Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Automatizace síťových prvků

Bakalářská práce

Autor: Gábrle Miloš

Studijní obor: AI3

Vedoucí práce: Ing. Vladimír Soběslav, Ph.D.

Hradec Králové duben 2015

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne

Miloš Gábrle

Poděkování:

Děkuji svému vedoucímu práce panu Ing.Vladimíru Soběslavovi za vedení práce a rodině za podporu při jejím psaní.

Anotace:

Bakalářská práce analyzuje základní nástroje, srovnává technologie a řešení, které je možné využít pro automatizaci. Na základě této analýzy budou v praktické části implementovány vybrané úlohy z oblasti automatizace provozu síťových prvků.

Annotation:

Bachelor thesis is analysing basic tools, compares the technologies and solutions which can be used for network automation. There will be implemented some chosen cases from the area of network automation on the network devices in the practical part of this thesis which will be based on this analysis.

OBSAH

1. Úvod

V dnešní době rozsáhlých topologií sítí, je na správce sítí v každé organizaci kladen stále větší důraz na šetření nákladů a času vynaložených na správu sítě, ale při tom musí poskytovat stále více širokopásmových služeb, neboť uživatelé stále zvyšují požadavky na síť využíváním stále více mobilních zařízení, jako jsou tablety, notebooky a chytré telefony, které zaměstnanci a návštěvníci firmy používají každý den. S rostoucím množstvím zařízení stoupají požadavky na spolehlivé připojení a zabezpečení. Co zabere správcům sítě velkých společností nejvíce času? Právě stále se opakující jednoduché monotónní úkoly spojené například s přesunem zařízení z jednoho místa na druhé, autentifikace zařízení, úvodní konfigurace portů a zařízení [1].

Pouhá úvodní konfigurace přepínače, například v datovém centru, ještě před jeho samotným uvedením do funkčního stavu je při velkém množství aktivních prvků velice zdlouhavé, neboť je potřeba napsat dlouhou řadu příkazů a nahrání správné image, která jsou pro zkušeného profesionála sice jednoduchá, ale vzhledem k počtu příkazů se i takto jednoduchý proces zabere mnoho času, který by mohl zaměstnanec využít daleko lépe [2].

Síťovou automatizaci lze realizovat hned několika způsoby. Můžeme si zakoupit si program, který přes profesionálně a přehledně vytvořené uživatelské rozhraní umožňuje jednoduchou práci se sítí a její správu bez zdlouhavého psaní kódu řádek po řádku do CLI síťového operačního systému aktivního prvku, nebo si můžeme napsat vlastní program pomocí nějaké z mnoha dostupných knihoven pro programovací jazyk, který nám nejvíce vyhovuje (pokud má tyto knihovny dostupné), nebo můžeme psát applety přímo do CLI, či vkládat skripty do zařízení z externích souborů, které mohou být spuštěny po výskytu požadované události.

Jak již bylo řečeno, technologii pro automatizování sítě si musí každý správce sítě zvolit sám. Záleží na používaném síťovém operačním systému, na jeho vlastních znalostech programovacího jazyka, který se chystá využít a jeho požadavcích na funkce. Každý programovací, nebo skriptovací jazyk má specifickou sadu příkazů a pravidel, kterými se musí řídit, z čehož vyplývají jeho vlastnosti a limitace [3].

2. CÍLE A METODIKA PRÁCE

Cílem této bakalářské práce je seznámení s pojmem automatizace síťového provozu, jejími výhodami a seznámení s dostupnými technologiemi pro její realizaci. Zaměřena bude především na technologie pro síťový operační systém Cisco IOS a to z důvodu jeho dostupnosti v laboratořích školy a znalosti tohoto operačního systému z vyučovacích hodin *Počítačových sítí 1-3*. Hlavním cílem je seznámení s novou technologií právě od společnosti Cisco, která se nazývá OnePK (One Platform Kit), ukázat tak její možné využití, které nám nabízí a na praktických úlohách ukáže implementaci vybraných úloh společně s porovnáním s dalšími technologiemi dostupnými k automatizování sítě, například se skripty v jazyce TCL nebo využití některých manažerů vložených v přímo v operačním systému IOs.

Práce nás seznámí s pojmem síťová automatizace, objasní proč je v dnešní době prakticky nezbytné automatizovat některé úlohy v počítačové síti a ukáže, v jakých oblastech je možné efektivně využít automatizaci k našemu prospěchu a usnadnění práce.

Budou zde popsány síťové operační systémy aktivních síťových prvků. Pro každý z těchto operačních systémů zde budou uvedeny nástroje a technologie umožňující automatizaci úloh. Ukážeme si technologie dostupné pro síťové operační systémy IOs, Junos a RouterOS.

Práce bude zaměřená na technologie pro operační systém IOs, které budou vyzkoušeny v praktické části této práce, především v nástroji All In One VM od společnosti Cisco, což je virtuální prostředí v distribuci Linux Ubuntu obsahující v sobě již všechny potřebné soubory, programy a návody umožňující návrh vlastní virtuální topologie k testování vyvinutých aplikací. Právě v tomto prostředí budou vyvinuty ukázkové aplikace za účelem vyzkoušení možností OnePK.

3. AUTOMATIZACE ÚLOH

V této kapitole se zaměříme na síť jako celek, na jednotlivá zařízení a protokoly potřebné k její funkčnosti a zaměříme se především na platformy a úkony související s tématem automatizace a na úkony, které je možno jakýmkoliv způsobem automatizovat.

3.1. ZAŘÍZENÍ V SÍTI

Nejprve se podíváme na jednotlivá zařízení, se kterými se setkáváme při práci se sítí [4]. Je velice důležité u každého zařízení vědět k čemu slouží a na jaké vrstvě daného síťového modelu pracuje. Mezi 2 hlavní síťové modely patří modely *OSI* a *TCP/IP*. Model OSI se skládá ze sedmi vrstev, kde každá vrstva popisuje služby, které daná vrstva poskytuje, funkce, které vykonává a protokoly, které na ní pracují. Vrstvy modelu OSI a jejich úkoly jsou následující:

- **Aplikační vrstva** poskytuje služby (přenos dat, e-mail, atd.) softwarovým aplikacím, které pak přímo komunikují s uživatelem.
- **Prezentační vrstva** překládá data obdržená od aplikací do dat, která mohou být dále zpracována nižšími vrstvami OSI modelu, a naopak při obdržení dat zpracovaných nižšími vrstvami je překládá do formátu, kterému rozumí koncová aplikace.
- **Relační vrstva** má na starost správu komunikace mezi jednotlivými aplikacemi. Vytváří, udržuje a ukončuje relace podle potřeb jednotlivých aplikací.
- **Transportní vrstva** dělí data obdržená z vyšší vrstvy do menších částí (segmentů), které předává vrstvě síťové. Má na starost identifikaci aplikace, pro kterou jsou data určena. Na straně příjemce segmenty opět skládá dohromady a předává je vrstvě relační.
- **Síťová vrstva** přebírá od transportní vrstvy segment, přidáním hlavičky (formát hlavičky je závislý na použitém komunikačním protokolu) z něho vytváří paket, který odesílá datové vrstvě k dalšímu zpracování. Na základě IP adresy nalezne cílového hosta, kde odebráním hlavičky získá opět segment a předává ho vrstvě transportní.

- **Datová vrstva** vytváří rámec, který odesílá na vrstvu fyzickou. Řídí přenos dat na fyzické vrstvě a zpřístupňuje médium vyšším vrstvám nezávisle na jeho typu. Adresuje na základě fyzických (MAC) adres koncových zařízení.
- **Fyzická vrstva** kóduje jednotlivé bity vytvořené vrstvou datovou do signálu a tento signál pak přenáší po fyzickém médiu. Do fyzické vrstvy patří přenosové médium (bezdrátový přenos, optický kabel, nebo elektrický kabel), způsob reprezentace bitů na daném médiu a způsob kódování dat.

Model TCP/IP má obdobné úkoly, jen se dělí do čtyř vrstev, kterými jsou:

- **Aplikační vrstva** zahrnuje stejné úkoly jako vrstvy aplikační, prezentační a relační modelu OSI
- **Transportní vrstva** odpovídá transportní vrstvě modelu OSI
- **Internetová vrstva** odpovídá síťové vrstvě modelu OSI
- **Vrstva přístupu k datům** vykonává to samé jako vrstvy datové a fyzické modelu OSI

Z hodin síťové akademie cisco [4] víme, že v síti se můžeme setkat se zařízeními jako:

- **Router** (směrovač) slouží ke směrování paketů mezi sítěmi, tyto pakety přeposílá na základě IP adres, které získává z hlaviček paketů a tak je jasné, že pracuje na síťové vrstvě modelu OSI. Směrovač také dále kontroluje chyby v hlavičce.
- **Switch** (přepínač) jednotlivé rámce odesílá pouze koncovým zařízením, kterým jsou určeny. Pracuje na druhé vrstvě OSI modelu a adresování provádí na základě fyzických adres koncových zařízení. Jelikož přepínání je realizováno výkonnými hardwarovými prostředky, samotné přepínání je velice rychlý proces (existují také L3 přepínače, které pracují na vrstvě síťové a chovají se stejně jako směrovače, jen samotné směrování na nich je realizováno daleko rychleji)
- **Hub** (rozbočovač) pracuje na vrstvě fyzické a je to zařízení zasílající data, která obdrží na jednom portu na všechny ostatní porty. Jelikož neustále

pouze odesílá data na všechny porty, je zde vysoké riziko kolize dat, které se lineárně zvyšuje počtem zařízení, která jsou k němu připojena.

