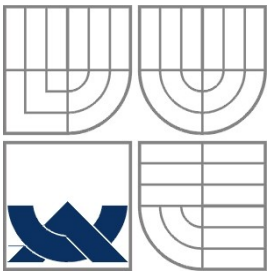


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

KOMPONENTA PRO MONITORING SERVERU

SERVER MONITORING COMPONENT

BAKALÁŘSKÁ PRÁCE
BACHELOR THESIS

AUTOR PRÁCE — JAROMÍR BOČEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN ROUPEC, Ph.D.
SUPERVISOR

BRNO 2011

ABSTRAKT

Bakalářská práce se zabývá zpracováním programu univerzální komponenty pro sběr dat ze serverů. Komponenta splňuje požadavky pro snadnou instalaci, nastavení a rozběh z daného serveru. Z monitorovaného stroje odcházejí informace o tom, že monitorovaný server je zapnutý a pro daný úkol funkční a zejména pak informace o vytížení CPU, stavu zaplnění disků a stavu všech přihlášených uživatelů. Navržená koncepce je multiplatformní s důrazem na nejrozšířenější platformy, jako jsou Linux a Windows. V rámci této práce je také vyřešena otázka průchodnosti získaných dat Internetem přes všechny překážky v podobě různě nastavených firewallů a jiných zabezpečovacích prvků.

ABSTRACT

This Bachelor thesis deals with the development of a universal server monitoring component for collecting data from various servers. This component meets the requirements for easy installation, setup and activation of the specified server. The monitored unit releases information that the monitored server is active and completely functioning, in particular, information on CPU utilization, state of disk utilization, and state of all logged users. The presented multiplatform concept puts emphasis on the most common platforms, such as Linux and Windows. In addition, transfer of acquired data through the Internet obstacles in the form of firewall configurations and other security technologies has been solved within this work.

KLÍČOVÁ SLOVA

Komponenta, program, server, vlákno, PERL, RRDtool, Round Robin, HTTP, POST, monitoring

KEYWORDS

Component, program, server, thread, PERL, RRDtool, Round Robin, HTTP, POST, monitoring

PODĚKOVÁNÍ

Rád bych poděkoval panu Ing. Janu Roupcovi, Ph.D., který mě jako vedoucí mojí bakalářské práce poskytnul velmi cenné informace ještě před započetím prací, v samotné fázi prvotních úvah o daném tématu a dále též děkuji firmě HEXAGEEK s.r.o., která mi umožnila při tvorbě vlastního kódu odzkoušení v reálném provozu na jejich serverech a poskytla mi cenné informace z praxe v oblasti provozování server hostingů a s tím spojených potřeb monitorování serverů.

Obsah:

1 ÚVOD

Společnost HEXAGEEK s.r.o. vznikla za účelem rozšířit oblast podnikání na území České Republiky jako součást holdingu společně s firmami HEXAGEEK LLC (USA), HEXAGEEK LTD (Canada) a HEXAGEEK LTD (UK). V současné chvíli má společnost alokovány vlastní prostory v několika datových centrech po celém světě (UK, CANADA – Montreal, USA – Dallas, Washington DC, Californie, NewYork, FRANCIE) a v ČR pak v datacentrech GTS, Casablanca, Sitel, TTC a Master Internet. Úspěšně tak pokrývá poptávku po službách v oblasti IT technologií, zejména pak hosting serverů a služeb, SaaS, IaaS a Cloud computing po celém světě. Zahraniční konektivita společnosti je řešena v rámci partnerství s PEER1, TATA COMMUNICATIONS (TELEGLOBE), Google, Cogent, Level3, Deutche telekom, Dial telecom. Společnost je členem RIPE, ARIN, INTERNIC, APNIC.

Právě se společností HEXAGEEK s.r.o. jsem se dohodl na realizace programové komponenty, která by sloužila pro monitoring serverů, a která by byla určena v první fázi pro jejich vlastní potřebu a v druhé fázi pak pro potřeby zákazníků obecně.

Komponenta pro monitoring serverů. Bakalářský projekt byl záměrně nazván tímto názvem, neboť úmyslem bylo již zpočátku naznačit, že byť se jedná o samostatně funkční část, jde také o součást plánovaného vyššího systému. Komponenta je část, pomocí které s ostatními dalšími částmi vzniká něco většího. [1]

Tato komponenta je součástí globálního systému pro monitoring serverů a zprostředkování informací o serverech směrem k zákazníkovi. Pod pojmem komponenta je také možné si představit z prostředí Windows známou „službu“ či z prostředí Linux „démona“, anebo ji prostě chápat jako monitorovacího agenta či funkční software nebo program.

2 PŘEDPOKLADY ÚSPĚŠNOSTI PROJEKTU

2.1 Základní otázka realizace

Na počátku každého započatého projektu si nejdříve kladu otázku, zda má vůbec smysl daný projekt realizovat a zda již v danou chvíli neexistuje řešení dané problematiky, či zda bude mít výsledné řešení vůbec uplatnění v praxi. V případě této bakalářské práce jsem si tuto otázku položil také. Otázka, zda má daný projekt smysl má ale několik aspektů a úrovní, ze kterých je možné na ni odpovědět.

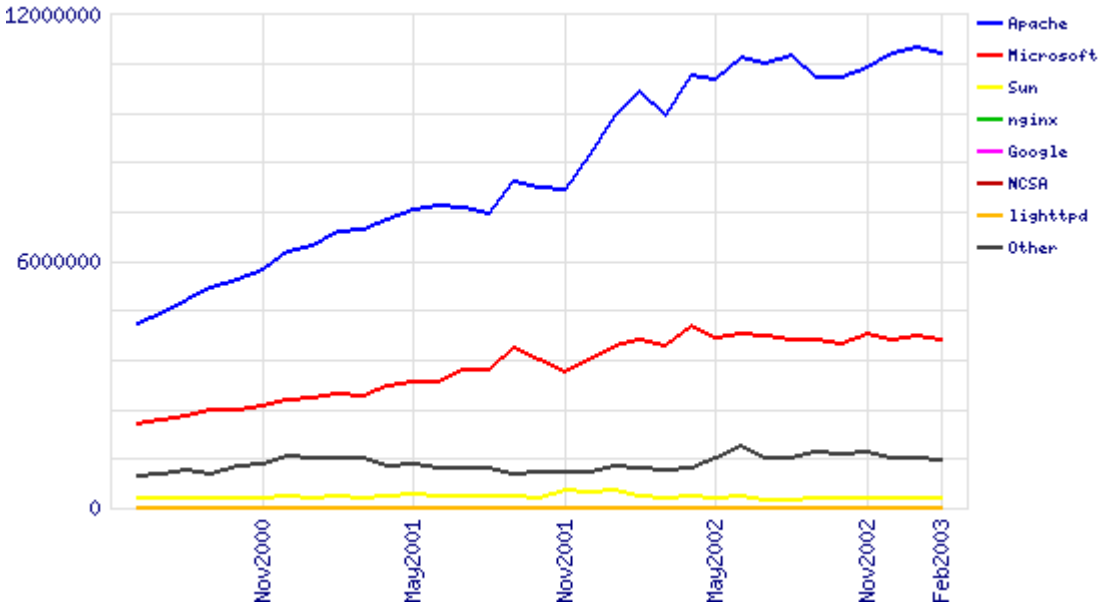
Přestože nebyly žádné větší pochybnosti o uplatnění projektu v praxi, poněvadž komponenta pro monitoring serverů, jakožto téma této bakalářské práce, je realizována víceméně na požádání hostingové společnosti, provedl jsem několik kroků pro ověření uplatnění v praxi.

Prvním aspektem je vlastní uplatnění komponenty, na základě něhož jsem provedl průzkum internetu vzhledem k nárůstu počtu serverů a tedy i potenciální možnosti nasazení komponenty do reálného provozu. Dalším aspektem je alespoň zběžná rekapitulace stávajících řešení dané problematiky a třetím aspektem je poté definice základních předpokladů, které musejí být v rámci této bakalářské práce vyřešeny tak, aby komponenta byla skutečně provozuschopná, v praxi uplatnitelná a v rámci konkurenčního prostředí i schopná. I přitom, že vyvstává velké množství dalších aspektů, potažmo nových otázek, dá se říci, že výše uvedené okruhy jsou pro započetí práce dostačující.

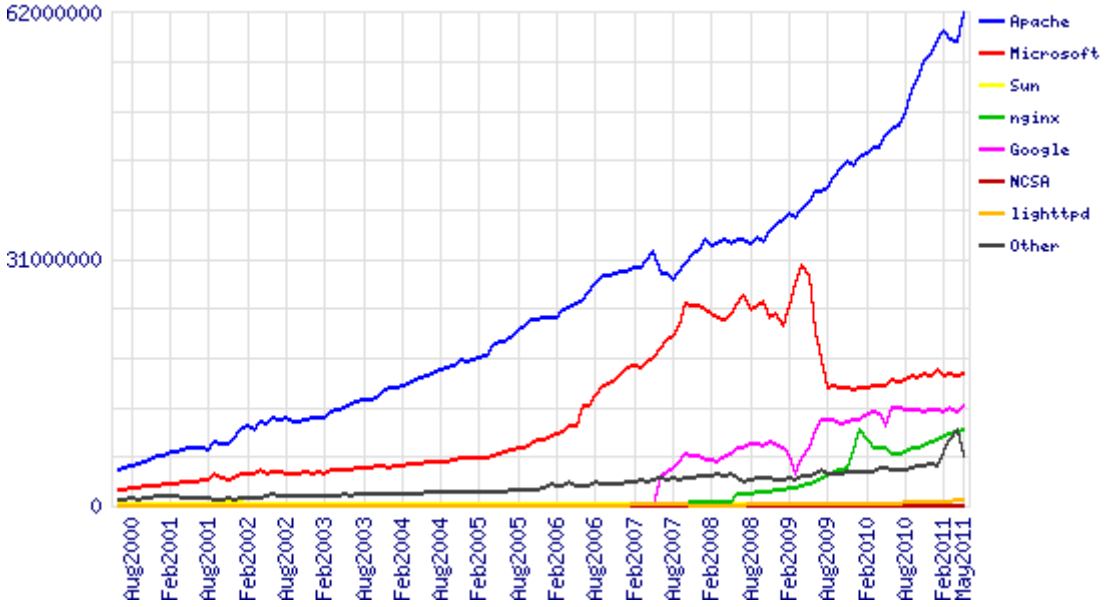
2.2 Aspekt 1 – uplatnění komponenty => Průzkum internetu – statistika rozvoje

Existuje nepřeberné množství způsobů a technik určených pro realizaci analýz Internetu a k následnému poskytování výsledků z těchto výzkumů a analýz v mnoha aspektech. Jednou ze společností, zabývajících se těmito výzkumy a analýzami je i společnost NETCRAFT, která provádí průzkum Internetu již od roku 1995 a je v tomto oboru uznávanou autoritou. [2]

V současné době nejaktuálnější statistiky z průzkumu trhu této společnosti jsou z května 2011 (pro bližší prostudování jsou k dispozici na stránkách společnosti NETCRAFT statistiky za každý měsíc v daném roce). Z nejstarší dostupné statistiky, tedy z února 2003 (Obr. 1) je vidět nárůst webových serverů zejména z rodiny Apache a z nejmladší dostupné statistiky, tedy právě z května 2011 (Obr. 2), pak je patrný tento pokračující trend za posledních 10 let.



