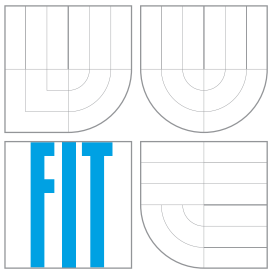
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO MONITORING SÍŤOVÝCH SERVERŮ A SLUŽEB

SYSTEM FOR MONITORING OF NETWORK SERVERS AND SERVICES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PETR HOLUBEC
AUTHOR

VEDOUCÍ PRÁCE — Ing. PAVEL OČENÁŠEK, Ph.D.
SUPERVISOR

BRNO 2013

Abstrakt

Tato bakalářská práce se zabývá tvorbou monitorovacího systému, který poskytuje aktivní periodické monitorování serverů a jejich konkrétních služeb a nejrůznější způsoby upozornění na překročení těchto monitorovaných hodnot. Navíc zobrazuje statistiky a grafy jednotlivých hlídaných hodnot a umožní sledovat i systémové prostředky monitorovaných serverů (za předpokladu spuštění monitorovacího agenta). Pro ovládání celého systému slouží klientská aplikace, která je dostupná pro standardní desktopové operační systémy a mobilní systém Android.

Abstract

This bachelor's thesis deals with creation of a monitoring system that provides active periodic monitoring of servers and their specific services and a variety of notification methods exceeded the monitored value. In addition, displays statistics and graphs of monitored values and provides monitoring of system resources (assuming you start a monitoring agent). To control the system serves a client application that is available for standard desktop operating systems and mobile system Android.

Klíčová slova

Monitoring, server, modularita, komunikace, statistiky, agent, Java, Android.

Keywords

Monitoring, server, modularity, communication, statistics, agent, Java, Android.

Citace

Petr Holubec: Systém pro monitoring síťových serverů a služeb, bakalářská práce, Brno, FIT VUT v Brně, 2013

Systém pro monitoring síťových serverů a služeb

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Pavla Očenáška, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Holubec
15. května 2013

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Pavlu Očenáškovi, Ph.D. za bezproblémovou spolupráci a užitečné rady, které vedly ke zlepšení mé práce.

Obsah

Kapitola 1

Úvod

Kdo v dnešní době nemá počítač? Snad každý, koho ve svém okolí znám, má anebo přinejmenším někde využívá klasický osobní počítač, notebook popřípadě mobilní telefon či tablet. Přesto, že jsem napsal velice obecné slovo „využívá“, měl jsem na mysli především prohlížení webu, psaní emailů, komunikaci s přáteli a všeobecně přístup ke službám na největší síti světa – internetu. Všechny zmíněné činnosti dozajista patří k tomu nejzákladnějšímu a pro většinu uživatelů k naprosto samozřejmému základu. Přesto není neustálý běh a bezchybná funkčnost těchto i jiných služeb takovou samozřejmostí, jakou si mnoho lidí myslí. Od softwaru obsluhujícího jednotlivé uživatele té které konkrétní služby, který nepochybně může obsahovat spoustu bugů[1] nebo kvůli nečekané události přestat fungovat, přes hardware, který jakožto elektrotechnické zařízení není věčný, až po propojení jednotlivých komunikujících stran. To vše může být a také je příčinou výpadků čili nedostupnosti nejrůznějších služeb. Za všechny uveďme jeden názorný příklad.

Představte si běžnou, nijak velkou ani přehnaně organizovanou, firmu. Jako každá jiná firma má i tato naše svoje webové stránky, které provozuje na svém vlastním serveru. Tento server není využit k žádným jiným účelům a není umístěn v žádné speciální místnosti (serverovně). Ovšem jak zákon schválnosti tomu chtěl, jednoho dne nepozorná uklízečka zakopla o síťový kabel, který se ze serveru vytrhl. Přesto si uklízečka ničeho nevšimla a pokračovala dál ve své práci. Protože na serveru byly pouze statické webové stránky, které se aktualizovaly jen několikrát do roka, nikdo si jejich výpadku nevšiml. Až po dlouhé době přijde majiteli firmy rozhněvaný email od nespokojeného potencionálního zákazníka, který si na jejich webových stránkách chtěl najít katalog a ceny služeb. Když ale ani po několika dnech stránky nefungovaly, rozhodl se využít konkurenční nabídky.

Majitel naší fiktivní firmy jistě nebude nadšen z toho, že přišel o zákazníka. Ještě více ho naštve, až si uvědomí, že dotyčný pisatel mohl být pouze špičkou ledovce a lidí, kteří se obrátili na konkurenci, mohlo být mnohem více. A to vše jen díky nepozorné uklízečce! Nebo že by nebyla na vině pouze uklízečka?

1.1 Monitoring

Zde se poprvé dostáváme k pojmu *monitoring*. Kdyby si majitel zajistil pro firemní server nějaký způsob monitorování, dozvěděl by se o výpadku webových stránek včas a mohl by rychle zajistit nápravu. Pokud se, jako v našem příkladu, dozví majitel o výpadku až od

[1] Bug je programátorská softwarová chyba, kterou udělal programátor při vytváření počítačového programu (definice převzata z wikipedia.org).

nespokojeného klienta, jedná se o nejhorší možnou variantu a indikuje to něco v nepořádku.

Monitoring serverů a síťových služeb je určen právě k tomu, aby průběžně testoval jejich dostupnost či funkčnost a v případě výpadku nebo nekorektního běhu služby o tom informoval patřičnou osobu. Samotné monitorování pak může být rozšířeno i o sledování systémových prostředků, které je kromě upozorňování na kritické hodnoty vhodné zejména pro odhalování slabých popřípadě silných míst hardwaru použitého v serveru. Dobrou vlastností kvalitního monitorovacího systému je široká škála možností upozornění na kritické události. Mezi ně můžeme zařadit například email, SMS nebo tzv. instant messaging.

Nástroje a systémy tohoto zaměření jsou nedocenitelným pomocníkem administrátorů všech větších sítí. Dokážete si představit denně kontrolovat funkčnost několika desítek serverů a všech jednotlivých služeb na nich běžících? Nicméně ani běžní provozovatelé jednoho či dvou serverů se nemusí spoléhat pouze na manuální kontrolu funkčnosti. Jednoduchý nástroj, který jednou za daný čas zkontroluje, že vše, co má běžet, skutečně běží, umožní majiteli serverů vytěsnit z hlavy další věc, na kterou nebude muset stále myslet. Pokud je systém opatřen srozumitelným a přehledným uživatelským rozhraním, není jeho využití zapovězeno naprosto nikomu.

1.2 Motivace vývoje aplikace pro systém Android

Mobilní systém Android[2] od společnosti Google je v posledních letech velice často skloňovaným systémem. S nástupem tzv. mobilní doby, kdy už nikdo nechce být vázán na svůj počítač, který mu leží doma na stole a chce být neustále ve spojení se světem, ať už jede tramvají nebo sedí v čekárně u doktora, se tento systém dostal do popředí zájmu velkého množství lidí. Mobilní telefony s tímto systémem si dnes kupují i lidé, kteří ani netuší, co to mobilní operační systém je a jejich cena se dostává na úroveň dvou tisíc korun. Další kategorií jsou dnes velmi diskutované tablety, které, s výjimkou produktů společnosti Apple, téměř výhradně disponují systémem Android. To vše nahrává rozšířenosti a také oblíbenosti tohoto mobilního systému, díky čemuž se tvorba těchto aplikací stává velice zajímavou.

Protože si je Google velice dobře vědom toho, že síla a úspěch jakéhokoliv systému (mobilního především) závisí na množství dostupných aplikací a uživatelské komunitě, dává vývojářům i dalším nadšencům mnoho prostředků a podpory pro tvorbu těchto aplikací. Mezi příjemné vlastnosti programování pro tento systém můžeme zahrnout například:

- Možnost psát programy v běžně používaném objektově orientovaném jazyce Java
- Používání některých běžných knihoven tohoto jazyka
- Podpora oblíbených nástrojů jako například vývojová prostředí Eclipse[3] a NetBeans[4]
- Rozsáhlá dokumentace a příklady[5]
- Velmi silná komunita aktivní na mnoha diskuzních fórech[6]

Z výše popsaného plyne, že tvorbou aplikací pro Android má určitě cenu se zabývat a do budoucna může být velice perspektivní prakticky si osvojit tyto schopnosti.[9]

[2] Více informací na http://www.android.com/.
[3] Více informací na http://www.eclipse.org/.
[4] Více informací na http://netbeans.org/.
[5] Dostupné na http://developer.android.com/index.html.
[6] Například http://androidforums.com/, http://forum.xda-developers.com/, http://androidcommunity.com/forums/ nebo české fórum http://androidforum.cz/.

Kapitola 2

Existující řešení

Na současném trhu je obrovský počet nejrůznějších nástrojů pro jednodušší nebo i komplexnější monitoring serverů a síťových služeb. Například v obchodu Google Play[1] najdeme nespočet aplikací, které se zaměřují na tuto problematiku. Tyto aplikace lze v zásadě rozdělit podle jejich složitosti a funkcí na striktně zjišťující dostupnost (buď jednorázové testování či pravidelné kontroly) a komplexnější nástroje poskytující další funkce jako sledování systémových prostředků či administraci vzdálených počítačů.

2.1 Pouze síťový monitoring

Tento typ aplikací je zpravidla tvořen jediným programem, a pokud se zde budeme bavit o převážně mobilních zařízeních, lze jich v patřičných obchodech najít nepřeberné množství. Po instalaci aplikace se zadají příslušné adresy sledovaných počítačů popřípadě nějaké doplňující informace a aplikace je připravena k použití. Zde můžeme rozdělit tento typ aplikací na jednorázové nástroje a kontinuální monitoring umožňující upozornění na změnu stavu.

2.1.1 Jednorázové zjištění dostupnosti

Poměrně jednoduché aplikace implementující testování pomocí ICMP echo zpráv, získávání hlavičky protokolem HTTP, traceroute a dalších. Slouží pro jednorázové otestování korektní funkčnosti či dostupnosti zadaných síťových prvků nebo služeb a využijí se zejména při zjišťování problémů na síti. Mezi reprezentanty lze zařadit Server Monitor[2] výrobce spc-m nebo Ping & DNS[3] výrobce Ulf Dittmer.

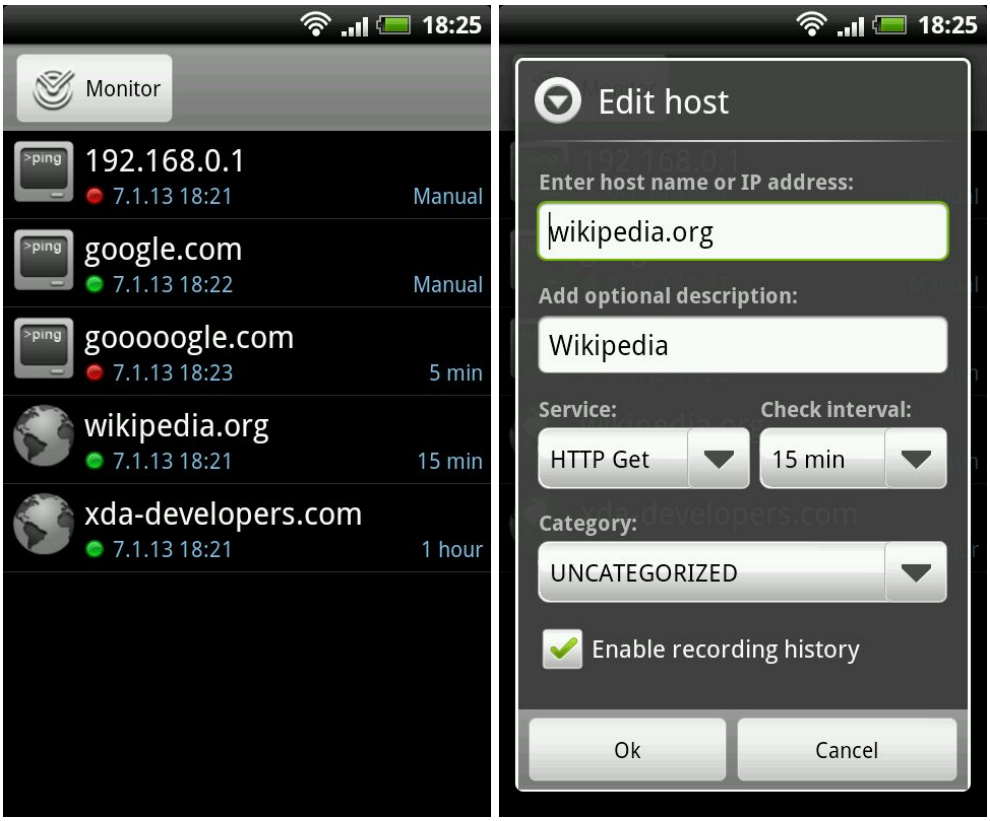
2.1.2 Periodické testování s notifikacemi

Aplikacím této kategorie již lze skutečně říkat monitoring. Po vložení jednotlivých sledovaných počítačů lze nastavit periodu testování a možnosti upozornění na vybrané stavy. Propracovanější aplikace nabízejí několik způsobů upozornění, například pomocí notifikace v mobilním zařízení, emailu, SMS nebo protokoly Jabber a podobnými. Aplikace typicky spustí službu běžící na pozadí, která v nastavených intervalech testuje zadané počítače a v případě výpadku nebo jiné sledované události na to patřičným způsobem upozorní. Mezi zajímavé a netradiční vlastnosti lze zařadit například možnost definování doby spánku

[1] Dříve nazývaný Android Market. Dostupný na https://play.google.com/store.
[2] Dostupný na https://play.google.com/store/apps/details?id=net.spc.app.svrmon.
[3] Dostupný na https://play.google.com/store/apps/details?id=com.ulfdittmer.android.ping.

a na to navázaná změna chování notifikací nebo simulace nastaveného upozornění. Mezi standardní vlastnosti by nepochybně mělo patřit generování statistik a tvorba přehledných grafů.

Jako zástupce této kategorie můžeme zmínit aplikace Network Tools[4] od Vladimira Kutse (viz obrázek 2.1), Server Monitor[5] společnosti SerafinSoft, Server Down Alarm[6] výrobce Juark.com dostupnou v trial verzi nebo Server Monitor Free[7] společnosti Programming Monkeys dostupnou zdarma (s reklamami).



Obrázek 2.1: Aplikace Network Tools pro monitoring serverů.

2.2 Komplexní systémový monitoring

Tyto systémy nejsou jen osamocenou mobilní či jinou aplikací k jednoúčelovému použití, ale nezřídka kdy obsahují více součástí (monitorovací klient a agent sbírající data) a zvládají velké množství nejrůznějších úkolů od prostého monitorování dostupnosti síťové služby až po komplexní sledování systémových prostředků dotyčného systému případně doplněné i o jeho administraci. Samozřejmostí jsou velice podrobné možnosti konfigurace notifikací, a to jak způsobu (upozornění v notifikační liště, email, SMS, instant messaging atd.), tak úrovně (varování, chyba, kritická chyba apod.).