- **Bridge** (most) pracuje na vrstvě datové. Odděluje od sebe různé sítě, na základě tabulky fyzických adres určuje, do jaké sítě odešle paket, který obdržel na jednom z portů.
- **Repeater** (opakovač) jedná se pouze o zařízení zesilující signál, který obdrží na jednom portu, na port druhý. Pokud data odesíláme na delší vzdálenost, nebo chceme pokrýt větší plochu (např. bezdrátově) použijeme opakovač k zesílení signálu, který by jinak byl příliš slabý na jeho bezchybné zpracování.

Nejedná se o výčet všech zařízení spojených se síťovým provozem, ale pouze o pár příkladů. Tato práce je nejvíce zaměřená na práci se směrovači, tzn. práce na síťové vrstvě modelu OSI.

3.2. AUTOMATIZACE V PRAXI

Jak řekl Glenn O'Donnell, senior analytik u Forrester Research: „Automatizace nejen eliminuje každodenní práci, ale také odstraňuje chyby vzniklé při manuálním zpracování“. Odstraňuje tedy možnost chyby vzniklé pomocí lidského faktoru a zároveň tak šetří čas i peníze. Pojem síťová automatizace v dnešní době může odkazovat prakticky na jakýkoliv počet automatizovaných procesů, od automatizování jednoduchých procesů až po automatizaci časově velmi náročných úkolů. Automatizaci v síti rozdělujeme do tří základních skupin podle její funkce [5].

- **Automatizace místní sítě -** Místní síť (LAN) je plná konfiguračních nastavení a profilů, které musejí být velice často modifikovány pro dobrý chod sítě. Manuální nastavení přepínačů a směrovačů a monitorování sítě není pouze časově velmi náročný proces, ale také pro velké množství řádků se zvýší prostor pro tvorbu chyb a tím tak chybně konfigurovat celou síť.

  Existují nástroje pro síťovou správu a konfiguraci, které vytváří databázi, v které udržují informace o zařízeních v síti, jejich konfiguraci a službách poskytovaných v síti. Dále kontrolují v této databázi, jestli zařízení nejsou špatně nastavená a pokud ano, informují správce o stavu daného

zařízení a ten může následovně sám chyby odstranit.

Dále existují nástroje, které generují nastavení pro jednotlivá zařízení a jejich služby. Patří mezi ně například open source program *Netomata Config Generator* [6] nebo jeho komerční obdoby *Alterpoint* [7] a *Network Configuration Manager* [8] od společnosti *Solarwinds*. Jedinými vstupy, které musíme do těchto nástrojů zadat, je model popisující vaši síť a šablony pro nastavení jednotlivých zařízení (směrovače, přepínače, atd.) a služby (DHCP, SNMP, DNS, atd.) pro které chcete vygenerovat nastavení. Na základě těchto vstupů nástroje vygenerují soubory s nastavením pro požadované zařízení a služby, které stačí už jen nainstalovat.

- **Automatizace pro virtualizované prostředí** - Ve společnostech realizujících virtualizaci se stala automatizace naprostou nezbytností. Všechna virtualizovaná prostředí nejsou nezbytně vázaná na konkrétní hardware a navíc virtuální stroje mohou být zapínány, vypínány a dokonce i přesunuty mezi jednotlivými servery. Pomocí automatizace je nám umožněno předcházet možným problémům pomocí sledování událostí a plánů.

  Pokud například potřebuje webová aplikace větší výpočetní výkon z důvodu velkého vytížení webové aplikace uživateli, může aplikace zažádat o spuštění více virtuálních strojů pro zvládnutí momentálního vytížení a po té nepotřebné virtuální stroje opět vypne.
- **Automatizace datových center** - V datových centrech je automatizace ještě důležitější, než byla v předchozím případě. Nachází se zde totiž mnoho jednotlivých systémů jako: úložné sítě, infrastruktura IP telefonů a podniková správa spotřeby energie běží společně v jednom datovém centru a automatizace bude velice důležitý faktor k tomu, aby všechny tyto systémy pracovaly kohezně.

  Například: podniková správa spotřeby má možnost snížit spotřebu energie v datovém centru, pomocí vypnutí nečinných služeb, odhlášení IP telefonů atd. Podobně, například při zálohování nebo při průběhu jiných náročných procesů, automatizovaná datová centra mohou upravit síť a prioritu serverů tak, aby těmto procesům uvolnila dostatek výpočetního výkonu pro jejich co možná nejrychlejší ukončení. A tak v datových centrech

snižuje potřebu dohledu člověka na systém a zároveň poskytuje uživatelům prostředky, které potřebují.

Jak vidíme, automatizaci můžeme realizovat ve velice široké oblasti, a proto se v této práci zaměříme na první ze tří výše uvedených možností automatizace

3.3. KONFIGURACE A SPRÁVA SÍTĚ

Jak bylo již řečeno na konci předchozí podkapitoly, tato práce se bude zaměřovat na oblast správy síťových zařízení. A o co se tedy v oblasti správy sítě jedná? Jak uvádí Alexander Clemm, Ph.D. v [9]:

„Síť je jako živý organismus, a proto ji lze přirovnat k pacientovi, který je na jednotce intenzivní péče. Nejblíže k síti u takového pacienta má krevní oběh. Musí se stále kontrolovat, zda je tlak v normě, jestli krev obíhá po celém těle a jakákoliv chyba si vyžaduje okamžitou reakci doktora. Na základě sledování stavu pacienta doktor může reagovat a stanovit určitý postup léčby, kde po delším sledování určí postup za úspěšný a v případě neúspěšného postupu ho může změnit.

Podobně jako u pacienta i síť musíme neustále sledovat a hlídat její stav, kde stejně jako u pulzu pacienta jakékoliv kritické hodnoty mohou naznačovat blížící se potíže, se kterými se bude potřeba co nejdříve vypořádat. Proto je potřeba síť neustále monitorovat a reagovat tak na jakékoliv změny v jejím chodu.“

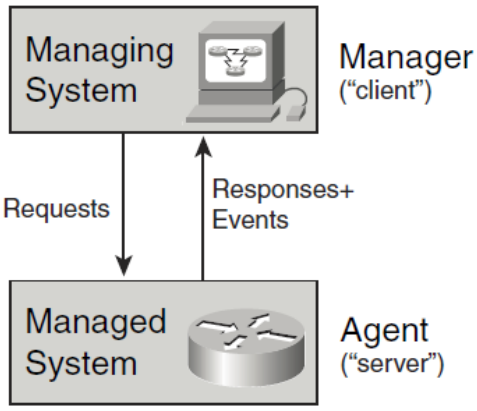
Dále také uvádí, že správa sítě se skládá z aktivit, metod, procedur nebo nástrojů, dělících se do následujících skupin:

- **Operace** zahrnuje všechny úkony spojené s udržováním sítě a všech služeb v chodu a bez jakýchkoliv problémů. Zaměřuje se na neustálé sledování sítě, aby mohl na vzniklý problém reagovat co nejdříve. V ideálním případě ještě dříve, než vzniklá chyba ovlivní koncového uživatele služeb dané sítě.
- **Administrace** zahrnuje udržování informací o všech prostředcích v síti a jejich nastavení. Všechny tyto informace udržuje pod jednou střechou, což je nezbytné k potřebné kontrole nad celým systémem.
- **Údržba** obsahuje opravy spojené s chybami v síti, ale zároveň také pokud dochází k modernizaci jakéhokoliv zařízení v dané síti. Například pokud na nějakém zařízení je potřeba nahrát nový operační systém nebo vyměnit

zásuvný modul či síťovou kartu. Zároveň do této skupiny také spadají různá preventivní a proaktivní opatření, aby vše běželo hladce.

- **Opatření** je považováno za každou akci, spojenou s konfigurací prostředků tak, aby podporovaly požadovanou službu. Například konfigurování zařízení tak, aby nový zákazník mohl využívat služby spojené s přenosem zvuku.

Aby zařízení mohlo být spravováno, musí poskytovat nějaké rozhraní, skrze které bude moci systém pro správu zařízení komunikovat s tímto zařízením. Takové rozhraní umožňuje systému zasílat požadavky konkrétním zařízením jako například požadavek na konfiguraci rozhraní, dotaz na status rozhraní atd. Podobně může také zařízení odesílat systému informace. Může se jednat o odpovědi na požadavky, nebo také informace o výskytu neočekávané události na zařízení. Komunikace mezi agentem (ang. Agent) a správcem (ang. Manager) je zobrazena na Obrázek 1. Agent (spravovaný systém) je libovolný síťový prvek, který realizuje změny, které mu zadává správce (systém, který spravuje).



Obrázek 1 - Komunikace mezi agentem a správcem

Komunikace mezi správci a agenty je realizována pomocí zasílání zpráv. Tyto zprávy představují jádro management protokolu. Jak dále uvádí Alexander Clemm, bohužel nelze zde využít klasického IP protokolu, neboť ten zaručí pouze to, že se agent a správce mohou slyšet, takže sám o sobě nezaručí, že budou mluvit stejným jazykem, tudíž si nemusí rozumět. Příklady management protokolů jsou například protokoly SNMP a Netconf.