Obr. 1 Součty aktivních serverů ve všech doménách v období červen 2000 – únor 2003. [3]



Obr. 2 Součty aktivních serverů ve všech doménách v období červen 2000 – květen 2011. [4]

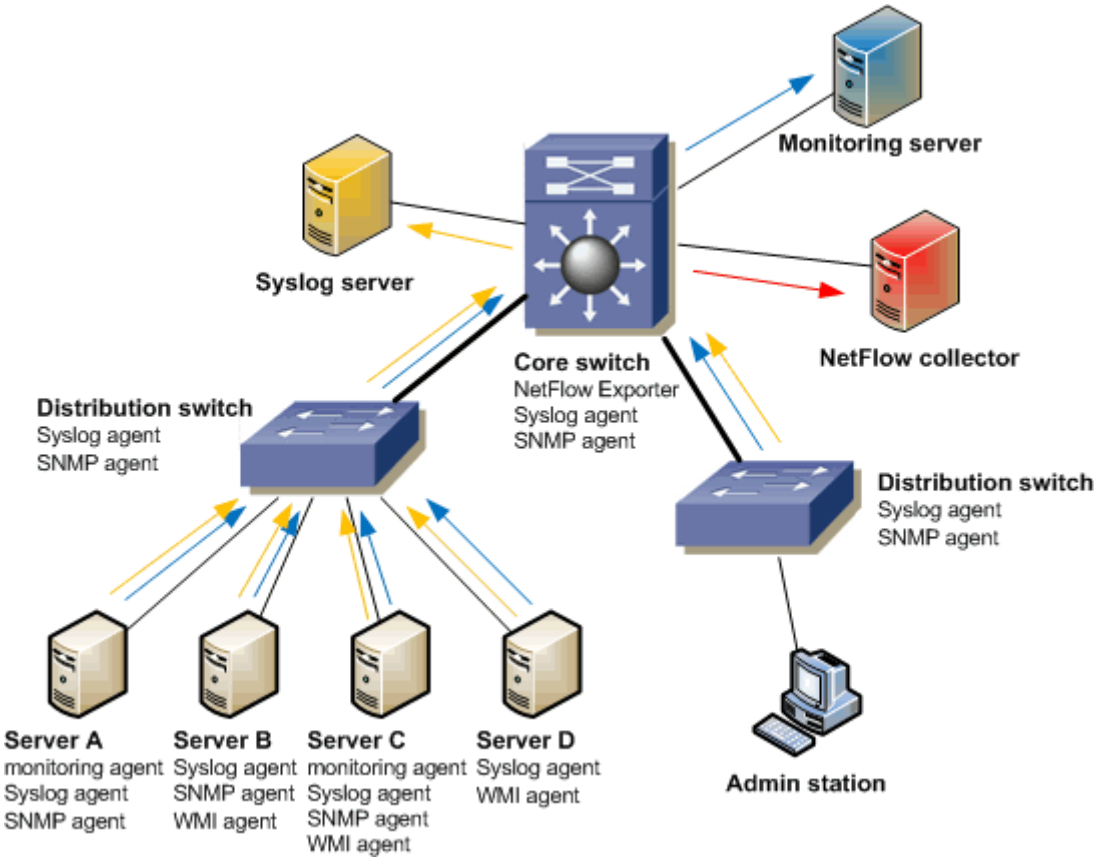
V následující tabulce je provedeno shrnutí statistických dat z období května 2011.

Developer	Duben 2011	Procenta	Květen 2011	Procenta	Index změny
Apache	58,382,787	55.12%	61,961,645	57.52%	2.40
Microsoft	16,333,388	15.42%	16,598,468	15.41%	-0.01
Google	11,892,939	11.23%	12,651,775	11.75%	0.52
nginx	9,192,405	8.68%	9,499,307	8.82%	0.14
lighttpd	660,001	0.62%	678,076	0.63%	0.01

Tab. 1 Součty aktivních serverů ve všech doménách v měsících duben a květen 2011. [4]

Výše uvedené statistiky se týkají pouze web serverů. Samozřejmě, že do mezinárodní sítě Internet jsou připojeny i další typy serverů služeb, včetně mnoha softwarových řešení. Pro posouzení, zda je plánovaná komponenta uplatnitelná v praxi se mi ovšem průzkum web serverů jeví jako dostačující, neboť nárůst v období posledních deseti let je tak markantní, že i při nasazení pouze na tyto typy serverů bude výtěžnost řešení zcela uspokojivá. Samotná zvyšující se penetrace nových řešení nginx nebo lighttpd pak poukazuje i na ten fakt, že monitorovací komponenta by měla být co nejvíce univerzální co do jednotlivých možností monitoringu.

2.3 Aspekt 2 – stávající řešení => Rekapitulace



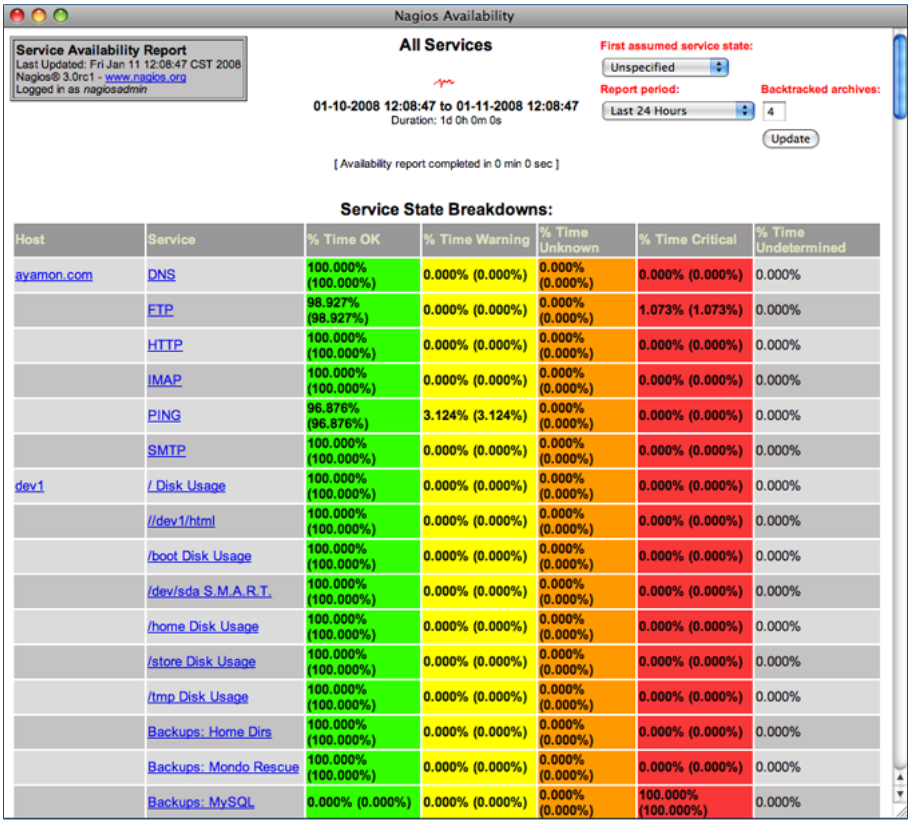
Obr. 3 Stručné schéma zapojení monitorovacích technologií do počítačové sítě. [5]

2.3.1 Nagios

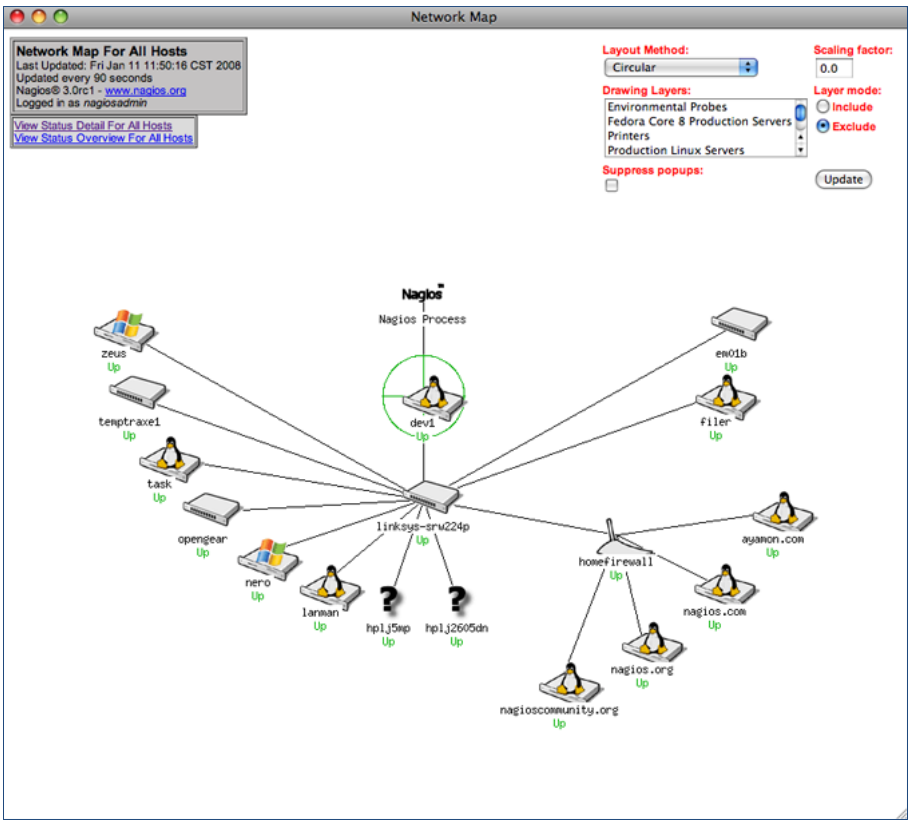
Nagios je velmi populární a rozšířený systém určený pro monitorování stavu počítačových sítí a jimi poskytovaných služeb. Systém je primárně vyvíjen pro Linux, ale je možné jej provozovat i na jiných unixových systémech. Je vydáván pod GPL licencí a je plně šířen jako *open source*. Vývojem a údržbou se zabývá Ethan Galstadt společně s mnoha dalšími *open source* vývojáři pluginů. [6]

Nagios je primárně určen jako monitorovací řešení pro koncové stanice, servery a jednotlivé prvky sítě.

Původně se projekt nazýval Netsaint a v roce 2002 byl přejmenován na Nagios. Nagios je nástroj, který umožňuje monitorovat počítačovou síť a v ní poskytované služby a v případě výskytu problému okamžitě informovat administrátora, který tak může rychle zasáhnout. Monitorovací služba periodicky spouští kontroly specifikovaných koncových uzlů a služeb. Používá k tomu externí moduly, které oznamují výsledek kontroly hlavnímu modulu Nagiosu. Pokud se vyskytne problém, služba pošle upozornění na předdefinované kontakty pomocí různých typů komunikace (email, SMS, nebo online zprávy – např. ICQ). Aktuální stav, historii záznamů a další výstupy jsou přístupné přes webové rozhraní. [6]



Obr. 4 Ukázka reportu dostupnosti služeb v Nagiosu. [7]

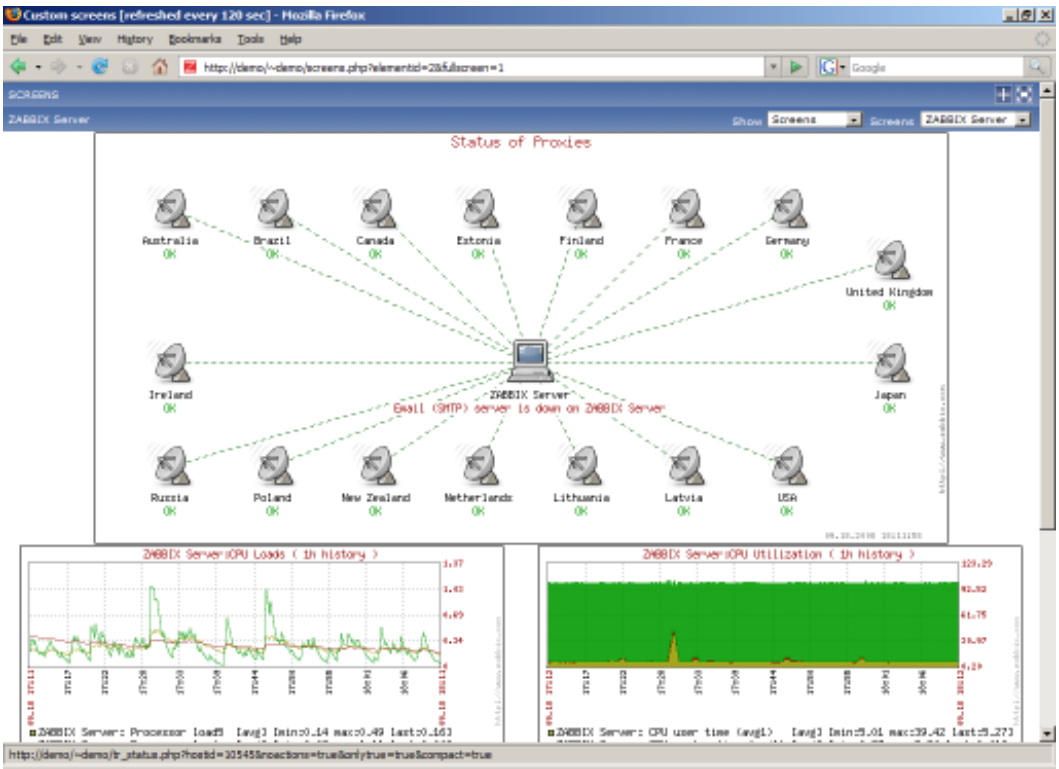


Obr. 5 Nagios – ukázka analýzy struktury sítě. [7]

Oproti mnoha výhodám, které systém Nagios nabízí, disponuje i, pro moje záměry, značnou nevýhodou a tou je zejména jeho časově náročná manuální konfigurace, která nepatří mezi jednoduché.

