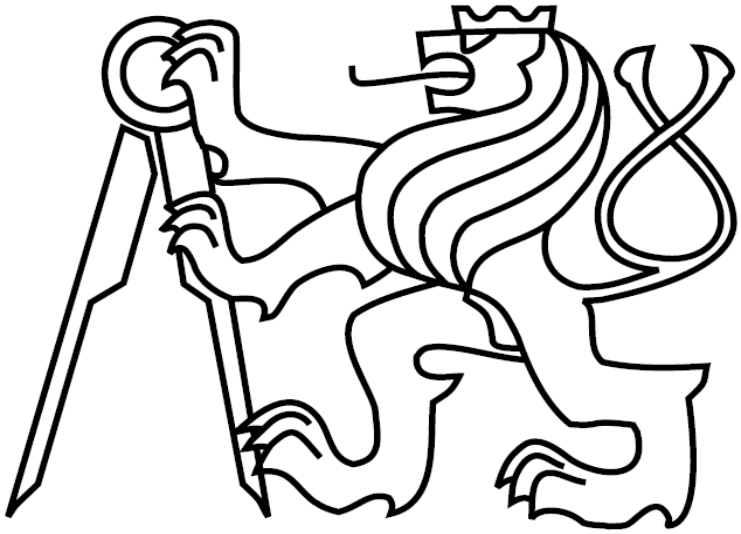
České vysoké učení technické v Praze
Fakulta elektrotechnická



Diplomová práce

Vizualizace počítačových sítí

Karel Pochop

Vedoucí práce: ing. Peter Macejko

Studijní program: Elektrotechnika a informatika dobíhající magisterský

Obor: Informatika a výpočetní technika

leden 2009

Poděkování

Děkuji ing. Petru Macejkovi za jeho pomoc při vytváření této práce. Děkuji také správcům sítě na Sinkuleho koleji za jejich připomínky a pomoc při testování programu.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

Nemám závažný důvod proti použití tohoto školního díla ve smyslu §60 Zákona č. 121/2000Sb., o právu autorském, o právech související s právem autorským a o změně některých zákonů (autorský zákon)

V Praze dne 20.1.2009 .....

Abstrakt

Tato diplomová práce se zabývá návrhem a implementací webové aplikace pro vizualizaci a správu počítačové sítě. Součástí aplikace bude modul zobrazující topologii zkoumané sítě. Tento modul bude implementován s použitím technologie Adobe Flash. Informace o síti se budou zjišťovat za pomocí protokolu SNMP. Aplikace bude navržena tak, aby umožňovala úpravu a tvorbu modelu pomocí uživatelského rozhraní.

Abstract

This thesis deals with an analysis and implementation of a web application for visualization and computer network administration. As a part of the application will be a module which will scan network topology to get information on the network using SNMP. This module will be implemented using Adobe Flash technology. Desing the application for adjustment and creation a model using the user interface.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

1.1 Motivace

Cílem projektu je zjednodušit a zefektivnit práci administrátorů počítačových sítí za pomocí nových vizualizačních technologií. K tomuto účelu se budou získávat data z měřené počítačové sítě za pomoci SNMP. A tyto data se budou uceleně zobrazovat ve webovém rozhraní. Hlavní součástí této aplikace budou vizualizační moduly, které budou vytvořeny pomoci technologie Adobe Flash.

1.2 Přínos projektu

Hlavním přínosem projektu je dříve požadované zefektivnění a zjednodušení práce administrátora. Důležitým přínosem je i možnost distribuce výpočtu mezi více agenty a podpora pro používání většího množství modelů v rámci jednoho webového rozhraní. Dalším přínosem je možnost použití aplikace pro zaznamenání historie měřených topologii, včetně jejich grafů a možnost použití aplikace jako zápisníku topologií s použitím virtuálních prvků. A konečně data o jednotlivých síťových prvcích a jejich portech je možné vyexportovat do CSV souborů.

1.3 Unikátnost projektu

Unikátnost projektu je v ojedinělém zobrazení topologie, kde se využívá technologie Adobe Flash. Výsledná grafová a topologická zobrazení nejsou totiž statická, ale dynamická a interaktivní. Administrátor může měnit vizuální uspořádání prvků na scéně a přidávat do ní virtuální prvky. A přitom všechny tyto činnosti je možné vykonávat ve webovém rozhraní. Celý projekt je zároveň unikátní i zvolenou 3-složkovou modulární architekturou, která umožňuje nahrazení jednotlivých modulů při zachování formátu přenášených zpráv. Zajímavostí je i možnost zobrazení zkoumaných topologií na externích webových stránkách.

1.4 Struktura práce

Výsledná práce se skládá z několika kapitol. V první kapitole jsou uvedeny obecné informace o výsledném produktu. V druhé kapitole jsou popsány hlavní technologie, které se použijí při řešení problému. Ve třetí kapitole je vylíčena analýza a řešení jednotlivých bloků aplikace. Ve čtvrté kapitole je zjednodušeně charakterizována použitá implementace. V páté kapitole je objasněno a popsáno zvolené testování aplikace. V šesté kapitole je zmíněno obecné zhodnocení projektu. A končené v sedmé kapitole jsou uvedeny použité zdroje

2 Základní informace o použitých technologiích

Při řešení zadaného problému bylo použito velké množství rozdílných technologií. Některé z nich měli stěžejní význam ve výsledné aplikaci a z toho důvodu budou dále podrobněji popsány. Jedná se prvořadě o Simple Network Management Protokol a o komunikační možnosti technologie Flash.

2.1 Simple Network Management Protocol

Simple Network Management Protocol(dále jen SNMP) je velmi rozšířený, jednoduchý a standardizovaný protokol sloužící k získávání nebo nastavování hodnot na určitém zařízení, které má IP adresu a vestavěnou podporu pro SNMP[1]. SNMP v překladu znamená „jednoduchý protokol pro správu sítě“ a pracuje na aplikační vrstvě síťového modelu ISO/OSI. SNMP je využíván pro detekci chyb, měření statistických veličin, nastavování vlastností, získávání grafů u široké škály zařízení jako jsou servery, pracovní stanice, routry, switche, tiskárny, teplotní čidla a další[2]. V rámci následujících odstavců bude protokol SNMP popsán podrobněji.

2.1.1 Vznik a historie SNMP

Představme si učebnici historie v příštím století. Jakou technickou formu bude mít taková učebnice? Co si o nás a o naši společnosti budou myslet naši vzdálení potomci? Na takové otázky je obtížné korektně odpovědět. Můžeme se však domnívat, že Internetu a technologiím umožňujícím jeho vznik bude věnován značný prostor. Vždyť již v současné době je laickou veřejností Internet dáván na úroveň nejdůležitějším technologiím historie lidstva. S podstatnou mírou povrchnosti je možné přijímat tvrzení, která důležitost Internetu dávají na úroveň významu technologií jako jsou parní stroje, výbušné spalovací motory, využívání elektrické energie a na úroveň významu technologií integrovaných obvodů i samotných počítačů.

Internet v dnešní rozšířené podobě zvyšuje dostupnost informací, rozšiřuje komunikační možnosti uživatele, zvyšuje mobilitu lidské práce, zefektivňuje tržní ekonomiku, zmenšuje variabilní i provozní náklady firem a mění formy zábavy uživatelů. A v neposlední řadě je internet nástroj, který technicky umožňuje přerod zastupitelských demokratických systémů na přímé demokracie, kde zákonodárný sbor je částečně nahrazen občany, kteří budou rozhodovat o zákonech přes uživatelské rozhraní hlasovací aplikace.

K tomu, aby celý internet a jednotlivé lokální sítě korektně a efektivně pracovali je nutné použít značné množství aktivních i pasivních síťových prvků a různých komunikačních kanálů. Na začátku 80.let, kdy světu vládly sálové počítače a ARPANET obsahoval stovky zařízení, se správou sítí zabývali prvořadě vědečtí pracovnicí. Ale po rozdělení ARPANETu na civilní a vojenskou část a nástupem osobních počítačů došlo k masivnímu rozmachu internetu a tím i k změně požadavků na správu sítě. Ve druhé polovině 80.let 20.století, kdy počet připojených počítačů přesáhl 10 tisíc[3], se internet začal stávat poměrně robustní sítí s velkým množstvím

aktivních prvků. Jelikož tyto prvky mohli mít výrazně odlišné fyzické umístění, pak jejich případná konfigurace přímo na místě byla velmi nákladná. Zároveň správa takových zařízení začala být problematická i z toho důvodu, že aktivní prvky začali vyrábět různí výrobci, kteří aplikovali jiný technický postup konfigurace takových prvků. Což mělo za následek, že obsluhující personál musel používat různý postup konfigurace pro různá zařízení. Výsledkem těchto problémů bylo, že na konci 80.let 20.století vznikla poptávka na vytvoření sofistikovaného protokolu, jenž umožní efektivní správu jednotlivých počítačových sítí a jejich aktivních prvků. Nejdůležitější požadavky na takový protokol můžeme klasifikovat do několika skupin, které si následně popíšeme.

1. Univerzálnost

Součástí internetových sítí jsou zařízení, která byla vyrobena rozdílnými výrobci. Zároveň mohou jednotlivá zařízení mít výrazně odlišný cíl činnosti a tedy vyžadovat nastavování rozdílných parametrů a tedy i poskytovat o své funkci rozdílné informace. Například administrátor spravující switch resp. síťovou tiskárnu požaduje informace o zatížení jednotlivých portu resp. zahlcení tiskové fronty a zároveň potřebuje zpětně konfigurovat jednotlivé porty resp. nastavovat časové limity v tiskové frontě. Ve výsledku se pak musí administrátor seznamovat s různým způsobem konfigurace pro každé takové rozdílné zařízení. Dalším problémem je také obtížnější návrh a implementace aplikací, skriptů, které takovou síť mohou spravovat. Ve výsledku se pak prodražuje bezproblémová správa takové počítačové sítě. Abychom zmírnili rozdíly mezi jednotlivými zařízeními tak je vhodné, aby výsledný protokol měl pro všechna zařízení stejnou strukturu, ale zároveň umožňoval včleňovat informace specifická pro dané zařízení.

2. Škálovatelnost

Vývoj síťových prvků se nezastavil ve 20.století. Jednotlivý výrobci svoje produkty neustále vylepšují, přidávají nové funkce a snaží se reflektovat vývoj tak, aby byli úspěšnější na trhu. Např. dnešní switche podporují výrazně větší množství funkci oproti původnímu vzorům. Např. dnešní L3 switch může podporovat různé typy STP, podporovat technologii Virtual LAN (dále VLAN) a nebo se může na různých portech zároveň chovat jako opakovač, switch i router. Proto by hledaný protokol měl podporovat přidávání dalších funkcí a parametrů bez nutnosti výraznějších změn takového protokolu.

3. Správa z jednoho místa

V rozlehlých počítačových sítích mohou být jednotlivá zařízení rozmístěna ve velkých fyzických vzdálenostech a tedy případná rekonfigurace přímo na místě výskytu se jeví jako značně problematická. Ačkoli je samozřejmě nutné velké množství prvotních a servisních konfigurací provádět přímo na daném zařízení, tak by bylo jistě efektivnější maximální množství úprav provádět ze vzdáleného zařízení. Proto by dané technické řešení mělo umožnit provádět většinu konfiguračních zásahů přes síť a to bez nutnosti přímé fyzické dostupnosti daného zařízení. Budeme-li nastavovat důležité konfigurační parametry jako např. změnu stavů jednotlivých portů na aktivních zařízení a tím i měnit topologii sítě, pak je nutné dostatečně zohlednit bezpečností požadavky takového postupu.

4. Bezpečnost

Jelikož konfigurační zásahy administrátora mohou být uskutečňovány vzdáleně přes síť ,a tedy dané pakety budou moci být odposlouchávány, je nutné zabezpečit danou komunikaci a zvolit vhodný způsob autorizace administrátora a zabezpečení přenášených informací. U větších sítí je vhodné umožnit přidělování administračních práv jednotlivých aktivních prvků různým administrátorům a tedy umožnit vypracovanější autorizační schémata.

5. Měření parametrů

Při správě počítačové sítě se administrátor nevyhne nutnosti optimalizovat výkonnost sítě. Aby takové optimalizace byli korektní, je vhodné si zjistit zatížení jednotlivých prvků a linek tak, aby se následně zvolilo správně řešení optimalizace. Pod takovým řešením si můžeme například představit koupi výkonnějšího aktivního prvku, přechod na odolnější kanál, změnu topologie a nebo jen odpojení nepřizpůsobivého uživatele. Hledaný protokol by měl tedy zajistit možnost měření potřebných parametrů. Další problém nastane pokud chceme měřit takové veličiny ve velmi krátkých intervalech. Při takovém měření by mohlo dojít ke zbytečnému zatížení zkoumaných prvků. A proto by bylo vhodné, aby daná zařízení si mohla ukládat naměřené veličiny a protokol by pouze umožnit sběr takovýchto dat. Naopak měříme-li parametry po velmi velkých intervalech pak by hledaný protokol měl řešit případné přetečení měřících čítačů.

6. Detekce změn

Krom sledování přenášených dat uzly sítě, vyhodnocování a následné optimalizaci sítě je vhodné zajistit okamžitou detekci nenormálních stavů a situací, které vážně ohrožují korektní fungování sítě. Ať se jedná o drobné změny topologie sítě, výpadek některých zařízení, výpadek elektrické sítě, zhroucení komunikačního kanálu, je jistě vhodné umožnit administrátorovi rychlé zjištění takové situace. Výsledný protokol by měl takovou možnost nabídnout a v případě detekce závažných změn by dané zařízení mělo mít možnost se pokusit automaticky poslat varovný trap na server, kde běží kontrolní proces, který dále zajistí např. poslání SMS zprávy nebo emailu administrátorovi.

7. Jednoduchost, rozšiřitelnost

Obtížným leč nutným a téměř psychologickým problémem, jenž je nutné zvážit při vzniku technologie, je zvolení vhodné hranice mezi robustností a jednoduchostí této technologie. Bude-li řešení oplývat bohatými funkcemi s velkým množstvím parametrů může se výsledné použití omezit kuli obtížnému používání takovéto technologie. Bude-li řešení naopak jednoduché, pak praktickému využití může bránit nedostatečné funkční pokrytí.

První návrh takového protokolu, který by vycházel z výše popsaných požadavků, prezentovali James Davin, Martin Schoffstall, Mark Fedor a Jeff Case pod označením Simple Gateway Monitoring Protocol(dále jen SGMP) v roce 1987. O rok později byl ze SGMP odvozen protokol SNMP. V roce 1988 byl protokol SNMP popsán ve třech doporučeních RFC 1065-1067 se statusem draft. A konečně v roce 1990 byl protokol SNMP popsán ve třech doporučeních RFC 1155-1157 se statusem standard.

Protokol SNMP je založen na struktuře agent-manager, kde agent je zařízení podporující SNMP protokol a odpovídající na dotazy manažerovi. Komunikace probíhá tak, že manager posílá dotazy agentovi a ten na ně odpovídá a nebo agent posílá trapy manažerovi. Posílané zprávy mají jednoduchý charakter kódovaného jazyka Abstraktní Syntaktická Notace verze 1 (ASN.1). Jednotlivé parametry jsou uspořádány ve stromové struktuře tzv. Management Information Base (dále jen MIB). MIB je možné rozšiřovat přidáváním dalších podstromů. Tím je možné přidávat informace specifické pro daného výrobce a dané zařízení bez toho, aniž by se zhoršila univerzálnost MIB. Jedno z rozšíření toho stromu je i Remote Monitoring (dále jen RMON), které zaznamenává časový vývoj parametrů, tak aby se při přesném měření průběhu monitorovací program nemusel neustále dotazovat na numerickou hodnotu parametru, ale vystačil si z pozvolnějším dotazováním na příslušné informace z podstromu RMONu. Velkým problémem SNMP bylo nedostatečné zabezpečení přenášených zpráv a omezené možnosti autorizace. To vyprovokovalo vývoj dalších verzí SNMP, které zlepšili bezpečnost protokolu.

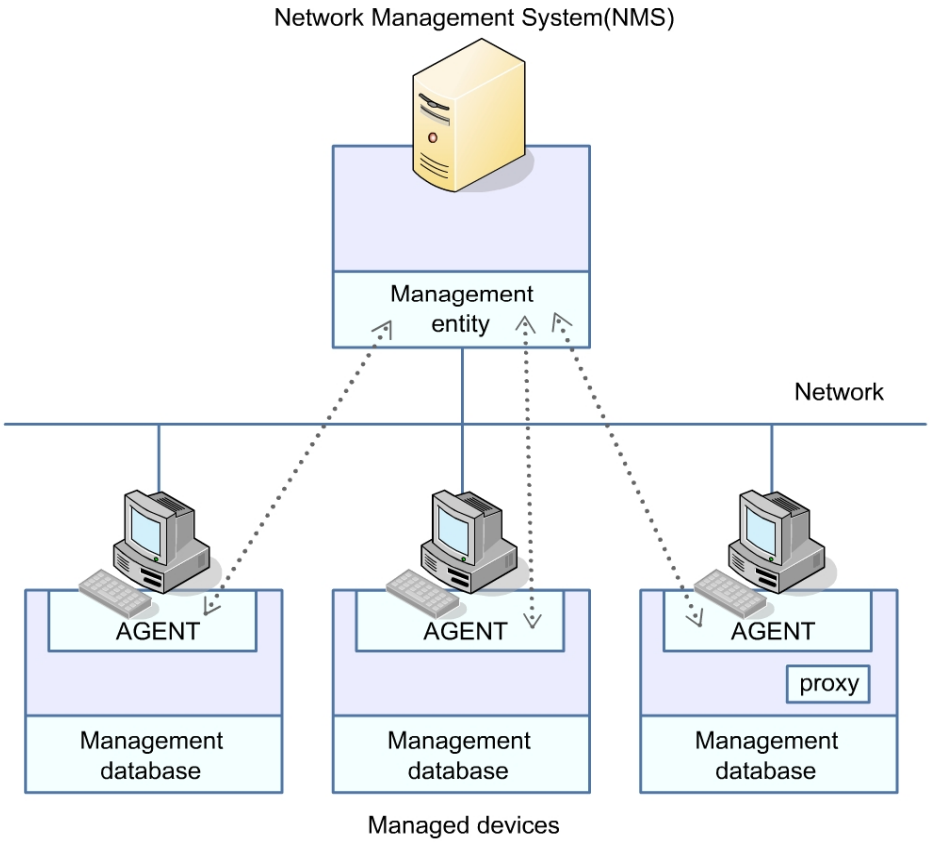
RFC 1155-1157	1990	Prvotní standard definující SNMP verze 1.
RFC 1213	1991	Rozšíření MIB stromu o podstrom MIB-II.
RFC 1351-1353	1992	Vylepšení možností autentizace a šifrování zasílaných správ.
RFC 1441-1452	1993	Definování SNMP verze 2.
RFC 1513, 1757	1995	Standard pro rozšíření RMON.
RFC 1901-1908	1996	Definování SNMP verze 2c.
RFC 1909-1910	1996	Definování SNMP verze 2u.
RFC 3410-3418	2002	Definování SNMP verze 3.
RFC 3584	2003	Popis koexistence mezi jednotlivými verzemi SNMP.

Tabulka 2.1: RFC dokumenty, které popisují protokol SNMP.

Pro definování SNMP mají podstatnou roli takzvaná doporučení Request for Comments (dále jen RFC) [4], která jsou vydávána organizací Internet Engineering Task Force (dále jen IETF). IETF je otevřená organizace, jenž vyvíjí internetové standardy a znatelně spolupracuje s organizací W3C a ISO/IEC [5]. V současné době existuje více jak 3 tisíce RFC dokumentů. Jednotlivé dokumenty RFC je možné rozdělit podle jejich typu do 6 kategorií. Tyto kategorie jsou označovány jako Proposed, Draft, Internet standards, Experimental, Informational a nebo Historic[6]. První tři se věnují novým protokolům a podléhají schvalovacímu řízení označeném jako standards track. Výsledkem toho řízení je následně dokument typu Internet standard. Experimentální dokumenty popisují důležité informace, které byli získány během vývoje nějaké technologie. Informativní dokumenty zato obsahují poznámky, které je vhodné zveřejnit v rámci řešení většího problému. RFC dokumenty patří do kategorie Historic, pokud byli nahrazeny aktuálnějším RFC dokumentem. Součástí těchto doporučení a standardů jsou zároveň standardy pro SNMP, kde ty nejdůležitější jsou popsány v tabulce 2.1.

2.1.2 Model manager-agent a bloková struktura SNMP

SNMP komunikace je založena na modelu manager-agent a podporuje přenos správ mezi správcem sítě tzv. managerem a agenty na jednotlivých zkoumaných zařízeních. Taková zařízení o sobě sdělují agentům základní vlastnosti a parametry. Tyto informace jsou následně uspořádány ve stromové struktuře MIB. Zjednodušený model takového systému je znázorněn na obrázku 2.1. V literatuře se pro pochopení činnosti uvádějí krom managerů a agentů další zařízení a bloky[2], které si následně popíšeme.



Obrázek 2.1: Klasifikace prvků v SNMP

•**Network elements** – Síťové elementy, někdy též označované jako managed devices jsou hardwarová zařízení jako počítače, routry, switche , která jsou připojená v síti.

•**Agents** – Agenti jsou softwarové moduly, které pracují nad síťovými elementy, kde shromažďují základní informace o daném zařízení a o síťovém provozu.

•**Managed devices** – Zařízení podporující SNMP, na kterými pracují agenti.

•**Structure of Management Information** (SMI) - SMI je soubor pravidel a specifikací pro popis objektů obsažených ve stromu MIB. SMI je nově definována v RFC 2578-2580 a pro její popis se používá jazyk ASN.1.

•**Network management stations**(NMSs) – NMSs jsou řídící stanice, které provádějí správu a kontrolu síťových elementů. NMS by měla být výkonově silnější pracovní stanice. A alespoň jedna NMS by měla být přítomna ve spravovaném prostředí.

•**Proxy agent** – Proxy agent je speciální typ agenta, kterému je umožněno sbírat informace o zařízeních, která nepodporují SNMP[7]. Proxy agent zároveň podporuje monitoring pasivních prvků počítačové sítě jako jsou HUBy, koncentrátory nebo opakovače. Informace o takovýchto zařízení jsou následně k nalezení MIB stromu daného proxy agenta.

•**Management protocol** – Protokol používaný pro přenos informací mezi NMS a agenty.

Na aktivní prvky a samotnou strukturu manager-agent je kladeno množství požadavků[7]. Tyto požadavky musí být zohledněny při implementaci agentů, managerů tak, aby aplikace využívající SNMP pracovali optimálně. V první řadě je nutné brát na vědomí, že cílem aktivních prvků není samotný SNMP protokol, ale zajištění optimální komunikace mezi síťovými elementy. Proto agenti neposkytují grafický interface a slouží jen pro sběr dat, nastavování základních parametrů a pro přenos informací. Následné zpracování získaných dat je řešeno managerem tak aby nedocházelo ke zbytečnému zatěžování síťových elementů. Proto by měl být agent nenáročný a jednoduchý proces, který bude mít minimální vliv na činnosti monitorovaného zařízení. Na druhou stranu samotný manažer je často server, který se přímo specializuje na zpracování SNMP, a proto mohou procesy měřící aplikace zatěžovat daný server mnohem výrazněji.

Dalším důležitým požadavkem je zajištění rychlé a nenáročné komunikace mezi agenty a managerem. Tak aby byla umožněna rychlá detekce případných problémů. Proto byl jako prostředek komunikace vybrán datagramový protokol. Výsledkem je sice vyšší rychlost komunikace, ale zato menší spolehlivost při přenosu dat. Pakety se ve výsledku nemusí doručit a nebo se mohou promíchat. Proto také většina aplikací, které využívají SNMP, nespoléhá jen na odposlouchávání trapů, ale periodicky se dotazuje (polling) na zkoumanou veličinu. Ale pokud síť obsahuje velké množství agentů pak je polling velmi náročný. Jedním z řešením je tzv. trap directed polling. V tomto řešení manager čeká na varovný trap od zařízení a následně se zaměří na problémové zařízení a nebo spustí polling. Ale i při tomto řešení musí administrátor a nebo vývojář aplikace přihlédnout k tomu, že datagramový paket nemusí k manažerovi dorazit a tedy ani upozornit na výskyt případného problému.

2.1.3 Struktura MIB a její zápis

Chceme-li získávat informace o daném zařízením, pak nejprve musíme znát uspořádání takovýchto informací. K tomu účelu SNMP agenti využívají MIB. MIB je soubor, který má hierarchickou stromovou strukturu a odpovídá danému zařízení. Základní struktura stromu je pro všechny výrobce jednotná, ale přes to umožňuje přidávání podstromů specifických pro dané zařízení a daného výrobce. Stromová struktura byla zvolena z toho důvodu, aby nedocházelo ke kolizím v situacích, kdy si výrobci definují své vlastní objekty. Pokud je totiž každému výrobci přidělen unikátní podstrom, pak nemůže docházet ke kolizím, jelikož případné stejně označené objekty budou mít odlišné fyzické umístění v MIB a tedy i jinou adresu.

K adresaci objektů se používá Object Identifier (dále jen OID). OID obsahuje indexy nebo názvy uzlů zaznamenaných na cestě mezi kořenem a cílovým listem hledaného objektu. Většina nejdůležitějších objektů je obsažena na adrese iso.org.dot.internet a v následujícím odstavci jsou popsány příslušné podstromy, které jsou zároveň zobrazeny na obrázku 2.2.

•**directory**(1) – rezervováno pro ISO.

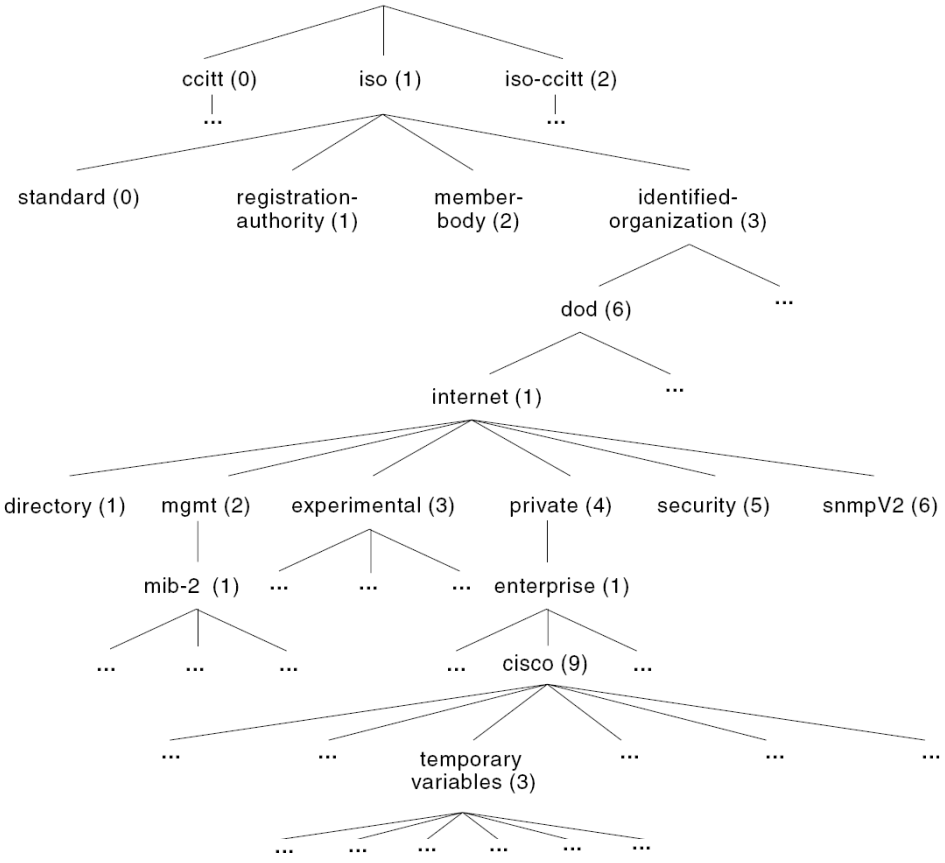
•**mgmt**(2) – obsahuje MIB objekty, které byly přímo definovány standardy vydanými IETF. Tyto objekty obsahují informace o běžných síťových zařízení a jsou respektovány většinou výrobců. Na druhou stranu některé objekty z mgmt nemusejí být vždy danými výrobci nastavovány. Budeme-li následně vytvářet monitorovací program pro počítačovou síť s aktivními prvky od různých výrobců, můžeme využít té skutečnosti, že topologie objektů v mgmt je pro taková zařízení stejná, ale musíme zároveň zohlednit, že všechny objekty v mgmt nemusí být plně podporovány a tedy ani naplněny daty.

•**experimental**(3) – obsahuje objekty, které jsou označené ve standardu jako vyvíjené.

•**private**(4) – privání MIB jednotlivých výrobců. Private je podstrom, který poskytuje prostor pro objekty, které si definuje sám výrobce zařízení. Například 1.3.6.1.4.1.9 reprezentuje podstrom k objektům pro zařízení firmy CISCO podobně, jak je znázorněno na obrázku 2.2[8].

•**security**(5) – rezervováno pro zabezpečovací funkce

•**snmpV2**(6) – definuje objekty využívané speciálně ve verzi 2.



Obrázek 2.2: Zobrazení stromové struktury MIB.

Jednotlivé objekty z MIB jsou popsány za pomoci jazyka ASN.1 a odpovídají vlastnostem zkoumaného zařízení. Na obrázku 2.3 je ukázána část definice objektu dot1Bridge. Definice takových objektu např. obsahuje:

- samotné jméno objektu
- OID objektu
- datový typ
- vymezení rozsahu hodnot
- formu přístupu k danému objektu (read-only, read-write, write-only, or not-accessible)
- status objektu (vyžadovaný, volitelný, zastaralý nebo (neschválený)
- textový popis objektu, nápovědu a poznámky.

```
BRIDGE-MIB DEFINITIONS ::= BEGIN

      IMPORTS
                Counter, TimeTicks   FROM RFC1155-SMI
                mib-2                FROM RFC1213-MIB
                OBJECT-TYPE          FROM RFC-1212
                TRAP-TYPE            FROM RFC-1215

      MacAddress ::= OCTET STRING (SIZE (6))
      BridgeId ::= OCTET STRING (SIZE (8))
      Timeout ::= INTEGER -- a STP timer in units of 1/100 seconds

      dot1dBridge   OBJECT IDENTIFIER ::= { mib-2 17 }

      -- groups in the Bridge MIB
      dot1dBase     OBJECT IDENTIFIER ::= { dot1dBridge 1 }
      dot1dStp      OBJECT IDENTIFIER ::= { dot1dBridge 2 }
      dot1dSr       OBJECT IDENTIFIER ::= { dot1dBridge 3 }
      dot1dTp       OBJECT IDENTIFIER ::= { dot1dBridge 4 }
      dot1dStatic   OBJECT IDENTIFIER ::= { dot1dBridge 5 }

      -- the dot1dBase group
      dot1dBaseBridgeAddress OBJECT-TYPE
          SYNTAX  MacAddress
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                  "The MAC address used by this bridge when it must
                  be referred to in a unique fashion.   It is
                  recommended that this be the numerically smallest
                  MAC address of all ports that belong to this
                  bridge.  However it is only required to be unique.
                  When concatenated with dot1dStpPriority a unique
                  BridgeIdentifier is formed which is used in the
                  Spanning Tree Protocol."
          REFERENCE
                  "IEEE 802.1D-1990: Sections 6.4.1.1.3 and 3.12.5"
          ::= { dot1dBase 1 }
        ...
END
```

Obrázek 2.3: Zobrazení objektu dot1dBridge zapsaného notací ASN.1.

2.1.4 Typy proměnných v SNMP

Typy SNMP objektů se v literatuře vesměs rozdělují do 3 kategorií: typy primitivní, typy definované a tabulky.

Primitivní typy:

•**Integer –** jednoduchá celočíselná hodnota. Ve většině implementací se tento typ omezuje na velikost 32 bitů, ačkoliv specifikace neurčuje horní limit[9].

•**Octet String** – sekvence bytů využívaná např. při popisu zařízení (sysDescr), pojmenování portů, MAC adresy atp..

•**Object Identifier** – využívá se pro označení jména objektu v MIB.

•**NULL**

Definované typy:

•**Counter** - nezáporný integer, který nabývá hodnot od nuly do 2^{32}-1. Proměnné typu counter se používají jako modulární čítače, které se po přetečení vynulují. Příkladem použití je například měření zatížení jednotlivých portů switche. Pro následné zpracování se místo absolutní hodnoty používá rozdíl hodnot čítače za určité období. Tak aby bylo možné čítač použít i pro velká zatížení a nebo pro měření delších časových intervalů, existuje zároveň odvozený typ Counter64[1], který nabývá hodnoty od 0 do 2^{64}-1.

•**Gauge** - nezáporný integer, který nabývá hodnot od nuly do 2^{32}-1. Gauge proměnná se od proměnné typu counter liší tím, že může pracovat i v opačném módu, kde se aktuální hodnota postupně zmenšuje.

•**TimeTicks** - nezáporný integer, zaznamenávající čas od určité události. Např měření doby od poslední detekované změny topologie počítačové sítě.

•**IpAddress** - 32 bitová IP adresa.

•**Opaque –** typ podporující zpětnou kompatibilitu vyšších verzí SNMP na SNMPv1 [1].

Jak již bylo výše zmíněno SNMP podporuje také tabulkové uspořádání objektů. Jednotlivé buňky tabulky jsou adresovány podobným způsobem jako skalární objekty. Rozdíl je v tom, že se k OID dané tabulky ještě připojí index příslušného sloupce a index příslušného řádku. Jedním z možných technik jak projít tabulku je postupné aplikování příkazu GET pro každou buňku tabulky. Pro použití takovéto techniky musíme znát OID adresy jednotlivých buněk. V některých případech, ale neznáme přesně indexy řádků, a proto je vhodné použít postupné volání příkazu GETNEXT, který nám umožní sekvenční průchod řádků takové tabulky. Dalším omezením je to, že se tabulky do sebe nemohou navzájem vnořovat, proto aby bylo možné jednotlivé buňky korektně a efektivně adresovat. Další technikou je použití příkazu GETBULK, který je ale podporován až od SNMPv2. Tyto SNMP příkazy budou popsány v další kapitole.

2.1.5 SNMP operace

Již od SNMPv1 a standardu RFC1157 jsou podporovány některé základní příkazy sloužící k asynchronní komunikaci se SNMP agentem. Aby daný protokol byl jednoduchý a přehledný je definováno jen minimální množství funkcí, které jsou popsány v následujícím odstavci. Jelikož příkazy GET a GETNEXT se nejevili efektivní při práci s tabulkami, tak od SNMPv2 se definoval příkaz GETBULK, který je při práci s tabulkami efektivnější. Ale budeme-li vytvářet monitorovací aplikaci využívající informace z tabulek, pak musíme zohlednit, že řada síťových elementů nemusí SNMPv2 podporovat a tedy ani odpovídat na GETBULK.

•**GET REQUEST** – umožňuje přečtení objektů z MIB. Aby agent na požadavek reagoval, musí zadané OID existovat v jeho MIB a čtený objekt musí mít samotné čtení umožněno v přístupových právech na read-only nebo read-write.

•**GETNEXT REQUEST** – umožňuje sekvenční čtení objektů z MIB. Vrátí objekt, který se jako první vyskytuje v MIB souboru od zadaného OID a má nastavená přístupová práva na read-only nebo read-write.

•**SET REQUEST** - umožňuje nastavení hodnot objektů v MIB, které mají nastavena přístupová práva na write-only nebo read-write.

•**TRAP** – je asynchronní zpráva, kterou agent posílá managerovi při detekci důležité události (např.: coldStart, linkDown, linkUp, authenticationFailure, risingAlarm, fallingAlarm, newRoot, topologyChange, warmStart) .

•**GETBULK REQUEST** – umožňuje rychlejší načtení velkého množství údajů z tabulek naproti příkazů GET a GETNEXT. GETBULK je podporován od SNMPv2.

•**INFORM –** příkaz podporován od SNMPv2 a pracující podobně jako TRAP, ale vyžadující potvrzení přijetí zprávy[2].

Pro komunikaci se v SNMP využívá UDP port 161 pro agenta a UDP port 162 pro manažera. Manager může poslat požadavek z libovolného dostupného portu na port 161 daného agent. A agent může odpovědět zpět na zdrojový port žádosti. Manažer přijímá trapy na portu 162 a agent může generovat trapy z libovolného dostupného portu.

2.1.6 Formát SNMP zpráv

Formát SNMP zprávy je zobrazen na obrázku 2.4. a obsahuje dvě hlavní části: hlavičku zprávy a Protocol Data Unit (PDU). Hlavička obsahuje verzi SNMP a community name(community string) pro potřeby autorizace. PDU obsahuje informaci o typu příkazu a seznam OID, kterých se daná komunikace týká.

SNMP zpráva obsahuje dále tyto položky:

•**Request ID –** identifikační číslo, které je shodné pro dotaz i odpověď. Použijeme-li stanici s multihread monitoringem pak nám requestID umožní paket odpovídajícího agenta přiřadit správnému vláknu.

