Dokumentacja techniczna programu: Mobilny asystent administratora

Darek Kucharzyk

Jacek Andrzejewski

Jakub Kośmider

Krystian Dużyński

Patryk Kiepas

Radosław Okomski

SPIS TREŚCI

1. OPIS PROGRAMU

1.1 Wstęp

Mobilny asystent administratora to narzędzia pozwalające w czasie rzeczywistym sprawdzać stan zasobów systemowych. System składa się z trzech części: serwera, agentów oraz mobilnego klienta. Cała architektura systemu jest typu klient- serwer.

1.2 Zadania poszczególnych modułów oraz opis działania

Aplikacja składa się z: serwera, klienta oraz agentów- linuksowego i windowsowego.

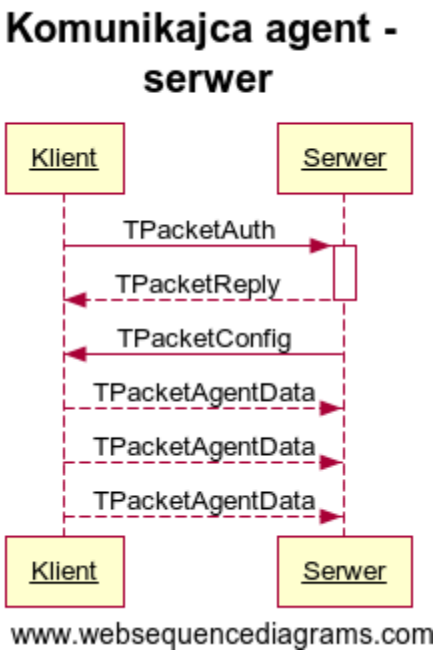
Podstawowym zadaniem serwera jest zbieranie danych z agentów, oraz przesyłanie ich do klientów. Serwer umożliwia ponadto przesyłanie danych konfiguracyjnych między urządzeniami. Kolejnym z zadań serwera jest przechowywanie danych z czujników w bazie danych, w celu późniejszego przeglądania w postaci wykresów użycia.

Zadaniem klienta jest wizualizacja danych zbieranych przez agentów. Klient umożliwia również zdalną, graficzną konfigurację agentów.

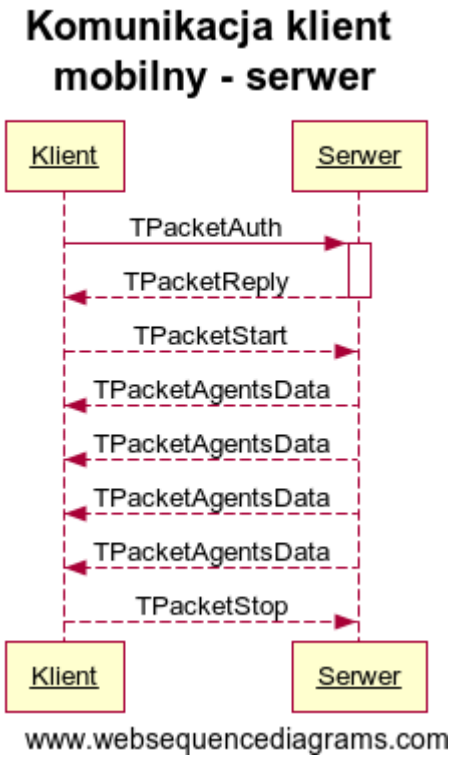
Zadaniem agentów jest zbieranie danych z czujników i przesyłanie ich do serwera. W przypadku braku połączenia, agenci przechowują dane do jednego dnia; po wznowieniu połączenia dane te są wysyłane na serwer.

1.2.1 Serwer

- Napisany w C++.
- Korzysta z bazy SQLite wraz z modułem MD5 do hashowania kluczy.
- Komunikacja z agentami oraz klientami.
- Identyfikuje agentów i klientów ich za pomocą kluczy autoryzacyjnych.
- Generuje nowe klucze autoryzacyjne.
- Czyści baze ze starych danych.
- Przechowuje konfiguracje agentów.
- Rozsyła nowe pakiety konfiguracyjne.
- Serwer pasywnie oczekuje na dane od agentów.