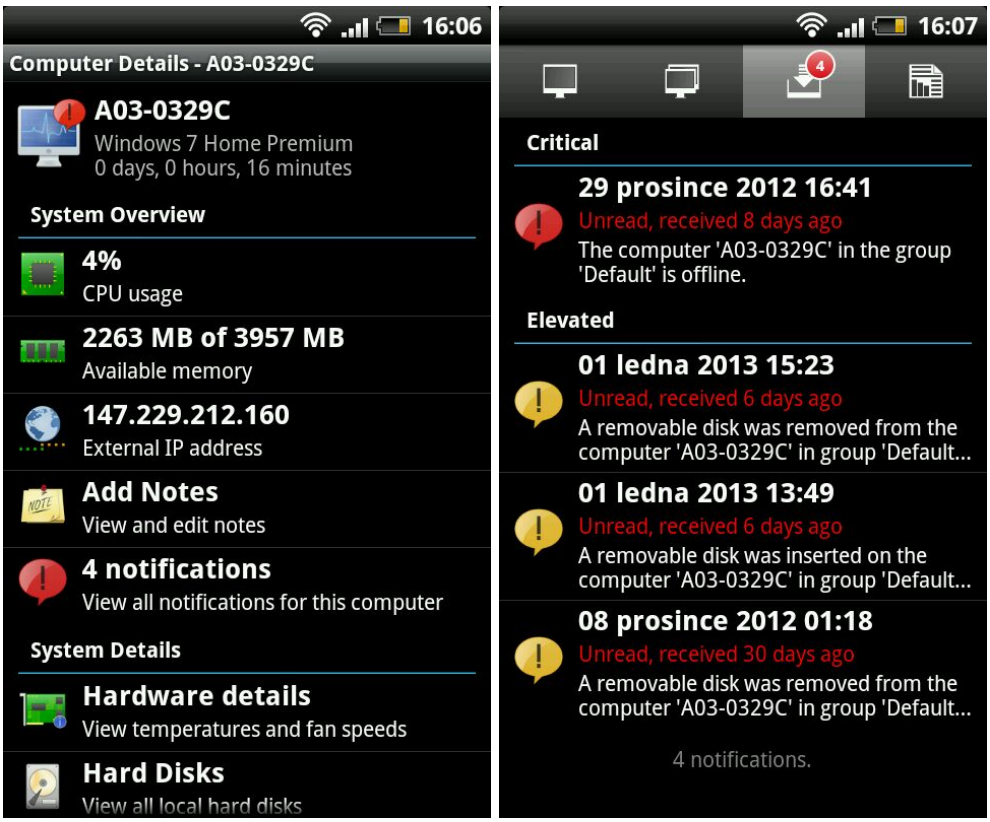
[4]Dostupná na https://play.google.com/store/apps/details?id=ua.cv.westward.networktools.
[5]Dostupná na https://play.google.com/store/apps/details?id=com.odinnet.servermonitor.
[6]Dostupná na https://play.google.com/store/apps/details?id=com.superofsuper.sda.
[7]Dostupná na https://play.google.com/store/apps/details?id=com.free.

2.2.1 PC Monitor

Systém PC Monitor[8] od společnosti MMSOFT Design slouží ke vzdálenému mobilnímu monitorování systémových prostředků a správě počítačů. Skládá ze dvou částí. První je agent běžící na pozadí monitorovaného systému, kde se zároveň v přehledném grafickém rozhraní provádí veškerá konfigurace. Tento agent je dostupný pro platformy Windows (32 i 64 bit), Linux a Mac. Druhou součástí tohoto systému je samotná mobilní aplikace pro sledování a ovládání monitorovaných počítačů, která je dostupná pro systémy Google Android, Apple iOS a Windows Phone. Navíc je dostupné webové rozhraní, aplikace pro Metro do Windows 8 a také je poskytováno PC Monitor API.

Celý systém nabízí obrovské množství nastavení od možnosti povolení či zakázání zobrazení jednotlivých parametrů a stavů monitorovaného systému, přes velice detailní konfiguraci upozornění, až po definici vlastních pravidel. Součástí je průzkumník souborů i možnost ovládat vzdálený počítač obsáhlou sadou příkazů. Komunikace se vzdáleným počítačem pomocí dialogů nebo jednoduše s využitím chatu se zde jeví jako samozřejmost.

Po prvním spuštění vyžaduje PC Monitor registraci uživatelského účtu, který je poté potřeba zadat v obou částech systému. Díky tomu se lze jednoduše dostat k vlastní konfiguraci monitorování odkudkoliv. Všechny komponenty jsou pěkně graficky zpracovány a vše působí velice přehledným a profesionálním dojmem (viz obrázek 2.2).



Obrázek 2.2: Aplikace PC Monitor pro monitoring a správu počítačů.

[8] Více informací na http://www.mobilepcmonitor.com/.

2.2.2 Nagios

Monitorovací systém Nagios[9] je reprezentantem open source systému pro PC s obrovskými možnostmi konfigurace a množstvím nejrůznějších typů monitorování. Je vyvíjen pod licencí GPL a primárně určen pro Linux. Umožňuje jak monitorování síťových služeb a zařízení, tak monitoring systémových prostředků sledovaných systémů. Celý koncept je založen na souboru modulů, které běží samostatně a předávají hlavnímu modulu Nagiosu dohodnutým způsobem data získaná testováním. Ten je pak dále zpracovává a interpretuje. Díky tomu je Nagios velice jednoduše rozšiřitelný o další funkčnost.

Jeho hlavními výhodami je například nastavení odlišných způsobů reakcí na různé typy výpadků nebo jiných monitorovaných hodnot. S velice rozsáhlou konfigurací pomůže systém šablon, které lze definovat podle konkrétního případu užití a s výhodou používat při konfiguraci chování a monitorování služeb. Dokáže zpracovat taktéž data pasivního testování jako například SNMP trapy. Monitorované systémy, ale i jednotlivé kontakty, lze pro snazší manipulaci sdružovat do obecných skupin. Mnoha nepříjemným chvílím taktéž předejde díky detekci flapování, což je dočasné kolísání mezi dvěma stavy (typicky dostupný/nedostupný). Systém počká, až se stav ustálí a teprve poté provádí patřičné akce.[6]

Jeho největší nevýhoda je paradoxně zároveň jeho výhoda a tou je jeho komplexnost a robustnost. Z toho plynou kromě kladných vlastností zmíněných výše také nedostatky, jimiž jsou velmi složitá a pouze manuální konfigurace, která zabere velké množství času. Tento systém je také poměrně náročný na server, na kterém běží, především na dostatek paměti.

První z nedostatků se snaží řešit nejrůznější grafické nadstavby, z nichž nejznámější a nejpovedenější se nazývá *Centreon*[10]. Tato poměrně rozsáhlá a propracovaná nadstavba zastřešuje téměř všechny možnosti konfigurace a výstupů systému Nagios a podává je administrátorovi v přehledném a graficky zdařilém kabátku. Samozřejmostí je grafická konfigurace téměř čehokoliv, přehledné statistiky, nastavení notifikací a spousta dalších přidaných uživatelsky cílených vlastností.

[9] Více informací na http://www.nagios.org/.
[10] Více informací na http://www.centreon.com/.

Kapitola 3

Analýza a specifikace požadavků

Cílem této bakalářské práce je vytvořit monitorovací systém, který bude poskytovat standardní funkce, jakými jsou aktivní periodické monitorování serverů a jejich konkrétních služeb a nejrůznější způsoby upozornění na překročení těchto monitorovaných hodnot. Navíc bude zobrazovat statistiky a grafy jednotlivých hlídaných hodnot a umožní sledovat i systémové prostředky monitorovaných serverů (za předpokladu spuštění monitorovacího agenta). Pro ovládání celého systému bude sloužit klientská aplikace, která bude dostupná pro standardní desktopové operační systémy a mobilní systém Android. Právě v mobilní verzi klientské aplikace je potenciál a příslib skutečně „nonstop“ dohledu nad svěřenými servery.

Způsobem organizace jednotlivých částí bude náš systém podobný systému PC Monitor zmíněnému v 2.2.1. Bude se skládat ze tří základních částí, klient, serverová část a agent, které jsou podrobněji popsány v kapitole 4. Jako úložiště dat bude sloužit relační databáze. Cílem je nemít žádná lokálně uložená data. To je jediný způsob, jak docílit toho, že se uživatel bude moci přihlásit odkudkoliv (přesněji z jakéhokoliv klienta) a vždy bude mít svoji konfiguraci. Na rozdíl od jiných podobných systému nebude disponovat webovým uživatelským rozhraním, ale jediným způsobem konfigurace a všeobecně uživatelské interakce budou klientské aplikace.

Systém musí být z uživatelského hlediska maximálně jednoduchý a přehledný. Zároveň je žádoucí maximální transparentnost existence serverové části. Po přihlášení do systému pomocí klientské aplikace by se vše mělo chovat, jako by se jednalo o jedinou aplikaci běžící na klientském počítači/mobilu. Dalším, z pohledu uživatele důležitým, detailem je, že upozornění o nedostupnosti sledovaného počítače se odešle jen, když tato událost skutečně nastane bez ohledu na to, zda je klientská aplikace připojena k internetu či nikoliv. Připojení jednotlivých klientských aplikací nemá žádný vliv na samotné testování, které probíhá zcela odděleně a nezávisle.

3.1 Aktivní a pasivní monitoring

Monitoring můžeme rozdělit podle množství režijních dat na lince na aktivní a pasivní, kde každý má samozřejmě své výhody i nevýhody.

V případě *pasivního* monitorování neodesíláme žádná další data navíc oproti běžnému provozu, a tudíž nezvyšujeme zahlcení linky. Díky tomu je testování přesnější než u aktivního monitorování. Pasivní monitorování je založeno na odchytu asynchronních událostí typu SNMP trap, upozornění od aplikací (například poštovní server) nebo dat NetFlow.

Dalším způsobem je pak sledování záznamových souborů standardu syslog. Nevýhodou zmíněných způsobů je, že jsme závislí na tom, zda nějaká taková data obdržíme. Z toho plyne, že takto nelze testovat neaktivní prvky.

U *aktivního* monitorování se předpokládá odeslání speciálního (testovacího) paketu, pomocí kterého zjistíme dostupnost a odezvu daného zařízení. Tento způsob může v případě nerozvážné konfigurace (především intervalů jednotlivých testování) zahlcovat linku a tím zároveň způsobovat nepřesnosti v měření. Využívá se přitom paketů ICMP, zpráv SNMP nebo prosté dotazování služnou Telnet na porty TCP. Tyto dotazy můžeme provést kdykoliv se nám zachce a tím pádem máme možnost testovat dostupnost pravidelně v daných intervalem. Z této nezávislosti na běžném provozu plyne, že můžeme testovat i neaktivní zařízení, na kterém aktuálně neprobíhá žádná komunikace.[7]

3.2 Fragmentace systému Android

S tím, jak se mobilní systém Android stále rozšiřuje, roste také jedna z jeho největších nevýhod, a tou je *fragmentace*. Velký počet současně používaných verzí systému doplňuje ještě nesrovnatelně větší množství zařízení nejrůznějších hardwarových konfigurací, na kterých Android běží. Pro zajištění ideálního zobrazení aplikace na různě velkých a orientovaných displejích nabízí Android propracovaný systém zdrojů (angl. resources). Dále je potřeba u každé aplikace specifikovat její cílovou a minimální verzi API[1]. To nám umožní definovat rámec verzí, pro které deklarujeme funkčnost naší aplikace. Tabulka 3.1 zobrazuje podíly jednotlivých verzí systému Android aktuální k datu 2. 4. 2013. Z té je patrné, že nejnižší verzí systému, pro kterou má význam aplikaci testovat je verze s označením Eclair (API 7). Proto by toto API mělo být definováno jako minimální nutné pro spuštění aplikace.

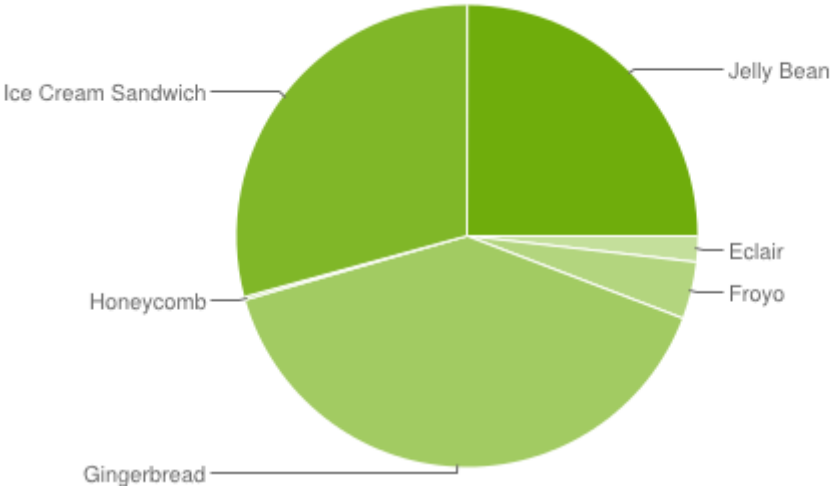
Verze	Označení	API	Podíl
1.6	Donut	4	0,1 %
2.1	Eclair	7	1,7 %
2.2	Froyo	8	4,0 %
2.3 - 2.3.2	Gingerbread	9	0,1 %
2.3.3 - 2.3.7		10	39,7 %
3.2	Honeycomb	13	0,2 %
4.0.3 - 4.0.4	Ice Cream Sandwich	15	29,3 %
4.1.x	Jelly Bean	16	23,0 %
4.2.x		17	2,0 %

Tabulka 3.1: Tabulka podílů jednotlivých verzí systému Android. Zdroj: [1].

Jak je vidět zároveň z grafu 3.1, největší podíl stále ještě zaujímá verze Gingerbread. Přesto její procento již klesá ve prospěch verze Ice Cream Sandwich a především nejnovější verze Jelly Bean. Tyto verze poprvé přišly se sjednocením systému pro mobilní telefony i tablety a v nejbližší době lze očekávat jejich velký rozmach a rozšíření i do levných chytrých telefonů.

Díky výše zmíněnému systému zdrojů je možné se efektivně vypořádat i s obrovským množstvím velikostí a rozlišení displejů. Systém Android vykresluje jednotlivé aktivity (ob-

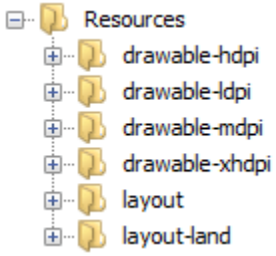
[1] Application Programming Interface – rozhraní pro programování aplikací.



Obrázek 3.1: Graf podílů jednotlivých verzí systému Android. Zdroj: [1].

razovky aplikace) na základě jim definovaných rozvržení (angl. layouts). Ty lze navrhnout v několika verzích, které se rozmístí do adresářů se specifickými názvy a systém pak vybere to nejvhodnější rozvržení pro konkrétní zařízení. Takto lze například efektivně definovat jedno rozložení komponent pro displej orientovaný na výšku a jiné pro režim na šířku.

Stejným způsobem je řešeno načítání obrázků a ikon. Protože obrázek v jediném rozlišení by mohl být na jednom displeji malý a na druhém zase příliš velký, lze do jednotlivých adresářů nahrát obrázky v různých rozlišeních a systém poté podle konkrétního DPI (Dots Per Inch – bodů na palec) vybere ten nejvhodnější. Ukázku takové adresářové struktury můžete vidět na obrázku 3.2.

