Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Diplomová práce

Systém pro správu a monitorování počítačové sítě

Plzeň, 2012s **Eduard Chromík**

Prohlášení

Prohlašuji že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 17.května 2012, Eduard Chromík

Abstract

System for managing and monitoring computer networks. This thesis deals with monitoring and adjusting various devices located on a computer network using various protocols, through a web browser or mobile device. For these purposes, the system uses the Google App Engine and its API's. Used architecture separates each part in both runtime and implementation way, using web services to achieve better maintainability and customization. For displaying data in a web browser is used Vaadin framewrok.

Obsah

1. Úvod

Velmi častým problémem počítačových uživatelů jsou nesnáze s počítačovou sítí a její monitorování. Ať se jedná o uživatele v domácí síti, který se snaží odhalit problém pomalé sítě, či společnosti která se snaží co nejefektivněji využít své hardwarové prostředky. Především v dnešní době boomu cloudových řešení, s tím spojené budování a údržba rozsáhlých datacenter.

Základními funkcemi síťového managementu jsou:

Správa výkonu - měření výkonnosti a zatížení jednotlivých systémů sítě.

Správa konfigurace - monitorování sítě a síťové konfigurace z důvodů poznání vlivu jednotlivých elementů sítě na síťové operace.

Účetní a evidenční správa - cílem je monitorování parametrů využití sítě jednotlivými uživateli.

Správa poruch a chyb - cílem je detekce chyb a poruch sítě, jejich izolace a záznam do chybového souboru.

Správa bezpečnosti - řídí přístup k síťovým zdrojům podle stanovených pravidel tak, aby nemohlo dojít k neoprávněnému přístupu do sítě (ať už úmyslnému, nebo neúmyslnému) a zničení nebo zneužití dat.

Nejčastěji citovaným, a v této oblasti nejdůležitějším nástrojem síťového managementu, je právě SNMP (Simple Network Management Protocol). SNMP agent umístěný uvnitř jednotlivých zařízení uchovává hodnoty v MIB (Management Information Base) struktuře. Problém nastává ve chvíli, kdy je potřeba data zobrazit uživateli, protože nezpracované výpisy MIB stromů ve formě souvislých číselných hodnot jsou pro uživatele přinejmenším nepřehledné. Za tímto účelem vznikla bakalářská práce, která se navíc snažila odlišit od běžných řešení rozdělením na agenta a monitorovací stanici. [bak]

Existuje řada programů a nástrojů pro sledování s využitím SNMP a i jiných protokolů, bohužel žádný z těchto nástrojů nenabízí požadované vlastnosti, přestože některé nástroje jsou velmi rozsáhlé a umožňují mnoho nastavení a statistik. Nejčastějším problémem těchto nástrojů byla jejich složitost a popř. i cena. Tato diplomová práce je tedy snahou o rozšíření předešlé bakalářské práce s důrazem kladeným na přívětivost k uživateli a využitelnosti od jednoho uživatele až pro velkou společnost. Dalším požadavkem byla nezávislost na protokolu využívaným pro sledování jednotlivých zařízení, tedy pro případy kdy neobsahují podporu pro SNMP.

2. Teoretická část

Tato část se skládá z kapitol, obsahují přehled znalostí v dané oblasti, popis prostředků použitých pro řešení problému a zhodnocení existujících řešení.

2.1 Java 1.7

První volbou pro vývoj byl výběr verze programovacího jazyka. Verze JDK 7 již vyšla jako stable a verze JDK 6 je již "zastaralá". Bylo tedy potřeba zjistit klady a zápory jednotlivých verzí.

2.1.1 Plán „A" a plán „B" pro JDK7

Původní návrhy pro nové vlastnosti JDK 7 byly vytvořeny ještě ve firmě Sun Microsystems. Mělo se jednat o doslova revoluční verzi JDK, v níž se měly do programovacího jazyka Java přidat konstrukce známé spíše z funkcionálních programovacích jazyků (například lambda výrazy, ovšem typované!), z JVM se měla stát universální platforma pro mnoho dalších, většinou dynamicky typovaných programovacích jazyků atd. Ovšem celý vývoj JDK 7 se dostal do časového skluzu, což je ostatně už u mnoha velkých projektů spíše pravidlem než výjimkou. Navíc mezitím firma Sun otevřela cca 95% svého JDK pod názvem OpenJDK a v roce 2009 došlo k převzetí firmy Sun Microsystems firmou Oracle, takže se poněkud změnily priority vývoje (nejenom) JDK 7. Výsledkem všech těchto skutečností bylo sestavení dvou plánů: plánu „A" a plánu „B", s tím, že se po zralé úvaze rozhodne, který z plánů bude uskutečněn. [root]

V plánu „A" se počítalo s tím, že se vydá „revoluční" JDK 7 s velkým množstvím změn, ovšem až v roce 2012, což je velký časový skluz. Přednosti tohoto plánu spočívají v tom, že se počet dostupných verzí podporovaných JDK nebude příliš zvyšovat, ovšem velkou nevýhodou je již zmíněný časový skluz. Naproti tomu se v plánu „B" vývoj JDK 7 rozdělil na JDK 7, které má být dokončené již v polovině příštího roku a na JDK 8, které by snad mohlo být k dispozici na konci roku 2012. Předností tohoto plánu je to, že vývojáři budou moci již příští rok využít některé z nových (již implementovaných a otestovaných) vlastností JDK 7, ovšem nevýhoda spočívá ve větší fragmentaci JDK a jejich podpory: zatímco dnes se stále ještě poměrně často používá verze 1.4.2 a samozřejmě též 5.0 a 6.0, bylo by to v roce 2012 hned pět různých verzí JDK (podpora pro 1.4.2 a 5.0 sice teoreticky vyprší, ale mnoho firem nemůže z různých důvodů na novou verzi ihned přejít). Dá se tedy počítat s tím, že JDK 7 nebude přijato všemi vývojáři, protože si mnozí raději počkají až na JDK 8.[root]

2.1.2 Projekt Coin – rozšíření syntaxe a sémantiky Javy

Na konferenci JavaOne bylo prezentováno rozhodnutí o dalším vývoji JDK 7 – nakonec zvítězil plán „B“. Nové vlastnosti JDK tedy byly rozděleny do dvou množin, z nichž první bude implementována v JDK 7 a druhá až v JDK 8 o dva roky později. V novinkách, které budou v JDK 7, je zahrnuta například podpora pro dynamicky typované jazyky běžící v JVM (nová instrukce v bajtkódu), část projektu Coin (změna syntaxe a sémantiky jazyka), rozšíření NIO.2, podpora XRender Graphics Pipeline operacemi implementovanými v Java 2D (s čímž souvisí i nárůst výkonu některých grafických operací), projekt Nimbus pro Swingové aplikace, zahrnutí nových verzí TLS 1.2 a JDBC 4.1, podpora pro Unicode 6.0 atd. Pro běžné programátory budou asi nejviditelnější změny, které jsou implementovány v rámci výše zmíněného projektu Coin.[root]

Jedná se o projekt, který do programovacího jazyka Java vnáší malé, ale o to příjemnější změny v syntaxi a sémantice. V některých případech se jedná o pouhý „syntaktický cukr“, ale na další vlastnosti čekali programátoři mnohdy i několik let. Zajímavé je, že i přes rozšíření syntaxe jazyka Java nebylo nutné v rámci projektu Coin přidat do tohoto programovacího jazyka žádné nové klíčové slovo a taktéž se nemusel rozšiřovat bajtkód o další instrukce. Mezi nové prvky Javy, které projekt Coin přináší již do JDK 7 (v JDK 8 je těchto prvků mnohem více), patří především:

- Deklarace celočíselných binárních konstant
- Použití podtržítka pro lepší čitelnost numerických konstant
- Příkaz switch a řetězce
- „Diamant“ – zjednodušený zápis deklarací při použití generik
- Automatic Resource Management (ARM) – automatické volání metody close v rámci rozšířeného bloku try
- Vylepšené zpracování výjimek
- Velká optimalizace náročnosti běhu
- a mnoho dalšího...

[root]

2.2 Protokoly pro sledování

Jednou z hlavních rolí pro návrh a vývoj měli samozřejmě protokoly a agenti umístění v jednotlivých zařízeních. Bylo tedy zapotřebí prozkoumat možnosti a vybrat nejvhodnější řešení.

2.2.1 ICMP

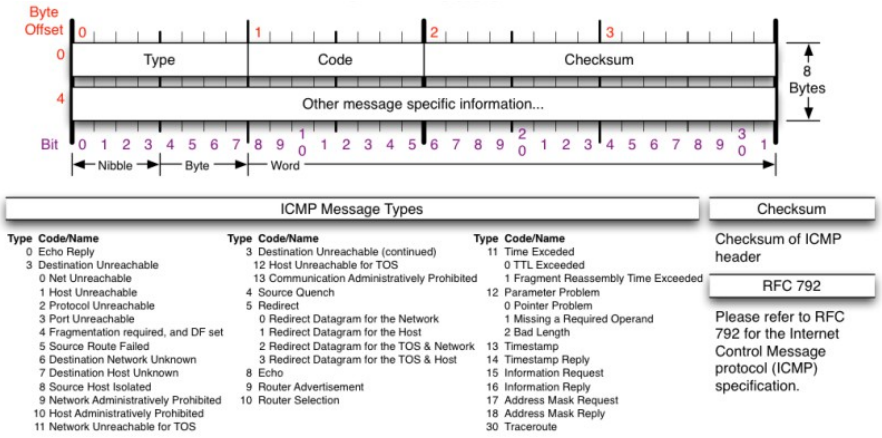
ICMP je dobře známý protokol jedná se o jeden ze základních protokolů pro hlášení chyb. Nabízí jen velmi málo operací, avšak je nedílnou a velmi podstatnou součástí sítí. Představím jej tedy také stručně.

2.2.1.1 Úvod

ICMP (Internet Control Message Protocol) protokol definovaný v RFC 792 jeden z nejdůležitějších protokolů ze sady protokolů internetu. Používají ho operační systémy počítačů v síti pro odesílání chybových zpráv, například pro oznámení, že požadovaná služba není dostupná nebo že potřebný počítač nebo router není dosažitelný. ICMP se svým účelem liší od TCP a UDP protokolů tím, že se obvykle nepoužívá síťovými aplikacemi přímo. Výjimkou je např. nástroj ping, který posílá ICMP zprávy „Echo Request“ (a očekává příjem zprávy „Echo Reply“), aby určil, zda je cílový počítač dosažitelný a jak dlouho paketům trvá, než se dostanou k cíli a zpět.[icmpwiki]

2.2.1.2 Princip

ICMP zprávy se typicky generují při chybách v IP datagramech (specifikováno v RFC 1122) nebo pro diagnostické a routovací účely. ICMP zprávy se konstruují nad IP vrstvou, obvykle z IP datagramu, který ICMP reakci vyvolal a tak (jako u UDP) ICMP nezaručuje doručení. IP vrstva patřičnou ICMP zprávu zapouzdří novou IP hlavičkou (aby se ICMP zpráva dostala zpět k původnímu odesílateli) a obvyklým způsobem vzniklý datagram odešle. Takže například každý stroj (jako třeba mezilehlé routery), který přeposílá IP datagram, musí v IP hlavičce dekrementovat políčko TTL („time to live“, „zbývající doba života“) o jedničku. Jestliže TTL klesne na 0 (a datagram není určen stroji provádějícímu dekrementaci), router přijatý paket zahodí a původnímu odesilateli datagramu pošle ICMP zprávu „Time to live exceeded in transit“ („během přenosu vypršela doba života“).[icmpwiki]



Obr.4 ICMP rámec [nmap]

2.2.2 SNMP

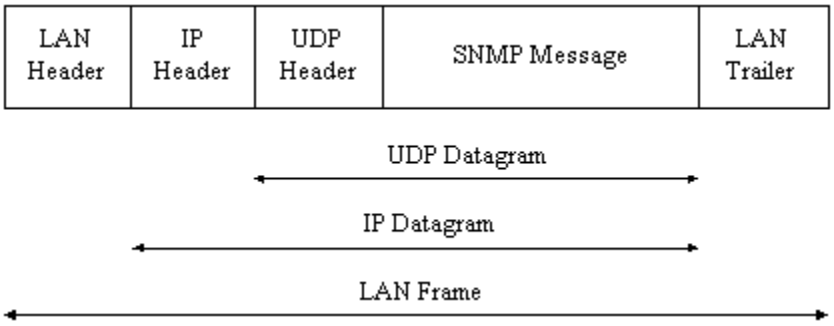
Jelikož je tento protokol již popsán v mé bakalářské práci, představím jej tedy pro úplnost jen ve zkrácené verzi.

2.2.2.1 Úvod

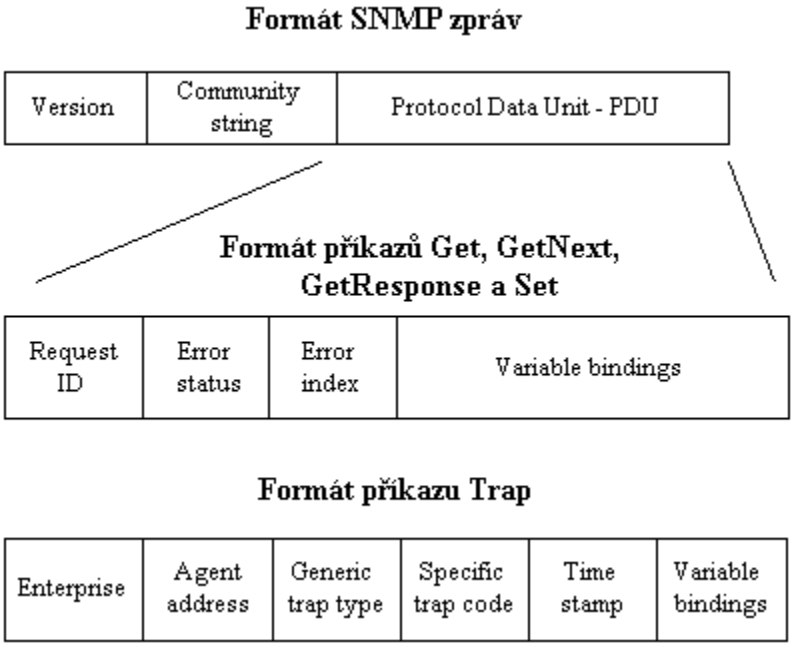
Protokol SNMP začal vznikat v roce 1988. SNMP (Simple Network Management Protocol) je, jak již název napovídá, jednoduchý protokol pro správu sítí. Umožňuje průběžný sběr nejrůznějších dat pro potřeby správy sítě a jejich následné vyhodnocování. Na tomto protokolu je dnes založena většina prostředků a nástrojů pro správu sítě.

2.2.2.2 Princip

Protokol SNMP je založený na architektuře klient-server. Server bývá označován jako agent, klient jako manažer. Agent běží na každém monitorovaném uzlu v síti. Komunikace mezi manažerem a agentem probíhá prostřednictvím protokolu SNMP, který běží na aplikační vrstvě a využívá služeb protokolu UDP (Port 161, 162). Oproti klasické klient-server architektuře není iniciování komunikace na straně klienta. SNMP je asynchronní protokol typu požadavek/odpověď. SNMP protokol má jen velmi malou sadu příkazů (proto označení Simple).[bak]

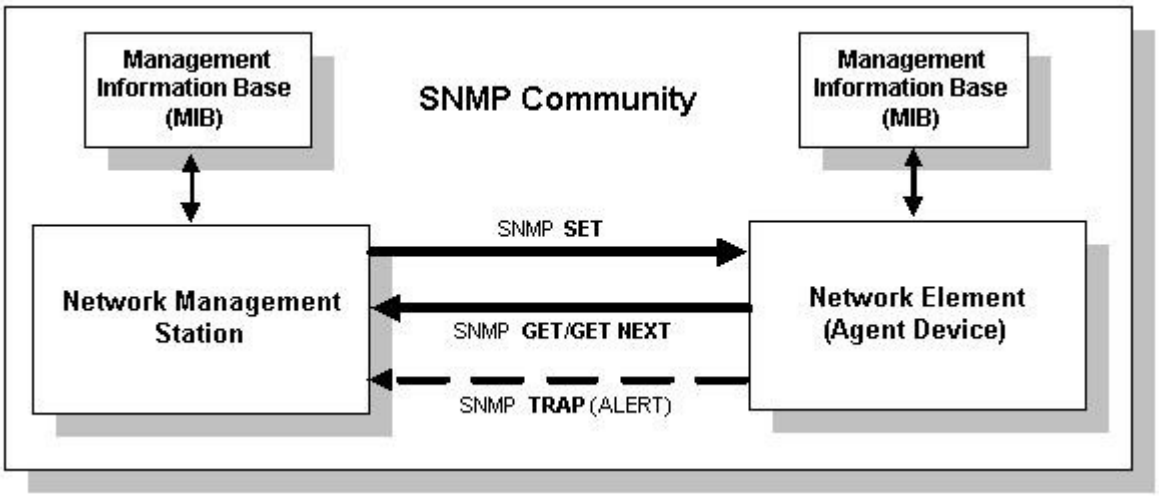


Obr.1 SNMP zpráva v LAN rámci [skriv]



Obr.2 Formát SNMP zprávy [skriv]

Na straně monitorované může existovat možnost takové konfigurace, kdy agent zašle manažerovi informace (SNMP MESSAGE) automaticky bez jeho požadavku. K tomu dojde zpravidla po splnění předem definované podmínky (výpadek, kolize, dosažení hraniční hodnoty…), agent nečeká na odpověď. Takové konfiguraci agenta se říká SNMP TRAP (tzv. past na události) viz. Obr. 3.

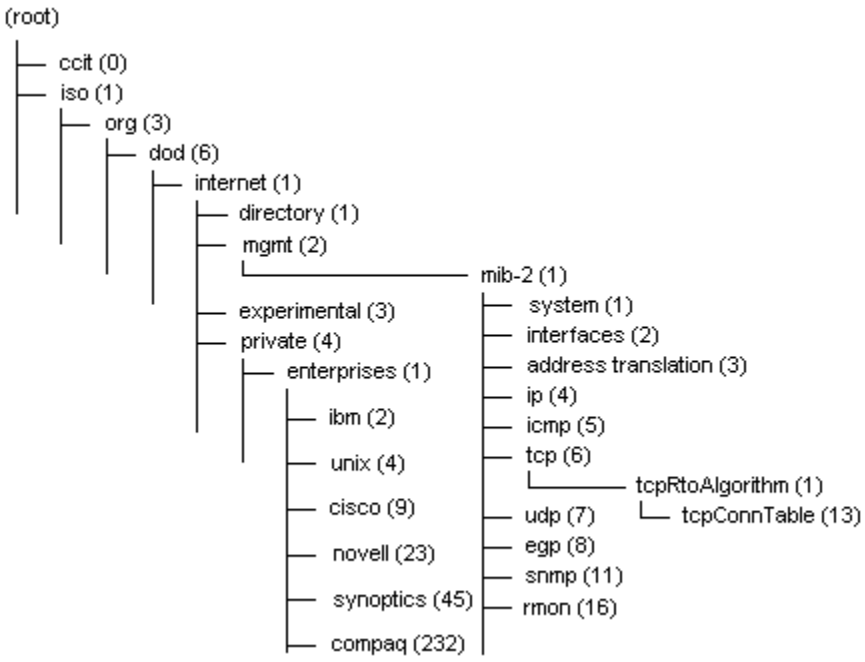


Obr.3 Ukázka komunikace Agent-Manažer [wtcs]

Agenti SNMP využívají pro uchování hodnot MIB tabulky, avšak tento princip využívá více protokolů a proto jsem uvedl stručný popis těchto tabulek v samostatné kapitole níže.