•**Error status** – obsahuje typ případné chyby.

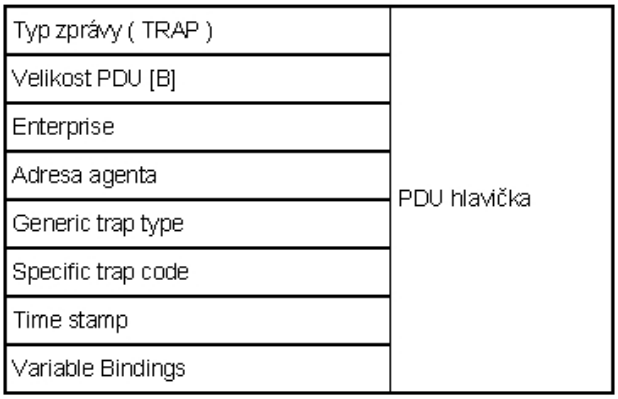
•**Error index** – určuje místo výskytu chyby, resp. ke kterému Variable binding v dotazu se chyba vztahuje. Tento index je zároveň zdrojem případných problémů. Ptáme-li se totiž v dotazu na více OID položek a pokud se vyskytne chyba při odpovědi na jedno OID. Pak není zaručeno, že odpovědi na následující OID budou doručeny. Např. vtpVlanName a dot1qVlanStaticName nám vrací jméno VLANy. VtpVlanName je podporovaný jen zařízením od výrobce CISCO. A pokud bychom chtěli v rámci jednoho dotazu použít „pro jistotu“ obě OID, pak „neCISCO zařízení“ nám vůbec nevrátí název VLANy, protože pří použití vtpVlanName na „neCISCO zařízeních“ se vygeneruje chyba a na druhé OID dot1qVlanStaticName nám pak zařízení již nemusí odpovědět.

•**Variable bindings** - obsahuje vlastní data SNMP PDU.

<table>
<tr><td colspan="2">Délka zprávy [B]</td><td rowspan="3">Hlavička SNMP-zprávy</td></tr>
<tr><td colspan="2">Číslo verze</td></tr>
<tr><td colspan="2">Comunity String</td></tr>
<tr><td>Typ zprávy(get, getnext, ...)</td><td rowspan="5">PDU hlavička</td><td rowspan="10">PDU (Protocol Data Unit)</td></tr>
<tr><td>Velikost PDU [B]</td></tr>
<tr><td>RequestID</td></tr>
<tr><td>Errorstatus</td></tr>
<tr><td>Error index</td></tr>
<tr><td>Velikost těla PDU [B]</td><td rowspan="5">PDU tělo dotazu</td></tr>
<tr><td>VariableBinding 1</td></tr>
<tr><td>VariableBinding 2</td></tr>
<tr><td>. . .</td></tr>
<tr><td>VariableBinding n</td></tr>
</table>

Obrázek 2.4: Struktura SNMP zprávy.

Na rozdíl od jednoduché zprávy, která byla popsána v předchozím odstavci, samotný trap obsahuje položky zobrazené na obrázku 2.5.



Obrázek 2.5: Znázornění struktury SNMP trapu.

SNMPv1 trap obsahuje tyto položky, u kterých se často v literatuře chybně uvádí[10], že mají jiný význam než ve skutečnosti:

•**Enterprise OID –** OID objektu, který vygeneroval trap.

•**Agent address** - je zdrojová adresa agenta, ze které přišel trap.

•**Generic trap type, specific trap code** – Standart definuje 7 trapů(SNMPv1), které mohou být generovány agentem SNMPv1. Šest z nich je generických a sedmý enterprise je určen pro použití privátních organizací při definování jejich vlastních specifických trapů. Pokud generic trap type je nastaven na enterprise pak specific trap je nenulový a udává typ trapu, definovaný výrobcem.

•**Time stamp –** udává množství času, které uplynulo mezi posledním startem agenta a odesláním trapu.

•**Variable bindings** - obsahuje OID objektů, které mají vztah k danému trapu .

2.1.7 Vývoj SNMP

Další vývoj protokolu SNMP byl předurčen nedostatky, které obsahovala první verze tohoto protokolu (dále jen SNMPv1).

•Prvořadou nevýhodou SNMPv1 je nedostatečné řešení autorizace a nedostatečné zabezpečení komunikace. Při autorizaci se používá Comunity String, přístupové heslo které je v nezašifrované podobě součástí paketů. Zároveň samotná komunikace probíhá taktéž v nezašifrované podobě.

•Při čtení více objektů z MIB stromu se musí jednotlivé položky číst postupně, protože příkazy GET a GETNEXT neumožňují číst celé bloky. Výsledkem je, že čtení objektů jako je tabulka je časově náročnější a více zatěžuje počítačovou síť.

•SNMPv1 nepodporuje komunikaci mezi managery a tím tedy omezuje distribuci výpočtů mezi těmito managery. Ve výsledku není vhodné použít SNMP na větší počítačové sítě.

•Při odchytávání trapů manager nepotvrzuje přijetí zprávy. Tj. může se stát, že datagram trapu se nedoručí a manager ani agent to nezjistí.

V rámci následujících kapitol si popíšeme některá řešení problémů SNMPv1.

2.1.7.1 Remote Monitoring

SNMP je jedním z nejpoužívanějších protokolů pro správu sítí, který poskytuje aktuální informace o zařízeních, na kterých běží příslušný agent. Agent neustále aktualizuje proměnlivé údaje jako např. počet příchozích a odchozích paketů na konkrétních portech a informace poskytuje žadatelovi. Nevýhodou je, že agent neuchovává přepisované hodnoty a tedy ani neumožňuje historickou analýzu průběhu takovýchto veličin. Abychom získali časový průběh takových veličin museli bychom se neustále dotazovat na aktuální stav dané hodnoty příkazem GET, GETNEXT nebo BULK a provádět polling, který byl zmíněn v předchozích kapitolách. Tímto způsobem můžeme měřit vývoj sledovaných veličin jako např. zatížení jednotlivých portů. Nevýhodou takového přístupu je nezanedbatelné zatížení zkoumané sítě i managerové stanice. Jako jedním z řešení se ukázalo využití Remote Monitoringu (dále jen RMON).

RMON byl navržen v RFC 1271 a později schválen jako standart RFC 1757. Jelikož původní návrh RFC 1271 obsahoval malou podporu pro sítě typu TokenRing tak vznikl standart RFC 1513, který podporu těchto sítí rozšiřuje. RMON je jedním z rozšíření MIB(viz obrázek 2.6), které umožňuje zaznamenávat a uchovávat historické hodnoty proměnných(objektů). Takovéto hodnoty je pak možné vizualizovat a zpracovávat bez toho aniž by bylo nutné výrazněji zatěžovat počítačovou síť a manažerovou stanici. RMON je proto výhodné použít ve velkých sítích a tam kde se vyžaduje velká frekvence měření jednotlivých proměnných. RMON podobně jako SNMP pracuje na dvousložkovém modelu, kde řídící aplikace nastavuje agenta, tak aby měřil požadované hodnoty a agent shromažďuje informace z kooperujícího síťového elementu.

Skupiny objektů obsažených v RMON :

•**statistics** – obsahuje objekty, které jsou měřitelné pro všechny interface na zařízení.

•**history –** umožňuje vytvářet pravidelné vzorky, které jsou ukládány pro pozdější analýzu. Podporuje nastavení intervalu měření a nastavení velikosti vzorku.

•**alarm –** Měřené veličiny jsou porovnávány s prahovými hodnotami. Pokud jsou prahové hodnoty generuje se událost. Prahové hodnoty i intervaly měření jsou nastavitelné.

•**hosts –** vytváří komunikační statistiky pro každý síťový uzel reprezentovaný MAC adresou.

•**hostTopN –** rozšiřuje předchozí objekty, tak aby bylo možné hosty setřídit podle zvolené statistiky. Výhodnost takového rozšíření je zjevná v případech, kdy chceme vybrat několik hostů, které nejvíce zatěžují počítačovou sít a nechceme přitom odesílat kompletní seznam

hostů.

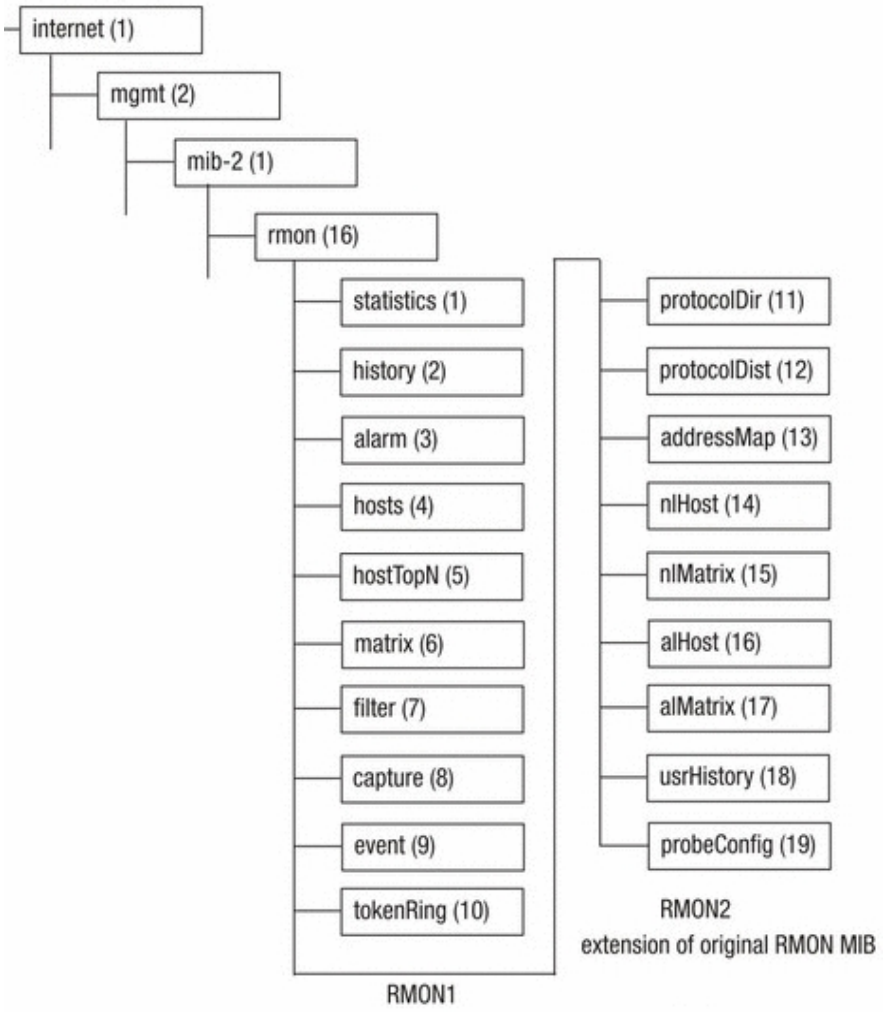
•**matrix** – udržuje statistické informace o komunikaci mezi dvěma uzly.

•**filter** – umožňuje vytvořit omezení pro výběr paketů určených k měření. Ve výsledku se mohou zaznamenávat jen pakety, které patří do určitých sad definovaných daným omezením.

•**capture** – umožňuje zachytávat pakety, na které se nevztahuje definované omezení.

•**event** – zaznamenává jednotlivé události.

•**tokenRing** – podpora pro vytváření statistik na TokenRing sítích.



Obrázek 2.6: RMON1 a RMON2 v MIB stromu na adrese 1.3.6.1.2.1.16 .

RMON se pro svoje výhody stal široce podporovaným rozšířením, které umožnilo rychlejší vyhledávání problémových míst a zefektivnilo správu síťových prvků. Jedním z nedostatků agentů podporující RMON je to, že dokáží monitorovat jen oblast svého síťového segmentu, jelikož takový agent zpracovává data z fyzické a linkové vrstvy. RMON agent tedy nemůže např. monitorovat zatěžování portů jednotlivými aplikacemi, které běží na koncových stanicích zákazníků. Proto časem vznikla poptávka na to, aby se monitorovací možnosti RMONu rozšířili i na vyšší vrstvy modelu ISO/OSI. Jako odpověď vznikl standart RMON2 [11], který podporuje protokoly pracující ve vyšších vrstvách modelu ISO/OSI. RMON2

umožňuje detailnější analýzu síťové komunikace např. umožňuje detekovat cílovou a zdrojovou IP adresu i port komunikující aplikace. Výsledkem je možnost zjišťovat základní statistiky o komunikujících uživatelích a jejich aplikacích. RMON2 již tedy není jen zaměřen na detekci problémů na síti, ale spíše na optimalizaci výkonnostních parametrů sítě.

RMON2 zároveň rozšiřuje MIB o další skupiny objektů :

- **protocolDir** – skupina určená pro konfiguraci měřených protokolů.
- **protocolDist –** souhrnné statistiky o výši provozu generovaného každým protokolem.
- **addressMap** – přiřazení MAC adresy k IP adrese.
- **nlHost** – dopravní statistiky na síťové vrstvě pro konkrétní jeden uzel.
- **nlMatrix** – dopravní statistiky na síťové vrstvě pro konkrétní pár uzlů.
- **alHost** – dopravní statistiky na aplikační vrstvě pro konkrétní jeden uzel.
- **alMatrix** – dopravní statistiky na aplikační vrstvě pro konkrétní pár uzlů.
- **userHistory** – uživatelem definované měřící vzorky a definovanou periodou měření.
- **probeConfig** – definuje standardní konfigurace parametrů pro RMON sondy.

2.1.7.2 Další verze SNMP

Jiným řešením nedostatků SNMPv1 bylo vytvoření nového standardu SNMPv2 (RFC 1441-1452). Tento standard se vyznačoval zvýšenou mírou zabezpečení, podporu komunikace mezi managery i lepší podporou zpracování obsáhlejších dat použitím příkazu GETBULK. Výsledné řešení bezpečnosti bylo ale velmi složité [12] a zapříčinilo vznik dalších upravených verzí SNMPv2.

Jedním z nich byla i verze SNMPv2c (RFC 1901-1908), která obsahovala vylepšení zmíněné verze 2 včetně podpory příkazu GETBULK pro zvýšení výkonnosti komunikace, ale zároveň řešila zabezpečení stejným způsobem jako původní SNMPv1 a postrádala podporu komunikace mezi managery.

Další rozšířením verze SNMPv2 je i SNMPv2u (RFC 1909-1910). SNMPv2u podporuje odlišný způsob zabezpečení oproti předchozím verzím. Toto zabezpečení je kvalitnější oproti SNMPv1, ale jednoduší oproti SNMPv2c.

Jelikož ani SNMPv2c nepodporoval šifrování ani autentizaci, tak vznikla další verze protokolu SNMP. SNMPv3 je definován standardy RFC 3411-3418, které byli vydány v roce 2002. SNMPv3 je podobný protokolu SNMPv2 rozšířeném o podporu vypracovanější formy administrace a bezpečnější komunikace, autorizace a autentizace.

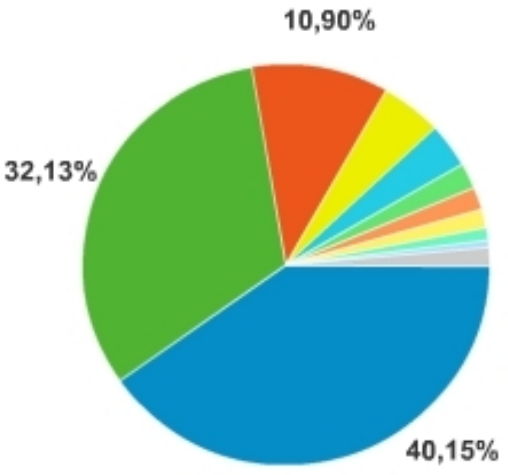
Jelikož cílem této práce není detailní popis těchto nových verzí SNMP. A protože ve výsledné aplikaci si vystačíme se samotným SNMPv1, odkážeme zájemce hlubší analýzy na externí zdroje informací např [13].

2.2 Flash a RIA

Adobe Flash dříve nazývaný Shockwave Flash nebo Macromedia Flash je sadou multimediálních technologií vyvíjenou firmou Adobe Systems a vytvořenou firmou Macromedia[14]. Od uvedením na trh v roce 1996 se Flash stal populárním prostředkem pro vytváření interaktivních animací, transparentních webových doplňků, reklam a her obsahující vektorovou i rastrovou grafiku tak i zvukové soubory. Nezanedbatelně Flash podporuje progresivní download i streaming videa, tvorbu 3D scén i jednoduchých virtuálních světů. A v neposlední řadě je Flash určen k podpoře RIA aplikací tj. k vývoji webových stránek jenž se svojí funkcionalitou blíží klasickým desktopovým aplikacím[15]. K vytváření aplikací Flash využívá vypracovaný systém víceúrovňových časových os a objektově orientovaný skriptovací jazyk nazývaný actionscript, který je postavený z mezinárodně standardizovaného ECMAScriptu.

Flash je možné použít všude tam, kam je možné naimportovat zásuvný modul, který interpretuje příslušný bytekód Flashe. Soubor bytekódu se označuje jako „Shockwave Flash file“ (SWF) a jeho interpret jako „Flash Player“. Flash Player existuje pro většinu webových klientů a jejich platforem u základních operačních systémů, což zajišťuje lepší platformní nezávislost než u SilverLightu[16]. Spolu s počáteční podporou společnosti Microsoft, který zahrnul Flash Player 4.0 do Microsoft Internet Exploleru 5.0 se stal Flash jednou z nejpoužívanějších multimediálních technologii vůbec[17] viz obrázek 2.7. Od roku 2004 se vývojáři Flashe začali zajímat o využití Flashe pro mobilní zařízení. Výsledkem byla nova forma Flash Playeru pro mobilní zařízení jenž se označuje jako Flash Lite, která ale není prozatím výrazněji využívána[16]. Další nádhernou možností je využíti Flashe pro tvorbu dynamických a interaktivních textur, jenž se následně importují do 3D aplikací. Tuto schopnost podporuje například VRML klient Cortona od společnosti ParallelGraphics. Nezanedbatelně se také Flash používá při vytváření firemních prezentací a grafického rozhraní DVD authoringu.

1.	10.0 r12	3 007	40,15%
2.	9.0 r124	2 406	32,13%
3.	9.0 r115	816	10,90%
4.	9.0 r47	363	4,85%
5.	9.0 r28	269	3,59%
6.	(not set)	155	2,07%
7.	9.0 r16	137	1,83%
8.	9.0 r45	113	1,51%
9.	6.0 r88	75	1,00%
10.	8.0 r22	44	0,59%

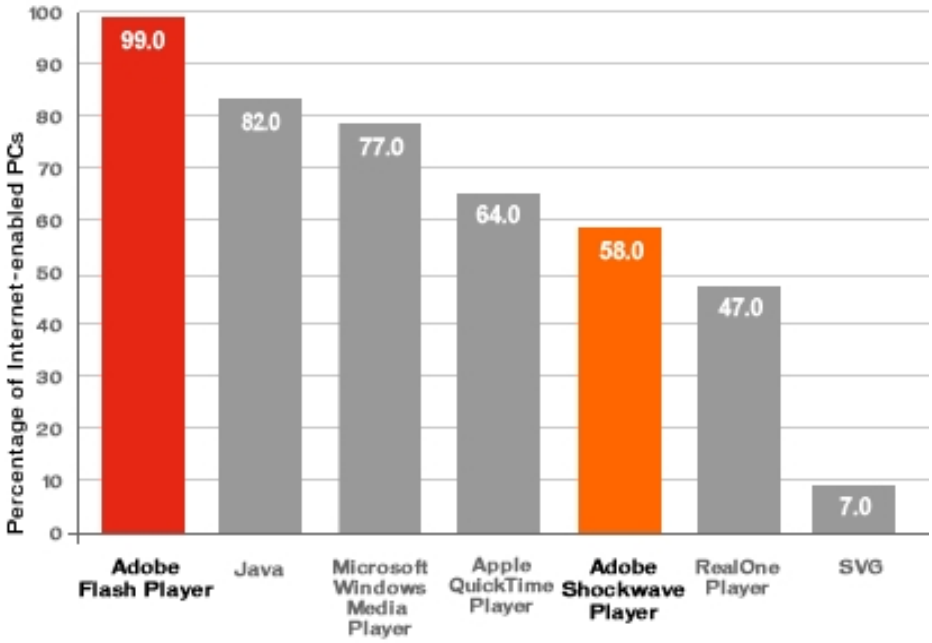


Obrázek 2.7: Míra výskytu Flash Playeru.

Na obrázku 2.7 je zobrazena míra využívání verze Flash Playeru. Z obrázku je patrné, že více jak 94% uživatelů používá verzi 9.0 nebo 10.0 a u více jak 2% uživatelů nebyl plugin zjištěn. Data byla měřena za pomoci Google Analytics v českém internetovém obchodě v prosinci roku 2008 u návštěvníků, kteří měli povolený javascript. Za toto období zaznamenal obchod 7489 unikátních návštěv. V prvním sloupci

tabulky je uvedeno pořadí, ve druhém typ pluginu a v následném sloupci kolikrát byl zaznamenán výskyt daného pluginu na webovém prohlížeči příchozích uživatelů.

Za konkurenty Adobe Flash se dnes považují technologie[18] SVG, JavaFX, AJAX a SilverLight viz. obrázek 2.8[19]. Tyto technologie spolu s produkty výrazněji podporující Flash jako například Flex, Breeze, AIR ovšem nejsou náplní této zprávy. Jen pro zajímavost uvedu, že za největší konkurenci Flashe je považován SilverLight podporovaný společností Microsoft a umožňující snadnější indexování než byte kód SWF. SilverLight byl vyvinut firmou Microsoft roku 2006 a vychází z technologie formátu XAML.



Obrázek 2.8: Výskyt multimediálních systému na trhu v období listopadu 2008.

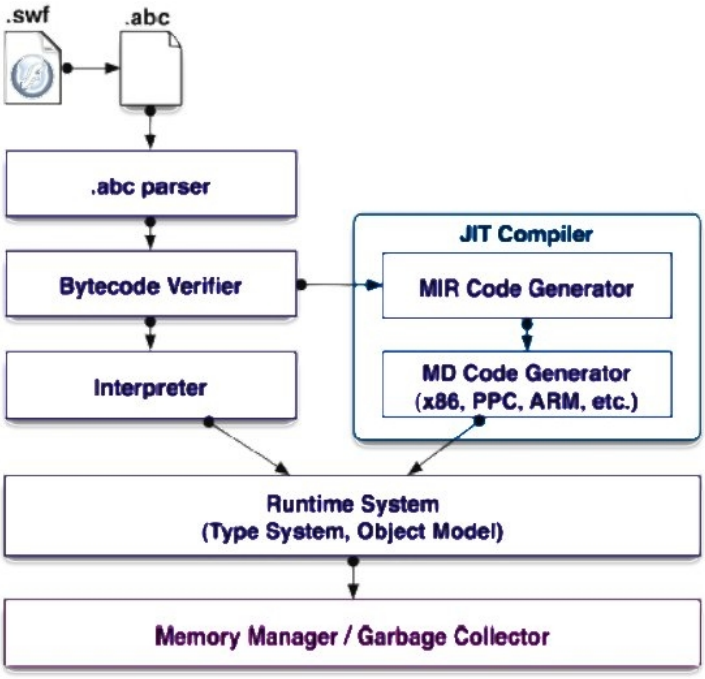
2.2.1 Vznik a vývoj technologie Flash

U zrodu Flashe stál Jonathan Gay, který spolu s Charlie Jacksonem a Michelle Welshem založili v lednu roku 1993 malou společnost s názvem FutureWave Software. Jejich první produkt SmartSketch by vyvinutý pro operační sytém PenPoint a později upraven pro využití na operačním systému Windows a Macintosh. SmartSketch umožňoval vytvářet rastrovové, vektorové obrázky a jednoduché animace. Záhy po zavedení SmartSketche na trh se začalo ukazovat, že by se vývoj dalšího produktu měl více směrovat na animace a na prostředí rychle rostoucího nového média Internetu. FutureSplash Animator vznikl v roce 1996 a od svého předchůdce se lišil hlavně ve vypracovanější tvorbě animací za pomocí časové osy. FutureSplash Animator[20] byl podporován, v té době dominantním, webovým klientem Netscape a jeho animace se začali objevovat na webových stránkách populárních produktů zábavního průmyslu. V literatuře jsou často zmiňovány webové stánky kultovního seriálu The Simpsons a stránky společnosti Walt Disney[20]. V prosinci roku 1996 byl FutureWave Software koupen společností Macromedia a FutureSplash Animator byl přejmenován na

Macromedia Flash 1.0. A ať již z komerčních nebo technologických důvodů je téměř každý rok publikována jedna verze Flash Playeru a upravené vývojové prostředí, jenž daný Flash Player využívá. V následující odstavcích si některé z nich popíšeme[14].

- **Macromedia Flash 2.0** (1997) - zabudována podpora stereo zvuku, vytvořena knihovna pro ukládání grafických objektů, podpora prvních příkazů pro řízení animací jako gotoAndPlay, gotoAndStop, nextFrame a nextScene.
- **Macromedia Flash 3.0** (1998) - první verze ActionScriptu[21], který umožňuje vytvářet složité nedeterministické víceúrovňové animace a vytvářet grafické elementy dynamicky bez použití časové osy a kreslících nástrojů.
- **Macromedia Flash 4.0** (1999) – rozšíření o podporu MP3 streamingu a vytváření Motion Tween animací, jenž umožňují z počátečních a koncových podmínek vypočítat příslušné transformace pro jednotlivé rámce. Díky zahrnutí Flash Playeru 4 do Microsoft Internet Explorer 5.0 a následné podpoře ve WindowsXP bylo zaznamenáno vice jak 100 miliónů[14] instalací Flash Playeru 4.0.
- **Macromedia Flash 5.0** (2001) – pro zlepšení komunikace mezi serverem byla do Flasee doplněna podpora pro Flash Remoting a vznikl server ColdFusion.
- **Macromedia Flash MX** (Macromedia Flash 6) (2002) – obsahuje podporu pro přehrávání videa a vytváření i sdílení komponent. Komponenty jsou programové třídy, které mohou obsahovat grafické uživatelské rozhraní . Komponenty a samotný objektový model actionscriptu umožnil otevřít Flash dalším vývojářům. Výsledkem je značný růst počtu šablon a zrychlení vývoje robustních aplikací.
- **Macromedia Flash MX 2004** (Macromedia Flash 7) (2003-2004) - Předchozí verze byla doplněna o řadu UI komponent, o komponenty rozšiřující možnosti komunikace se servery, o komponenty pracující s daty i XML dokumenty. Nemalý význam má i zabudování podpory webových služeb a podpory pro XPath. Změny nastaly i u skriptovacího jazyku, jenž dostal výraznější objektový ráz a byl přejmenován na ActionScript 2. Zrodil se Flash Lite1.1 pro mobilní zařízení, Flex 1.0 což je obdoba Flashe určená k vývoji programátorsky obtížnějších aplikací a Breeze 1.0 (Adobe Acrobat Connect) aplikace pro podporu webových konferencí. Flex 1.0 byl primárně určen pro vývoj RIA aplikací, ale pro svou vysokou cenu nebyl moc úspěšný.
- **Macromedia Flash 8** (2005) - Zakomponování kodeku On2 VP6 s lepším kompresním poměrem[22] a s podporou alfa kanálů umožnilo vytvářet nádherné prezentace složené z několika navzájem provázaných vrstev obsahující různá videa. Další novinkou je využití filtrů pro tvoru některých komplexnějších animací.
- **Adobe Flash 9** (2006), CS3(2007) – Po odkoupení společnosti Macromedia společností Adobe Systems se změnil označení Macromedia Flash na Adobe Flash. Nasazením Flash Playeru 9 nastalo výrazně zvýšení výkonnosti což bylo zapříčiněno použitím výkonnějšího interpreta byte kódu tzv. „just in time“ a použitím optimalizovaného virtuálního stroje „ActionScript Virtual Machine 2“ (dále jen AVM2) popsaného na obr 2.9. Zároveň došlo k znatelnějšímu oddělení Flashe od editoru Flex a to vydáním Flex 2.0. Aby se usnadnil vývoj a zvětšila vývojářská komunita Flexu došlo k uvolnění SDK spolu s frameworkem a kompilátorem pro příkazovou řádku. Někteří autoři uvádějí, že se tímto způsobem snaží Adobe Systems vytvořit silnou vývojářskou komunitu a zároveň se připravit na konfrontaci se SilverLightem od společnosti Microsoft[24].

•**Adobe Flash 10** (2008) – Ze stránek společnosti Adobe plyne, že obsahuje výraznější podporu 3D prostoru[26]. Např.: možnost používat 3D základní elementy, 3D lineární transformace, inverzní kinematiku atp. Nová verze zároveň podporuje další dynamické transformační techniky rastrové grafiky jako je PixelBender. PixelBender je jazyk pro vytváření platformě nezávislých filtrů, které je možné uplatnit na celé obrázky, textové objekty i samotná videa. Zároveň došlo k rozšíření Sound API o možnosti výraznějších transformací nad zvukovou stopou[26].



Obrázek 2.9: Dvouúrovňového zpracování byte kódu AVM2

Obrázek 2.9. Znázorňuje dvouúrovňového zpracování byte kódu pomocí AVM2 [25]. V prvním kroku parser rozdělí bytecode do skupin podle funkcionality. V dalším kroku verifikační proces zkontroluje strukturální integritu, typovou korektnost a postupně předává Intermediate Representation (dále jen IR) do JIT. JIT přeloží dvoustupňově bytecode do nativního kódu. Interpret umožňuje zpracovat bytocode přímo, pokud je například daná subrutina v kodu uvedena jen jednou může být Interpret rychlejší než JIT. Výsledný kód je následně prováděn s podporou dynamické dealokace paměti a případného debugingu.

2.2.2 Komunikační možnosti technologie Flash

Ať již pro vytváření RIA aplikací a nebo přímo pro tvorbu virtuálních světů vytvoření s pomocí enginů Away3D, papervision3D, flashsandy, popřípadě ve Flash Playeru 10, je značně důležitá možnost Flashe dorozumívat s webovými servery. Z toho důvodu si popíšeme základní možnosti komunikace v následných odstavcích. Podrobnější popis a ukázky takovéto komunikace pro různé webové servery je k nalezení např. v publikaci Nate Weisse[27].

2.2.2.1 Parametry a FlashVars

První a nejednoduší cestou komunikace je statické předání informací přes atributy HTML tagu. Na internetu se objevuje velké množství způsobů zakomponování Flashe do HTML. Ale všechny by měli umožňovat správnou funkci Flashe ve všech prohlížečích, neměly by být zcela závislé na JavaScriptu, zajistit korektní předání parametrů, pokud možno být validní a při chybějícím pluginu umožnit zobrazení alternativního obsahu. Na obrázku 2.10 je uvedeno klasické validní přiložení SWF souboru do HTML, které má ovšem tu nevýhodu, že neošetřuje problém některých verzí Internet Exploreru, které podmiňují spouštění Flashe uživatelovým poklepáním na zvýrazněný okraj Flashe [28,29]. Tento problém byl řešen nasazením dodatečných externích skriptů podobně jako je uvedeno na obrázku 2.11., ale tento způsob vkládání SWF souboru není již nezbytný, jelikož Microsoft záplatou KB 945007 problém s importováním Flashe vyřešil[30].

```
<object type="application/x-shockwave-flash" data="flvplayer.swf" width="160" height="160">
  <param name="movie" value="flvplayer.swf" />
  <param name="flashvars"  value="image=view.jpg,file=video.flv">
  <p>
   <img src="view.jpg" alt="view" title="view">
   <br>Macromedia Flash Player is not ready
  </p>
 </object>
```

Obrázek 2.10: Doporučený kód pro přiložení SWF souboru k HTML stránce 1.

```
<script type="text/javascript" >
AC_FL_RunContent('width','480','height','360','movie','flvplayer','flashvars','image=view.jpg&file=video.flv');
</script>
<noscript>
<object data="flvplayer.swf" . . . width="480" height="360">
    <param name="movie" value="flvplayer.swf" />
    <param name="flashvars" value="image=view.jpg,file=video.flv" />
    <embed src="flvplayer.swf" flashvars="image=view.jpg,file=video.flv"
     width="480" height="360" type="application/x-shockwave-flash">
    </embed>
 </object>
</noscript>
```

Obrázek 2.11: Doporučený kód pro přiložení SWF souboru k HTML stránce 2.

Kód zobrazený na obrázku 2.10 ukazuje možné přiložení přehravače flv souborů do HTML prezentace. Parametr FlashVars obsahuje odkaz na flv soubor videa a obrázkový náhled. Kuli zamezení automatického stahovaní filmů ze většinou neuvádí přesná adresa videa, ale výsledek hašovací funkce. Součástí kódu je také text, který se zobrazí při neexistenci Flash Playeru. Za to kód na obrázku2.11 obsahuje javascript doporučený společností Adobe Systém[31] pro přiložení SWF souboru k HTML stránce.

Parametry, které může nastavit webový server při vytváření HTML dokumentu, výrazně ovlivňují zpracování animace Flash Playerem, proto si uvedeme význam některých z nich. Zásadní význam má hlavně parametr FlashVars, který je primárně určen pro předávání programátorem definovaných atributů Flash Playeru, který bude zpracovávat přiložený SWF soubor. Tento typ využití se často používá při inicializaci náročnější komunikace. Jako například když server nastaví do FlashVars adresu videa, které se bude následně přehrávat. Zajímavé využití je při tvorbě dynamicky generovaných obrázků, kde místo pomalejšího vytvoření obrázku přímo na straně serveru se nastaví parametry Flashové animaci. Například pokud chceme zobrazit obrázek teploměru, který bude obsahovat naměřenou hodnotu z teplotního čidla. V tomto případě můžeme například obrázky vytvářet na straně serveru pro každou teplotu jiný obrázek a nebo si vytvořit částečně transparentní, několikavrstvou animaci ve Flashi, která bude teplotní stupnici posouvat podle hodnoty zadaného parametru.

•**allowfullscreen**(true,false) – pokud je nastaven na false pak není možné nastavit Flash do fullscreen modu. Tento parametr je užitečný u serveru, kde se prezentuji takové animace, u kterých není zaručená jejich korektnost. Abychom zabránili obtěžování zákazníka náhle spouštěným fullscreen modem nastavíme allowfullscreen na false.

•**allowscriptaccess**(never,always,sameDomain) – omezuje resp. povoluje možnou komunikaci mezi javascriptem a animaci. Defaultně je nastaven sameDomain, který umožní volání javascriptu jen pokud adresa domény SWF souboru se shoduje s doménou webové stránky [17]. Volání javasriptu se prování pomocí příkazů fscommand(), getURL("javascript:...") a metod třídy ExternalInterface. Této vlastnosti se např. používá na herních serverech, aby se zabránilo manipulacím Flashové animace s dalším obsahem webové stránky.

•**bgcolor** – nastavuje barvu pozadí animace.

•**devicefont**(true,false)– povoluje použití fontu zařízení ve Flashi.

•**flashvars** – parametr určený pro korektní předávání parametrů.

•**height, width** - nastavení výšky, šířky animace.

•**loop** – parametr určuje má-li se animace po ukončení znovu spustit.

•**menu**(true,false)- určuje zda se uživateli zobrazí kontextová nabídka, když uživatel klikne na animaci pravým tlačítkem myši.

•**movie/src** –určuje adresu umístění animace.

•**play**- udává jestli se má animace spustit automaticky a nebo jestli má počkat na spuštění scriptem.

•**quality**(hight,best)-nastavení kvality přehrávání.

•**wmode**(transparent,opaque)- velice důležitý parametr, jenž určuje zda animace může být průhledná. Nastavením wmode na transparent můžeme naše webové projekty obohatit o částečně průhledné animace. Wmode je také využíván pro načítání vrstvených PNG (Portable Network Graphics) obrázků, protože webový klienti mají často problém s korektním zobrazením překrývaných, částečně transparentních obrázků. Wmode je přednastaven na opaque tj. animace je nastavena jako neprůhledná. Častý problém, jenž se řeší v internetových diskuzí je jak zajistit korektní funkčnost všech formulářových prvků, odkazů a javascriptů, které se nachází pod transparentní animací a to pro všechny webové klienty. Tento problém je stále otevřený.

2.2.2.2 LoadVars

Třída LoadVars je jedním ze základních objektů v actionscriptu, jenž slouží ke komunikaci se vzdálenými objekty. LoadVars i jeho potomci používají soubor několika základních metod, funkcí a parametrů jejichž použití je ukázáno na obrázku 2.12. Základní funkce a objekty jsou popsány v následujícím odstavci.