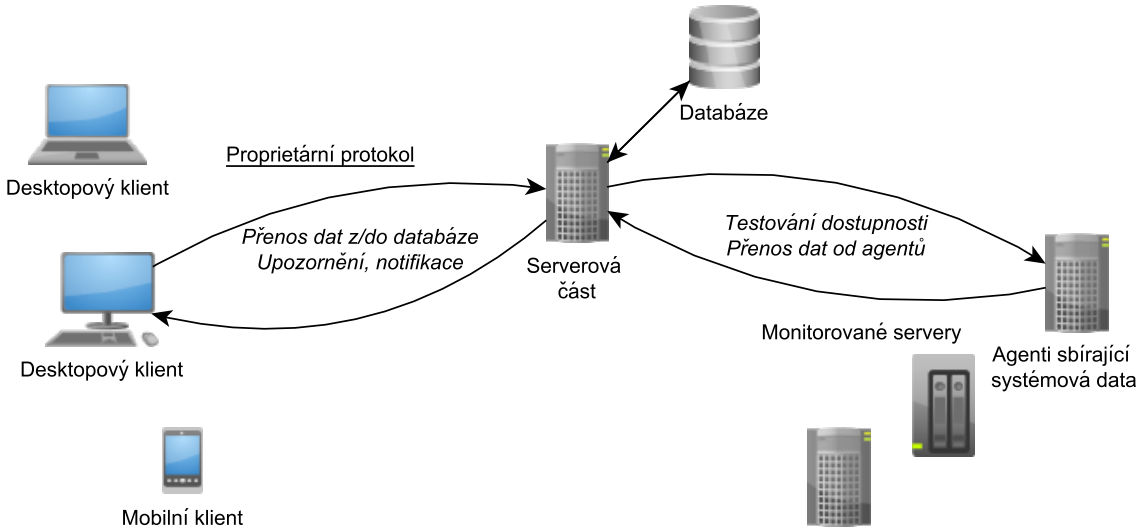


Obrázek 3.2: Ukázka adresářové struktury zdrojů systému Android.

Kapitola 4

Návrh systému

Celý systém by se měl skládat z několika částí. Těmi jsou klient sloužící jako uživatelské rozhraní celého systému, serverová část provádějící samotný monitoring, databáze, s níž bude pracovat serverová část systému a volitelně využitelný agent sbírající systémové údaje o monitorovaném počítači. Celé schéma je vidět na obrázku 4.1.



Obrázek 4.1: Schéma celého monitorovacího systému.

Protože by se mělo jednat o multiplatformní monitorovací systém, byla jako implementační jazyk zvolena *Java*. Jedná se o jednoduchý objektově orientovaný jazyk, který se řadí mezi nejpopulárnější programovací jazyky na světě. Z důvodů využití nejnovějších technik jsou aplikace vyvíjeny v JDK verze 7.

4.1 Klient

První částí je klient, který slouží jako jediné uživatelské rozhraní celého systému. Pomocí něho bude moci uživatel systém konfigurovat a zároveň jím bude upozorňován na jednotlivé problémy. Klient bude dostupný pro mobilní systém Android a desktopové systémy podporující běh aplikací v jazyce Java.

Uživatel se pomocí tohoto klienta přihlásí na svůj účet a bude moci konfigurovat jednotlivé monitorování, zobrazovat statistiky, grafy atd. Princip přihlašování ke svému účtu má obrovské výhody, pokud uživatel využívá více klientů na více zařízeních. Vždy bude mít po ruce svoji konfiguraci, a jakmile provede nějakou změnu, projeví se i na ostatních zařízeních.

Dokud klient poběží alespoň na pozadí, bude uživatel zároveň informován jistou formou notifikace o jednotlivých výpadcích nebo dalších překročeních sledovaných hodnot. Velká výhoda klienta bude právě v jeho mobilní verzi, kdy s minimálními datovými toky může být uživatel kdykoliv upozorněn na překročení hlídané hodnoty.

Tato část systému je navržena jako tzv. tenký klient (angl. thin client). To znamená, že ke svému fungování potřebuje další aplikace či systémy. Sám o sobě provádí minimum výpočetních úkonů a většinu složitějších operací deleguje na jinou část systému. Lokálně se neudržují ani žádná data. Přesto by měla být pro uživatele existence serverové části maximálně transparentní.

Implementace klienta bude v obou případech (mobilní i desktopová verze) založena na obsluze událostí. Programovací model systému Android k tomu vyloženě vyzývá a desktopové aplikace s grafickým uživatelským rozhraním jsou v naprosté většině také takto navrženy. Události budou vznikat primárně od uživatele a jeho interakce s uživatelským rozhraním, sekundárně pak na základě příchodů asynchronních zpráv od serverové části systému (např. různé typy upozornění). Pokud uživatel bude chtít například zobrazit všechna jeho monitorování, vyvolá se událost, jejíž obslužná rutina odešle požadavek na tato data serverové části systému. Poté je aplikace připravena obsluhovat další události – v tomto případě pravděpodobně příchozí zprávu od serverové části s požadovanými daty, které klient následně zobrazí uživateli.

4.2 Serverová část a databáze

Druhou komponentou je serverová část (angl. backend) provádějící samotný monitoring. Ta bude podle nastavené konfigurace autonomně testovat jednotlivé počítače a ukládat získané hodnoty do databáze. Zároveň upozorní aktivní klienty na výskyt kritických událostí a samozřejmě provede i další nastavená upozornění. Také bude klientům poskytovat informace o aktuální konfiguraci a umožňovat tato nastavení měnit. Na vyžádání klienta bude poskytovat statistické údaje získané z databáze.

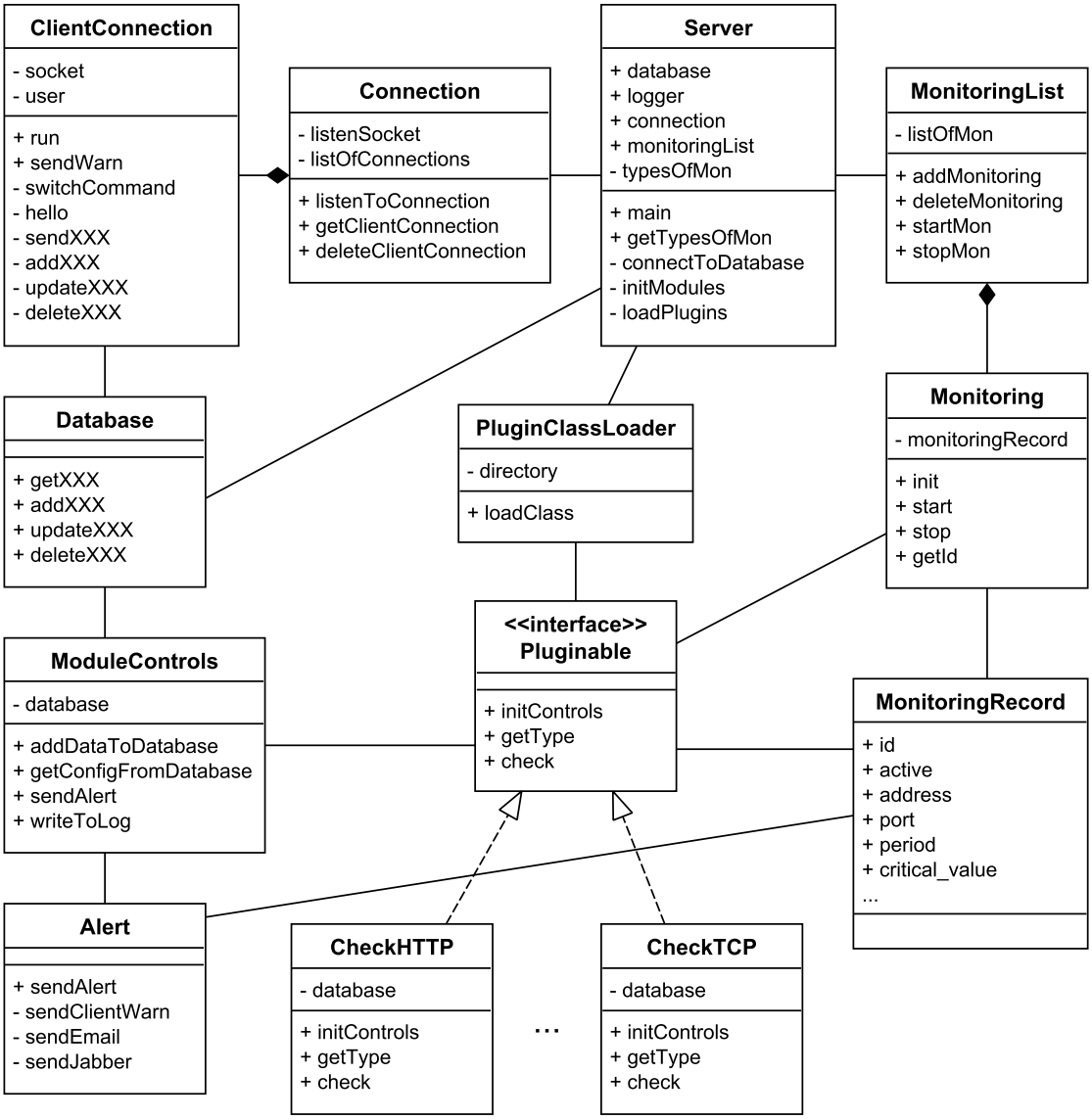
Tato část si po většinu času bude tzv. žít vlastním životem. Bude podle zadaných monitorování v určitých intervalech testovat jednotlivé počítače a získané hodnoty zapisovat do databáze. Pokud přijde od klienta požadavek na nějaká data, bude tento dotaz vyřízen přednostně a požadovaná data mu budou zaslána. V případě překročení kritické hodnoty nastavené v konfiguraci monitorování budou informováni všichni aktivní klienti přihlášení na daný účet a budou provedena všechna nastavená upozornění.

Ve standardním případě se počítá, že serverová část a databáze poběží na tomtéž serveru, avšak pokud by nastávaly problémy s nedostatečným výkonem jednoho serveru při větším množství sledování nebo si uživatel z jakéhokoliv důvodu přál mít tyto dvě komponenty odděleny, není problém provozovat databázi na odděleném serveru.

Stěžejním prvkem implementace serverové části bude systém testování nastavených serverů. Každý monitoring bude spuštěn v samostatném vlákně, kde se bude v pravidelném intervalu nastaveném uživatelem testovat patřičný server. Získaná data se poté uloží do databáze.

Důležitou vlastností bude možnost přidávat vlastní moduly nabízející další typy testování monitorovaných serverů. Tato schopnost bude velikým přínosem do budoucnosti, protože monitorovací systém nebude závislý na jediné množině poskytovaných funkcí a díky modularitě bude schopný se přizpůsobit nově vzniklým požadavkům.

Na obrázku 4.2 můžete vidět zjednodušený diagram tříd serverové části systému. V něm jsou zachyceny jednotlivé třídy, jejich metody, atributy a interakce mezi nimi. Kvůli předejití zbytečné složitosti jsou vynechány veškeré datové typy a kompletní výpisy nabízených metod.



Obrázek 4.2: Zjednodušený diagram tříd serverové části.

4.3 Agent

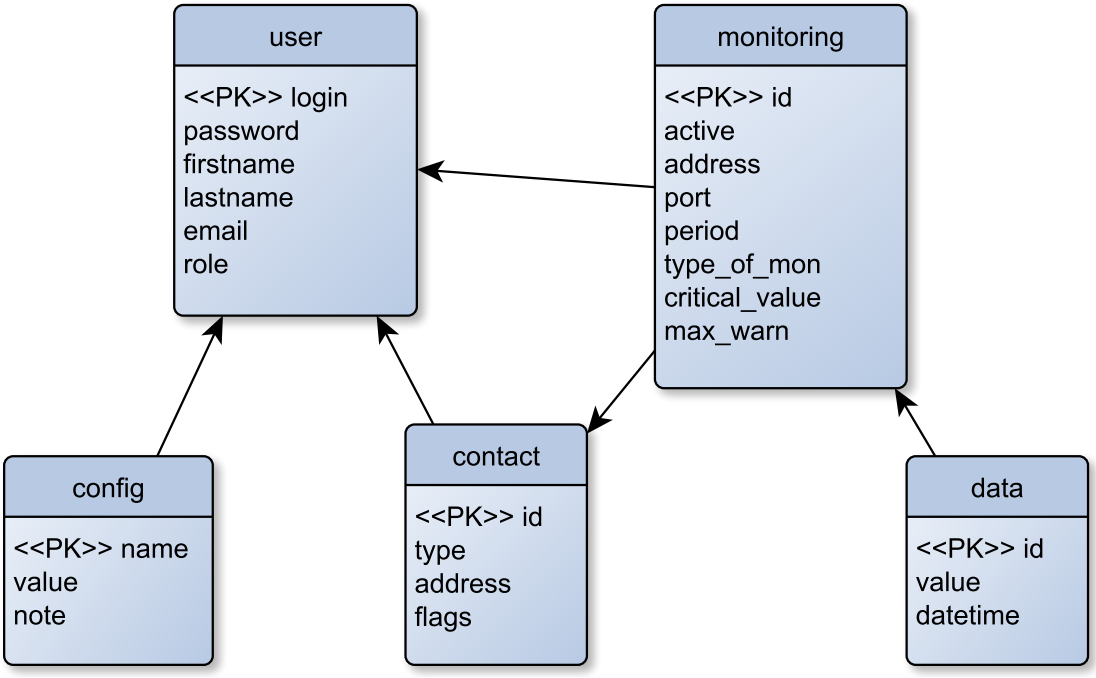
Koncovou částí je agent běžící na monitorovaném počítači. Jeho využití není povinné a slouží pro sbírání systémových údajů jako například vytížení procesoru, zbývající kapacita

pevného disku nebo obsazení paměti RAM. Tyto možnosti monitorování bude možné využít pouze na počítačích, na které má uživatel přístup a může na nich spustit tohoto agenta. Agent poběží na pozadí a bude naslouchat na zadaném portu. Pokud obdrží dotaz na konkrétní systémovou informaci, zjistí ji, odešle zpět a opět se přepne do naslouchacího módu. Pro komunikaci se serverovou částí systému bude sloužit podobný protokol, který je navržen pro komunikaci klienta. Pouze budou použita jiná, nová, klíčová slova.

4.4 Reprezentace a uchování dat

Jak již bylo zmíněno výše, veškerá data budou uchovávána v databázi, která standardně poběží na serveru společně se serverovou částí systému. Všechna data jsou tím pádem centralizovaná a mohou být jednoduše a pravidelně zálohována. V případě požadavku na vysokou spolehlivost by se tato data dala zajistit i záložním databázovým serverem. Největší výhodou ale zůstává možnost přístupu ke svým datům (konkrétně ke konfiguracím a statistikám) odkudkoliv, čehož bychom v případě decentralizovaného přístupu jen těžko dosahovali.

Nicméně tento přístup také pokládá poměrně vysoké nároky na rychlost databáze. V případě vysokého počtu jednotlivých monitorování a častých požadavků různých klientů na tato data vzniká velké množství operací (jak čtení, tak zápisu) nad touto databází. Naše řešení však necílí na komerční sféru ani na profesionální monitoring rozsáhlých sítí, a proto by s rychlostí databáze neměly nastat větší problémy.