2.2.2.3 MIB

Jak už bylo zmíněno výše, MIB (Management Information Base) je datová struktura využívaná agenty. Jedná se o standard, který definuje, jaká data daný agent udržuje, a jaké operace jsou nad těmito daty povoleny. MIB má stromovou strukturu (viz. Obr6). Data jsou uložena v listech stromu. Všechny uzly ve stromu mají své označení, a to jak slovní tak číselné (OID - Object identifier). Na každý uzel ve stromu je možné se odkázat pomocí cesty od kořenu, kterou je pozice uzlu ve stromu vždy jednoznačně určena, neboť označení (slovní i číselné) každého uzlu je v rámci jeho bratrů unikátní.[bak]



Obr 6. Příklad MIB stromu [skriv]

MIB definice jednotlivých položek (listů) rovněž specifikuje, jaký typ dat může položka obsahovat. Obvykle obsahují hodnoty typu integer nebo string, ale mohou obsahovat i složitější struktury jako např. tabulky. Položky v MIB stromu se nazývají objekty. Objekty jsou sice listy MIB stromu, nicméně mohou mít více instancí. Například objekt ifNumber z kategorie "interfaces" obsahuje číselnou hodnotu vyjadřující počet síťových rozhraní, které dané zařízení obsahuje, ale objekt ipRoutingTable z kategorie "ip" obsahuje směrovací tabulku.[bak]

2.2.3 RMON

Přestože jsem tento protokol v mé bakalářské práci zavrhl, v této práci se pro něj opět otevřel prostor a možnost jej implementovat nebo jej aspoň zařadit jako možné rozšíření. Uvedu jej tedy opět ve zkrácené verzi.

2.2.3.1 Úvod

SNMP polling (neboli kontinuální monitorování) má své nevýhody:

- Ve větších sítích může významně přispět k zatížení sítě
- Při monitorování mnoha údajů může také podstatně zatížit správcovskou konzoli

Výše uvedené důvody, spolu s potřebou co nejdokonalejší detekce poruch na síti, analýzy dat pro potřeby proaktivního managementu, ladění výkonnosti, plánování změn a možnosti dálkového monitorování vzdálených segmentů i přes pomalé WAN spoje, vedly k vývoji standardu RMON. RMON tak umožňuje sběr různých statistických údajů o Ethernet i Token Ring segmentech, navíc RMON MIB (Management Information Base) standard umožňuje vzdálené monitorování využitím již zavedeného protokolu SNMP.[bak]

Standard RMON významně mění způsob komunikace mezi zařízeními a správcovskou konzolou. Dochází k rozšíření možností správy a k odlehčení komunikace přesunutím větší části činnosti na agenta, což je často v přetížených sítích velkým přínosem. RMON agenti slouží k monitorování provozu na síťových segmentech a mohou to být buď samostatné sondy, které se připojí stejným způsobem jako jakékoliv jiné zařízení k danému segmentu, nebo jsou součástí inteligentních síťových zařízení.[bak]

2.2.3.2 Princip RMON

RMON standardy jsou definovány jako MIB, tzn. SNMP datové struktury, skládající se ze statistických tabulek dosažitelných SNMP příkazy. RMON sonda používá tyto tabulky ke sběru dat nezávisle na managemen aplikaci, která zařízení sleduje, čili bez její výpočetní zátěže a zátěže síťového provozu. RMON je typickým příkladem distribuovaného řešení. Stejně jako klasický SNMP model, i RMON model se skládá ze dvou částí. Tou první je agent (sonda), která je umístěna na monitorovaném segmentu. Druhou částí je management aplikace.[bak]

Pomocí této RMON aplikace můžeme zkonfigurovat agenta ke sběru požadovaných informací, které jsou zasílány na centrální konzolu při vyžádání nebo při výskytu definované události. Těchto agentů - sond může být rozmístěno v síti tolik, kolik má síť segmentů. Samozřejmě u velkých sítí se tyto sondy umisťují strategicky jen na nejdůležitější segmenty, případně podle potřeby na ty segmenty, kde se objevují nějaké problémy.[bak]

2.2.4 IPMI

IPMI (Intelligent Platform Management Interface) je vznikající standard od roku 1998, pro management a správu serverů. Není zatím rozšířen tolik jako výše uvedené protokoly, avšak domnívám se, že má velký potenciál.

2.2.4.1 Úvod

Servery s funkčností standardu IPMI umožňují administrátorovy přístup a monitoring hardware serverů, jeho diagnostiku, správu napájení a přesměrování konzole po síti. Vývoj této specifikace rozhraní vedl Intel Corporation a je podporován více než dvěma sty dodavatelů počítačových systémů. Mezi nimiž jsou např Dell, HP a další. Avšak tito výrobci mají své implementace využívající toto rozhraní (ILOM, DRAC atp.)

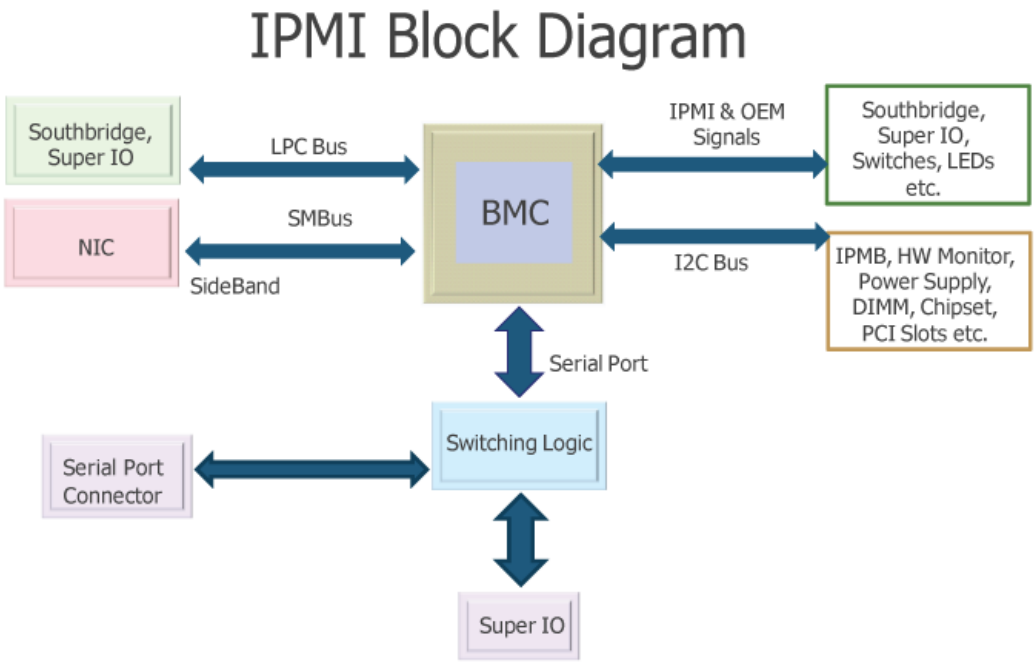
2.2.4.2 Princip

IPMI je specifikace rozhraní postaveného na vlastním Hardwaru, které zasílá zprávy. IPMI sub-systém funguje nezávisle na operačním systému (OS), aby správcům umožnil spravovat systém na dálku v případě absence operačního systému .

Správci systému mohou:

- Použít IPMI zprávy ke sledování stavu platformy (např. systémové teploty, napětí, ventilátorů,napájení atp.),
- Pro přehled hardwarových záznamů
- Pro provedení obnovy řízení, jako je vypnutí či restart aj.
- Standard také definuje upozorňovací mechanismus, umožňující systému poslat SNMP Platform Event Trap (PET).

Tyto schopnosti jsou umožněny speciálním procesorem zvaným baseboard management controller (BMC), umístěným na základní desce serveru nebo v šasi blade serveru. Procesor BMC je spojen s hlavním procesorem a dalšími jednotkami základní desky jednoduchou sériovou sběrnicí. Pomocí ní monitoruje základní parametry motherboardu (např. teplotní senzory, status CPU, otáčky větráků a napájení), umožňuje vzdálené řízení napájení serveru (čili schopnost restartu serveru) a přístup ke konfiguraci BIOSu a informacím konzole operačního systému. Protože BMC je samostatný procesor, systém pracuje bez ohledu na to, běží-li hlavní procesor serveru nebo ne.



Obr.5 IPMI komponenty [wiki]

Přístup administrátora k informacím poskytovaným procesorem BMC je možný buď pomocí správní aplikace kompatibilní se standardem IPMI ze síťového desktopu nebo vzdáleně pomocí webového rozhraní z tzv. out-of-band zařízení se zabudovaným IPMI firmware.

Veze 2.0 specifikace IPMI podporuje nově také přesměrování sériové konzoly na IPMI pomocí protokolu IP. Administrátor má tak plný vzdálený přístup i k těmto čistě textovým informacím a může tak ovládat např. i BIOS. Dosud byl tento přístup omezen pouze na sériovou konzoly serveru.[svetsiti]

2.3 WS - Webové služby

Webová služba je softwarový systém umožňující interakci dvou strojů na síti. Webové služby umožňují jednoduchou komunikaci mezi aplikacemi ve velmi různorodém prostředí, protože komunikace je založena na platformě nezávislých standardech – především na jazyce XML a protokolu HTTP. Aplikace si mezi sebou posílají XML zprávy, které přenášejí dotazy a odpovědi jednotlivých aplikací. Webové služby tak nabízejí vrstvu pro přístup k datům a data přístupné přes standardní internetové protokoly. Avšak nejedná se o GUI pro uživatele, spíše lze WS vidět jako API pro ostatní aplikace. Mezi 3 nejznámější a nejpoužívanější interakční modely patří RPC(Remote procedure call), SOAP(Simple Object Access Protocol), REST(Representational State Transfer), které jsou popsány níže.[kosek]

2.3.1 RPC

Nejstarší ze všech uvedených přístupů avšak v některých případech se stále využívá. V rámci diplomové práce jej uvádím spíše z důvodů pro srovnání.

2.3.1.1 Úvod

RPC (Remote procedure call, neboli vzdálené volání procedur) je technologie dovolující programu vykonat proceduru, která může být uložena na jiném místě než je umístěn sám volající program. Příkladem budiž výpočet funkce na jiném počítači v síti.[rpcwiki]

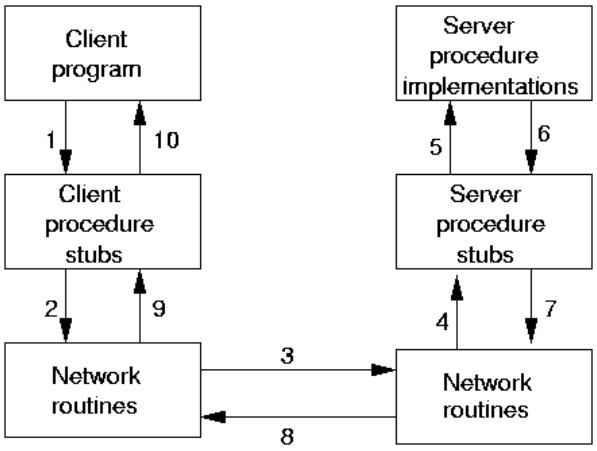
Vznikly tři základní RPC standardy:

- ONC (Open Network Computing) - Distribuovaná aplikační architektura, navržená firmou Sun Microsystems.
- DCE (Distributed Computing Environment) - Tato implementace byla navžena skupinou OSF (Open Software Foundation) a v současnosti ji dodává řada komerčních firem (narozdíl od ONC RPC za peníze)
- Microsoft COM/DCOM „standard‘‘ - Distribuovaná aplikační architektura, navržená firmou Microsoft.

RPC byl ještě rozšířen na **XML-RPC**. Tedy na protokol, s jehož pomocí lze velice jednoduše provádět vzdálené volání procedur. XML-RPC nepřineslo do světa vzdáleného volání procedur novou technologii. Jedná se totiž o soubor pravidel, které pouze říkají, jak použít již funkční a dokonce standardizované technologie pro potřeby RPC. Data jsou zapouzdřena pomocí značkovacího jazyka XML (eXtensible Markup Language) a přenášena díky protokolu HTTP. Taková koncepce umožňuje aplikacím, napsaných v různých programovacích jazycích, komunikaci mezi různými počítačovými architekturami a jejich operačními systémy. V současné době je projekt ukončen, nicméně stal se předlohou pro protokol SOAP.

2.3.1.2 Princip

1. Nejprve proběhne jednoduché zabalení parametrů a identifikátorů procedury do formy vhodné pro přenos mezi počítači (marshalling)
2. Síťové funkce OS se zavolají pro odeslání zprávy
3. Odeslání zprávy
4. Balíček se na vzdáleném místě rozbalí, zjistí se o jakou proceduru jde (unmarshalling)
5. Samotné zavolání a provedení procedury
6. Procedura se dokončí a vrátí výsledek
7. Výsledek procedury se opět zabalí (marshalling)
8. Zpráva se odešle
9. Klient zprávu přijme
10. Provede se unmarshalling a přijatá hodnota se předá proceduře



Obr.7 princip fungování RPC [newmarch]

2.3.2 SOAP

SOAP je nástupce XML-RPC, ačkoliv si zapůjčuje jeho způsob přenosu dat a další vlastnosti. Nejčastěji se tedy používá v modelu požadavek/odpověď.

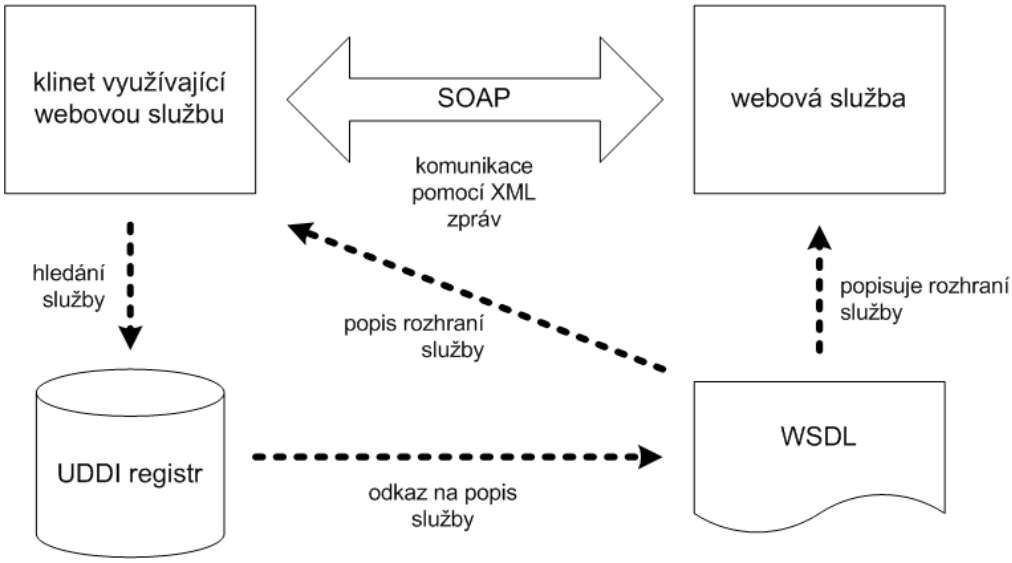
2.3.2.1 Úvod

SOAP (Simple Object Access Protocol) je protokolem pro výměnu zpráv založených na XML přes síť, hlavně pomocí HTTP. Existuje několik různých druhů šablon pro komunikaci na protokolu SOAP. Nejznámější z nich je RPC šablona, kde jeden z účastníků komunikace je klient a na druhé straně je server. Server ihned odpovídá na požadavky klienta.

2.3.2.2 Princip

Ke každé webové službě by měl být k dispozici její formální popis v jazyce WSDL. Z tohoto popisu již jde automaticky vygenerovat soapový požadavek. Ve větších systémech nebo přímo v otevřeném prostředí Internetu se popis služby může zaregistrovat do UDDI (Universal Description Discovery and Integration viz. použité zkratky a pojmy) registru. Ten slouží jako jakýsi telefonní seznam („zlaté stránky“), který umožňuje vyhledávání služeb s určitými parametry. Klient, který chce využít webovou službu, získá buď přes UDDI, nebo přímo její popis. Z něj je jasné, jakou strukturu má mít soapová zpráva a kam se má webové službě poslat, aby ji rozpoznala.

Jedna aplikace pošle v XML zprávě požadavek druhé aplikaci, ta požadavek obslouží a výsledek zašle jako druhou zprávu zpět původnímu iniciátorovi komunikace. V tomto případě bývá webová služba vyvolána webovým serverem, který čeká na požadavky klientů a v okamžiku, kdy přes HTTP přijde soapová zpráva, spustí webovou službu a předá jí požadavek. Výsledek služby je pak předán zpět klientovi jako odpověď.[kosek]



Obr.8 SOAP architektura [kosek]

2.3.3 REST

V současnosti nejvíce používaným přístupem k WS. Jedná se o nejnovější přístup, který byl navrhnut a popsán v roce 2000 Royem Fieldingem.

2.3.3.1 Úvod

REST je architektura, který umožňuje přistupovat k datům na určitém místě pomocí standardních metod HTTP (HTTP 1.1 methods). Pomocí REST lze ovládat i stav aplikace, pokud jej dokážeme popsat takovým způsobem, že si vystačí s modelem „zdroje – CRUD (Create, Read, Update, Delete) akce“.

REST nabývá na významu a stává se spolu s JSON (JavaScript Object Notation, popsaný níže.) defacto standardem pro API webových služeb. Jeho rozšíření napomáhá i technika AJAX, které REST vychází vstříc. Jeho rozšíření napomohlo i to, že se nijak zásadně neliší od standardního volání a získávání dat pomocí HTTP, pouze je zobecňuje. Dá se říct, že REST je architektura, která umožňuje CRUD operace pomocí standardních HTTP dotazů.

Moderní frameworky pro vývoj server-side aplikací pomáhají vytváření REST rozhraní tím, že dokáží nadefinovat patřičné procedury pro všechny potřebné metody, takže vytvoření vlastního RESTful rozhraní je opravdu snadné. Čím dál tím častěji je vidět v přehledech vlastností frameworků kromě (např.) „Podporuje MVC“ i „Podporuje REST“.