3.3.1. SNMP

Simple Network Management Protocol (dále jen SNMP) je protokol, který jak již bylo řečeno, realizuje komunikaci mezi správcem a agentem. Originální SNMP protokol se dnes nazývá SNMP verze 1 (dále jen SNMP v1), a pro svou jednoduchost je do dnešní doby stále velmi využíván u zařízení s omezeným výkonem. V dnešní době existuje už i verze SNMP v3, která už v sobě obsahuje jistá bezpečnostní opatření a tak dělá SNMP konečně bezpečným protokolem. Většinou SNMP v1 využívá protokolu UDP, což znamená, že doručení paketu není zaručeno. SNMP poskytuje 5 primitivních operací, protože na jednoduchosti je celý protokol postaven a kvůli jednoduchosti je také využíván do dnešního dne. Těmito operacemi jsou podle Alexandera Clemma:

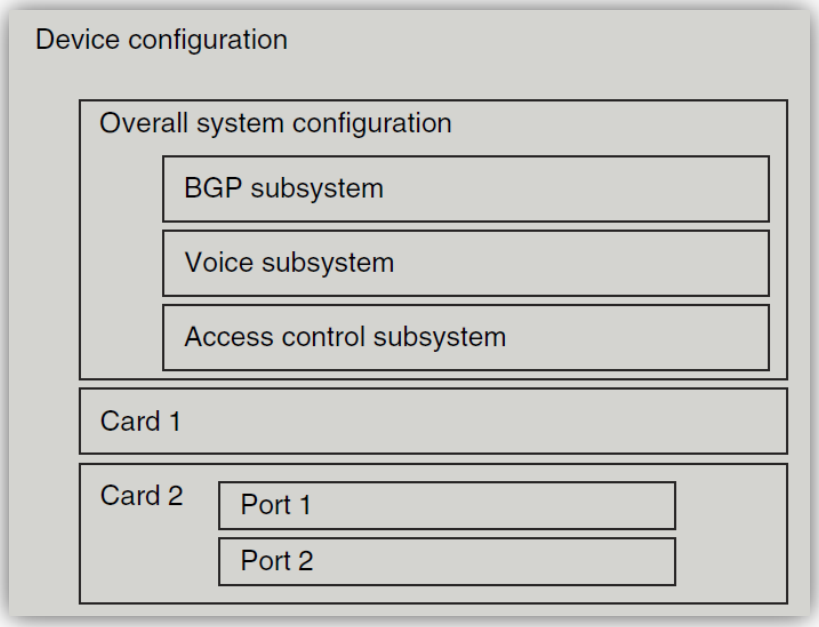
- **GET REQUEST** - Správce používá tuto operaci, aby získal informace od agenta. Parametrem tohoto požadavku je seznam proměnných, které definují, jaké objekty správce požaduje. Hodnoty těchto parametrů jsou nastaveny na hodnotu null. Ačkoli je možné odeslat požadavek na více objektů, je důležité vědět, že je možné odeslat pouze zprávu o určité velikosti. Po přesažení této velikosti mohou aplikace, komunikující skrze SNMP, zaznamenávat problémy.
- **GET-NEXT REQUEST** – Opět se jedná o získání informací správcem od agenta, nicméně zde správce žádá objekt, který v systému následuje po posledním vyžádaném objektu.
- **GET RESPONSE** – Zde agent zasílá zprávu správci, jako odpověď na požadavky *get*, nebo *get-next*. Odpověď musí obsahovat identifikaci požadavku, na který odpovídá, informace o tom, jestli daný požadavek obsahuje chyby a samotné objekty, které byly vyžádány.
- **SET REQUEST** – Odesílá správce agentovi, pokud chce nějaký objekt nastavit na určitou hodnotu. Struktura je stejná jako u *get request*, s tím rozdílem, že hodnoty jednotlivých objektů nejsou nastaveny na null, ale mají určitou hodnotu, kterou jim po přečtení agent nastaví.
- **TRAP** – Je používán k tomu, aby agent sdělil správci údaje o události, která u něho nastala. Správce na tuto zprávu neodesílá žádnou odpověď.

3.3.2. NETCONF

Alexander Clemm také dále uvádí, že tento protokol už je k dispozici téměř deset let, předpovídající vzrůst popularity jazyka XML. Deset let je ve světě moderní technologie hodně dlouhá doba a tento protokol se po tuto dobu stále drží na špici.

Byl navržen specificky pro správu konfigurace síťových zařízení a momentálně není zaměřen na monitorování sítě a získávání informací o jejím stavu. Pro tyto účely jsou využívány jiné protokoly (například již zmíněný SNMP). Protože je Netconf tak specificky zaměřený pouze na konfiguraci a není tak obecný, vyplňuje mezeru, kterou nestačí zaplnit ostatní protokoly.

Je založen na myšlence, že konfigurační datové úložiště (sada příkazů, které musí být provedeny, aby se zařízení nastavilo do požadovaného stavu) může být zamýšleno jako soubor a jako k souboru se k němu tedy můžeme chovat [9]. Netconf poskytuje operace, umožňující základní práci s těmito datovými úložišti. Například může upravovat jejich obsah a tím tak měnit konfiguraci zařízení, nebo celý soubor může získat, či odeslat na určité zařízení. Data v těchto úložištích jsou hierarchicky uspořádána (Obrázek 2), a tím jsou lépe přístupná aplikacím a celá konfigurace tak může být rozdělena do několika podkonfigurací.



Obrázek 2 - Hierarchická struktura datových uložišť (převzato z [9])

Jak již bylo naznačeno Netconf je pro zachování své hierarchické struktury a jednoduchou orientaci v dokumentu zakódován v jazyce XML. Jednotlivé konfigurace nebo podkonfigurace tedy mohou být přenášeny mezi agenty a správci jako XML dokument.

Operace protokolu Netconf jsou následující:

- **GET-CONFIG** je používán pro získání konfiguračního souboru ze zařízení. První parametr je cesta k souboru, který chceme získat. Druhý parametr umožňuje určit, jestli chceme celý konfigurační soubor nebo jen nějakou jeho část, kterou jednoduše určíme jejím názvem v XML souboru. Primárně získává running config (aktivní konfiguraci) daného zařízení.
- **GET** je jediný příkaz, který umožňuje získat nějaké informace z agenta. Umí ale zobrazit pouze informace, které bychom mohli získat příkazem show v CLI.
- **EDIT-CONFIG** je používán k úpravě konfiguračních datových úložišť. Prvním parametrem je cesta k souboru, který má být upraven. Následně je potřeba uvést, kde se změna bude provádět (opět celý dokument nebo jen jeho část) a posledním parametrem je samotná změna, kterou chceme provést.
- **COPY-CONFIG** je velice podobný příkazu *edit-config*, nicméně konfigurace je nahrazena celá a nelze určit, zda má být nahrazena jen její část jako v předchozím případě. Parametry jsou pouze cesty ke zdrojovému a cílovému souboru.
- **DELETE-CONFIG** vymaže konfiguraci ze zařízení. Running config samozřejmě smazána být nesmí.
- **LOCK** a **UNLONCK** buď povolí, nebo zakážou správci přístup ke konfiguraci. Jakmile jeden správce drží soubor zamčený, nikdo jiný k němu nemá přístup.

3.3.3. CLI

Command-Line Interface (dále jen CLI) je uživatelské rozhraní pro operační systém, který umožňuje komunikaci mezi správcem sítě a zařízením. Správce říká zařízení, co má dělat zadáváním určitých příkazů do konkrétního řádku tohoto rozhraní a zařízení mu skrze toto rozhraní odpoví [10]. Umožňuje tedy komunikaci pouze pomocí psaní příkazů a pomocí nich systém také ovládat.

CLI není protokol jako takový a Alexander Clemm to podporuje následujícím tvrzením:

„Přesně řečeno, CLI není ani zdaleka management protokol. Je to rozhraní příkazového řádku vymyšleno pro lidi, kteří komunikují se zařízením přímo, ne skrze aplikaci, která schová všechny detaily o tom, jak probíhá samotná komunikace se zařízením.„

Dále ale také uvádí, že ne všechny funkce jsou podporovány předchozími protokoly a proto aplikacím někdy nezbývá nic jiného, než využít právě CLI. U menších sítí se dokonce může stát, že komunikace mezi zařízením a správcem sítě bude probíhat jen a pouze skrze toto rozhraní. Z tohoto důvodu je třeba si zde rozhraní CLI také vysvětlit.

Neexistuje žádný standart pro CLI, což umožnilo vznik několika verzím tohoto rozhraní, které se zpravidla liší pouze mezi operačními systémy. Jelikož je tato práce zaměřena na práci s operačním systémem IOs od společnosti CISCO, bude zde blíže popsána práce s touto verzí.

Cisco IOS využívá pro komunikaci s uživatelem právě rozhraní CLI, ve kterém pomocí příkazů, které zadáváme řádek po řádku, smíme ovládat celé zařízení. Tento CLI má několik příkazových módů, které jsou hierarchicky uspořádány, a v každém módu je možné používat pouze určitou sadu příkazů. Mezi jednotlivými módy je možné se libovolně přepínat a ovládat tak různé aspekty zařízení v závislosti na příkazech, které jsou dostupné pro právě aktivní příkazový mód. Podrobnější informace o jednotlivých módech lze nalézt v [11], jejich přístupových metodách a změn, které můžeme použít v daném módu, nalezneme v tabulce 1.