Obrázek 4.3: Schéma databáze.

První důležitou položkou databáze je tabulka se záznamy o jednotlivých uživatelích. Ta krom dalšího uchovává především uživatelské jméno a heslo v zašifrované podobě. S těmito záznamy se bude hledat shoda při pokusu o přihlášení uživatele do systému. Další velmi důležitá tabulka obsahuje záznamy o jednotlivých monitorováních. Zde se uchovávají

hodnoty jako adresa počítače, typ monitorování, interval, kritická hodnota nebo kontakt, kam bude zasláno upozornění. S touto tabulkou úzce souvisí tabulka obsahující získané hodnoty včetně data a času jejich naměření. Uchovává se určitý počet starých záznamů pro tvorbu statistik a grafů. Dále si může uživatel definovat libovolný počet kontaktů, na které si pak může nechávat zasílat konkrétní upozornění. Tyto kontakty jsou uloženy v samostatné tabulce. Poslední tabulka nutná pro bezproblémový chod systému uchovává globální nastavení jednotlivých uživatelů. Schéma tabulek databáze je nastíněno na obrázku 4.3.

4.5 Komunikační protokol

Pro komunikaci mezi klientem a serverovou částí, ale také pro získávání dat od agentů, bude sloužit jednoduchý proprietární textový protokol navržený speciálně pro tyto účely. Hlavní důraz byl při jeho návrhu kladen na jednoduchost, snadné parsování, nízkou režii a intuitivnost jeho pochopení. Každá zpráva bude začínat jedním slovem (obecně řetězcem znaků) složeným z velkých písmen anglické abecedy, které jednoznačně určí typ zprávy. Poté můžou volitelně, podle typu zprávy, následovat data. Formát těchto dat je dán konkrétním typem zprávy. Konec zprávy je identifikován znakem konce řádku (\n).

4.6 Techniky testování

Z předchozích odstavců lze jednoduše usoudit, že testování bude probíhat dvojím způsobem podle typu získávané hodnoty. Ta může být buď síťového charakteru, tzn. dostupnost potažmo zpoždění paketů, nebo charakteru systémového, tzn. vytížení procesoru, zaplnění paměti RAM atd.

V případě testování síťové dostupnosti se spolehneme na již existující a ověřené techniky. Prvním způsobem bude využití protokolu ICMP[10] a jeho zpráv typu Request/Reply. Tento způsob využívá například nástroj ping a umožní nám zjistit, zda je daná IP adresa nebo doménové jméno dostupné. Tento způsob je nezávislý na číslu portu (ICMP operuje na síťové vrstvě). Druhým způsobem je prosté připojení na zadaný TCP nebo UDP port. Pokud se nám na tento port podaří připojit, můžeme o něm prohlásit, že je otevřený a měřit jeho odezvu. Poslední, konkrétněji zaměřenou, metodou je testování webového serveru pomocí zaslání dotazu GET protokolu HTTP[3]. Přesto, že webový server běží standardně na portu 80, nemusí tomu tak být vždy, a proto je možné i u této metody zadat jiný konkrétní port (další typické porty pro webový server jsou např. 8080, 8008).[4]

Pokud budeme chtít získat informace o aktuálně využitých systémových prostředcích, musíme se na ně zeptat agenta běžícího na monitorovaném serveru. S tímto agentem budeme komunikovat stejným protokolem, kterým komunikuje klientská aplikace se serverovou částí systému. Ta zašle agentovi požadavek na konkrétní hodnotu. Ten se po příchodu požadavku aktivuje, zjistí požadovaná data ze systému a odešle je zpět pro zapsání do databáze. Poté se opět přepne do režimu naslouchání na zadaném portu. Konkrétně bude tento agent poskytovat údaje o aktuálním vytížení procesoru, využití paměti RAM, zaplnění pevných disků a počtu běžících procesů. Tento typ monitorování bohužel nebude fungovat bez agenta běžícího na straně monitorovaného serveru, protože neexistuje jiná možnost, jak data o aktuálním využití systémových prostředků získat.

Kapitola 5

Implementace

Jelikož se jedná o vytvoření monitorovacího systému, navíc obsahujícího hned několik netriviálních částí, byla implementace největší a také nejdůležitější součástí této bakalářské práce. V následujících odstavcích naleznete popis řešení konkrétních principů a problémů, se kterými se můžeme v oblasti monitoringu, síťové komunikace, ale i programování pro mobilní systémy setkat.

5.1 Databáze

Struktura databáze nutná pro běh systému obsahuje 5 tabulek, jejichž podobu můžete vidět na obrázku 4.3. Součástí zdrojových kódů je i inicializační skript, který vytvoří v dané databázi všechny potřebné tabulky a relace. Zároveň ji naplní testovacím vzorkem dat, což zahrnuje vytvoření uživatele (login: admin, heslo: admin), jeho výchozí konfigurace a nastavení dvou ukázkových monitorování a jednoho kontaktu. Tento skript je vytvořen pro databázi MySQL, ale pro uživatele běžně znalého jazyka SQL a databázových technologií jistě nebude problém skript upravit do podoby pro další nejpoužívanější databáze.

Ke komunikace s databází je využito API *Java Database Connectivity* (JDBC)[1] nabízené přímo společností Oracle. To je dnes považováno za standard pro komunikaci aplikací v jazyce Java s databází, protože nabízí metody pro práci s daty nezávislé na typu databáze. Díky tomu je programátor aplikace odstíněn od specifických API, které jednotlivé databáze nabízejí a může ke všem typům přistupovat přes jednotné rozhraní nabízené JDBC. Ovladač pro toto rozhraní dnes již poskytují všichni velcí výrobci databázových řešení.

Pro připojení k databázi je nutné sestavit její URL. To bude použito při spouštění serverové části, o čemž se více dočtete v kapitole 5.2.1. Toto URL se skládá z řetězce `jdbc:` identifikujícího komunikaci přes API JDBC, protokolu závislého na použitém typu databáze (například `mysql://`), adresy a portu počítače, na kterém databáze běží v běžně používaném formátu `adresa:port` (například `localhost:3306`) a cestě ke konkrétní databázi, kde jsou jednotlivé úrovně odděleny znakem `/`. Výsledné URL pak může vypadat například takto: `jdbc:mysql://localhost:3306/monitoring`.[2]

Práci s databází má v serverové části na starost objekt `Database`, jehož instance je přístupná jako veřejná proměnná hlavní třídy `Server`. Tento objekt nabízí řadu metod sloužících pro získávání, přidávání, úpravu a mazání dat v databázi.

Z hlediska paměťové náročnosti databáze bylo nutné vyřešit problém jejího nekonečného zaplňování stále novými daty získanými testováním jednotlivých monitorovaných serverů.

[1] Více informací na http://www.oracle.com/technetwork/java/overview-141217.html.

Proto byla do globální konfigurace uživatele přidána položka `lengthOfHistory` (s výchozí hodnotou 100), která určuje maximální počet záznamů uchovávaných pro jeden monitoring. Toto nastavení je zároveň shora omezeno hodnotou 1 000, aby se předešlo nerozvážné konfiguraci ze strany uživatele ovlivňující běh a plynulost databáze. Po každém vložení nových dat je zkontrolován počet záznamů, a pokud je větší než nastavená hodnota, je smazán patřičný počet nejstarších záznamů. Díky tomuto principu je udržena přijatelná velikost celé databáze i při větším množství uložených monitoringů.

5.2 Serverová část

Serverová část slouží jako jádro celého systému. Její funkce jsou v monitorování nastavených serverů, obsluze připojených klientů a práci s databází. Implementaci serverové části představuje aplikace v programovacím jazyce Java bez grafického nebo jiného konzolového rozhraní.

5.2.1 Spuštění

Pro spuštění serverové části je nutné zadat programu dva parametry. Prvním z nich je číslo portu, na kterém bude program naslouchat pro příchozí požadavky klientských aplikací. Jeho omezení jsou vcelku jasná – musí být zadáno celé kladné číslo v rozsahu 1 – 65 535. Doporučuje se vybírat číslo portu větší než 1023, kde je menší šance kolize s jinou službou. Pokud bude přesto port obsazený, aplikace to oznámí a následně se ukončí. V tomto případě je nutné zvolit jiné číslo portu a spustit aplikaci znovu.

Druhým parametrem je textový soubor obsahující údaje potřebné pro přihlášení k databázi. Tento soubor může být zadán relativní nebo absolutní cestou a musí obsahovat pouze prostý text. Na prvním řádku musí být URL databáze, ke které se chceme připojit. Vytvoření tohoto URL je popsáno v kapitole 5.1. Na druhém řádku musí být uživatelské jméno, kterým se chceme k databázi přihlásit. Tento uživatel musí mít práva pro čtení i zápis do databáze specifikované na prvním řádku. Na třetím řádku může a nemusí být heslo pro přihlášení k databázi výše uvedeného uživatele. Další řádky jsou ignorovány. Pokud budou v souboru uvedeny všechny tři potřebné údaje, aplikace se pokusí ihned po spuštění připojit k databázi. V případě, že v souboru není uloženo heslo, bude uživatel vyzván k jeho zadání při každém spuštění aplikace.

Program pro dodržení dobrých mravů taktéž reaguje na parametr `-h` nebo `--help`, který vypíše nápovědu programu a jeho použití.

Po spuštění si aplikace vytvoří logovací soubor (pokud již neexistuje), do kterého se zapisují případné vzniklé chyby a další informace. Ten může sloužit administrátorovi pro detekci problému monitorovacího systému nebo jeho špatnou konfiguraci.

Mezi poslední položky úvodní sekvence příkazů patří spuštění naslouchání příchozích požadavků na zadaném portu. To je spuštěno ve vlastním vlákně. Více o obsluze připojených klientů se dozvíte v kapitole 5.2.2. Poté se inicializují vestavěné a přídavné moduly. Tento koncept modularity je podrobněji rozebrán v kapitole 5.2.5. Nakonec se spustí samotné monitorování popsané v kapitole 5.2.3.

5.2.2 Obsluha klientů

Po inicializaci objektu zajišťujícího komunikaci s klienty se metodou `listenToConnection()` spouštěnou ve svém vlastním vlákně zahájí naslouchání pro příchozí spojení. Pokud se

nějaký klient pokusí k serveru připojit, vytvoří se nový objekt typu `ClientConnection`, který toto spojení reprezentuje a především realizuje. Díky tomu, že tato třída implementuje rozhraní `Runnable`, je možné její instanci přímo využít pro vytvoření nového vlákna. Jako parametr bude této třídě předán síťový soket, nad kterým bude v inicializační části vytvořen vstupní (`BufferedReader`) a výstupní (`BufferedWriter`) textový datový kanál (angl. stream). Poté tato instance začne čekat na vstupním kanálu, dokud nepřijde nějaký klientský požadavek. Pokud ještě není uživatel přihlášen, na všechny požadavky s výjimkou přihlášení a registrace je odpovídáno záporně. Vlákno realizující naslouchání se ihned po vytvoření objektu nového spojení a jeho spuštění v novém vlákně vrací v nekonečně smyčce zpět k naslouchání dalším spojením.

Po přijetí klientského požadavku je na základě prvního slova dekódována požadovaná akce a je zavolána patřičná funkce. Pokud se jednalo o požadavek na data, budou tato data získána z databáze a odeslána zpět v patřičném formátu popsaném v kapitole 5.5. V případě, že se jednalo o požadavek na provedení nějaké akce, bude tato akce provedena a zpět odesláno potvrzení o úspěchu (`ACCEPT`). Pokud se vyskytne chyba nebo budou data či akce odepřeny, odesílá se zpět klientské aplikaci informace o neúspěchu (`DENIED`).

Celá komunikace je tedy iniciována ze strany klienta a server zde zastává roli pouze posluchače a reaguje na příchozí požadavky. Jediným momentem, kdy do komunikace vstupuje server aktivně, je zasílání upozornění (`WARN`). To vznikne nezávisle na komunikaci v části zabývající se monitorováním a je zasláno klientovi asynchronně k ostatní komunikaci. Blíže je tento princip a celý systém upozornění popsán v 5.5.

5.2.3 Monitorování

O inicializaci a řízení monitorování se stará třída `MonitoringList`. Ta si ve svém konstruktoru získá z databáze všechna monitorování a postupně je inicializuje a uloží do seznamu `listOfMon` pro pozdější přístup k nim. Z každého získaného záznamu o monitorování se nejprve vytvoří třída `MonitoringRecord`, která obsahuje veřejně přístupné proměnné odpovídající parametrům monitoringu. Ta je využita pro inicializaci třídy každého jednotlivého monitorování nazvanou jednoduše `Monitoring`.

Třída `MonitoringList` nabízí metody pro řízení monitorování. Jednotlivé monitoringy lze přidávat, mazat, spouštět či zastavovat. Poslední dvě metody přijímají jako parametr identifikátor monitoringu, který je potřeba spustit nebo zastavit. Tento parametr je vyhledán v množině inicializovaných monitorování a akce je delegována přímo na objekt monitoringu.

V konstruktoru třídy `Monitoring` se v případě aktivního (spuštěného) monitorování zavolá metoda `start()`. Ta vytvoří novou instanci třídy `Timer` z balíčku `java.util`. Tento časovač se nastaví na pravidelné opakování v intervalu daném periodou monitoringu a bude volat funkci `check(record)` z modulu odpovídajícího typu monitorování. Časovač po jeho spuštění běží implicitně ve vlastním vlákně, což se nám velice hodí a díky tomu budou všechna monitorování na sobě nezávislá. Zpoždění před spuštěním časovače je generováno náhodně v rozmezí 0 až 1 000 ms kvůli desynchronizaci najednou spouštěných monitorování a předejití konfliktů při žádostech o zdroje (například dotaz na databázi či přístup k síťové lince).

Zjištění monitorované hodnoty, ať již odezvy nebo některého ze systémových prostředků, zůstává na zodpovědnost jednotlivým modulům. Systém obsahuje celkem tři vestavěné moduly:

- **CheckHTTP** testující dostupnost (odezvu) webových serverů. Tento modul neřeší

návratový kód webové stránky, pouze její odezvu (v ms).

- **CheckTCP** testující, zda je zadaný TCP port otevřený a s jakou odezvou se k němu dá připojit (v ms).
- **CheckUDP** testující, zda je zadaný UDP port otevřený. Měří se doba odpovědi na zaslaná data. Tento modul nezaručuje spolehlivé měření z důsledku nespolehlivosti protokolu UDP (v ms).

V systému jsou dále moduly, které jsou zaváděny dynamicky:

- **CheckICMP** testuje dostupnost (odezvu) serveru pomocí metody `isReachable()` objektu `InetAddress`. Tento způsob však nezaručuje použití protokolu ICMP a může se chovat odlišně na různých hostitelských systémech (v ms).
- **RAMFree** testuje množství volné paměti RAM na monitorovaném systému (v MB).
- **DiskFree** testuje volnou kapacitu na systémovém disku (v MB).
- **CpuPerc** testuje vytížení procesoru (v promile).
- **ProcessCount** testuje počet spuštěných procesů v monitorovaném systému.

Poslední čtyři moduly vyžadují pro svoji správnou funkčnost spuštěného agenta na monitorovaném serveru. Více se o koncepci modulů dozvíte v 5.2.5.