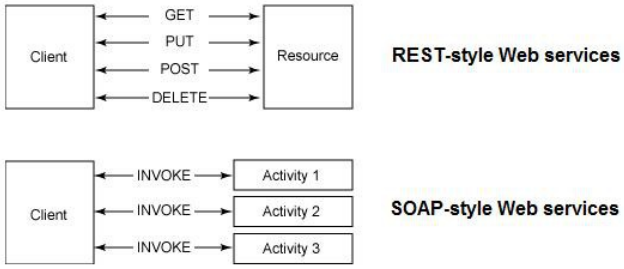
REST svou bezstavovostí vychází vstříc moderním metodám vývoje webových aplikací, které jsou založené na paralelním zpracování distribuovaného obsahu. Tato jeho vlastnost, spolu s výše zmíněnými výhodami (jednoduchost a nenáročnost) naznačuje, že se s rozhraními, postavenými na modelu REST, budeme do budoucna setkávat stále častěji.[zdroják]

2.3.3.2 Princip

Rozhraní REST je použitelné pro jednotný a snadný přístup ke zdrojům (resources). Zdrojem mohou být data, stejně jako stavy aplikace (pokud je lze popsat konkrétními daty). REST je tedy na rozdíl od známějších XML-RPC či SOAP, orientován datově, nikoli procedurálně. Všechny zdroje mají vlastní identifikátor URI a REST definuje čtyři základní metody pro přístup k nim.[restwiki]

Základní principy

- Stav aplikace a chování je vyjádřen takzvaným resourcem (klíčová abstrakce)
- Každý resource má unikátní identifikátor (URL, URI)
- Je definován jednotný přístup pro získání a manipulaci s resourcem v podobě čtyř operací CRUD (Create - PUT, Read - GET, Update - POST, Delete - DELETE)
- Resource může mít různé reprezentace (XML, HTML, JSON, SVG, PDF)



Obr.9 REST oproti SOAP [itf]

2.4 Webový server

Webový server může být:

- Počítač, který je odpovědný za vyřizování požadavků HTTP od klientů - programů zvaných webový prohlížeč. Vyřízením požadavků se rozumí odeslání webové stránky. Webové stránky jsou obvykle dokumenty HTML.
- Počítačový program, který provádí činnosti popsané v předchozím bodě (démon).

Jedná se tedy o prostředí vhodné pro běh webových služeb. Pro hostování webového serveru je zapotřebí aplikačního serveru, který může být pod správou jednotlivce (firmy) nebo řešen jako takzvaný "Cloud".

2.4.1 Aplikační server

Aplikační server tvoří vrstvu mezi operačním systémem a aplikacemi. Podobně, jako operační systém poskytuje základní funkce programům (například pro přístup k souborovému systému, nebo ke správě procesů), poskytuje aplikační server často používané funkce enterprise aplikacím. Vytváří další vrstvu abstrakce, aby bylo psaní aplikací jednodušší. Příkladem takových funkcí mohou být podpora transakčního zpracování požadavků, persistence objektů do databáze, výměna zpráv mezi aplikacemi a další. Nabízí se samozřejmě otázka, co to vlastně je enterprise aplikace a jak se liší od běžné aplikace. Není to nic složitého, de facto se jedná o běžnou aplikaci, na kterou jsou kladeny určité nároky co se týče spolehlivosti, dostupnosti, robustnosti, výkonnosti. Typická je také potřeba obsloužit současně velké množství požadavků (klientů). Klasickými zástupci enterprise aplikací jsou moderní webové aplikace a řešení postavená na servisně orientované architektuře (SOA).

Většina aplikačních serverů je vyvíjena v programovacím jazyce Java. Není to jenom tím, že Java je nezávislá na platformě, ale také tím, že existuje standard pro implementaci enterprise aplikací právě v jazyce Java. Tento standard se nazývá Java Enterprise Edition (JEE). Pokud bychom hledali paralelu se světem operačních systémů, mohli bychom zmínit standard POSIX.[wsroot]

Implementací JEE existuje celá řada:

Open-Source:

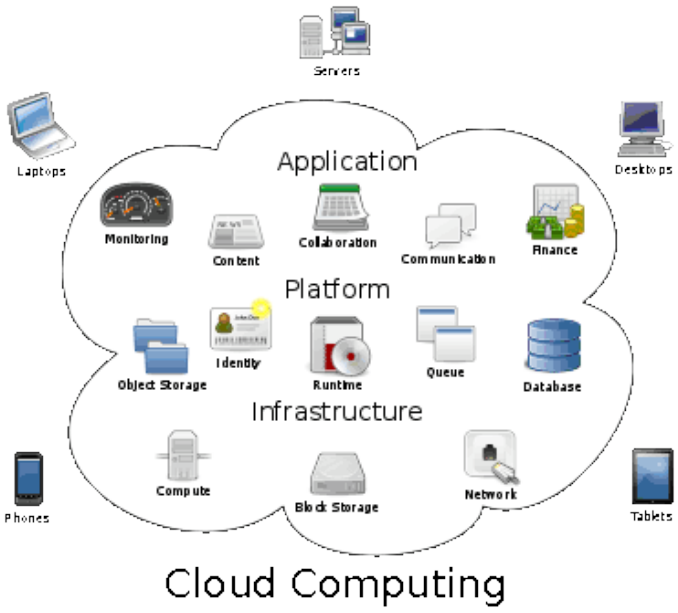
- Apache Tomcat
- Blazix
- GlassFish
- Geronimo
- Enhydra
- Jetty
- JOnAS
- JBoss a další

Placené:

- IBM WebSphere Application Server
- Oracle WebLogic Server

2.4.2 Cloud

Cloud computing je na Internetu založený model vývoje a používaní počítačových technologií. Lze ho také charakterizovat jako poskytování služeb či programů uložených na serverech na Internetu s tím, že uživatelé k nim mohou přistupovat například pomocí webového prohlížeče nebo klienta dané aplikace a používat je prakticky odkudkoliv.



Obr. 10 Cloud Computing

Technologie Cloud computingu se vyznačuje následujícími atributy:

- Multitenancy - tento pojem lze volně přeložit jako "více nájmů". Jedná se o to, že počítačové zdroje jsou sdílené mezi všemi uživateli.
- Obrovská škálovatelnost a elasticita - umožní uživatelům rychle změnit výpočetní zdroje dle potřeby.
- Pay as you go - tento přístup je založen na principu kolik toho uživatel spotřebuje, tolik zaplatí.
- Aktualizovanost (Up-to-date) - všechen software je automaticky aktualizovaný, uživatel nemusí do tohoto procesu nijak zasahovat, vše zařídí poskytovatel.
- Přístup přes internet - uživatelé se mohou ke svému softwaru připojit kdekoliv po celém světě.

Cloud computing dělíme podle toho, **jak je poskytován** a podle **služby kterou poskytuje**.[cloud]

2.4.2.1 Model nasazení

Tento model nám říká, jak je cloud poskytován:

- Veřejný (Public cloud computing) - Někdy je označován jako klasický model cloud computingu. Jedná se o model, kdy je poskytnuta a nabídnuta široké veřejnosti výpočetní služba např. Skype.
- Soukromý (Private cloud computing) - Oblak je v tomto případě provozován pouze pro organizaci a to buď organizací samotnou, nebo třetí stranou.
- Hybridní (Hybrid cloud computing) - Hybridní cloudy kombinují jak veřejné tak soukromé cloudy. Navenek vystupují jako jeden cloud, ale jsou propojeny pomocí standardizačních technologií.
- Komunitní (Comunity cloud computing) - Jedná se o model, kdy je infrastruktura cloudu sdílena mezi několika organizacemi, skupinou lidí, kteří ji využívají. Tyto organizace může spojovat bezpečnostní politika, stejný obor zájmu.[cloud]

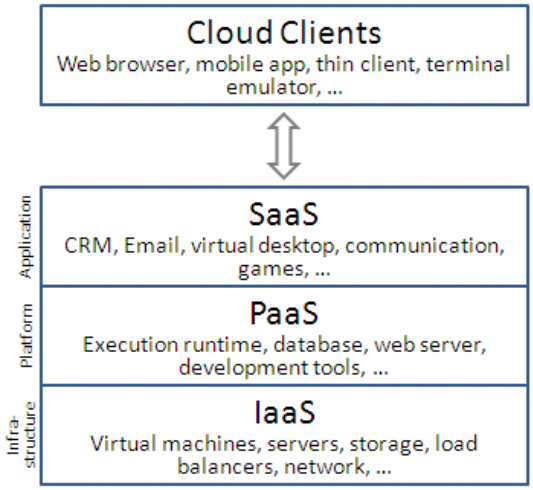
2.4.2.2 Distribuční model

Tento model se zabývá tím, co je v rámci služby nabízeno, obvykle software nebo hardware či jejich kombinace.

- **IAAS** - infrastruktura jako služba ("Infrastructure as a Service") — v tomto případě se poskytovatel služeb zavazuje poskytnout infrastrukturu. Typicky se jedná o virtualizaci. Hlavní výhodou tohoto přístupu je to, že se o veškeré problémy s hardwarem stará poskytovatel. Na druhou stranu je někdy velice těžké toto akceptovat vzhledem k tomu, že hardware se bere jako něco, co vlastníme, na co můžeme sáhnout a jsme za to zodpovědní. IAAS je vhodné pro ty, kteří vlastní software (či jejich licence) a nechtějí se starat o hardware. Příkladem IAAS jsou Amazon WS, Rackspace nebo Windows Azure.

- **PAAS** - platforma jako služba (z "Platform as a Service") — poskytovatel v modelu PAAS poskytuje kompletní prostředky pro podporu celého životního cyklu tvorby a poskytování webových aplikací a služeb plně k dispozici na Internetu, bez možnosti stažení softwaru. To zahrnuje různé prostředky pro vývoj aplikace jako IDE nebo API, ale také např. pro údržbu. Nevýhodou tohoto přístupu je proprietární uzamčení, kdy může každý poskytovatel používat např. jiný programovací jazyk. Příkladem poskytovatelů PAAS jsou Google App Engine nebo VMforce.

- **SAAS** - software jako služba (ze "Software as a Service") - aplikace je licencována jako služba pronajímaná uživateli. Uživatelé si tedy kupují přístup k aplikaci, ne aplikaci samotnou. SaaS je ideální pro ty, kteří potřebují jen běžné aplikační software a požadují přístup odkudkoliv a kdykoliv. Příkladem může být známá sada aplikací Google Apps, nebo v logistice známý systém Cargopass.[cloud]



Obr 11. Dělení cloudu

2.4.2.3 Google App Engine

Google App Engine (často také GAE, App Engine, GAE/J) je PaaS na cloudová platforma pro vývoj a hostování webových služeb(aplikací). Aplikace běží v sandboxu a běží napříč servery Googlu. Nabízí tak automatickou škálovatelnost aplikací, tak jak vzrůstá počet žádostí na aplikaci. GAE automaticky přidělí více prostředků a obstará tak narůstající požadavky.[gae]

Google App Engine je zdarma do určitého množství požadavků. Platí se tedy až tehdy když je potřeba více uloženého prostoru, přenosového pásma či výpočetního výkonu. Jedná se o poměrně novou platformu nasazenou v roce 2011.

Hlavní výhody Google App Engine:

- Vysoká dostupnost
- Výkon serverů dle požadavků aplikace
- Žádná investice do hardwaru
- Snadný vývoj a rozšiřitelnost
- Neomezený počet uživatelů a dat aplikace
- Vysoká bezpečnost a ochrana dat i aplikace samotné

API List:

Trusted Tester
SSL access on custom domains
Full Text Search API
Monitoring API

Experimental
Conversion API (Python, Java)
Google Cloud Storage API (Python)
Files API (Python, Java, Go)
Mapreduce API (Python)
Prospective Search API (Python, Java)
ProtoRPC API (Python)
Task Queue REST API (Python, Java)
OAuth API (Python, Java)
OpenID (Python, Java)
App Identity API (Python, Java)

Production
Blobstore API (Python, Java, Go)
Capabilities API (Python, Java, Go)
Channel API (Python, Java, Go)
Datastore API (Python, Java, Go)
Datastore Async API (Python, Java)
Images API (Python, Java)
Log Service API (Python)
Mail API (Python, Java, Go)
Memcache API (Python, Java, Go)
Multitenancy API (Python, Java)
Remote API (Java)
Task Queue API(Python, Java, Go)
URLFetch API (Python, Java, Go)
Users API (Python, Java, Go)
XMPP API (Python, Java)

2.4.2.4 VMforce

VMforce podobně jako GAE poskytuje PaaS na cloudové platformě. VMforce umožňuje vývojářům aplikací Java trvale se napojit na služby platformy Force.com včetně databáze Force.com, spolupracovat v rámci programu Chatter, účastnit se pracovních postupů, analýz a vyhledávání. VMforce zahrnuje novou technologii VMware vCloud, která dramaticky usnadňuje správu a sladění činností aplikací na bázi infrastruktury VMware vSphere™. Část cloudové platformy Force.com umožní vývojářům aplikací Java snadno a rychle budovat další generaci podnikových aplikací Cloud 2, které jsou sociální, mobilní a schopné spolupracovat.[lupa]

Hlavní přednosti VMforce:

- Důvěryhodnost: Protože je technologie provozována na bázi důvěryhodné celosvětové infrastruktury salesforce.com, platforma VMforce dědí výhody infrastruktury pro dodávání služeb, které se dostalo jedněch z nejpřísnějších bezpečnostních akreditací v oboru včetně ISO 27001, SysTrust a SAS70 Type II.
- Otevřenost: Platforma VMforce bude podporuje standardní kód Java: prosté staré objekty Java (POJOs), stránky Java Server (JSPs), Java Servlets a mnohé další pomocí oblíbeného rozhraní Spring.
- Pružnost: S pomocí VMforce je možno aplikace automaticky přizpůsobovat. Zákazníci VMforce se nemusí dále obávat rozšiřování aplikačních serverů, databází nebo infrastruktury, aby vyhověli požadovanému výkonu. VMforce je schopna automaticky řešit rozšiřitelnost jako službu.
- Úplnost: VMforce poskytuje vyčerpávající řešení pro vývoj podnikových aplikací Java v cloudu včetně důvěryhodné celosvětové infrastruktury, virtualizační platformy, sladění a řídící technologie, relační databáze cloudu, vývojové platformy a služeb spolupráce, času zpracování aplikace, vývojového rámce, nástrojů a dalších věcí.
- Cloud 2: Cloud 2 v sobě spojuje mobilitu další generace, přátelskou spolupráci a přístup k informacím v reálném čase a vytváří tak nové úrovně produktivity a vyzdvihuje podnikovou technologii cloud computing na vyšší úroveň.[lupa]

2.5 Databáze

Nutnost data ukládat mimo operační paměť je zřejmým požadavkem, každé dlouhodobě běžící aplikace. Existují dva hlavní přístupy k databázím a oba se pokusím zde přiblížit.

2.5.1 Relační databáze

Relační databáze je databáze založená na relačním modelu. Často se tímto pojmem označuje nejen databáze samotná, ale i její konkrétní softwarové řešení. Relační databáze je založena na tabulkách, jejichž řádky obvykle chápeme jako záznamy a eventuálně některé sloupce v nich (tzv. cizí klíče) chápeme tak, že uchovávají informace o relacích mezi jednotlivými záznamy v matematickém slova smyslu.[rdb]

2.5.1.1 JDBC

JDBC API poskytuje základní rozhraní pro unifikovaný přístup k databázím. Základem konceptu JDBC je využití funkčnosti poskytované JDBC ovladačem, který je následně překládá do nativních volání dané databáze. Díky tomu je aplikační programátor odstíněn od specifického API databáze a může se naučit jednotné rozhraní JDBC, které pak použije pro přístup do libovolné databáze, která poskytuje JDBC ovladač. V dnešní době to jsou prakticky všechny hlavní systémy a ovladače jsou optimalizované a vyvíjené samotnými výrobci databázových strojů.[interval]

2.5.2 Objektově relační mapování - ORM

Objektově relační mapování (ORM, O/RM nebo O/R mapování) je programovací technika v softwarovém inženýrství, která zajišťuje automatickou konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem.[orm]

2.5.2.1 JPA

Java Persistence API (JPA) je framework programovacího jazyka Java, který umožňuje objektově relační mapování (ORM). To usnadňuje práci s ukládáním objektů do databáze a naopak. Je určen jak pro Java SE, tak pro Java EE.[jpa]

2.5.3.2 JDO

JDO (Java Data Objects) je standardizované aplikační rozhraní pro persistenci objektů v Javě, vyvinuté přímo firmou Sun Microsystems, která je i původcem Javy jako takové. JDO umožňuje implementovat libovolný způsob ukládání objektů - může sloužit jako aplikační rozhraní objektové databáze, ale stejně tak může provádět mapování objektů na relační tabulky a ukládat data do relační databáze. Podstatné ovšem je, že aplikační rozhraní je zcela nezávislé na vlastní implementaci persistence, a proto aplikace využívající JDO mohou bez větších změn ukládat data do různých relačních i objektových databází.[jdointerval]

2.5.4.3 Hibernate

Hibernate je framework napsaný v jazyce Java, který umožňuje tzv. objektově-relační mapování (ORM). Usnadňuje řešení otázky zachování dat z objektů i po ukončení běhu aplikace. Provádí podobné věci jako např. JPA – Java Persistence API.[hibernate]

2.6 GUI

2.6.1 GWT

Myšlenka Google Web Toolkit (zkráceně GWT) je jednoduchá. Poskytněme framework, ve kterém si uživatel bude moci jednoduše vytvořit webovou aplikaci složenou ze znovupoužitelných UI komponent (renderovaných jako HTML) a logiky (ve formě JavaScriptu) na straně webového prohlížeče a asynchronního volání serverové logiky skrze AJAX. To vše na platformě Java s využitím jazyka, knihoven a vývojových nástrojů této platformy.[gwtinterval]

2.6.2 Vaadin

vaadin Vaadin je softwarový framework pro tvorbu webových aplikací. Tyto aplikace se v internetovém prohlížeči zobrazují a chovají stejně jako desktopové aplikace (RIA). Kód je psán v Javě, který je za pomocí GWT překládán do JavaScriptu a ten je následně interpretován v internetovém prohlížeči.[vaadin]

2.7 Formáty pro přenos dat

Dva nejvýznamnější formáty pro přenos dat mezi webovými klienty a webovými službami jsou JSON využívaný ve spojitosti s REST a XML využívaný společně se SOAP řešením.

2.7.1 JSON

JavaScript Object Notation (JavaScriptový objektový zápis, JSON) je způsob zápisu dat (datový formát) nezávislý na počítačové platformě, určený pro přenos dat, která mohou být organizována v polích nebo agregována objektech. Vstupem je libovolná datová struktura (číslo, řetězec, boolean, objekt nebo z nich složené pole), výstupem je vždy řetězec. Složitost hierarchie vstupní proměnné není teoreticky nijak neomezena.[json]

2.7.2 XML

Extensible Markup Language (zkráceně XML, česky rozšiřitelný značkovací jazyk) je obecný značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C. Je zjednodušenou podobou staršího jazyka SGML. Umožňuje snadné vytváření konkrétních značkovacích jazyků (tzv. aplikací) pro různé účely a různé typy dat. [xml]

2.8 Obdobné SW - Zabbix, Nagios, Cacti, Nimsoft, DU Meter

Přestože řešení od Nimsoftu a Paessler jsou pěkná, jedná se o placený software a proto jsem jej do srovnání nezahrnul. Nejlepšími obdobnými Open-Source řešeními, jsou Zabbix, Nagios a Cacti. Pro ukázku jednoduchosti a přátelského prostředí jsem vybral DU Meter.

2.8.1 Zabbix

- Oznamování událostí (email, SMS)
- Síťové mapy
- Monitoring míry dostupnosti služeb (SLA)
- Uživatelsky definovatelné grafy
- Centralizovaný monitoring
- Architektura klient-server
- PHP frontend
- Data se ukádají do MySQL | PostgreSQL
- Hodnoty získané pomocí agentů IPMI,SNMP.