- serwer aktywnie wysyła dane po nawiązaniu połączenia i żądania danych. Po odebraniu pakietu ‘start’,



serwer zaczyna wysyłać ciągłe pakiety z interwałem, który wyśle klient.

- Baza danych wkompilowana statycznie

Opcje uruchomienia:

-c FILE, gdzie FILE to plik konfiguracyjny. w przypadku nie podania tego parametru, zostanie podjęta próba skorzystania z pliku config.cfg

-k generuje klucz i kończy działanie programu.

1.2.2 Agenci

Konfiguracja agenta odbywa się za pomocą pliku konfiguracyjnego (przy uruchomieniu) oraz (w dalszej części) przez sieć, za pomocą danych z serwera.

Autoryzacja klienta odbywa się za pomocą szesnastoznakowego klucza. Klucz ten jest generowany przez serwer i wyświetlany jednorazowo na kliencie androidowym.

Agent wysyła dane do serwera w sposób aktywny w odstępach czasu skonfigurowanych przez administratora.

Plik konfiguracyjny zawiera port, adres serwera oraz klucz uwierzytelaniający (generowany przez serwer), częstotliwość wysyłania danych, porty i protokoły z nazwami usług do monitorowania.

Agent nie posiada GUI.

Agent pobiera dane o systemie zgodnie ze specyfiakcją plaformy, która jest umieszczona w ‘sensors.cpp’.

W przypadku braku połączenia agent przechowuje historię do 1 dnia.

Agent pobiera i wysyła dane dotyczące: temperatury procesora (bez podziału na rdzenie), użycia RAMu, użycia procesora (bez podziału na rdzenie), zajętości dysków twardych (z podziałem na partycje), czas uruchomienia systemu oraz uruchomione usługi.

Linux:

Agent został napisany całkowicie w języku C++ z wykorzystaniem systemowych funkcji Linuxa.

Agent linuksowy zawiera w pliku konfiguracyjnym ścieżkę dostępu do pliku z sensorem temperatury oraz dzielnik wartości dla tego sensora.

Dane na linuksie są gromadzone w następujący sposób:

usługi/serwisy: próba połączenia z podanym portem (TCP), sprawdzanie pliku /proc/net/udp (UDP)

temperatura: poszukiwanie odpowiedniego czujnika w /sys/class/hwmon/ lub wpisanie ręczne

dyski: sprawdza, które są podmontowane w /proc/mounts a następnie pobiera informacje korzystając ze struktury statvfs

RAM: struktura sysinfo

uptime: struktura sysinfo

Opcje uruchomienia -c FILE, gdzie FILE to plik konfiguracyjny. w przypadku nie podania tego parametru, zostanie podjęta próba skorzystania z pliku config.cfg

Windows:

Agent napisany w języku C++ z wykorzystaniem usługi WMI i biblioteki Sigar.

Akwizycja poszczególnych danych przebiega w następujący sposób:

temperatura: przy pomocy Windowsowej usługi WMI (w zależności od dostępności: temperatura CPU bądź płyty głównej)

pamięć RAM: systemowe funkcje GlobalMemoryStatusEx() oraz GlobalMemoryStatus()

użycie procesora: funkcja sigar_cpu_perc_calculate() biblioteki Sigar

dyski: funkcja sigar_file_system_usage_get() biblioteki Sigar

uptime: funkcja GetTickCount() - odpowiednio wydzielone wartości, aby uzyskać czas

serwisy: funkcja próbuje nawiązać komunikację TCP, w przypadku UDP sprawdza, czy coś nasłuchuje na danym pocie

Agent Windowsowy wyświetla w konsoli informacje o systemie - te, które zostają potem wysłane. Cała komunikacja agenta oparta jest o WinSock2. Agent pobiera z serwera plik konfiguracyjny i nadpisuje nim stary plik. Domyślnie korzysta z pliku config.cfg.

Jeżeli server nie odpowiada klient lokalnie zapisuje zebrane dane w celu wysłania ich w możliwym momencie. W przypadku wyłączenia klienta dane, które nie zostały wysłane zostanę zapisane lokalnie.

Jeżeli w momencie uruchomienia agenta były jakieś dane, które nie zostały wysłane, a połączenie z serwerem jest ustalone, to zaległe dane zostaną fragmentami przesłane do serwera.