2.3.2 Zabbix

Mezi nejznámější *open source* alternativy vůči Nagiosu, pod licencí GPL, patří řešení s názvem Zabbix. Jeho konfigurace je mírně složitější a celkové řešení je náročnější na provoz a ukládání získaných dat. Mezi jeho hlavní přednosti patří možnost ukládání dat do MySQL databáze. Takto uložená data pak mohou posloužit jako vstupní prvek pro realizaci různých nástaveb, statistických nástrojů a analýzu.



Obr. 6 Ukázka ze software Zabbix. [8]

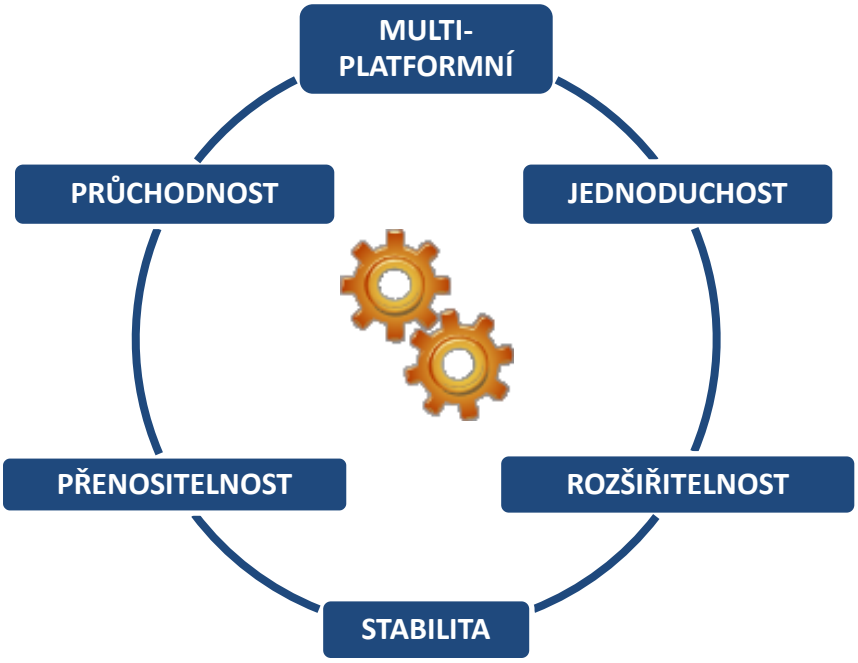
2.3.1 Další software pro monitoring

Mezi další rozšířený monitorovací software patří např. *real-time* monitorovací nástroj od společnosti Zenoss, Inc. [9], nebo propracované monitorovací nástroje společnosti Spluk Inc. [10]

Velmi zajímavým softwarem je i softwarový nástroj Cacti, který je založen na funkcionalitě RRDtool. [11]

Velmi rozšířené jsou pak také monitorovací nástroje velkých společností, jako je např.: Microsoft Systém Center Operations Manager, HP OpenView, CiscoWorks LAN Management Solutions nebo IBM Tivoli Netcool. [5]

2.4 Aspekt 3 – základní předpoklady vlastního řešení => Vytyčení cílů



Obr. 7 Vytyčení cílů projektu.

- Cíl 1. — Multiplatformní řešení:

  Jak je patrné z předchozí kapitoly, statistiky naznačují, že v současnosti jsou velmi rozšířené zejména platformy Linux a Windows. Komponenta by tedy měla být schopná fungování zejména na těchto platformách. Nicméně ani ukazatele statistik by neměly být pro komponentu limitující a proto vzniká požadavek neomezovat se pouze na platformy Linux a Windows, ale najít způsob, jakým by mohla komponenta být funkční i na mnoha dalších platformách.

- Cíl 2. — Jednoduchost řešení:

  Tímto cílem je stanoven požadavek na jednoduchost instalace komponenty, jednoduchost její konfigurace a v neposlední řadě také na jednoduchost spuštění a samotné funkčnosti komponenty. Tento cíl byl stanoven s ohledem na požadavek rychlého a snadného zprovoznění komponenty vzhledem k potřebám v několika dnech rozběhnout i řádově stovky serverů a neztrácet při této činnosti příliš mnoho času zprovozňováním propracovaných typů monitorovacích nástrojů na úkor zprovozňování vlastních serverů.

- Cil 3. — Rozšiřitelné řešení:

  Každé řešení je v daném okamžiku omezené danou potřebou řešeného projektu a obzorem jeho řešitele. Právě z tohoto důvodu je jako další cíl projektu nastavena rozšiřitelnost řešení, neboť co se zdá v této chvíli jako vyhovující a dostačující, může být o několik měsíců později pochopeno jako již nedostačující a v horším případě i nevyhovující. Pokud bude tedy komponenta již od počátku chápana s možností její rozšiřitelnosti, lze se alespoň částečně vyhnout neúspěchu tohoto typu.

- Cíl 4. — Stabilní řešení:

  Stabilita řešení je zcela oprávněným faktorem pro posouzení, poněvadž u systému, který má sám o sobě vykonávat supervizi nad jiným systémem nemůže být nestabilní řešení akceptovatelné. Snaha o co nejvyšší možnou stabilitu je tedy jakýmsi automatickým požadavkem na software takového typu.

- Cíl 5. — Přenositelné řešení:

  Cílem pro získání přenositelného řešení je zejména snaha o odbourání potřeby provádět kompilaci zdrojového kódu. Tento cíl tedy vymezuje oblast použití programovacího jazyka na interpretovaný programovací jazyk, jako je např. PERL, PYTHON, SHELL atd.

- Cíl 6. — Vyřešení průchodnosti dat Internetem:

  Posledním cílem, který považuji za neméně důležitý, je prostupnost získaných dat z monitorovaného stroje až k cílovému bodu, tedy na vlastní monitorovací server. V mezinárodní síti Internet na takto získaná data číhá nepřeberné množství překážek v podobě různě nastavených zabezpečovacích prvků a firewallů, včetně důmyslné obezřetnosti různě technicky zdatných administrátorů a správců jednotlivých sítí.

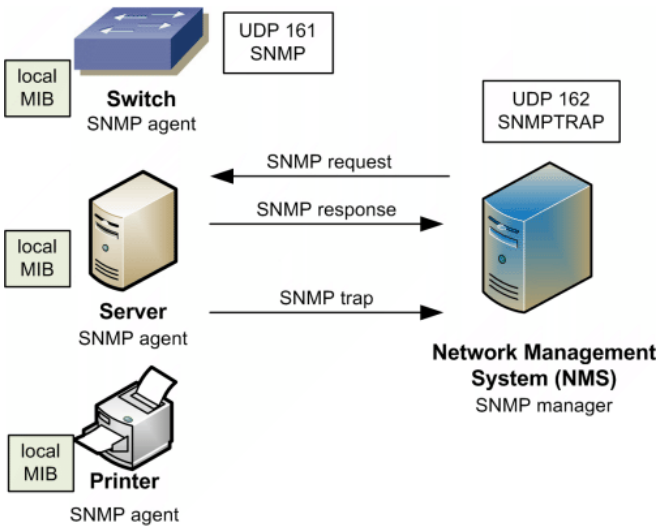
3 PŘEDBĚŽNÁ ANALÝZA ŘEŠENÍ

3.1 Známé nevýhody řešení monitoringu

Při návrhu komponenty jsem musel zvážit, jakými nedostatky trpí jiné aplikace a metody, které se zabývají monitoringem. Musel jsem postupně projít tyto nedostatky a zvážit, které se mi vlastním návrhem podaří překonat a které nikoliv.

Existuje mnoho aspektů, dle kterých je možné monitorovací aplikace a systémy řešení rozdělit. Pro lepší přehlednost jsem zvolil poměrně obecné rozdělení z pohledu použití monitorovacího agenta na systémy s použitím monitorovacího agenta a systémy bez použití monitorovacího agenta.

Monitorovacího agenta je možné si představit jako samostatně funkční část systému monitoringu (klienta), umístěnou přímo na monitorovaném serveru a zprostředkující sběr informací o daném serveru. Získaná data jsou následně přenášena na vzdálený centrální server, který je zprostředkovává směrem k administrátorům, potažmo zákazníkům. Systémy bez použití agenta, jak již vyplývá z názvu, nemají na monitorovaném serveru žádnou svou funkční část a povětšinou se omezují na vzdálené testování vybraných služeb serveru, popř. využívají k monitoringu předem určených protokolů, např. SNMP (*Simple Network Management Protocol*), WMI(*Windows Management Instrumentation*), IPMI(*Intelligent Platform Management Interface*), apod.



Obr. 8 Princip komunikace v SNMP. [12]

V praxi se ovšem tyto typy monitorovacích systémů velmi často kombinují, resp. spojují a poté se stávají vždy systémy s použitím agenta s tím, že využívají i prostředků, kterých využívají systémy bez použití agenta.

Mezi známé nevýhody řešení monitoringu s použitím agenta patří:

- Nutnost existence agenta pro daný operační systém, resp. platformu.
- Musí být zajištěna možnost instalace agenta na daný monitorovaný server.
- Instalací agenta na server přidáváme další aplikaci s potenciální možností způsobení chyby.

Mezi známé nevýhody řešení monitoringu bez použití agenta patří:

- Pouze vzdálené testování vybraných služeb.
- Nutnost zajistit průchodnost standardních protokolů.
- Celkové snížení bezpečnosti používání nestavových protokolů (UDP) – snadnější napadení.
- Zvýšení možného napadení hackerem z důvodů většího počtu otevřených portů a většího počtu služeb.

3.2 Syntéza koncepce vlastního řešení

Při návrhu vlastního řešení jsem se zaměřil zejména na hlavních šest cílů vytyčených v kapitole 2.4 Aspekt 3 – základní předpoklady vlastního řešení => Vytyčení cílů. V této kapitole je popsána syntéza mého postupu při hledání odpovědí na jednotlivé vytyčené cíle k řešení dané problematiky.

Primární řešení mé bakalářské práce je zaměřeno na různé distribuce Linux, neboť nejvíce serverů, které je potřeba obsluhovat v rámci hostingu společnosti, která dala podnět k této práci, jsou webové servery Apache. Ovšem serverů z prostředí Windows není také zanedbatelné množství, nemluvě o dalších platformách. Po zvážení jednotlivých programovacích jazyků jsem zvolil interpretovaný programovací jazyk PERL. V momentě, kdy byl programovací jazyk vybrán, bylo nutné si zpětně odpovědět, zda byla tato volba správná.

PERL je „interpretovaný“ programovací jazyk. Mezi hlavní výhody „interpretovaného“ jazyka patří především rychlý vývoj bez nutnosti kompilace a linkování - program je zkompilován po každém spuštění a je kdykoli možné "přikompilovat" další kód. Je možno i provést část kódu před kompilací zbytku a nastavit tak konstanty, které následně může využít optimalizační část kompilace. Je také například možné vynechat kód pro ladicí výpisy, pokud není program spuštěn s určitým parametrem. [13] Mezi jeho další výhody patří efektivita programování, automatická práce s pamětí, kdy není potřeba explicitně uvolňovat a alokovat paměť, reference na statické, dynamické a anonymní datové struktury a stabilita mnoho let vyvíjeného programu. PERL podporuje znakovou sadu UNICODE a umožňuje dynamické volání procedur. Existuje mnoho dalších výhodných vlastností tohoto jazyka, ale zmíním už jen poslední a tou je výhoda „svobodného software“ licencovaného pod GNU *General Public License*.