2.8.3 Cacti

- Udržuje seznam monitorovaných zařízení, jejich dostupnost a umí upozorňovat na jejich výpadky e-mailem
- Grafy můžeme sdružovat do přehledů a ty pak snadno zobrazovat za libovolné období
- Využívá import a export šablon (xml), export naměřených dat
- Data ukládaná v RRD (Round Robin Database)
- Je možné debugovat RRD příkazy
- Pomocí Cacti si snadno vytvoříme vlastní zdroje dat, šablony grafů, šablony celých zařízení
- Umožňuje nastavit práva pro uživatele, třeba jen pro prohlížení některých grafů
  díky pluginům přidat mnoho nových funkcí
- PHP
- MySQL

2.8.2 Nagios

Nabízí v podstatě ty samé vlastnosti jako Zabbix či Cacti a je postaven na stejném základu. Velkým záporem pro mne představovalo složitější nastavení oproti předchozím řešením.

2.8.4 DU Meter

Slouží pro sledování síťových přenosů a upozorněních při přenesení určitého počtu dat. Síla tohoto programu tkví v jednoduchosti instalace, konfigurace a využívání. Uživatel jej nainstaluje a od té chvíle program začne plnit bez jakéhokoliv dalšího nastavování svou funkci.

3. Realizační část

Tato část obsahuje kritéria výběru a zdůvodnění výběru řešení, jeho obecný popis a rozpracování až do realizačních detailů.

Kritéria výběru:

- Vlastní systém vzdáleného monitorování a správu počítačové sítě
- Periodické sledování stavu vzdálených zařízení
- Ovládání těchto zařízení
- Naměřené hodnoty uložené v databázi
- Přehledné zobrazování dat dle požadavků
- Uživatelsky přátelské prostředí
- Zpracování asynchronních událostí
- Škálovatelnost
- Nízké zatížení zdrojů (sítě, počítačů)
- Jednoduché nasazení, používání a údržba
- Modulární řešení
- Přístup přes mobilní zařízení
- Bezpečnost
- Nezávislost na protokolu sledovaného zařízení

3.1 Návrh architektury a nástrojů pro realizaci

Tato část se zabývá návrhem architektury celého systému. Výběrem protokolů, nástrojů pro jednotlivé části systému a zdůvodnění jejich výběru.

3.1.1 Návrh architektury

Při návrhu architektury byl kladen důraz na malé zatížení systémových a síťových prostředků, škálovatelnost, bezpečnost, modulárnost systému, jeho jednoduché nasazení a údržba.

3.1.1.1 Jedno-vrstvá architektura

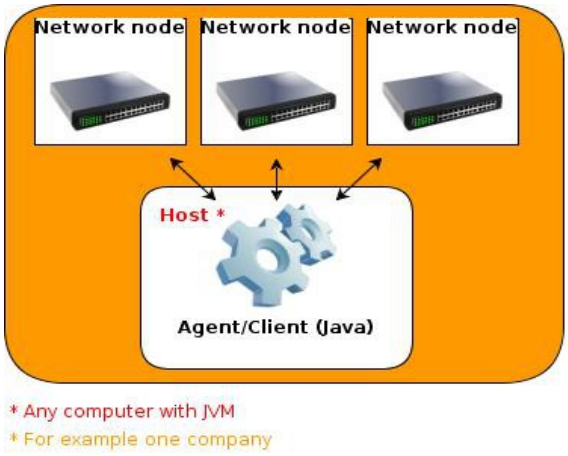
Jedná se o nejjednodušší řešení kdy je celý systém v jediném .jar souboru. Jeho možnosti jsou tak velice omezené a nabízí jen přehlednější zpracování dat z jednotlivých zařízení.

Klady:

+ Jednoduchá instalace (spuštění .jar souboru)

Zápory:

- Ukládání do paměti (chyba, restart či jiná událost způsobí ztrátu dat)
- Nemožnost statistik sítě (sledování začíná ve chvíli spuštění agenta)
- Nulová optimalizace zatížení sítě (Při mnoha prvcích či dosti vzdálenými)
- Velké zatížení hostujícího stroje při sledování více prvků (sběr dat a zobrazování zároveň)
- Obtížené sledování prvků za NATem (přístup, bezpečnost, náročnost na zatížení sítě)
- Žádné zabezpečení
- Neškálovatelné



Obr.12 Jedno-vrstvé řešení

3.1.1.2 Dvou-vrstvá architektura

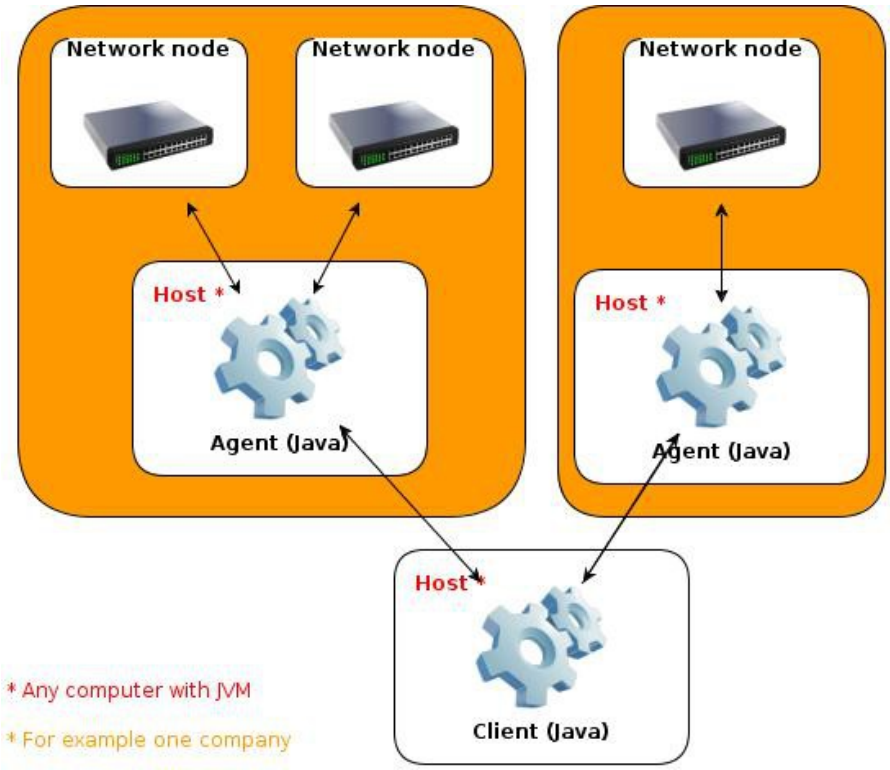
Toto řešení nabízí více možností nežli jednovrstvá architektura, avšak stále převládají zápory nad přínosy. Největším záporem je ztráta všech dat při ukončení systému.

Klady:

+ Možnost optimalizace zatížení sítě (cache u agentů umístěných blízko sledovaných prvků)
+ Možnost rozložení zátěže hostujícího stroje při sledování více prvků (sběr dat a zobrazování zároveň)
+ Možné sledování prvků za NATem (avšak stále nevhodný přístup)
+ Možné škálovat, rozhodně ne však ideálně a přehledně

Zápory:

- Ukládání do paměti (chyba, restart či jiná událost způsobí ztrátu dat)
- Nemožnost statistik sítě (sledování začíná ve chvíli spuštění agenta)
- Nízké zabezpečení pro přístup
- Při větším počtu agentů obtížný management



Obr.13 Dvou-vrstvé řešení

3.1.1.3 Tří-vrstvá architektura (agent - databáze – client)

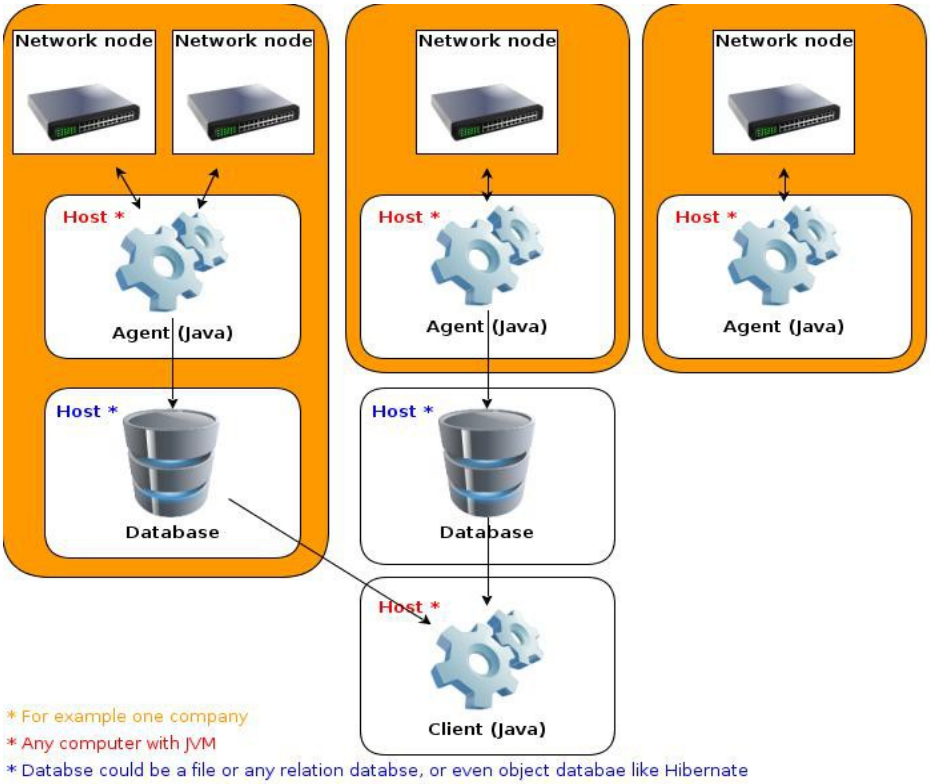
Tato architektura byla navržena a realizována v mé bakalářské práci. Byla v ní odstraněna největší nevýhoda předchozích řešení, tedy ztráta dat při ukončení klienta. Data jsou uložena v databázi a je tak možné je později zobrazovat v grafech a utvářet statistiky.

Klady:

+ Ukládání do databáze(chyba, restart či jiná událost NEzpůsobí ztrátu dat)
+ Možnost statistik sítě (sledování začíná ve chvíli nasazení AGENTŮ)
+ Možnost optimalizace zatížení sítě (cache u agentů umístěných blízko sledovaných prvků, blízko umístěné databáze např. pro jednotlivé pobočky firmy)
+ Možnost rozložení zátěže hostujícího stroje při sledování více prvků (sběr dat a zobrazování zároveň)
+ Možnost rozložení zátěže databáze při sledování velkého množství prvků
+ Možné sledování prvků za NATem (avšak stále neideální přístup)
+ Dobře škálovatelné

Zápory:

- Složitější instalace (JVM, databáze)
- Nízké zabezpečení (především client – databáze)
- Agenti bez správy (Globálně neřízené)



Obr.14 Tří-vrstvé řešení

3.1.1.4 Tří-vrstvá architektura se serverem (agent - server - databaze – client)

Toto řešení nabízí především přístup přes Webový prohlížeč. Avšak k serveru tak musí být přístup z vnější sítě nebo server musí být umístěn mimo lokální síť. Obě možnosti nejsou vhodné.

Klady:

+ Ukládání do databáze(chyba, restart či jiná událost NEzpůsobí ztrátu dat)
+ Možnost statistik sítě (sledování začíná ve chvilí nasazení AGENTŮ)
+ Možnost optimalizace zatížení sítě (cache u agentů umístěných blízko sledovaných prvků, blízko umístěné servery např. pro jednotlivé pobočky firmy)
+ Možnost rozložení zátěže hostujícího stroje při sledování více prvků (sběr dat a zobrazování zároveň)
+ Možnost rozložení zátěže serveru/databáze při sledování velkého množství prvků
+ Možné sledování prvků za NATem (přístup přes WS, bezpečnější, náročnost na zatížení sítě hodně optimalizované)
+ Dobře škálovatelné
+ Řešení je strukturované (jednoduše upravitelné komponenty, výměna agentů za jiné)
+ Přístup přes webové služby (klientem je webový prohlížeč)
+ Agenti jsou spravovány serverem (Globálně řízené obdoba PVM)
+ Rozšiřitelnost (agenti v různých jazycích, různé protokoly sledovaných zařízeních atd.)
+ Dobré zpracování asynchronních událostí (reakce na trapy)

Zápory:

- Složitější instalace (JVM, databáze)
- Dobré zabezpečení (Client nemá přímý přístup k agentům či databázi, avšak stále má přístup ke kriticky důležitému serveru)

Obr.15 Tří-vrstvé řešení se serverem

3.1.1.5 Tří-vrstvá architektura s Backend a Frontend serverem

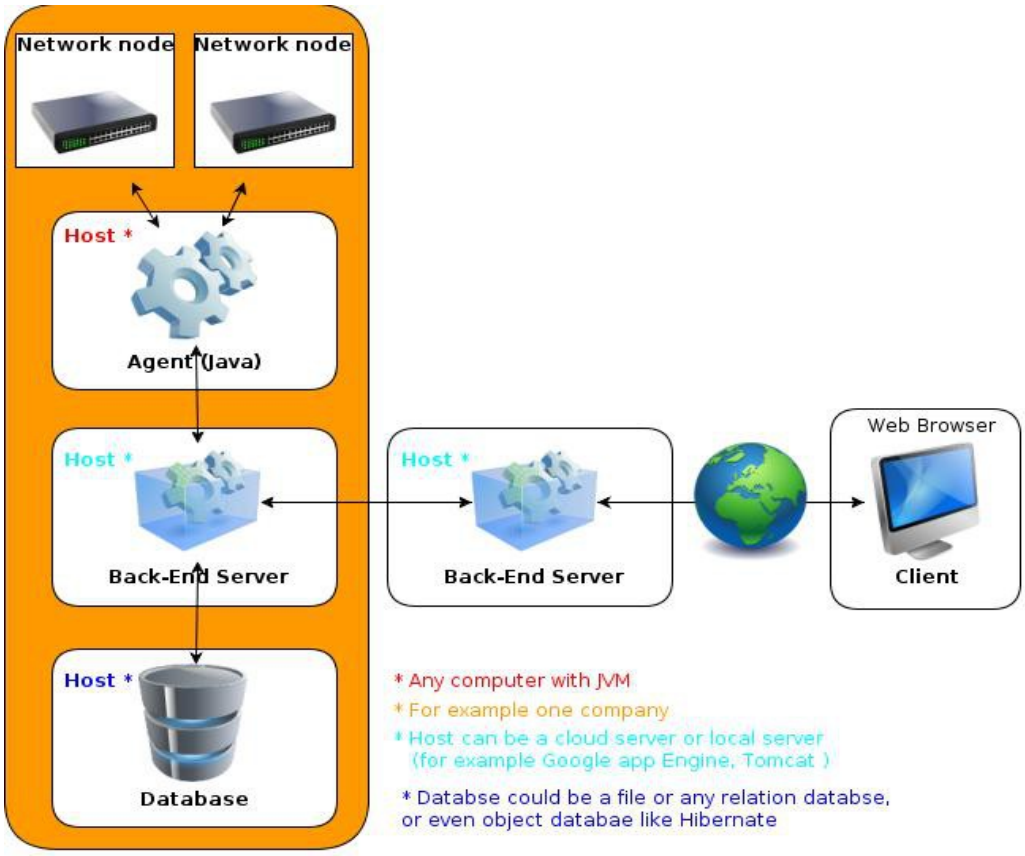
Vylepšením oproti předchozí verzi je rozšíření o Frontendový server, který zakrývá a tedy i chrání Backendový server a umožňuje i zobrazení pro mobilní zařízení. Backendový server tedy může například zůstat uvnitř firmy aby snížil zatížení sítě a zůstal dobře zabezpečený. Frontendový server jen pak předává informace.

Klady:

+ Backend server zabezpečný od okolí
+ Přístup z mobilních zařízení

Zápory:

- Malou nevýhodou oproti předchozímu řešení je opět nárůst náročnosti řešení

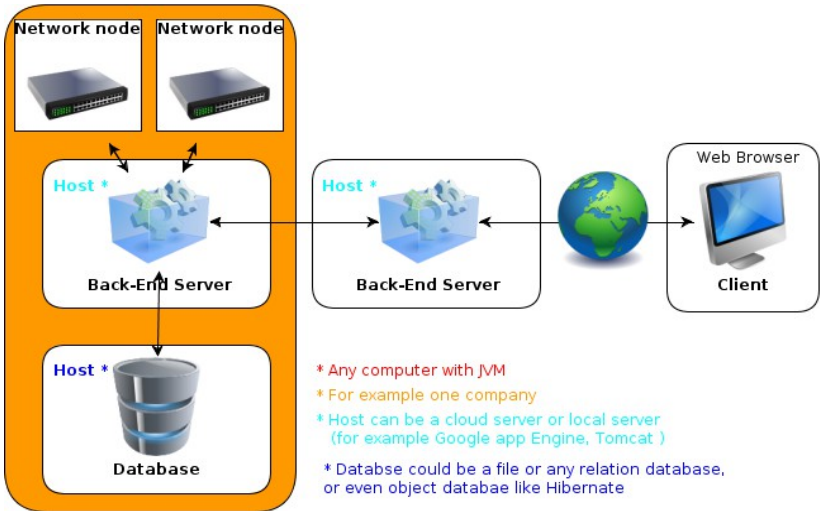


Obr.16 Tří-vrstvé řešení s Backend a Frontend serverem

3.1.1.6 Zavrhnutá řešení

1. Řešení bez agenta (spojit Agenta a Backend server)

Problémem tohoto řešení je ztráta variability, zbytečné zatížení pro hostitele Agenta-Backend serveru, nemožnost mít Backend server umístěný mimo vnitřní síť, navíc v konečném řešení je možno aplikaci takto nakonfigurovat (Agent a Backend server na stejném stroji).