1.2.3 Klient Android

Klient został napisany w języku Java zgodnie z Android SDK w wersji 13, na której bazuje Android 3.2. Aplikacja jest kompatybilna również z późniejszymi wersjami.

Aplikacja składa się z serii 3 Activity. Pierwsze, widoczne od razu po uruchomieniu aplikacji służy do logowania się do serwera, kolejne odpowiada za wyświetlanie statystyk w czasie rzeczywistym, kolejne to widok wykresów.

Przy uruchomieniu aplikacji, bindowany (uruchamiany i przypisywany do danego okna) jest Serwis, działający tylko w ramach uruchomionego Activity. Służy on do komunikacji z wątkiem sieciowym. Serwis, po utworzeniu, uruchamia nowy wątek, w którym następuje komunikacja sieciowa z serwerem. Wątek ten zwraca otrzymane dane bezpośrednio do Activity, już z pominięciem serwisu. Po wyłączeniu aplikacji Serwis jest zatrzymywany; przy jej zminimalizowaniu, jego działanie nie jest gwarantowane, jednak obsługiwane.

Główna część komunikacji z serwerem odbywa się w sposób pasywny, co znaczy, że klient wysyła żądanie rozpoczęcia przesyłu danych, a następnie oczekuje od serwera kolejnych pakietów. Mimo to, by połączenie nie zostało zerwane, do serwera wysyłany jest ping.

Wysyłanie i odbiór pakietów do/z serwera składa się z dwóch części: wysłania/odebrania nagłówka z informacją o typie przesyłanego pakietu, oraz z informacją o jego rozmiarze. dopiero po obsłudze tego pakietu, dokonywane jest wysłanie/odbiór faktycznych danych.

Klient może być połączony tylko z jednym serwerem jednocześnie.

Klient umożliwia ustawienie za pomocą interfejsu graficznego danych połączenia z serwerem: adresu IP i portu serwera, klucza uwierzytelniającego oraz częstości otrzymywania danych od serwera.

Dla każdego z agentów możemy zdefiniować: nazwę, częstotliwość wysyłania danych do serwera, ścieżkę do sensora temperatury oraz dzielnik dla tego sensora. Ponadto możemy definiować serwisy. W skład każdego wchodzi nazwa, port oraz typ (TCP/UDP) usługi.

Klient umożliwia generowanie kluczy dla agentów.

Klient automatycznie zapamiętuje dane służące do połączenia z serwerem. Wykorzystywany jest do tego obiekt SharedPreferences.

Wykresy rysowane są przy użyciu obiektu SurfaceView, który został przystosowany do potrzeb aplikacji.

Lista agentów jest utworzona za pomocą ListView, a każdy obiekt w tej liście jest zdefiniowany w klasie ArrayAdapter dostosowanej do potrzeb aplikacji.