- **getBytesLoaded -** metoda vracející počet nahraných bytů ze serveru.
- **getBytesTotal** - metoda vracející celkový počet bajtů, které celkem vrátí server během operace load nebo sendAndLoad. Poměr mezi getBytesLoaded a getBytesTotal se často využívá pro vytváření preloaderů, měřící kolik procent potřebných dat se již stáhlo.
- **load** (URL) **-** metoda , jenž zahájí získávání dat ze serveru zadaného adresou URL
- **send** (URL, [tergetWin,method]) - metoda, jenž nasměruje prohlížeč na novou URL, součástí tohoto procesu je odeslání proměnných metodou POST nebo GET
- **sendAndLoad** (URL, [method]) - metoda při níž jsou data poslány na webový server bez toho aniž by prohlížeč byl přesměrován na jinou stránku
- **onData -** událost jenž nastane po kompletním načtení dat, ale ještě před jejich rozdělováním na dvojice název/hodnota
- **onLoad -** událost jenž nastane po načtení dat a po jejich rozdělení na dvojice název/hodnota

```
var sendbutton:Button=new Button();
sendbutton.onPress = function() {
        var MyLoader:LoadVars = new LoadVars();
        MyLoader.onLoad = function(success:Boolean) {
                if (success) {
gotoAndStop(10);
                                forum.text = MyLoader.allitems;}
                else {forum.text = "Chyba aplikace";}
                };
        var MySender:LoadVars = new LoadVars();
        MySender.jmeno                  = form.name.text;
        MySender.vzkaz          = form.question.text;
        MySender.email          = form.email.text;
        MySender.url            = form.urls.text;
        MySender.sendAndLoad("forum.php", MyLoader, "POST");
        };
```

Obrázek 2.12: Příklad použití LoadVars.

Actionscript kód na obrázku 2.12 prezentuje ukázku použití LoadVars objektu pro odesílání dotazů v diskusním fóru. Na začátku je k vytvořenému tlačítku zaregistrována funkce, jenž se provede po stisknutí tlačítka. V této funkci se vytvoří instance objektu LoadVars, nazvěme ji MySender, která naplní svoje vnitřní proměnné a ty odešle funkcí sendAndLoad na forum.php. Součástí funkce, která se zavolá po stisknutí tlačítka je i MyLoader, jenž čeká na odpověď od serveru, kterou výsledně uloží do forum.php.

Objekt LoadVars je poměrně často používaný díky své jednoduchosti a přehlednosti. Nevýhodou LoadVars je určitá neohrabanost při práci s velkým počtem proměnných a strukturovanými daty, protože LoadVars načítá a posílá jen jednoduché řetězce složené z dvojic název a hodnota. Důležitou alternativou k LoadVasr je použití tříd podporujících XML.

2.2.2.3 Třídy XML, XMLNode

Třídy XML resp. XMLNode jsou v actioscriptu dostupné od verze 5.0 a reprezentují zpracovávaný XML soubor resp. slouží k jeho procházení a editaci. Samotná třída XML je z programátorského hlediska potomkem LoadVars obsahující dodatečně funkce jako například parseXML a IgnoreWhite. ParseXML vytvoří datový model DOM z načteného XML dokumentu a IgnoreWhite indikuje zda se mají ignorovat uzly, které obsahují jen bílé znaky. Bílé znaky jsou takové znaky, jenž slouží ke zvětšení přehlednosti dokumentu a jsou součástí přenášených dat . Mírnou změnou od LoadVars je, že ve třídě XML je povolen jen POST jako způsob přeposílání dat.

```
var BASE: XML = new XML();
BASE.contentType = "text/xml";
BASE.ignoreWhite=true;

BASE.onLoad = function(success:Boolean){//definování listeneru
  if (success){//xml dokument je načten
            firstParse(BASE);//inicializace vnitřních proměnných
            showModel();
            }
      else{trace("error");}
      };

BASE.load(_global.where+"&mod="+Math.random());// žádost o zaslání dat z adresy _global.where

firstParse =function(TREE:XML){//inicializace proměnných
for (var i = 0; i<TREE.childNodes.length; i++){
 if ((TREE.childNodes[i].nodeName == "ELEMENTS") && (TREE.childNodes[i].nodeType == 1))
                  { parseelements(TREE.childNodes[i]);continue;};
        };};

// funkce vytvoří nový element "E", který bude obsahovat elementy "P"
makeNewElement=function(ports:Number){
            var HELP:XML=new XML();
            HELP.nodeName="E";
            HELP.nodeType=1;
            for (var i:Number=0;i<ports;i++){
                  var HELP2:XML=new XML();
                  HELP2.nodeName="P";
                  HELP2.nodeType=1;
                  HELP.appendChild(HELP2);
            }
            _global.ELEMENTS.appendChild(HELP);
      };
```

Obrázek 2.13: Příklad použití XML a XMLNode

Actionscript na obrázku 2.13 poukazuje na použití XML a XMLNode. V první části kódu se definuje třída BASE, do které se načte obsah nahraného dokumentu. Proběhne-li nahrání korektně zavolá se funkce firstparse, která projde první úroveň vytvořeného stromu a podle typu elementu volá jednotlivé pomocné funkce. Funkce makeNewElement vnoří do _global.ELEMENT jeden element E, který bude obsahovat několik elementů P.

Třída XMLNode přistupuje přímo k jednotlivým uzlům XML dokumentu. Pro přístup k uzlům požívá vlastnosti jako childNodes, firstChild, lastChild, nextSibling, parentNode nebo previousSibling. Pro přístup k atributům a vlastnostem elementů využívá funkce a proměnné jako attributes, nodeName, nodeType a nodeValue. Třída XMLnode zároveň umožňuje dynamickou editaci struktury XML dokumentu. Pro vytvoření nového elementu slouží funkce createElement, createTextNode, insertBefore a appendChild. Pro odstranění existujícího elementu se využívá funkce removeNode a konečně cloneNode(deep), která vytvoří kopii vybraného uzlu. Atribut deep udává zde se v kopii vyskytují i potomci kopírovaného uzlu. Typickou ukázku činnosti třídy XML a XMLNode je ukázána v programu na obrázku 2.13, který obsahuje část jednoduchého parseru a funkci makeNewElement, která přidává další elementy do XML dokumentu.

```
test.xml:
<employees>
   <person type="m"> Novak</person>
   <person type="f">  Janíková</person>
   <person type="m"> Pochop</person>
   <person type="f">  Nováková</person>
   <person type="m"> Peterka</person>
 </employees>

import mx.xpath.XPathAPI;
myXML.onLoad = function(success){
   path = "/employees/person[@type="m"]";
   arNodes = mx.xpath.XPathAPI.selectNodeList(this.firstChild,path);
   }
myXML.load(test.xml);
```

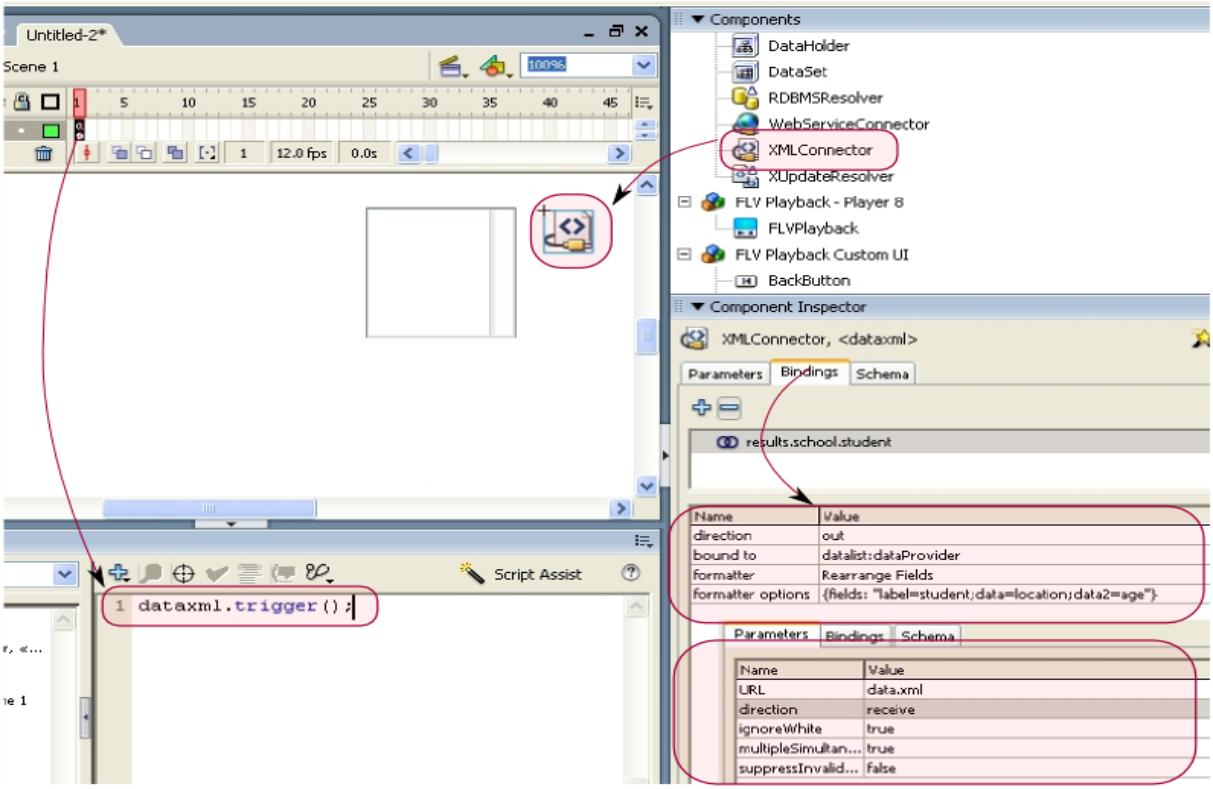
Obrázek 2.14: Příklad použití XPath.

Další důležitá vlastnost zpracovaní XML dokumentů ve Flash je možnost použití XPath, jenž je podporován od verze Macromedia Flash MX 2004. XPath(*XML Path Language*) je jazyk určený k adresaci částí XML dokumentů a to včetně výběru elementů, atributů a dalších hodnot. Základem je třída XPathAPI, která obsahuje dvě statické metody selectNodeList a selectSingleNode. Metoda selectNodeList má parametry node a path. Node je instancí objektu XPathAPI, nad kterou se provede XPath příkaz, který bude obsažen v parametru path. Na následujícím příkladu je zobrazen klasický příklad, kde je zpracován krátký XML dokument pomocí XPath ve Flashi viz. Obrázek 2.14. Na tomto obrázku je zobrazen obsah XML souboru a část actioscriptu, který daný XML soubor načte a s pomocí mx.xpath.XPathAPI.selectNodeList zjistí označení všech mužských zaměstnanců.

2.2.2.4 XMLConnector

XMLConnector je komponenta dostupná od verze Macromedia Flash 2004 MX. Komponentu XMLConnector si můžeme představit jako třídu XML s doplňujícím grafickým rozhraním. Primárním cílem této komponenty je usnadnit propojení externího XML souboru s objekty ve scéně. Tato možnost se například využívá pro propojení XMLdat s textovými poli, rozevírajícími seznamy a stromy. Mezi základní vlastnosti a funkce komponenty XMLConnector patří URL, jenž obsahuje adresu XML souboru

a trigger, metoda jenž spustí komunikaci a její výsledek umístí do results, což je instance třídy XML. Pro určení směru komunikace se používá parametr direction(send,retrive,send/retrive) a pro ignorování bílých znaků se nastavuje IgnoreWhite.



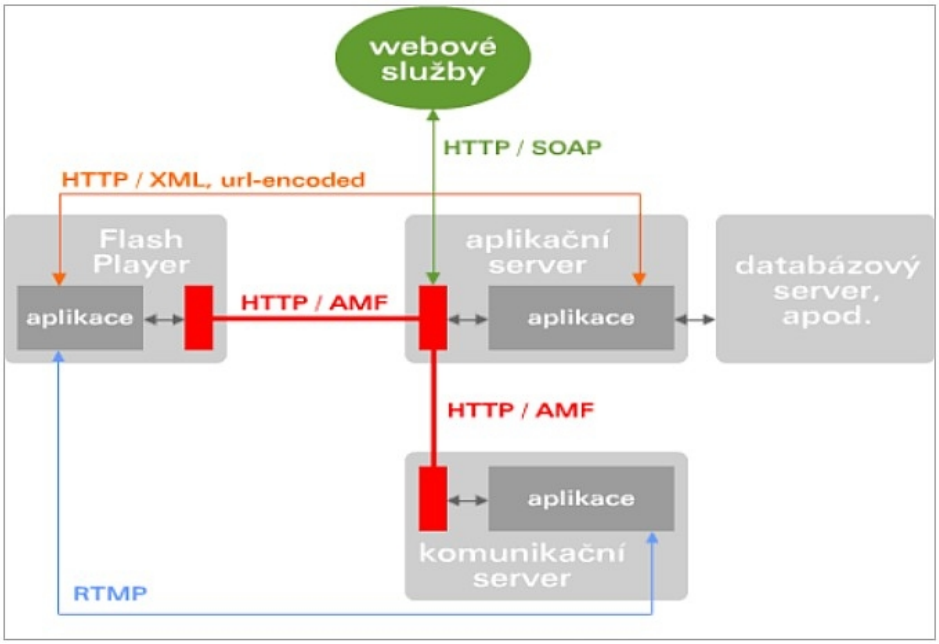
Obrázek 2.15: Příklad nastavení XMLConnectoru přes uživatelské rozhraní.

Na obrázku 2.15 je znázorněn příklad nastavení XMLConnectoru. V prvním kroku se z balíku komponent, který se nalézá ve Window/Components, vybere příslušný objekt a přenese na pracovní plochu. Po vložení na plochu nastavíme v Komponent Inspektoru ve složce Parametrs vlastnosti instance XMLConnector, jako je například adresa zpracovaného souboru. Ve složce Bindings nastavíme způsob provázání dat s objekty ve scéně. A pro úspěšné spuštění komunikace vložíme na začátek animace actionscript obsahující funkci trigger , která spustí komunikaci komponenty .

2.2.2.5 Flash Remoting

Od roku 2002, kdy byl vydán Flash Player 6.0, se datuje podpora technologie Flash Remoting. Flash Remoting podporuje komunikaci s větší mírou komunikační abstrakce na rozdíl od dříve zmíněných technologií. Místo toho abychom data posílali ve formátu jako XML, můžeme napsat jen vytvořil potřebnou obsluhu na straně serveru, aniž bychom se starali o řízení komunikace[32]. Flash Remoting pracuje podobně jako webové služby s tím rozdílem, že místo SOAP komunikace se používá výměna informací pomocí uzavřeného binárního formátu Action Message Format (dále jen AMF), který umožňuje rychlejší komunikaci s menším množstvím řídících a neužitečných dat, jak je znázorněno na obrázku 2.16[32]. Pro korektní přenos zpráv je nutné zajistit konverzi datových typů mezi actionsctiptem a

jazykem aplikačního serveru, serializaci i deserializaci včetně převodu zpráv do AMF. Na straně klienta tuto činnost zajišťuje třída NetConnection a na straně serveru proces, jenž se označuje jako Flash Remoting Gateway. Knihovny pro vytvoření brány jsou dostupné pro prostředí jako J2EE, NET, JRun, ColdFusion nebo PHP. Ale bohužel velká většina z takovýchto řešení není zdarma.



Obrázek 2.16: Blokové schéma popisující komunikaci v RIA aplikacích.

2.2.2.6 Webové služby

Podpora webových služeb je dostupná od verze Flash MX 2004 Professional. Podobně jako Flash Remoting tak i webové služby umožňují větší míru komunikační abstrakce. Webové služby na rozdíl od Flash Remoting nepoužívají AMF ale SOAP, což je otevřený způsob komunikace založený na XML. Znatelnou výhodou je i to, že pro komunikaci se serverem není nutná brána Flash Remoting Gateway, ale postačuje jakákoli implementace vyhovující standardům webových služeb.

V rámci činnosti webových služeb se setkáváme často s těmito pojmy: WSDL (Web Services Description Language) - formát založený na XML pro popis metod a funkcí dané webové služby, SOAP(Simple Object Access Protocol) - formát XML komunikace mezi serverem a klientem webové služby pomocí HTML, UDDI(Universal Description Discovery and Integration) - veřejné seznamy webových služeb a klient webové služby - každý software, který dokáže komunikovat s webovou službou.

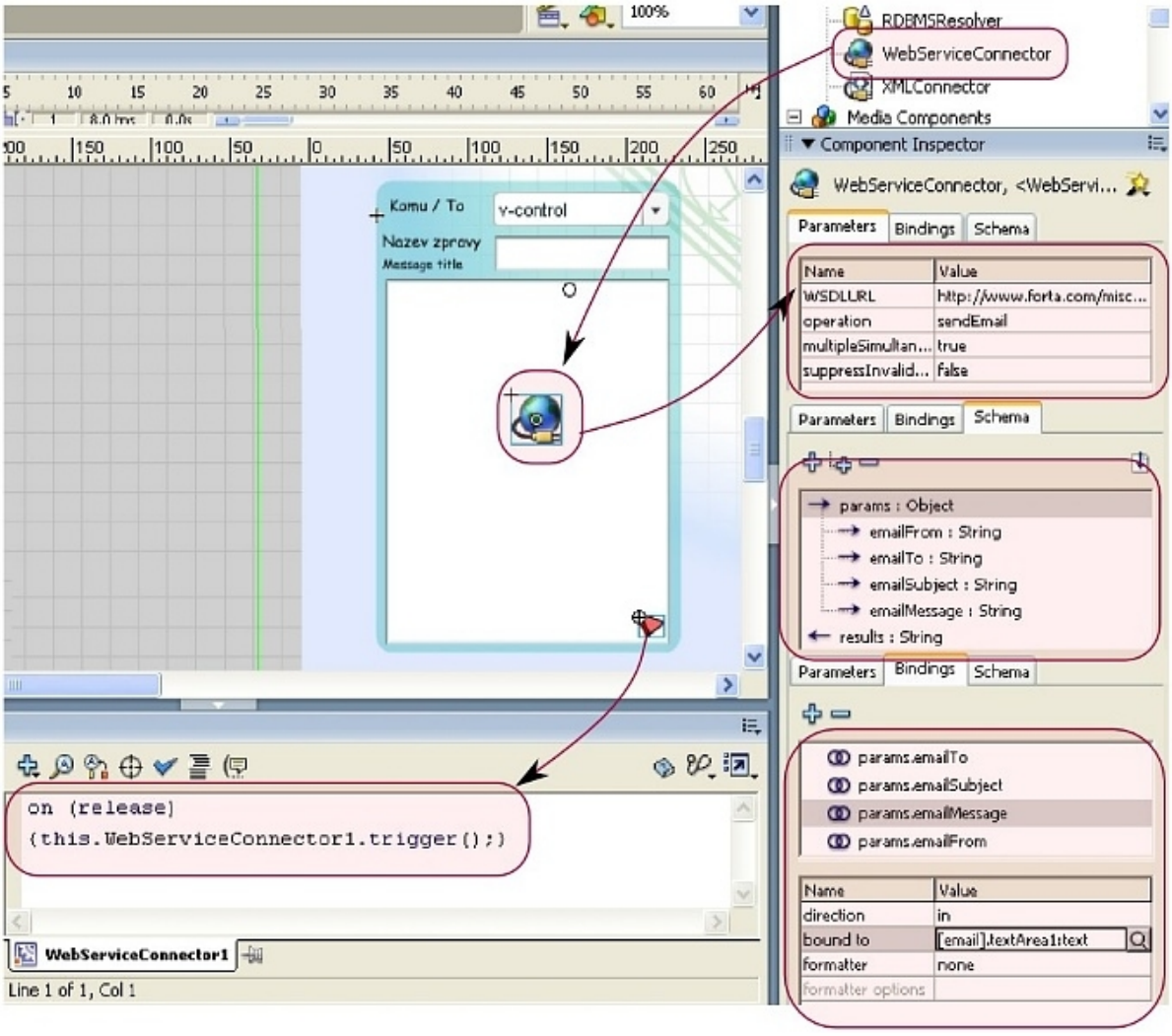
Velkou výhodou při práci s Flashem jsou dvě klientské formy webových služeb. Tj. při pouhém návrhu SWF aplikace je klientem vývojové prostředí Flashe, které zprostředkuje popis a seznam služeb, tj. zaměřuje se hlavně na komunikaci typu WSDL. Ale výsledná Flashová animace se chová jako klient webových služeb, který je již zaměřen na samotnou datovou komunikaci tj. na SOAP.

Podobně jak vznikla komponenta XMLConnector pro pohodlnější práci s XML soubory, tak z obdobného důvodu vznikla komponenta WebServiceConnector využívající třídu WebService, což je třída podporující komunikaci s webovými službami. Komponenta WebServiceConnector má několik

základních parametrů a metod, jenž si na následujících řádcích objasníme.

Parametry, události a metody komponenty WebServiceConnector:

- **WSDLURL –** je URL požadované webové služby.
- **SuppressIvalidCalls-** určuje, zda se má webová služba potlačit, dojde-li k chybnému zadání údajů.
- **multipleSimultaneousAllowed** – povoluje vícenásobné volání služby.
- **params** - pole parametrů, které se předávají webové službě.
- **trigger** - metoda, která povolí vykonání webové služby.
- **results –** data, jsou-li taková, která byla navrácena webovou službou.
- **result** - událost, která se přivolá po úspěšném dokončení metody webové služby.
- **send** - událost, která se odpálí při volání metody trigger, ještě před tím než se kontaktuje server.



Obrázek 2.17: Příklad použití komponenty WebServiceConnector.

Obrázek 2.17 popisuje použití komponenty WebServiceConnector. V prvním kroku se komponenta vloží na pracovní plochu a tím se aktivuje nabídka v component inspektoru. V dalším kroku nastavíme adresu služby WSDLURL, což způsobí automatické načtení informací o dané službě. Postupně provážeme vstupní a výstupní proměnné webové služby s objekty ve scéně. V našem případě vstupní okna formuláře jako je adresa odesilatele, název zprávy a text zprávy provážeme s parametry webové služby. A po stisknutí tlačítka se zavolá trigger, který způsobí provedení služby tj. k odeslání emailu.

2.2.2.7 SharedObject

Zajímavou možností jak Flash dokáže udržovat informace o sezení je třída SharedObject, která poskytuje API pro ukládání a získávání informací o sezení uloženém na stroji uživatele. Zajímavou vlastností je možnost sdílet informace mezi několika Flashovými animacemi v rámci jednoho sezení.

Parametry, události a metody objektu SharedObject:

- **SharedObject.getLocal**(name,[path]) -statická metoda jenž vytvoří novou instanci SharedObject, která bude odkazovat na data sdružená s name
- **so.clear()** - vymazání obsahu SharedObject
- **so.data** - pole obsahující informace, které mají být sdružené s SO
- **so.flush()** - pokusí se uložit data na disk
- **so.getSize() -** vrátí velikost sdíleného objektu SharedObject

```
nacist_data = function (obj:String):String {
   var so:SharedObject = SharedObject.getLocal(obj);
   var str:String = so.data.username;
   return str;
   };

ulozit_data = function (obj:String, val:String):Void {
   var so:SharedObject = SharedObject.getLocal(obj);
   so.data.username = val;
   so.flush();
   };
```

Obrázek 2.18: Příklad použití objektu SharedObject.

Příklad použití SharedObject ukazuje obrázek 2.18. Funkce nacist_data vrátí hodnotu parametru username, který je uložen ve sdíleném objektu sezení, jehož označení je uloženo v proměnné obj. Funkce ulozit_data uloží hodnotu val do proměnné username ve sdíleném objektu obj v rámci jednoho sezení.

2.2.2.8 Shrnutí technologií

FlashVars

•nevýhody: pro jednosměrnou statickou komunikaci, pro malé objemy dat

•výhody: nejjednodušší metoda, předá parametry i animacím, u nichž neznáme jejich strukturu

•využití: tvorba dynamických obrázků, nastavování parametrů přehrávačům videa, při autorizaci a při navázaní složitější komunikace

LoadVars

•nevýhody: nepřehledné pro větší objemy dat, čte a posílá data v jednoduchém formátu název/hodnota

•výhody: jednodušší metoda komunikace než XML, Flash Remoting nebo Web Services, využitelná pro větší objemy dat než komunikace za pomoci flashvars

•využití : pro jednoduché RIA aplikace

XML,XMLnode

•nevýhody: součástí přenosu je i velké množství nepotřebných dat

•výhody: přenos strukturovaných údajů, XMLConnector, Xpath

•využití: pro vykreslovaní složitých struktur popsaných XML soubory jako kreslící diskusní fóra, dynamická stavba formulářů, simulace SVG

Web Services

•nevýhody: součástí přenosu je i velké množství nepotřebných dat

•výhody: není zapotřebí Flash Remoting Gateway , možnost použití WebServiceConnector

•využití : pro robustní RIA aplikace a aplikace které používají externí webové servery, například při posílání emailu přímo z Flashové animace

Flash Remoting

•nevýhody: je požadována brána Flash Remoting Gateway

•výhody: AMF umožňuje efektivnější a rychlejší komunikaci s menším množstvím nepotřebných dat

•využití : pro robustní RIA aplikace, kde je vyžadovaná rychlá komunikace s objemnými daty

3 Analýza problému a návrh řešení

V této kapitole si popíšeme techniky a možnosti správy počítačové sítě. Uvedeme základní požadavky kladené na takovéto systémy spravující počítačové sítě. Následně si určíme jednotlivé vlastnosti a schopnosti , které budeme od takového systému požadovat a které posléze budeme implementovat ve výsledné aplikaci. Analýzu takového systému následně rozdělíme na tři části. První část bude jednoduchý agent, který pracuje přímo s počítačovou sítí. Další částí bude server, který shromažďuje data od agentů a zpracovává data pro následnou vizualizaci. A poslední část bude modul klient, který bude vykreslovat měřená data, utvářet uživatelské rozhraní a který bude následně předávat serveru požadavky od uživatele.

3.1 Analýza stávajících řešení

Ještě než začneme analyzovat a implementovat výslednou aplikaci, tak si uvedeme některé stávající systémy, které se v současné době používají při správě počítačových sítí.

V prvé řadě můžeme takové systémy rozdělit na komerční (HP OpenView, Cisco Works) a volné (MRTG, CACTI, Nagios ...). Komerčními systémy se zabývat nebudeme, jelikož je jejich testování obtížné pro jejich cenovou náročnost. Zaměříme se hlavně na nejrozšířenější volné systémy, které jsou šířeny pod licencí GPL a je tedy snadnější otestovat jejich vlastnosti. Takové systémy mají často rozsáhlou vývojářskou základnu a na internetu je možné najít velké množství diskuzních skupin, které ve výsledku poskytují rozsáhlou uživatelskou podporu takovýmto systémům. Tyto systémy hlavně umožňují rozsáhlý management sítí, monitoring, plánování, optimalizaci, vyhodnocování, měření veličin i detekci topologie. Mnoho z nich také podporuje vytváření měřících sestav, které mohou mít nastaveny různé intervaly měření a mohou seskupovat různé veličiny do jednoho grafu. V následujících odstavcích si některé z těchto systémů popíšeme.

3.1.1 MRTG

Multi Router Traffic Grapher (dále jen MRTG) je opensource systém vytvořený Tobiasem Oetikerem. MRTG je jedním z nejstarších podobných systémů a v současné době je na ústupu. MRTG byl napsán v jazyku Perl a původně byl podporován pouze na platformě Linux/UNIX. Součástí MRTG je také modul RRDTool, který byl vytvořen také Tobiasem Oetikerem. RRDTool je poměrně rozšířený a univerzální nástroj pro ukládání a vizualizaci měřených časových průběhů.

Celý průběh činnosti je takový, že uživatel nejprve určí OID měřené informace, IP adresu zařízení na kterém se měření bude provádět a určí interval dotazování. V následném kroku se měřící proces SNMP requestem dotazuje na dané OID pro dané zařízení a při daném intervalu měření. Zjištěné hodnoty jsou předávány modulu RRDTool, který je uloží do své databáze a následně z nich generuje obrázky časových průběhů z měřených dat.

Jednou z nevýhod MRTG je jazyk Perl, který v současné době je již na ústupu a tedy i komunita programátorů, jenž obohacují systém novými funkcemi, není tak velká jako u dalších podobných systémů. Za další nevýhodu se považuje i malá svoboda uživatele při vytváření grafů, protože časové průběhy různých veličin není možné libovolně skládat do sestav a vizualizovat společně. Další nevýhodou je malá modularita systému, který neumožňuje jednoduše spojovat MRTG s dalšími moduly a rozšířeními jako je například modul pro zobrazení topologie zkoumané sítě atp.

3.1.2 CACTI

Jedním z nejpopulárnějších zařízeních pro správu sítě je CACTI. CACTI je napsáno v jazyku PHP s tím, že některé skripty a rozšíření jsou napsány v Perlu. CACTI je bezplatný nástroj pro monitorování jednotlivých zařízení v síti a o jeho vývoj se stará objemná komunita vývojářů[33]. CACTI podobně jako MRTG používá protokol SNMP pro sběr informací ze zkoumaných zařízení a modul RRDTool pro uskladnění a vizualizaci těchto informací. Zároveň je ale CACTI velice univerzální nástroj, který dává uživateli značnou volnost při instalaci doplňků, při vytváření šablon i při seskupování měřených hodnot do jednoho grafu.

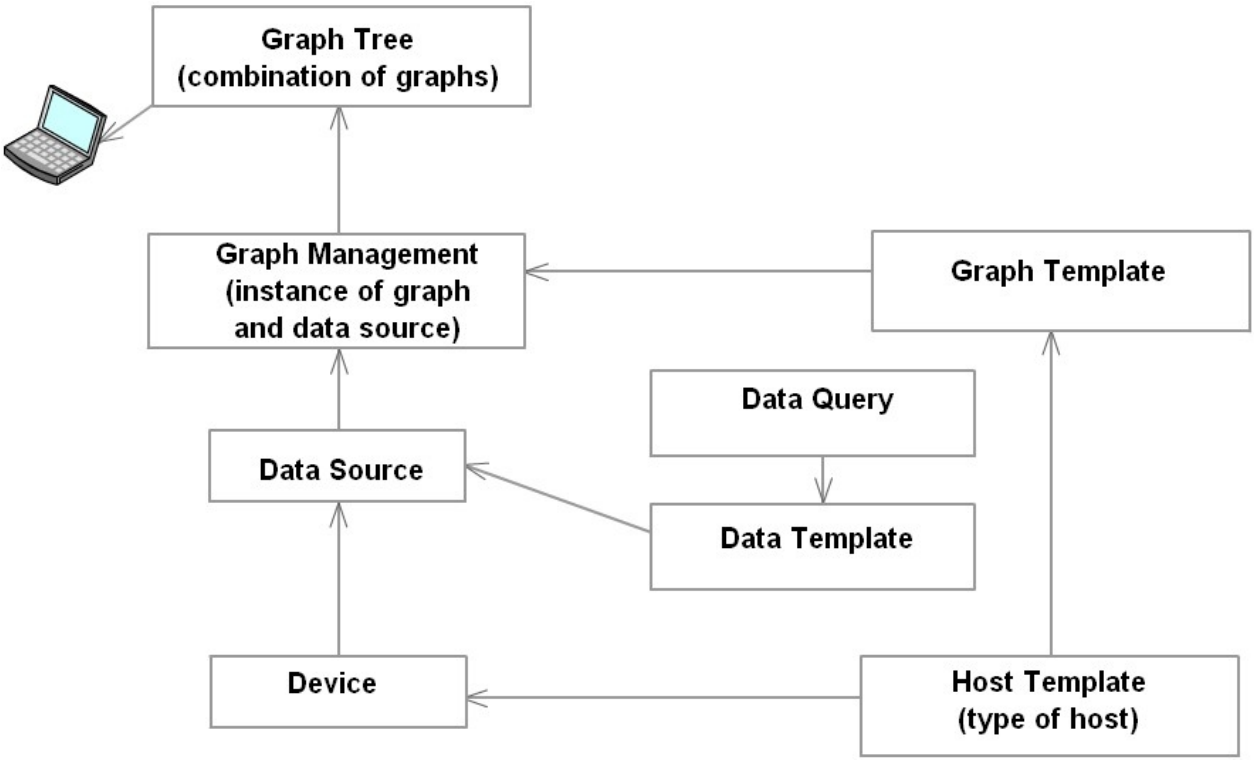
Aby CACTI správně fungovalo potřebuje tzv. Poller. Poller je modul, který plní požadavky PHP skriptů a získává data ze zkoumané sítě[33]. Poller je proces na zkoumané síti, kterému se nastavují informace jako perioda měření, maximální počet vláken a nebo maximální počet OID v jednom SNMP requestu. Poté co Poller obdrží požadované OID, začne komunikovat s požadovaným zařízením. Poller po obdržení odpovědi na request pošle naměřené informace zpět serveru, který informace uloží do databáze a nebo je předá modulu RDDTool. V následném kroku se dané informace vizualizují podle předdefinovaného vzoru.

Hlavní výhodu CACTI je rozpracovaná forma řízení vizualizace grafů. Při vizualizaci grafů se postupuje podle návodu zobrazeném na obrázku 3.1.

Nejprve se pomocí „Device“ vybere zařízení, nad kterým se budou provádět dané akce. Danému zařízení adresujeme skupinu dotazů „Data Query“ a šablonu „Host Template“, jíž přiřadíme možné grafové vizualizace „Graph Template“ danému zařízení. Takto zvolená skupna dotazů formuluje příkazy, které jsou určeny pro získávání informací o zařízení stejného typu.

V dalším kroku se automaticky vytvoří zdroj dat „Data Source“ a to také podle typu zvoleného zařízení. Z datového zdroje se vytvoří graf „Graph Management“, který se následně přiřadí do skupiny grafu „Graph Tree“. A po příslušném požadavku se daná skupina vizualizuje ve webovém prohlížeči.

Jako jedny z menších nevýhod můžeme uvést poměrně složitější instalaci, při které není možné nakonfigurovat Poller přes webové uživatelské rozhraní a nutnost doinstalovat řadu doplňujících modulů, které nejsou dostupné v základním instalačním balíku. Další z nevýhod je omezená možnost použití interaktivních dynamických grafů, jelikož vizualizační modul RDDTool nepodporuje vyspělejší formáty pro vizualizaci dat.



Obrázek 3.1: Popis vytváření grafů v systému CACTI.

3.1.3 Nagios

Nagios je opensource systém, za jehož hlavního tvůrce je považován Ethan Galstad[34]. Na rozdíl od výše popsaných aplikací byl Nagios primárně vyvíjen spíše pro rychlou detekci změn nebo poruch na monitorované síti než na samotné měření průběhu veličin. Cílem takového systému bylo neprodleně informovat správce počítačové sítě tak, aby mohla být případná porucha rychle vyřešena. Samotný Nagios je démon vytvoření v C/C++ a krom samotného SNMP používá i jiné protokoly jako například ICMP,SMTP[34]. Z toho důvodu Nagios může ověřovat přítomnost zařízeních pomocí příkazu ping a při tom posílat varovné emaily administrátorovi sítě.

Jednou z nevýhod Naigosu je slabší podpora vizualizace měřených hodnot oproti CACTI. Další nevýhodou je poměrně složitější konfigurace a nutnost předem definovat monitorované zařízení. Tj. Nagios nepodporuje automatickou detekci prvků, které se musejí přidávat ručně zapsáním do konfiguračních souborů.

Některé nevýhody tohoto systému řeší doplňkový modul IDEALX Management Console (IMC). IMC rozšiřuje Naigos o uživatelské webové rozhraní a o podporu automatického zjišťování topologie na úrovni třetího modelu ISO/OSI.

3.1.4 NeDi

Network Discovery Suite(dále jen NeDi) je systém vytvořený v jazyku Perl [35]. Ve své současné podobě umožňuje ucelený monitoring sítě, vizualizaci průběhů a detekci změn podobně jako předchozí aplikace. Ale pro zjišťování informací o počítačové síti požívá v prvé řadě protokol Cisco Discovery Protocol (dále jen CDP). CDP umožní systému NeDi získat velice přesné informace o topologii zkoumané sítě, které je pak možné následně vizualizovat.

CDP je protokol vyvinutý společností CISCO a je prvořadě podporován zařízeními tohoto výrobce. CDP slouží k detekci okolních zařízení a k předávání základních informací o těchto zařízení. Pracuje na tom principu, že každý prvek, který podporuje CDP posílá po 60 sekudnách o sobě informace na CDP mulitcast adresu 01-00-0C-CC-CC-CC. Každé CISCO zařízení tyto informace přijme a uchová si je v tabulce, kde je možné si je přečíst příkazem show cdp neighbors. Takováto tabulka si uchovává informace do té doby dokud nepřijdou nové informace od stejného zařízení a nebo dokud nevyprší časový limit takové informace. Informace z jednotlivých zařízení nemusejí být stejné a liší se podle typu daného zařízení a operačního systému, který běží na daném zařízení[36]. Součástí této tabulky je např. IP adresa, port ze kterého přišla CPD zpráva, základní informace o VLANách nebo typ daného zařízení. Zajímavým poznatkem je i to, že CDP zprávy mohou posílat i transparentní zařízení, a proto tedy CDP umožňuje získat velice přesné informace o zkoumané síti na nejnižších úrovních modelu ISO/OSI. CDP byl zároveň podporován i zařízeními od společnosti Hewlett-Packard, ale v současné době je od této podpory upuštěno. A novější zařízení od výrobce Hewlett-Packard přechází na protokol Link Layer Discovery Protocol (dále jen LLDP).

Z výše zmíněného plyne základní nevýhoda tohoto protokolu a to omezení funkčnosti pouze na CISCO zařízení.