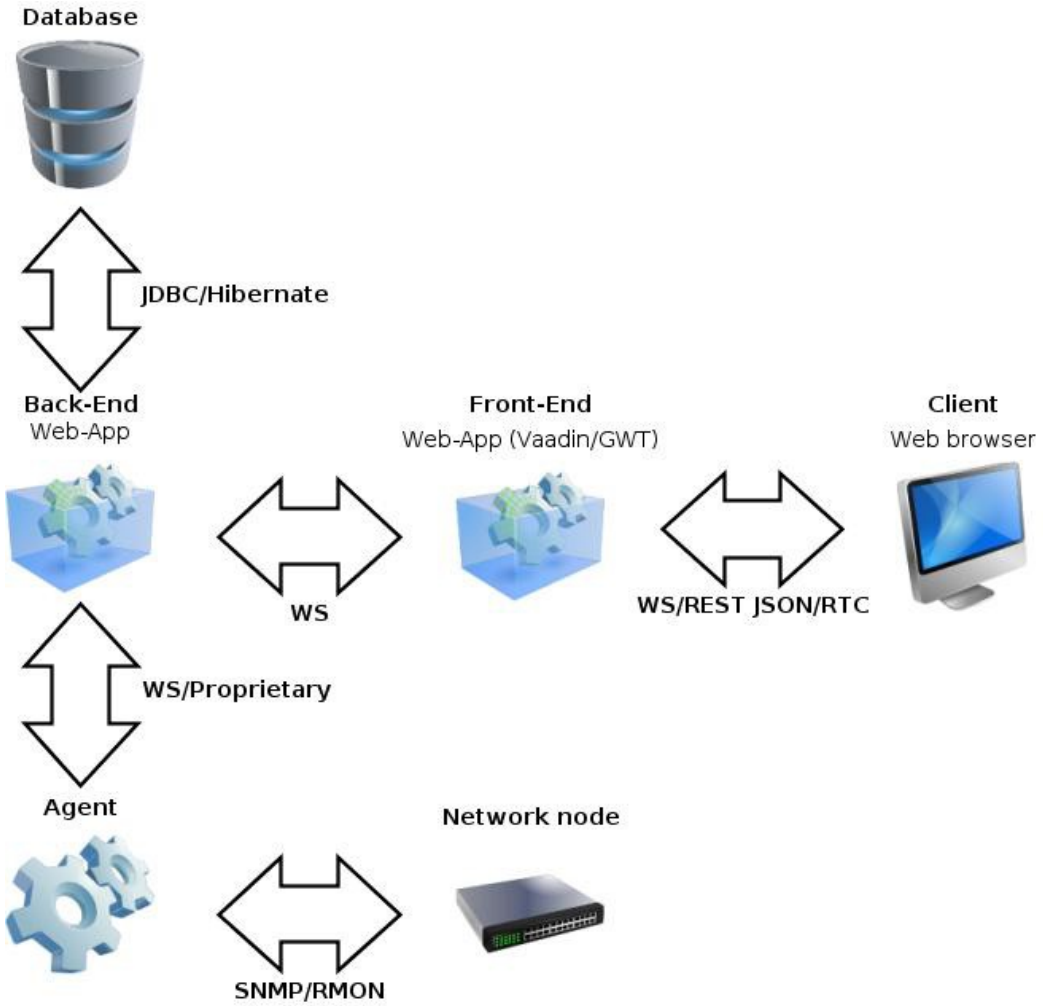
Obr. 17 Zavrhnuté řešení 1.

2. Řešení s komunikací agent - databáze

Přínosem by bylo snížení zatížení sítě v případě, že Backend server je umístěn mimo vnitřní síť. Problémem tohoto řešení je komunikace agenta přímo s databází a s tím i spojená ztráta modularity. Navíc agent musí komunikovat se serverem i s databází.

3.1.2 Výběr protokolů a jednotlivých řešení

Dle analýzy a návrhu architektury vznikl model viz. Obr.17.



Obr.17 Návrhový model

Snahou tohoto modelu je tedy mít dobře zabezpečený, dostupný, spolehlivý a jednoduše škálovatelný Backend server s přístupem přes mobilní zařízení. Tenkého klienta i agenta. Agenta s co nejmenším zatížením na síť, výpočetní a paměťový výkon hostujícího stroje. A Frontend s co nejpřehlednějším a uživatelsky přátelským grafickým rozhraním. Bylo tedy zapotřebí vybrat z možností jednotlivých řešení a jejich implementací.

3.2.2.1 Výběr modelu webového serveru:

Výběr proběhl mezi již zmíněnými RPC, SOAP, REST.

Tradiční RPC

- V rámci jedné firmy
- Volání procedur
- Obvykle daný transportní protokol
- Těsně vázané
- Efektivní zpracování na čas i velikost

SOAP

- Lze i mezi firmami
- Nezávislé na jazyce
- Posílání zpráv (XML)
- Snadno vyměnitelný transportní protokol
- Volně vázané
- Poměrně neefektivní zpracování (XML)

REST

- Lze i mezi firmami
- Nezávislé na jazyce
- Posílání zpráv(XML, Text, JSON)
- Snadno vyměnitelný transportní prot.
- Volně vázané
- Efektivní zpracování na čas i velikost (JSON)
- Jednoduché nasazení a implementace

Při porovnání možných řešení pro webový server byl tedy vybrán REST v kombinaci s přenosovým formátem JSON, čímž by se měla snížit zátěž na přenosy, především při častém vzorkování agenta. Toto řešení navíc splňuje požadavky na modularitu, kdy je možné velmi snadno zaměnit jednu část za jinou a to i pro případ kdy je nová komponenta napsána v jiném jazyce. Je tedy např. možné napsat agenta v jazyce Python, Go atp.

3.2.2.2 Výběr řešení pro Back-End webový server

Požadavky :

- Bezpečnost - pokud by se jednalo o firemní data, bylo by žádoucí, aby byla uložena na vlastním intranetím serveru.
- Dostupnost - přístup k datům zvenčí bez narušení bezpečnosti vnitřní sítě
- Zatížení sítě - pokud budou oba servery Frontend i Backend mimo lokální síť je možné znatelné zatížení sítě
- Cachování (spojení agentů - ven jeden požadavek, ale nutnost lokálního serveru)

Back-end server tedy musí být nasaditelný jak na vnitřním webovém serveru tak může být nasazen i na cloudovém řešení. V rámci dp. práce jsem se rozhodl že řešení bude realizováno pouze na Google App Engine s možností následného přesunu na lokální webový server (např. Tomcat či privátní cloud).

GAE byl vybrán z důvodů:

- Rychlý vývoj a nasazení
- Jednoduchou a přehlednou administraci
- Žádné starosti o hardware, patche, zálohy
- Téměř neomezená škálovatelnost
- Nejomezenější množinou pro použití Javy má právě GAE. Pokud by tedy někdy v budoucnu byl zvažován přechod na jinou platformu neměl by s sebou přinést mnoho úskalí
- Podporuje i více jazyků pokud by se zvažovalo o změně některé z komponent
- Podporuje JPA/JDO/JDBC
- RESTful API a mnoho dalších
- Vše výše zmíněné zdarma (do určitého množství zdrojů pak je možné zaplatit za zvýšení limitů)

3.2.2.3 Výběr Front-End serveru

Výběr proběhl mezi Vaadin(popř. Ext JS) a GWT (popř. extGWT). Hlavním kritériem byla přenositelnost a samozřejmě grafické možnosti. Vaadin není nasaditelný jen na GAE a nabízí velké množství rozšířených vlastností a doplňků. Mezi doplňky je např. Add-in Timeline umožňující vytváření velmi pěkných a přehledných grafů s možností je v určitých momentech anotovat.

3.2.2.4 Výběr databázového API

Jelikož byl jako Backend server vybrán Google App Engine, bylo možné vybrat mezi Google Datastore, který má přístup přes JPA/JDO, nebo Google Cloud SQL čím by byl přístup řešen přes JDBC/DB-API.

V první fázi se tedy jednalo o výběr mezi relačním a objektovým přístupem k databázi. Největší výhodou relační databáze je její rychlost zatímco objektový přístup nabízí rychlejší implementaci a nezávislost aplikace na konkrétním databázovém systému. Z těchto důvodů byl zvolen ORM přístup.

Ve druhé fázi výběru přístupu k databázi bylo tedy zapotřebí se rozhodnout mezi JPA/JDO mapováním. Použitím JPA API je možné provést nahrazení databáze s minimálními nároky na pracnost a to i při přechodu na relační databázi. Protože např. Hibernate je implementací JPA, je i tento přechod možný. Dalším důvodem je využívání jinými možnými řešeními jako je např. VMforce který taktéž používá JPA. Zvolil jsem tedy JPA 1.0, jelikož JPA 2.0 je stále ještě ve fázi „experimental“.

3.2.2.5 Výběr implementace formátu komunikace

Přestože GAE API nabízí RESTful API jedná se o “experimentální” verzi ve, které se mi nepodařilo najít podporu pro JSON formát.

Výběr tedy proběhl nad nejvíce rozšířenými implementacemi REST API:

- Restlet
- Resteasy
- Jersey

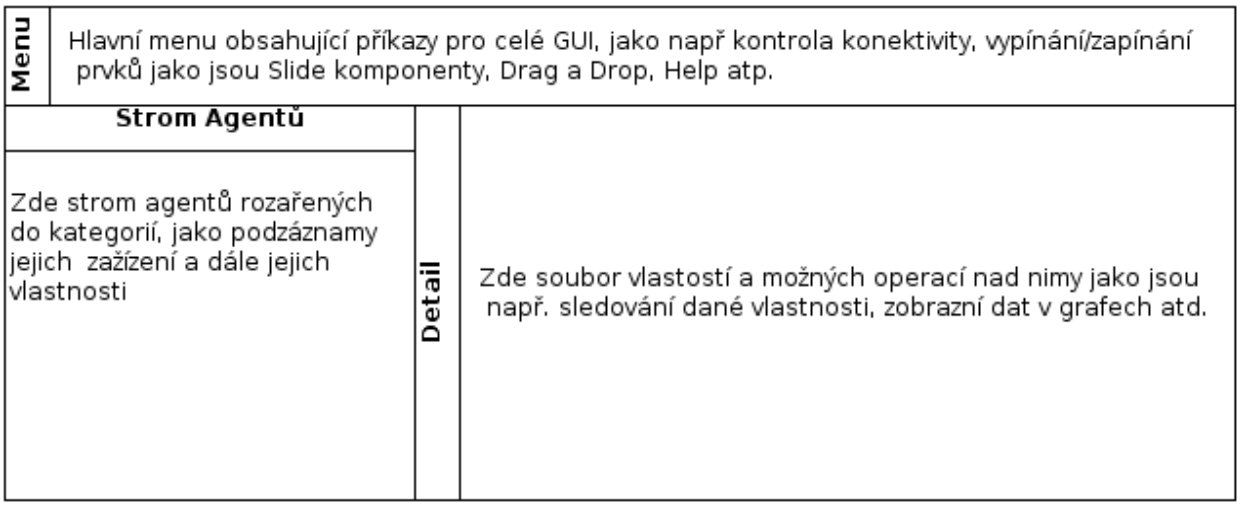
Nejlepší implementací je dle mého názoru Jersey. Především svým širokým rozšířením a implementovanou klientskou částí.

3.2.2.6 Výběr verze programovacího jazyka Java pro agenta

Přestože je JDK 7 již několik měsíců vydána většina vybraných frameworků ji bohužel zatím nepodporuje (nebo jen jako experimentální verzi) a také je zřejmé, že jako nově vydaná verze s sebou nese vysoké riziko chyb. Navíc pokud má Oracle dle plánu “B” již zanedlouho vydat JDK 8, usoudil jsem že vývoj na lety ověřené a stabilní verzi JDK 6 s pozdějším přechodem na JDK 8 je pro tuto práci vhodnější. Navíc JDK 7 by pro tuto práci nebylo žádným převratným přínosem.

3.2.2.7 Návrh GUI

Při návrhu GUI jsem se snažil co nejvíce o přehlednost a přívětivost k uživateli, což měla především zajistit přehledná tabulko-stromová struktura v levé části.



Obr 12. Návrh GUI

3.2.2.8 Výběr protokolu sledovaného zařízení

Snahou byl výběr protokolu, který se používá v praxi a je rozšířený do všech nebo alespoň do většiny zařízení a je schopen nabídnout co nejvíce údajů.

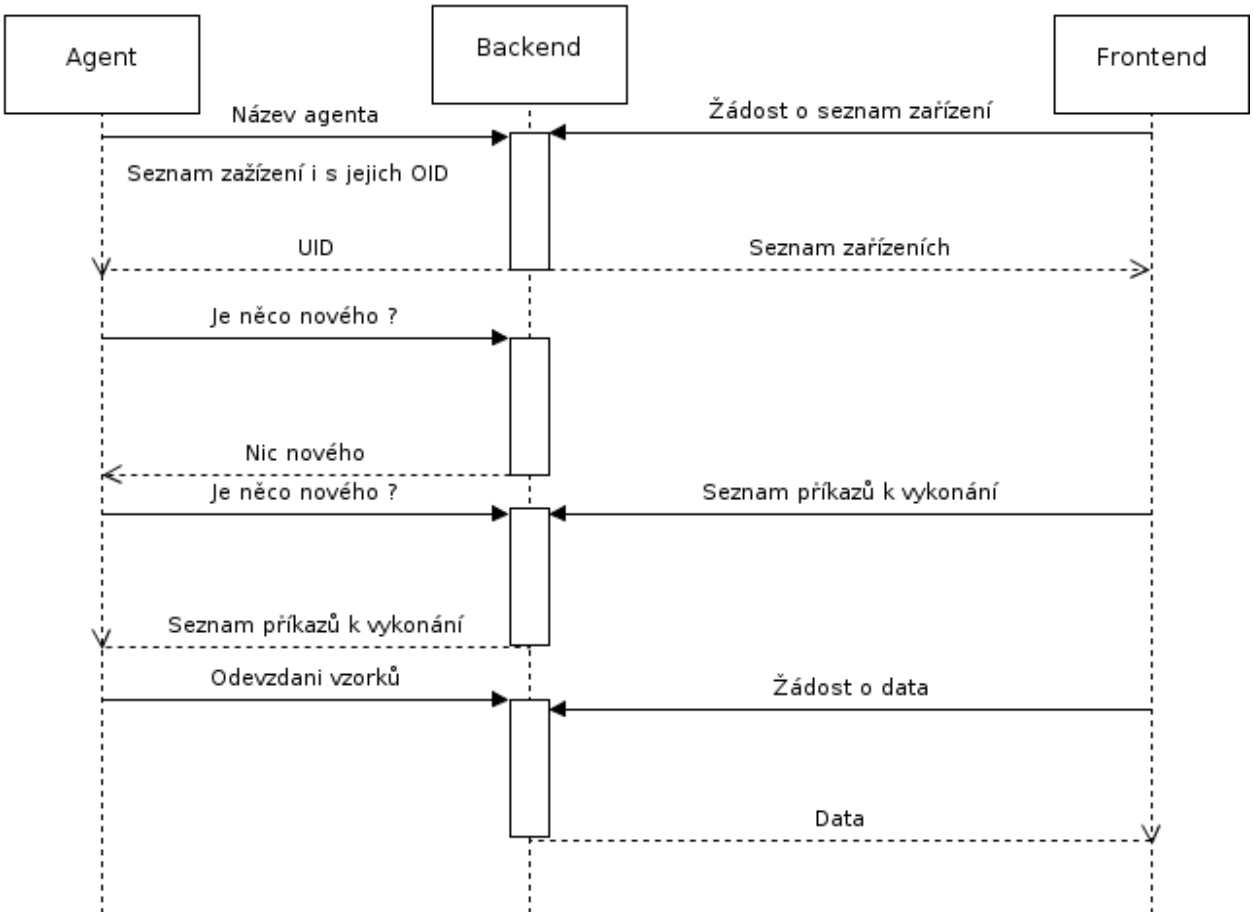
Výběr tedy proběhl nad těmito protokoly:

- SNMP - je ideální protokol rozšířený do téměř všech zařízení.
- RMON - je sice pokročilejší, avšak nerozšířený. (drahá zařízení)
- ICMP - nabízí je základní možnosti pro sledování
- IPMI - zatím bohužel jsem pro servery
- Proprietární - pro sledování pc využít knihovny Java.

Nerozhodl jsem se tedy pro jeden konkrétní protokol ale zkusil jsem vytvořit agenta nezávisle na využívaném protokolu pro sledování zařízení.

3.2.2.9 Návrh komunikace Agent-BackEnd-FrontEnd

Komunikace bude fungovat na klasickém HTTP Request/Response. Následující obrázek znázorňuje fungování pro případ kdy je agent nakonfigurován pro automatické zjištění zařízeních v okolí.



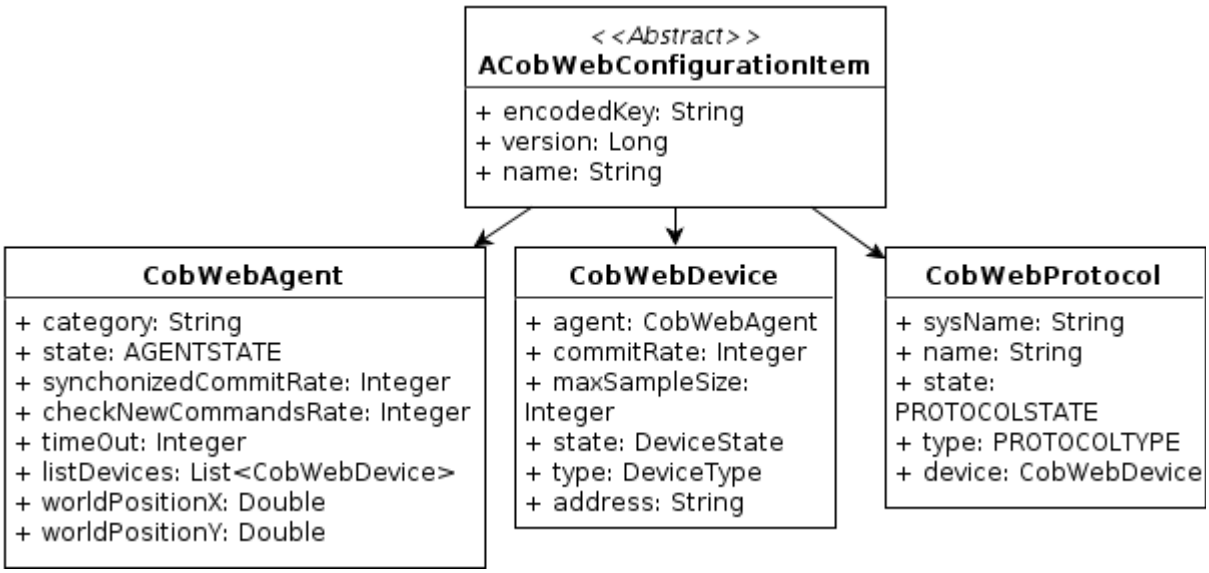
Obr. 18 Sekvenční graf popisující průběh komunikace

3.2.2.10 Návrh entit

Entity jsou rozděleny do takzvaných EG (Entity Groups). Jedná se o soubor entit vzájemně spojených. Při operaci jako je mazání, přidání či selekce se dle klíče nejvyšší entity provedou operace nad celou touto entity grupou.

3.2.2.10.1 Entity group – Agent

Tato EG je „srdcem“ aplikace. Předávají se v ní informace co se má provádět, v jakých intervalech a jaký je aktuální stav.



ACobWebConfigurationItem

- **key** – každá entita má svůj unikátní klíč
- **version** – je potřeba zjišťovat zda-li se konfigurace změnila, resp zda-li vlastněná entita je aktuální.
- **name** – každou instanci entit je možné si pojmenovat

CobWebAgent

- **category** – do jaké uživatelem definované skupiny patří. Popř. Automaticky spadá do kategori „nezařazení“.
- **state** – určuje stav agenta, zda-li je běžící, pozastaven nebo se má ukončit.
- **synchronizetCommitRate** – tato vlastnost určuje zda-li se nasbírané vzorky ukládají synchonizovaně, tedy po uběhnutí zadaného času.
- **checkNewCommandsRate** – jako často si má agent kontrolovat nové příkazy
- **timeOut** – po jak dlouhé době se má agent sám ukončit pokud nedostal odpověď od serveru.
- **listDevices** – seznam zařízeních tohoto agenta.