5.2.4 Upozornění

Možnost upozornění na výskyt sledované události je velice důležitou součástí celého systému. Právě díky této schopnosti se můžeme dovědět o výpadku serveru například v tramvaji cestou do práce.

Odesílání upozornění zajišťuje třída `Alert`, která poskytuje jedinou veřejnou metodu `sendAlert()`. Po jejím zavolání se z databáze získá kontakt, který je u monitoringu nastaven a podle jeho typu se odešle patřičné upozornění. Poté, pokud tato vlastnost není zakázána v globálním nastavení uživatele, se odešle upozornění všem připojeným klientům daného uživatele.

Tato třída též nabízí tři různé délky textu upozornění, které jsou voleny dle nastavení každého kontaktu.

- *Krátké upozornění* se může hodit například při přeposílání emailů SMS zprávami, kde je omezený počet znaků.
- *Standardní upozornění* slouží jako výchozí délka upozornění. Posílá se také všem připojeným klientům.
- *Dlouhé upozornění* obsahuje více detailů o vzniklém problému a aktuální konfiguraci monitoringu.

Prvním typem upozornění je odeslání *emailu*. Pro implementaci této funkce je využito API *JavaMail*[2], které je pro platformu Java SE poskytováno zdarma jako volitelný balíček.

[2] Více informací na http://www.oracle.com/technetwork/java/javamail/index.html.

V nastavení každého uživatele je uložen emailový účet, ze kterého budou upozornění posílána. Po přihlášení k tomuto účtu je sestaven nový email a odeslán na adresu nastavenou u monitoringu. Tento typ lze taktéž použít pro upozornění SMS zprávou. Při využití služby typu „SMS mail“ stačí zadat patřičnou emailovou adresu a vybrat zasílání krátkého upozornění, pokud je omezen počet znaků zprávy.

Druhým typem upozornění je odeslání zprávy protokolem *XMPP (Jabber)*. Zde je využita knihovna *Smack*[3] dostupná pod licencí Apache poskytující rozhraní pro práci s tímto protokolem. Stejně jako u upozornění emailem je i zde v nastavění uživatele uložen účet, ze kterého se budou upozornění odesílat. Princip odeslání zprávy je velice jednoduchý – nejprve dojde k připojení k Jabber serveru a přihlášení k nastavenému účtu, poté je odesláno upozornění patřičné délky a nakonec proběhne uzavření spojení se serverem.

5.2.5 Modularita

Jednou z velice důležitých vlastností celého monitorovacího systému je *modularita*, kterou zajišťuje především serverová část. Funkčnost systému tak není omezena pouze na případy, na které se myslelo při jeho tvorbě, ale podle vznikajících požadavků lze jednoduše doplňovat potřebnou funkcionalitu. Vytvářením nových modulů lze přidávat další typy a způsoby testování monitorovaných serverů. To umožňuje testovat odezvu pomocí dalších, méně obvyklých, protokolů, získávat si data od vlastních agentů a mnoho dalšího.

Moduly jsou akceptovány ve formě přeložené třídy jazyka Java (soubor s příponou `class`). Tato třída musí implementovat rozhraní `Pluginable`, které vyžaduje vytvořit metody nutné pro správnou funkci modulu. Těmi jsou funkce `initControls()`, která je volána vždy po inicializaci modulu a je jí předána instance třídy `ModuleControls` (více o této třídě se dozvíte v dalších odstavcích). Další se nazývá `getType()` a slouží k identifikaci modulu v systému. Tato funkce musí vracet řetězec neobsahující mezery ani jiné tzv. bílé znaky, který by měl sloužit jako název typu monitorování poskytovaného daným modulem. Poslední metoda se nazývá `check()` a je tím nejdůležitějším z celého modulu. Zde by mělo být implementováno samotné testování popřípadě jiný způsob získávání hlídané hodnoty. Tato funkce je volána vždy, když monitoring s patřičným typem má po dané periodě opět zjistit stav monitorované hodnoty. Zde je také namístě využít instanci třídy `ModuleControls`.

Instance této třídy je po inicializaci předána každému modulu a je pouze na něm, jak s ní naloží. Doporučený postup však je si instanci uložit do soukromého atributu třídy pro její pozdější využití v metodě `check()`. Třída `ModuleControls` totiž obsahuje funkce potřebné pro interakci modulu se zbytkem systému. Jsou zde metody `addDataToDatabase()` pro uložení získaných dat do databáze, `getConfigFromDatabase()` pro získání hodnoty daného nastavení, `sendAlert()` pro odeslání upozornění na překročení kritické hodnoty a `writeToLog()` pro zapsání informace (například chyby při získávání hlídané hodnoty) do logovacího souboru serverové části. Poskytnutím pouze instance této třídy mohu jejím obsahem striktně určit, které funkce tvůrci dodatečného modulu poskytnu a především, jak moc ho nechám zasahovat do celého systému a databáze. To snižuje riziko nepovolaného zásahu do systému, zvyšuje bezpečnost uložených dat a snaží se předejít narušení správného běhu aplikace.

Po spuštění serverové části se nejprve načtou vestavěné moduly (`CheckHTTP`, `CheckTCP` a `CheckUDP`). Poté se spustí načítání přídavných modulů tzv. pluginů. Ty jsou vyhledávány ve složce `plugins`, kde jsou všechny ostatní nepoužitelné soubory (jiné než končící `.class`) ignorovány. Pro jejich načítání slouží třída `PluginClassLoader` dědící z abstraktní třídy

[3] Více informací na http://www.igniterealtime.org/projects/smack/.

`ClassLoader`. Ta se nejdříve pokusí požadovanou třídu najít v již načtených a v systémových třídách. Pokud ji nenalezne, načte ji sama z definovaného umístění. Po úspěšném načtení již proběhne pouze kontrola, zda třída skutečně implementuje potřebné rozhraní a je uložena do množiny načtených modulů `HashMap<String, Pluginable> modules` pro její další použití. Zde můžeme díky typu `String` vidět, že je modul skutečně reprezentován a také identifikován řetězcem, který je získán z již zmiňované funkce `getType()`.

Z předchozího odstavce je patrné, že zásuvný modul lze jednoduše přidat jeho nahráním do složky `plugins` a následným restartováním serverové části kvůli opětovnému načtení všech modulů. Přesto toto nemusí být vždy příliš pohodlný způsob, a proto desktopový klient (samozřejmě s podporou v komunikačním protokolu) nabízí možnost nahrání zásuvného modulu přímo z uživatelského rozhraní s automatickým znovu načtením všech přídavných modulů bez nutnosti restartu serveru. Protože by zajisté nebylo z bezpečnostních důvodů vhodné spouštět na serveru jakýkoliv cizí kód (předpokládá se, že fyzický přístup k serveru má pouze oprávněná osoba), má možnost nahrání nového zásuvného modulu pouze uživatel v roli administrátora.

Napsání nového zásuvného modulu je principiálně velice jednoduché. Vytvoříme novou třídu v libovolném balíčku, která implementuje rozhraní `Pluginable`. Poté implementujeme všechny metody vyžadované rozhraním. Při překladu je nutné přidat jako knihovnu `*.jar` balíček serverové části, aby překladač nalezl potřebné třídy. Výsledek může vypadat například nějak takto (jsou vynechány importy a deklarace balíčku):

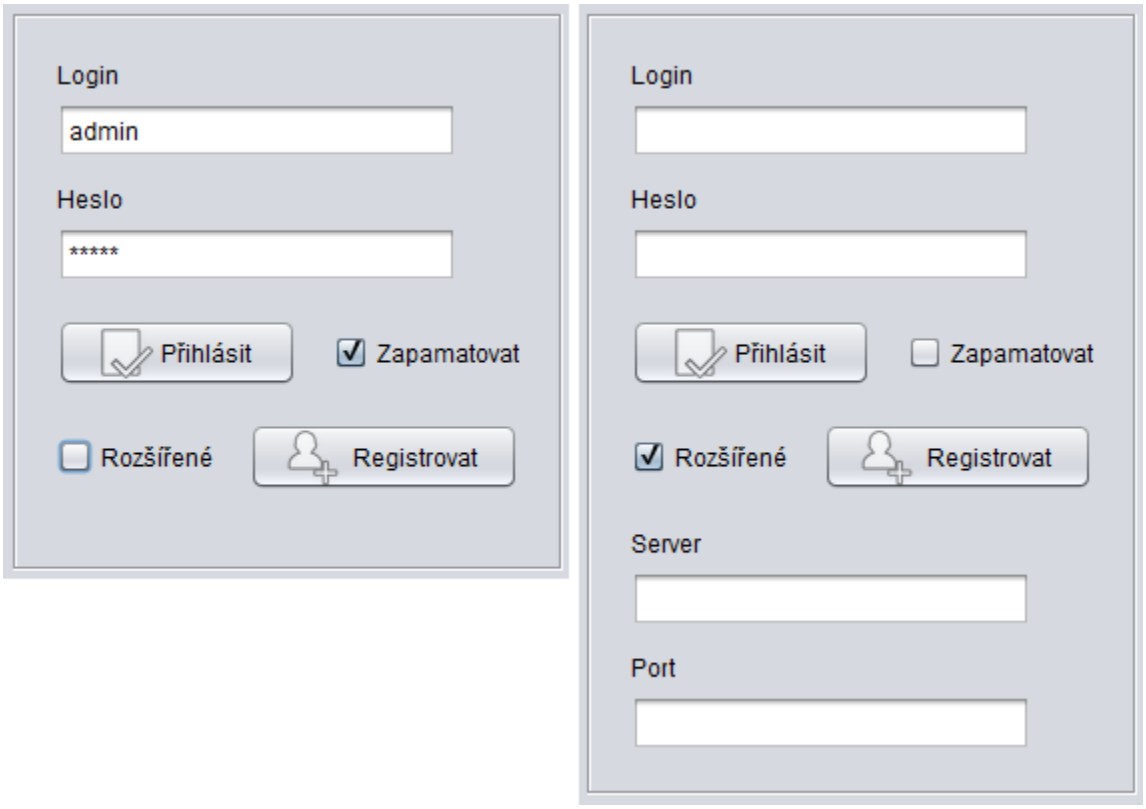
```
public class MujNovyPlugin implements Pluginable {

        private ModuleControls controls;

        @Override
        public void initControls(ModuleControls controls) {
                this.controls = controls;
        }

        @Override
        public String getType() {
                return "MujNovyPlugin";
        }

        @Override
        public void check(MonitoringRecord record) {
                // Implementace testovani
        }
}
```

5.3 Klientské aplikace

Klientské aplikace jsou jediným rozhraním pro ovládání celého systému. Přesto, že jsou obě aplikace napsány v jazyce Java a mnoho funkčnosti je řešeno stejně, specifika mobilního systému Android jsou natolik nezanedbatelná, že některé problémy musely být řešeny speciálně na míru tomuto systému. V následujících odstavcích se dozvíte o společných rysech

obou aplikací, dále pak v samostatných podkapitolách specifika implementace jednotlivých klientů.

Po spuštění klientské aplikace by jistě uživatel nebyl nadšen, kdyby pokaždé musel vyplňovat přihlašovací údaje a především adresu serveru a jeho port. Navíc by neustálé vyplňování údajů o serveru nepřispívalo ani k transparentnosti jeho existence, o kterou se celý systém snaží. Proto jsou po prvním zadání tyto údaje uloženy a při dalším spuštění klienta opět načteny. Pole s adresou serveru a portem jsou v tomto případě skryta (viz obrázek 5.1), aby přihlašovací dialog maximálně navozoval atmosféru lokální aplikace. Zatržením přepínače Zapamatovat se zároveň uloží i zadané uživatelské jméno a heslo a nebude je nutné při příštím spuštění opět zadávat. Tato možnost se z důvodů bezpečnosti nedoporučuje za okolností, kdy se může k počítači dostat i jiná, neoprávněná, osoba. Konkrétní způsob uložení přihlašovacích údajů je popsán v kapitolách o jednotlivých klientech.



Obrázek 5.1: Rozdíl v přihlašovacím dialogu s načtenými údaji a bez nich.

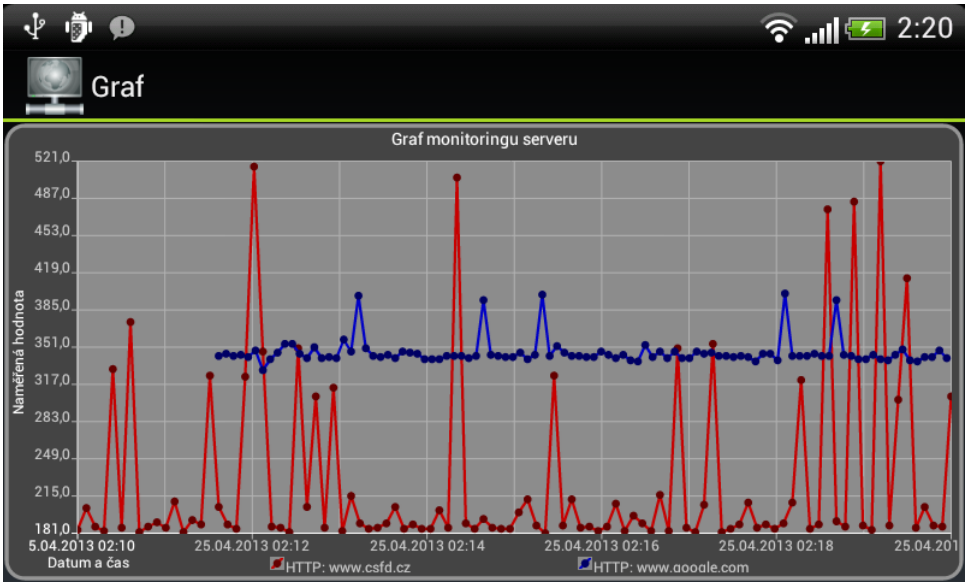
Po kliknutí na tlačítko Přihlásit se klient pokusí připojit k serveru a následně mu odešle požadavek o přihlášení s uživatelským jménem a heslem. Protože komunikační kanál mezi klientem a serverem je textový a nijak zabezpečený, nebylo by vhodné odesílat heslo v podobě, jak ho uživatel zadal. Zároveň nepatří mezi dobré praktiky ukládat hesla do databáze v čitelné podobě. Proto je před odesláním na server heslo v klientské aplikaci zakódováno hashovací funkcí *MD5*. Ta z libovolného vstupu dat vytvoří jeho otisk fixní délky, kde malá změna dat na vstupu vede k velké změně výsledného otisku. Protože se jedná o hashovací funkci, je algoritmus MD5 pouze jednosměrný. Zpětná rekonstrukce původních dat je proto velice náročná, přesto dnes již není nemožná.[11] Na serveru se proto porovnávají pouze otisky originálních hesel. Pro vytvoření otisku hesla je v aplikaci využita

třída `MessageDigest` a její metoda `digest()`, která ze vstupní sekvence bajtů obsahující zadané heslo vytvoří pole bajtů s jeho otiskem.