3.2 Požadavky kladené na výslednou aplikaci

Informace zmíněné v předchozích kapitolách mají skutečně podstatný význam při návrhu a implementaci výsledné aplikace. SNMP nám umožní získávat informace o zařízeních na počítačové síti. Technologie Adobe Flash nám umožní vytvářet kreativní vizualizace topologií sítě a časových průběhů měřených hodnot. Obdobně i předchozí odstavec o již existujících řešeních má podstatný význam pro naši práci. Výše popsané systémy totiž nejen že umožňují správu počítačové sítě, ale zároveň používají odlišné technologie a přístupy, kterými se můžeme nechat inspirovat. CACTI obsahuje propracovaná vizualizační schémata při tvorbě grafů a také spolupracuje s měřícím procesem umístěným v dosahu zkoumané sítě. Naigos rychle detekuje případné změny v síti za pomoci i jiných protokolů než jen SNMP. NeDi dokáže velice přesně určit topologii sítě za použití specifického protokolu CDP. MRTG se svoji jednoduchostí dokázal prosadit a rozšířit bez ohledu na malou funkcionalitu. Na základě těchto a dalších vlastností se pokusíme formulovat požadavky, na jejichž základě navrhneme možné řešení. Cílem projektu není ani tak vytvoření aplikace, která by se svoji funkcionalitou blížila výše popsaným systémům, ale alespoň částečně navrhnout takový systém, který by podporoval základní operace nad sítí a zároveň využíval moderní grafické prvky jako jsou interaktivní animace při vizualizaci požadovaných informací.

Základní požadavky kladené na výslednou aplikaci:

- **Distribuce měření** - Pokud měříme informace o prvcích v malé síti, pak si ve výsledku postačíme s jedním měřícím procesem. Problém nastane když je měřená síť velkých rozměrů. Pokud bychom v takové síti měřili například topologii sítě, docházelo by k výraznému zatěžování měřící stanice a tedy i k větším ztrátám datagramových paketů. V takové případě je vhodné použít větší množství měřících procesů, kterým přidělíme určitý části sítě. Výsledná aplikace by tedy měla podporovat takovéto rozložení měřících procesů. Takovou vlastnost např. použijeme i v situacích, kdy se jeví výhodné sloučit několik lokálních sítí do jednoho vizualizačního modelu.
- **Modularita** - Pod modularitou se míní obecná vlastnost aplikace, která je tvořena částečně nezávislými bloky. V našem případě aplikaci rozdělíme do tří hlavních bloků a při komunikaci používáme jednoduchý XML formát přenášených dat.
- **Podpora většího množství modelů –** Výsledná aplikace by měla umožňovat správu více sítí na jednou a to bez toho aniž by bylo nutné pro každou novou zkoumanou síť znovu pracně nastavovat konfigurační soubory na webové serveru.
- **Získání základních informací o prvcích v síti –** Výsledná aplikace by měla zároveň umožnit získání základních informací o jednotlivých elementech a jejich portech. Tyto informace by měla následně poskytnout při vizualizaci a při exportu.
- **Export informací –** Pro administraci i s právu sítě se často nepoužívá pouze jeden systém. Naopak často se tyto systémy kombinují tak, aby si administrátor mohl vzít od daného systému to nejlepší**.** Proto by bylo zajímavé poskytnout naměřená data ne jen jako formuláře a obrázky v HTML, ale i jako formátované soubory, které si může administrátor například zálohovat a tím uchovávat starší vlastnosti sítě.
- **Webové uživatelské rozhraní –** Aplikace by měla být ve výsledku platformě nezávislá a výsledky měření by neměli být dostupné jen z jedné terminálové stanice, ale i všem autorizovaným uživatelům na internetu.
- **Vizualizace topologie –** Jedním z možných grafických výstupů, který poskytují i výše popsané systémy je vizualizace topologie. Výsledná aplikace by mohla takovouto topologii vytvořit. Při tom je ale nutné zajistit získání všech potřebných dat. Dále je nutné navrhnout algoritmus, který v takovém modelu propojí jednotlivé prvky a uspořádá je přehledně do výsledného obrázku. Při vytváření modulu pro vizualizaci topologie je vhodné také přihlédnout k zajímavým možnostem technologie Flash, která umožňuje distribuovat SWF objekty i na jiné webové stránky. Například přehrávač YouTube videa si tvůrci webů mohou vkládat na své vlastní webové stránky jen jednoduchým přiložením krátkého HTML kódu do svých vlastních webových stránek.
- **Vizualizace naměřených hodnot a grafů –** Další pochopitelný požadavek je i vytváření grafů z naměřených časových průběhů. V tomto případě se jeví jako vhodné použít technologii interaktivních dynamických grafů, které např. používá společnost Google ve svých aplikacích .
- **Podpora virtuálních sítí** - Zajímavým požadavkem jak obohatit výslednou aplikaci je i přidání podpory pro vytváření virtuálních sítí. Např. když potřebujeme pouze manuálně zaznamenat model topologie reálné sítě, ale nechceme zatěžovat zkoumanou síť měřícím procesem.
- **Reakce na událost –** Výsledná aplikace by měla být případně schopna zpracovávat SNMP trapy.
- **Nastavování parametrů** – Požadavek pro možnost nastavení parametrů na SNMP zařízení.

Ve výsledné aplikaci se uplatnili některé výše zmíněné požadavky. Aplikace je rozdělena do třech částečně nezávislých modulů na základě požadavku modularity a distributivnosti. Aplikace je rozdělena na agenty, server a klienty. Agenti umožňují rozdistribuovat měření na více stanic. Každému agentovi je přiřazen základní konfigurační XML soubor, který si agent přečte před svým spuštěním. Agenti zajišťují sběr informací za pomoci SNMP a upravují síť podle požadavků serveru. Zároveň posílají XML soubory s výslednými záznamy o jednotlivých elementech zpět serveru. Server přijímá data od více agentů a podle toho vhodně upravuje informace v databázi. Server zároveň plní funkci webového serveru a za pomoci PHP skriptů vytváří HTML stránky. Server zpracovává uložená data a vytváří reprezentativní XML soubor, který posléze posílá Flashové animaci, která je součástí webové stránky. Klient je označení pro HTML a Flash klienta. Klient obsahuje základní uživatelské rozhraní pro práci s aplikací. Klient umožňuje vytvářet, mazat a upravovat jednotlivé modely. Klient zároveň zobrazuje topologii sítě jednotlivých modelů v propracovaném interaktivním Flashovém modulu. Klient umožňuje zobrazit detailní informace o jednotlivých síťových prvcích a jejich portech. Autorizovanému uživateli klient poskytuje možnost si zaregistrovat měření zatížení jednotlivých portů a posléze zobrazit graf, který měřené hodnoty zobrazuje. Klient zároveň podporuje tvorbu virtuálních modelů.

Při návrhu systému se muselo řešit velké množství situací, aby se docílila správná funkce takovéto aplikace. V následujících odstavcích si některé tyto problémy popíšeme, uvedeme jejich jednoduchou analýzu a popíšeme výsledné řešení.

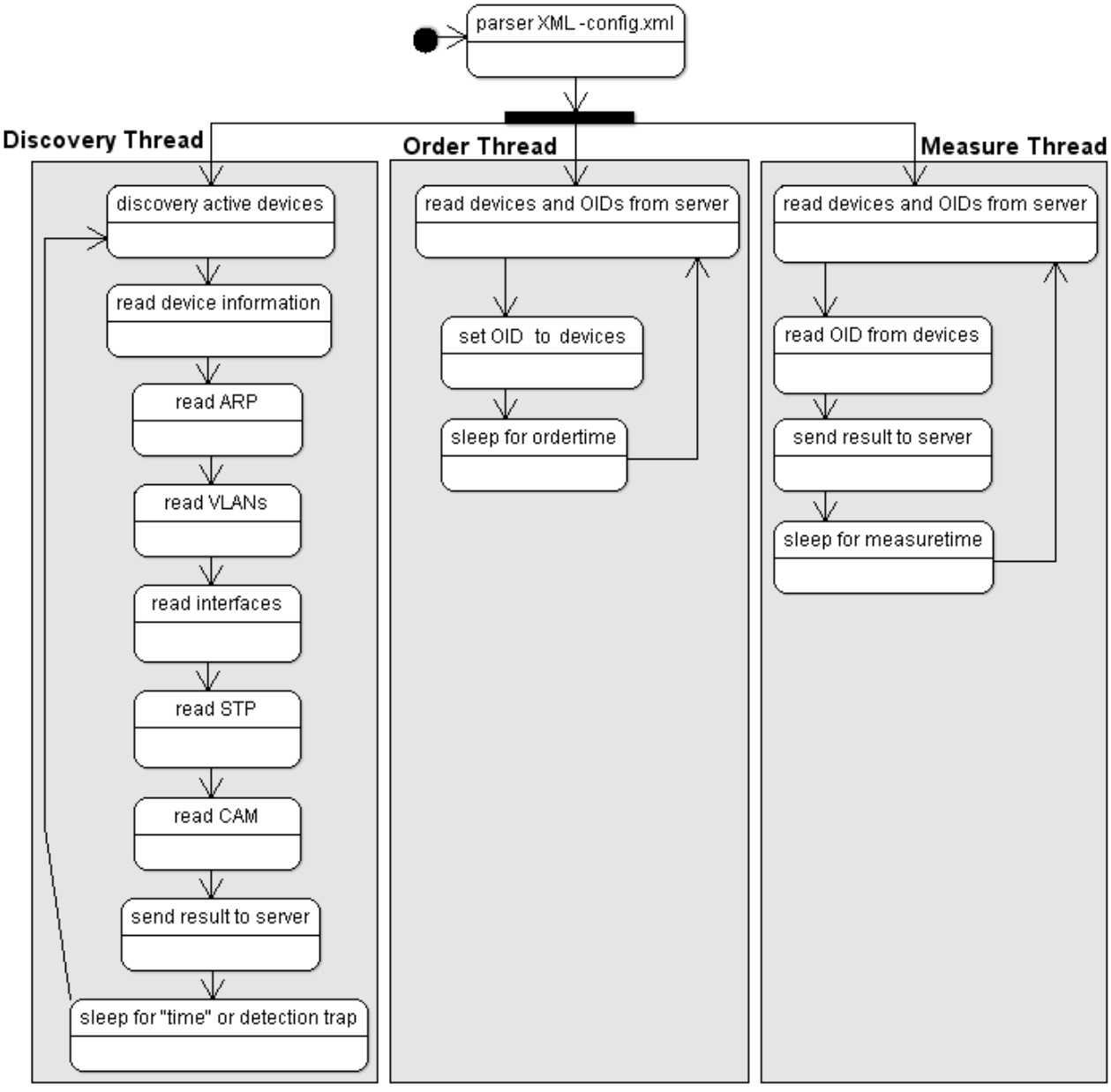
3.3 Agent

3.3.1 Požadavky kladené na agenta

Jak již bylo v předchozí kapitole řečeno, agent je jedním z modulů naší aplikace. Agent by měl být nenáročná aplikace specializovaná pro sběr dat. Proto se ani konfigurace nebude provádět přes uživatelské rozhraní, ale za pomoci konfiguračního souboru. Na základě již dříve zmíněných požadavků bude agent umožňovat distribuovaný sběr dat a paralelní činnost dalších agentů na stejné síti. Abychom dostatečně podpořili modularitu celého systému, bude konfigurační soubor i data předávaná serveru ve formátu XML. V zadání je uvedeno, že výsledná aplikace by měla vycházet ze SNMP a tedy i agent bude měřit veličiny pomocí SNMP. Agent bude v nastavené frekvenci zjišťovat požadavky uložené pro něj v databázi na serveru a tyto požadavky se bude pokoušet realizovat. Při provádění operací se požadavky nespouští sekvenčně, ale za pomoci paralelně běžících vláken, tak aby se daná realizace urychlila.

Naměřené veličiny se budou zpracovávat až na straně serveru. A to z toho důvodu, že si agent neukládá původní hodnoty měřených informací, proto aby nezatěžoval pracovní stanici, na které běží. Například při měření zatížení konkrétního portu potřebujeme rozdíl dvou po sobě jdoucích hodnot z příslušných čítačů uložených v MIB daného zařízení. Protože ale agent si neuchoval původní hodnotu, je nutné toto zatížení vypočítat až na straně serveru. Dalším z problémů, které se řeší až na straně serveru je kompletace topologie, která se provádí až na serveru z toho důvodu, že je nutné zkompletovat informace z více agentů.

Na obrázku 3.3 je znázorněn jednoduchý stavový diagram. Tento diagram zjednodušeně popisuje jak pracuje daný agent. Po nahrání agenta na příslušnou pracovní stanici a po vyplnění konfiguračního souboru je možné daného agenta spustit. V prvním kroku agent projde konfigurační soubor config.xml. V tomto souboru si přečte například adresu nadřazeného serveru, tak i časové intervaly, které určí časové prodlevy pro měřící vlákna. V dalším kroku se paralelně spustí 3 vlákna. Měřící vlákno si nejprve ze serveru zjistí seznam měřených OID a seznam měřených zařízení. V dalším kroku se spustí skupina vláken, jenž dané veličiny přečtou z MIB stromů měřených zařízení. Výsledné hodnoty se pošlou zpátky na server a měřící vlákno se uspí na dobu určenou konstantou measuretime, která je definovaná v config.xml. Podobným způsobem pracuje i vlákno „Order Thread“, které se ale místo měření pokouší změnit hodnoty z MIB stromů na požadované hodnoty získanou ze serveru. Dalším vláknem „Discovery Thread“ se budeme zabírat v dalších kapitolách.



Obrázek 3.2: Stavový diagram agenta.

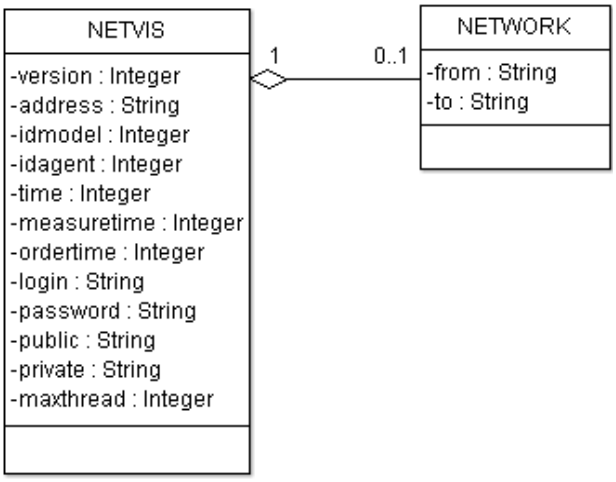
3.3.2 Volba jazyka a knihovny SNMP

Podstatným problémem, jehož řešení může předurčit vlastnosti a funkcionalitu výsledné aplikace, je i volba programovacího jazyka a podpůrných knihoven. Ačkoli i pro jazyk C/C++ existuje celá řada knihoven pro podporu SNMP, tak i přes to výsledná volba padla na jazyk Java. Volba na jazyk Java padla z toho důvodu, že Java je platformě nezávislá, a proto není nutné pro různé platformy vytvářet různé agenty. Menší nevýhodou Javy je, že ani zdaleka neposkytuje tak výraznou podporu pro detekční nástroje jako knihovny C/C++. Pokud bychom chtěli používat nástroje jako traceroute, tcpdump atp., museli bychom použít nativní třídy(JNI s jpcap) a nebo funkci pro spouštění externích aplikací Runtime.exec(). Ale ani jedno z těchto řešení není univerzálně platformě nezávislé. A jelikož nás stejně zadání předurčuje k používání SNMP, nebudeme se jinými technologiemi než SNMP hlouběji zaobírat.

Pro Javu existuje velké množství knihoven[37], které podporují SNMP. Jako například : Westhawk's Java SNMP stack, SNMP4J, Java SNMP API nebo AdventNet SNMP API. Java SNMP API a AdventNet SNMP API jsou sice silně podporované, ale také zpoplatněné knihovny. Westhawk's Java SNMP stack a SNMP4J mají podobnou funkcionalitu. Obě knihovny podporují SNMPv1/v2/v3, odposlouchávání trapů a podporují multithreadový přístup měření. Pro knihovnu SNMP4J jsem se rozhodl, jelikož je uvolněna jako opensource a zároveň má poměrně rozsáhlá diskuzní fóra s řešením řady problémů

3.3.3 Základní konfigurace

Aby bylo možné nastavit základní vlastnosti mnou vytvořeného agenta, je v instalačním balíku přiložen XML soubor config.xml. Při spuštění si agent z tohoto dokumentu vyparseruje potřebné informace, které dále využije při své činnosti. Na obrázku 3.3 je znázorněna struktura tohoto dokumentu. Dokument obsahuje dva elementy NETVIS s NETWORK. Element NETWORK svými atributy udává IP rozsah, na kterém bude agent pracovat. NETVIS je zkratka vzniklá ze slova NETwork VISualisation a element takto označený obsahuje parametry určující funkčnost samotného agenta.



Obrázek 3.3: Struktura XML souboru config.xml.

Konfigurační soubor obsahuje atributy:

- **version** - nepovinny údaj, který určuje vývojovou verzi agenta
- **address** - určuje HTTP adresu nadřazeného serveru.
- **idmodel** - určuje identifikační číslo modelu, ve kterém se následně zobrazí měřená data.
- **idagent** - se používá k odlišení agentů v rámci jednoho modelu.
- **time** - určuje dobu, na kterou se uspí vlákno discoverythread po dokončení jednoho cyklu.
- **measuretime** - určuje dobu, na kterou se uspí vlákno measurethread.
- **ordertime** - určuje dobu, na kterou se uspí vlákno orderthread.
- **login** - autorizační jméno agenta, které je uvedeno v administraci u daného modelu idmodel.
- **password -** autorizační heslo, které je uvedeno v administraci u daného modelu idmodel.
- **public** - community string pro čtení hodnot ze SNMP zařízeních podporující SNMPv1.
- **private** - heslo pro zápis hodnot na SNMP zařízení podporující SNMPv1.
- **maxthread** - maximální počet vláken, které se mohou spustit při v prohledávacím vláknu.

3.3.4 Prohledávací vlákno

Prohledávací vlákno a nebo také discoverythread slouží z získání informací o jednotlivých elementech a jejich portech. Tyto informace následně vyžije server pro prozkoumání topologické struktury a pro vytváření textových výpisů ve formě CSV souborů. V klientské části programu bude následně možné si zobrazit topologii sítě, typ a základní vlastnosti jednotlivých zařízeních a jednotlivé porty a jejich vlastnosti. Data potřebná pro zkonstruování topologie jsou dostupná až na straně serveru, kde je zároveň i prováděn průzkum topologie. A také proto je algoritmus tvorby takovéto topologie popsán právě tam a to včetně zdůvodnění nutnosti měřit některých OID. Proto se při vysvětlování discoverythread omezíme jen na základní vlastnosti. Průběh programu prohledávacího vlákna je zobrazen na obrázku 3.2.

V prvních dvou krocích algoritmu se z nastaveného rozsahu IP adres, který je uložen v config.xml, určí seznam prvků, které podporují protokol SNMP. A u takto zjištěných prvků si zjistíme jejich základní informace zobrazené v tabulce 3.1, kde v první sloupci je OID adresa, ve druhém typ objektu a ve třetím jeho označení.

1.3.6.1.2.1.1.1	DisplayString	sysDescr
1.3.6.1.2.1.1.4	DisplayString	sysContact
1.3.6.1.2.1.1.5	DisplayString	sysName
1.3.6.1.2.1.1.6	DisplayString	sysLocation
1.3.6.1.2.1.1.7	Integer	sysServices
1.3.6.1.2.1.17.1.1	MacAddress	dot1dBaseBridgeAddress
1.3.6.1.2.1.17.1.2	Integer	dot1dBaseNumPorts

Tabulka 3.1: Zobrazení základních informací o daném zařízení.

V prvé řadě si zjistíme popis aktivního prvku sysDescr, emailovou adresu definovaného administrativního pracovníka sysContact, zkrácený název zařízení sysName, umístění prvku sysLocation a reprezentativní MAC adresu dot1dBaseBridgeAddress[39].

Součástí tabulky 3.1 jsou také OID sysServices a dot1dBaseNumPorts, ze kterých si určíme typ zařízení. MIB objekt sysServices je 8bitová hodnota, kde nastavením jednotlivých bitů se deklarují funkcionální možnosti daného zařízení. Např. Je-li 0-tý bit nastaven na 1 znamená to, že dané zařízení pracuje jako repeater, je-li nastaven 1-tý bit na 1, pak dané zařízení pracuje jako switch, je-li nastaven 2-tý bit na 1, pak dané zařízení pracuje jako router, je-li nastaven 3-tý bit na 1, pak dané zařízení pracuje jako host. Zajímavou vlastností je i to že se funkcionalita u některých zařízení může překrývat a dané zařízení může například pracovat jakou router i jako switch. Ačkoli nám sysServices může označit nějaké zařízení jako switch tak to ještě neznamená, že takové zařízení jako switch skutečně pracuje. A z toho důvodu použijeme dot1dBaseNumPorts, který nám určí zda na daném zařízení jsou aktivní porty. Typ zařízení využijeme při vizualizace topologie a také v dalším kroku algoritmu, kde potřebujeme rozhodnout jestli dané zařízení obsahuje ARP tabulku.

Ve třetím kroku algoritmu si zjistíme ARP tabulky ze všech zařízení pracující jako routery. ARP tabulku využijeme při překladu MAC adres na IP adresy u detekovaných koncových zařízení.

1.3.6.1.2.1.4.22.1.2	Integer	ipNetToMediaPhysAddress
1.3.6.1.2.1.4.22.1.3	VlanName	ipNetToMediaNetAddress

Tabulka 3.2: Objekty reprezentující záznamy ARP tabulky.

Ve čtvrtém kroku algoritmu zjistíme identifikační čísla VLAN pro zkoumaná zařízení. Seznam VLAN využijeme při čtení Content Addressable Memory Table (CAM tabulka) pro konkrétní VLANu. Tato adresace probíhá tak, že se v SNMP dotazu za IP adresu zkoumaného zařízení přidá @[idVLAN]. V tabulce 3.3 je uveden způsob jak zjistit identifikační čísla VLAN. Postupně pomocí GETNEXT budeme procházet tabulky vtpVlanState, vtpVlanType a vtpVlanName. A u VLAN kde bude vtpVlanState rové 1 a vtpVlantType také rovno 1 uložní index tabulky do seznamu VLAN. Objekt vtpVlantType=1 nám určuje zda je VLANa ethernetového typu a vtpVlanState=1 zda je VLANa aktivní. Pokud dané zařízení není typu CISCO, pak pomocí GETNEXT si projdeme tabulku dot1qVlanStaticName, kterou by měla podporovat i zařízení jiného typu než je CISCO. Objekty, které nám pomohou k získání informací z ARP tabulky jsou uvedeny v tabulce 3.2. Objekt ipNetToMediaPhysAddress reprezentuje MAC adresu a objekt ipNetToMediaNetAddress jeho IP adresu.

1.3.6.1.4.1.9.9.46.1.3.1.1.2	Integer	vtpVlanState
1.3.6.1.4.1.9.9.46.1.3.1.1.3	VlanName	vtpVlanType
1.3.6.1.4.1.9.9.46.1.3.1.1.4	DisplayString	vtpVlanName
1.3.6.1.2.1.17.7.1.4.3.1.1	SnmpAdminString	dot1qVlanStaticName

Tabulka 3.3: Objekty použité k získání id jednotlivých VLAN.

V pátém kroku algoritmu detekční vlákno převezme u jednotlivých zařízeních důležité informace o jednotlivých interface, které se na daném zařízení nacházejí. K tomuto účelu jsou v tabulce 3.4 určeny základní objekty, které k popisu interface potřebujeme. Objekt ifIndex určuje identifikační číslo interface, které následně použijeme pro propojení informací adresovaných k jednotlivým portům. K tomuto propojení použijeme objekty zobrazené v tabulce 3.5 tj. dot1dBasePort a dot1dBasePortIfIndex. Objekt ifDescr obsahuje popis portu. Objekt ifType obsahuje identifikační číslo, které nám umožní učit typ interface. Těchto typů se mně na internetu podařilo najít 238 a následné jsem vytvořil jsem funkci, která tyto čísla převede na korektní textový popis na straně serveru. Objekt ifType zároveň použijeme i při vizualizaci typů objektů. Například pokud mají interface daného objektu hodnotu 71, pak se daný objekt zobrazí jako Wireless access point(dále jen AP). Dalším objektem je ifPhysAddress, který nám vrátí MAC adresu interface. Objektu ifPhysAddress například použijeme při konstrukci topologie z CAM tabulek, kde potřebujeme zjistit jestli daná MAC adresa nepatří nějakému aktivnímu switchi, protože současné L3 switche mohou mít více jak jednu MAC adresu. A konečně objekt ifOperStatus nám určí zda není daný interface zakázán(1=up,2=down).

1.3.6.1.2.1.2.2.1.1	Integer	ifIndex
1.3.6.1.2.1.2.2.1.2	DisplayString	ifDescr
1.3.6.1.2.1.2.2.1.3	Integer	ifType
1.3.6.1.2.1.2.2.1.6	PhysAddress	ifPhysAddress
1.3.6.1.2.1.2.2.1.8	Integer	ifOperStatus

Tabulka 3.4: Objekty použité k získání informací o portech.

1.3.6.1.2.1.17.1.4.1.1	Integer	dot1dBasePort
1.3.6.1.2.1.17.1.4.1.2	Integer	dot1dBasePortIfIndex

Tabulka 3.5: Objekty k získání id portů a id interface.

V šestém kroku algoritmu si pro jednotlivé porty a VLANy zjistíme hodnoty zanechané v MIB od Spanning tree protocol (STP). Tyto hodnoty dále použijeme při konstrukci topologie. Objekt dot1dStpPortState nám určuje jestli skončilo učení STP a zda port pracuje v normálním režimu[40]. Nás tedy hlavně zajímají ty porty, které mají dot1dStpPortState nestaven na forwarding. Objekt dot1dStpPortEnable nám určuje jestli na daném portu jsou připojena nějaká aktivní zařízení a tedy jestli na daném portu vůbec je provozován STP. Velice důležité objekty jsou dot1dStpPortDesignatedBridge a dot1dStpPortDesignatedPort, které nám vrátí identifikační číslo elementu a jeho port. DesignatedBridge je unikátní pro daný segment a označuje prvek, který je nejblíže rootu. Důležitou vlastností DesignatedBridge je to, že jej má zaznamenán i port, který je vlivem STP odpojen. Identifikační číslo switche je 8bytová hodnota, která se skládá z priority daného prvku a z jeho MAC adresy. V našem případě budeme potřebovat pouze MAC adresu.

1.3.6.1.2.1.17.2.15.1.3	Integer	dot1dStpPortState
1.3.6.1.2.1.17.2.15.1.4	Integer	dot1dStpPortEnable
1.3.6.1.2.1.17.2.15.1.8	BridgeId	dot1dStpPortDesignatedBridge
1.3.6.1.2.1.17.2.15.1.9	OctetString	dot1dStpPortDesignatedPort

Tabulka 3.6: Objekty pro podporu STP.

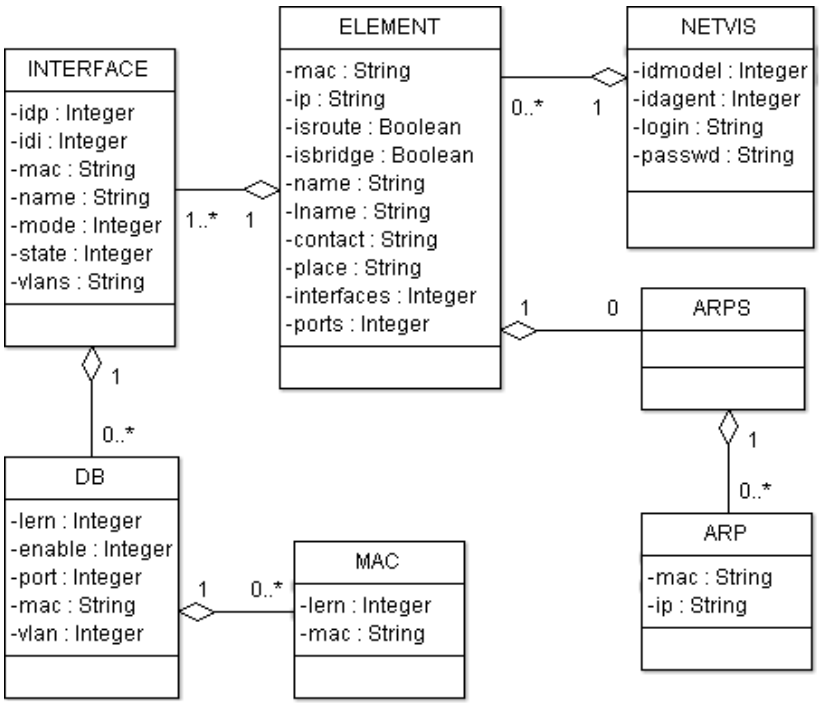
V sedmém kroku algoritmu si pro každý port a každou VLANu přečteme CAM tabulku a to pomocí objektů uvedených v tabulce 3.7. Objekt dot1dTpFdbAddress označuje cílovou MAC adresu a objekt dot1dTpFdbPort označuje port, na který se daný rámec pošle. Objekt dot1dTpFdbStatus nám zase určí jestli je daná MAC adresa již naučena(3=lerned).

1.3.6.1.2.1.17.4.3.1.1	MacAddress	dot1dTpFdbAddress
1.3.6.1.2.1.17.4.3.1.2	Integer	dot1dTpFdbPort
1.3.6.1.2.1.17.4.3.1.3	Integer	dot1dTpFdbStatus

Tabulka 3.7: Objekty pro čtení CAM tabulky.

V osmém kroku algoritmu budou změřené informace poslány nadřazenému serveru ve formě XML souboru. Struktura tohoto souboru je popsaném na obrázku 3.4. Jmenovitě bude XML soubor poslán skriptu setagent.php, který vytvoří instanci objektu BaseParser. BaseParser zpracuje příchozí XML soubor a následně upraví příslušné položky v databázi.

Názvy elementů a jejich atributů jsou voleny taky, aby bylo intuitivně zřejmé, které informace jsou ve kterém elementu a atributu přiřazeny. Cílem toho pojmenování je zajistit snadnou srozumitelnost komunikace mezi agentem a serverem, tak aby se usnadnil případný další vývoj nových agentů, jenž snadno navážou na popisované komunikační schéma. Kořenovým elementem je znovu element NETVIS, který obsahuje atributy pro správnou autorizaci. NETVIS v sobě obsahuje elementy typu ELEMENT, které obsahují atributy odvozené ve druhém kroku algoritmu. Např. atribut name je odvozen z objektu sysName, atribut lname z objektu sysDescr a další. ELEMEMT zároveň obsahuje redundantní informace jako je interfaces resp. ports, které určují počet interface resp. počet portů daného elementu. ELEMENT obsahuje elementy INTERFACE a také může obsahovat elementy obsahující záznamy z ARP tabulek. Element INTERFACE reprezentuje informace získané v pátém kroku algoritmu. Např. idp určuje pořadové číslo portu, idi pořadové číslo interface a vlans obsahuje redundantní informace o VLANách vyskytujících se na daném interface. Pro každou takovou VLANu může existovat vnořený element DB obsahující informace ze STP a obsahující element MAC obsahující informace z CAM tabulek.



Obrázek 3.4: XML dokument, který obsahuje data o zkoumané síti.

Po ukončení jednoho cyklu prohledávání se vlákno algoritmu uspí a znovu se spustí až po uplynutí doby určenou atributem time, který je definován v konfiguračním souboru.

Ve výsledné aplikaci je zároveň vytvořena podpora pro odposlouchávání trapů. Takže pokud se v síti vyvolá trap a agentovi se jej podaří odchytit listenerem, pak je vlákno algoritmu probuzeno a může pokračovat v práci.

3.3.5 Vlákno pro zpracování příkazů

Další vlákno, které se spustí v agentovi je orderthread. Činnost vlákna je znázorněna na obrázku 3.2. V prvním kroku se vyjmou z databáze příslušné příkazy. Tyto příkazy obsahují IP adresu zařízení, pořadové číslo interface, OID adresu objektu a hodnotu, kterou mu nastavíme. V dalším kroku se dané příkazy provedou. V současné době se nastavuje pouze objekt ifAdminStatus znázorněný v tabulce 3.8, kterým můžeme aktivovat nebo deaktivovat daný interface. A po ukončení výpočtu se vlákno uspí po dobu ordertime, která byla načtena z konfiguračního souboru.

1.3.6.1.2.1.2.2.1.7	Integer	ifAdminStatus

Tabulka 3.8: Objekty pro nastavení ifAdminStatus

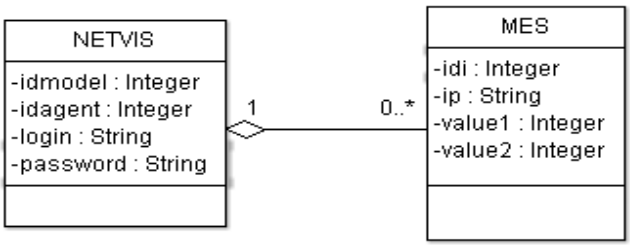
3.3.6 Vlákno pro měření časových průběhů

Další vlákno, které se spustí v agentovi je measurethread resp. vlákno pro měření časových průběhů. Činnost vlákna je znázorněna na obrázku 3.2. V prvním kroku se načtou požadavky na měření. Tyto požadavky obsahují IP adresu zařízení, pořadové číslo interface a unikátní identifikační číslo portu převzaté z databáze. Přednastaveně se měří pouze časový průběh hodnot objektů ifInOctets a IfOutOctets, jak je znázorněno v tabulce 3.9. Objekt ifInOctets udává aktuální hodnotu čítače, který měří počet vstupních bytů a ifOutOctets udává aktuální hodnotu čítače měřícího aktuální počet výstupních bytů.

V následné vizualizaci, ale nepotřebujeme znát hodnotu čítače, ale rozdíl jeho hodnot. Jelikož si agent neuchovává historii, je tento rozdíl vypočítán až na straně serveru, kde je udržována poslední hodnota čítače, která umožní vypočítat rozdíl hodnot a také ošetřit případné přetečení čítače typu Counter32 i typu Counter64 . A tato rozdílová hodnota se posléze uloží v databázi. Samozřejmě by bylo možné ukládat v databázi přímo hodnotu čítačů, ale při následné vizualizaci grafů by vznikl problém při pracném počítání rozdílů hodnot pro vizualizované dlouhé období. Naměřené údaje jsou zpět posílány serveru ve formě XML dokumentu popsaného na obrázku 3.5. A po odeslání dat měřící vlákno čeká po dobu measuretime, která byl zjištěna z konfiguračního souboru, na další spuštění.

1.3.6.1.2.1.2.2.1.10	Counter	ifInOctets
1.3.6.1.2.1.2.2.1.16	Counter	ifOutOctets

Tabulka 3.9: Objekty pro získání informací o zatížení portů.



Obrázek 3.5: Popis XML dokumentu vytvořeného měřícím vláknem.

3.4 Server

3.4.1 Požadavky kladené na server

Za serverovou část naší aplikace se považuje skupina skriptů a modulů, které běží na systému, který je specializován pro poskytování webových služeb. Jak již bylo výše zmíněno, server

přijímá data od více agentů a podle toho vhodně upravuje informace v databázi. Agent předává serveru některé svoje atributy, kterými server kontroluje oprávněnost daného agenta. A pokud se agentovi podaří autorizovat, pak server převezme jeho XML soubory a vydoluje z nich potřebné informace a zároveň odpoví agentovi na případné dotazy. Server také vytváří uživatelské webové rozhraní, které generuje za pomoci PHP skriptů. Součásti tohoto rozhraní je také vizualizace síťové topologie vybraného modelu. Kde tato topologie je vybudována z měřených dat, která jsou poskytována agenty a nebo virtuálně administrátorem. Přes uživatelské rozhraní se uživatel může také autorizovat a tím se mu zobrazí další funkce, které může používat při práci s aplikací. Autorizovaný uživatel například může vytvářet, mazat a upravovat jednotlivé modely, registrovat jednotlivé porty k měření průtoku dat, prohlížet si jednotlivé grafy, zobrazovat si CSV soubory, vkládat do modelu virtuální elementy a upravovat vlastnosti těchto elementů. V rámci dalších odstavců si popíšeme některé základní vlastnosti a funkce serveru.

3.4.2 Volba technického řešení

Aby mohla serverová část aplikace korektně fungovat, tak pro svoji činnost potřebuje webový server s podporu skriptovacího jazyku a databázi. Webový server bude obsluhovat požadavky od webových klientů a spolu s interpretem skriptovacího jazyka bude realizovat zdrojové skripty serverové části naší aplikace. Daný skriptovací jazyk by měl mít zabudovanou podporu pro práci s XML dokumenty a podporu pro realizování operací nad zvolenou databází. V současné době existuje velké množství skriptovacích jazyků (ASP, PHP, Cold Fusion, ...), webových serverů (Apache, IIS, lighttpd) a databází (MySQL, ODBC, Oracle, PostgreSQL, MSSQL...), které podporují naše požadavky. Ale aby se podpořilo rozšíření celé aplikace je jistě vhodnější aby se použila technologie, která je již značně rozšířená a která je poskytována zdarma. Z toho hlediska se jeví jako nejvýhodnější kombinace webového serveru Apache, skriptovacího jazyka PHP a databáze MySQL. Kombinace těchto technologií se totiž považuje za nejrozšířenější [41]

3.4.3 Konstrukce topologie

Pro konstrukci topologie je možné použít různé technologie, kde některé si popíšeme v následujících odstavcích. Při rekonstrukci budeme vycházet z možností protokolu SNMP, a proto se budeme zaměřovat na technologie, které SNMP podporují.