CobWebDevice

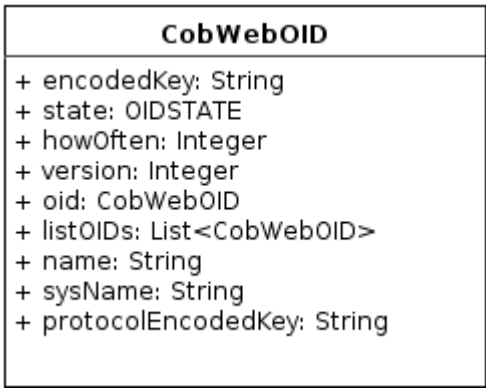
- **agent** – odkaz na agenta ke kterému zařízení patří.
- **commitRate** – jak často se mají vzorky odesílat na server.
- **maxSampleSize** – maximální množství udržovaných vzorků.
- **state** – určuje stav zařízení.
- **type** – typ sledovaného protokolu zařízení.

CobWebProtocol

- **name** – Jedná se o název určený uživatelem
- **state** – opět určuje stav sledované vlastnosti (zdali se má sledovat)
- **type** – určuje o jaký typ protokolu se jedná
- **device** – odkaz na zařízení ke kterému tato vlastnost patří

3.2.2.10.1 Entity group – OID (Vlastosti)

Tato oddělená EG vznikla především, protože datastore GAE nepodporuje vícenásobné rodiče (tato entita by musela ukazovat na protokol a na sebe sama jelikož se jedná o stromovou strukturu vlastností). Navíc takto oddělené vlastnosti od konfiguračních položek nabízí možnosti Lazy Loadingu a tedy další úspory na zatížení sítě. Jedná se tedy jen o jeden typ entity nesoucí údaje a seznam podřízených vlastností. Hlavičku těchto EG tvoří vždy entita nesoucí název „root“, klíč na rodičovský protokol (pro specifické zařízení i agenta) a seznam potomků sebe sama (tedy jednotlivých vlastností).



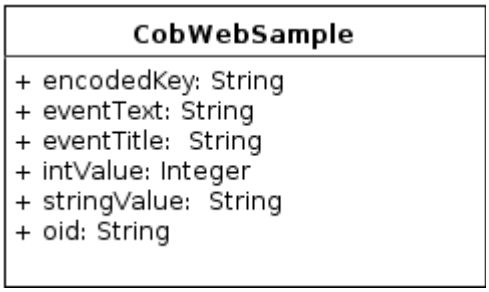
CobWebOID

- encodedKey – Unikátní klíč
- state – Aktuální stav vlastnosti (RUN,STOP – určuje se tak zda-li je vlastnost sledována či ne
- howOften – Určuje jak často se má tato vlastnost sledovat
- version – Verze Entity
- oid – Odkaz na rodiče
- listOIDs – Odkazy na potomky

- name – Jméno zvolené uživatelem
- sysName – Systémová jméno vlastnosti
- protocolEncodedKey – Slouží u vlastnosti „root“ jako odkaz na protokol ke kterému patří

3.2.2.10.1 Entity group – Sample

Tato EG je velmi prostá. Jedná se tedy jen o jednu entitu nesoucí navzorkované údaje a klíč vlastnosti ke které patří.

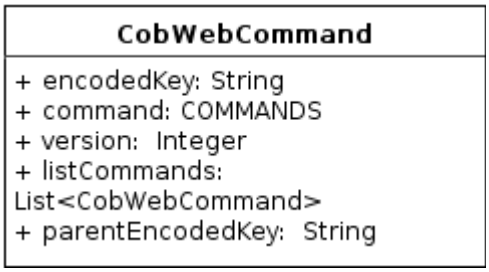


CobWebSample

- encodedKey – Unikátní klíč
- eventText – Pokud došlo k události uloží se zde její text
- eventTitle - Pokud došlo k události uloží se zde její název
- intValue – Číselná hodnota vzorku
- stringValue – Textová hodnota vzorku
- oid – Klíč určující ke které vlastnosti tento vzorek patří

3.2.2.10.1 Entity group - Commands

Tato EG přenáší seznam příkazů které se mají na agent či zařízení vykonat.



CobWebCommand

- encodedKey – Unikátní klíč
- Command – Příkaz k vykonání typu COMMANDS
- ListCommands – pokud se jedná o „root“ entitu
- ParentEncodedKey – pokud se jedná o „root“ entitu je zde uložen klíč pro koho se mají příkazy spustit

3.3 Realizace

Jako vývojové IDE prostředí jsem zvolil Eclipse. Důvodem byla velmi silná podpora přímo od GAE, tedy distribuce Eclipse Indigo s GAE SDK. Jako pracovní název projektu jsem zvolil Cobweb (pavučina).

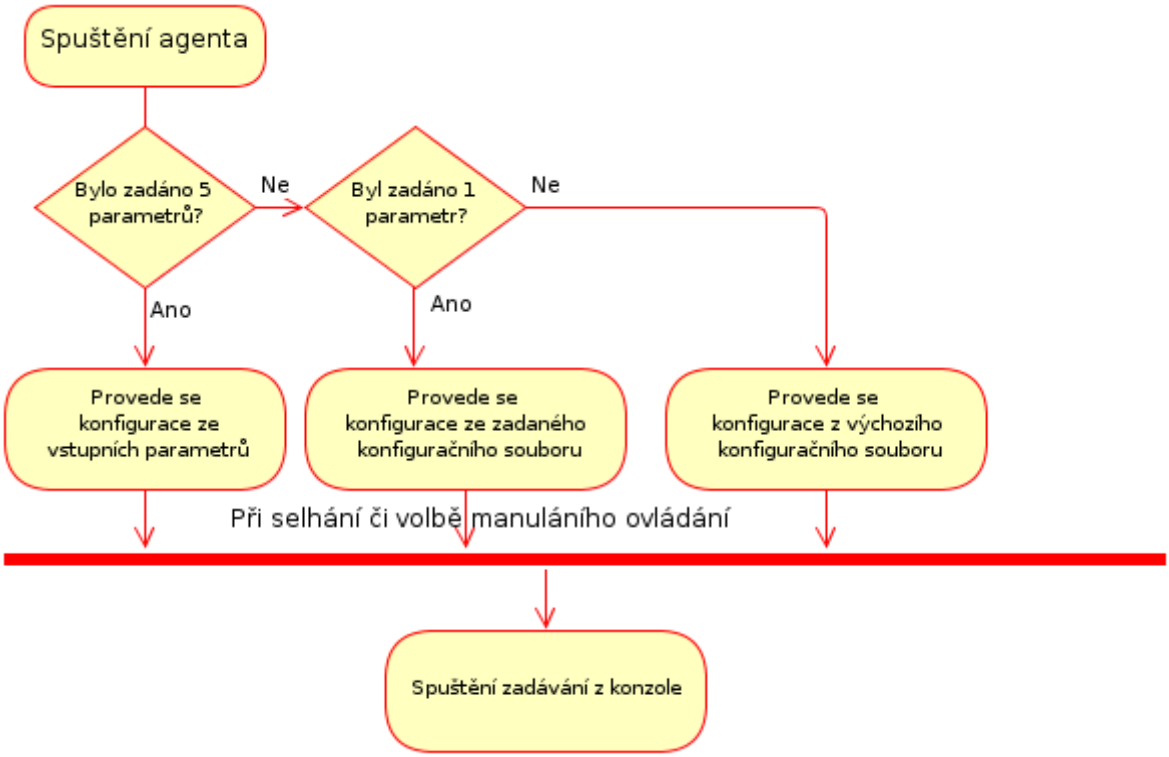
Projekty jsem rozdělil tak jak je znázorněno v sekvenčním grafu viz. Obr 18. Tedy Agent, Backend server , Frontend server a navíc projekt s entitami Shared. Datové struktury využívané v aplikaci jsou Listy. Jejich důvodem pro použití bylo prolnutí množiny podporovaných datových struktur Jersey a GAE. Všechny projekty obsahují balíček .experimental zde se nacházejí pomocné třídy pro vývoj a ladění aplikace. Pro verzování zdrojových kodů jsem využil Git a pro tvorbu diagramů Diagramly.

3.3.1 Agent

Při konstrukci agenta jsem dbal především na snadnou instalaci a konfiguraci. Také byl kladen důraz na zatížení sítě a hostujícího stoje, z toho důvodu jsem se rozhodl udělat agenta jen CLI.

Agenta je možné nakonfigurovat třemi způsoby:

- XML souborem (výchozí konfigurační soubor - agentconfiguration.xml)
- Vstupními parametry
- či přímým zadáním do konzole

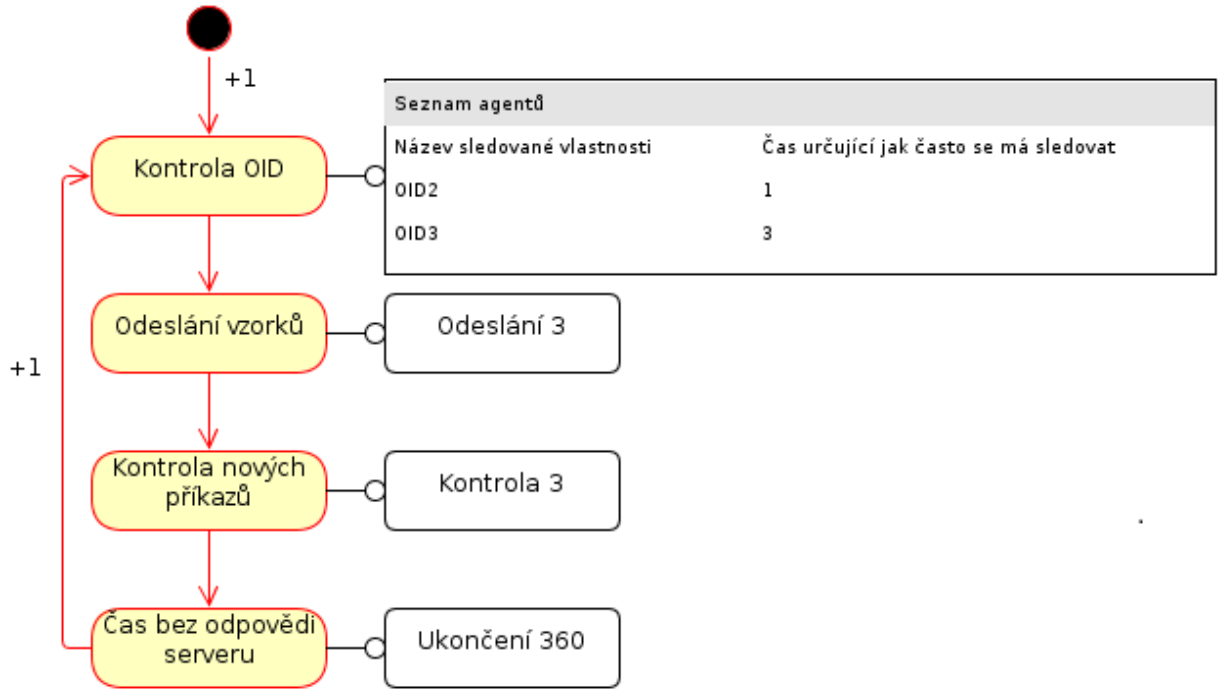


Obr. 13 Konfigurace agenta při prvním spuštění

Agent si vždy po zvolené době (pokud je zvolen automatický běh) kontroluje zda-li pro něj nejsou na serveru nové příkazy k vykonání. Kontrolují se tedy verze agentovo entit (agent, device, oid) oproti serverovým a v případě změny agent provede překonfigurování. Agent v pravidelných intervalech (interval je určený nejnižší hodnotou z hodnot pro kontrolu vlastností, konfigurace nebo odeslání vzorků viz. Obr.14 kdy je čas obrátky 1s) provádí operace:

- Kontrola a sběr informací ze zařízeních v seznamu
- Odeslání nasbíraných vzorků
- Kontrola nových příkazů
- a kontrolu zdali již není odpojen od serveru moc dlouho (v tom případě se sám ukončí)

Jednotlivé akce se provedou vždy pokud se naplní hodnota dané vlastnosti. Vhodným řešením je míti stejnou dobu synchronizovaného odesílání vzorků jako kontrolu nových příkazů. Při odeslání vzorků se čítač pro ukončení nuluje. Při ukončení sebe sama (time out) ještě agent odešle zprávu o ukončení serveru.



Obr. 14 Běh agenta

Snahou bylo tedy vytvořit co nejméně náročného tenkého agenta, aby byla popř. možná jednoduchá výměna za jiného.

Agent byl rozdělen do následujících balíčků:

- **zcu.cobweb.agent**
  - Main – Vstupní bod aplikace
  - CobWebAgentManager – Třída obstarávající vyhodnocení vstupních argumentů a jejich další dělení.
- **zcu.cobweb.agent.configuration**
  - AgentSettings – Slouží k udržování informací o nastavení
  - ConfigurationXMLHandler – Třída slouží k načítání a ukládání do XML souboru. Pokud je agent spuštěn podruhé z XML je tak možné pokračovat v běhu tam kde přestal předtím. Dle enccodedKey si dotáhne nejnovější konfigurační údaje pro daného agenta a nakonfiguruje se.
- **zcu.cobweb.agent.protocols.dal**
  - CobWebDeviceManager – Třída zajišťující všechny operace nad zařízeními.
  - SNMPHelper – Třída zajišťující komunikaci přes SNMP protokol.*
  - RMONHelper – Třída zajišťující komunikaci přes RMON protokol.*
  - COBWEBHelper – Třída zajišťující komunikaci se proprietární zařízeními s COBWEB agentem.*
  - ICMPHelper – Třída zajišťující komunikaci přes ICMP protokol.*
  - IPMIHelper – Třída zajišťující komunikaci přes IPMI protokol.*
- **zcu.cobweb.agent.dal**
  - RestHandler – Třída která obstarává komunikaci se serverem.
- **zcu.cobweb.agent.watch**
  - CobWebWatch – běžící kontrola na operace které se mají provést a rekonfigurace.
  - WatchedCobWebDevice – Pomocná třída pro udržování zařízení které se mají sledovat a operací s tím spojených.
  - WatchedCobWebProtocol – Pomocná třída pro udržování protokolů jednotlivých zařízení, která se mají sledovat.
  - WatchedCobWebOID – Pomocná třída pro udržování vlastností které se mají sledovat a operací s tím spojených.

Agent využívá následující knihovny:

- Jersey – Knihovny určené pro komunikaci přes REST API.
- Jackson – Knihovny sloužící pro marshaling a unmarshaling dat předávaných ve formátu JSON.
- Sigar – Knihovny využívané pro sledování přes COBWEBHelper.
- AngryIP – Knihovy využívané pro skenování sítě.
- CobWebShared – Knihovna obsahující všechny Entity využívané pro přenosy a ukládání do databáze.
- JDOM – Knihovny sloužící pro práci s XML soubory.

3.3.2 Backend Server

Backendový server jak již bylo naznačeno se stará především o komunikaci a perzistování dat do a z databáze. Jeho vnější rozhraní je REST API a jsou tedy možné následující operace:

- Ukládání vzorků
- Získání vzorků pro jednotlivé vlastnosti či zařízení
- Vytváření, aktualizace a uchovávání konfiguračních entit agenta-serveru
- Operace nad agenty (ukončení všech, či kontrola běhu atp.)

U Backendového serveru bylo největší obtíží správně anotovat entity tak aby došlo k jejich správné serializaci a deserializaci, popř. perzistenci přes Entity Managera. Velkým problémem bylo přenesení stromové struktury a entit v seznamech. Výchozí provider Jersey pro unmarshalling JSON objektů nepracoval správně a tak bylo zapotřebí si napsat svého nebo nalézt vhodného providera. Naštěstí Jackson provider který je součástí Jersey distribuce již serializuje a deserializuje objekty správně.

Backend byl rozdělen do následujících balíčků:

- **zcu.cobweb.backend.handlers**
  - AgentsConfigurationsHandler – Servlet který obsluhuje REST požadavky na Entity groupu CobWebAgent.
  - OIDHandler – Servlet který obsluhuje REST požadavky na Entity groupu CobWebOID
  - SamplesHandler – Servlet který obsluhuje REST požadavky na Entity groupu CobWebSample.
- **zcu.cobweb.backend.jpa**
  - JPADAL – Třída sloužící pro přístup k datům do databáze.
  - EMF – Entity manager factory – Tedy třída pro vytváření EntityManagera, který zajišťujíce operace na databází.

Backend server využívá následující knihovny:

- Jersey – Knihovny určené pro komunikaci přes REST API.
- Jackson – Knihovny sloužící pro marshaling a unmarshaling dat předávaných ve formátu JSON.
- CobWebShared – Knihovna obsahující všechny Entity využívané pro přenosy a ukládání do databáze.
- AppEngine – Knihovy využívané pro běh na App Engine prostředí.

Backend server dále využívá:

- persistance.xml – Soubor ve kterém je nutné definovat jaké třídy a jakým způsobem je možné perzistovat.
- appengine.we.xml – Definuje způsob a kam se má aplikace v rámci GAE nasadit.

3.3.3 Frontend Server

Tento projekt jak již název vypovídá se staral především o vykreslování GUI do webového prohlížeče postavený na frameworku Vaadin. Komunikace s Backend serverem probíhá samozřejmě přes REST API a ve formátu JSON. Asi největší překážkou bylo zjištění že původně plánovaný Timeline addon není z důvodů využívání knihovny awt.Color kompatibilní s GAE (knihovna awt.Color je uvedena na seznamu zakázaných knihoven)

Finální vzhled uživatelského GUI a jeho možnosti je vidět v příloze č.1 Uživatelská příručka. Jelikož aplikace měla být silně orientována na mobilní zařízení dělil jsem okna pro rozlišení 480x800. Možnosti rozšíření GUI je mnoho stejně tak jako nabízených addonů pro Vaadin.

Frontend byl rozdělen do následujících balíčků:

- **zcu.cobweb.frontend.vaadin**
  - CobwebFrontEndVaadinApplication – Hlavní servlet.
- **zcu.cobweb.frontend.vaadin.data**
  - BackendDAL – Třída pro přístup k Backend serveru přes REST API.
- **zcu.cobweb.frontend.vaadin.ui**
  - AFormWithConfigurationItem – Abstraktní třída sdružuje společné vlastnosti formulářů.
  - AgentFieldFactory – Třída pro vytváření jednotlivých polí formuláře pro agenty.
  - AgentsContainerWrapper – Kontejner obsahující data. Je tak oddělen způsob uložení dat a přístup k nim.
  - CategoryFieldFactory - Třída pro vytváření jednotlivých polí formuláře pro kategorie.
  - CobWebTabItem – Kontejner na položky pro tab menu.
  - CobWebTreeTable – Strom v levém menu.
  - ConfigurationItemForm – Kontejner obsahující jednotlivé formuláře.
  - DeviceFieldFactory - Třída pro vytváření jednotlivých polí formuláře pro zařízení.
  - FormWithAgent – Formulář obsahující údaje o agentovi.
  - FormWithCategory – Formulář obsahující údaje o kategorii.
  - FormWithDevice - Formulář obsahující údaje o zařízení.
  - FormWithOID - Formulář obsahující údaje o vlastnosti.
  - FormWithProtocol - Formulář obsahující údaje o protokolu.
  - ITEMTYPE – Enum sloužící k rozeznání o jakou položku se ve stromovém menu jedná.
  - MenuBarWithIconsExample – Hlavní menu.
  - OidFieldFactory - Třída pro vytváření jednotlivých polí formuláře pro vlastnosti.
  - ProtocolFieldFactory - Třída pro vytváření jednotlivých polí formuláře pro protokoly.
  - SplitPanelBasicExample – Kontejner obsahující levou i pravou část s možností změny poměru stran.
  - TabSheetScrollingExample – Kontejner obsahující tab menu.
  - TreeTableItem – Pomocná třída obsahující jednotlivé položky stromu.
  - WindowNotificationBuilder – pomocná třída pro vytváření oken s oznámením.