Jakmile se klientská aplikace přihlásí k serveru, je spuštěno pravidelné zasílání požadavku `HELLO` kvůli kontrole funkčnosti spojení. Více informací o tomto principu se dočtete v kapitole 5.5. Pokud klient nedostane odpověď na tento požadavek, může se jednat buďto o jednorázový problém v komunikaci nebo o nefunkčnost serverové části způsobenou nekorektním během nebo jeho ukončením. Proto je zde implementován čítač, který po dvou po sobě následujících výpadcích klienta odhlásí. V případě nefunkční komunikace by setrvání v přihlášeném stavu nemělo žádný význam, protože veškerá data jsou načítána ze serveru a taktéž všechny akce jsou tam delegovány. Proto by snaha o práci s klientem způsobovala pouze výpisy chybových hlášení bez dalšího užitku.

Vlastnost inspirovaná aplikací Server Down Alarm zmíněnou v 2.1.2 je možnost odeslání *simulovaného upozornění*. Po nastavení kontaktu a konfiguraci monitoringu může uživatel chtít zjistit, zda vše nastavil dobře, nepřeklepl se například v adrese kontaktu nebo ho pouze zajímá, jak bude případné upozornění vypadat a zda mu vyhovuje jeho formát. K tomuto účelu je u každého monitoringu přidána možnost odeslat toto simulované upozornění, které se chová a vypadá přesně, jako kdyby se jednalo o běžné upozornění odeslané na základě překročení nastavené kritické hodnoty.

V jednotlivých obrazovkách a prvcích klientů bylo často potřeba vybrat nebo identifikovat konkrétní monitoring. Přesto, že v databázi a celém systému je každé monitorování jednoznačně rozlišeno jeho číselným identifikátorem, rozhodně se nejedná o způsob vhodný pro prezentování uživateli. Proto byl jako nejvhodnější zvolen formát *typ: adresa*. Tento řetězec určitě není jednoznačným identifikátorem, nicméně pro prezentaci uživateli je jednoduchý, dostatečně vypovídající a v praxi není příliš pravděpodobné, že by docházelo k duplicitě těchto řetězců.



Obrázek 5.2: Ukázka zobrazení grafu v mobilní klientské aplikaci.

Velice důležitou součástí pro celkový uživatelský dojem je možnost zobrazit naměřená data ve formě *grafu*. Nejprve je inicializován prázdný graf, kde bude na ose X zobrazeno datum a čas a na ose Y naměřená hodnota. Tato hodnota musí být bezrozměrná, protože jednotlivé moduly měří různé veličiny (odezva v milisekundách, údaje o operační paměti

nebo pevném disku v megabajtech atd.). Poté je do grafu přidána tzv. série obsahující naměřená data. Hodnoty osy X jsou pro správné proporcionální zobrazení převedeny na číslo typu `long` pomocí funkce `getTime()` třídy `Date`. Ta převede hodnotu data a času na jediné celé číslo reprezentující počet milisekund od 1. ledna 1970, 00:00:00 GMT. Nastavením správného formátu zobrazování hodnot na ose X jsou v grafu opět data a časy čitelné uživateli. Právě z důvodů proporcionálního zobrazení jsou hodnoty na ose X rovnoměrně rozloženy mezi první a poslední naměřený údaj. Toto chování může udělat graf velmi nepřehledným v případě přerušovaného zapínání monitoringu (nespojité sekvence naměřených hodnot). Přesto se tento způsob zobrazení jeví jako nejlepší možnost.

Kvůli snadnému odhalování slabých míst síťové infrastruktury nebo hardwaru serveru by bylo vhodné vidět data naměřená konkrétním modulem v kontextu ostatních. Proto je umožněno proložit několik sérií naměřených dat do jednoho grafu, jak můžete vidět na obrázku 5.2. Díky dodržování proporcí osy X budou stejné časové okamžiky vyneseny přesně nad sebe. Tato schopnost může správci sítě velice usnadnit například diagnostiku, zda má webový server pomalou odezvu z důvodu zahlcení linky nebo nadměrného vytížení procesoru.

U desktopové verze klienta je pro vykreslování grafů využita knihovna *JFreeChart*[4] dostupná pod licencí LGPL. Jedná se o poměrně rozsáhlou knihovnu pro tvorbu a zobrazování grafů v prostředí programovacího jazyka Java, ze které naše aplikace využívá pouze zlomek jejího potenciálu. Přesto je knihovna velice snadná na použití a vytvoření grafu je otázkou několika desítek řádků komentovaného kódu. Pro správné fungování vyžaduje podpůrnou knihovnu JCommon od stejných tvůrců.

Mobilní klient používá pro vykreslování grafů knihovnu *AndroidPlot*[5] dostupnou pod licencí Apache, což je, na rozdíl od předchozí zmiňované knihovny, malý balík funkcí a metod, které slouží pro zobrazování grafů na platformě Android. Vlastnosti knihovny přesně splňují požadavky naší aplikace a výsledný graf dobře zapadá do prostředí systému Android.

5.3.1 Desktopový klient

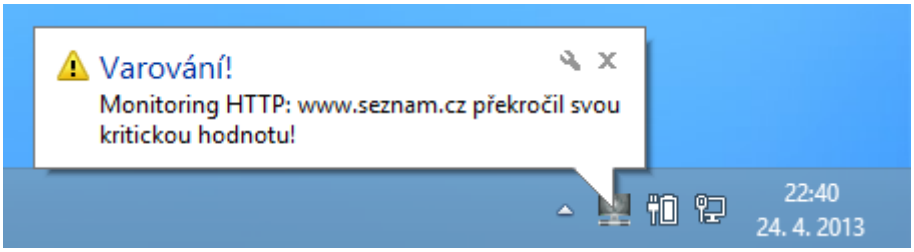
Klientská aplikace určená pro osobní počítače je postavena na principu záložek (tzv. tabů). Cílem je neobtěžovat uživatele množstvím vyskakujících dialogových oken, potvrzovacích dialogů apod. Ihned po přihlášení je otevřena záložka se seznamem jednotlivých monitorování. Akcemi jako jsou přidání nebo editace monitorování, zobrazení grafu nebo změna nastavení, jsou otvírány nové záložky, mezi kterými je dále možné přepínat. Této funkčnosti je docíleno využitím komponenty `jTabbedPane`. Jako nové záložky jsou otevírány i dialogové výzvy požadující po uživateli rozhodnutí.

Pro dodržení konceptu neobtěžování uživatele vyskakujícími okny bylo nutné navrhnout metodu jeho informování o zdaru či neúspěchu jednotlivých akcí. Pro tento účel slouží informační nápis umístěný ve stavové liště. Ten zobrazí po určitý okamžik informaci, kterou je třeba předat uživateli. Pokud je zpráva pouze informativního charakteru, například potvrzení úspěšného přidání kontaktu, je nápis zobrazen výchozí černou barvou. V případě, že se jedná o upozornění na chybu či nezdar, například nemožnost vytvořit nový monitoring, protože nebyly vyplněny všechny položky, je nápis zobrazen výraznou červenou barvou.

Protože je klientská aplikace určená k dlouhodobému běhu a měla by uživatele co nejméně obtěžovat, po kliknutí na tlačítko minimalizace je okno minimalizováno do oznamovací oblasti systémové lišty. Ikonka aplikace v systémové liště poskytuje také kontextové

[4] Více informací na http://www.jfree.org/jfreechart/.
[5] Více informací na http://androidplot.com/.

menu, kterým lze aplikaci částečně ovládat. Takto minimalizovaná aplikace neruší uživatele při běžné práci a zároveň umožňuje zobrazování upozornění ve formě bublinových varování, jak můžete vidět na obrázku 5.3. Po kliknutí levým tlačítkem myši na minimalizovanou ikonku se opět vyvolá okno aplikace. Pokud není možné vytvořit v hostujícím operačním systému ikonku v systémové liště (příkladem může být grafické prostředí Unity), je tlačítku minimalizace zachována jeho standardní funkce a uživatel je ochuzen o výše popsané možnosti.



Obrázek 5.3: Upozornění desktopového klienta bublinou v oznamovací oblasti systémové lišty.

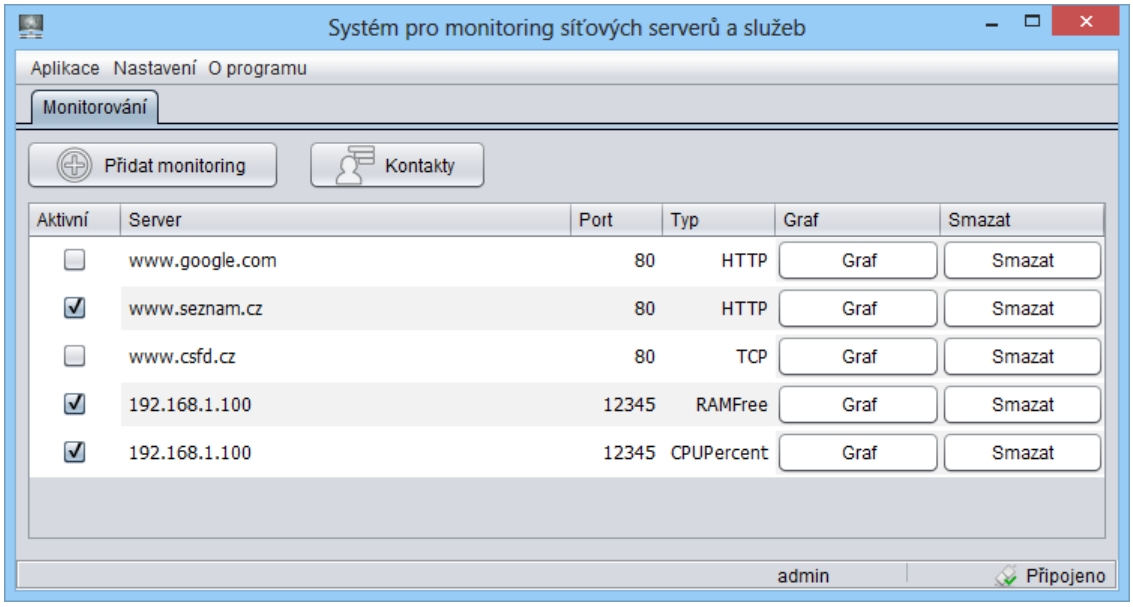
Zapamatování přihlašovacích údajů pro další spuštění je vyřešeno jejich uložením do souboru ve formátu XML[6]. Za tímto účelem je využita knihovna *dom4j*[7] dostupná pod licencí stylu BSD, která poskytuje rozhraní pro práci s XML, XPath a XSLT. Po spuštění aplikace je soubor načten, zpracován a hodnoty uloženy do struktury `HashMap<String, String>`, která umožňuje uložení dvojic klíč/hodnota. Z této struktury jsou poté naplněny jednotlivá textová pole v přihlašovacím formuláři. Pokud soubor neexistuje (pravděpodobně první spuštění), budou textová pole ponechána prázdná a pro uložení nových hodnot bude vytvořen nový soubor. Po stisku tlačítka Přihlásit se zadané údaje zapíší opět ve formátu XML zpět do souboru.

Jediným rozdílem desktopové aplikace od její mobilní verze je možnost nahrávat na server nové zásuvné moduly. I přes schopnosti dnešních chytrých telefonů nejsou stále tyto přístroje nástrojem k programování, a proto byla možnost nahrání nového zásuvného modulu ve formě třídy jazyka Java ponechána pouze desktopové aplikaci. Tuto možnost má navíc z důvodů bezpečnosti pouze uživatel v roli administrátora. Po kliknutí na patřičnou položku v menu se objeví dialog pro výběr zásuvného modulu. Po jeho vybrání je soubor načten, převeden do datového formátu Base64 a odeslán na server. Po úspěšném nahrání se na serveru provede znovu načtení všech modulů.

Algoritmus *Base64* [5] reprezentuje ideální formát pro přenos binárních dat, jako jsou v našem případě přeložené třídy jazyka Java ve formátu *.class, po textovém přenosovém kanálu. Tato binární data jsou zakódována pomocí tisknutelných znaků ASCII, které již není problém odeslat po textové lince. Jedinou nevýhodou tak je navýšení objemu přenášených dat obyčejně o jednu třetinu.

Na obrázku 5.4 můžete vidět hlavní obrazovku (záložku) desktopové verze klienta, která zobrazuje nastavená monitorování. Zaškrtávací políčko (angl. checkbox) slouží pro zapínání a vypínání monitoringu, dále je zobrazena adresa, port a typ monitorování. U každého záznamu jsou k dispozici tlačítka pro zobrazení grafu nebo smazání. Detaily, statistiky a zároveň editace hodnot monitoringu se zobrazí po dvojkliku na konkrétní řádek. Okno

[6] Extensible Markup Language – více informací na http://www.w3.org/TR/REC-xml/.
[7] Více informací na http://dom4j.sourceforge.net/.



Obrázek 5.4: Seznam monitorování zobrazený v desktopové klientské aplikaci.

aplikace dále nabízí jednoduché menu pro ovládání programu a stavovou lištu obsahující informace o stavu připojení a přihlášeném uživateli. Na této liště jsou také zobrazovány zprávy pro uživatele o úspěchu či neúspěchu jeho příkazů.

5.3.2 Klient pro mobilní systém Android

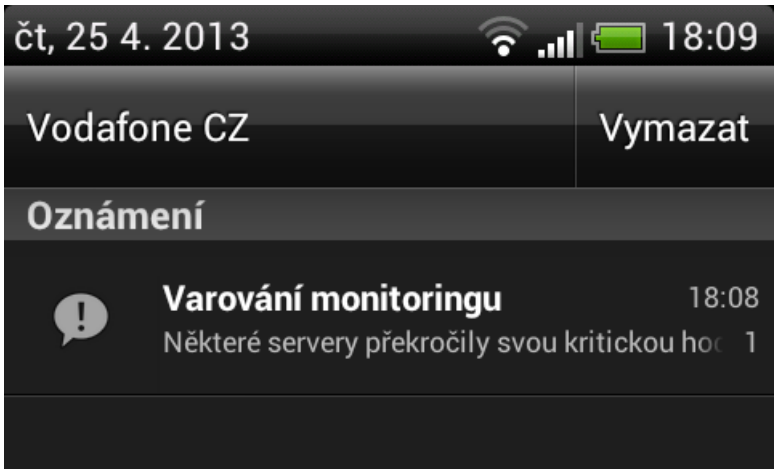
Implementace klienta pro mobilní systém má zajisté svá specifika, jako například běh na, v jisté míře, omezeném hardwaru zařízení nebo kvůli úspoře energie naprosto odlišný životní cyklus na rozdíl od klasické desktopové aplikace. Z důvodů udržení spojení i po pozastavení aplikace je třída, která se stará o spojení se serverovou částí, implementována jako služba – `public class ConnectionService extends Service`. Ta se spustí společně s aplikací a její běh je udržován v systému, dokud je k ní nějaká aktivita připojena. Tato služba udržuje spojení se serverovou částí systému a nabízí připojeným aktivitám rozhraní pro ovládání komunikace. Struktura této třídy je tedy velmi podobná obdobné třídě v desktopové aplikaci a poskytuje metody pro získávání, přidávání, úpravu a mazání dat, se kterými pracuje serverová část.