Volbou tohoto programového rozhraní jsem si odpověděl hned na několik vytyčených cílů. V první řadě jazyk PERL je možné provozovat na mnoha platformách, Linux a Windows nevyjímaje. Tedy Cíl 1 – multiplatformní řešení, je tímto naplněn. Dalším cílem, který je splněn volbou PERLu, je i Cíl 5 – přenositelné řešení, díky jeho vlastnosti „interpretovaného jazyka“. Částečně je splněn i Cíl 4 – stabilní řešení, díky stabilitě vlastního programovacího jazyka. Otázku stability je ovšem nutné dále řešit, neboť mít stabilní programovací jazyk ještě neznamená mít stabilní program. Nedílným prvkem stability aplikace je samozřejmě samotná práce programátora a propracovaný návrh řešení.

Na otázku stability jsem nakonec našel řešení v podobě více-vláknového programování. Celou komponentu jsem tedy koncipoval jako hlavní vlákno, které se stará o obsluhu dalších podřízených vláken s důležitými funkčními částmi. Oddělil jsem tak od sebe části, které by mohli být problematické, a zároveň jsem tímto krokem celkově zpřehlednil zdrojový kód programu. Při této koncepci např. vlákno, které obsluhuje zápis získaných dat na pevný disk, při výpadku díky neznámé chybě, neohrozí chod vlákna obsluhujícího komunikaci s monitorovacím serverem nebo jakéhokoliv jiného prvku aplikace či vlákna. Tato koncepce se mi zdála tak výhodná, že vlastní snímání hodnot jsem rozdělil do dalších sub-vláken, čímž vznikla strategická možnost přidávání dalších nových obslužných sub-vláken a v podstatě tedy koncepce zásuvných modulů (pluginů). Použitím této konstrukce komponenty jsem si odpověděl dodatečně na Cíl4 – stabilní řešení a kompletně i vyřešil Cíl 3 – rozšiřitelné řešení.

V této chvíli už konečná podoba komponenty začala vypadat zcela reálně. Instalace balíčku PERLu je pro administrátory poměrně jednoduchou záležitostí a v mnoha případech je defaultně nainstalována v operačním systému od začátku, a pokud se ve zdrojovém kódu omezím na základní balíčky PERL distribuce, bude poměrně jednoduchou záležitostí i instalace komponenty pro monitoring serverů. Přičtu-li k tomuto i plánovaný způsob použití zásuvných modulů, kdy o jejich existenci a potřebě zavedení rozhoduje konfigurační soubor v textové podobě, bude splněn i vytyčený Cíl 2 – jednoduchost řešení.

Zbývalo tedy odpovědět na poslední nezodpovězený cíl, a to Cíl 6 – vyřešení průchodnosti dat Internetem. V dané chvíli jsem měl způsob, jak přečíst data z konkrétního senzoru, např. vytížení CPU. Měl jsem i způsob, jakým jej uchovat pro pozdější použití. Zápisem do binárního souboru prostřednictvím nástrojů RRDtool, *open source* projektu pro logování a následné zobrazování sériových dat [14]. Chyběl ale stále způsob, jak dostat lokální „log“ soubor na monitorovací server. Při náhodné procházení jedné z knih o PERLu mne napadla myšlenka použití LWP modulů pro interakci s www stránkou prostřednictvím PERLu. Znamenalo to sice rozšířit distribuci PERLu o další instalaci,

nicméně pro administrátora to bude jen další jednoduchý krok. LWP moduly obsahují řadu protokolů a metod, včetně těch pro práci s HTTP, HTTPs, FTP, NNTP a další [15]. Zvolil jsem metodu POST protokolu HTTP. Protokol HTTP je průchozí přes většinu nastavených firewallů a port 80 není většinou administrátory blokován hlavně z důvodu, že HTTP je využíváno pro WWW a je nejpoužívanějším protokolem na Internetu. Binární soubor s nasbíranými daty tak bude s velkou pravděpodobností možné dopravit z jakéhokoliv umístění do monitorovacího serveru, kde již může být libovolným způsobem, např. opět využitím RRDtool, zpracován. Protokol HTTP umožňuje jednodušší kooperaci s bezpečnostními prvky sítě, neboť jeho struktura komunikace je plně standardizována a její začlenění do bezpečnostní politiky organizace je poměrně jednoduché a finančně nenáročné. Navíc jednoduchou úpravou je možné data během komunikace šifrovat pomocí SSL v rámci HTTPS.

Touto syntézou jsem chtěl rekapitulovat způsob vzniku celé koncepce tématu mojí práce a to návrh a realizaci komponenty pro monitoring serverů, která by splňovala vytyčené cíle.

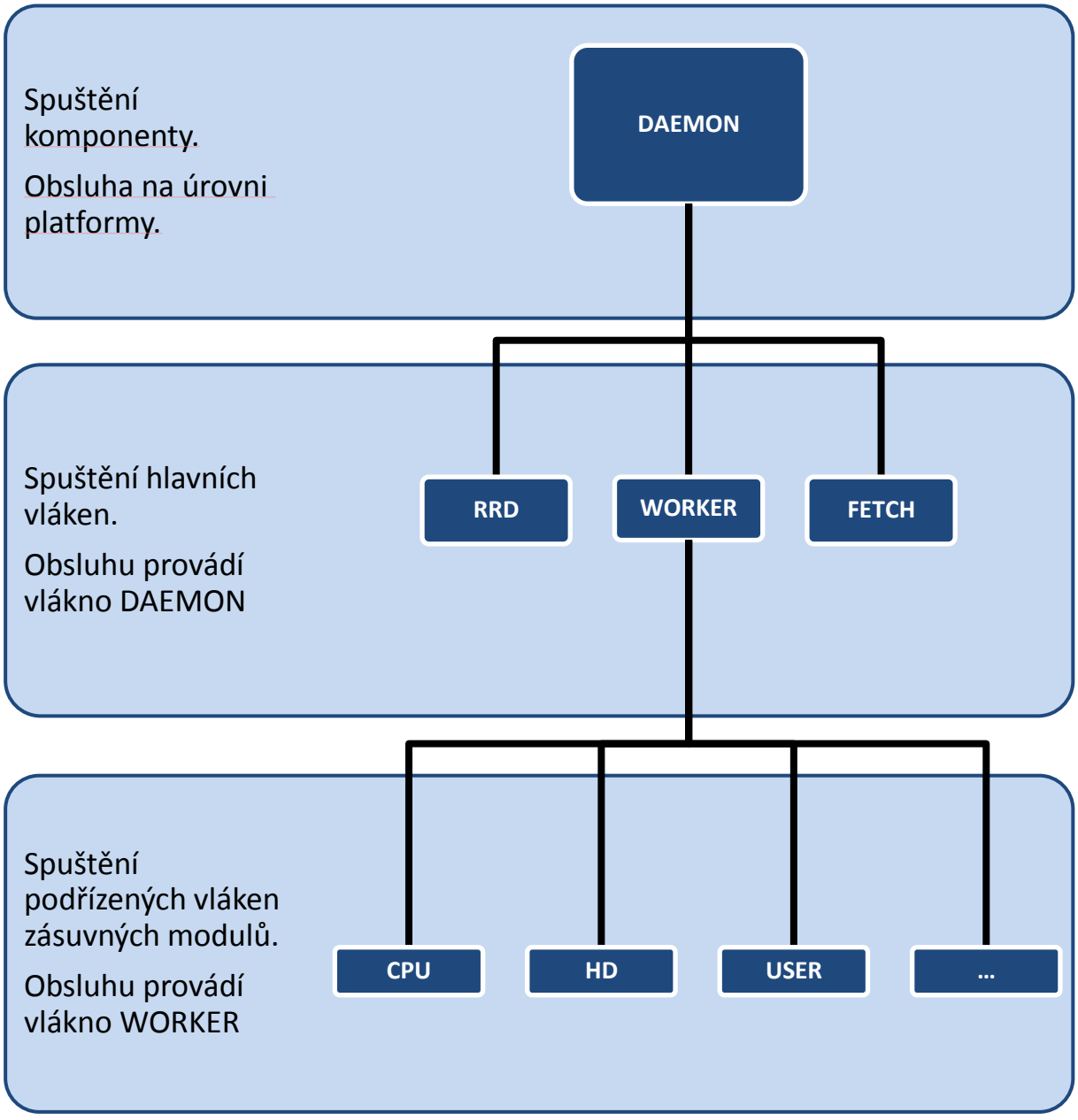
3.3 Výhody vlastního řešení

Mezi výhody vlastního řešení komponenty patří

- Univerzální přístup k informacím ze senzorů pomocí interpreteru
- Základní využití jazyka bez nutnosti dalších balíčků, vyjma LWP
- Řešení pomocí zásuvných modulů – nový modul = nové vlákno
- Jednoduchost použití
- Přenositelnost kódu
- Průchodnost sítí prostřednictvím HTTP
- Stabilita více-vláknového programování

4 NÁVRH KONCEPCE

4.1 Blokové schéma



Obr. 9 Blokové schéma komponenty.

4.2 Více-vláknová řešení

Nejjednodušším vysvětlením více-vláknového provozu (*Multitheading*) je, že provoz (fungování / vykonávání kódu) aplikace je rozdělen do více vláken, kde každé vlákno zpracovává jiný úkol bez ohledu na ostatní vlákna. Jednotlivá vlákna pak mohou spolu komunikovat, být synchronizována, nebo případně být uváděna do provozních stavů dle daného programovacího jazyku nebo možností implementace v rámci daného operačního systému.

Za více-vláknovou aplikaci je považována pak taková aplikace, která vykonává kód ve dvou anebo více vláknech. Vláknem se rozumí samostatně vykonávaná posloupnost instrukcí. O samotné pořadí vykonávání vláken se stará plánovač úloh. Primárně se provoz vláken aplikace jeví jako by fungovali plně paralelně, bez ohledu na fyzický počet procesorů. Pokud totiž nelze každému vláknu přidělit jeden procesor, je běh jednotlivých vláken přerušován a vykonávání kódu je předáváno jiným vláknům.

Pro všechna vlákna je viditelný společný paměťový prostor a mohou si prostřednictvím paměťových objektů například předávat data. Ve všech případech mohou jednotlivá vlákna číst veřejné paměťové objekty bez omezení, avšak zápis do daných paměťových objektů musí být ošetřen pomocí kritických sekcí. V praxi to funguje tak, že vlákno ukládající výsledky měření ze senzoru do alokované paměti aplikace, tuto paměť zamkne pro své výhradní používání pomocí kritické sekce a jiné vlákno do dané oblasti paměti tak nemůže zapisovat. Velmi často se stává, na systémech s více procesory, že v daný okamžik se dvě různá vlákna snaží zapisovat do stejného paměťového prostoru (bloku), i když s tím vývojář aplikace původně nepočítal. Takové stavy je proto nutné ošetřovat právě pomocí kritických sekcí a semaforového řízení k objektům.

Určitou další pomyslnou úrovní je ve více-vláknovém provozu tzv. paralelismus, kdy je proces rozložen do více vláken operačním systémem. Implementace paralelismu je vždy závislá na programovacím jazyku nebo operačním systému.

Výhody více-vláknových aplikací

- Aplikace, využívající více-vláknového provozu jednoznačně umožňují lepší a rychlejší fungování aplikace.
- Při více-vláknovém provozu dochází k hospodárnějšímu využívání systémových zdrojů.
- Vykonávání kódu aplikace není svěřeno pouze do jednoho procesoru, ale mezi více procesorů. Jsou tak plně využity všechny možnosti daného systému nebo platformy.