Frontend server využívá následující knihovny:

- Jersey – Knihovny určené pro komunikaci přes REST API.
- Jackson – Knihovny sloužící pro marshaling a unmarshaling dat předávaných ve formátu JSON.
- CobWebShared – Knihovna obsahující všechny Entity využívané pro přenosy a ukládání do databáze.
- Vaadin – Nadstavba nad GWT s addonem Visualizations
-

Frontend server dále využívá:

- themes – složka obsahující jednotlivé vzhledy
- web.xml – Definuje mapování servletů na jednotlivé url adresy a zároveň inicializační parametry.
- appengine-web.xml – Definuje způsob a kam se má aplikace v rámci GAE nasadit.

3.3.4 Shared

Projekt za účelem jednotných entit anotovaných pro serializaci (marshalling), deserializaci (unmarshalling) a perzistování do databáze. Při vzniku tohoto projektu bylo největší překážkou správný „enhance“ entit. Tedy aby se při svém překladu zkompilovali do správného tvaru. Nejdříve jsem zkoušel v prostředí Eclipse vytvořit vlastní projekt s DataNucles Enhancerem bohužel toto řešení bylo spíše přítěží nežli přínosem. Nakonec jsem vyřešil situaci založením GAE projektu ve kterém je již vše předkonfigurováno společně s DataNucleus Enhancerem. I dále pak působil Enhancer problémy jelikož při volbě automatického „Enhance“ entit docházelo často k Enhance jen editovaných entit. Vznik takovýchto entit však vedl k jednotnosti aplikace a možností provádět všechna porovnání nad encodedKey (unikátními klíči). Tyto klíče jsou generovány JPA vrstvou při ukládání.

Shared byl rozdělen do následujících balíčků:

- **zcu.cobweb.shared.entities**
    - ACobWebConfigurationItem – Abstraktní třída definující společné vlastnosti Entity groupy Agent.
    - AGENTCOMMANDS – Enum obsahující příkazy pro agenta.
    - AGENTSTATE – Enum pro stavy agenta(RUN,STOP atd.).
    - CobWebAgent – Entita agent.
    - CobWebDevice – Entita zařízení.
    - CobWebOID – Entita vlastnosti.
    - CobWebProtocol – Entita protokolu.
    - CobWebSample – Entita vzorků.
    - DEVICESTATE – Enum stavů zařízení.
    - DEVICETYPE – Enum typů zařízení.
    - ICobWebConfigurationItem – Rozhraní pro entity Entity Groupy Agent.
    - OIDSTATE – Enum stavu vlastnosti.

- PROTOCOLSTATE – Enum stavu protokolu.
- PROTOCOLTYPE – Enum typů protokolů.

Shared využívá následující knihovny:

- Jersey – Knihovny určené pro komunikaci přes REST API.
- Jackson – Knihovny sloužící pro marshaling a unmarshaling dat předávaných ve formátu JSON.
- AppEngine – Především již zmíněný Datanucles Enhancer.

Z tohoto projektu jsem tedy generoval .jar soubor využívaný ostatními projekty.
Popis anotací:

@XmlRootElement – Slouží pro Jersey jako výchozí bod pro serializaci/deserializaci
@Entity – anotace určující pro JPA co je entita
@XmlType – Využívá se pro dědičnost v Jersey
@MappedSuperclass - Využívá se pro dědičnost JPA
@Id – určuje primární klíč pro JPA
@GeneratedValue – Pro automaticky generovaný klíč
@Extension – rozšíření – použil jsem jen pro generování primárního klíče přes gae.encoded-pk
@JsonManagedReference – Určení pro Jersey že se jedná o seznam prvků
@OneToMany – Určuje pro JPA o jakou vazbu se jedná
@Version – JPA určení automatického verzování entit. Bohužel fungovalo chybně a tak jsem musel ošetřit verzování aplikačně.
@JsonBackReference – Určení pro Jersey že se jedná o zpětnou vazbu (aby nedošlo k zacyklení při serializaci/deserializaci)
@ManyToOne – To samé jako @JsonBackReference avšak pro JPA

4. Závěr

V této práci jsem se především snažil o průzkum možností spojený s následnou ukázku nových technologií a zajímavého řešení. Také bych rád podotkl, že se nejedná o celkové řešení, ale o nástin a kostru pro vývoj obdoby již zmíněného Zabbixu a Nagiose. Věřím, že na takto postaveném základu je možné v relativně krátké době tuto vizi uskutečnit. Navíc GAE je velmi rychle a dynamicky se rozvíjející platforma jejíž možnosti se budou zajisté v budoucnu ještě prohlubovat.

Výsledkem je tedy obsáhlý průzkum možností, implementace základní kostry postavené na aktuálně nejpoužívanějších technologiích a webové GUI přístupné i pro mobilní zařízení. Oproti původní představě se práce dosti odlišila především na základě průzkumu konkurenčních SW řešení. Nejednalo se tedy o malou aplikaci směřovanou na jeden konkrétní protokol, ale především o průzkum, vyhodnocení a implementaci základní funkčnosti nového a konkurence schopného řešení.

Některé cíle tak byli o mnohé překonány a jiné se nepodařilo splnit vůbec. Například jednoduchá instalace, nasazení, konfigurace a škálovatelnost se rozrostly za hranici zadání. Důvodem je především zajímavý leč časově náročný průzkum možností spojený s „donucením“ jednotlivých frameworků ke spolupráci.

Jak jsem již zmínil výše na tomto základu je možno pokračovat mnoha směry a vylepšeními např.:

- Interaktivní mapa zařízeních (Drag a Drop nabízí Vaadin v základu)
- Doplnění o real-time protokol
- Spojení s aktivními sítěmi
- Implementovat zabezpečení Autentizace a Autorizace (GAE API)
- Zasílání zpráv Sms, Email Chat (GAE API)
- a mnoho dalších...

Seznam použitých zkratek a pojmů

AJAX	Asynchronous JavaScript and XML
API	Application programming interface
BMC	Baseboard Management Controller
CPU	Central processing unit
CRUD	Create, Read, Update, Delete
CLI	Command-line interface
DRAC	Dell Remote Access Controller
EG	Entity Group
GAE	Google App Engine
GWT	Google Web Toolkit
HTTP	Hypertext Transfer Protocol Secure
IAAS	Infrastructure as a service
ICMP	Internet Control Message Protocol
ILOM	Integrated Lights Out Manager
IP	Internet Protocol
IPMI	Intelligent Platform Management Interface
JEE	Java Enterprise Edition
JDBC	Java Database Connectivity
JDO	Java Data Objects
JDK	Java Development Kit
JPA	Java Persistence API
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MIB	Management Information Base
NAT	Network address translation
OS	Operating system
OID	Object identifier
ORM	Objektově relační mapování
PVM	Parallel Virtual Machine
PAAS	Platform as a service
REST	Representational State Transfer
RMON	Remote Network MONitoring
Request	Požadavek
Response	Odpověď
RPC	Remote procedure call
SOAP	Simple Object Access Protocol
SNMP	Simple Network Management Protocol
SAAS	Software as a Service
SOA	Servisně orientovaná architektura
TCP	Transmission Control Protocol

TTL	Time To Live
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
WS	Web Service, Web Server
WSDL	Web Services Description Language
XML	eXtensible Markup Language

Zdroje

[1] CASE, J., A Simple Network Management Protocol (SNMP), RFC 1157, SNMP Research, květen 1990, Dostupné z URL: <http://www.ietf.org/rfc/rfc1157.txt>

[2] POSTEL, J., INTERNET CONTROL MESSAGE PROTOCOL, RFC 792, ISI, září 1981, Dostupné z URL: <http://www.ietf.org/rfc/rfc792.txt>

[3] Intelligent Platform Management , Interface Specification , Second Generation v2.0, Document Revision 1.0 , February 12, 2004 , Intel , Hewlett-Packard, NEC, Dell Dostupné z URL <http://download.intel.com/design/servers/ipmi/IPMIv2_0rev1_0.pdf>

[4] KOSEK, J., *PHP – tvorba interaktivních internetových aplikací*. 1 vydání. Praha: Grada Publishing, 1999. 492 s. ISBN 80-7169-373-1. Podpora dostupná na http://www.kosek.cz/php/.

[5] THURLOW, R. RPC: Remote Procedure Call Protocol Specification Version 2, RFC 5531, Sun Microsystems, Network Working Group, květen 2009 Dostupné z URL: <**http://www.ietf.org/rfc/rfc5531.txt**>.

[6] MAIER, D., The Theory of Relational Databases. Computer Science Press 1983, ISBN 0-914894-42-0

[7] Tim Bray, Jean Paoli, Sperberg C. M. Mcqueen, Eve Maler, Franois Yergeau. Extensible markup language (XML) 1.0 (third edition), 2004

[bak] *Grafické zpracování dat SNMP*. Plzeň, 2008. Bakalářská práce. ZČU. Vedoucí práce Ing. Jiří Ledvina CSc.

[root] JDK 7. In: *Root* [online]. 2010 [cit. 2012-05-17]. Dostupné z: http://www.root.cz/clanky/novinky-v-nbsp-jdk-7-aneb-mirny-pokrok-v-nbsp-mezich-zakona-1/

[skriv] Simple Network Management Protocol. In: *Skriv* [online]. 2007 [cit. 2012-05-17]. Dostupné z: http://skriv.wz.cz/MPS/SNMP/index.html

[wtcs] Simple Network Management Protocol. In: Wtcs [online]. 2007 [cit. 2012-05-17]. Dostupné z: http://www.wtcs.org/snmp4tpc/snmp.htm

[nmap] Internet Control Message Protocol. In: Nmap [online]. 2007 [cit. 2012-05-17]. Dostupné z: http://nmap.org/book/tcpip-ref.html

[icmpwiki] Internet Control Message Protocol. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-17]. Dostupné z: http://cs.wikipedia.org/wiki/ICMP

[ipmiwiki] Intelligent Platform Management Interface. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-17]. Dostupné z: http://en.wikipedia.org/wiki/Intelligent_Platform_Management_Interface

[svetsiti] IPMI. Svět sítí [online]. 2005 [cit. 2012-05-17]. Dostupné z: http://www.svetsiti.cz/clanek.asp?cid=IPMI-V20-zjednodusuje-spravu-serveru-__28112005

[kosek] SOAP. Kosek [online]. 2005 [cit. 2012-05-17]. Dostupné z: http://www.kosek.cz/diplomka/html/websluzby.html

[rpcwiki] Remote procedure call. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-17]. Dostupné z: http://cs.wikipedia.org/wiki/Remote_procedure_call

[newmarch] RPC. Newmarch [online]. 2005 [cit. 2012-05-17]. Dostupné z: http://jan.newmarch.name/go/rpc/chapter-rpc.html

[zdrojak] REST. Zdroják [online]. 2009 [cit. 2012-05-17]. Dostupné z: http://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

[restwiki] Representational State Transfer. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-17]. Dostupné z: http://cs.wikipedia.org/wiki/Representational_State_Transfer

[itf] SOA. Information technology forum [online]. 2009 [cit. 2012-05-17]. Dostupné z: http://information-technology-forum.blogspot.com/2009/06/technical-interoperability-of-disparate.html

[wswiki] Webový server. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-17]. Dostupné z: http://cs.wikipedia.org/wiki/Webový_server

[wsroot] Aplikační server. Root [online]. 2008 [cit. 2012-05-17]. Dostupné z: http://www.root.cz/clanky/jboss-aplikacni-server/

[cloud] Cloud computing. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-17]. Dostupné z: http://cs.wikipedia.org/wiki/Cloud_computing

[gae] Google App Engine. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-17]. Dostupné z: http://en.wikipedia.org/wiki/Google_App_Engine

[lupa] VMforce. Lupa [online]. 2010 [cit. 2012-05-17]. Dostupné z: http://www.lupa.cz/tiskove-zpravy/salesforce-com-a-vmware-uzaviraji-partnerstvi/

[rdb] Relační databáze. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-17]. Dostupné z: http://cs.wikipedia.org/wiki/Relační_databáze

[interval] JDBC. Interval [online]. 2003 [cit. 2012-05-17]. Dostupné z: http://interval.cz/clanky/uvod-do-jdbc/

[orm] ORM. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-17]. Dostupné z: http://cs.wikipedia.org/wiki/Objektově_relační_mapování

[jpa] JPA. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-17]. Dostupné z: http://cs.wikipedia.org/wiki/Java_Persistence_API

[jdointerval] JDO. Interval [online]. 2005 [cit. 2012-05-17]. Dostupné z: http://interval.cz/clanky/jdo-java-data-objects/

[hibernate] Hibernate. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-17]. Dostupné z: http://cs.wikipedia.org/wiki/Hibernate

[gwtinterval] GWT. Interval [online]. 2006 [cit. 2012-05-17]. Dostupné z: http://interval.cz/clanky/google-web-toolkit/

[vaadin] Vaadin. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-17]. Dostupné z: http://cs.wikipedia.org/wiki/Vaadin

[json] JSON. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-17]. Dostupné z: http://cs.wikipedia.org/wiki/JavaScript_Object_Notation

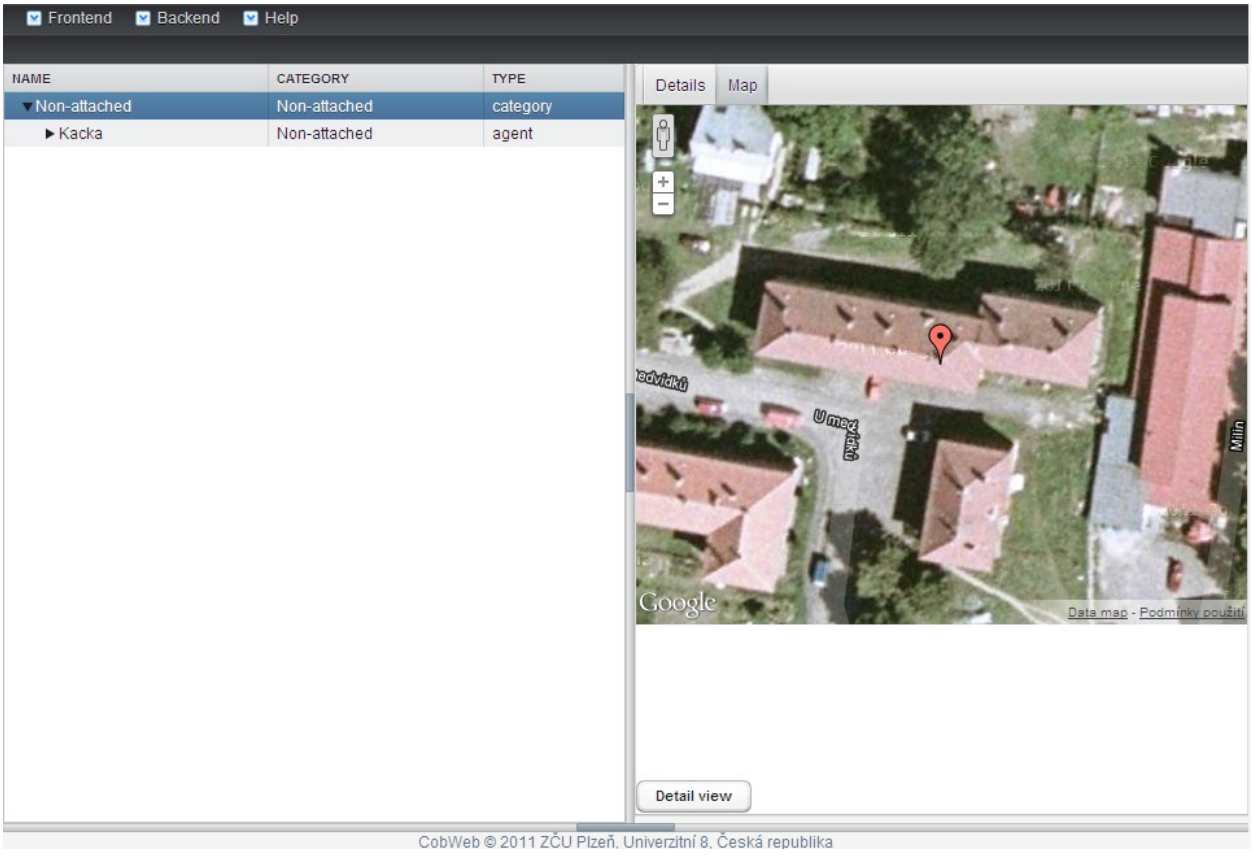
[xml] XML. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-05-17]. Dostupné z: http://cs.wikipedia.org/wiki/Extensible_Markup_Language

Přílohy

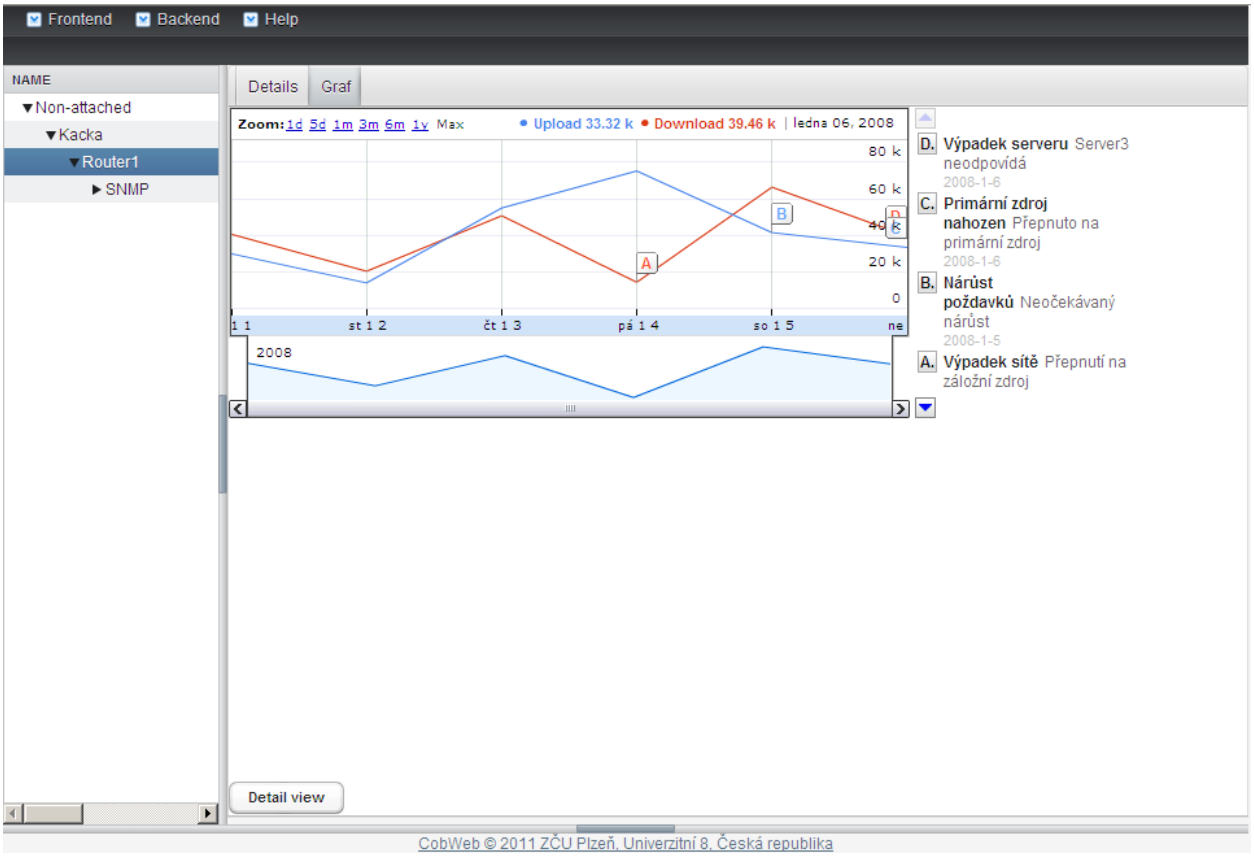
Příloha 1 Uživatelská příručka

GUI Frontendového serveru je tedy rozděleno do dvou částí o rozlišení 480x800px se společným menu. Je tomu tak především kvůli optimalizaci pro mobilní zařízení. V horním menu je možné odemykat a zamykat poměr stran které je možný změnit tažením dané hrany. Skrz nabídku Frontend->Splitpanels-> Splitpanels bar locked. Je možné zkontrolovat komunikaci s Backend server skrze menu Backend->Check communication. Tlačítko „Refresh tree“ spustěné z hlavního menu nebo po rozkliku možností pravým tlačítkem přímo ve stromu dotáhne všechna aktuální data (POZOR pokud jste udělali změny a nebyli odeslány na server budou smazány). Tlačítko „Commit all changes“ uloží všechny provedené změny na Backend server. Tlačítka „Help“ a „About“ není potřeba přibližovat.