Příkazový mód	Přístupová metoda	Nápověda	Využití
User EXEC	Defaultní po přihlášení	Router>	Změna nastavení terminálu Základní testy(ping, atd.) Zobrazení statusu zařízení
Privileg EXEC	V user EXEC zadáme příkaz enable	Router#	Příkazy show a debug Kopírování image do zařízení Restart zařízení Správa konfiguračních složek a souborů systému
Global configuration	V privileg EXEC zadáme příkaz configure terminal	Router (config)#	Konfigurace zařízení
Interface configuration	v global configuration zadáme příkaz interface	Router (config-if) #	Konfigurace konkrétního rozhraní
Line Configuration	V global configuration zadáme příkaz line vty nebo line console	Router (config-line)#	Konfigurace konkrétní linky zařízení
ROM monitor	V Privileg EXEC zadáme příkaz reload a během prvních 60s načítání systému zmáčkneme tlačítko break	Rommon # >	Načte se jako defaultní mód, pokud nenajde platnou image Může se zde obnovit heslo

Tabulka 1(převzato z [11]

Pokud si například nemůžeme vzpomenout, jak přesně příkaz zní, nebo si chceme urychlit práci, lze využít pomocné příkazy, které nám poskytnou nápovědu všech příkazů pro daný příkazový mód, seznam všech argumentů souvisejících s příkazem, který si nepamatujeme celý a další. Detailní popis těchto pomocných příkazů nalezneme v tabulce 2.

Příkaz	Význam
Help	Zobrazí stručný popis služby **help** v jakémkoliv příkazovém módu
?	Zobrazí seznam všech příkazů dostupných v aktuálním příkazovém módu
příkaz**?**	Zobrazí seznam všech příkazů obsahující námi zadaný textový řetězec (mezi řetězcem a otazníkem nesmí být mezera)
příkaz**<Tab>**	Doplní očekávaný příkaz podle nedokončeného příkazu, který jsme napsali (mezi příkazem a <Tab> nesmí být mezera)
příkaz **?**	Vypíše list všech klíčových slov, argumentů, nebo obojích souvisejících s příkazem (mezera mezi příkazem a otazníkem)
klíčové slovo **?**	Zobrazí seznam všech argumentů souvisejících s daným klíčovým slovem (mezera mezi klíčovým slovem a otazníkem)

Tabulka 2 (převzato z [11])

4. NÁSTROJE PRO AUTOMATIZACI

V této kapitole se seznámíme s jednotlivými teoretickými pojmy, které bude potřeba znát k pochopení automatizace a její následné implementace ve vybraných úlohách.

Nejprve musíme vědět, jak probíhá běžná komunikace mezi správcem sítě a zařízením, poté potřebujeme znát síťové operační systémy, se kterými se můžeme v praxi setkat a jednotlivé možnosti automatizování úloh na každém z nich.

4.1. JUNOS

Junos OS [12] je operační systém od firmy Juniper, který integruje směrování, přepínání a bezpečnost. Je postaven na Unixovém jádře FreeBSD. Dále kromě klasické CLI podporuje psaní skriptů a nabízí vlastní SDK umožňující vývoj aplikací pro zařízení s OS Junos .

Jelikož je založen na Unixovém jádře, uživatel se může dostat až k jádru samotnému a provádět na něm klasické Unixové příkazy a upravit si vše podle vlastních potřeb a tím nabízí široké možnosti personalizace.

4.1.1. SKRIPTY V OS JUNOS

Skripty operačního systému Junos jsou interpretované programy, což znamená, že není potřeba je překládat do kódu strojového jako například programy psané v jazycích C a Java.

Dělí se do dvou hlavních skupin podle jejich umístění vůči zařízení, na kterém se skripty realizují. Jedná se o rozdělení do *on-box* a *off-box* automatizací. Jak napovídá sám název, tato rozdělení určují, zda je skript uložen v paměti daného zařízení, nebo jinde a na požadovaném zařízení je pouze spouštěn. U *off-box* automatizace se skript nachází na jiném zařízení, než na kterém je spouštěn a pro vzdálený přístup a spouštění jednotlivých příkazů využívá určitý protokol (v případě OS Junos se jedná o protokol NETCONF). Hlavní výhodou tohoto uložení skriptu je to, že skript lze napsat pouze jednou a pak ho spouštět z více síťových prvků. *On-box* automatizace je v tomto ohledu daleko jednodušší, neboť je skript uložen fyzicky na daném zařízení a není třeba řešit komunikaci mezi skriptem a

zařízením. On-box automatizace se dále dělí do 3 základních skupin podle jejich vlastního využití[13]. Jsou jimi:

- **Operation scripts** se vykonávají v operačním módu zařízení k vytvoření náročnějších příkazů, nebo ke změně konfigurace zařízení. Provedou se pokaždé po jejich zavolání, mohou být spuštěny přímo z příkazového řádku *CLI*, nebo *Event skriptem* (viz níže).

  Ukázkovým příkladem Operation skriptů jsou různé varianty a na míru upravené příkazy *show*, které shromažďují data z několika příkazů show, a poté na výstupu zobrazí pouze požadovanou informaci.

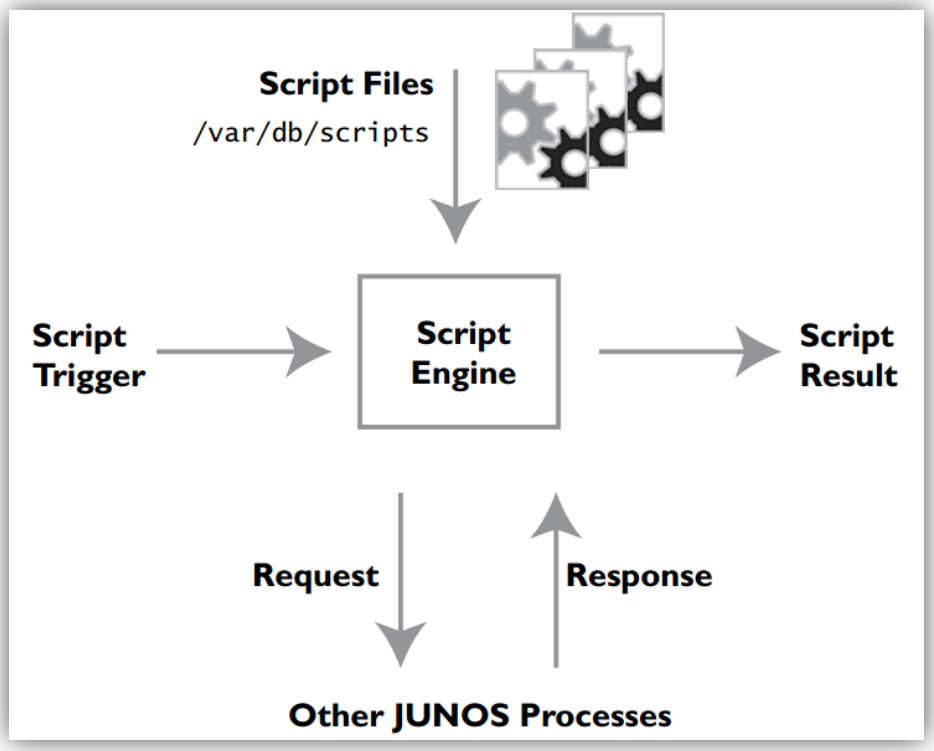
  Dalším příkladem by bylo využití automatizované konfigurace sítě, kde skripty mění konfiguraci samy vzhledem k informacím, které jsou jim předány pomocí argumentů v CLI, nebo například výstupy příkazů show.

- **Event scripts** jsou spuštěny automaticky po výskytu určité události (např.: zpráva SYSLOG, SNMP trap nebo pouze nějaký časovač spouštějící skripty v určitých intervalech). Můžeme tak sbírat informace, které bychom pak mohli využít při řešení určitých problémů v síti, nebo *Event skripty* mohou reagovat samy na vzniklou událost a přímo měnit konfiguraci sítě v závislosti na vyskytlých událostech či na čase.

  Jako příklad typického Event skriptu by mohl být skript pro vytváření náhradních cest. Například pokud rozhraní některého směrovače spadne, spustí se *Operation skript*, který přesměruje všechen provoz skrze jiné rozhraní a poté uloží SYSLOG zprávu o změně cesty.
- **Commit scripts** jsou v operačním systému Junos využívány během commit procesů, které nahrazují *running* konfiguraci tzv. konfigurací *candidate*. Každý commit skript je tedy nejdříve spuštěn v konfiguraci candidate a teprve pokud vše proběhne v souvislosti s danou změnou v pořádku, je running konfigurace nahrazena candidate konfigurací, která obsahuje změnu daného commit procesu. Pokud se však během aplikování změn do konfigurace candidate vyskytne nějaká chyba, je zaznamenána, proces výměny konfigurací nebude proveden a poté je možné také vygenerovat chybovou hlášku, nebo například zapsat chybu do SYSLOGu.

A jak vlastně takové spouštění skriptů funguje? Automatizační skripty neobsahují nic jiného, než určitou posloupnost kroků, které se musí dodržet při vykonávání daného skriptu. Junos smí spouštět pouze skripty, které jsou zahrnuté jako součást konfigurace zařízení. Do této konfigurace je dovoleno vkládat skripty jen uživatelům s patřičným oprávněním.