3.4.3.1 Použití specializovaných protokolů

Čtenáře této práce může napadnout použít protokol CDP, který byl popsán v kapitole o nástroji NeDi. CDP je protokolem, který byl vyvinut a je podporován společností CISCO[35]. CDP umožňuje pro jednotlivé elementy detekovat jejich okolní zařízení. A zároveň informace od CDP jsou dostupné za pomoci SNMP protokolu a to v tabulce cdpCacheTable. V tabulce 3.10

jsou uvedeny některé objekty, které se mohou použít při konstrukci topologie. Propojení prvků v síti by se pak jednoduše provedlo tak, že bychom pomocí CDP informací procházeli jednotlivé prvky a ty bychom propojovali s danými sousedy. Značnou nevýhodu takového postupu je ale fakt, že ne všechny zařízení podporují CDP. Většinou se uvádějí tři typy takových to zařízení První typ jsou zařízení, které plně podporují CDP. Druhým typem jsou zařízení které jen přeposílají CDP zprávy, ale sami CDP zprávy neposílají. A konečně třetím typem jsou zařízení, které nepodporují CPD ani nepřeposílají CDP zprávy. Pokud bychom tedy konstrukci topologie založili na CDP pak je zřejmé, že je to možné jen na zařízeních typu CISCO.

1.3.6.1.4.1.9.9.23.1.2.1.1.3	CiscoNetworkProtocol	cdpCacheAddressType
1.3.6.1.4.1.9.9.23.1.2.1.1.4	CiscoNetworkAddress	cdpCacheAddress
1.3.6.1.4.1.9.9.23.1.2.1.1.7	DisplayString	cdpCacheDevicePort

Tabulka 3.10: Objekty pro získání informací z CDP

Další z možných technologií, která je pro konstrukci topologie využitelná je i LLDP protokol. Tento protokol je novější oproti CDP a je i podporován větší skupinou výrobců oproti CDP[42]. Protokol byl formálně přijat jako standart IEEE 802.1AB v roce 2005. LLDP měl nahradit proprietární protokoly jako je CDP, Extreme Discovery protokol nebo Nortel Discovery protokol (dále jen SONMP). Zařízení podporující LLDP si také uchovávají hodnoty o okolních zařízeních ve svém MIB stromu. A tedy při následné konstrukci topologie se projdou tyto záznamy podobně jako u CDP protokolu. Jeho nevýhodou je to, že jej nepodporují starší zařízení a zároveň není podporován univerzálně u všech výrobců aktivních zařízení.

Shrneme-li obecné nedostatky těchto protokolů, pak se tedy jedná prvořadě o neuniverzálnost, ale také o neschopnost zjišťování informace o jednotlivých koncových počítačích, které jsou v síti zapojeny. V dalším textu se jak CDP tak LLDP věnovat nebudeme, a proto odkážeme zájemce na výše přiložené reference.

3.4.3.2 Použití STP a tabulky MAC adres

Jednou z možností jak zkonstruovat požadovanou topologii je i použitím informací obsažených v CAM tabulce. Jak již bylo dříve zmíněno, CAM tabulka obsahuje informace, které daný switch použije při přeposílání příchozích rámců. CAM tabulka nebo také „MAC address table“ totiž realizuje funkci, která každé cílové MAC adrese příchozího rámce přiřadí výstupní port switche. Aby nedocházelo k nežádoucím smyčkám, tak většina switchů podporuje STP. STP totiž vhodně zablokuje komunikaci na některých portech tak, aby v síti nevznikali smyčky v rámci dané VLANy. Jelikož máme všechny informace již uloženy v databázi, pak je ve výsledku jasné, že tímto způsobem můžeme získat seznam dostupných MAC adres u každého portu zkoumaného switche. Včetně toho, že jsou pak zjistitelné zajímavé statistické informace o dané síti. Jako např. počet unikátních MAC adres v síti atp.. Ale problém je to, že switche mohou obsahovat velmi velké množství záznamů MAC adres. A proto jednou z nevýhod

algoritmů[43], jenž používají CAM tabulku, jsou velké přenosy dat v rozlehlých sítích. Další podstatnou nevýhodou je to, že STP nám ze skutečné topologie vytvoří její kostru. Například pokud máme jednoduchou kruhovou síť, pak po určité době STP zablokuje jeden z portů tak aby se odstranila smyčka, ale tím se také zneviditelní záznamy z CAM tabulky zablokovaného portu a ve výsledku se při vizualizaci nezobrazí kruhová topologie.

Jednou ze zajímavých studií, která se pokouší řešit výše popsané nevýhody je i „Physical Topology Discovery for Metro Ehernet Networks“ od autorů Myung-Hee Son, Bheom-Soon Joo, Byung-Chul Kim, and Jae-Yong Lee [44]. Tento algoritmus pracuje ve dvou krocích.

V prvním kroku se aktivní switche rozdělí do dvou kategorií na switche okrajové a switche centrální. Okrajové switche jsou takové switche, které mají na svých portech připojeno alespoň jedno koncové zařízení (host). A centrální switche jsou takové switche, které nejsou připojeny k žádnému koncovému zařízení a tedy mohou být připojeny pouze k dalším switchům. K roztřídění switchů na okrajové a centrální se použije dot1dStpPortEnable viz. tabulka 3.6 a objekt dot1dBaseNumPorts. Objekt dot1dStpPortEnable určuje zda na daném portu je nutné spouštět STP, resp. jestli na daném portu je krom koncových a transparentních zřízeních také připojen nějaký aktivní prvek. A objekt dot1dBaseNumPorts určuje počet aktivních portů na daném switchi. A odtečeme-li počet portů, kde je spuštěno STP, od celkového počtu aktivních portů dostaneme počet portů, na kterých nejsou aktivní switche. A tím si i určíme jestli dané zařízení je okrajové nebo centrální.

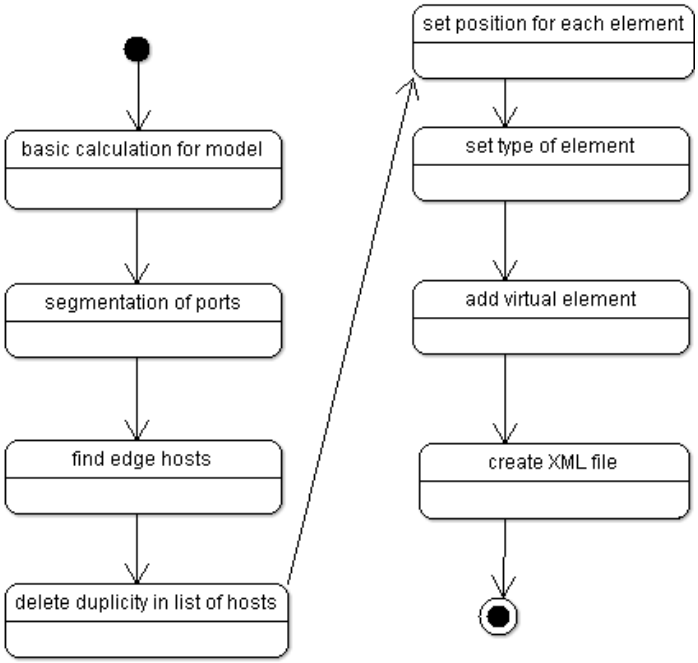
Ve druhém kroku algoritmu se čtou CAM tabulky pouze na okrajových switchích a to pouze na portech, na kterých není povoleno STP a tedy ani není připojen aktivní prvek. Jelikož dané porty nevedou na jiný aktivní zařízení, pak daná CAM tabulka obsahuje seznam koncových zařízení, která jsou na daném portu připojena. Ale jak se vyřeší propoje mezi jednotlivými switchi? K tomu se použijí objekty dot1dStpPortDesignatedBridge a dot1dStpPort-DesignatedPort popsané v tabulce 3.6. Tyto hodnoty jsou v rámci jednoho segmentu sítě unikátní a STP je používá k určení prvku s nejvyšší prioritou. Tímto způsobem spojíme porty do segmentů a určíme vazby mezi jednotlivými porty switchů.

Hlavní výhodou tohoto algoritmu je to, že rozdělením switchů do dvou kategorii zabráníme zbytečnému čtení obsáhlých CAM tabulek z centrálních prvků. Dále algoritmus poukazuje na dvě krásné myšlenky. A to že můžeme jednotlivé porty rozškatulkovat do segmentů za pomoci dot1dStpPortDesignatedBridge a dot1dStpPortDesignatedPort a že pokud na daném portu není aktivní element, pak CAM tabulka na tomto portu obsahuje MAC adresy koncových zařízení, které jsou na daném portu již reálně zapojeny například s dalšími transparentnímy prvky.

Hlavním problémem této aplikace je to, že ignoruje možnou existenci hubů a koncových prvků na spojích mezi aktivními prvky a že vychází z předpokladu, že na síti se nepoužívají VLANy. Další problém se odhalil při testování, kdy se přišlo na to, že dot1dStpPortEnable se často nastavuje na true i na portech, kde nejsou žádná aktivní zařízení.

3.4.3.3 Popis konstrukce topologie

V rámci tohoto odstavce si popíšeme algoritmus, který byl vytvořen pro získání informací, které budou následně použity při konstrukci topologie zkoumané sítě. Tento algoritmus je uložen v souboru getmodel.xml a jedná se o PHP skript, který generuje XML soubor, který bude následně použit při výsledné vizualizace klientem. Takže cílem tohoto algoritmu je zpracovat připravená data, která agenti nabírali při distribuovaném procházením sítě a vygenerovat XML soubor, který bude reprezentovat zkoumanou síť. Tento XML soubor by měl tedy obsahovat elementy reprezentující koncové stanice, aktivní prvky, transparentní zařízení, virtuální prvky a informace o jejich vzájemném propojení. Jednoduchý nástin algoritmu je uveden v obrázku 3.6.



Obrázek 3.6: Struktura PHP skriptu getmodel.xml.

V prvním kroku algoritmu se provedou základní výpočty a inicializace struktur, které budeme dále používat při zpracování topologie vybraného modelu. V prvé řadě se vyplní asociativní pole, které bude obsahovat záznamy z ARP tabulky, tak aby bylo možné překládat MAC adresy koncových zařízeních. Zároveň se vyplní funkce, která deklaruje příslušnost MAC adresy k nějakému aktivnímu prvku, ze kterého máme informace ze SNMP. Jde o to, že řada zařízení pracující jako switche používá při práci se STP více jak jednu MAC adresu, a proto je tato funkce zapotřebí, aby při průchodu CAM tabulek se dvě rozdílné MAC adresy neinterpretovali jako rozdílná zařízení. Dále je inicializováno pole seznamů „SEGMENT“, které bude pro každý segment obsahovat identifikační čísla portů, která se v tomto segmentu budou vyskytovat. Dále se budou inicializovat proměnné, které pro dané elementy uloží jejich vypočítanou pozici, která se následně zapíše do příslušných atributů při výpisu XML souboru.

Ve druhém kroku algoritmu se pro všechny porty a jimi podporované VLANy určí jejich Designated Bridege a Designated Port. Za pomoci těchto hodnot se rozškatulkují jednotlivé

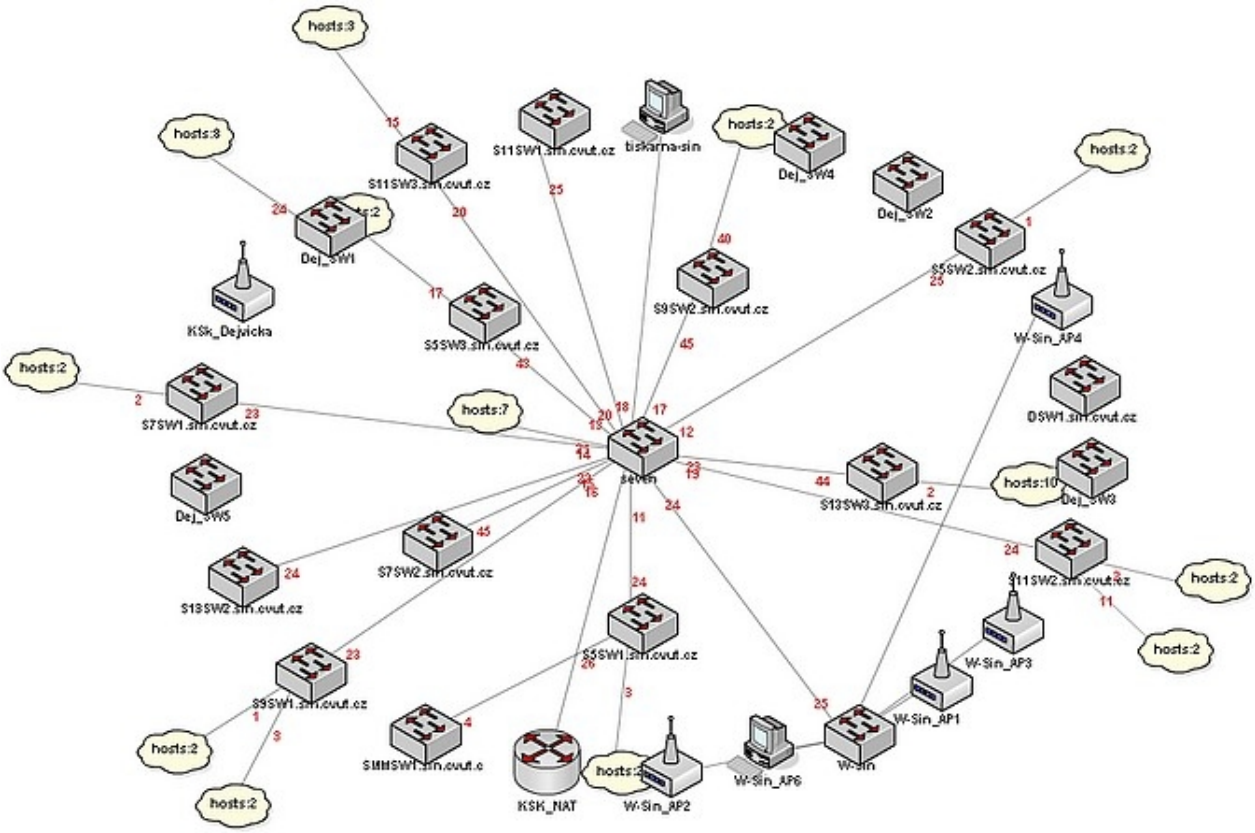
porty do segmentů a podle počtu portů v takovýchto segmentech se následně rozhodne jestli mezi aktivními elementy zobrazíme hub nebo jen jednoduché spojení, jednu koncovou stanici a nebo nic a tím se vyvarujeme jedné z nevýhod u dříve popsaného algoritmu[44]. Celý druhý krok algoritmu probíhá tak, že si tedy k prvnímu zdrojovému portu za pomoci identifikačního čísla elementu Designated Bridege určíme identifikační číslo druhého portu Designated Port. Následně se zjišťuje příslušnost těchto portů k již vytvořeným segmentům. Pokud se oba porty v žádném segmentu nenacházejí a oba pocházejí z odlišných elementů, pak se vytvoří nový segment, do kterého se vloží oba tyto porty. Je-li jeden z těchto portů již přiřazen nějakému segmentu, pak i druhý port se přiřadí do téhož segmentu. Pokud se oba porty nacházejí v odlišných segmentech pak se tyto segmenty sloučí do jednoho. Tímto druhým krokem se zároveň zbavujeme některých omezení ,která souvisejí s používáním VLAN. A to tím způsobem, že slučujeme informace získaných z VLAN. Mějme například port na jednom switchi, který podporuje 2 různé VLANy a má tedy přiděleny dvě různé dvojice Designated Bridge a Designated Port. Jelikož tyto dvě dvojice jsou přímo dostupné ze zdrojového portu, pak to znamená, že všechny tři porty těchto elementů patří do stejného segmentu.

Třetí krok algoritmu představuje část, která je nejnáročnější na výpočet, protože je v ní nutné projít informace, které byli převzaty z CAM tabulek Cílem je to, že si určíme, která koncová zařízení jsou zapojená ve kterých segmentech. Abychom si tuto informaci zjistili, tak si nejprv určíme vstupní MAC adresy a to tak, že pro všechny porty daného segmentu si přečteme CAM tabulku a tím získáme množinu vstupních MAC adres. Zároveň potřebujeme vědět výstupní MAC adresy pro každý element, který je včleněný do právě zkoumaného segmentu. Odstraněním výstupních MAC adres z množiny vstupních adres získáme malé množství MAC adres, které sice CAM tabulky navedou do zkoumaného segmentu, ale již je neadresují ven. Tyto MAC adresy uložíme do pole seznamů HOST[idSegment] resp. EXTERN[idHost] podle toho jestli daná MAC adresa ukazuje na koncové zařízení resp. na aktivní zařízení. Tj. získáme MAC adresy koncových zařízeních, která jsou přímo zapojena v daném segmentu. Tedy tímto postupem odstraníme další z nevýhod algoritmu popsaného v předchozí kapitole [44]. Při tvorbě výsledného XML souboru, budeme vizualizovat daný segment i podle toho kolik výstupních MAC adres obsahuje. Například pokud segment obsahuje pouze jeden port, ale zároveň obsahuje více jak jedno koncové zařízení, pak jej vizualizujeme jako hub s několika okolními hosty. Na druhou stranu pokud segment obsahuje jen 2 porty, ale má alespoň jedeno koncové zařízení, pak tento segment nebude zobrazen jako spoj mezi dvěma aktivním elementy, ale jako hub ke kterému jsou tyto elementy připojeny spolu s danými koncovými zařízeními. Další vizualizační záležitostí je, že pokud daný segment obsahuje jen jeden port a jen jedno koncové zařízení pak se dané koncové zařízení zobrazí jako závislý prvek zobrazený hned vedle zdrojového elementu.

Pokud bychom ve třetím kroku skončili, pak by výsledná topologie byla zobrazena korektně. Ale mohou nastat problémy s existencí aktivních prvků, které nepodporují STP a nebo jinak nevyplňují objekty v MIB stromu, které algoritmus využívá. Budeme-li mít takové zařízení a přečteme-li si jeho CAM tabulku, pak většina těchto záznamů, která je v CAM tabulce, bude považována za koncová zařízení. Abychom vyřešili tento problém, tak ve čtvrtém kroku algoritmu smažeme duplicitní koncová zařízení. A to takovým způsobem, že po třetím kroku algoritmu si projdeme všechny seznamy HOST i EXTERN. A pokud je nějaká MAC adresa obsažena v nějakém segmentu v seznamu HOST a přitom tento segment má prázdný seznam EXTERN pak daná MAC finálně patří do tohoto segmentu a při jejím případném

výskytu v jiných segmentech ji smažeme. Tato technologie byla úspěšně testována na síti obsahující aktivní elementy od výrobce CISCO a 3Com.

V dalším kroku se určí rozmístění elementů při výsledné vizualizaci. Tato část algoritmu probíhá tak, že se elementy nejprve sestupně setřídí podle počtu výskytů v segmentech, které obsahují alespoň dva porty. A následně se postupně vykreslují. První element se vykreslí ve středu plochy, dalších 20% elementů se vykreslí ve vnitřní elipse a ostatní elementy ve vnější elipse. Úhel otočení je definován tak aby byli elementy v jednotlivých elipsách rovnoměrně rozloženy. Ukázka takového vykreslení u uvedena na obrázku 3.7. Toto rozmístění se uplatní, pokud nejsou souřadnice elementů již předem definovány administrátorem přes uživatelské rozhraní jako například při prvním načtení hodnot agentem.



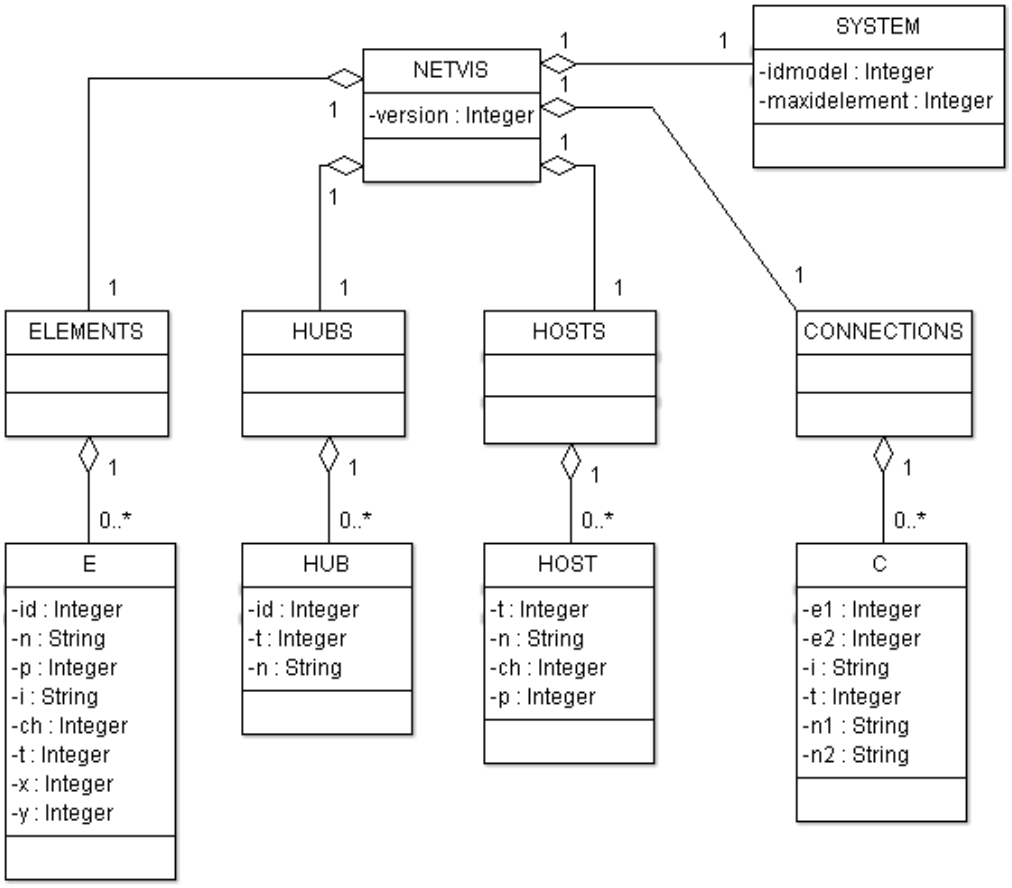
Obrázek 3.7: Rozmístění prvků prvotním algoritmem bez koncových zařízení.

.

V šestém kroku algoritmu se určí grafický typ zobrazovaného zařízení. Tento typ je hlavně určen zvoleným grafickým stylem a tím jestli dané zařízení má nastaveno v databázi atributy isroute a isbridge, který charakterizují některé základní vlastnosti zkoumaných elementů. Například pokud dané zařízení obsahuje porty typu 71, pak je vykresleno jako AP. A pokud dané zařízeni je virtuální pak se přebírá administrátorem definovaný typ. Ukázky takového vykreslení jsou zobrazeny v příloze.

V sedmém kroku se pouze přidají virtuální elementy, které byli administrátorem vytvořeny v modelu pomocí uživatelského rozhraní Adobe Flash a přidají se spoje mezi nimi.

V osmém kroku se vytvoří XML soubor, který je následně poslán klientovy za účelem vizualizace naměřených dat. Struktura takového souboru je zobrazena na obrázku 3.8. Pro zajímavost se čtenář může podívat na rozdíl mezi tímto XML dokumentem a XML dokumentem, který posílá agent serveru. Na základě toho porovnání je patrné, že XML dokument určený klientovy obsahuje pouze data, která jsou potřebná pro vizualizaci a zároveň názvy jednotlivých atributů zjednodušuje, tak aby řídící znaky nezabíraly tolik místa ve výsledném dokumentu.



Obrázek 3.8: Struktura XML dokumentu, který popisuje topologii sítě.

3.4.4 Obsluha agenta

Základními nástroji pro komunikaci serveru s agentem jsou PHP třídy BaseParser, OrderParser, MeasureParser a jejich podpůrné skripty measure.php, order.php a setagent.php.

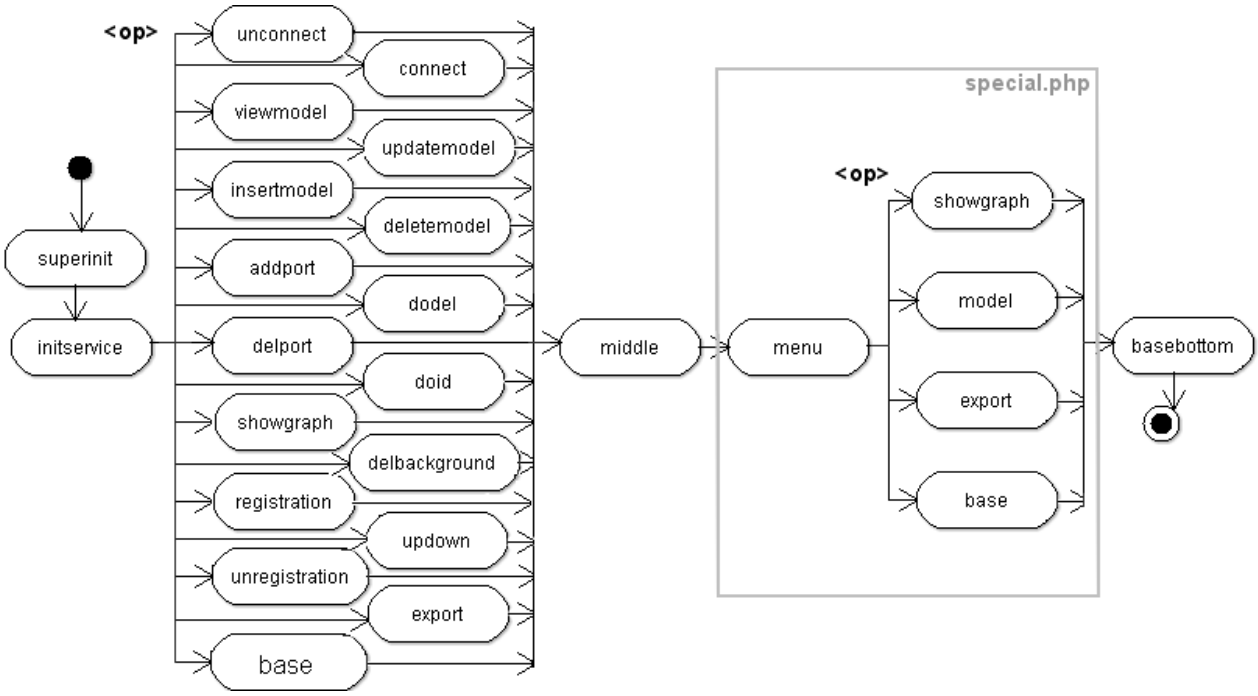
OrderParser přijímá požadavky od vlákna orderthread ve formě velice jednoduchých XML zpráv. Toto agentovo vlákno se u něj autorizuje jménem a heslem a zároveň mu předává svoje identifikační čísla. Pokud jsou všechny údaje správné, pak OrderParser nahlédne do databáze a pokud tam jsou nějaké požadavky na zařízení, které jsou v rozsahu agenta tak je agentovi následně pošle a dané požadavky pak odstraní z databáze.

Třída MeasureParser pracuje podobným způsobem jako třída OrderParser. Detekuje požadavky od vláken typu measurethread a pokud se danému vláknu podaří autorizovat, pak mu pošle seznam IP adres měřených elementů, identifikačních čísel interface v rámci daného elementu a identifikační číslo portu v rámci databáze. Vlákno measurethread začne měřit veličiny ifInOctets a IfOutOctets na zařízení, které je určeno předanou IP adresou a na interface, které bylo také předáno třídou MeasurParser. Následně agent pošle příslušné odpovědi znovu třídě MeasurePareser. MeasurePareser rozpozná v příchozím souboru XML element MES a pokud se schodují vrácená identifikační čísla portů s identifikačním číslem již existujícího portu v databázi a vlastník tohoto portu je v rozsahu agenta, pak je daná hodnota přijata a dále zpracována. Nezapomínejme, že přijatá veličina udává velikost hodnoty v čítači Counter. Abychom určili průtok daným portem potřebujeme zjistit rozdíl hodnot a to tak, že od právě změřené hodnoty odečteme uloženou hodnou z předchozí komunikace s agentem. K tomuto účelu je tabulka measure, které obsahuje měřená data, vybavena atributem isvar. Isvar je atribut, který deklaruje zda hodnota v tabulce je požadovaným rozdílem a nebo přečtenou hodnotou čítače. Výsledná úprava hodnot pak probíhá tak, že se přečte naměřená hodnota s atributem isvar=true a tato hodnota se modulárně odečte od příchozí hodnoty. Tímto způsobem zjistíme hledanou hodnotu průtoku, kterou následně zapíšeme do databáze spolu s aktualizací položky v databázi, která určuje poslední hodnoty čítače.

Třída BaseParse je hlavní a nejrozsáhlejší XML parser na serveru. Tato třída zpracovává XML dokumenty, které již byli dříve zobrazeny na obrázku 3.4. Začátek komunikace vlákna discoverythread se třídou BaseParser probíhá podobně jako v předchozích případech. Agentské vlákno spolu s objemným balíkem hodnot pošle zároveň autorizační a identifikační údaje. Identifikační údaj idmodel slouží k tomu aby se nahrály resp. upravili data ve správném modelu. A identifikační údaj idagent rozlišuji různé agenty v rámci jednoho modelu. A tedy idagent podporuje rozparcelování zkoumané sítě pro více agentů a také umožňuje shromažďování naměřených údajů z více oddělených sítí tak abych např. několik malých sítí bylo vizualizováno na jedné pracovní ploše. Po autorizování BaseParser přijme XML soubor a postupně začne zpracovávat jednotlivé informace. Pokud pro nalezené zařízení obsažené v XML souboru neexistuje ekvivalent v databázi, pak BaseParser toto zařízení vytvoří. Pokud dané zařízení v databázi již existuje tak BaseParser aktualizuje informace o zařízení v databázi, krom informací jako je x-ová, y-ová pozice a rodič. Tyto informace se neaktualizují, protože se ze SNMP nezjišťují, ale jsou nastavovány administrátorem přes uživatelské rozhraní. Podobným způsobem se přistupuje i k interface. Pokud daný interface již existuje v databázi ,pak se neupravují příznaky, které určují zda se má měřit zatížení na takovýchto portech. Na druhou stranu položky v databázi, které reprezentují ARP tabulku resp. CAM tabulku a informace o STP se při přečtení příslušného zařízení resp. portu, který tyto informace vlastní, smažou. Následně jsou data nahrazena aktuálnějšími informacemi. Zajímavé je i řešení situace , kdy je dané zařízení ze sítě odstraněno a tedy se ani nevyskytuje ve čteném XML souboru. Totiž ještě před před čtením detailních informací si BarseParser zjistí které elementy jsou danému agentovi v daném modelu přiřazeny. A pokud se dané zařízení ve čteném XML dokumentu nenajde, je následně odstraněno i se všemi příslušnými přidruženými údaji v dalších tabulkách. Proto také každé zařízení, které je uloženo v databázi obsahuje atribut idagent. Další důležitou zajímavostí je i to, že BaseParser nepracuje jako DOM parser, ale jako XML parser postupně zpracovávající jednotlivé elementy. Tento výběr parserování byl vybrán z toho důvodu, tak aby nebylo nutné přehnaně zatěžovat serverovou stanici.

3.4.5 Funkce pro obsluhu klientských požadavků

Krom toho že server vytváří XML dokument pro následnou vizualizaci topologie, tak i zároveň poskytuje podporu pro další funkce klienta. Prvořadě jde o tvorbu a obsluhu kompletního uživatelského rozhraní. O obsluhu a vytváření uživatelského rozhraní se stará třída manager, která je uložena v PHP souboru manager.php. Jednoduchý stavový diagram této třídy je znázorněn na obrázku 3.9. Základním parametrem, který řídí třídu manager, je op. Op vznik zkrácením slova „operation“ a obsahuje zkráceny název operace, kterou vyvolal uživatel webového uživatelského rozhraní.



Obrázek 3.9: Stavový diagram popisující třídu manager.

Při provádění nějaké takové operace se nejprve provede základní inicializace superinit a initservice. Superinit inicializuje session proměnné, připojí se k databázi a načte soubory funkcí fcebase.php a special.php. Initservice zato zpracuje základní vstupní parametry, včetně ošetření zakázaných znaků a načte si session proměnné.

V dalším kroku se podle textové hodnoty proměnné op vyvolá příslušná operace, která upraví informace v databázi a případně i změní některé proměnné, které se dále použijí v další části programu. Connect resp. unconnect se vyvolá pokud se chce uživatel autorizovat resp odhlásit ze systému. Viewmodel, updatemodel, createmodel, detelemodel se použijí při vytváření, úpravě a mazání modelů. Addport resp. delport se využiji u virtuálních elementů, u kterých chceme přidat další porty resp. odebrat stávající. Dodel se využije při mazání virtuálního elementu. Doid se zato využitkuje pro vyvolání poměrně robustní funkce pro úpravu všech portů zvoleného elementu. Funkce volané po doid se používá z toho důvodu aby uživatel nemusel postupně upravovat informace o jednotlivých portech a jejich zapojení, ale pro celou

skupinu portů najednou. Tj. HTML tag „form“ nemusíme používat pro každý řádek zvlášť, ale použijeme pouze jeden pro kompletně celou tabulku portů a jejich připojeních. Registration resp. unregistration se použije při registraci resp. odhlášení měřícího požadavku pro dané porty. Delbackground se zase použije, chce-li uživatel smazat obrázek, tvořící pozadí pracovní plochy vybraného modelu. Updown se vyvolá pokud uživatel vyžaduje povolení resp. zakázání portu. Pokud má op hodnotu „export“ tak algoritmus pro všechna zařízení a jejich porty vytvoří přehledné CSV soubory, které budou obsahovat základní informace o daných elementech a jejich portech. Pokud op nabývá hodnoty „base“ nebo „showgraph“ a nebo uživatel nebyl autorizován, pak se žádné operace neprovádějí a přejde se od dalšího stavu tj. do přechodného stavu middle, který je předělem mezi zpracováním dat managerem a přímým vytvářením HTML tagů, které se posléze pošlou webovému klientovi. Dále se vyvolá funkce, která na základě předaných parametrů vytvoří menu. V dalším kroku se čte parametr op znovu, ale tentokrát již obsahuje jinou hodnotu než na vstupu, protože jednotlivé funkce, které byli vyvolány prvotně si tento parametr změnili. Např. funkce volané v blocích viewmodel, updatemodel, createmodel, detelemodel změní hodnotu proměnné op na „model“. To je zjevná zajímavost a krása tohoto přístupu tj. že i po více jak 15ti různých operací je možné výsledné HTML odpovědi vytvořit jen za pomoci následujících čtyřech funkcí, které jsou reprezentovány čtyřmi stavy.

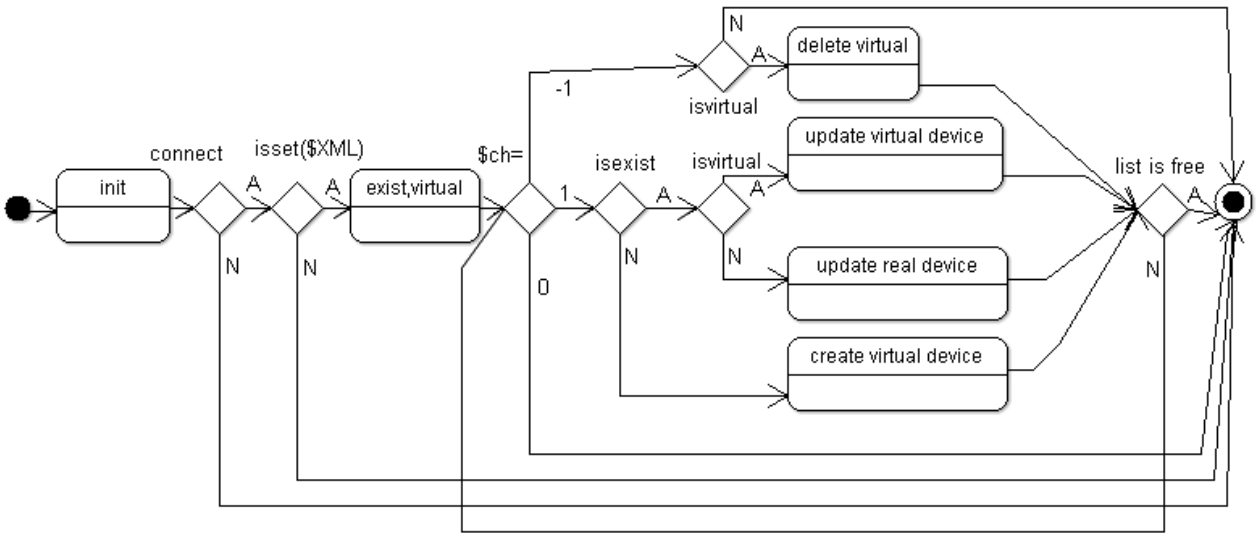
Jednou z těchto funkcí je „export“, která vytvoří uživatelské rozhraní pro prohlížení seznamu CSV souborů. Další funkcí je „model“, která zobrazí uživatelské rozhraní pro vytváření, úpravu i mazání modelů. Důležitou funkcí je „base“, která vygeneruje uživatelské rozhraní, které bude obsahovat jak vizualizaci topologie tak i případný výpis vlastností vybraného elementu i jeho portů. Tento výpis je vygenerován odlišně podle toho jestli je uživatel autorizován a jestli se vypisují informace o virtuálním elementu a nebo o fyzické elementu. Např. pokud není uživatel autorizován pak se mu zobrazí jen informace, bez možnosti cokoli měnit. Pokud je uživatel autorizován a zobrazují se mu informace o reálném elementu, pak může například vnořovat dané elementy do racků a nebo registrovat/odhlašovat měření na portech. Na druhou stranu pokud je uživatel autorizován a zobrazují se mu informace o virtuálním elementu, pak může např. měnit vlastnosti daného zařízení, přidávat, upravovat a mazat jednotlivé porty a upravovat spoje mezi těmito porty.