Nevýhody, resp. problémy více-vláknových aplikací

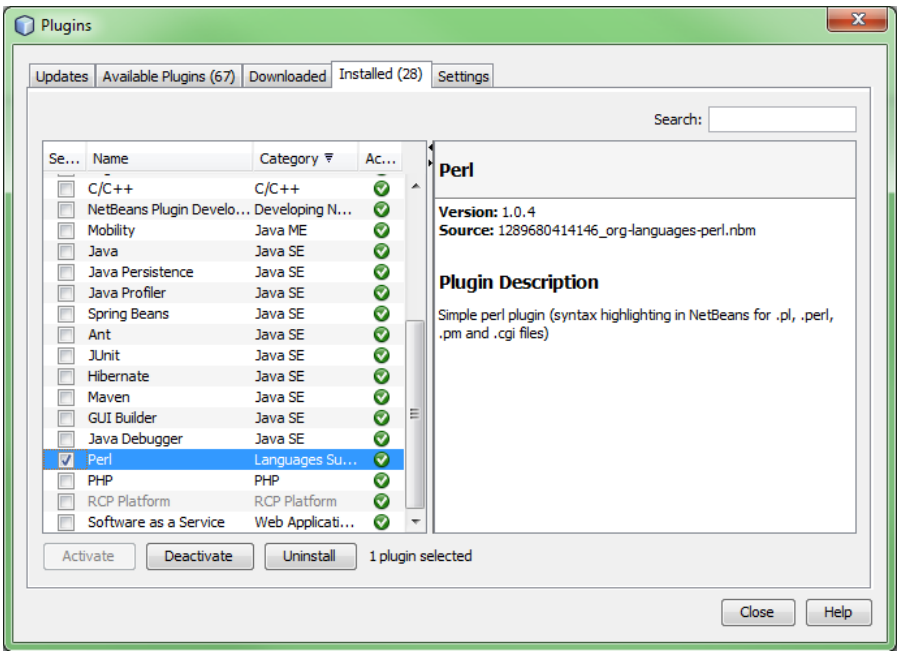
Nesprávné použití vláken způsobené nevhodným návrhem aplikace může vést ke konfliktům v rámci přidělování paměti a výpočetního času, resp. systémových prostředků. Typické problémy, na které je možné narazit u více-vláknových aplikací, jsou:

- Nesprávné rušení vláken bez znalosti následků u přidružených procesů nebo objektů. Např.: Jedno vlákno přijímá data na sítové vrstvě a předává je procesu pro analýzu paketů. V případě, že je takové vlákno ukončeno, proces řídící analýzu paketů nemá co zpracovávat. Samotná situace sice nevyvolá pád celého programu, ale program negeneruje žádné výsledky a jeho fungování je zcela zbytečné.
- Velké množství vláken v aplikaci může být i kontraproduktivní a zpomalit samotnou aplikaci. Na různých platformách je přidělování paměti a procesoru přímo závisle na vnitřních pravidlech jádra operačního systému, které se snaží blokovat problematické aplikace a neomezovat prostředky ostatních aplikací v systému.
- Samotná obsluha a synchronizace velkého množství vláken spotřebovává další výpočetní čas a paměť, resp. systémové prostředky.
- Počet procesů a vláken, které mohou v systému současně existovat je omezený. Špatným návrhem aplikace může dojít k vyčerpání povoleného množství vláken a díky tomu i k pádu samotné aplikace.

5 VÝVOJOVÉ PROSTŘEDÍ

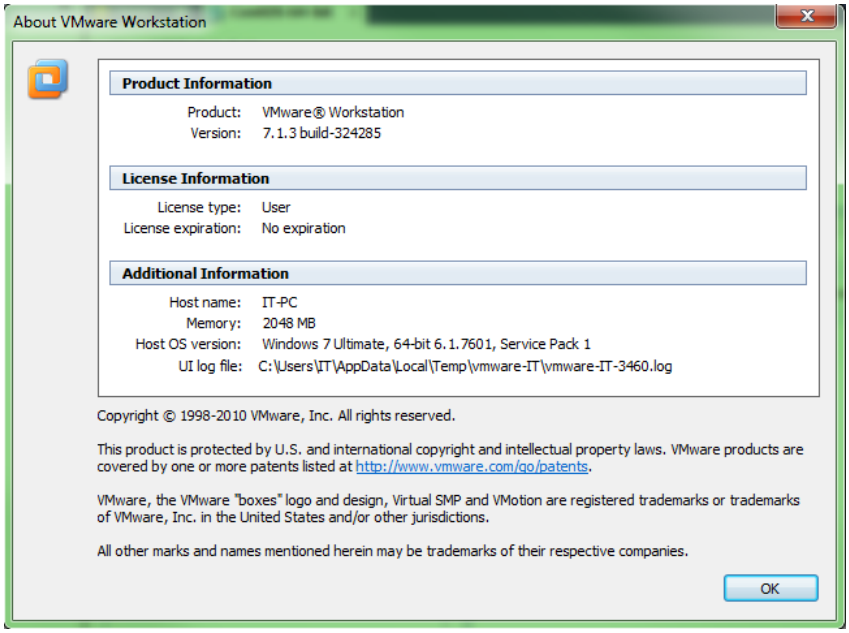
Vývojové prostředí pro realizaci programového kódu aplikace komponenty jsem zvolil volně šiřitelný *open source* nástroj pro vývojáře NetBeans IDE. Tento software disponuje podporou mnoha programovacích jazyků, jako jsou např. C/C++, PHP, JavaScript a Groovy. Další jazyky, jako je např. PERL, který je důležitý pro moji práci, je možné instalovat do NetBeans IDE formou zásuvného modulu.

Obr. 10 Informace o použitém vývojovém prostředí NetBeans IDE.

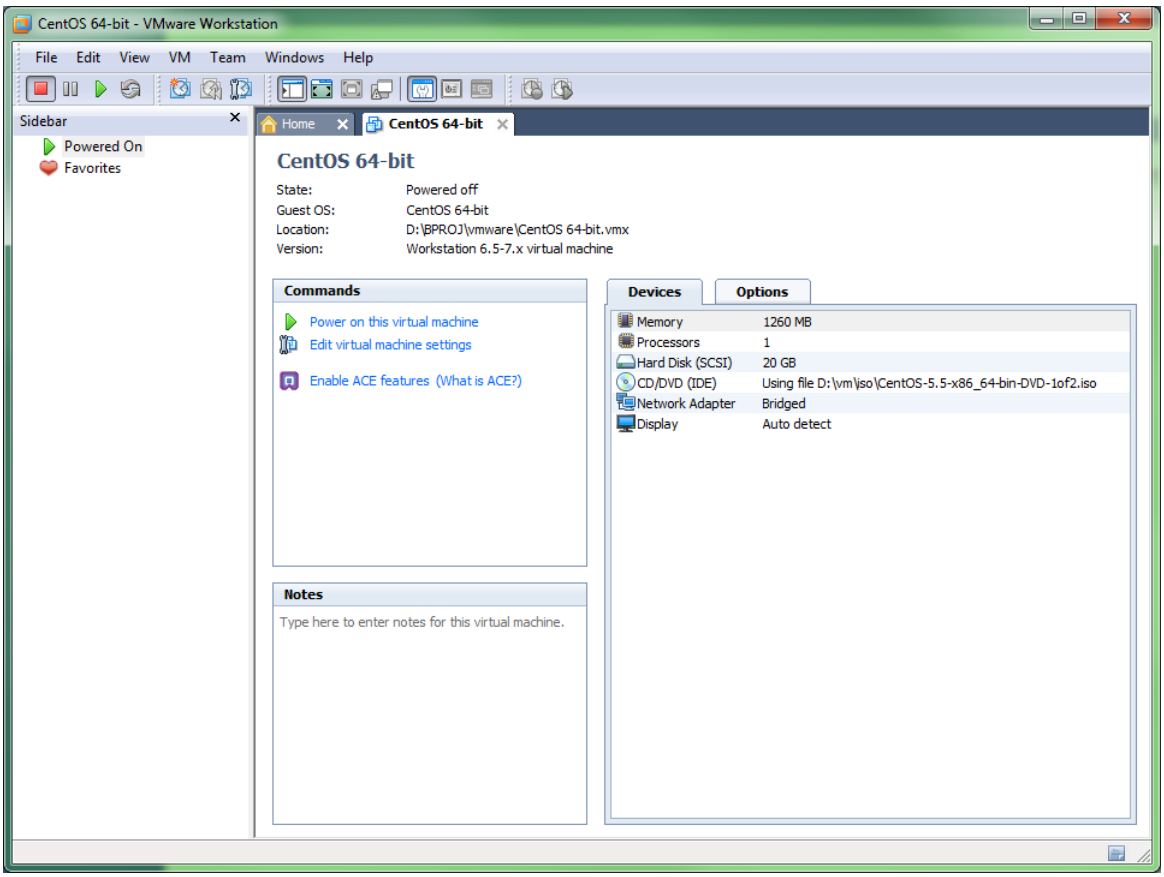


Obr. 11 Informace o použitém pluginu PERL pro vývojové prostředí NetBeans IDE.

Vlastní aplikace pak byla laděna na virtuálním serveru na platformě Linux, konkrétně 64-bitový operační systém CentOS 5.5. K virtualizaci jsem použil aplikaci VMware Workstation.



Obr. 12 Informace o použitém virtualizačním software VMware Workstation.



Obr. 13 Ukázka nastaveného virtuálního stroje v prostředí VMware Workstation.

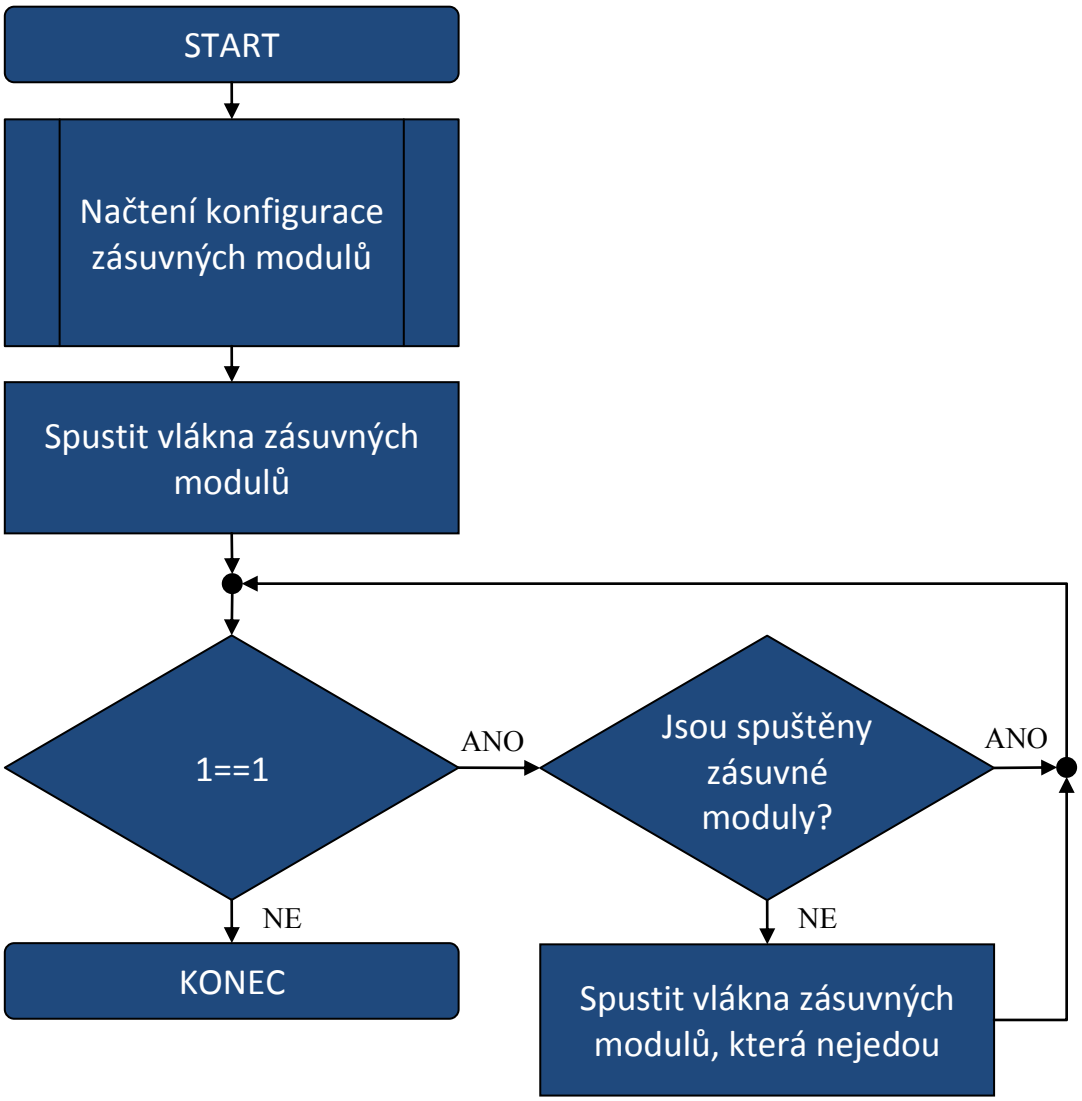
6 NÁVRH KOMPONENTY

Komponenta je sestavena podle koncepce uvedené v kapitole 4 Návrh koncepce (Obr. 9). Hlavní vlákno DAEMON obsluhuje načtení konfigurace z konfiguračního souboru a následně spouští a obsluhuje jednotlivá hlavní vlákna komponenty (RRD, WORKER, FETCH).