Obrázek 3 ukazuje základní postup při zpracování určitého skriptu. Junos ukládá všechny skripty do předem nastavené složky s cestou */var/db/scripts/*, kde čekají na jejich spuštění, ke kterému dojde po výskytu jemu přiděleného spouštěče. Např. commit skripty jsou spouštěny příkazem commit v konfiguračním módu. Jakmile se skript spustí, Skript Engine ho začne zpracovávat řádek po řádku a vykonávat jednotlivé akce jednu po druhé. Tyto akce mohou také obsahovat požadavek na další procesy Junos. Jakmile Script Engine dokončí zpracování skriptu, odešle jeho výsledek na výstup definovaný ve skriptu.



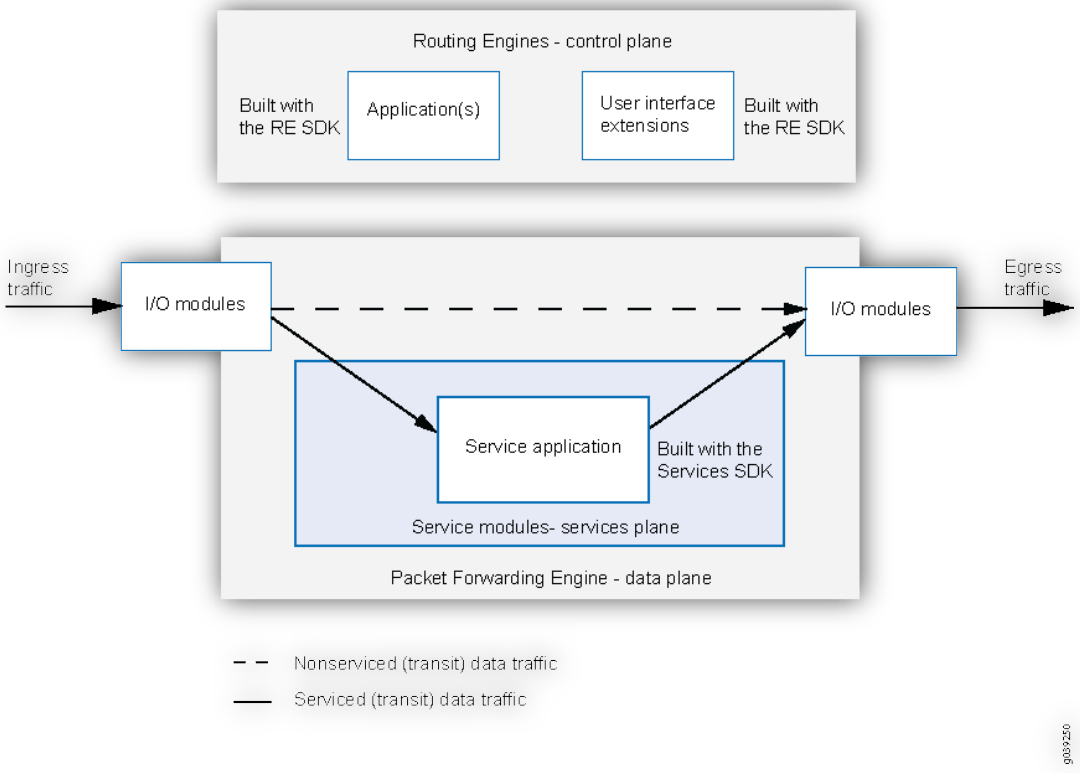
Obrázek 3 - Zpracování skriptu v OS Junos (převzato z [13])

Skripty pro Junos mohou být napsány ve dvou jazycích. Jedná se o XSLT (nebo také SLAX.XSLT) a druhým jazykem je XML. XSLT je zkratka pro „eXtensible Stylesheet Language Transormation“ a byl vyvinut pro převod jednoho dokumentu

XML do druhého. SLAX znamená „Stylesheet Language Alternative syntaX“ a byl vytvořen vývojáři Juniperu, aby poskytl uživatelům příjemnější a intuitivnější možnost psaní skriptů. Nabízí lépe čitelnou syntaxi a je bližší uživatelům, kteří již někdy pracovali v programovacích jazycích jako C a Pearl.

4.1.2. JUNOS SDK

Junos SDK (Junos Software Development Kit) je nástroj umožňující uživatelům OS Junos vytvářet si vlastní aplikace, které si mohou uživatelé pomocí poskytovaných API ušít na míru přesně pro své vlastní potřeby.



Obrázek 4 - Architektura Junos (převzato z [14])

Na Obrázek 4 vidíme jednotlivé úrovně architektury Junos [14] a blíže si jednotlivé úrovně a jejich smysl vysvětlíme níže. Jsou jimi:

- **Řídící úroveň** je logický element OS Junos, který, jak naznačuje sám název, řídí běh samotného zařízení a zbylých dvou vrstev. Pro řídící úroveň je určen tzv. *Routing Engine*. Ten může být jak externím fyzickým modulem v zařízení, tak implementován přímo v daném síťovém zařízení. Řídící úroveň má globální pohled na software a

hardware daného zařízení. Zobrazuje uživatelské rozhraní a spravuje vlastnosti systému Junos.

- **Datová úroveň** je element spojující několik aspektů šasi zařízení a jeho modulů. Jeho rolí je předávat pakety dál tak, aby byly ze zařízení odeslány podle směrovací tabulky, ale především seřazeny v pořadí, určeném funkcemi jako QoS (Quality of Service), které jsou ovládány v řídící úrovni. Datovou úroveň také lze nazývat „Packet Forwarding Enginge“, neboť jejím hlavním účelem je předávat pakety se zaměřením na rychlé směrování vzhledem k jeho hardwarovým možnostem.
- **Úroveň služeb** může být vysvětlena jako rozšíření úrovně datové. Běžně se o zpracování paketů stará vrstva datová a vrstva služeb běží na zásuvných modulech. Úroveň služeb tak spojuje všechny možné služby poskytované jednotlivými moduly, které jsou v zařízení zasunuty. Poskytuje služby spojené s přenosem paketů. Hlavní služby související se správou paketů jsou monitorování a transformace paketů, mezi transformaci paketů můžeme zařadit například úpravu, zahození, zdržení anebo vytvoření nového paketu. U sledování paketů se jednotlivé pakety kopírují. Tato kopie je poté přeposlána dále, kde je zpracována, tudíž nedochází k žádnému omezení datového přenosu.

Aplikace napsané v Junos SDK mohou fungovat na vrstvě řídící či vrstvě datové, podle toho, jakou činnost mají vykonávat. Jak bylo vysvětleno výše, úroveň služeb nemusí být v daném zařízení poskytována, neboť závisí na přítomnosti zásuvných modulů v daném síťovém zařízení.

4.2. RouterOS

Router OS [15] je produktem firmy MikroTik, postavený na Linuxovém jádře. Tento OS také integruje všechny potřebné vlastnosti ke směrování, ale oproti předchozím systémům nabízí výhodu instalace i na běžný PC, který se poté bude chovat jako plnohodnotný směrovač s podporou VPN, hotspot, bezdrátové přístupové body a tak dále.

Pro přepínače vyvinula firma MikroTik podobný OS s názvem SwitchOS,

který obsahuje vše potřebné pro práci na své vrstvě a rovněž ho lze nainstalovat na PC a změnit ho tak na přepínač.

4.2.1. SKRIPTOVÁNÍ V ROUTEROS

Ke skriptování v RouterOS se využívá tzv. *RouterOS scripting language*. Jedná se o skriptovací jazyk, který je již vložen do zařízení s RouterOS . Jednotlivé skripty můžeme psát jako je zvykem přímo do CLI (v RouterOS nazýváno konzole), nebo mohou být uloženy v tzv. skript repository [16]. Zdroj také dále uvádí, že skripty uložené v skript repository lze spustit následujícími způsoby:

- **Událostí -** Skripty jsou spouštěny automaticky po výskytu dané události
- **Jiným skriptem** - Pokud nějakou část skriptu vykonáváme častěji, je možné si tu část vytáhnout, napsat ji jako samostatný skript a pak v potřebnou chvíli skript zavolat z ostatních skriptů. Je možné si tak ušetřit psaní kódu u často opakovaných úkonů.
- **Manuálně** - V konzoli je možné spustit příkaz run následovaný názvem požadovaného skriptu, nebo jeho ID číslem.

Jednotlivé události, na které můžeme v RouterOS reagovat, jsou podle zdroje[16] následující:

- **Scheduler** - Jedná se o plánovač, který umožňuje spouštět skript buď v určitý čas, nebo po daném časovém intervalu. Je možné zde napsanému skriptu přiřadit spuštění po vykonání nějakého jiného skriptu, spustit skript v určitý den v roce, nebo dokonce přidat počet spuštění skriptu.
- **Netwatch** - Monitoruje stav hostů připojených v síti pomocí zasílání pingu na předem určené adresy. Do každého řádku netwatch tabulky je možné přiřadit IP adresu, interval pingu a určitý skript. Hlavní výhodou netwatch je možnost spuštění libovolných skriptů po změně stavu připojení libovolného hosta.
- **VRRP** - Je zkratka pro Virtual Router Redundancy Protocol. Je využíván ve velkých sítích pro zjišťování nedosažitelných směrovačů. Běžně je v RouterOS využíván Neighbour Discovery Protocol , ale tomu to ve větších sítích může trvat až desítky vteřin, než zjistí přítomnost nedosažitelného směrovače. VRRP toho dosáhne pomocí rozdělení celé sítě do malých

virtuálních podsítí a pokud zjistí v síti přítomnost nedosažitelného směrovače, umožňuje spustit skript, který může chybu v síti opravit.

Jednotlivé ukázky skriptů a syntaxe jazyka RouterOS scripting language lze nalézt na [16].