Poslední z těchto funkcí je „showgraph“, která generuje uživatelské rozhraní obsahující interaktivní graf, jenž zobrazí měřené zatížení portů na třech časových stupnicích. A to po jedné minutě, deseti minutách a po hodině. Daný graf je implementován technologií Flash a data jsou mu předávána pomocí parametru FlashVars, který jsme si popsali v kapitole o Flashi. Komunikace přes FlashVars byla zvolena z toho důvodu, že se požaduje relativně malé množství přiřazených dat. SQL příkaz, který je použit pro zjištění součtů hodnot pro hodinové intervaly je zobrazen na obrázku 3.10. Na první pohled je zřejmá jednoduchost použitého skriptu. A to je právě ono! Pokud bychom totiž do databáze ukládali přímo hodnoty čítačů a rozdíly hodnot zjišťovali před vykreslování grafů, pak by bylo výsledné řešení složitější a časově náročnější.

```
SELECT UNIX_TIMESTAMP(date) as t,
            YEAR(date)      as y,
            MONTH(date)     as mo,
            DAY(date)       as d,
            HOUR(date)      as h,
            sum(value)      as svalue
FROM measure
WHERE idport='$idport' AND form='$form' AND isvar=false
GROUP BY YEAR(date), MONTH(date), DAY(date), HOUR(date)
ORDER BY t desc
LIMIT 0,100;
```

Obrázek 3.10: SQL příkaz, jenž se použije při vizualizaci grafu.

Další důležitou funkcí agenta je zpětné ukládání změn, které provede administrátor ve vizualizované topologii sítě. Tyto změny Flash předá formou XML dokumentu zpět serveru a to s obdobnou strukturou jako je uvedena na obrázku 3.8. Stejná struktura XML dokumentu byla zvolena z důvodu modularity celého systému. Popis funkce je znázorněn na obrázku 3.11. V první kroku se inicializují potřebné knihovny, zjistí se zda se podařilo připojit k databázi a zda se načetl XML soubor. Po té co se načte pole, které deklaruje existenci zařízení v databázi a pole, které určuje zda dané zařízení je virtuální, vyvolá se cyklus, který přečte všechny XML elementy. Důležitým atributem je „ch“, který vznikl zkrácením slova change a deklaruje klientem požadovaný typ změny položky v databázi. Pokud je ch=0 tak to znamená, že daná položka v databázi nebude změněna. Pokud je ch=-1 a pokud je zařízeni virtuální pak takové zařízení bude smazáno. Pokud je ch=1 tak Flash požaduje změny informací o zařízení. Jelikož od posledního předání dokumentu mohlo dojít k odstranění tohoto zařízení, tak se nejprve zjistí zda dané zařízení ještě vůbec existuje. Existuje-li tak se v následném kroku zjistí zda je virtuální či není. Pokud zařízení není virtuální, pak klient má právo pouze měnit pozici takového zařízení a nastavovat jeho rodiče. Na druhou stranu pokud je dané zařízení virtuální pak se informace jako název, popis, IP adresa změní podle toho co uvedl administrátor ve Flashovém uživatelském rozhraní.



Obrázek 3.11: Zpracování příchozího XML souboru skriptem setmodel.php.

3.5 Klient

V předchozí kapitole o serveru bylo popsáno jak se zpracovávají uživatelovi příkazy z klientského rozhraní a jak je takové rozhraní generováno. Zároveň bylo popsáno generování XML souboru, ze kterého se následně vytvoří vizualizovaná topologie a také bylo vysvětleno zpracování XML souboru, který je poslán z klienta zpět na server, a který reprezentoval administrátorovi úpravy, které se následně uložili v databázi. Jelikož uživatelská příručka bude obsahovat popis klientského rozhraní a velká část funkcionality byla již vysvětlena v kapitole o serveru, zaměříme se pouze na popis možných technických řešeních při vizualizaci topologie a konečně na popis konstrukčního algoritmu, který z XML souboru vytvoří působivou vizualizaci topologie zkoumané sítě.

3.5.1 Volba technického řešení

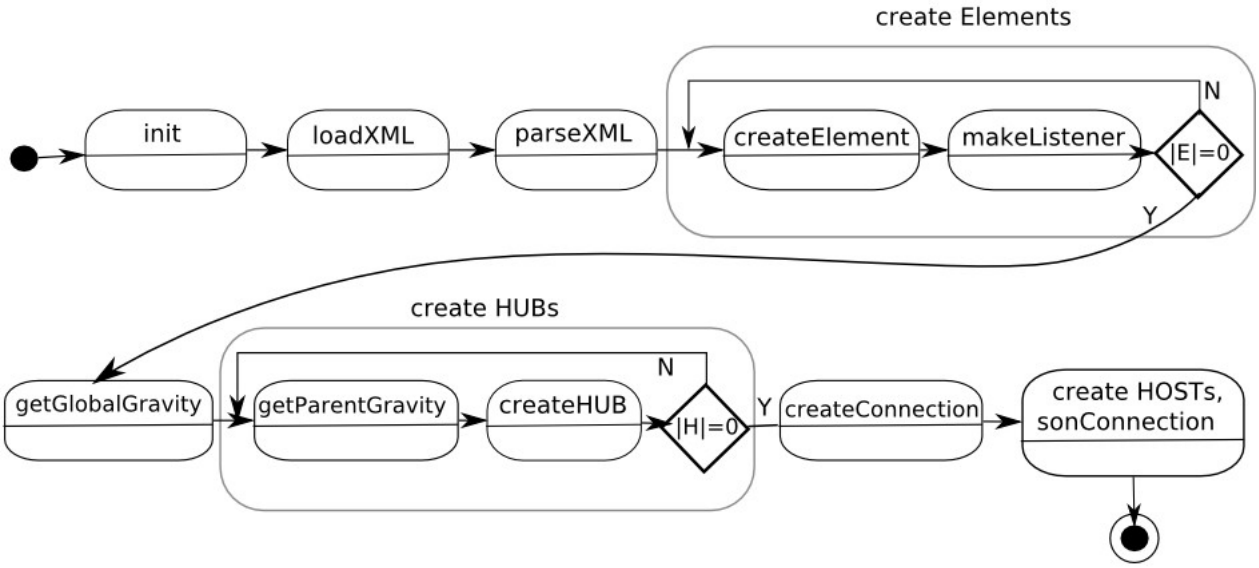
Při volbě technologie jsme uvažovali prvořadě o technologiích SVG, SilverLight a Flash. SVG je vektorový grafický formát založený na XML. SVG je prvořadě založen na vykreslování 2D grafiky, ale zároveň umožňuje také jednoduché interaktivní akce a vytváření animované grafiky. Velkou výhodou, kterou bychom mohli využít je XML základ SVG formátu. Bylo by totiž možné generovat jednoduchý SVG souboru na serveru a ten by si pak zobrazil webový klient za pomoci SVG pluginu. Velkou nevýhodou takové technologie by byla obtížná tvorba složitých interaktivních operací a nemožnost vytvářet virtuální modely sítí.

Další důležitou technologii, která se v tomto případě může použít je SilverLight. SilverLight je technologie, která byla vytvořena společností Microsoft s cílem konkurovat technologii Flash. Jelikož produkty společnosti Microsoft ovládají trh s operačními systémy je zároveň možné, že SilverLight bude postupně vytlačovat technologii Flash, která je brána jako dominantní technologie interaktivní grafiky na poli webových stránek. V rámci naší úlohy jsme se rozhodli pro technologii Flash a to z několika důvodů. Za prvé, že plugin Flash Player je podporován větším procentem uživatelů než další obdobné technologie viz. obrázky 2.8 a 2.7. Dalším důvodem je to, že Flash obsahuje třídy umožňující propracovanou komunikaci FlashPlayeru s webovým serverem a že Flash umožňuje jednoduše zpracovávat XML soubory a tím nám ve výsledku umožní vybudovat aplikaci, která bude díky použití technologie XML velice modulární.

Menší nevýhodou těchto technologií je to že obtížně vizualizují řádková data z proměnlivou velikostí, proto se také Flash požije jen pro vizualizaci grafů a topologie, ale řádková data budou zobrazována standartě pomocí HTML.

3.5.2 Vykreslování topologie

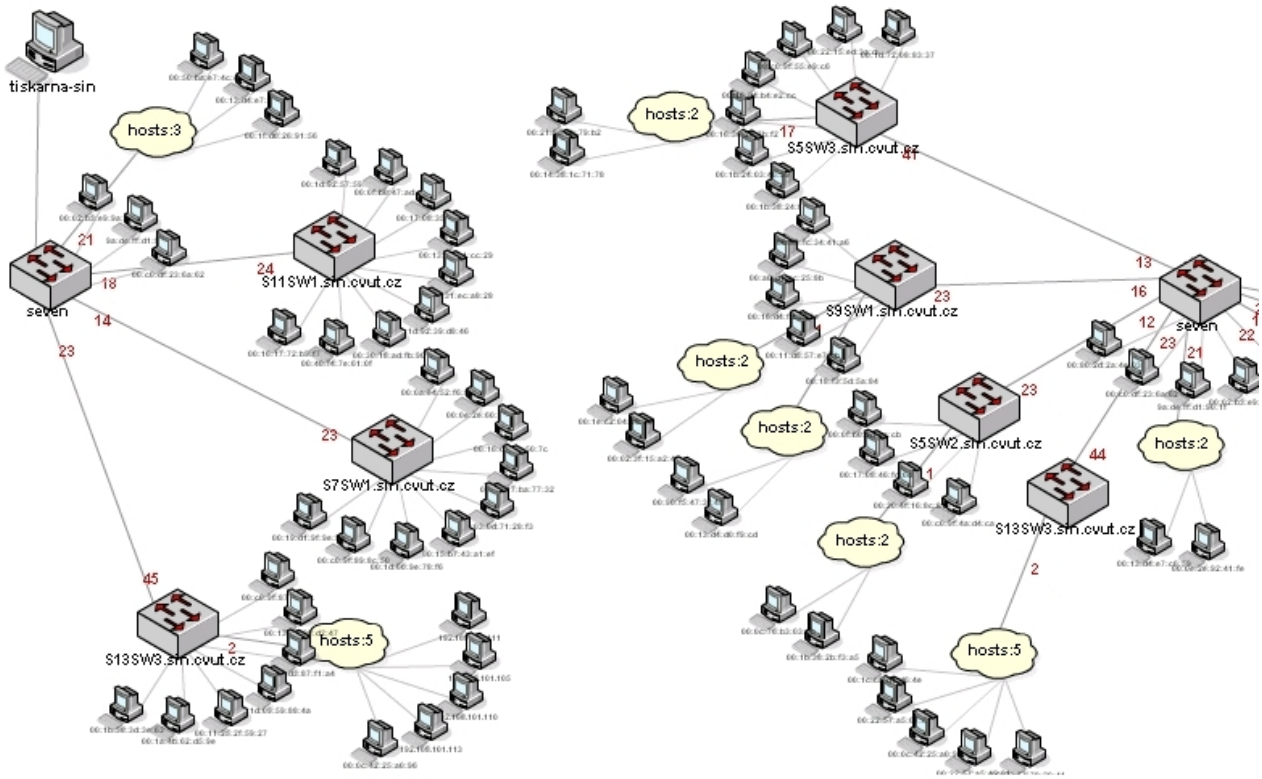
Základní částí aplikace je zobrazení topologie zkoumané sítě. O zobrazení topologie z příchozího souboru se stará vykreslování algoritmus, která je součástí zkompilovaného kódu Flashové animace. Kreslící algoritmu je zobrazen na obrázku 3.12.



Obrázek 3.12: Popis průběhu algoritmu, který vytvoří topologii.

V prvním kroku algoritmu se inicializují potřebné objekty a dynamické struktury , které se následně využijí při vykreslování. Příkladem může být třída XML, která byla popsána v kapitole 2.2.2.3. Tato třída se použije pro načtení XML dokumentu, který obsahuje data potřebná pro vizualizaci. Po dokončení načítání se spustí funkce, která daný XML soubor zpracuje a při tom naplní dynamické struktury, které se při vykreslování použijí. Příkladem může být asociativní pole NodeElement, které pro každé identifikační číslo vizualizovaného zařízení odkazuje na příslušnou část XML objektu, kde se nacházejí atributy daného zařízení .V dalším kroku se procházejí všechny XML elementy, které jsme označili v XML dokumentu(obrázek 3.8) jako „ELEMENT“. Při zpracování těchto objektů čteme jejich typ, pozici, jméno a rodiče. Na základě přečteného typu vybereme z grafické knihovny potřebný MovieClip. MovieClip je jeden z nejzákladnějších objektů Flashe, který se používá k uspořádání objektů ve scéně. Jméno vypíšeme do textového políčka tohoto MovieClipu a pokud je identifikační číslo předka rovno -1, pak daný MoveClip vykreslíme na udanou pozici a zaregistrujeme pro něj listenery. Tyto objekty budou reagovat na uživatelovy akce a budou řídí další činnost programu. V okamžiku, kdy máme všechny objekty typu „ELEMENT“ vykresleny, přejdeme do dalšího kroku algoritmu.

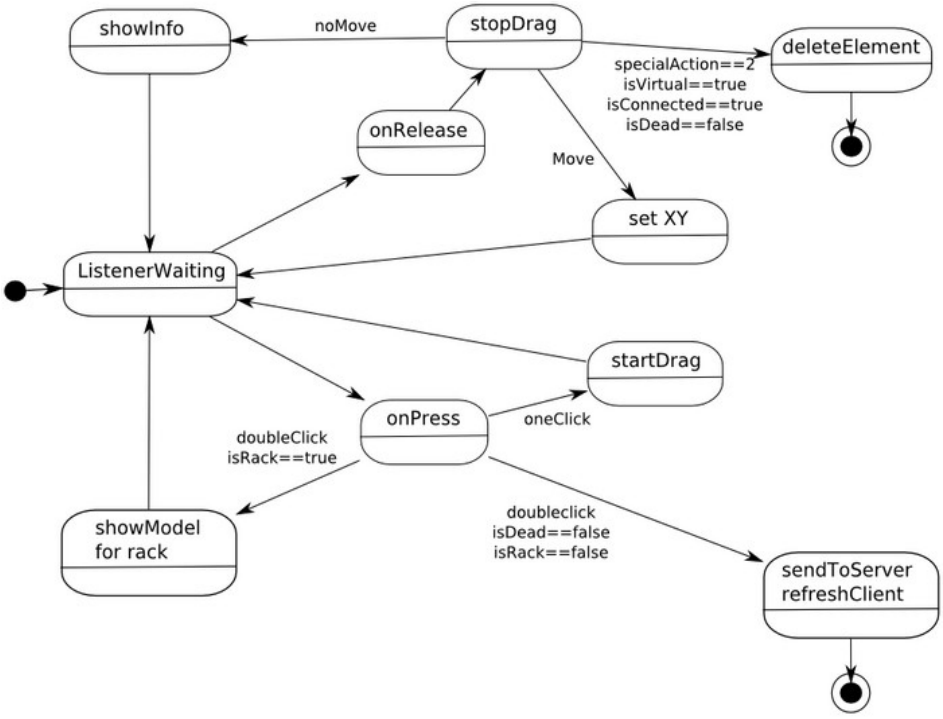
V tomto kroku si zjistíme těžiště vykreslených MovieClipů. Kdyby totiž uživatel mohl měnit ve výsledku pozice úplně všech prvků ve scéně, byla by výsledná práce s modelem velice chaotická a obtížná. Například pokud bychom chtěli přesunout obrázek switche, tak bychom museli přesunout třeba i 40 koncových stanic, jenž budou k němu zapojeny. Proto pozice objektů typu „HOST“ a objektů typu „HUB“ se odvozuje z objektů typu „ELEMENT“. A z tohoto důvodu potřebujeme zjistit i těžiště všech objektů typu „ELEMENT“. Při vykreslování koncových stanic totiž postupujeme podobně, jak je znázorněno na obrázku 3.14. Tj. že koncové stanice ve výsledku postupně obklopují příslušný aktivní prvek a nebo příslušný HUB. Ale abychom maximálně zpřehlednili výslednou scénu vykreslujeme koncové prvky co nejdál od míst předpokládaného největšího překřížení spojů a tedy co nejdál od těžiště. A pomocí těžiště si určíme počáteční úhel vykreslování koncových prvků, jak je znázorněno na obrázku 3.14.



Obrázek 3.13: Příklady vykreslení závislých prvků.

V dalším kroku algoritmu vykreslíme MovieClipy odvozené od objektů typu „HUB“. Objekty tohoto typu jsou také považovány za závislé, protože mohou spojovat více objektů typu „ELEMENT“. A proto je umístíme do těžiště objektů typu „ELEMENT“, které dané huby propojují. Po vykreslení všech těchto objektů vykreslíme i jejich spoje s elementy a přestoupíme do dalšího kroku algoritmu. V tomto kroku vykreslíme spoje mezi jednotlivými MoveClipy, které reprezentují objekty typu „ELEMENT“. A v posledním kroku vykreslíme objekty typu “HOST“ a jejich spoje s nadřazenými prvky. Toto vykreslení ale proběhne tak že se dané MovieClipy, které reprezentují hosty, vloží přímo do MovieClipu svého nadřazeného objektu. Např. pokud ke switchi je připojeno několik počítačů, pak při vizualizaci se MovieClipy počítačů vloží do poslední vrstvy MoveClipu nadřazeného prvku. Po skončení vykreslování je daný model hotový a uživatel může s daným modelem pracovat.

K tomu aby daný model topologie byl interaktivní, je zapotřebí odposlouchávání uživatelova chování. Chování aplikace definují listenery. Tyto listenery definují co se má stát, když daný uživatel stiskne tlačítko myši a když dané tlačítko uvolní. Jednoduchý stavový digram těchto reakcí na jednom MovieClipu je vyobrazen na obrázku 3.14. Tento diagram si popíšeme. Pokud uživatel stiskne tlačítko, pak se spustí funkce onPress. Tato funkce si zjistí kdy naposledy uživatel jednou klikl v rámci daného MovieClipu. Pokud klikl před více jak před 350ms, pak se daná reakce považuje za jednoduchý jeden klik a operací startDrag se daný MovieClip připne na uživatelovu myš. Pokud ale uživatel klikl dvakrát v rozmezí 350ms, pak je daný klik označen jako dvojklik. Při dvojkliku si funkce onPress zjistí typ zařízení, které reprezentuje její MoviClip. A je-li dané zařízení rack, pak se zobrazí jeho obsah. Ale pokud dané zařízení není rack a Flashová animace má nastaven isDead na false, pak se načte nová HTML stránka, která bude obsahovat zase vykreslený model tak i podrobný popis vybraného zařízení včetně jeho portů. Pokud je uživatel zároveň autorizovaný, pak se i provede poslání provedených změnách ve formě XML souboru zpět na server.



Obrázek 3.14: Funkce onPress a onRelease pro jednotlivé elementy.

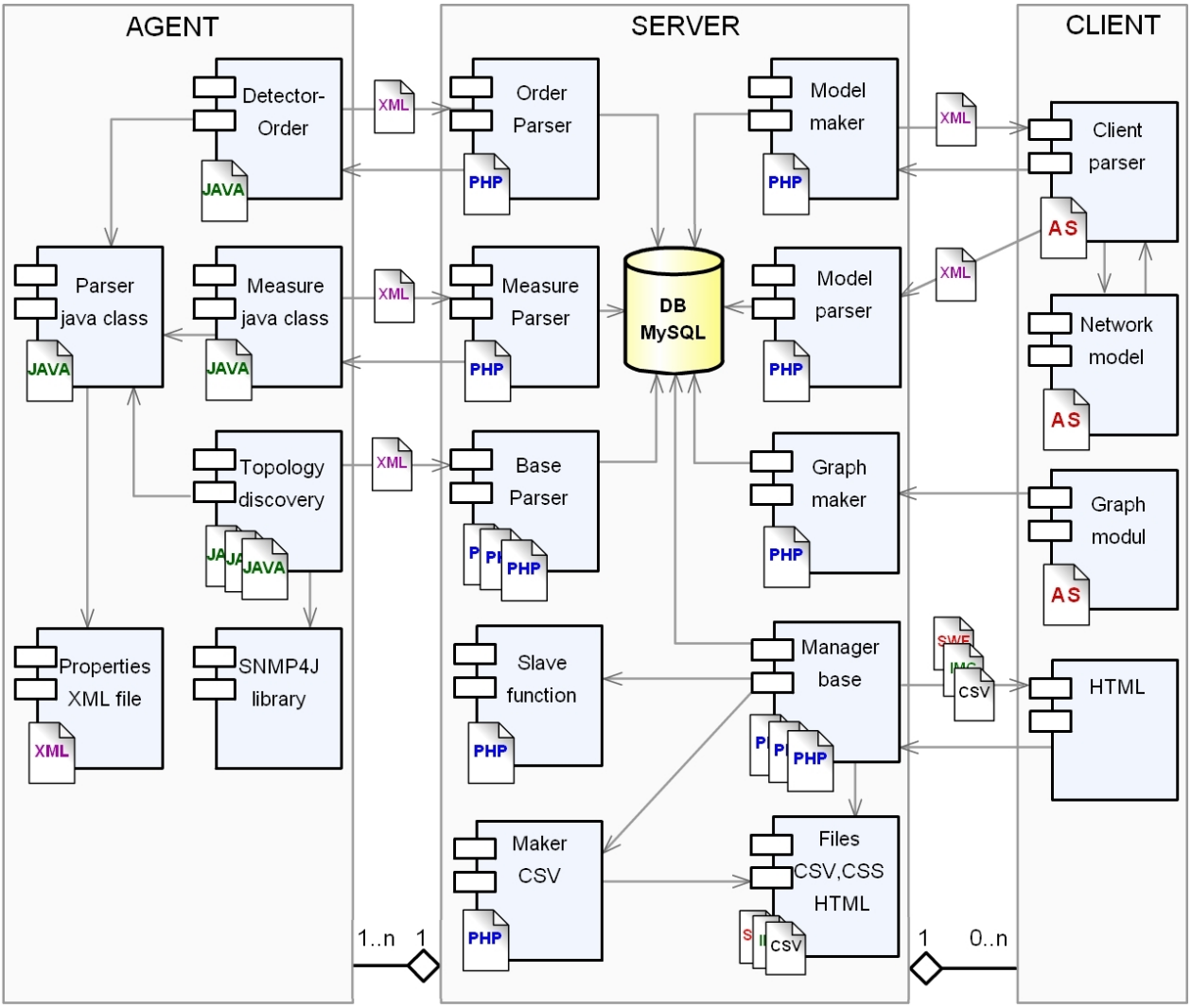
Další funkce, která se v souvislosti s reakcí na uživatelovo chování vyskytuje, je i onRelease. Tato funkce se zavolá při uvolnění tlačítka myši. Při uvolnění tlačítka myši se zavolá funkce stopDrag, která případně odejme přimknutý MovieClip z tlačítka myší. A na základě toho o jak velký kus se pohnul daný MovieClip se zvolí další průběh. Pokud se daný MovieClip téměř nepohnul, pak se pouze vyvolá informační menu o daném zařízení. Na druhou stranu pokud uživatel pohnul myší o více jak 10 pixelů, pak se změní pozice daného zařízení a celý model se překreslí, protože se změnilo jeho těžiště. V součinnosti se změnou pozice se nová pozice uloží do příslušné části XML dokumentu. Ve stejné části XML dokumentu se také změní také atribut „ch“ na 1. Další možnost, kde se onRelease použije je u mazání elementů. Součástí pracovní plochy je i menu, kde je možné najít mazací tlačítko. Pokud uživatel na toto tlačítko klikne, pak se nastaví specialaction na 2. A pokud posléze autorizovaný uživatel klikne na nějaký MovieClip, který reprezentuje XML objekt typu „ELEMENT“ a tento „ELEMENT“ je virtuální, pak je daný MovieClip odstraněn. A v příslušné části XML dokumentu se nastaví atribut „ch“ na -1. Jen pro připomenutí, atribut „ch“ je používán serverovým skriptem setmodel.php při zpracování příchozího XML souboru od klienta.

Další operace, kterou může uživatel provádět je vkládání virtuálních elementů. To se dělá tak, že autorizovaný uživatel klikne na požadovaný element. Tímto kliknutí se nastaví globální proměnná specialoperation na 0, specialoperationid na identifikační číslo typu vybraného zařízení. A pokud posléze autorizovaný uživatel klikne na pracovní plochu, pak se spustí událost onPress na příslušném MovieClipu, který reprezentuje danou plochu. Tato funkce si zjistí jestli je nastavená specialoperation na 0 a poté vytvoří nový MoviClip, který bude reprezentovat typ zařízení specialoperationid. Zároveň se vytvoří nový uzel v XML dokumentu, na který se bude dané zařízení odkazovat a jenž bude mít nastaven atribut „ch“ na 1. Okamžitě po takovémto vytvoření se daný XML dokument pošle zpět na serveru a skript setmodel.xml na jeho základu vytvoří nový záznam v tabulce.

4 Implementace

V následných odstavcích uvedeme jen ty nejdůležitější analýzy ze softwarové inženýrství, jelikož by kompletní analýza aplikace, mající více jak 6tis. řádků kódu v různých vývojových prostředí, byla značně nepřehledná, poměrně zbytečná a přesáhla počet stran doporučených pro popsání výsledného projektu. Cílem autora totiž není dokázat značnou složitost a pracnost řešení problému, ale vybrat takové informace a diagramy, které čtenářům umožní snadno pochopit vlastnosti, výhody a nevýhody výsledné implementace.

4.1 Model komponent

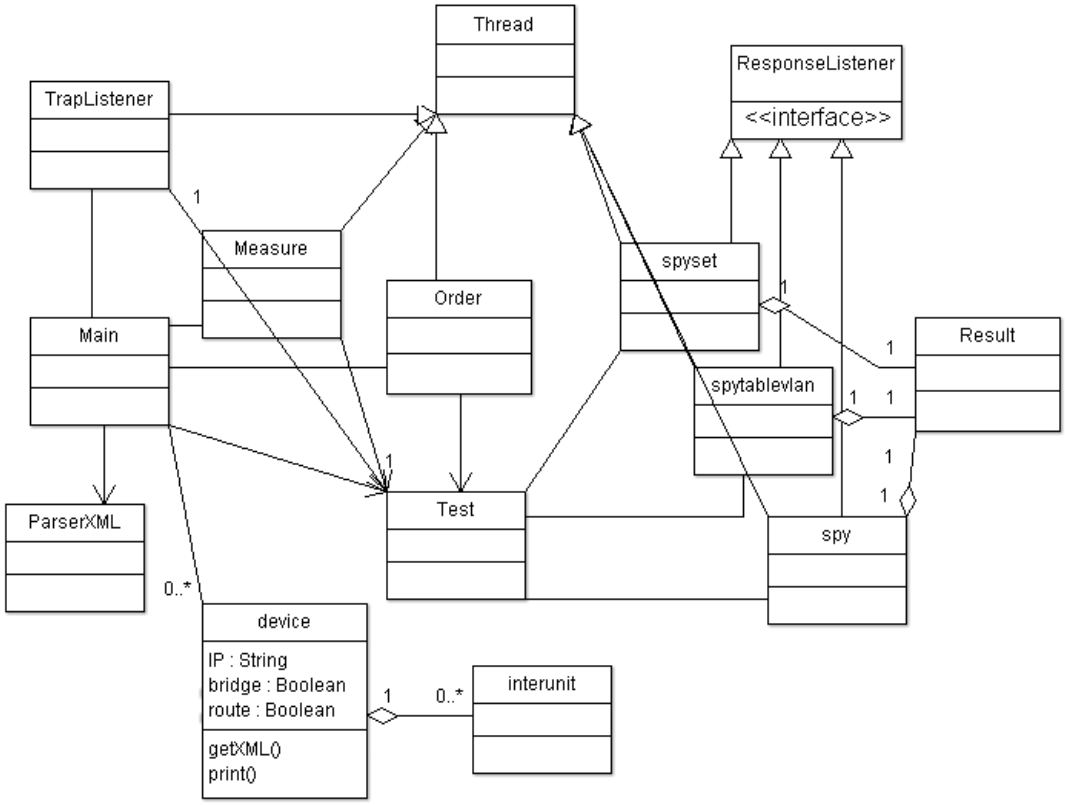


Obrázek 4.1: Zobrazení komponentní struktury aplikace

Na obrázku 4.1 je zobrazena struktura aplikace, kde je poměrně patrná modulární architektura systému, který obsahuje jeden server a větší množství agentů a klientů, kteří se mezi sebou dorozumívají za pomoci XML dokumentů. Taky je zde patrné využívání různých programových

prostředků. Agent je implementován v Javě, server pomocí PHP a klient pomocí ActionScriptu.

4.2 Struktura agenta



Obrázek 4.2: Zobrazení struktury agenta.

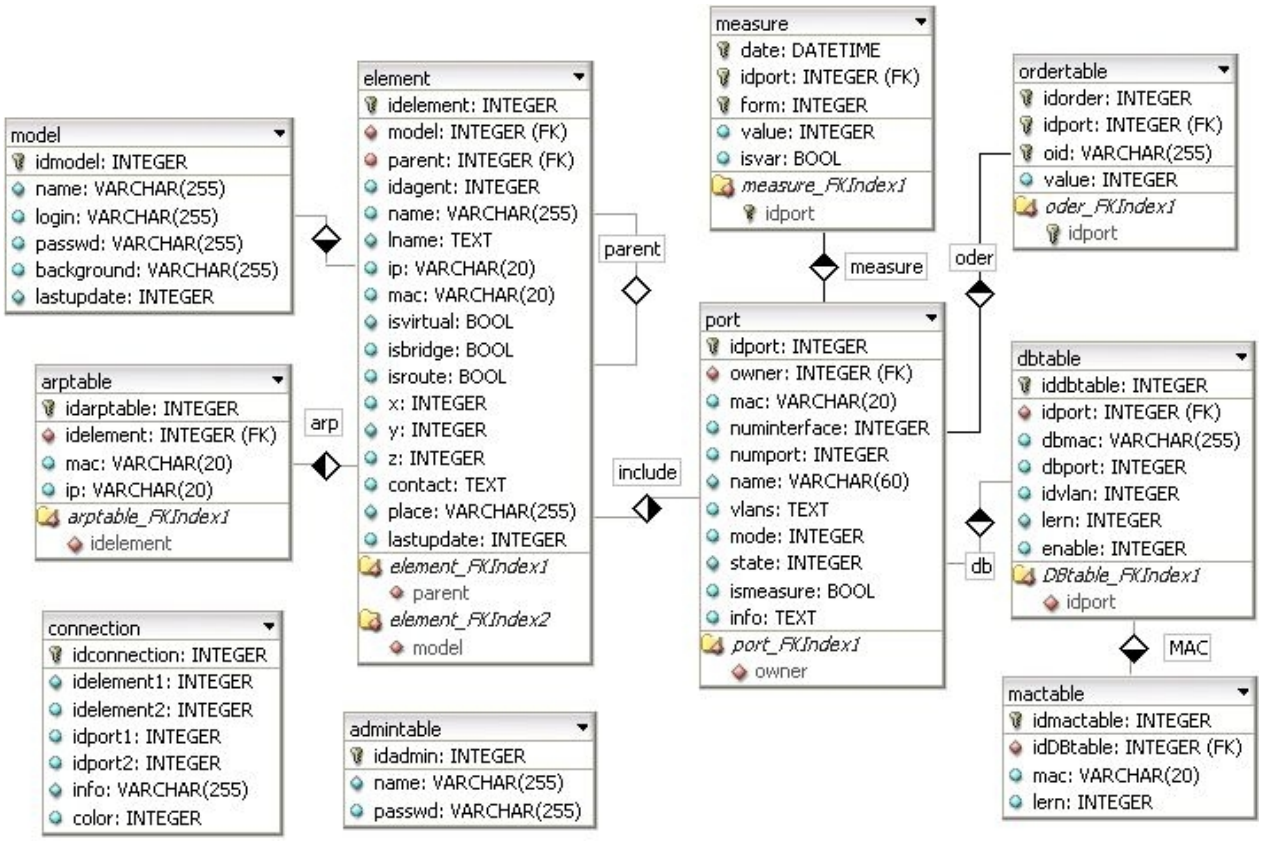
Na obrázku 4.2 je zobrazena struktura agenta, která obsahuje některé objekty, které si nyní popíšeme.

- **Main -** hlavní třída, spouští a řídí celého agenta, vyvolá příslušní vlákna a v cyklicky spouští detekční algoritmus.
- **ParseXML -** pareser XML dokumentů, který zpracuje základní konfigurační údaje z dokumentu config.xml. Tyto údaje předá třídě Main, která podle nich řídí činnost vláken.
- **Test -** pomocná třída, která obsahuje třídy a metody pro měření informací ze síťových zařízeních, která podporují SNMP. Pro velké množství těchto tříd a metod , je následně popsána pouze třída spy a spyTableVlan.
- **Order** -třída pro zpracování a provedení příkazů ze serveru.
- **Measure** - třída pro měření zatížení portů jednotlivých aktivních zařízeních.
- **Device** - pomocná třída, která shromažďuje naměřená data z detekčního algoritmu. Každá instance třídy device reprezentuje jedno zařízení, které podporuje SNMP.
- **Interunit** - pomocná třída, která je používána ve třídě device. Interunit reprezentuje

jeden port a jeho základní vlastnosti.

- **TrapListener** - třída pro odposlouchávání trapů.
- **Spy** - třída , která se použije pro načtení objektů z MIB stromu pro všechna SNMP zařízení v přiděleném rozsahu IP adres.
- **SpyTableVlan** – třída určená pro načtení tabulky z MIB stromu, u které se při adresaci používá id VLANy
- **Result** – pomocná třída, která je používána pro uložení mezivýsledků tříd spy a spyTableVlan.

4.3 Struktura serveru



Obrázek 4.3: Zobrazení tabulek používané serverem.

Na obrázku číslo 4.3 je zobrazena tabulková struktura databáze, která spolu s PHP skripty tvoří stěžejní část serveru. V následujícím odstavci si jednoduše popíšeme tabulky a důležité skripty, které v serveru použijí.

Použité tabulky:

- **Model –** tabulka reprezentující jednotlivé modely. Obsahuje autorizační údaje, obrázek pozadí, čas poslední změny v modelu.
- **Element** – tabulka , která reprezentuje virtuální elementy a nebo zařízení, která

podporující SNMP. Element v sobě spojujte vizualizační informace s technickým popisem prvku.

- **Arptable** – tabulka obsahující informace o ARP tabulce.
- **Admintable** – obsahuje hesla a uživatelská jména administrátorů aplikace. Při instalaci se vytvoří předdefinovaný účet, kde logine je „admin“ a heslo je „netvis“.
- **Connection –** netypická tabulka, která reprezentuje spoje pouze mezi virtuálními elementy. Další její zvláštnost je i to že není jednoznačně dáno jestli spojuje porty a nebo elementy, a proto ve schématu nejsou vyjádřeny žádné vazby a různé typy spojení se realizují až PHP skripty.
- **Port** – tabulka, která reprezentuje porty a jejich základní vlastnosti.
- **Measure** – tabulka kam se ukládají naměřené veličiny.
- **Ordertable** – tabulka, která udržuje příkazy klienta, které jsou určeny agentovi. V okamžiku, kdy je příkaz převzat agentem , tak se příkaz odstraní tabulky.
- **Dbtable** – tabulka obsahující základní informace od STP ve zkoumané síti.
- **Mactable** – tabulka, které obsahuje informace o CAM tabulkách pro daný port a danou VLANu.