6.1 Vlákno WORKER

6.1.1 Popis vlákna

Vlákno WORKER je možné si představit jako nekonečnou smyčku, kdy na počátku je provedeno načtení konfigurace (parser textového souboru), resp. načtení jednotlivých typů zásuvných modulů a následně jsou tyto zásuvné moduly spuštěny. V nekonečné smyčce jsou pak jednotlivé zásuvné moduly v podobě podřízených vláken testovány, zda jsou funkční a spuštěná. Pokud některé z podřízených vláken přestane být funkční, dojde k jeho opětovnému nastartování. Toto řešení je imunní vůči výpadku kteréhokoliv z podřízených vláken zásuvných modulů a neohrozí tak chod celé komponenty.



Obr. 14 Zjednodušený vývojový diagram vlákna WORKER.

6.1.2 Zásuvné moduly

Jednotlivá měření v komponentě pro monitoring jsou realizována danými zásuvnými moduly, které jsou spouštěny periodicky pro získávání potřebných dat (např. vytížení CPU, volné místo na disku, apod.). Získaná data poté každý konkrétní zásuvný modul předává vyšší vrstvě / vláknu RRD, která s daty (hodnotami senzorů) provede update do binárního souboru rrd, respektive v rámci příslušného DS, potažmo RRA archívu.

6.1.3 Zásuvný modul na platformě LINUX/UNIX

Ukázka ze zásuvného modulu pro získání informací o volném místě na disku:

```
use strict;
use warnings;

our $DF_CMD = "df -kP";

sub measure
{
    my ( $self, $ts ) = @_;
    $ts = time;    # TODO

    if($self->{pgconf}{df_args}) {
      $DF_CMD .=" ".$self->{pgconf}{df_args};
    }

    my $out = '$DF_CMD 2>&1';
    if ($?) {
        my ( $sig, $st ) = ( $? & 127, $? >> 8 );
        $log->error("Command '$DF_CMD' failed: signal: $sig, status: $st");
        $log->error("Output: $out");
        return 0;
    }
    my @lines = split( /\n/, $out );
    shift(@lines);    # Header

    for my $line (@lines) {
        $line =~ s/^\s*//;
        my @fields = split( /\s+/, $line );
        my ( $dev, $size, $used, $mount ) = @fields[ 0, 1, 2, 5 ];
        if ( @fields == 6 ) {
            $self->update_sensor( 'fs_size', $ts, $size * 1024, $mount );
            $self->update_sensor( 'fs_used', $ts, $used * 1024, $mount );
        }
        else {
            $log->warn("Couldn't parse df line: '$line' ");
        }
    }
    return 1;
}
1;
```

Popis funkčnosti zásuvného modulu z ukázky výše.

Zásuvný modul využívá jednoduchého příkazu/aplikace z Linuxu "df". Příkaz DF spouští aplikaci zobrazující informace o využití jednotlivých součástí disků. Aplikace je vydávána pod GNU, je plně *open source* a existuje ve všech distribucích Linuxu. Příkaz je spouštěn s parametry „–k" a „–P". Parametr „k" znamená, že výstupní hodnoty pro velikosti bloku budou počítány v řádu kilobyte (kB). Parametr „P" nastavuje formátování výstupu tak, že každý záznam o jednotlivém disku bude vždy na jednom řádku. Tato vlastnost ve výpisu informaci příkazu DF zaručuje lepší formátování výstupních hodnot pro pozdější zpracování. Hodnoty vypisované aplikací DF nejsou nijak strukturně standardizované nebo v definovaném formátu například HTML, XML, CSV apod.

Fungování zásuvného modulu v krocích:

1. Nastavení výchozích hodnot
2. Spuštění příkazu „df -kP" pomocí interpreta příkazového řádku
3. Zpracování výstupu z příkazu „df –kP"
4. Předání jednotlivých zpracovaných hodnot vláknu RRD, starajícímu se o ukládání hodnot.

Ukázkový výstup při použití příkazu „df –kP" v BASH:

```
[root@lnx03 ~]# df -kP
Filesystem         1024-blocks      Used Available Capacity Mounted on
/dev/mapper/vg_lnx03-lv_root  51606140  43464280   5520420      89% /
tmpfs                   499432         0    499432       0% /dev/shm
/dev/sda1               495844     64015    406229      14% /boot
/dev/mapper/vg_lnx03-lv_home 153301060 150064412         0     100% /home
```

6.1.4 Zásuvný modul na platformě WINDOWS

Ukázka ze zásuvného modulu pro zjištění volného místa a získání informací o disku C:

```
use strict;
use warnings;
use Filesys::DfPortable;

sub measure
{
    my ( $self, $ts ) = @_;
    $ts = time;    # TODO

    my $ref = dfportable("C:\\");
    if(defined($ref)) {
    $used = $ref->{bused};
    $size = $ref->{blocks};

    $self->update_sensor( 'fs_size', $ts, $size, $mount );
    $self->update_sensor( 'fs_used', $ts, $used, $mount );
  }
  return 1;
}
1;
```

Popis funkčnosti zásuvného modulu z ukázky výše.

Zásuvný modul pro WINDOWS využívá funkce připojeného balíčku Filesys-Dfportable a funkce dfportable pro získání informací o disku.

Fungování zásuvného modulu v krocích:

1. Nastavení výchozích hodnot
2. Zavolání funkce dfportable(„c:\\“) pro získání informací o disku.
3. Zpracování výstupu z volání funkce.
4. Předání jednotlivých zpracovaných hodnot vláknu RRD, starajícímu se o ukládání hodnot.

6.1.5 Porovnání implementace obou platforem.

Při porovnání jednotlivých kroků na platformě WINDOWS a LINUX je jasné, že se programový kód nijak výrazně neliší. Hlavní kostra kódu je vždy stejná, ale mění se samotná funkce pro získání informací o disku dle dané platformy.

Jednoduchost modifikace při tvorbě zásuvného modulu pro jednotlivé platformy dává celému řešení jednoduchost přenositelnosti mezi platformami, či portaci na další nové operační systémy, kde plně funguje interpreter jazyka PERL.

Pro tvorbu dalších zásuvných modulů při rozšiřování komponenty jsou stanovena následující pravidla tvorby:

- Pro získávání jednotlivých dat je nutné použít co nejběžnější funkce systému
- Pro získávání jednotlivých dat je nutné využívat nejnižší funkce systému.
- Při tvorbě zásuvného modulu nealokovat zbytečně žádné velké množství paměti.
- Kód zásuvného modulu musí být co nejjednodušší

Dodržování těchto pravidel umožní plnění všech definovaných cílů v tomto projektu.

6.2 Vlákno RRD

6.2.1 Round Robin technika

Vlákno RRD je implementací *open source* nástroje RRDtool, který napsal Tobias Oiteker s přispěním mnoha lidí z celého světa. [16] Zkratka RRD vlákna vychází ze samotné funkčnost tohoto vlákna a tou je implementace techniky *Round Robin* na databázi – tedy *Round Robin Database*.

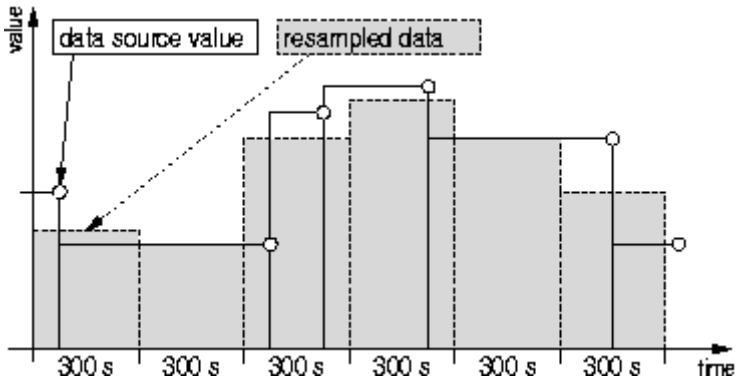
Round Robin je technika, která pracuje s fixní velikostí dat a ukazatelem na jednotlivé elementy, tedy místa, kam je možné uložit data. Můžeme si to představit jako uzavřený kruh s jednotlivými body po jeho obvodu, čímž jsou myšlena místa pro ukládání dat a šipka směřující ze středu do některého z bodů, což je pak ukazatel na data. Pokud jsou data zapsána nebo přečtena z daného elementu, ukazatel se přesune na další element v pořadí. Právě proto, že jde o uzavřený kruh, může se ukazatel neustále posunovat dále a dále. Jakmile jsou pro zápis vyčerpány všechny prázdné elementy kruhu, jsou znovu použity, od nejstaršího, již obsazené elementy. Tato koncepce je výhodná zejména proto, že datový soubor má od svého vzniku danou pevnou velikost a tím pádem nepotřebuje žádnou další údržbu v tomto smyslu. [16]

RRDtool je pak nástrojem, který pracuje s *Round Robin* databází a jeho prostřednictvím je možné data do databáze ukládat a zpětně z databáze načítat. [16]

6.2.2 Výhody RRD databáze

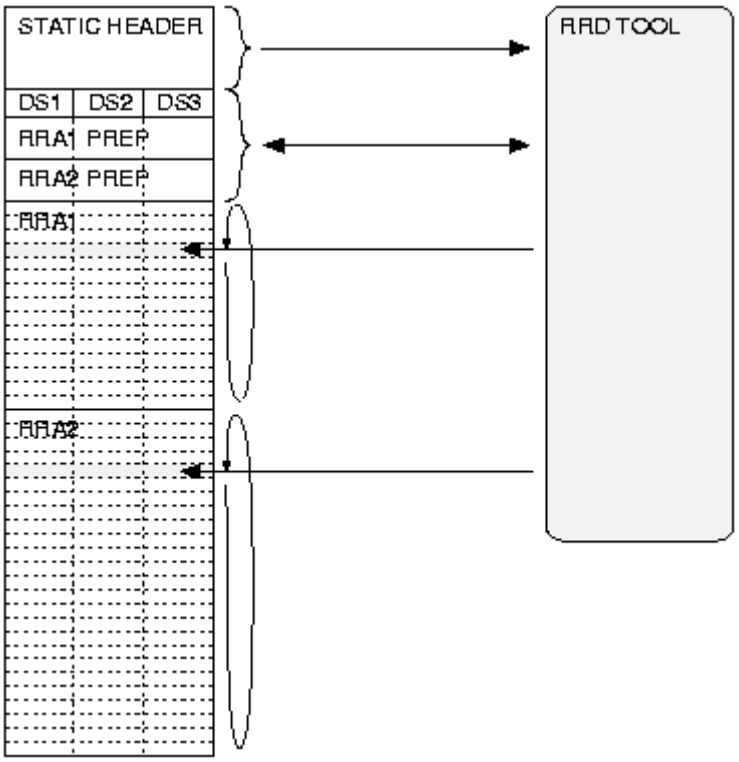
- Velikost databáze je určena při vzniku, respektive při jejím vytvoření a již se nemění.
- Databáze je uložena v binárním souboru.
- Databáze může uchovávat změny oproti předcházejícímu stavu.
- Databáze je efektivně navržena pro zobrazení grafu z uložených dat.

V podstatě se dá říci, že jde o sériový zápis dat s předdefinovaným intervalem zápisu hodnot.