4.2.2. API PRO ROUTEROS

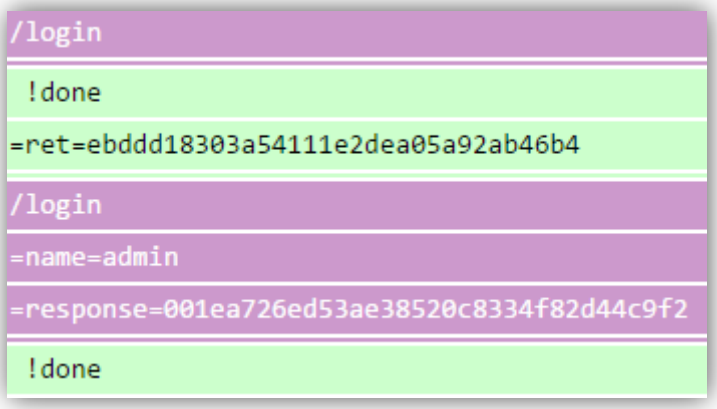
API umožňuje uživatelům vytvořit vlastní softwarové řešení pro komunikaci s RouterOS umožňující sbírat informace, upravovat konfiguraci a spravovat směrovač. API má totožnou syntaxi s příkazovým řádkem CLI, aby mohl zajistit jednoduchou a bezchybnou komunikaci mezi zařízením a uživatelským softwarem.

Jedná se vlastně o jednoduchý protokol na základě komunikace pomocí vět, skládajících se ze slov. První slovo každé věty zaslané směrovači je příkaz, který je následovaný dalšími slovy bez určeného pořadí. Každá věta končí slovem nulové délky. Pokud směrovač obdrží celou větu (začíná příkazem a končí slovem nulové délky) příkaz se vyhodnotí, provede a následně se zformuluje odpověď, která je pak zaslána zpět.

Každé slovo je zakódováno následovně *<<zakódovaná délka slova><slovo>>* samotné slovo se tak dělí do 5 skupin, které jsou následující [17]:

- **Slovo příkazu** - Jak již bylo řečeno, každá věta musí začínat právě slovem příkazu začínajícího znakem „/“ a následován slovem příkazu, který je shodný s příkazem v CLI, kde je lomítko nahrazeno mezerou.
- **Slovo atributu** - Každý příkaz má svůj odpovídající seznam atributů. Každé slovo atributu začíná znakem „=“, za kterým je samotný název atributu, poté další znak „=“ a na závěr odpovídající hodnota daného atributu. Atributů může být pro určitý příkaz víc a nezáleží na jejich pořadí.
- **Slovo atributu API** - Toto slovo začíná znakem „.“ následuje název atributu, poté znak „=“ a hodnota daného atributu. V tuto chvíli jediným API atributem je tzv. tag. Pokud nějaká věta obsahuje tento atribut, odpověď na tuto větu bude mít stejnou hodnotu tohoto atributu. Rozlišuje se tak, které větě odpověď náleží.

- **Slovo dotazu** - Každý dotaz začíná znakem „?“ následovaný názvem atributu, poté se píše opět „=“ s hodnotou atributu hned za ním. Jedná se o formu filtrace výsledků, kde například můžeme z výpisu všech rozhraní vypsat pouze rozhraní Ethernet (jednotlivých dotazů může být v jedné větě i více a záleží na jejich pořadí).
- **Slovo odpovědi** - Je odesílán směrovačem pouze jako odpověď na danou větu. Začíná znakem „!“ a API může obdržet následující odpovědi:
    - !done – po úspěšném ukončení příkazu zaslanou větou
    - !trap – pokud se ve vykonávání příkazu stane nějaká chyba
    - !re – pokud věta vyžaduje nějaká data a směrovač je zasílá zpět
    - !fatal – pokud neexistuje spojení se směrovačem



Obrázek 5: Komunikace v RouterOS API (převzato z [17])

4.3. IOs

Vzhledem k tomu, že se v této práci budeme věnovat platformám právě s operačním systémem IOs, bude rozebrán podrobněji než předchozí dva systémy a budou zde vysvětleny i základy práce s ním. Cisco IOS (Internetwork Operating System) je momentálně asi nejznámější a jeden z nejvíce využívaných síťových operačních systémů v praxi. Integruje v sobě směrování, přepínání i telekomunikaci [18].

Pro práci s tímto systémem využíváme již zmíněný Command Line Interface (CLI), ve kterém pomocí několika příkazů můžeme ovládat celé zařízení. Pro tuto platformu jsou k automatizaci dostupné následující technologie.

4.3.1. TCL

Tool Command Language (TCL) [19] byl vytvořen už v roce 1980 Johnem K. Ousterhoutem. Jedná se o interpretovaný programovací jazyk, což znamená, že se zdrojový kód nepřekládá do kódu strojového (není potřeba kompilovat).

Hlavními výhodami interpretovaného jazyka oproti kompilovanému je možnost jednoduchých úprav v kódu během vývoje, možnost přenosu na jinou platformu a protože je psaný v běžném textu, každý uživatel může kód vidět tak jak byl napsán a tak ho zkopírovat a použít jako vlastní, což dosti komplikuje udržení samotného kódu v tajnosti před ostatními. Další nevýhodou je nutnost mít interpret do daného jazyka a program je oproti kompilovanému jazyku pomalejší.

Ke spuštění skriptů bylo vyvinuto a vloženo několik metod přímo do Cisco IOS. Můžeme použít TCL Shell (dále jen TCLsh), který umožňuje zadávat TCL příkazy řádek po řádku přímo do CLI, nebo může číst TCL příkazy ze souboru. TCLsh čte příkazy až do té doby, než nedostane na vstupu příkaz k jeho ukončení, nebo v souboru nedojde na jeho konec [20]. Další možností je vytvořit skript mimo IOS na svém PC, ten poté jej nahrát do paměti a spustit ho z paměti.

4.3.2. EEM

Embedded Event Manager (EEM) [21] je velice mocný nástroj implementovaný v Cisco IOS, který skrze detektory událostí sledující jak hardware, tak software a v případě vzniku požadované události umožňuje spuštění TCL skriptu nebo appletu přiřazenému právě vzniklé události.

EEM politika [22] je entita definující událost a akci, která po výskytu této události bude provedena. Existují dva typy EEM politiky a těmi jsou: *applety* jsou definované v rámci CLI konfigurace a *skripty*, které jsou vytvořeny pomocí TCL mimo síťový prvek a následně do něj nahrány.

Applety jsou bohužel svou funkčností lehce omezené oproti skriptům v TCL. Umožňují pouze jednu akci, která může nastat po výskytu dané události. Skripty naopak nabízí možnost definovat více možných akcí pro danou událost, a pomocí podmínek po jejím výskytu určit, která akce bude provedena. Applety se tedy kvůli své větší jednoduchosti (není potřeba nahrávat programy zvlášť z externího zdroje

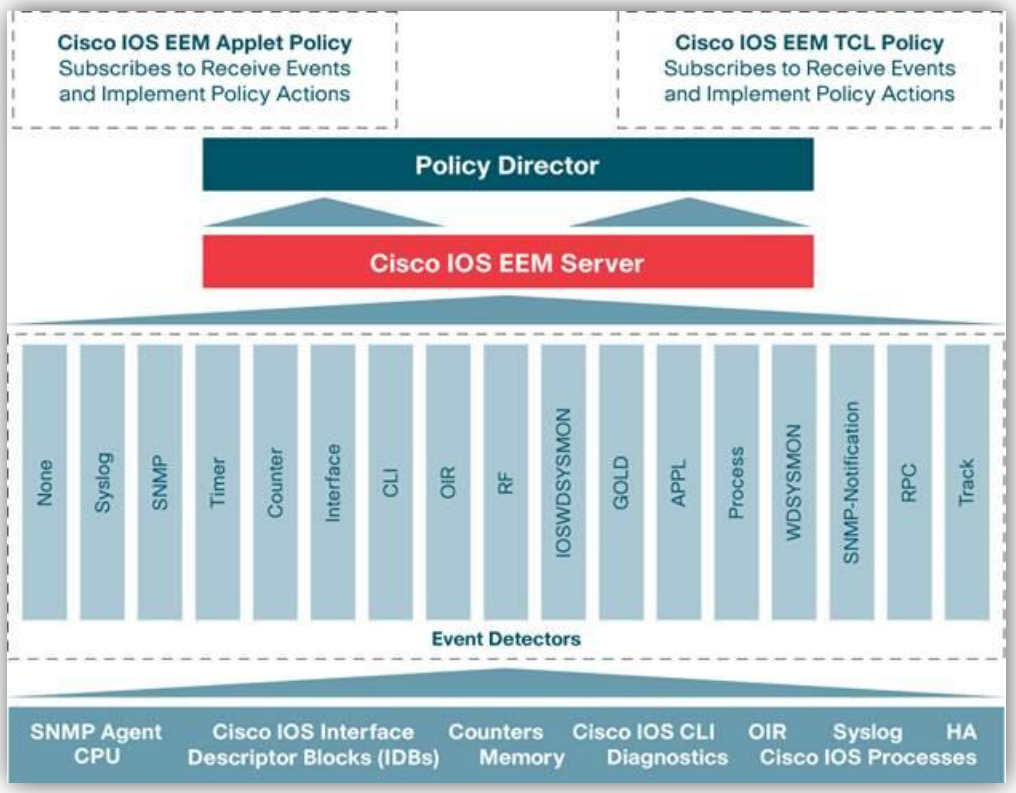
do paměti zařízení) použijí v politikách, kde nám jejich omezenost nevadí, nebo předpokládáme využití jen na jednom zařízení. Na druhou stranu, využití skriptovacího jazyka TCL nám umožní daleko větší možnosti a flexibilitu, neboť stačí daný skript napsat pouze jednou a poté už se pouze nahraje do flash paměti i několika požadovaných zařízení a spustí se odtamtud.