Pro uložení zapamatovaných přihlašovacích údajů zde slouží tzv. *preference* (angl. preferences). Použitím třídy `SharedPreferences` můžeme vytvořit množinu dvojic klíč/hodnota, které budou sdíleny všemi komponentami aplikace. Tyto preference podporují uložení datových typů `boolean`, `String`, `float`, `long` a `int`, což ideálně poslouží pro uložení všech potřebných hodnot. Tento princip je nejčastěji využívaným způsobem pro uložení dat mezi jednotlivými spuštěními aplikace. Pro čtení preferencí je využívána přímo instance třídy `SharedPreferences` získaná metodou `getSharedPreferences()`. Pro editaci pak slouží třída `SharedPreferences.Editor`. Po zápisu všech změn je nutné provést jejich potvrzení metodou `commit()`.[8]

S výhodou je využito i dalšího prvku uživatelského rozhraní typického pro systém Android. Tím je poloprůhledný informační text nazvaný anglicky *toast*. Ten je perfektní pro informování uživatele bez násilného vyžadování jeho pozornosti nebo přerušování aktivity

v popředí. Třída `Toast` obsahuje statickou metodu `makeText()`, která vytvoří a zobrazí požadovaný text. Ten je zobrazen pouze na několik sekund a délka jeho viditelnosti je dána jednou z konstant `LENGTH_SHORT` a `LENGTH_LONG`. V klientské aplikaci je proto využíván pro oznamování úspěchu, neúspěchu nebo chybách uživatelských akcí.[8]

Pro co nejlepší a nejpohodlnější upozornění uživatele na hlídanou událost je využito *notifikací* systému Android. Jak můžete vidět na obrázku 5.5, upozornění se objeví přímo v oznamovací oblasti horní stahovací roletky. Pro upoutání pozornosti uživatele je výskyt notifikace doprovázen zavibrováním telefonu, zvukovým upozorněním a až do zrušení této notifikace je její existence indikována LED diodou (pokud jí zařízení disponuje). Tohoto chování je docíleno použitím konstant `DEFAULT_VIBRATE`, `DEFAULT_SOUND` a `DEFAULT_LIGHTS` při vytváření notifikace. V případě výskytu více upozornění nebudou přibývat další položky v oznamovací oblasti, ale původní notifikace bude aktualizována a v jejím pravém dolním rohu se bude zvyšovat počet aktuálních upozornění. Po kliknutí na notifikaci zmizí z oznamovací oblasti a bude uživateli zobrazen seznam všech upozornění (již v kontextu aplikace).



Obrázek 5.5: Notifikace o výskytu hlídané události na systému Android.

Pro kompatibilitu notifikací i se staršími verzemi systému Android byla využita knihovna zpětné podpory (angl. Support Library)[8], která nabízí využití aktuálních komponent na starších verzích systému. Z této knihovny je využita třída `NotificationCompat.Builder` sloužící pro vytvoření notifikace nezávislé na aktuální verzi systému Android.

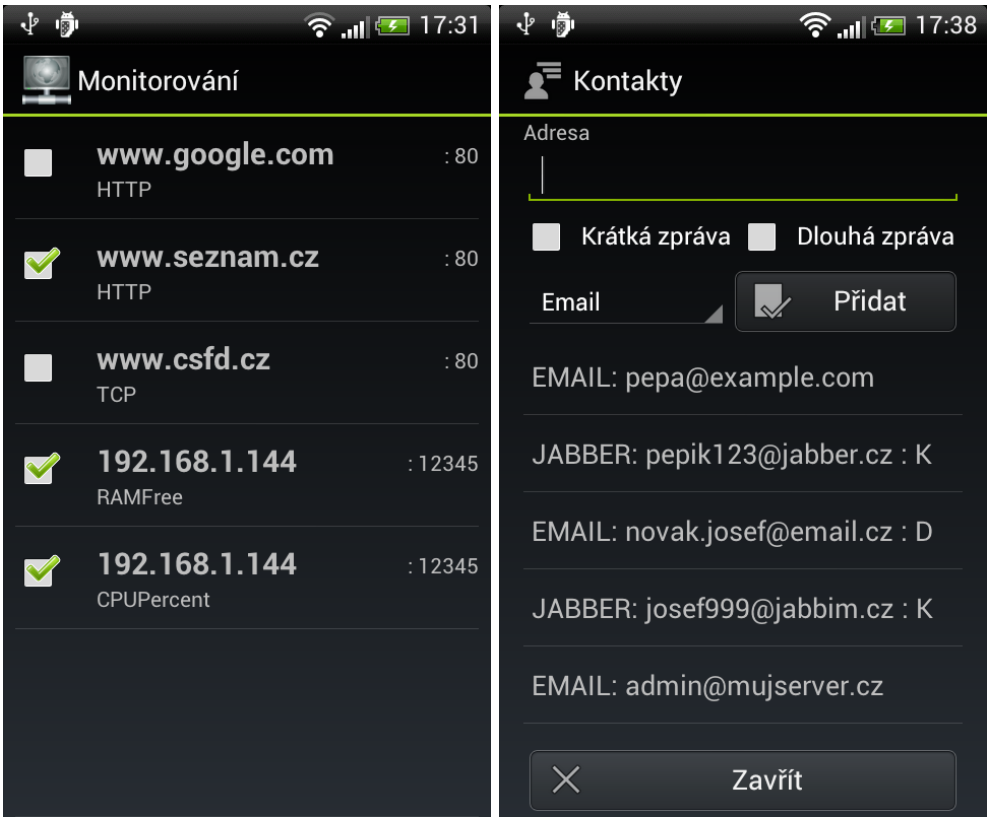
Kvůli bezpečnostní politice systému Android je nutné v souboru `AndroidManifest.xml` deklarovat, jaké součásti systému chce naše aplikace využívat. Je dobré zde uvádět opravdu jen to, co skutečně využíváme, protože mnoho uživatelů může velké množství požadovaných oprávnění odradit od instalace aplikace. Definice oprávnění vypadá v našem případě takto:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.VIBRATE" />
```

S ohledem na rozmístění prvků uživatelského rozhraní a především na komfort ovládání mobilní aplikace jsem byl na vážkách, zda na „podobrazovky“ umísťovat tlačítko Zavřít. V systému Android slouží k návratu na předchozí aktivitu hardwarová klávesa Zpět nebo,

[8] Více informací na http://developer.android.com/tools/extras/support-library.html.

v případě nových zařízení bez fyzických kláves, softwarové tlačítko pro návrat poskytované přímo systémem Android. Přesto můžeme u mnoha i profesionálních aplikací vidět vlastní tlačítka pro návrat. Po diskuzi s uživateli Androidu ve svém okolí jsem dospěl k názoru, že většina těchto lidí se nebrání používání návratových tlačítek implementovaných přímo konkrétní aplikací přesto, že všude mohou využívat hardwarovou klávesu nebo tlačítko systému Android. Několik uživatelů také zmínilo větší pohodlnost a lepší ergonomii při používání tlačítek aplikace. Proto jsem se rozhodl tato tlačítka do aplikace zahrnout a každá obrazovka (vyjma přihlašovací) disponuje alternativní možností, jak se navrátit zpět.



Obrázek 5.6: Obrazovka seznamu monitorování a kontaktů výsledné aplikace pro Android.

Na obrázku 5.6 vlevo můžete vidět obrazovku zobrazující seznam všech uživatelových monitoringů. Zaškrtávacím políčkem lze zapínat a vypínat konkrétní monitorování. Dále jsou pro každý monitoring zobrazeny adresa serveru, port a typ monitorování. Každý záznam disponuje kontextovým menu, přes které lze s monitoringem pracovat (například smazat, zobrazit graf, statistiky atd.). Ovládání celé aplikace se provádí přes kontextové menu vyvolané příslušným systémovým tlačítkem.

Na tomtéž obrázku vpravo můžete vidět obrazovku kontaktů. Ta ve své horní části umožňuje přidat nový kontakt, dole je pak zobrazen seznam všech vytvořených kontaktů. Řetězec reprezentující jednotlivý kontakt je složen z typu kontaktu, adresy a volitelně písmene `K` pro krátkou zprávu nebo písmene `D` pro zprávu dlouhou. Vyvoláním kontextového menu nad jednotlivými záznamy je možné kontakt smazat.

5.4 Agent na monitorovaném serveru

Pro sbírání údajů o systémových prostředcích na monitorovaném serveru bylo zapotřebí nástroje, který bude jednoduchý a zároveň platformě nezávislý. Řešit získávání těchto údajů individuálně na jednotlivých systémech by bylo velice neefektivní, ne-li dokonce nemožné, když si uvědomíme, jaké množství operačních systémů existuje. Především kvůli nezávislosti na platformě jsem se rozhodl i tuto nejmenší část systému implementovat v jazyce Java.

Pro získání systémových prostředků je využita knihovna *JavaSysMon*[9] dostupná pod licencí NetBSD, která nabízí metody pro získání aktuálního využití systémových prostředků nezávisle na operačním systému monitorovaného serveru (aktuálně podporuje systémy Mac OS X, Linux, Windows a Solaris). Mezi poskytovanými informacemi jsou údaje o obsazení operační paměti, vytížení procesoru nebo detailní přehledy běžících procesů. Pro získání informací o zaplnění pevných disků jsou využity metody `getFreeSpace()` a `getTotalSpace()` poskytované třídou `File`, která je součástí prostředí Java. Celý agent sestává pouze z jediné třídy `SystemInfoDaemon`.

Při implementaci tohoto agenta byla snaha o maximální jednoduchost a funkčnost návrhu. Agent je určen k dlouhodobému běhu na pozadí serveru, a proto by měl vyžadovat co nejméně systémových zdrojů. Pro jeho spuštění je nutné zadat jediný parametr – číslo portu, na kterém bude agent naslouchat. Ihned po spuštění a inicializaci začne naslouchat požadavkům na data. Pokud takový přijde, v novém vlákně se spustí jeho vyřízení, kde se na základně klíčového slova identifikuje požadovaná informace, která se ze systému získá a její hodnota se odešle zpět serveru. Mezitím se původní vlákno opět vrátí k naslouchání příchozím požadavkům.

Všechny údaje týkající se kapacity operační paměti nebo pevného disku jsou odesílány v megabajtech (MB). Protože naměřené údaje jsou do databáze ukládány jako celá čísla, jsou z důvodu vyšší přesnosti procentuální údaje odesílány v promile.

5.5 Komunikace

Komunikaci mezi jednotlivými komponentami systému zajišťuje proprietární textový protokol. Ve dvojici komunikujících stran vždy jedna vystupuje jako aktivní a druhá jako pasivní. To znamená, že pasivní strana pouze naslouchá a reaguje na požadavky strany aktivní – nikdy neiniciuje žádnou komunikaci. V případě komunikace klienta a serveru, kde roli aktivního člena zastává klient, však platí jediná výjimka. Tou je asynchronní zasílání upozornění ze strany serveru všem připojeným klientům. V komunikaci mezi serverem a agentem zastává aktivní roli server.

Protokol pro komunikaci serveru a klientů obsahuje 30 klíčových slov:

`HELLO, ACCEPT, DENIED, LOGIN, LOGOUT, REGISTER, STARTMON, STOPMON, MONLIST, MON, CONTACTLIST, CONFIGLIST, CONFIG, DATA, USER, STATS, TYPES, ADDMON, ADDCONTACT, ADDPLUGIN, UPDATEMON, UPDATECONTACT, UPDATEUSER, UPDATEPASSWD, UPDATECONFIG, DELETEMON, DELETECONTACT, DELETEUSER, WARN, SIMULATEWARN`

Pro komunikaci mezi serverem a agentem sbírajícím údaje o systémových prostředcích je potřeba těchto 9 klíčových slov:

`RAMFREE, RAMUSED, RAMPERC, DISKFREE, DISKUSED, DISKPERC, PROCPERC, PROCESSCOUNT, DENIED`

[9] Více informací na https://github.com/jezhumble/javasysmon.

Pokud aktivní strana požaduje nějaká data či provedení akce od strany druhé, odešle požadavek, který začíná jedním z klíčových slov, které identifikuje požadovanou akci. Podle typu požadavku může být obsahem klíčové slovo samotné nebo následované potřebnými parametry. Klíčové slovo a jednotlivé parametry jsou od sebe odděleny mezerou a celý požadavek je ukončen znakem konce řádku (\n). Podrobný popis navrženého protokolu včetně příkladů najdete v příloze B.

Na požadavky vyvolávající provedení nějaké akce je odpovídáno klíčovými slovy `ACCEPT` v případě úspěchu nebo `DENIED` v případě neúspěchu. Pokud jsou vyžadována data z databáze, jsou odeslána ve formátu, kde jednotlivé atributy jsou odděleny čárkou (`,`) a záznamy znakem středníku (`;`). Odpověď obsahující seznam kontaktů může pak vypadat například nějak takto:

```
CONTACTLIST 1,2,johny007@jabber.com,SHORT;2,1,john@example.com,LONG
```

Jak vyplývá z principu komunikace, aktivní strana vždy čeká na odpověď na svůj požadavek (ať kladnou nebo zápornou). Tento fakt zásadně komplikuje zpracování asynchronně zasílaných upozornění. Když klient odešle požadavek, začne naslouchat na vstupním datovém kanálu a očekává, že na něm najde svoji odpověď. Proto musí na tomto datovém vstupu figurovat funkce, která nejprve tzv. vyzvedne všechna došlá upozornění, a poté teprve vrátí odpověď, kterou klientská aplikace požadovala. Tato došlá upozornění poté může předat dále funkcím, které je zobrazí uživateli.

Takto by celá komunikace včetně vyzvedávání a prezentace došlých upozornění mohla fungovat, nicméně si můžeme všimnout faktu, že pokud nebude klientská aplikace vytvářet žádnou aktivitu, což je velice pravděpodobné v situacích, kdy bude pouze spuštěna na pozadí, nebudou vyzvedávána ani zobrazována žádná upozornění. Tento problém je vyřešen přidáním tzv. „hello timeru“. Po přihlášení uživatele k systému se spustí ve svém vlastním vlákně časovač, který v pravidelném intervalu odesílá serveru zprávu typu `HELLO`. Na tu je vždy bez okolků na straně serveru kladně odpovězeno. Tím se za prvé testuje funkčnost spojení mezi oběma stranami, ale především se při čekání na odpověď vyzvednou a zobrazí všechna upozornění. Hodnota intervalu odesílání zprávy typu `HELLO` je uložena v nastavení každého uživatele a její výchozí hodnota je nastavena na 10 s.

Kapitola 6

Testování

Testování probíhalo především na úrovni kompatibility výsledných aplikací na co nejširším spektru hostitelských systémů. Poté byla na vzorku příkladů ověřena validita dat naměřených monitorovacím systémem. K tomu byly použity hodnoty získané alternativními způsoby na systémech Windows i Linux.

6.1 Funkčnost aplikací

Kvůli silné fragmentaci systému Android popsané v kapitole 3.2 je vhodné, ne-li dokonce nezbytné, otestovat implementovanou mobilní aplikaci na co největším počtu konfigurací. Mobilní verze klientské aplikace byla otestována na těchto fyzických zařízeních:

- **HTC Desire S**, uhlopříčka 3,7", verze systému Android 4.0.4
- **Samsung Galaxy S**, uhlopříčka 4", verze systému Android 4.2.2
- **HTC Evo 3D**, uhlopříčka, 4,3", verze systému Android 4.0.3
- **Sony Ericsson Xperia mini**, uhlopříčka 3", verze systému Android 2.3.4
- **HTC Wildfire**, uhlopříčka 3,2", verze systému Android 2.2.1
- **Google Nexus 7**, uhlopříčka 7", verze systému Android 4.2.2
- **Point Of View Tablet ProTab 2**, uhlopříčka 9,7", verze systému Android 4.0.3

Dále byla aplikace testována na emulovaných zařízeních s verzemi systému 2.1 (Eclair), 2.3.3 (Gingerbread), 4.0.3 (ICS) a 4.2.2 (Jelly Bean). Především kompatibilita s nejstarší podporovanou verzí (Eclair) musela být zajištěna drobnými úpravami aplikace, které však neměly žádný dopad na výslednou funkčnost.

Ostatní části systému (serverová část, desktopový klient a agent) byly testovány na operačních systémech Microsoft Windows (verze XP, 7 a 8) a Linux (konkrétně Ubuntu 12.04, Debian 6.0.7 a Fedora 18). Celý systém pracoval správně napříč různými hostitelskými systémy.

Prakticky byly vyzkoušeny také způsoby upozornění. Zasílání varování na email nebo Jabber účet fungují bez jakýchkoliv komplikací. Odesílání SMS pomocí služby typu SMS mail bylo vyzkoušeno pro operátora O2, kde tato služba funguje standardně bez nutnosti aktivace či konfigurace. Protože se ukázalo, že zpráva dojde velice ořezaná (přesný počet

užitných znaků je závislý na délce emailové adresy odesilatele), bylo nutné zkrátit „krátký typ“ zprávy na absolutní minimum. Pokud tedy nebude adresa monitorovaného serveru příliš dlouhá, přijde uživateli SMS s kompletním a pochopitelným textem upozornění.

6.2 Ověření naměřených hodnot

Hodnoty získané monitorovacím systémem bylo třeba validovat a ověřit, že metodika jejich získávání není chybná. Proto byly pro několik typů monitorování získány hodnoty i jiným (alternativním) způsobem, aby poté mohly posloužit k porovnání jednotlivých výsledků. Srovnání můžete vidět v tabulce 6.1.

Monitoring	Naměřená hodnota			Metoda pro získání hodnoty
	Systém	Win	Linux	
ICMP seznam.cz	55	52	53	Nástroj `ping`
ICMP oracle.com	199	197	198	Nástroj `ping`
ICMP example.com	157	156	159	Nástroj `ping`
RAMFree win_pc	956	956	–	Program Sledování prostředků
RAMFree linux_pc	84	–	84	Soubor `/proc/meminfo`
DiskFree win_pc	51 031	51 031	–	Místní disk – vlastnosti
DiskFree linux_pc	5 780	–	5 319	Nástroj `df`
ProcessCount win_pc	76	81	–	Nástroj `tasklist`
ProcessCount linux_pc	146	–	149	Nástroj `ps`

Tabulka 6.1: Porovnání údajů získaných monitorovacím systém a alternativními způsoby.

Pro každý monitoring byla hodnota získána nejprve pomocí monitorovacího systému, poté pak alternativním způsobem popsaným v posledním sloupci tabulky. Pokud to bylo možné, byla hodnota naměřena v systémech Windows a Linux. U typu ICMP bylo naměřeno vždy 10 hodnot a jako výsledek byl použit jejich průměr.

Ze síťových typů monitorování bylo možné jednoduše validovat pouze typ ICMP, a to pomocí nástroje `ping`. Srovnání dopadlo velice dobře a rozdíly se pohybují pouze v řádech milisekund. Testování volné paměti RAM vykázalo naprosto shodné výsledky jak na systému Windows (byl využit program Sledování prostředků), tak v Linuxu (informace o paměti v souboru `/proc/meminfo`). Volná kapacita systémového disku byla naprosto přesně změřena pro Windows, u systému Linux však vznikla poměrně velká odchylka o 461 MB. Počty spuštěných procesů se na obou operačních systémech lišily řádově o jednotky, což lze brát jako uspokojivý výsledek.

Kapitola 7

Závěr

V rámci této práce byl vytvořen systém pro monitoring síťových serverů, služeb a systémových prostředků. Tento systém sestává ze serverové části zajišťující samotný monitoring, obsluhu klientů a práci s databází. Dále pak z klientských aplikací pro desktopové systémy a mobilní systém Android sloužící jako uživatelské rozhraní celého systému a agenta sbírajícího údaje o využitých systémových prostředcích na monitorovaných serverech. Pro komunikaci byl navržen proprietární textový protokol a všechny části systému byly implementovány.

Přesto, že existují spousty různých aplikací pro monitoring serverů, mnou navržené a implementované řešení je jedinečné kombinací monitorování serverovou částí, která může běžet na serveru s dostatečně dimenzovaným připojením k internetu a mobilního klienta, který nabízí skutečně nepřetržitý dohled nad svěřenými servery. Těmto vlastnostem sekunduje možnost monitorovat i systémové prostředky vybraných serverů. Pro zvýšení komfortu ovládání systému při práci na běžném počítači pak slouží klientská aplikace pro desktopové systémy. Navíc je systém rozšiřitelný zásuvnými moduly, které mohou přidat další typy monitorování a umožnit tak uživateli přizpůsobit si systém svým vlastním potřebám. Tyto vlastnosti dělají z vytvořeného systému unikátní nástroj nejen pro správce sítí.

Možnosti dalšího rozšíření vidím v důkladnějším zpracování statistik získávaných z naměřených dat. Ty by mohly být podstatně obsáhlejší a detailnější pro poskytnutí vyčerpávajícího přehledu o monitorovaném serveru. Dále by bylo vhodné implementovat pokročilejšího správce zásuvných modulů, který by přehledně zobrazil všechny používané moduly a umožňoval by je individuálně povolovat, zakazovat či přímo odstraňovat ze systému. V této práci byla také odsunuta do pozadí zájmu grafická podoba klientských aplikací. Pro budoucí rozvoj by tedy bylo vhodné navrhnout nový vzhled uživatelského rozhraní, který by sjednotil design jednotlivých klientů a dal systému atraktivní a charismatickou grafickou podobu.

Literatura

[1] Android: Dashboards. [online], 2013 [cit. 2013-04-25].
URL http://developer.android.com/about/dashboards/index.html

[2] Šeda, J.: Úvod do JDBC. [online], 2003 [cit. 2013-04-21].
URL http://interval.cz/clanky/uvod-do-jdbc/

[3] Fielding, R.; Gettys, J.; Mogul, J.; aj.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard)[online], Červen 1999, updated by RFCs 2817, 5785, 6266, 6585.
URL http://www.ietf.org/rfc/rfc2616.txt

[4] Halsall, F.: *Computer Networking and the Internet*. Addison Wesley, páté vydání, 2005, ISBN 0-321-26358-8, 832 s.

[5] Josefsson, S.: The Base16, Base32, and Base64 Data Encodings. RFC 4648 (Proposed Standard)[online], Říjen 2006.
URL http://www.ietf.org/rfc/rfc4648.txt

[6] Kretchmar, J. M.; Dostálek, L.: *Administrace a diagnostika sítí: pomocí OpenSource utilit a nástrojů*. Brno: Computer Press, 2004, ISBN 80-251-0345-5, 216 s.

[7] Kurose, J. F.; Ross, K. W.: *Computer Networking: A Top-Down Approach*. Pearson, šesté vydání, 2012, ISBN 978-0132856201, 864 s.

[8] Meier, R.: *Professional Android 2 Application Development*. Indianapolis: Wiley Publishing, 2010, ISBN 978-0-470-56552-0, 543 s.

[9] Murphy, M. L.: *Android 2: Průvodce programováním mobilních aplikací*. Brno: Computer Press, 2011, ISBN 978-80-251-3194-7, 375 s.

[10] Postel, J.: Internet Control Message Protocol. RFC 792 (Standard)[online], Září 1981, updated by RFCs 950, 4884, 6633.
URL http://www.ietf.org/rfc/rfc792.txt

[11] Rivest, R.: The MD5 Message-Digest Algorithm. RFC 1321 (Informational)[online], Duben 1992, updated by RFC 6151.
URL http://www.ietf.org/rfc/rfc1321.txt

Příloha A

Obsah CD

Na přiloženém CD najdete všechny čtyři části monitorovacího systému – serverovou část (Server), klienta pro desktopové systémy (DesktopClient), klienta pro mobilní systém Android (AndroidClient) a agenta monitorujícího systémové prostředky (SystemInfoDaemon). Všechny části jsou zde umístěny jak ve spustitelné verzi (`*.jar` a složka s potřebnými knihovnami nebo `*.apk`), tak ve formě zdrojových kódů. Dále je zde tato technická zpráva ve formátu `pdf` a taktéž její zdrojové kódy pro sázecí program LaTeX. Součástí je také programová dokumentace popisující spuštění jednotlivých částí systému. Struktura přiloženého CD je podrobněji rozepsána v následujícím výčtu.

- **AndroidClient** – Soubor pro instalaci klienta do systému Android
- **AndroidClient-source** – Zdrojové kódy klienta pro systém Android
- **DesktopClient** – Spustitelný soubor klienta pro desktopové systémy
- **DesktopClient-source** – Zdrojové kódy klienta pro desktopové systémy
- **Server** – Spustitelný soubor serverové části
- **Server-source** – Zdrojové kódy serverové části
- **SystemInfoDaemon** – Spustitelný soubor agenta sbírajícího systémové prostředky
- **SystemInfoDaemon-source** – Zdrojové kódy agenta sbírajícího sys. prostředky
- **zprava-source** – Zdrojové kódy technické zprávy pro program LaTeX
- **initDatabase.sql** – SQL skript pro inicializaci databáze
- **readme.txt** – Popis požadavků a spuštění jednotlivých částí systému
- **zprava.pdf** – Technická zpráva ve formátu `pdf`

Příloha B

Komunikační protokol

Schéma příkazu	Popis
`RAMFREE`	Hodnota volné paměti RAM [MB]
`RAMUSED`	Hodnota zaplněné paměti RAM [MB]
`RAMPERC`	Využití paměti RAM v promile
`DISKFREE`	Hodnota volného místa na systémovém disku [MB]
`DISKUSED`	Hodnota zaplněného místa na systémovém disku [MB]
`DISKPERC`	Zaplnění systémového disku v promile
`PROCPERC`	Vytížení procesoru v promile
`PROCESSCOUNT`	Počet spuštěných procesů
`DENIED`	Záporná odpověď v případě nezdaru

Tabulka B.1: Tabulka popisující navržený komunikační protokol mezi serverovou částí a agentem sbírajícím údaje o systémových prostředcích.

Pozn.: Kromě záporného `DENIED` jsou všechna klíčová slova zároveň požadavek i odpověď (v tom případě za nimi následuje požadovaná hodnota).

Schéma příkazu	Příklad	Popis	Směr
`ACCEPT`		Kladná odpověď serveru	Odpověď
`DENIED`		Záporná odpověď serveru	Odpověď
`LOGIN login passwd`	`LOGIN admin hash`	Přihlášení uživatele	Požadavek
`LOGOUT`		Odhlášení uživatele	Požadavek
`REGISTER login passwd firstname lastname email`	`REGISTER pepa hash Pepa Novák pepa@example.com`	Registrace nového uživatele	Požadavek
`STARTMON` id	`STARTMON 2`	Spustí požadovaný monitoring	Požadavek
`STOPMON id`	`STOPMON 2`	Zastaví požadovaný monitoring	Požadavek
`MONLIST`		Dotaz na seznam monitoringů	Požadavek
`MONLIST id,address, period,...;id,address,...`	`MONLIST 2,example.com,5,...; 3,mujserver.cz,15,...`	Vrací seznam všech uživatelových monitoringů	Odpověď
`MON id`	`MON 2`	Dotaz na informace o monitoringu	Požadavek
`MON`	`MON 2,example.com,5,...`	Vrací informace o monitoringu	Odpověď
`CONTACTLIST`		Dotaz na seznam kontaktů	Požadavek
`CONTACTLIST id,type, address,flags;id,type,...`	`CONTACTLIST 1,2, pepa@jabber.com,SHORT;2,3,...`	Vrací seznam všech uživatelových kontaktů	Odpověď
`CONFIGLIST`		Dotaz na seznam nastavení	Požadavek
`CONFIGLIST name,value,note; name,value,note;...`	`CONFIGLIST timeoutTCP,3000,TCPTimeout;...`	Vrací seznam globálních nastavení přihlášeného uživatele	Odpověď
`DATA monitoring`	`DATA 3`	Dotaz na data monitoringu	Požadavek
`DATA value,datetime;...`	`DATA 324,2013-04-14 18:59:16;...`	Vrací všechna naměřená data konkrétního monitoringu	Odpověď
`USER`		Dotaz na údaje o uživateli	Požadavek
`USER login,firstname, lastname,email`	`USER pepa,Pepa, Novák,pepa@example.com`	Vrací údaje o přihlášeném uživateli	Odpověď

Tabulka B.2: Tabulka popisující navržený komunikační protokol mezi klientem a serverovou částí (první část).

Schéma příkazu	Příklad	Popis	Směr
`STATS monitoring`	`STATS 3`	Dotaz na statistiky monitoringu	Požadavek
`STATS min,max,avg,perc,last`	`STATS 155,397,226,3, 2013-04-17 01:08:41`	Vrací statistiky konkrétního monitoringu	Odpověď
`TYPES`		Dotaz na typy monitorování	Požadavek
`TYPES type,type,...`	`TYPES HTTP,TCP,UDP,...`	Vrací poskytované typy	Odpověď
`ADDMON address port period type crit warn contact`	`ADDMON example.com 80 15 HTTP 500 3 2`	Přidá nový monitoring	Požadavek
`ADDCONTACT type address flags`	`ADDCONTACT 1 pepa@xyz.cz LONG`	Přidá nový kontakt	Požadavek
`ADDPLUGIN name data`	`ADDPLUGIN ICMP Base64Data`	Nahraje na server nový zás. modul	Požadavek
`UPDATEMON id address port period type crit warn contact`	`UPDATEMON 2 example.com 80 10 TCP 350 1 3`	Upraví hodnoty monitoringu	Požadavek
`UPDATECONTACT id type address flags`	`UDPATECONTACT 3 1 pepik@example.com SHORT`	Upraví hodnoty kontaktu	Požadavek
`UPDATEUSER login firstname lastname email`	`UPDATEUSER pepa Pepik Novák pepik@example.com`	Upraví údaje o uživateli	Požadavek
`UPDATEPASSWD old new`	`UPDATEPASSWD oldHash newHash`	Změní uživatelské heslo	Požadavek
`UPDATECONFIG name value`	`UPDATECONFIG timeoutHTTP 4000`	Změní hodnotu nastavení	Požadavek
`DELETEMON id`	`DELETEMON 2`	Smaže požadovaný monitoring	Požadavek
`DELETECONTACT id`	`DELETECONTACT 3`	Smaže požadovaný kontakt	Požadavek
`DELETEUSER login`	`DELETEUSER pepa`	Smaže požadovaného uživatele	Požadavek
`WARN text`	`WARN Upozorneni monitoringu...`	Asynchronní upozornění klienta na výskyt hlídané události	Odpověď
`SIMULATEWARN id`	`SIMULATEWARN 2`	Vyvolá simulované upozornění	Požadavek
`HELLO`		Periodické testování spojení	Požadavek

Tabulka B.3: Tabulka popisující navržený komunikační protokol mezi klientem a serverovou částí (druhá část).