Obr. 15 Ukázka ukládání dat v intervalu 300s. [17]

6.2.3 Popis vytvoření archivu pomocí RRDtool



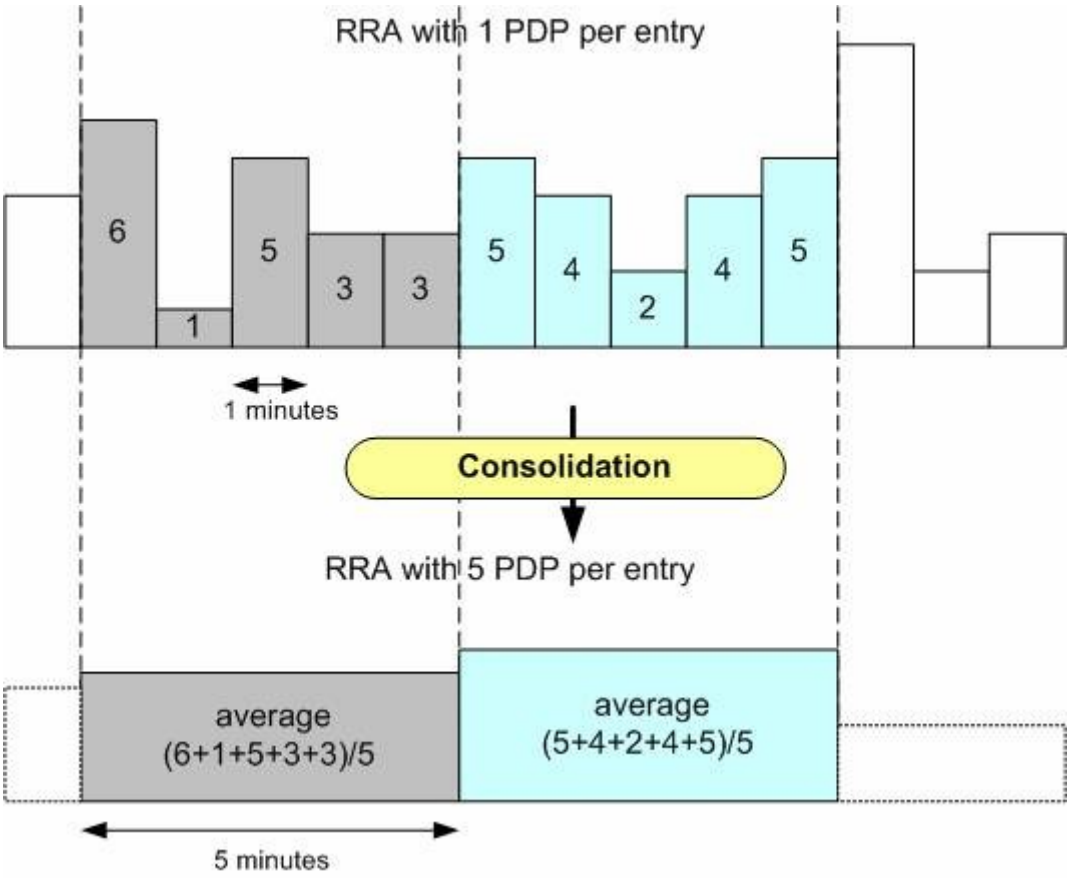
Obr. 16 Znázornění obsluhy RRA archivu. [18]

RRDtool nástroje vytvářejí databázový soubor v podobě „jméno_souboru.rrd“. Pro vytvoření souboru příkazem „rrdtool create“ je nutné zadaní startovacího času (*Start*), který se udává v součtu sekund od data 01. 01 1970 (začátku epochy) a kroku (*Step*), který určuje, jak často bude očekávána nová snímaná hodnota hodnota.

Dalším parametrem je zdroj dat, který je uveden klíčovým slovem DS (*Data Set*), za kterým následuje název proměnné datového zdroje. Nastavení datového zdroje pak obsahuje ještě typ datového zdroj DST (*Data Source Type*), který může nabývat hodnot COUNTER, DERIVE, ABSOLUTE a GAUGE. V případě prvních tří typů se ukládají ohodnocení změny vůči předchozímu vzorku, v případě typu GAUGE se bere přímo naměřená hodnota. Např. pro snímání vytížení CPU je vhodný typ GAUGE datového zdroje. Dále je nutné nastavit pro datový zdroj *heartbeat*, jinými slovy čas, po kterém má být uložena hodnota UNKNOW do databáze, pokud nepřijde relevantní hodnota v definovaném čase *Step*. Tato vlastnost RRDtool je výhodná zejména při kalkulaci vzorku při porovnávání s předchozím vzorkem. Pokud by totiž byla při neznámé hodnotě vkládána například

hodnota 0, docházelo by ke zkreslování celého výsledku sériového měření. Dalšími a posledními parametry datového zdroje jsou pak horní a dolní mez, které určují, v jakém rozmezí se má pohybovat očekávaná naměřená hodnota. Pokud je hodnota mimo rozsah, je opět do databáze vložena namísto naměřené hodnoty hodnota UNKNOW. Jednotlivé snímané hodnoty datového zdroje DS jsou nazývány *Primary Data Point*, zkratkou PDP.

Posledním parametrem příkazu „rrdtool create“ je RRA. Tedy vytvoření vlastního *Round Robin* archívu. Za klíčovým slovem RRA následuje určení tzv. konsolidační funkce, která může nabývat hodnot AVERAGE, MINIMUM, MAXIMUM a LAST. Princip konsolidace dat je patrný z obrázku (Obr. 17). Jde o to, že do archivu je možné jednotlivé PDP z datového zdroje seskupovat, resp. konsolidovat dle zadaných parametrů. Datovou jednotkou v RRA je pak, oproti PDP datového zdroje, CDP, neboli *Consolidate Data Point*.



Obr. 17 Ukázka rozdílné konsolidace PDP. [17]

Příklad příkazu:

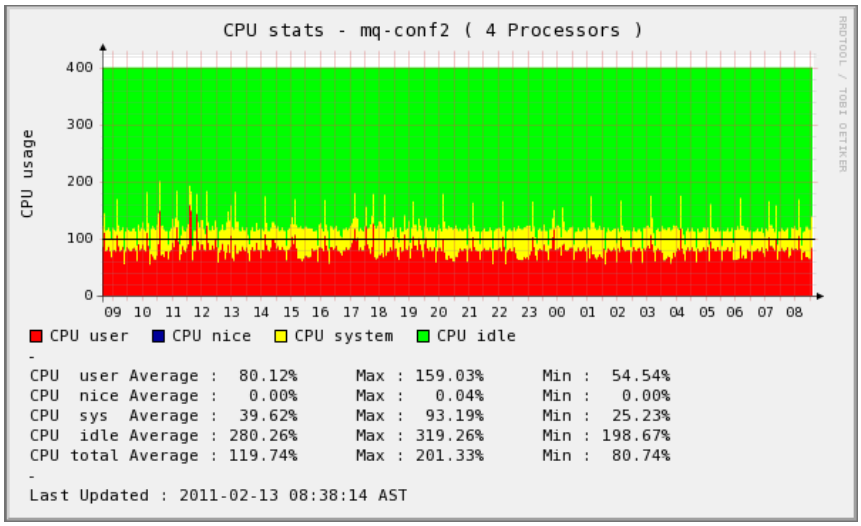
```
rrdtool create priklad.rrd \
     --start 1023654125 \
     --step 300 \
     DS:pamet:GAUGE:600:0:1048576 \
     RRA:AVERAGE:0.5:12:24
```

Ve výše uvedeném příkladu je vytvořen soubor priklad.rdd, do kterého budou ukládána data ze snímání paměti v intervalu každých 300s, tedy 5 minut. Snímat se budou skutečné hodnoty (typ GAUGE datového zdroje) v rozmezí (0 – 1GB). Konsolidační funkce archivu RRA je nastavena na AVERAGE, tedy 12 kroků (*Step*) PDP je aritmeticky průměrováno do výsledného CDP a 24 řádků takto konsolidovaných CDP je následně archivováno. 12 PDP je vlastně 12x 300 sekund, tedy jedna hodina. Jedno CDP tedy reprezentuje data z 1 hodiny. Těchto CDP bude uloženo 24. Uložená data

tedy reprezentují 1 celý den. Záměrně je použito slovo reprezentuje, neboť naměřená data již při zpracování podlehla matematické operaci použitím aritmetického průměru.

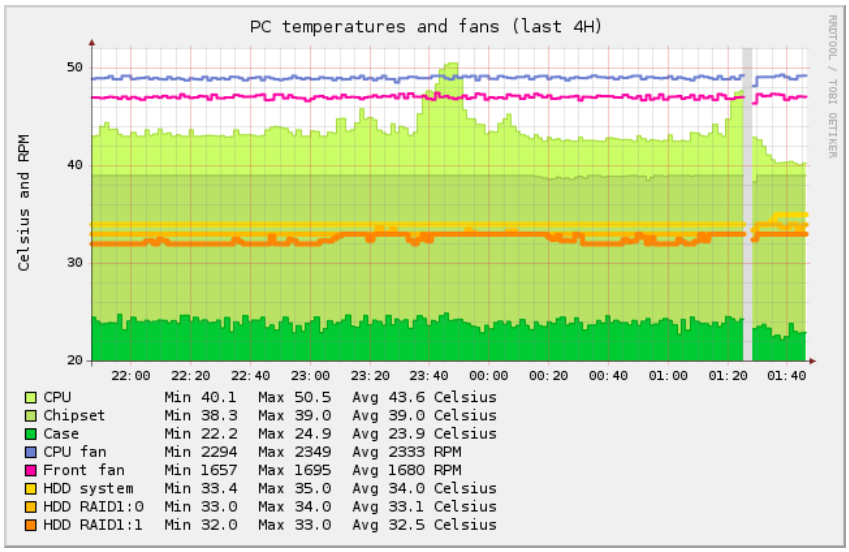
V rámci jednoho příkazu „rrdtool create“ je možné vytvořit více RRA archivů, jak je patrné z obrázku (Obr. 16).

6.2.4 Další možnosti RRDtool



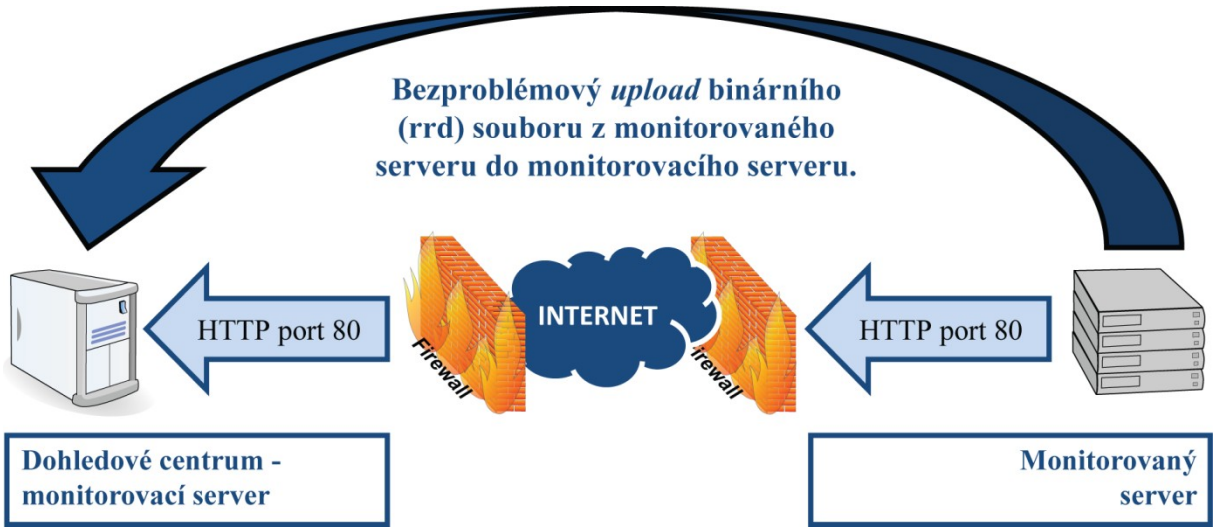
Obr. 18 Ukázka grafu snímání vytížení CPU pomocí RRDtool. [19]

Mezi další možnosti, potažmo výhody používání RRDtool patří možnost z uložených rrd souborů vytvářet a zobrazovat výsledné grafy, jak je patrné z předchozího popisu vytvoření rrd souboru, respektive jednotlivých RRA archivů. Jak již bylo uvedeno dříve, předmět této bakalářské práce, komponenta pro monitoring serverů, je součástí globálního systému. Vlastním zpracováním rrd souborů se zabývá jiná část tohoto globálního systému pro monitoring. Vlastní komponenta pouze snímá dané veličiny, prostřednictvím vlákna WORKER, potažmo jednotlivých zásuvných modulů, vytváří jednotlivé rrd soubory, prostřednictvím vlákna RRD a dopravuje je na monitorovací server, prostřednictvím dále popsaného vlákna FETCH k dalšímu zpracování. Další zpracování spočívá v podstatě ve vhodné manipulaci s jednotlivými binárními soubory rrd a s vytvářením reportů pro jednotlivé zákazníky, potažmo s vytvářením výstražných zpráv pro jednotlivé administrátory. Pro ilustraci uvádím některé z mnoha možností, jak mohou vypadat výsledné grafy s použitím RRDtool (Obr. 18, 19).