EEM politika se zpracovává ve 2 fázích. A těmi podle [23]jsou:

- **Detekce události -** Musíme nakonfigurovat Cisco IOS EEM tak, aby reagoval na detektory událostí využitím appletů nebo skriptů. Každý detektor události tak může spustit určitý skript na základě zjištěné události. Například pokud spadne nějaké rozhraní, je možné nastavit záložní cestu a po opětovném spuštění daného rozhraní lze nastavit původní cestu jako výchozí.
- **Implementace požadované politiky** - Po detekování dané události detektorem se provede akce přiřazená vyskytnuté události, jako např. zaslání emailu, nastavení náhradní cesty, atd. Seznam akcí, které je možné využít a jeho dostupnost, vzhledem k verzi námi používaného IOSu nalezneme v tabulce 4.

Jak zobrazuje Obrázek 6, detektory událostí (Event Detectors) monitorují systém a kontrolují výskyt určitých událostí v něm. Jakmile se událost objeví, detektor ji zachytí a odešle informaci o výskytu této události na EEM server, ten poté ve spolupráci s odběrateli událostí zjistí, jestli má uvedená událost přiřazenou nějakou politiku (skript nebo applet). Pokud je nalezena politika, která je přiřazená k události nahlášené jejím detektorem EEM serveru, IOS spustí akci přiřazenou vyskytnuté události.

EEM server má tedy na starost především komunikaci mezi odběrateli událostí (politikami) a detektory událostí, a to prostřednictvím registrace všech aktivních skriptů a appletů ve své paměti, kde se po výskytu nějaké události hledá politika, která je jí přiřazená.

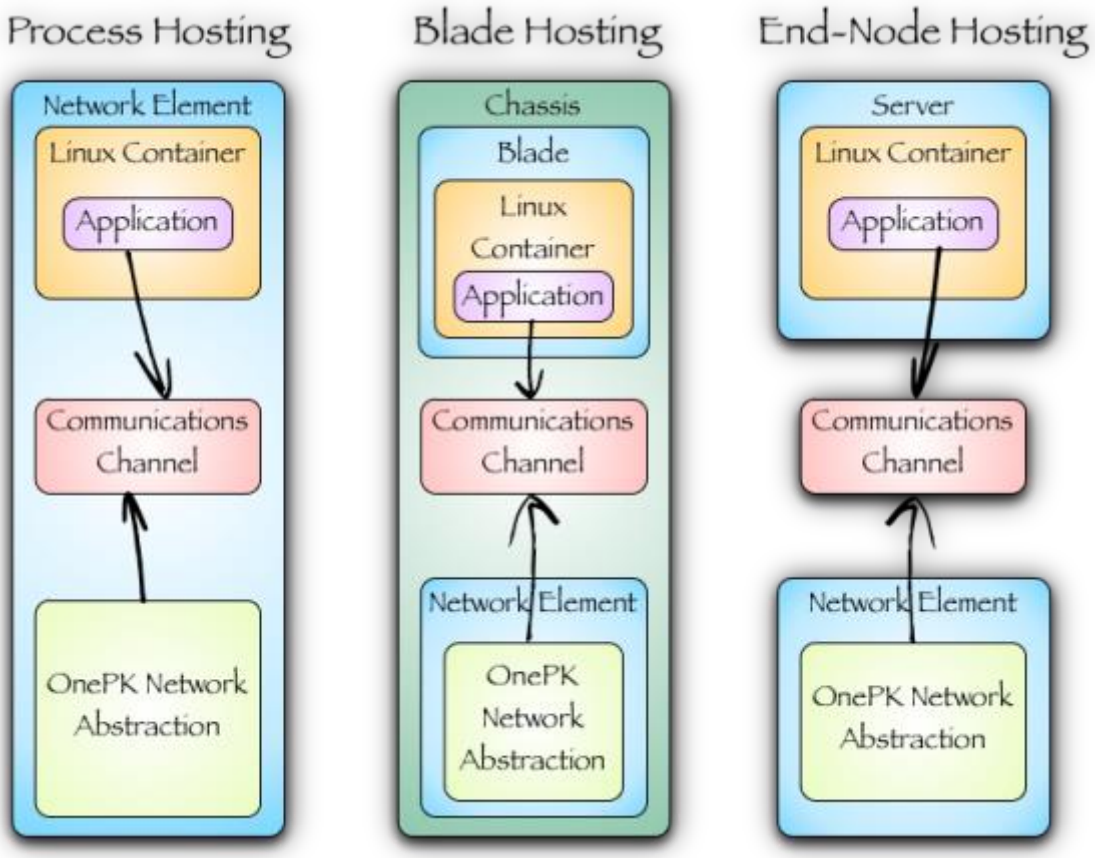


Obrázek 6 - Architektura Cisco IOS EEM (převzato z [23])

4.3.3. ONEPK

Jednou z dalších možností automatizace mnoha úkonů spojených se síťovým provozem je vytvoření vlastní desktopové aplikace, která nám umožní jednoduše komunikovat s požadovaným zařízením skrze vlastnoručně vytvořené uživatelské rozhraní. Dnes již existuje mnoho nástrojů umožňujících vývoj aplikací pro komunikaci se síťovými prvky. Mezi hlavní patří několik knihoven programovacích jazyků, jako *Boost.Asio* pro programovací jazyk *C++*, nebo software typu *one Platform Kit (onePK)* vytvořené firmou Cisco, která nám dovoluje nejen si vytvořit program podle vlastních potřeb, ale navíc si můžeme vybrat, v jakém programovacím jazyce ho chceme vytvořit [24]. Po instalaci je možnost si vybrat z jazyků *C, Java* a *Python*, ovšem je možné si stáhnout rozšiřující balíčky s dalšími jazyky. Proto potřebujeme znát velice dobře nejen komunikační protokoly a funkčnost sítě, ale také vybraný programovací jazyk, ve kterém budeme daný program vyvíjet.

OnePK je momentálně pod kontrolovanou distribucí, takže pokud s ním chcete pracovat, je třeba se registrovat a uvést z jakých důvodů s ním chcete pracovat a následně Cisco rozhodne, jestli vám bude onePK uvolněn k práci.



Obrázek 7 - Komunikace mezi aplikací a zařízením (převzato z [24])

U OnePK je velice důležitý tzv. *Deployment model* (model nasazení). Určitý model si vybereme podle našich dostupných prostředků a náročnosti aplikace. Modely jsou následující:

- **Process hosting** - Zde je aplikace spuštěna přímo na síťovém zařízení v tzv. *linuxovém kontejneru* (Linux Container), který je oddělený od samotného systému zařízení. Nicméně ale aplikace stále využívá prostředků zařízení.
- **Blade hosting** - Aplikace běží na tzv. *blade serveru,* to je modul serveru (anglicky tzv. *blade*), uložený ve stejném šasi jako původní server a disponuje vlastními výpočetními prostředky (CPU, RAM i flash), takže

aplikace běží blízko našemu síťovému zařízení, avšak ho výpočetně nezatěžuje jako předchozí model.

- **End-point hosting** - Tento typ umístění je využit, pokud je potřeba pro aplikaci velký výpočetní výkon. Aplikace je umístěná na PC nebo na externím serveru a mezi síťovým zařízením a aplikací probíhá zabezpečená komunikace

5. IMPLEMENTACE VYBRANÝCH ÚLOH

V této části si ukážeme vývoj našich aplikací v praxi. Hlavní zaměření bude na programování aplikace pomocí nástroje OnePK ve virtuálním prostředí All-In-One-VM. Toto prostředí je zcela zdarma, nicméně má prozatím omezenou distribuci, takže je potřeba se registrovat na stránkách DevNetu [25], kde vám bude po registraci umožněno si stáhnout soubor s názvem *„all-in-one-VM-1.2.1-194.ova“, který* má necelé 3GB a obsahuje vše potřebné pro vývoj aplikace.