Použité skripty:

- **manager.php –** základní skript, který obsahuje definici objektu manager. Manger obsahuje základní automaty, které zpracovávají klientovy požadavky.
- **special.php –** skript, který obsahuje množinu základních funkci, které manager používá při vytváření HTML stránek.
- **getmodel.xml** – skript, který vrátí XML soubor popsaný na obrázku 3.8.
- **BaseParser.php** – třída, která je určena k parserování XML dokumentů od agenta. Tyto dokumenty jsou popsány na obrázku 3.4 a obsahují data ze zkoumané sítě.
- **setagent.php –** pomocná funkce, která vytvoří instanci třídy BaseParser.
- **MeasureParser.php –** třída, která je určena pro podporu měřícího vlákna agenta.
- **measure.php –** pomocná funkce, která vytvoří instanci třídy MeasureParser.
- **OrderParser –** třída, která je určena pro podporu vlákna zpracovávajícího příkazy.
- **order.php –** pomocná funkce, která vytvoří instanci třídy OrderParser.
- **about.php** – skript pro zpracování tabulkových hodnot, které se pošlou serveru při úpravě portů virtuálního klienta.
- **fcebase.php** – obsahuje základní funkce, které jsou používány dalšími skripty.
- **setmodel.php** – skript, který převezme XML soubor od klienta a případné upraví databázi.
- **showgraph.php** – funkce, která vytvoří Flash objekt v HTML a naplní FlashVars. Webovému klientovi se pak zobrazí interaktivní graf, který bude obsahovat takto získané hodnoty.

4.4 Struktura klienta

Jelikož podrobná struktura klienta byla již představena v analýze a uživatelská část klienta je generovaná a obsluhován skripty na straně serveru, pak v následujících odstavcích si popíšeme jen nejzákladnější funkce a proměnné, které tvoří klientskou část aplikace.

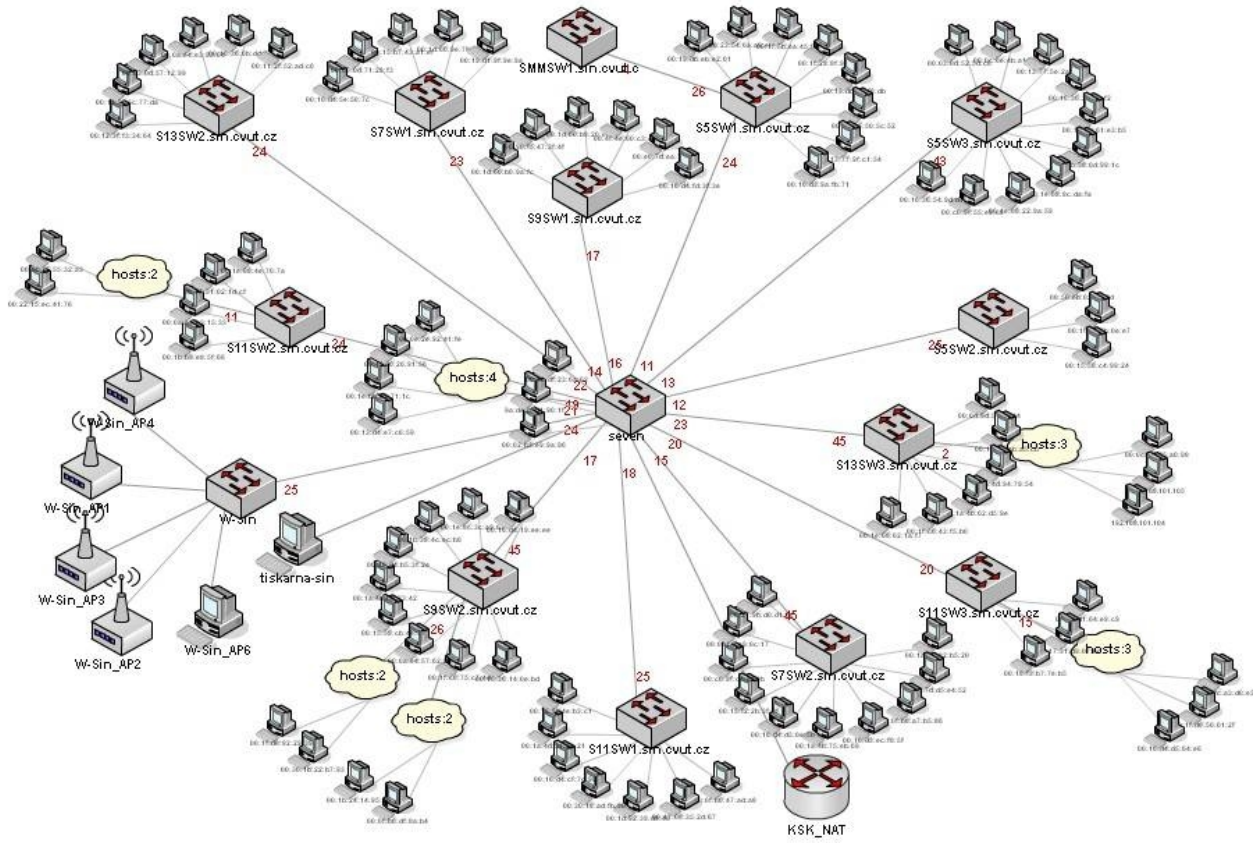
Použité funkce a proměnné

- **_root.BASE** – instance objektu XML, která obsahuje data získaná ze serverové funkce getmodel.xml. Do této instance jsou postupně vkládány změny, které uživatel provede.
- **firstParse –** funkce, která zpracuje objekt BASE a výsledky tohoto zpracování uloží do pomocných proměnných a dynamických objektů, které se dále využijí v programu.
- **showModel** – funkce, která spustí celý proces zobrazený na obrázku 3.12.
- **createElement** – třída, která vytvoří MovieClip, který bude reprezentovat daný element.
- **showInformation –** funkce, která vyvolá informační okno daného elementu.
- **getLibStyle –** funkce, která z identifikačního čísla stylu navrátí odkaz na konkrétní objekt v knihovně grafických prvků.
- **showLine2 –** funkce, jenž vykreslí spojnici mezi dvěma MoviClipy.
- **showLinesAll2 –** funkce, která vykreslí všechny závislé prvky a jejich spoje.
- **returnModel -** funkce, která se volá při uzavření racku a návratu na vyšíúroveň.
- **ShowCreateWin –** funkce, která vyvolá okno, kde je vytvořeno základní editorské rozhraní pro tvorbu a mazání elementů.
- **deleteElement -** smaže MovieClip a nastaví příslušnou atribut „ch“ na -1 v _root.BASE.
- **makeNewElement -**vytvoří nový MovieClip a do _root.BASE vloží nový XML node, který bude reprezentovat vytvořený objekt a jeho atribut „ch“ bude mít hodnotu 1.

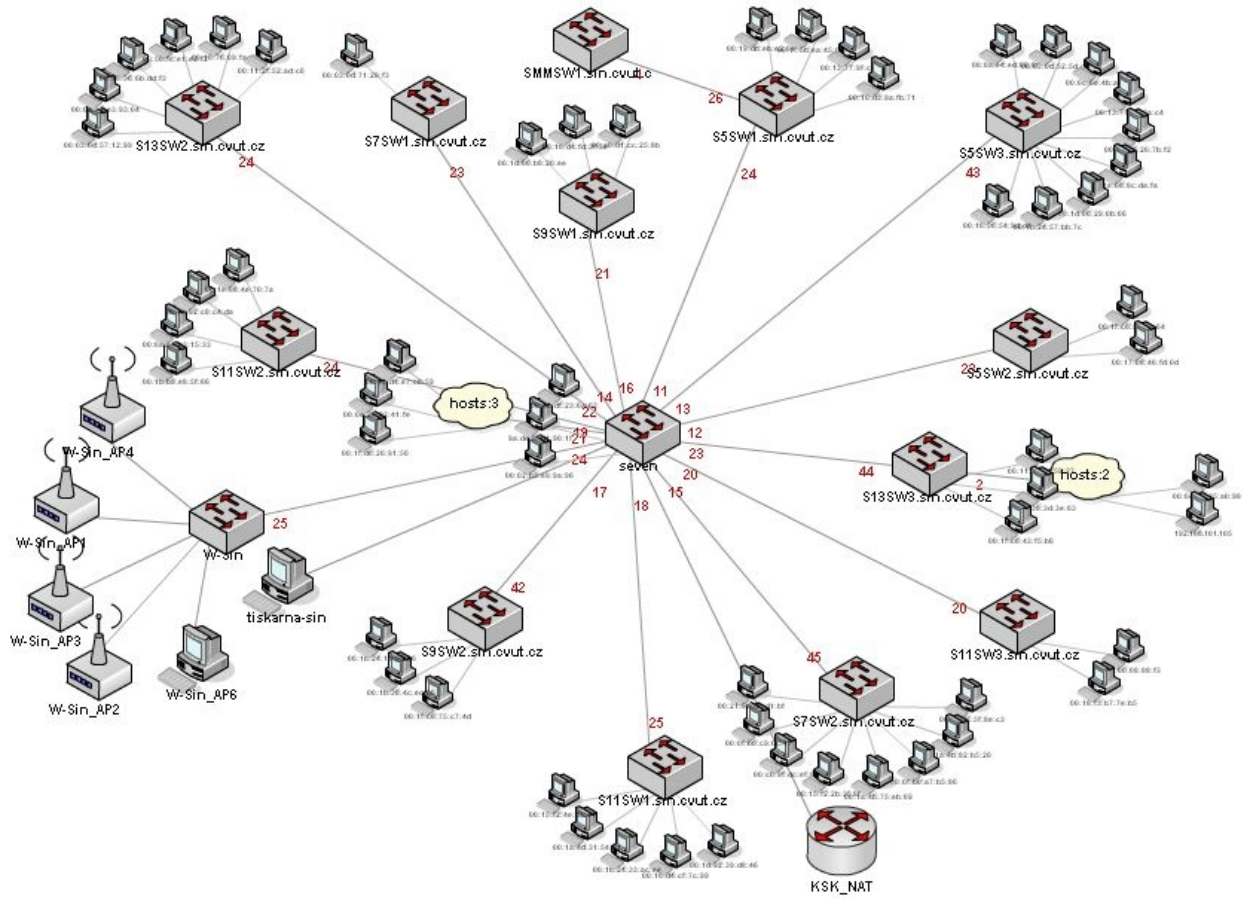
5 Testování

Při testování bylo postupováno podle uživatelské příručky. Aby bylo možné otestovat funkčnost výsledné aplikace bylo nejprve nutné nainstalovat jednotlivé komponenty celého systému. Prvořadě bylo nutné nastavit serverovou část aplikace a vytvořit modely, do kterých se budou ukládat data. To se provedlo tak, že se na webový server, který podporuje PHP a MySQL, nahrály soubory na přiloženém CD a do souboru fcebase.php byli nastaveny přístupové údaje k databázi. Následně se do databáze nahrál inicializační skript, který vytvořil potřebné tabulky a také předdefinovaný model a jeho uživatelské jméno a heslo. Následně bylo nutné nastavit agenta a zajistit mu přístup na síť. Agentovi byli nastaveny přístupové údaje, adresa webového serveru, interval mezi měřením a maximální počet spuštěných detekčních vláken maxthread=30. Veškeré testování bylo prováděno v reálném provozu na síti, kde se počet hostů pohybuje v intervalu 30-300. Na následujícím obrázcích jsou zobrazeny výsledky měření, které probíhalo na dané síti nepřetržitě a bezproblémově po dobu 48 hodin.

V prvním kroku byla testována detekce a vykreslování topologie zkoumané sítě.

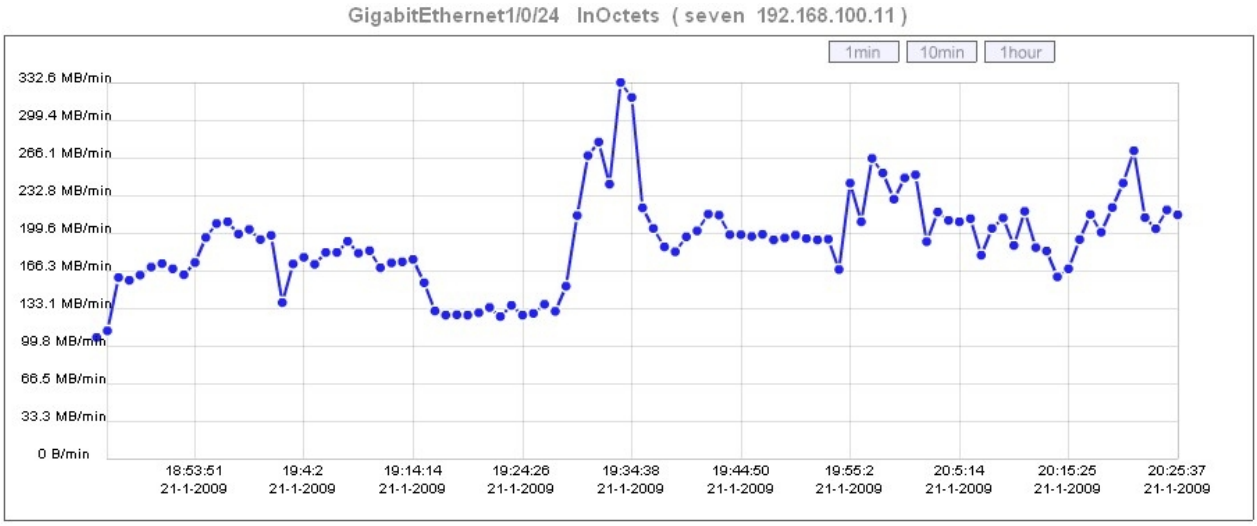


Obrázek 5.1: Zobrazení zkoumané sítě v sobotu 17:00.

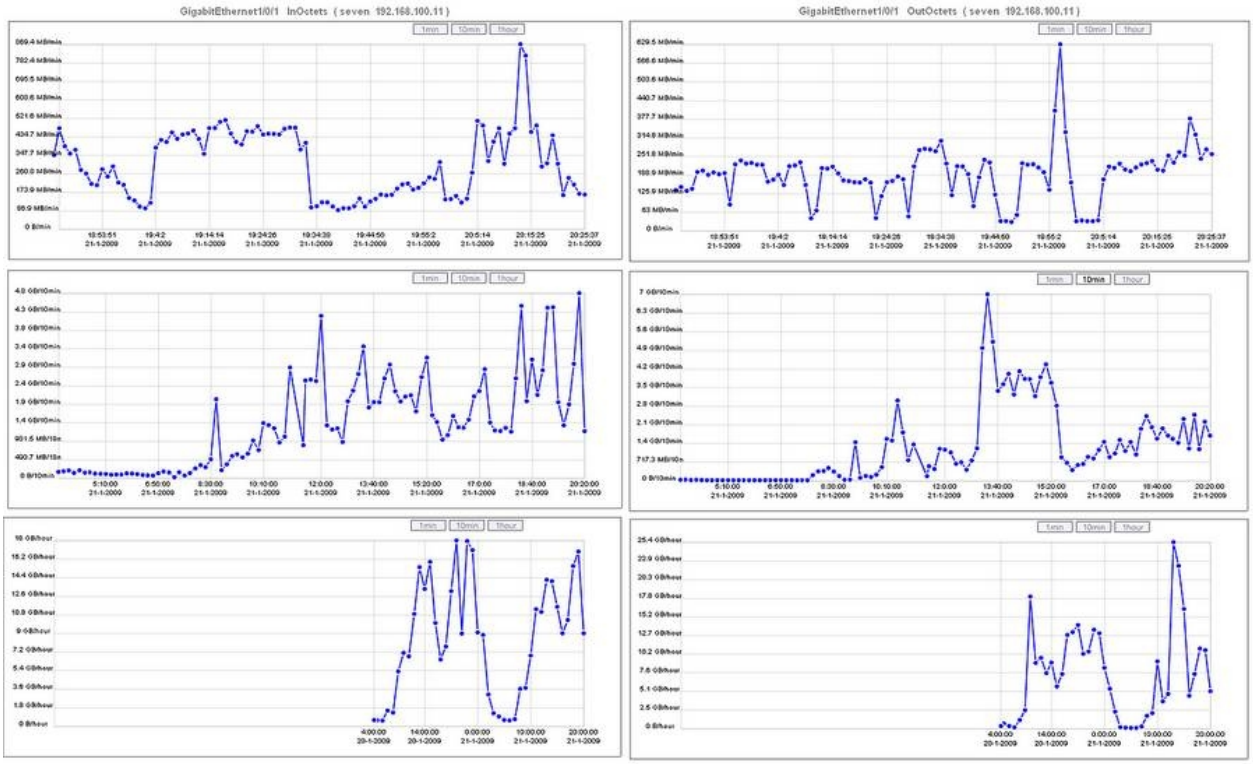


Obrázek 5.2: Zobrazení zkoumané sítě v sobotu 24:00.

V dalším testování bylo měřeno zatížení vybraných portů na nalezených aktivních zařízeních. K vizualizaci tohoto měření byli požit interaktivní a dynamický modul pro vizualizaci grafů, který byl implementován ve výsledné aplikaci.

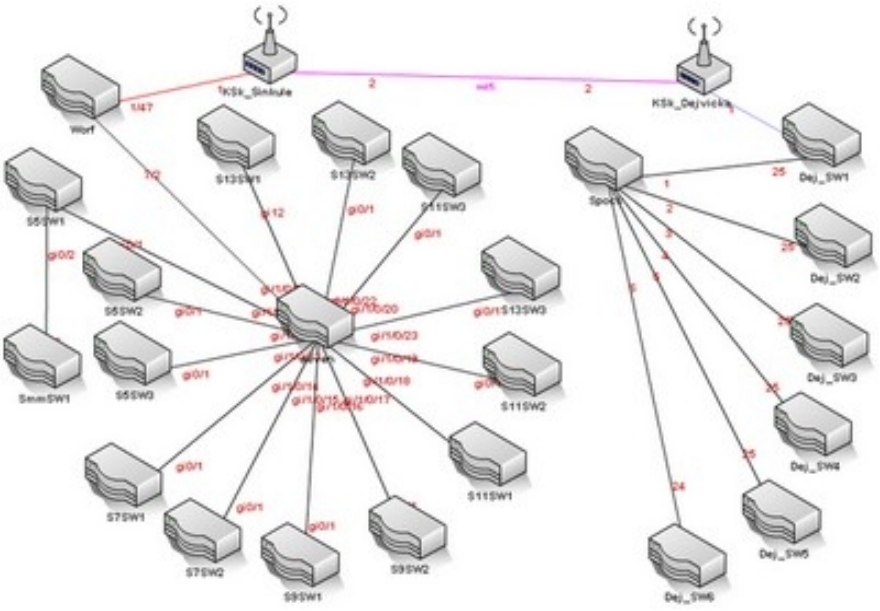


Obrázek 5.3: Zobrazení zatížení portu GigabitEthernet1/0/1 na 192.168.100.11.



Obrázek 5.4: Dlouhodobé měření zatížení portu GigabitEthernet1/0/1 na 192.168.100.11.

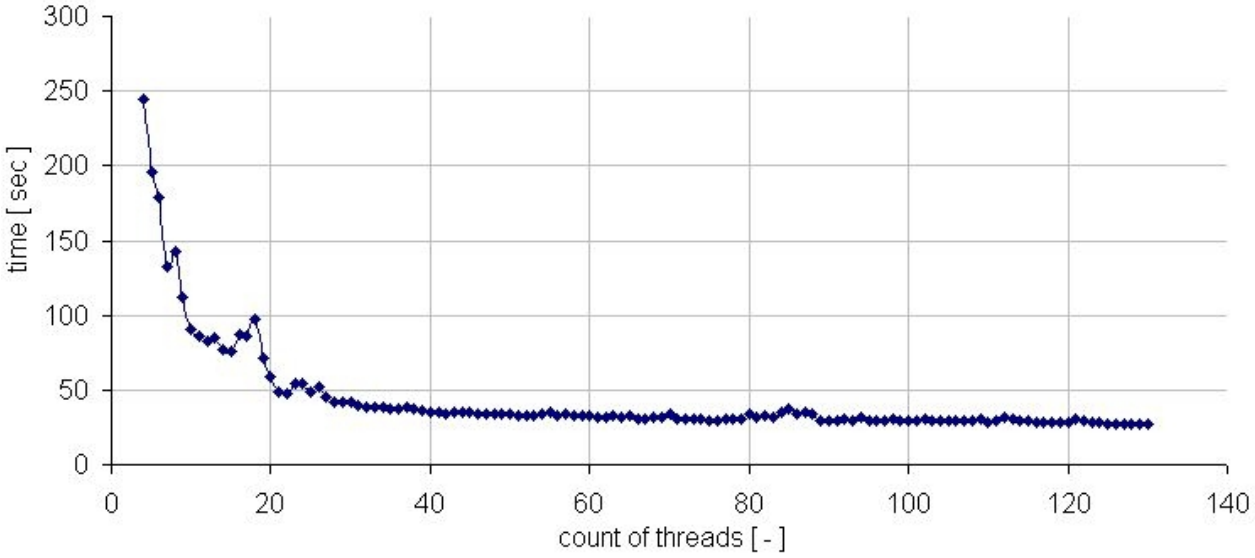
Obrázek 5.4 zobrazuje zatížení portu GigabitEthernet1/0/1 na zařízení s vybranou IP adresou 192.168.100.11. V levém sloupci je měřeno vstupní zatížení a v pravém sloupci výstupní zatížení. První řádek obsahuje grafy, které zobrazují množství přenesených dat po jedné minutě. Druhý řádek obsahuje grafy, jenž ukazují množství přenášených dat po desetiminutových intervalech. A konečně na posledním řádku jsou grafy, jenž zobrazují množství přenesených v hodinových intervalech.



Obrázek 5.5: Virtuální topologie zkonstruovaná přes uživatelské rozhraní.

V rámci hlavního testu byli ještě provedeny jednoduché testy funkčnosti uživatelského rozhraní. Testovali se základní operace, které daná aplikace nabízí pro autorizované i neautorizované uživatele. Testovalo se vytváření, upravování a mazání jednotlivých modelů. Zároveň se zkoušela úprava obrázkového zátiší modelů, úprava vizualizované topologie jako například změna umístění jednotlivých zařízeních, jejich vnořování do racků, přidávání, mazání a upravování virtuálních elementů podobně jak je zobrazeno na obrázku 5.4. U agenta se testovala změna atributu idmodel, tak aby se neustále model nepřepisoval aktuálními daty, ale uchoval se pro další použití.

Při výkonnostním testu bylo provedeno měření, kde se měnil parametr maxthread v config.xml. A měřila se časová náročnost detekce prvků, pokud budeme měnit maximální počet detekčních vláken. Na obrázku 5.5 je dobře patrná klesající průběh takové závislosti.



Obrázek 5.6: Náročnost detekčního algoritmu na parametru maxthread.
(Z důvodu názornosti jsou jednotlivé hodnoty propojené.)

V rámci testování byl úspěšně provedena detekce topologie zkoumané sítě. Na vizualizovaných obrázcích byli patrné huby, které k příslušným portům aktivních zařízeních připojovali více jak jedno koncové zařízení. Zároveň se úspěšně zobrazili AP, které jsou součástí dané sítě. Při měření zatížení portů se nevyskytli výraznější problémy a zobrazované grafy dobře ukazují poklesy zatížení v ranních hodinách a vzestupy zatížení v odpoledních hodinách. Vyexportované CSV soubory obsahovali hodnoty, které odpovídali skutečnosti a bezproblémově se zobrazovali v externích programech jako Excel nebo v OpenOffice. Při testování funkcionality se taktéž nevyskytli výraznější problémy. Součástí testování uživatelského rozhraní byla i tvorba jednoduchého modelu, který obsahoval virtuální elementy. V tomto modelu bylo možné jednotlivé prvky vytvářet, mazat, měnit jejich popisky, navzájem je propojovat, přidávat a ubírat jim porty. Zároveň bylo možné vnořovat jednotlivé prvky do racků. Při posledních testech se testovalo chování Flashové animace. Pokud například byl odinstalován FlashPlayer tak se správně zobrazilo upozornění na jeho neexistenci. A pokud byl model vyexportován do jiných webových stránek, tak se model zobrazil tak jak měl včetně dodatečných bezpečnostních omezení.

6 Závěr

Výsledkem celého projektu je zajímavá a poměrně unikátní aplikace, která je využitelná při vizualizaci informací o zkoumané síti. Během analýzy a implementace se muselo řešit velké množství problémů, které byli popsány v rámci textu této práce. Celá aplikace je rozdělena do tří samostatných bloků, které mezi sebou komunikují pomocí přenosu XML dokumentů a z toho důvodu je také aplikace velmi modulární. Tato modularita například umožňuje rozdistribuovat měření zkoumané sítě mezi větší množství agentů a nebo nahradit agenta i klienta jinou aplikací, která ale zachovává formát XML zpráv. Další výhodou výsledné aplikace je to, že agent je implementován v Javě bez použití nativních tříd, a proto je i daná aplikace platformě nezávislá.

Zároveň byla aplikace navržena tak, aby podporovala větší množství modelů, které mohou být nezávisle a paralelně plněny daty. Takováto technologie modelů umožňuje udržovat naměřené hodnoty i od agentů, kteří již se serverem nekomunikují. Možností, kde se výsledná technologie může využít, je velké množství. Například pokud administrátor chce výrazně změnit počítačovou síť, ale při tom si chce udržet původní informace, pak jednoduše poupraví konfigurační soubor agenta a tím zabrání přepsání původních informací.

Další zajímavou možností použití je i exportování výsledné topologie sítě do externích webových stránek. Například pokud administrátor píše odbornou zprávu o nějaké síti, pak jako ukázku přiloží interaktivní a neustále aktualizovaný model této sítě.

Podstatná je také podpora tvorby virtuálních elementů. Resp. administrátor může do modelů vkládat i prvky, které nejsou odvozeny z reálné sítě a které je zároveň možné mezi sebou propojovat i vnořovat. Takováto technologie se uplatní například v situacích kdy nechceme zatěžovat sít protokolem SNMP, ale zároveň chceme někde zaznamenat topologii dané sítě spolu s informacemi o jednotlivých portech. Pro tento případ je možné použít výslednou aplikaci. Např. administrátor vloží jednotlivé virtuální elementy na pracovní plochu, nadefinuje porty a upraví u nich spojení s dalšími elementy. Ve výsledku máme topologii sítě, kde si můžeme vygenerovat CSV soubory, které reprezentují dané elementy.

Budoucí vývoj této aplikace je možné spatřovat ve využití knihoven jako papervision3D, Away3D atp.. Tj. v rozšíření vizualizované topologie o další dimenzi a to spolu s využitím grafických objektů typu collada. Následný vývoj je vhodné spatřovat také ve využití vyšších verzí SNMP včetně RMON. Jelikož je projekt silně modulární je možné jednotlivé moduly nahrazovat novějšími verzemi.

V současné době je aplikace využívána například Katedrou telekomunikací na ČVUT při vizualizaci páteřních sítí a sítí IP telefonie. Zároveň se aplikace využívá při vkládání výsledných vizualizaci do odborných prezentací a publikací v HTML, kde se výhodně používá toho, že taková vizualizace není statická a neměnná jako u jednoduchých obrázků, ale interaktivní, dynamická, vektorová, rozměrově přizpůsobitelná, ale hlavně aktuálně odpovídající zkutečnému stavu zkoumané sítě. Aplikace například nalezla využití i na Sinkuleho koleji, kde správcům sítě zobrazuje aktuálně připojené počítače v lokální síti. Administrátor pak s pomoci aplikace může zjistit, na kterém aktivním prvku je uživatel připojen, jak jeho počítač přesně zatěžuje zkoumanou síť a případně může zakázat port, na kterém je daný uživatel připojen a to vše a mnoho další přes jednoduché uživatelské rozhraní.

7 Literatura

[1] Stallings William (1996), SNMP, SNMPv2 and RMON: Practical Network Management, Second Edition, Addison-Wesley

[2] Wikipedia, Simple Network Management Protocol, [Internet] [1.1.2009]
http://en.wikipedia.org/wiki/Simple_Network_Management_Protocol

[3] Wikipedia, Internet history, [Internet] [1.1.2009]
http://en.wikipedia.org/wiki/Internet#History

[4] IETF, List of RFC, [Internet] [1.1.2009]
http://www.ietf.org/rfc.html

[5] Wikipedia, Internet Engineering Task Force, [Internet] [1.1.2009]
http://en.wikipedia.org/wiki/IETF

[6] Harris S.(2001), The Tao of IETF- A Novice's Guide to the Internet Engineering Task Force
http://www.ietf.org/rfc/rfc3160.txt?number=3160

[7] Luboš Klaška(2000), Správa počítačových sítí: Model Manager-Agent, [Internet] [1.1.2009]
http://www.svetsiti.cz/view.asp?rubrika=Tutorialy&clanekID=30

[8] CISCO, Overview of Basic SNMP Building Blocks, [Internet] [1.1.2009]
http://www.cisco.com/en/US/docs/ios/internetwrk_solutions_guides/splob/guides/dial/dial_nms/snmpover.pdf

[9] Luboš Klaška(2000), Správa počítačových sítí: SNMP objekty a MIB, [Internet] [1.1.2009
http://www.svetsiti.cz/view.asp?rubrika=Tutorialy&clanekID=31

[10] Luboš Klaška(2000), Správa počítačových sítí: Formát SNMP zpráv, [Internet] [1.1.2009]
http://www.svetsiti.cz/view.asp?rubrika=Tutorialy&clanekID=32

[11] Prof. Won-Ki Hong(2006), NMS : RMON & RMON2, [Internet] [1.1.2009]
http://dpnm.postech.ac.kr/cs637

[12] Luboš Klaška(2000), Správa poč. sítí: Další vývoj protokolu SNMP , [Internet] [1.1.2009]
http://www.svetsiti.cz/view.asp?rubrika=Tutorialy&clanekID=33

[13] Blumenthal, U.; Hien, N.; and Wijnen(1999), B. SNMPv3 Handbook. Addison-Wesley.

[14] Wikipedia, Adobe Flash, [Internet] [1.1.2009]
http://en.wikipedia.org/wiki/Adobe_Flash

[15] Wikipedia, Rich Internet application, [Internet] [1.1.2009]
http://en.wikipedia.org/wiki/Rich_Internet_application

[16] Wikipedia, Silver Light, [Internet] [1.1.2009]
http://en.wikipedia.org/wiki/SilverLight

[17] Adobe Systems, Flash content reaches 99.0% of Internet viewers, [Internet] [1.1.2009]
http://www.adobe.com/products/player_census/flashplayer/version_penetration.html

[18] Jan Rylich (2007), Jaká je budoucnost interaktivních aplikací?, [Internet] [1.1.2009]
http://www.lupa.cz/clanky/jaka-je-budoucnost-interaktivnich-aplikaci

[19] Adobe Systems, Worldwide Ubiquity of Adobe FP by Version, [Internet] [1.1.2009]
http://www.adobe.com/products/player_census/flashplayer

[20] Wikipedia, FutureSplash Animator, [Internet] [1.1.2009]
http://en.wikipedia.org/wiki/FutureSplash_Animator

[21] Wikipedia, Actionscript, [Internet] [1.1.2009]
http://en.wikipedia.org/wiki/Actionscript

[22] Wikipedia, TrueMotion VP6, [Internet] [1.1.2009]
http://en.wikipedia.org/wiki/VP6

[23] Wikipedia, Just in time compilation, [Internet] [1.1.2009]
http://en.wikipedia.org/wiki/Just-in-time_compilation

[24] Richard Krejčí(2006), Adobe uvedla Flash Player 9 a Flex 2, [Internet] [1.1.2009]
http://www.grafika.cz/art/webdesign/flashflex.html

[25] Gary Grossman (2006), AS 3.0 and AVM2: Performance Tuning, [Internet] [1.1.2009]
http://www.onflex.org/ACDS/AS3TuningInsideAVM2JIT.pdf

[26] Adobe Systems(2008), Flash Player 10, [Internet] [1.1.2009]
http://labs.adobe.com/technologies/flashplayer10

[27] Nate Weiss(2004), Flash MX 2004 pro vývojáře webových aplikací, Zoner Press

[28] www.dgx.cz (2007), Jak spravně vložit flash do stránky, [Internet] [1.1.2009]
http://latrine.dgx.cz/jak-spravne-vlozit-flash-do-stranky

[29] www.dgx.cz (2007), Jak obejít aktivování pluginu v IE, [Internet] [1.1.2009]
http://latrine.dgx.cz/jak-obejit-aktivovani-pluginu-v-ie

[30] Microsoft support(2006), IE ACA KB942615, [Internet] [1.1.2009]
http://support.microsoft.com/kb/945007

[31] Adobe Systems(2007), Preparing your website to handle the Microsoft changes to Internet Explorer, [Internet] [1.1.2009]
http://www.adobe.com/devnet/activecontent/articles/devletter.html

[32] Šimek Pavel (2003),Rich Internet Applications a Flash Remoting 1., [Internet] [1.1.2009]
http://interval.cz/clanky/rich-internet-applications-a-flash-remoting-1/

[33] CACTI, Official manual,[Internet][1.1.2009]
http://www.cacti.net/

[34] Wikipedia, Nagios, [Internet] [1.1.2009]
http://en.wikipedia.org/wiki/Nagios

[35] nedi.ch, NeDi, [Internet] [1.1.2009]
http://www.nedi.ch

[36] Wikipedia,Cisco Discovery Protocol , [Internet] [1.1.2009]
http://en.wikipedia.org/wiki/Cisco_Discovery_Protocol

[37] snmplink.org, Developer SNMP Software [Internet][1.1.2009]
http://www.snmplink.org/snmpsoftware/fordeveloper/

[38] Frank Fock, Jochen Katz, SNMP4J, [Internet][1.1.2009]
http://www.snmp4j.org

[39] E.Decker,P.Langille,1993,Definitions of Managed Objects for Bridges,[Internet][1.1.2009]
http://tools.ietf.org/html/rfc1493

[40] Wikipedia, Spanning Tree Protocol , [Internet] [1.1.2009]
http://en.wikipedia.org/wiki/Spanning_tree_protocol

[41] Netcraft(2008), Web Server Survey , [Internet] [1.1.2009]
http://survey.netcraft.com/Reports/200809/

[42] Wikipedia, Link Layer Discovery Protocol , [Internet] [1.1.2009]
http://en.wikipedia.org/wiki/Link_Layer_Discovery_Protocol

[43] Yigal Bejerano, Yuri Breitbart, Minos Garofalakis, Rajeev Rastogi, 2003,
Physical Topology Discovery for Large Multi-Subnet Network, [Internet] [1.1.2009]
http://www.ieee-infocom.org/2003/papers/09_02.PDF

[44] Myung-Hee Son, Bheom-Soon Joo, Byung-Chul Kim, and Jae-Yong Lee, "Physical Topology Discovery for Metro Ethernet Networks," ETRI Journal, vol.27, no.4, Aug. 2005, pp.355-366

8 PŘÍLOHA: Instalační a uživatelská příručka

8.1 Nastavení severu

- Pro nainstalování serverové části aplikace je nutné nahrát obsah adresáře „server“ na webový server, který podporuje PHP a MySQL. Adresář „server“ je uložen na přiloženém CD.
- Dále je nutné vytvořit požadované tabulky v MySQL databázi. To se provede tak že v adresáři „server“ nalezneme soubor database.sql a tento soubor nahrajeme do MySQL databáze.
- V dalším kroku vložíme autorizační údaje pro MySQL do skriptu fcebase.php. A to tak že na pozici address vložíme adresu databázového serveru, na pozici login vložíme uživatelské jméno, na pozici password vložíme heslo a na pozici namedatabase vložíme označení databáze v rámci databázového serveru.
- V dalším kroku nastavíme samotný název domény do konstanty $cserver ve special.php, proto aby fungoval systém pro exportování modelů do externích webových aplikací.
- A v dalším kroku nastavíme přístupová práva do adresářů „background“ a „export“ na 777 a povolíme spouštění PHP skriptů z XML souborů !
- V posledním kroku změníme předdefinované administrátorské heslo a jméno v tabulce admin.

8.2 Nastavení agenta

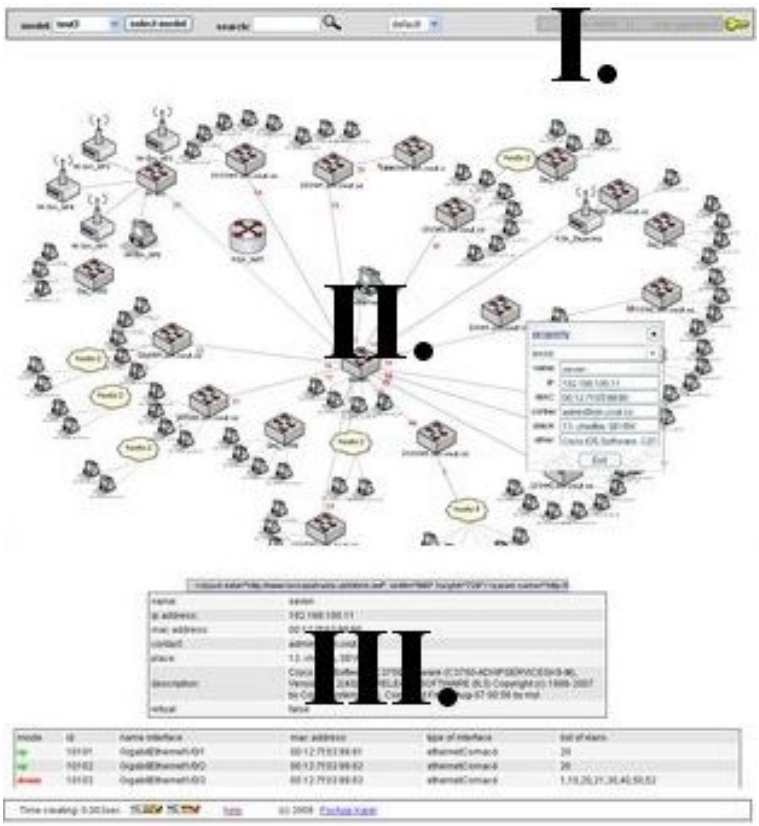
Nejprve nahrajeme obsah adresáře „agent“ do monitorovací stanici, ze které budeme měřit požadovanou počítačovou síť a která bude mít zabudovanou podporu Javy v minimální verzi 1.5. V adresáři agent se nachází konfigurační soubor, který obsahuje dále vypsané atributy.

