczkowania oprogramowania tworzonego w erlangu z zachowaniem możliwości wykonywania hot-upgrade podczas aktualizacji pakietu.
  - Zaprojektowanie sposobu dekompozycji erlangowego release'u w zestaw pakietów *deb*. Stworzenie ogólnych skryptów (debian maintainer scripts) używanych podczas instalacji, aktualizacji i usuwania pakietów. Obsługa mechanizmu hot-upgrade podczas aktualizacji.
  - Implementacja skryptów użytkownika do tworzenia pakietów *deb* na podstawie erlangowego release'u wraz z ich konfiguracją.
  - Integracja mechanizmu menedżera pakietów z platformą do automatycznych aktualizacji.
  - Dokumentacja użytkownika i techniczna dotycząca sposobu wykorzystania menedżera pakietów *deb* w projekcie.
- Tomasz Kowal
  - Testowanie możliwości serwera Mochiweb (ostatecznie użytego jako serwer http).
  - Zbadanie różnych możliwości komunikacji klienta i backendu (*gen_tcp*, *gen_rcp*, użycie *JSON*).
  - Implementacja serwera przyjmującego zgłaszające się urządzenia klienckie.
  - Szkielet niskopoziomowej komunikacji.
- Przemysław Dąbek
  - Przygotowanie interfejsu do operacji na bazie danych (mnesia).
  - Zaprojektowanie webowego interfejsu użytkownika oraz logo.
  - Implementacja logiki odpowiedzialnej za przetwarzanie żądań HTTP.
  - Dodanie mechanizmu logowania (lager).
- Małgorzata Wielgus
  - Stworzenie prototypu interfejsu graficznego przy użyciu serwera *Yaws*.
  - Integracja backendu i frontendu serwera zarządzającego.
  - Rozszerzenie interfejsu graficznego o funkcjonalność zlecania zadań urządzeniom.
  - Testowanie różnych scenariuszy zakończenia *joba*.

5. Wyniki projektu

Wynikiem projektu jest działające oprogramowanie wraz z dokumentacją. Oprogramowanie składa się z dwóch podstawowych części: serwera zarządzającego oraz aplikacji klienckiej.

Serwer zarządzający po zainstalowaniu i uruchomieniu udostępnia repozytorium pakietów i obsługuje zgłaszające się urządzenia. Dzięki użyciu lekkich wątków erlangowych jest w stanie obsłużyć wiele urządzeń jednocześnie. Serwer udostępnia również webowy interfejs użytkownika dla administratora systemu umożliwiający operacje niezbędne z punktu widzenia zarządzania oprogramowaniem na dużej liczbie urządzeń. Pozwala on na przeglądanie stanów urządzeń i zainstalowanych na nich wersji aplikacji, grupowanie urządzeń oraz zlecanie aktualizacji urządzeniom.

Aplikację kliencką można zawrzeć w dowolnym *releasie* erlangowym, co zapewnia możliwość prostego wykorzystania systemu do różnych zastosowań, w tym na urządzeniach mobilnych. System jest w pełni zintegrowany z menedżerem pakietów *apt*. Można tworzyć paczki z programami erlangowymi w taki sposób, że poszczególne aplikacje znajdują się w osobnych pakietach, co jest dużą zaletą w stosunku do ręcznej aktualizacji, gdyż pobierane są jedynie paczki zawierające nowe wersje aplikacji.

Platforma została zaprojektowana z myślą o łatwości instalacji i użytkowania, a także rozwoju. Serwer zarządzający może zostać zainstalowany w dowolnym systemie operacyjnym z *Erlangiem*. Instalacja odbywa się w jednym kroku, automatycznie zostają pobrane wszystkie zależności. Webowy interfejs użytkownika z natury może być obsługiwany z dowolnego urządzenia z przeglądarką internetową. Aktualnie system docelowy, na którym działa aplikacja kliencka musi być wyposażony w menedżer pakietów *apt* oraz naturalnie emphErlanga. Aplikację w systemie klienta można również w prosty sposób zainstalować w postaci paczki. System został przetestowany na urządzeniu *beagleboard* z systemem operacyjnym *debian*.

Możliwości dalszego rozwoju

W dalszej kolejności można popracować nad integracją z pozostałymi menedżerami pakietów, takimi jak *yum*, *port*, *opkg*. Protokół komunikacyjny jest na tyle elastyczny, że użytkownik w prosty sposób może definiować własne zadania, które z pomocą systemu zostaną przydzielone do wykonania na wielu urządzeniach. Kolejnym krokiem jest poprawa bezpieczeństwa przez zaimplementowanie autoryzacji zgłaszających się urządzeń oraz serwera, a także szyfrowania przesyłanych wiadomości. Aby umożliwić podgląd stanu systemu w dowolnej chwili warto dopisać aplikację na *Androida* i *iOSa*, które udostępniałyby te same funkcjonalności, co interfejs webowy. Dodatkowo wzbogaciłoby również dodanie dodatkowych informacji na temat urządzeń - na przykład położenia geograficznego.