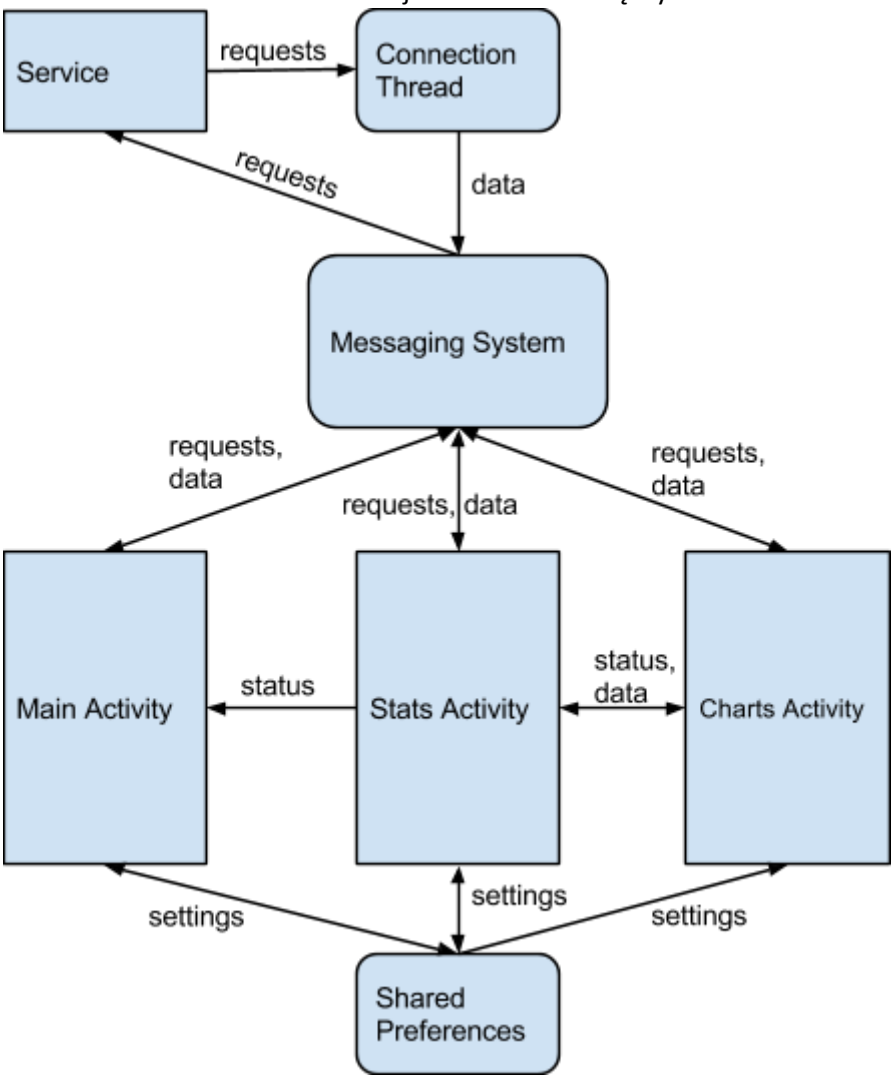
Na liście wyświetlane są: opcjonalna nazwa wraz z adresem IP, temperatura, zużycie procesora, zużycie ramu, czas działania maszyny, lista serwisów oraz użycie dysków.

Zaimplementowano również system alertów. Obsługiwana jest zbyt wysoka temperatura oraz zbyt duże zapełnienie dysków twardych.

W widoku wykresów zaimplementowano użycie CPU, pamięci RAM, zapełnienie dysków oraz temperatury z wybranego okresu. Czas próbkowania uzależniony jest od rozmiarów ekranu. punkt na osi czasu jest średnią z danego okresu należącego do danego piksela na ekranie.

Do rysowania wykresów zastosowano interpolację Hermite’a.

Schemat komunikacji między elementami aplikacji:



2. INSTRUKCJA KOMPILACJI

2.1 Serwer

Wymagany unixowy build system (np. w debianie pakiet build-essentials). Kompilację wykonujemy przez wywołanie polecenia ‘make’ w katalogu z plikami źródłowymi. W efekcie tworzy się katalog build/ w którym znajduje się plik wykonywalny.

2.2 Agent linuxowy

Dokładnie te same wymagania co przy serwerze:

Wymagany unixowy build system (np. w debianie pakiet build-essentials). Kompilację wykonujemy przez wywołanie polecenia ‘make’ w katalogu z plikami źródłowymi. W efekcie tworzy się katalog build/ w którym znajduje się plik wykonywalny.

2.3 Agent windowsowy

Wymagany kompilator MS Compiler w wersji 16.0 bądź późniejszy dostarczony wraz ze środowiskiem Visual Studio 2010 (bądź późniejszy). Po stworzeniu nowego projektu i dodaniu kodów źródłowych z folderów ‘agent_win’ oraz ‘common’ należy podlinkować następujące biblioteki: ‘sigar-x86-winnt.lib;Wbemuuid.lib’. Teraz można już spokojnie przystąpić do kompilacji. Podczas uruchamiania agent będzie wymagał biblioteki dynamicznej ‘sigar-x86-winnt.dll’, która znajduje się w folderze ‘agent_win/bin’.

3. DOKUMENTACJA KODU.

Szczegółowa dokumentacja została umieszczona w folderze doc nazwa pliku: ZPI_AdminTools.pdf. Dokumentacje wygenerowana na podstawie doxygen 1.8.5

4. Licencja

Projekt zrealziowany na licencji zgdonej z BSD.