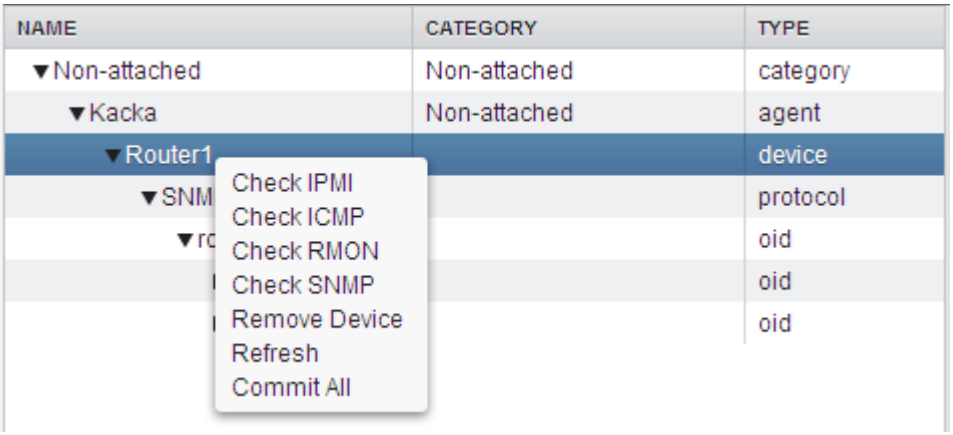
Hlavní částí je tedy levé menu se stromem. Při zvolení položky ve stromu se nám nabídne soubor statistik a možného nastavení v pravém menu pod jednotlivými záložkami. Velmi důležité jsou formulářové údaje podle kterým se sledování řídí. Pokud chceme agenta přeřadit do jiné kategorie je nutné v jeho formuláři změnit kategorii → Commit all changes a následně Refresh. Kategorie je automaticky vytvoří. Jedním ze zajímavých prvků kategorie je mapa Agentů zobrazených na Google maps.



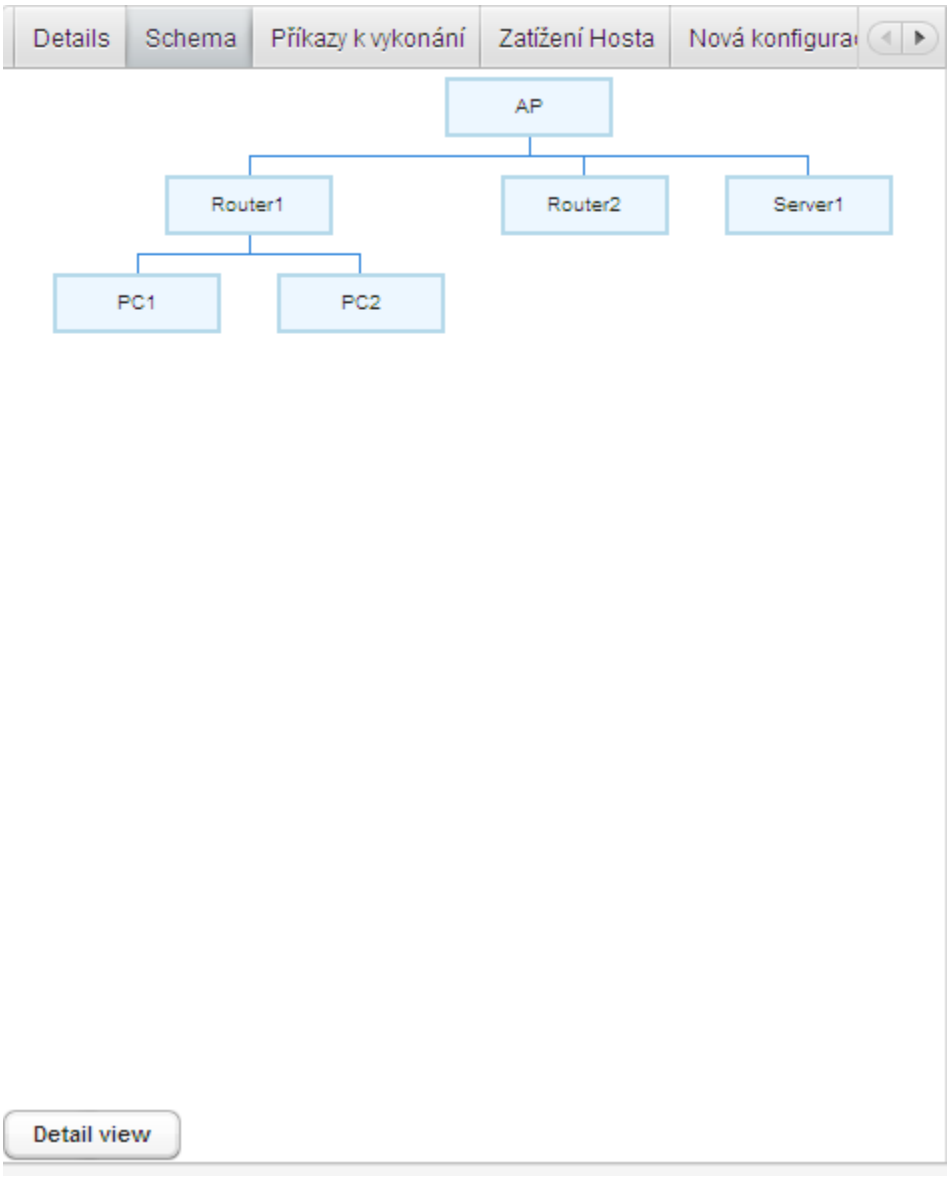
Všechny statistické údaje je možné si otevřít pro porovnání v novém okně viz. obr. Níže. V „Timeline“ grafu je možné si přejíždět posuvníkem nebo se pohybovat po událostech v pravé části. V Motion grafu je možné sledovat zrychlený průběh vlastnosti.



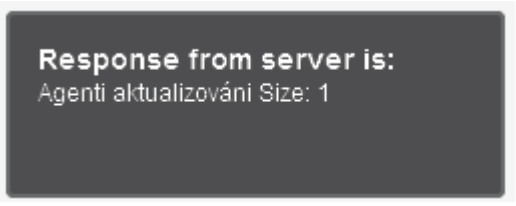
Při kliknutí pravým tlačítkem ve stromu dojde k otevření rozšířené nabídky pro danou položku.



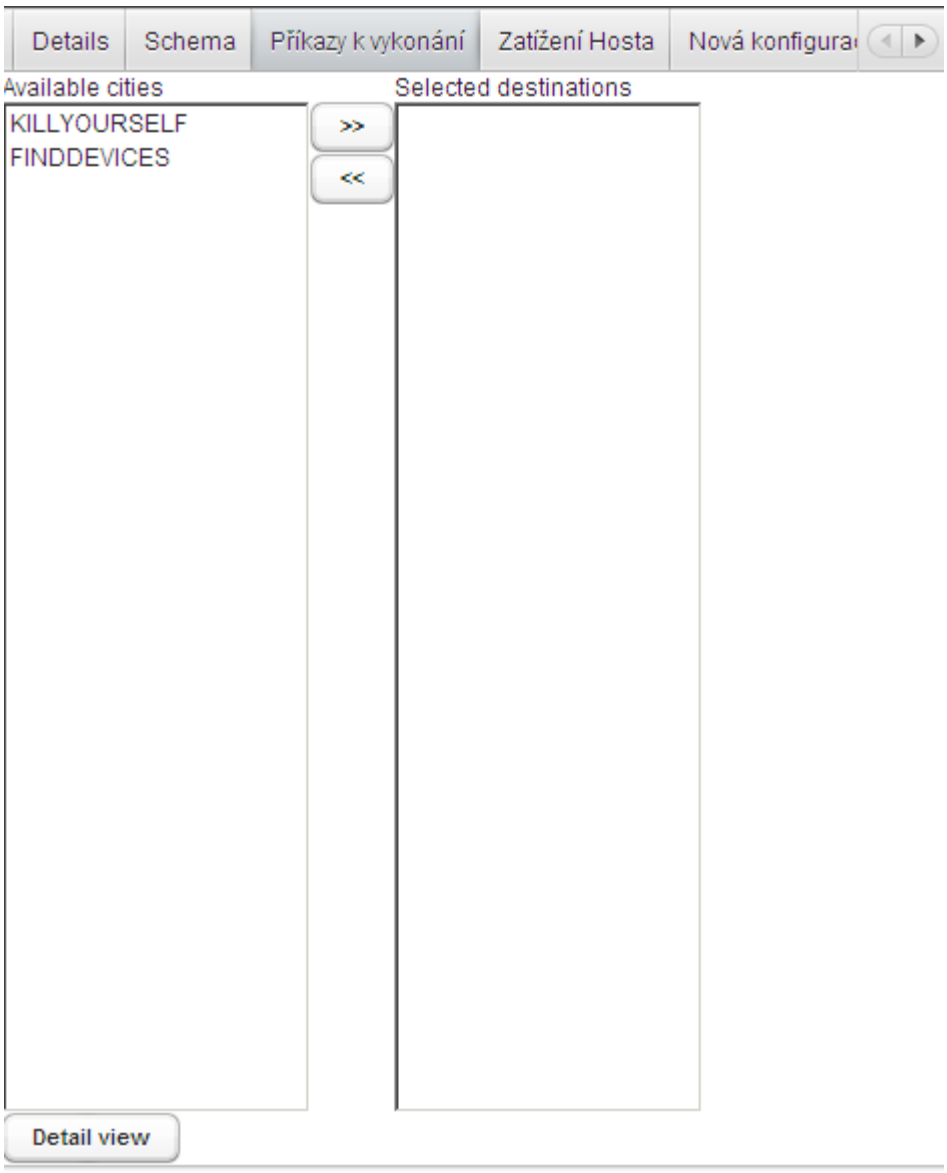
U položky agenta je také možné vidět schéma zařízení.



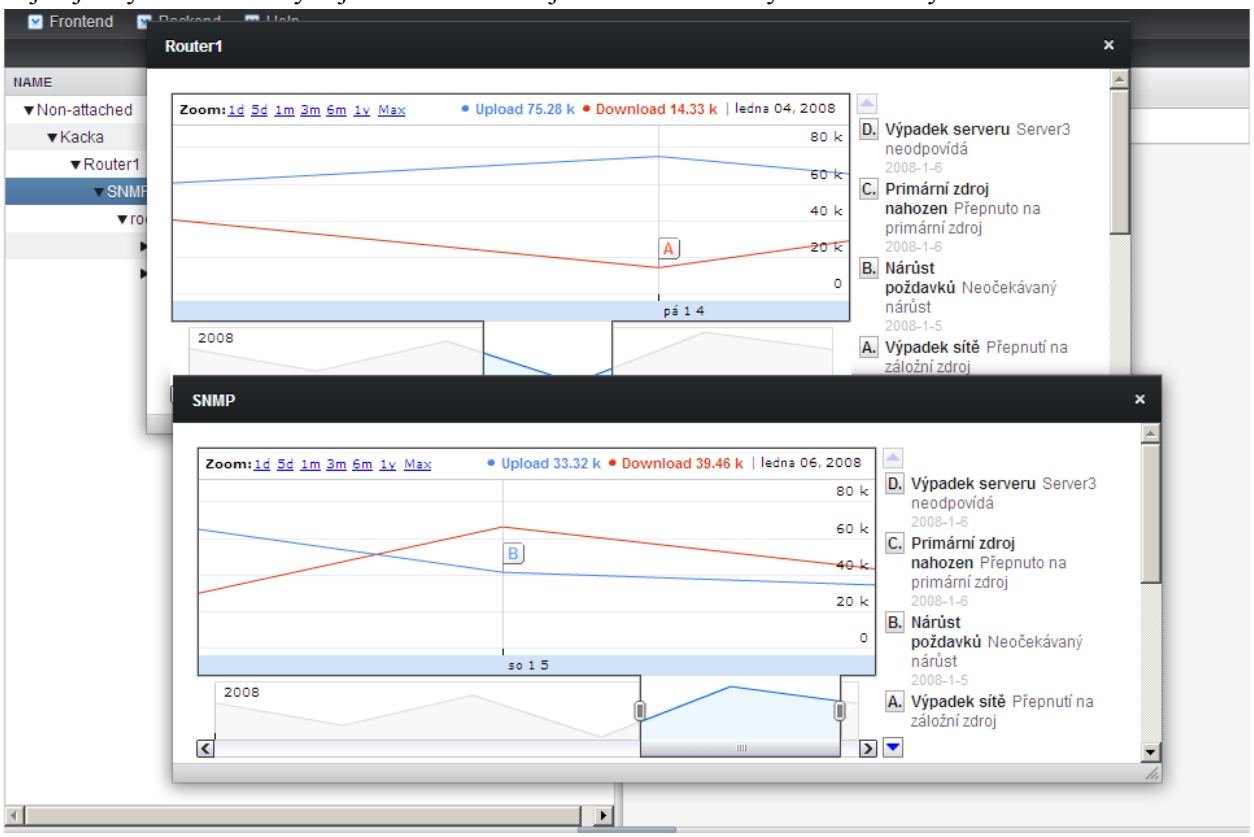
V pravém dolním roku jsou zobrazovány upozornění.



U agenta a zařízení je možné vybrat příkazy k provedení.



A jak již bylo zmíněno výše je možné otevírat jednotlivé statistiky v samostatných oknech.



Nastavení a běh:

1.Agent

Agent je nutnou součástí pro sledování. Je vhodné jej umístit do vnitřní sítě, např. na server. Jeho paměťová náročnost se pohybuje v řádech jednotek MB. Agenta je možné spustit 4mi způsoby, přes výchozí agentconfiguration.xml, vlastní xml, parametry zadanými při spuštění nebo zadáváním do konzoly.

Zadávají se údaje:

-adresa serveru

-port

-název agenta

-jak často si má agent zjišťovat novou konfiguraci (0) znamená manuální ovládání

-Y/N - zda-li si má agent najít zařízení v okolí automaticky

```
Agent spuštěn
Kontroluji spojení se serverem
Odpověď serveru ....
Agents Configuration service is up and running
Přidávám agenta
Odpověď serveru....
Agent přidán Name: Kacka, GUID: aglub19hcHBfaWRyEgsSC0NvYldlYkFnZW50GNoEDA, Version:0
Agent je nakonfigurován ze vstupních parametrů, manuálně:true
Vyberte z následujících možností:
 1 - Dotázat se na novou konfiguraci
 2 - Zjistit zařízení v okolí
 3 - Začni sledovat zařízení v okolí
 9 - Konec
2
Prosím zadejte název tohoto zařízení (např. Počítač-doma)
Router1
Prosím zadejte adresu zařízení (např. 192.168.1.11, localhost):
192.168.1.253
Vyberte protokol pro zařízení Router1 :
 1 - Zjistit nové SNMP zařízení
 2 - Zjistit nové RMON zařízení
 3 - Zjistit nové COBWEB zařízení
 4 - Zjistit automaticky všechny protokoly tohoto zařízení
 9 - Zpět
1
Aktualizuji agenta
Odpověď serveru....
Agent aktualizován Name: Kacka, GUID: aglub19hcHBfaWRyEgsSC0NvYldlYkFnZW50GNoEDA, Version:1
Aktualizuji vlastnosti zařízení
Odpověď serveru....
Počet přidaných vlastností: 1
Vyberte z následujících možností:
 1 - Dotázat se na novou konfiguraci
 2 - Zjistit zařízení v okolí
 3 - Začni sledovat zařízení v okolí
 9 - Konec
```

2.Frontend

V této fázi je agent nakonfigurován a je potřeba mu říci co tedy má sledovat. To se nastaví v GUI. Rozklikneme tedy v levém stromu kategorii „Nezařazení“ nalezneme vlastnosti kterou si přejeme sledovata změníme ji na „RUN“, poté je potřeba zakliknout „Apply“ čímž se změny uloží v GUI. Poté můžeme pokračovat ve výběru dalších vlastností pro sledování. Pokud jsme nastavili již vše co jsme chtěli stiskneme „Commit all chnages“, čímž se změněná data uloží na server a agent se tedy podle nich může nakonfigurovat.

3.Agent

V této fázi, pokud je tedy běh agenta zvolen manuálně je potřeba provést viz. obrázek níže.

```
Vyberte protokol pro zařízení Router1 :
 1 - Zjistit nové SNMP zařízení
 2 - Zjistit nové RMON zařízení
 3 - Zjistit nové COBWEB zařízení
 4 - Zjistit automaticky všechny protokoly tohoto zařízení
 9 - Zpět
1
Aktualizuji agenta
Odpověď serveru....
Agent aktualizován Name: Kacka, GUID: aglub19hcHBfaWRyEgsSC0NvYldlYkFnZW50GOIEDA, Version:1
Aktualizuji vlastnosti zařízení
Odpověď serveru....
Počet přidaných vlastností: 1
Vyberte z následujících možností:
 1 - Dotázat se na novou konfiguraci
 2 - Zjistit zařízení v okolí
 3 - Začni sledovat zařízení v okolí
 9 - Konec
1
Získávám agenta
Odpověď serveru....
Agent získán Name: Kacka, GUID: aglub19hcHBfaWRyEgsSC0NvYldlYkFnZW50GOIEDA, Version:2
Vyberte z následujících možností:
 1 - Dotázat se na novou konfiguraci
 2 - Zjistit zařízení v okolí
 3 - Začni sledovat zařízení v okolí
 9 - Konec
3
```

4.Frontend

Agent je tedy nakonfigurován a sleduje zvolené vlastnosti. Teď je tedy možné tyto data procházet zobrazovat a měnit.