- **version** - nepovinny údaj, který určuje vývojovou verzi agenta
- **address** - určuje http adresu nadřazeného serveru.
- **idmodel** - určuje identifikační číslo modelu, ve kterém se následně zobrazí měřená data.
- **idagent** - se používá k odlišení agentů v rámci jednoho modelu
- **time** - určuje dobu, na kterou se uspí vlákno discoverythread po dokončení jednoho cyklu
- **measuretime** - určuje dobu, na kterou se uspí vlákno measurethread
- **ordertime** - určuje dobu, na kterou se uspí vlákno orderthread
- **login** - autorizační jméno agenta, které je uvedeno v administraci u daného modelu idmodel
- **password -** autorizační heslo, které je uvedeno v administraci u daného modelu idmodel
- **public** - community string pro čtení hodnot ze SNMP zařízeních podporující SNMPv1
- **private** - heslo pro zápis hodnot na SNMP zařízení podporující SNMPv1
- **maxthread** - maximální počet vláken, které se mohou spustit při v prohledávacím vláknu

Ještě před tím, než začneme upravovat atributy agenta je nutné pro něj získat autorizační údaje. Ty získáme tak, že je dostaneme přiděleny od administrátora serveru a nebo si sami vytvoříme server a v něm si vytvoříme vlastní model, u které ho si nastavíme login a password, které bude agent používat. Vrátíme-li se ke konfiguračnímu souboru config.xml, pak je do něj nutné nastavit id daného modelu, login, password, public a webovou adresu, kde se daný server nachází. V dalším kroku je také nutné nastavit požadovaná zpoždění, maxthread podle velikostí zpoždění v rámci počítačové sítě. V posledním kroku nastavíme rozsah IP adres a identifikační číslo agenta. IP rozsah slouží k tomu abychom nastavili, kde všude bude agent hledat aktivní prvky. Id agenta je zase k tomu, abychom rozdělili síť mezi více agentů. Například nahrajeme dva různé agenty na rozdílné pracovní stanice v rámci jedné sítě a každému z nich nastavíme rozdílný IP rozsah adres a rozdílné idagent. Po nastavení serveru a agentů můžeme přistoupit ke spuštění agenta. To se provede tak, že spustíme soubor run.bat na přiloženém CD a nebo agenta aktivujeme příkazem java -jar agent.jar.

8.3 Struktura aplikace

Struktura aplikace je znázorněna na obrázku 8.1 a obsahuje tři části. A to základní menu, pracovní plochu a zobrazení informací o vybraném zařízení a o jeho portech. Základní menu složí k autorizaci uživatele, k procházení modelů a nebo nastavování různých zobrazovacích stylů při vizualizaci topologie. Pokud je daný uživatel autorizován, pak se mu zobrazí další funkce menu.



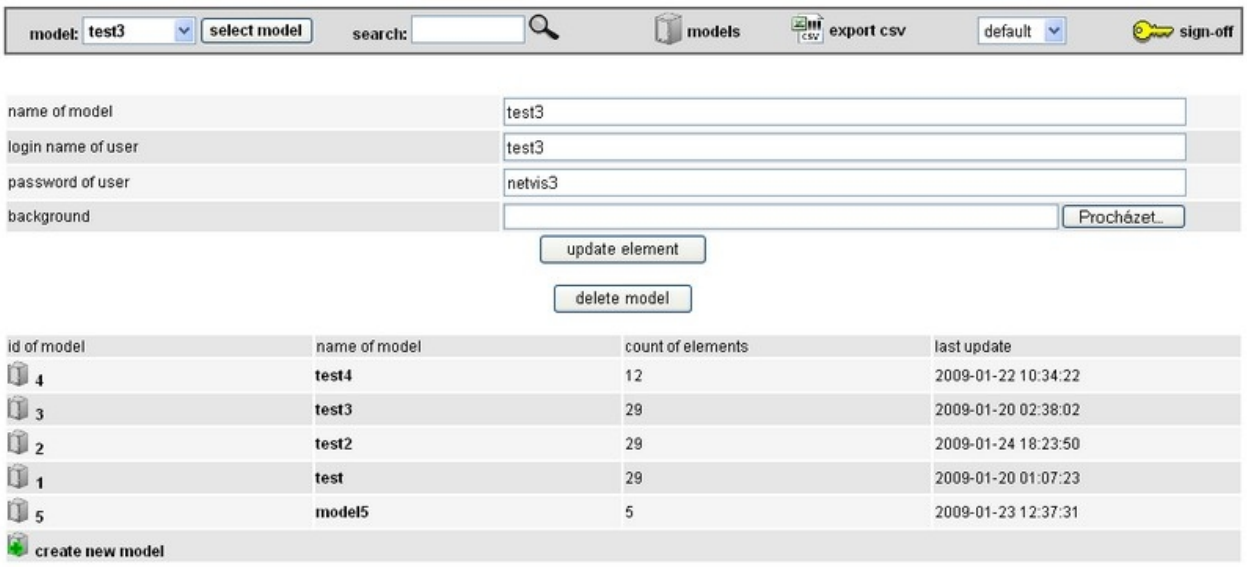
Obrázek 8.1: Zobrazení struktury uživatelského rozhraní

8.4 Autorizace a práce s modely

Autorizace se provádí v přihlašovacím políčku, které je zobrazeno na obrázku 8.2. Pokud se uživateli podaří autorizovat, pak může aktivovat další funkce, které se mu zobrazí. Autorizovaný uživatel může vytvářet, upravovat, mazat modely a upravovat jejich grafické pozadí, prohlížet exportované CSV soubory, upravovat topologii vizualizované sítě, vytvářet, mazat a upravovat virtuální elementy. Autorizovaný uživatel také může vytvářet, upravovat a mazat porty virtuálních elementů a vytvářet spoje mezi nimi. Dále autorizovaný uživatel může registrovat a odhlašovat měření zatížení jednotlivých reálných portů a zobrazovat grafy tohoto zatížení.

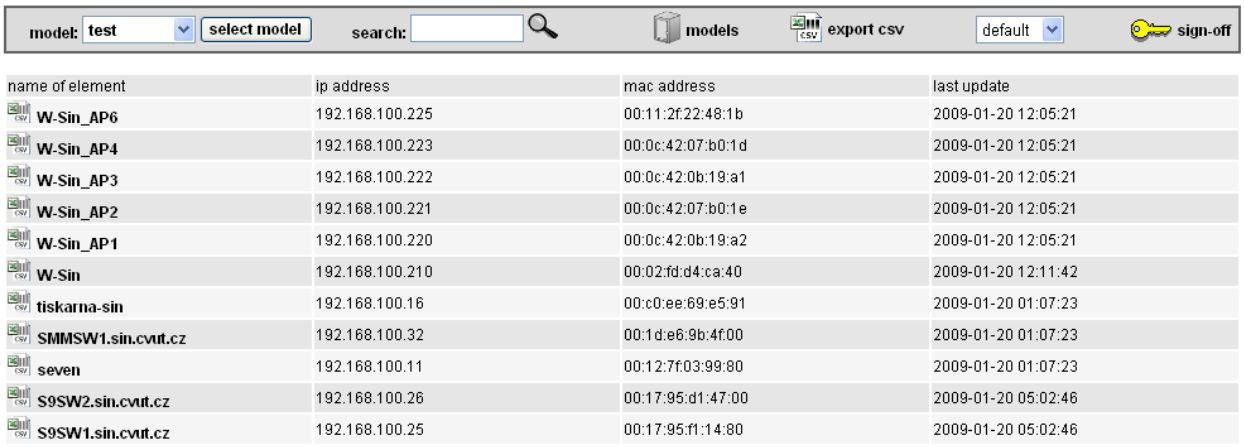
Obrázek 8.2: Přihlašování a odhlašování uživatele.

Modely jsou samostatné a navzájem oddělené bloky, které umožňují spravovat větší množství sítí a modelů v rámci jedné instance aplikace. Rozhraní pro správu modelů je zobrazeno na obrázku 8.3. V dolní části výpisu je možné vybrat konkrétní modely, se kterými chceme pracovat. Po vybrání modelů můžeme upravit jeho vlastnosti, měnit jeho pozadí a nebo jej smazat. Další funkcí je „create new model“, která zobrazí upravené rozhraní přizpůsobené pro vytváření modelů.



Obrázek 8.3: Rozhraní pro správu modelů.

Součásti administrátorského rozhraní je také rozhraní pro zobrazení CSV souborů zobrazené na obrázku 8.4. CSV soubory obsahují informace o daném zařízení a jeho portech.

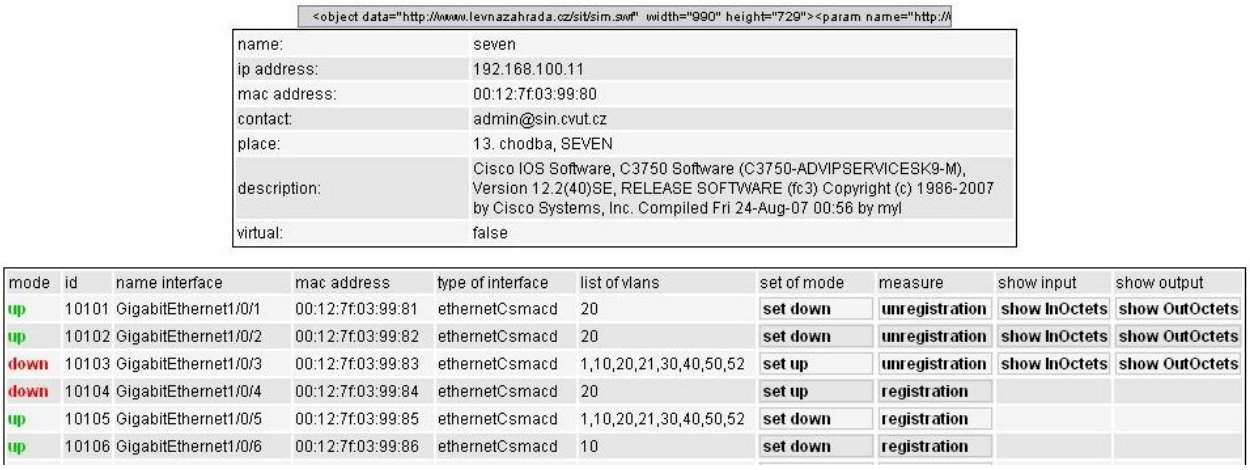


name of element	ip address	mac address	last update
W-Sin_AP6	192.168.100.225	00:11:2f:22:48:1b	2009-01-20 12:05:21
W-Sin_AP4	192.168.100.223	00:0c:42:07:b0:1d	2009-01-20 12:05:21
W-Sin_AP3	192.168.100.222	00:0c:42:0b:19:a1	2009-01-20 12:05:21
W-Sin_AP2	192.168.100.221	00:0c:42:07:b0:1e	2009-01-20 12:05:21
W-Sin_AP1	192.168.100.220	00:0c:42:0b:19:a2	2009-01-20 12:05:21
W-Sin	192.168.100.210	00:02:fd:d4:ca:40	2009-01-20 12:11:42
tiskarna-sin	192.168.100.16	00:c0:ee:69:e5:91	2009-01-20 01:07:23
SMMSW1.sin.cvut.cz	192.168.100.32	00:1d:e6:9b:4f:00	2009-01-20 01:07:23
seven	192.168.100.11	00:12:7f:03:99:80	2009-01-20 01:07:23
S9SW2.sin.cvut.cz	192.168.100.26	00:17:95:d1:47:00	2009-01-20 05:02:46
S9SW1.sin.cvut.cz	192.168.100.25	00:17:95:f1:14:80	2009-01-20 05:02:46

Obrázek 8.4: Rozhraní pro zobrazení souborů CSV.

8.5 Práce s elementy a jejich porty

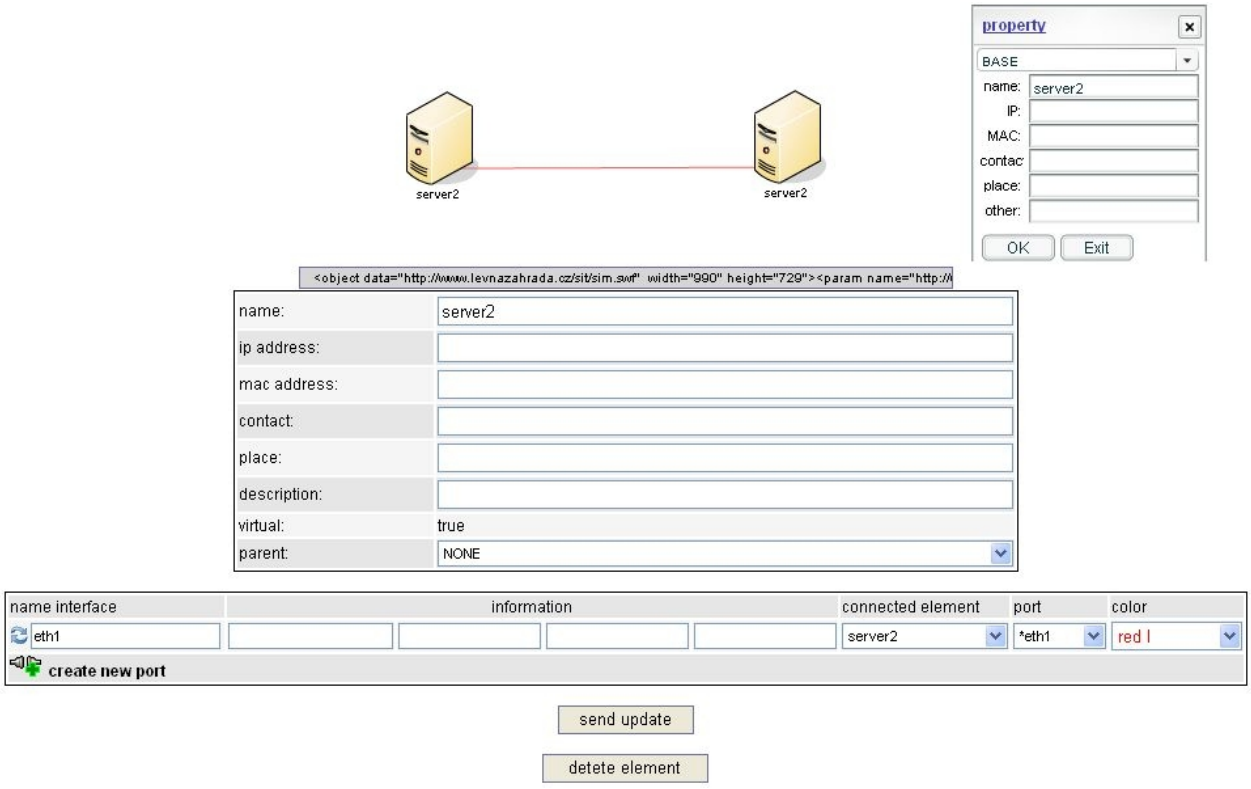
Elementy jsou zařízení, která v rámci zkoumané sítě podporují SNMP a nebo to jsou virtuální prvky V rámci topologie může neautorizovaný uživatel posouvat jednotlivé elementy a zobrazovat si o nich a o jejich portech informace. Autorizovaný uživatel může úpravy topologie trvale uložit v databázi. Autorizovaný uživatel zároveň může například nahrát obrázek půdorysu domu na pozadí modelu a zjištěné elementy rozmístit tak aby jejich pozice na půdorysu přibližně odpovídala reálnému umístění. Pokud uživatel dvakrát klikne na element, pak se mu zobrazí seznam portů. Ale pokud klikne na reálný element autorizovaný uživatel, pak se mu zobrazí seznam portů spolu s dalšími operacemi, které může nad daným zařízením provádět, tak jak je to zobrazeno na obrázku 8.5. Autorizovaný uživatel může jednotlivé porty reálných elementů zaregistrovat k měření a posléze si nechat zobrazit výsledné grafy.



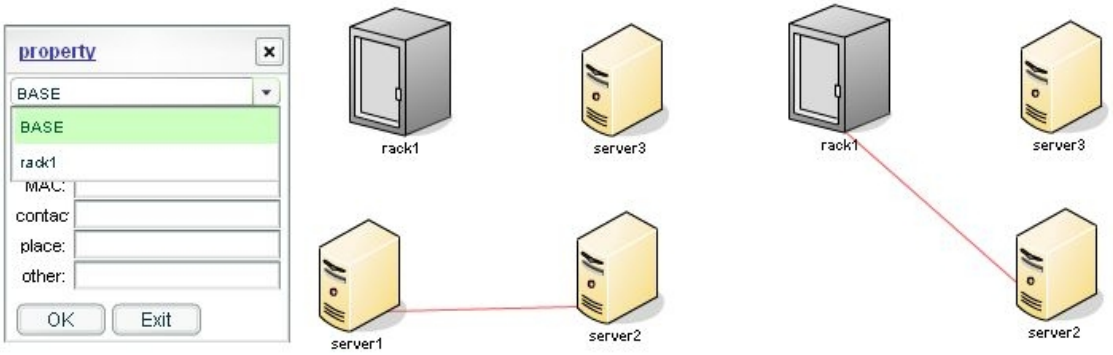
mode	id	name interface	mac address	type of interface	list of vlans	set of mode	measure	show input	show output
up	10101	GigabitEthernet1/0/1	00:12:7f:03:99:81	ethernetCsmacd	20	set down	unregistration	show InOctets	show OutOctets
up	10102	GigabitEthernet1/0/2	00:12:7f:03:99:82	ethernetCsmacd	20	set down	unregistration	show InOctets	show OutOctets
down	10103	GigabitEthernet1/0/3	00:12:7f:03:99:83	ethernetCsmacd	1,10,20,21,30,40,50,52	set up	unregistration	show InOctets	show OutOctets
down	10104	GigabitEthernet1/0/4	00:12:7f:03:99:84	ethernetCsmacd	20	set up	registration		
up	10105	GigabitEthernet1/0/5	00:12:7f:03:99:85	ethernetCsmacd	1,10,20,21,30,40,50,52	set down	registration		
up	10106	GigabitEthernet1/0/6	00:12:7f:03:99:86	ethernetCsmacd	10	set down	registration		

Obrázek 8.5: Rozhraní o reálném elementu pro autorizovaného uživatele

U virtuálních elementů autorizovaný uživatel může upravovat všechny jejich informace v rozšířeném rozhraní podobně jak je zobrazeno na obrázku 8.6. V tomto rozšířeném rozhraní může administrátor vyplňovat jednotlivé vlastnosti portů a virtuálního elementu a také vhodně propojit již existující elementy mezi sebou tak, jak je znázorněno na daném obrázku 8.6 a 8.7.



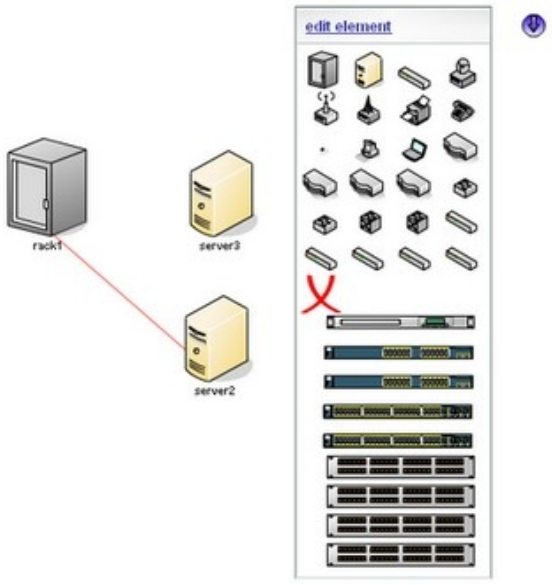
Obrázek 8.6: Uživatelské rozhraní o virtuálním elementu pro autorizovaného uživatele.



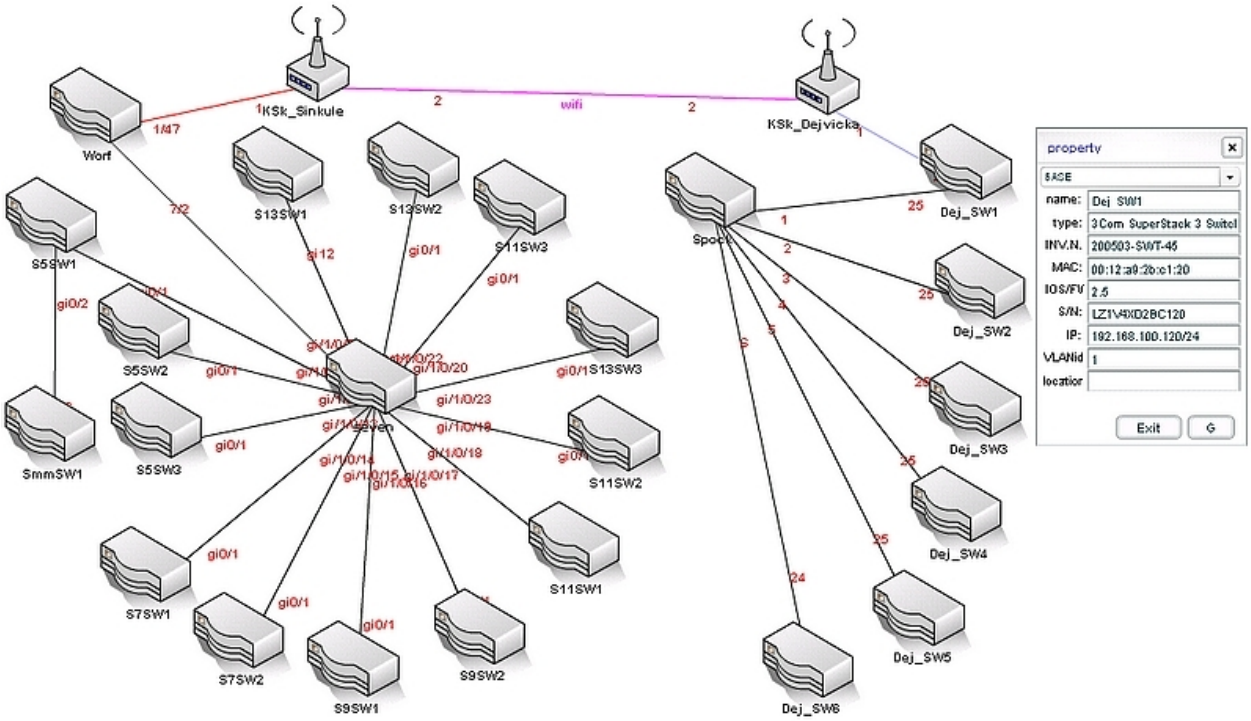
Obrázek 8.7: Ukázka vnoření elementu do racku.

Jedno takové spojení prvků za pomoci rozšířeného rozhraní je ukázáno na obrázku 8.7. Tento obrázek také zobrazuje jak probíhá vnořování jednotlivých prvků do racků. Přes property a

nebo znovu přes rozšířené uživatelské rozhraní si nastavíme atribut parent a následně daný rack převezme spojové vlastnosti předchozího elementu. Další možností, kterou autorizovaný uživatel má, je odstranění a vytvoření elementu. To se provádí přes pracovní menu, které se vyvolá kliknutím na kruhové tlačítko v pravé horní části pracovní plochy tak jak je ukázáno na obrázku 8.8. Zato na obrázku 8.9 je zobrazen větší virtuální model, kde je aktivováno informační okno se základním popisem vybraného prvku.



Obrázek 8.8: Zobrazení pracovního menu.



Obrázek 8.9: Ukázka virtuálního modelu.

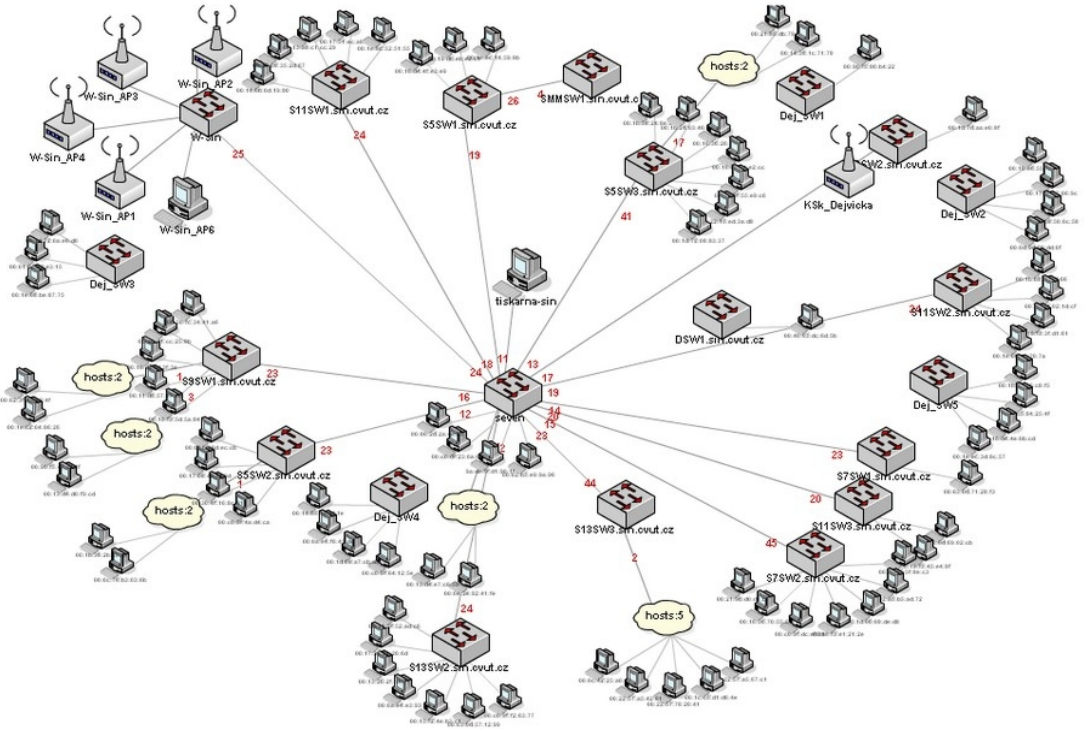
9 PŘÍLOHA: Seznam použitých zkratek

AJAX	Asynchronous JavaScript and XML
AMF	Action Message Format
AP	Wireless access point
ASN.1	Abstract Syntax Notation One
AVM2	ActionScript Virtual Machine 2
CAM	Content Addressable Memory Table
CDP	Cisco Discovery Protocol
CSV	Comma separated values
HTML	HyperText Markup Language
ICMP	Internet Control Message Protocol
IETF	Engineering Task Force
IR	Intermediate Representation
JNI	Java Native Interface
LLDP	Link Layer Discovery Protocol
MAC	Media Access Control
MIB	Management Information Base
MRTG	Multi Router Traffic Grapher
NMS	Network Management Systém
OID	Object Identifier
PDU	Protocol Data Unit
PHP	Hypertext Preprocessor
PNG	Portable Network Graphics
RMON	Remote Monitoring
SGMP	Simple Gateway Monitoring Protocol
SNMP	Simple Network Management Protokol
SMI	Structure of Management Information
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SONMP	Nortel Discovery protokol
STP	Spanning tree protocol
SWF	Shockwave Flash file
UDDI	Universal Description Discovery and Integration
URL	Uniform Resource Locator
VLAN	Virtual Local Area Networks
WSDL	Web Services Description Language
XML	Extensible Markup Language
Xpath	XML Path Language

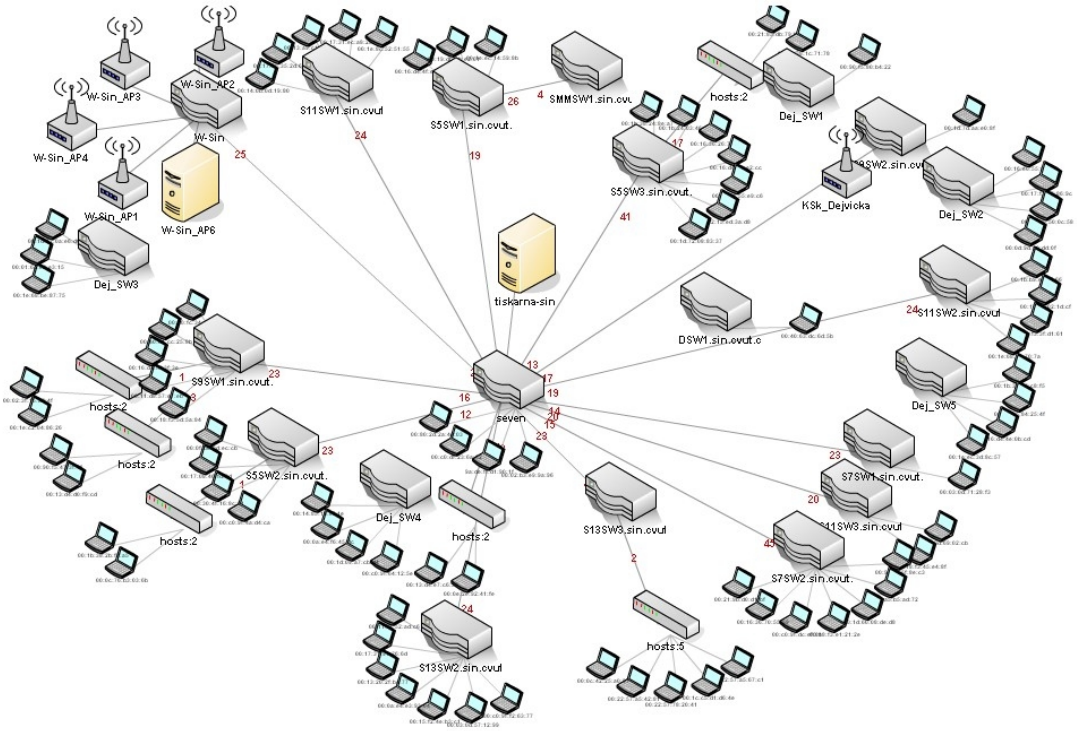
10 PŘÍLOHA: Obsah instalačního balíku

- **[agent]** - obsahuje spustitelného agenta a jeho konfigurační soubory.
  - **[lib]** - obsahuje knihovnu SNMP4J.jar.
  - **agent.jar** - spustitelný agent
  - **config.xml** - konfigurační soubor agenta
  - **run.bat** - soubor, který spustí agenta
- **[data]** - obsahuje ukázková data
- **[server]** - instalační skripty pro server
- **[src]** - zdrojové soubory agenta a serveru
- **[test]** - text diplomové práce v PDF
- **install.txt** - postup instalace
- **readme.txt** - popis obsahu adresáře

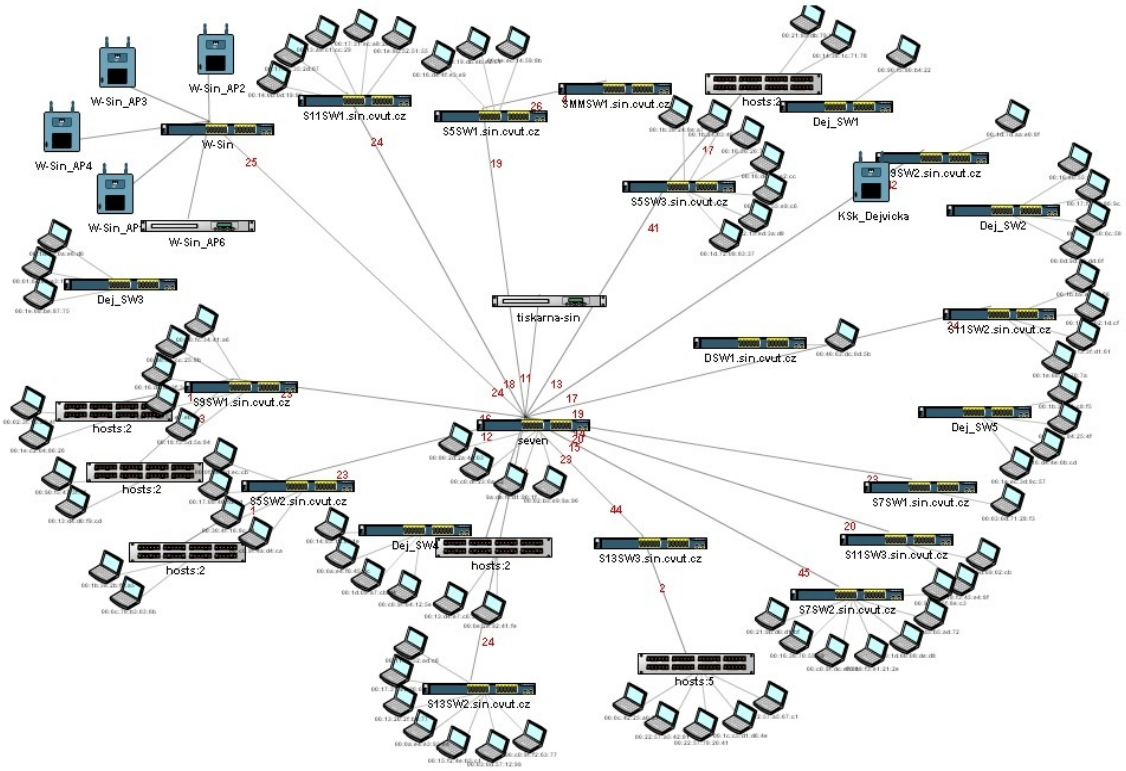
11 PŘÍLOHA: Ukázky grafických stylů



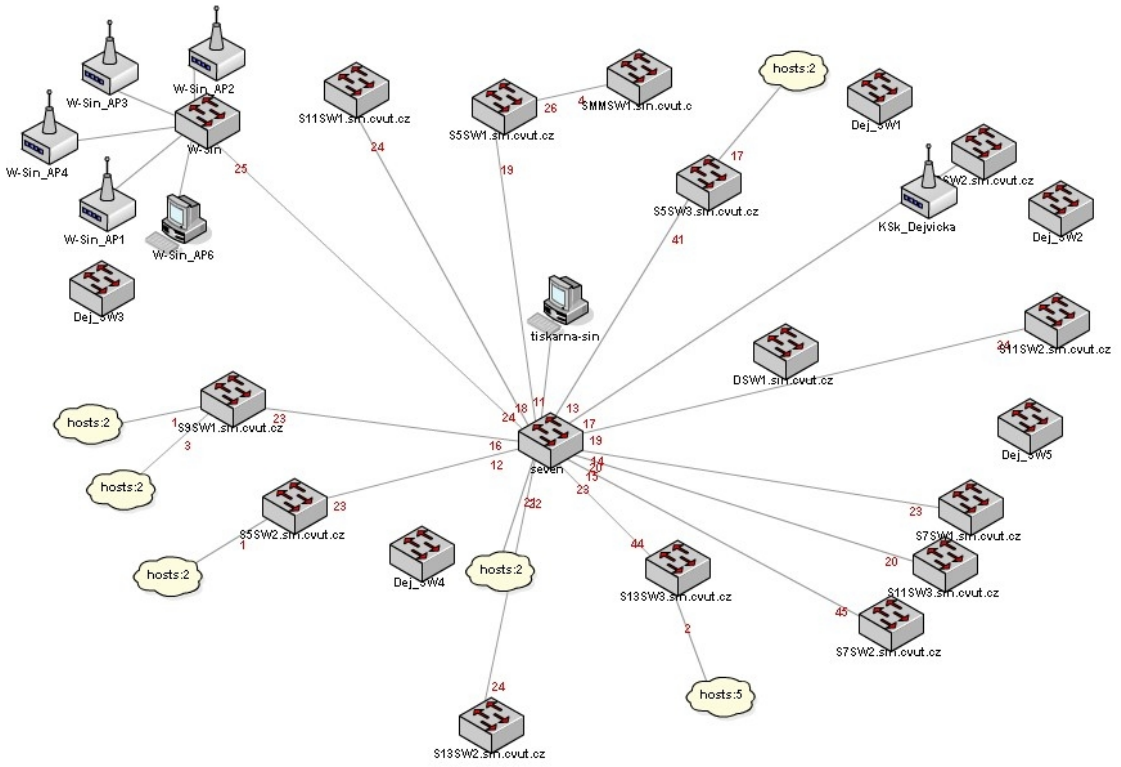
Obrázek 11.1: Zobrazení sítě v grafickém módu default.



Obrázek 11.2: Zobrazení sítě v grafickém módu default2.



Obrázek 11.3: Zobrazení sítě v grafickém módu hardware.



Obrázek 11.4: Zobrazení sítě v grafickém módu noport.