Obr. 19 Ukázka grafu snímání teploty a ventilátorů pomocí RRDtool. [20]

6.3 Vlákno FETCH



Obr. 20 Znázornění principu funkce vlákna FETCH.

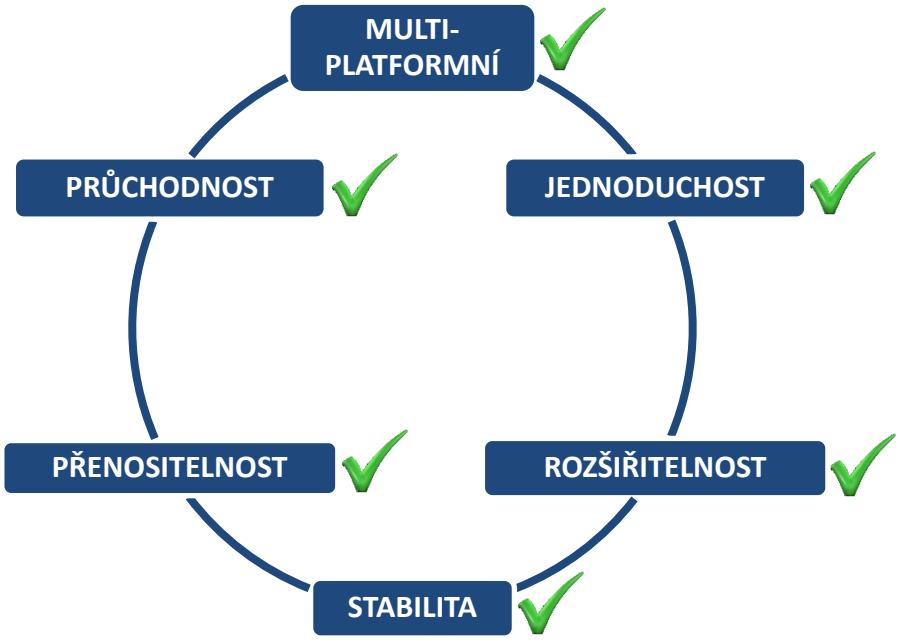
Vlákno FETCH slouží k přesunu binárního souboru rrd s RRA archívy z monitorovaného serveru na server monitorovací. Jak je patrné z obrázku (Obr. 20), jedná se o využití protokolu HTTP. Tento protokol jsem zvolil poté, co jsem hledal odpověď na otázku, jak co nejjednodušším způsobem přesunout data z jednoho serveru na druhý tak, aby byla co největší pravděpodobnost, že se přenos někde cestou nezastaví díky některému z bezpečnostních prvků.

Struktura komunikace pomocí protokolu HTTP je plně standardizována a začlenit ji do bezpečností politiky organizace je více než samozřejmý krok. Většinou právě porty 80 jsou standardně otevřené na všech prvcích po cestě z monitorovaného do monitorovacího serveru. Pomocí jednoduché úpravy pak je možné data během komunikace šifrovat v rámci HTTPS pomocí SSL.

Vlákno je v jazyce PERL napsáno s využitím LWP modulů, které obsahují řadu protokolů a metod pro práci s HTTP a HTTPS (včetně dalších). Jedním ze základních modulů LWP pro interakci se stránkou www je modul LWP::UserAgent, který jsem využil pro svoje účely. Jedná se o implementaci metody POST protokolu HTTP, prostřednictvím které je možné na danou www stránku provést upload souboru.

Celkový princip je takový, že dle nastaveného času, např. 5minut, se provádí upload kopie binárního souboru s archivem nasnímaných hodnot na monitorovací server. Na monitorovacím serveru tak vznikají po daném čase, tedy po pěti minutách, kopie binárního lokálního souboru rrd s archivem z daného monitorovaného serveru. Tímto způsobem je zajištěna i historie na několik dní zpět, přestože se např. binární soubor rrd v monitorovaném serveru nastaví pouze na jednodenní archiv (viz kapitola 6.2 Vlákno RRD). Tento způsob přenesení informace má i ten vedlejší efekt, že je možné na monitorovacím serveru sledovat, zda se pro daný sledovaný server v příslušné složce objevují po daném čase aktuální soubory a v případě, že nikoliv, je možné na tento stav reagovat jako na chybu komunikace či chybu monitorovaného serveru, obdobně jako např. při testu PING, v intervalu pět minut, na daný server.

7 ZÁVĚR



Obr. 21 Zhodnocení vytyčených cílů projektu.

Cílem této bakalářské práce byl vývoj komponenty pro monitoring serverů, která bude splňovat zadání, potažmo požadované cíle, definované v kapitole 2.4 Aspekt 3 – základní předpoklady vlastního řešení => Vytyčení cílů.

- Cíl 1. — Multiplatformní řešení:

  Díky volbě programovacího jazyka PERL, potažmo jeho vlastnosti interpretovaného jazyka, je tento cíl splněn. Komponenta je funkční všude tam, kde je možné provozovat PERL. Koncepcí zásuvných modulů a jejich poměrně jednoduchou úpravou pro danou platformu je tak možné komponentu provozovat na platformách Windows i Linux a s dalšími úpravami i na dalších operačních systémech.

- Cíl 2. — Jednoduchost řešení:

  Celková koncepce komponenty tak, jak je vytvořena, je velmi jednoduchá jak na instalaci a konfiguraci, tak i na samotné spuštění a vlastní fungování. Cíl jednoduchosti řešení je tedy splněn.

- Cil 3. — Rozšiřitelné řešení:

  Vytvořením komponenty formou zásuvných modulů s jednoduchou konfigurací a možností přidávání a odebírání dalších modulů je možné konstatovat, že rozšiřitelné řešení bylo splněno.

- Cíl 4. — Stabilní řešení:

  Komponenta je naprogramována jako více-vláknové řešení, kde každé vlákno samo o sobě je poměrně jednoduché a využívá zejména základních vlastností a možností programovacího jazyka PERL. Díky této stavbě lze považovat řešení za stabilní.

- Cíl 5. — Přenositelné řešení:

  Cíle přenositelného řešení je jednoznačně dosaženo volbou interpretovaného jazyka PERL.

- Cíl 6. — Vyřešení průchodnosti dat Internetem:

  Vlákno FETCH, které se stará o přesun dat z monitorovaného serveru na monitorovací, využívá metody POST protokolu HTTP. Díky této implementaci je vyřešena průchodnost Internetem,

poněvadž bezpečnostní politika velké většiny společností počítá s protokolem HTTP jako se standardním typem komunikace a průchodnost bezpečnostními prvky sítě je pro tento protokol většinou povolena. Cíl vyřešení průchodnosti dat Internetem je tedy také splněn.

Jak je patrné z rekapitulace jednotlivých vytyčených cílů, komponenta pro monitoring serverů naplňuje ve všech stanovených bodech očekávání a je schopná začlenění do globálního systému monitoringu společnosti HEXAGEEK s r.o. Vzhledem k možnosti rozšíření komponenty o další potřebné zásuvné moduly a vzhledem k multiplatformní koncepci komponenty se také předpokládá v blízké budoucnosti její rozšíření na jednotlivé servery společnosti a její aktivní využívání ve světě stále rostoucího počtu fyzických i virtuálních serverů.

8 CITOVANÁ LITERATURA

[1] **Cambridge University Press.** Definition: component. *Cambridge Dictionaries Online.* [Online] 2011. [Citace: 12. květen 2011.]
Dostupné z: http://dictionary.cambridge.org/dictionary/british/component.

[2] **Netcraft.** About Netcraft. *Internet Research, Anti-Phishing and PCI Security Services | Netcraft.* [Online] 2011. [Citace: 2. duben 2011.]
Dostupné z: http://news.netcraft.com/about-netcraft/.

[3] —. February 2003 Web Server Survey. *Internet Research, Anti-Phishing and PCI Security Services | Netcraft.* [Online] 2003. [Citace: 2. duben 2011.]
Dostupné z: http://news.netcraft.com/archives/2003/02/25/february_2003_web_server_survey.html#more-28.

[4] —. May 2011 Web Server Survey. *Internet Research, Anti-Phishing and PCI Security Services | Netcraft.* [Online] 2011. [Citace: 2. duben 2011.]
Dostupné z: http://news.netcraft.com/archives/2011/05/02/may-2011-web-server-survey.html#more-4490.

[5] **Bouška, Petr.** Začínáme s monitoringem sítě. *Fórum www.samuraj-cz.com, články.* [Online] 1. září 2009. [Citace: 12. květen 2011.]
Dostupné z: http://www.samuraj-cz.com/clanek/zaciname-s-monitoringem-site/.

[6] **Wikipedie.** Nagios. *Wikipedie - Otevřená encyklopedie.* [Online] 2011. [Citace: 10. květen 2011.]
Dostupné z: http://cs.wikipedia.org/wiki/Nagios.

[7] **Nagios.** Nagios Core Screenshots. *Nagios - The Industry Standard in IT Infrastructure Monitoring.* [Online] 2011. [Citace: 6. leden 2011.]
Dostupné z: http://www.nagios.com/products/nagioscore/screenshots.

[8] **ZABBIX SIA.** Screenshots. *Homepage of Zabbix :: An Enterprise-Class Open Source Distributed Monitoring Solution.* [Online] 2011. [Citace: 18. leden 2011.]
Dostupné z: http://www.zabbix.com/screenshots.php.

[9] **Zenoss, Inc.** Real-Time Server Monitoring. *Zenoss, The Cloud Management Company.* [Online] 2011. [Citace: 16. leden 2011.]
Dostupné z: http://www.zenoss.com/product/server-monitoring.

[10] **Splunk Inc.** What is Splunk? *Splunk Inc Company Web.* [Online] 2011. [Citace: 16. leden 2011.]
Dostupné z: http://www.splunk.com/product.

[11] **The Cacti Group.** About Cacti. *Cacti the complete rrdtool-based graphing solutions.* [Online] 2009. [Citace: 16. leden 2011.]
Dostupné z: http://www.cacti.net/index.php.

[12] **Bouška, Petr.** Zařízení v síti pod kontrolou. *Fórum www.samuraj-cz.com, články.* [Online] 21. září 2009. [Citace: 12. květen 2011.]
Dostupné z: http://www.samuraj-cz.com/clanek/zarizeni-v-siti-pod-kontrolou/.

[13] **Wikipedie.** Perl. *Wikipedie - Otevřená encyklopedie.* [Online] 2011. [Citace: 4. květen 2011.]
Dostupné z: http://cs.wikipedia.org/wiki/Perl.

[14] **Oiteker, Tobias.** About RRDtool. *RRDtool logging & graphing.* [Online] 2011. [Citace: 2. leden 2011.]
Dostupné z: http://www.mrtg.org/rrdtool/index.en.html.

[15] **Suehring, Steve.** *Beginning Web Development with Perl: From Novice to Professional.* 2560 Ninth Street, Suite 219, Berkeley, CA 94710 : Apress, 2006. ISBN: 1-59059-531-9.

[16] **Bogaerdt, Alex van den.** RRDtool - rrdtutorial. *RRDtool logging & graphing.* [Online] 2011. [Citace: 10. květen 2011.]
Dostupné z: http://oss.oetiker.ch/rrdtool/tut/rrdtutorial.en.html.

[17] **SARL, LUTEUS.** Round Robin Archive - RRA. *Network Monitoring Software - LoriotPro.* [Online] 2010. [Citace: 2011. duben 16.]
Dostupné z:
http://www.luteus.biz/Download/LoriotPro_Doc/RRD%20documentation/Round_robin_archive_RRA_EN.htm.

[18] **SCRIGROUP Int.** Introduction of MRTG and RRD tool. *SCRITUBE.* [Online] 2011. [Citace: 2. únor 2011.]
Dostupné z: http://www.scritube.com/limba/engleza/computers/INTRODUCTION-OF-MRTG-AND-RRD-T14881.php.

[19] **Rahumathullah, Amzath Ali.** RRDtool Gallery. *RRDtool logging & graphing.* [Online] 10 2010. [Citace: 14. 3 2011.]
Dostupné z: http://www.mrtg.org/rrdtool/gallery/index.en.html.

[20] **Popovici, Ciprian.** RRDtool Gallery. *RRDtool logging & graphing.* [Online] 10 2007. [Citace: 14. březen 2011.]
Dostupné z: http://www.mrtg.org/rrdtool/gallery/index.en.html.