5.1. ALL IN ONE VIRTUAL MACHINE

Jedná se o virtuální stroj operačního systému Linux Ubuntu, který hned po stáhnutí obsahuje pro nás vše potřebné k tomu, abychom se mohli pustit do vývoje vlastního programu. Nalezneme zde i potřebné tutoriály, vzorové aplikace, dokumentaci i SDK a to pro následující jazyky

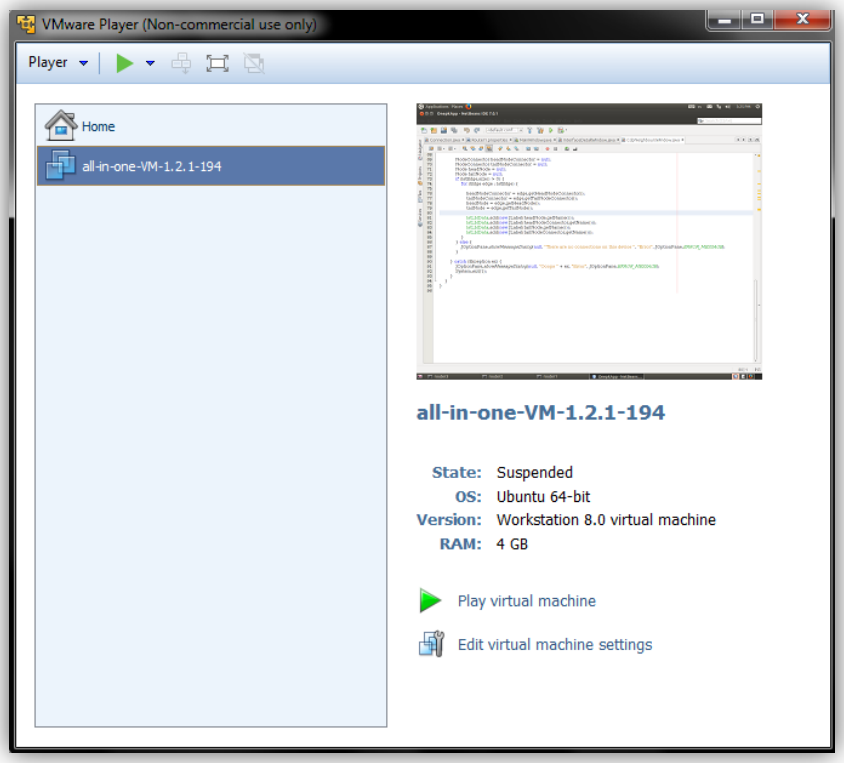
- **Java**
- **C**
- **Python**

Eclipse je dokonce přednastavený tak, že po spuštění má otevřené všechny dostupné java tutoriály připravené ke spuštění. Pro otevření souboru *„all-in-one-VM-1.2.1-194.ova“* potřebujeme podle [26] některý z následujících virtualizačních programů:

- VMWare ESXi 5 nebo novější
- VMWare Workstation 9 nebo novější
- VMWare Fusion 5 nebo novější
- Oracle VirtualBox 4.2 nebo novější

Jelikož tyto programy jsou zpoplatněny, využijeme VMWare Player, který je pro nekomerční účely zcela zdarma. Pro otevření souboru musíme spustit VMWare Player a tam klikneme na „*Open a Virtual Machine“*, kde následně vybereme cestu k námi staženému souboru, nastavení pro virtuální stroj můžeme nechat

v defaultních hodnotách a poté už se nám objeví náš virtuální stroj v levém menu, který bude spustitelný, jak je vidět na Obrázek 8.



Obrázek 8 - VMWare Player

Po spuštění bude virtuální OS vyžadovat zadání přihlašovacího jména a hesla, které jsou nastaveny po prvním spuštění na následující:

- Uživatel: cisco
- Heslo: cisco123

Následně budete systémem vyzváni k zadání nového hesla pro uživatele cisco a k souhlasu s licenčními podmínkami. Po odsouhlasení licenčních podmínek budete vyzváni k zadání uživatelského jména a hesla, které budou použity v Cisco IOS pro komunikaci mezi OnePK aplikacemi a NDE (*Network Device Emulator* je virtuální síťové zařízení umožňující testování našich aplikací).

5.1.1. PRÁCE S VIRTUÁLNÍ SÍTÍ A NDE

Virtuální zařízení nám ze základu nabízí topologii zvanou *3-node*, kterou můžeme využít pro testování. Ke spuštění této virtuální topologie stačí dvakrát

kliknout na ikonu s názvem *Start 3node*, která je umístěna na ploše viz Obrázek 9. Jak můžeme také vidět na obrázku a jak již bylo řečeno, vše potřebné je ve virtuálním PC připraveno na ploše k použití, ikony pro ovládání virtuální sítě, potřebná dokumentace i Eclipse pro psaní vlastního programu.

Obrázek 9 - Plocha All-in-one VM

Po spuštění připravené virtuální topologie musíme chvíli počkat, než se nám spustí CLI jednotlivých směrovačů. Tato virtuální zařízení se chovají stejně jako reálná, nicméně jejich konfiguraci načítají z konfiguračních souborů s příponou *„.con“*, což znamená, že po zadání příkazů jako *„copy running-config startup-config“* se neudrží v konfiguraci zařízení po jeho restartu. Pokud tedy chceme uchovat některé nastavení, musíme ho zadat přímo do konfiguračního souboru daného směrovače, který se v případě této topologie nachází na cestě: *„/home/cisco/vmcloud-example-networks/3node“*

Zde lze nalézt také vše potřebné k vytvoření virtuální topologie, celou složku se všemi soubory je možné nalézt na přiloženém CD. Zde také nalezneme složku *„3node.virl“*, což je XML soubor obsahující všechny potřebné informace o virtuální topologii, kterou budeme za pomoci těchto informací tvořit. Jak vidíme v příloze I., soubor začíná tagem *<topology>* a končí tagem *</topology>*. Všechny informace jsou pro maximální přehlednost uspořádány hierarchicky a jejich uspořádání je

následující:

- **Topology**
  - **Node**
    - **Extensions**
    - **Interface**
  - **Connection**

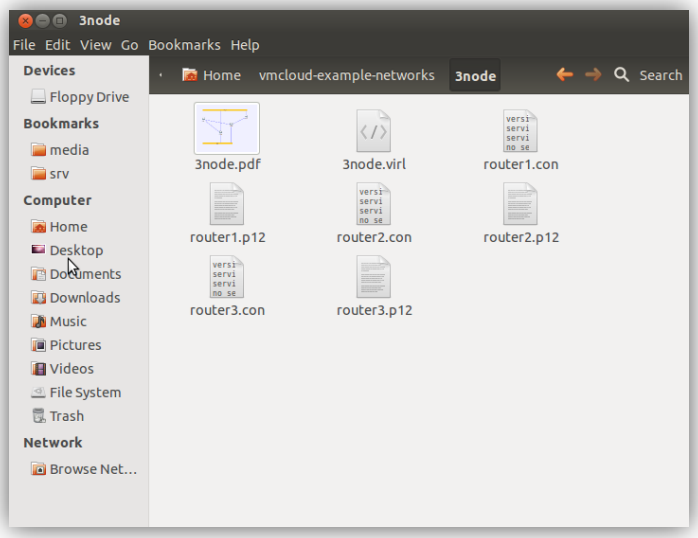
Tag *node* definuje síťový element a odkazuje na všechny potřebné soubory. Mezi tagy *Extensions* se nachází odkaz na konfigurační soubor daného elementu. Ukázka souboru takového konfiguračního souboru je v příloze II. Interface nám určí rozhraní, která chceme v topologii používat a jejich název. Atributy tagu jsou: *type, vmImage, name, subtype a location*. *VmImage* odkazuje na image virtuálního operačního systému, který bude zařízení implementovat, *name* definuje název zařízení a poslední zajímavý atribut je *type* určující typ zařízení, který přijímá následující hodnoty:

- **SIMPLE** – pro nedistribuční zařízení
- **DETAILED** – distribuční směrovač (nicméně je tato možnost zatím nedostupná)
- **ASSET** – použijeme, pokud se jedná o reálné fyzické zařízení a jeho jméno (*name)* definuje adresu NIC (Network Internet Card) konkrétního zařízení

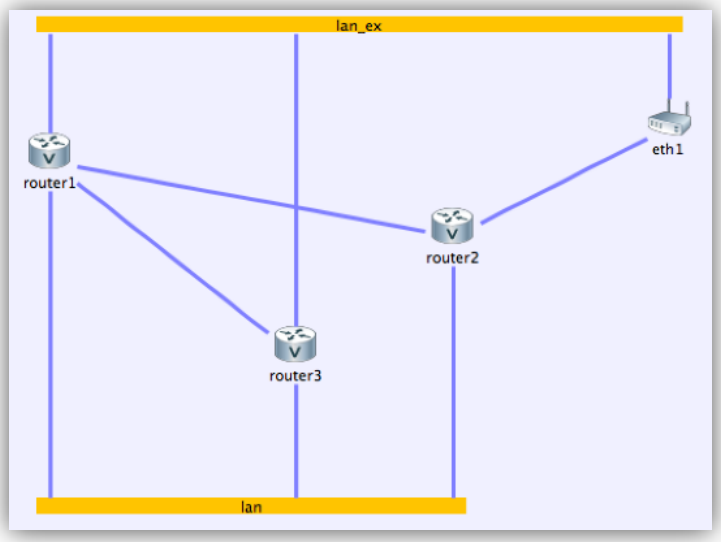
Pomocí *connection* říkáme, mezi kterými rozhraními jednotlivých elementů chceme vytvořit spojení. Jsou zde důležité dva atributy a těmi jsou *src* (zdroj) a *dst* (cíl) spojení. Rozhraní se zadává pomocí cesty k němu v XML souboru tak, jak jsme ho zadali. V případě prvního rozhraní prvního směrovače by cesta zadaná do atributu byla: „/topology/node[1]/interface[1]“

Je tedy velice jednoduché si odebrat zařízení. Stačí smazat řádky o jeho připojeních a pak odebrat ze souboru celý node, kterým ho definujeme. Pokud ovšem chceme zařízení přidat, je nejprve potřeba vytvořit konfigurační soubor ve

složce a poté teprve můžeme přidat *node* o tomto zařízení a jeho připojení do XML souboru 3node.virl. Celou topologii 3node a její jednotlivé konfigurační soubory si můžeme prohlédnout na Obrázek 10 a Obrázek 11.



Obrázek 10 - Konfigurační soubory topologie 3node

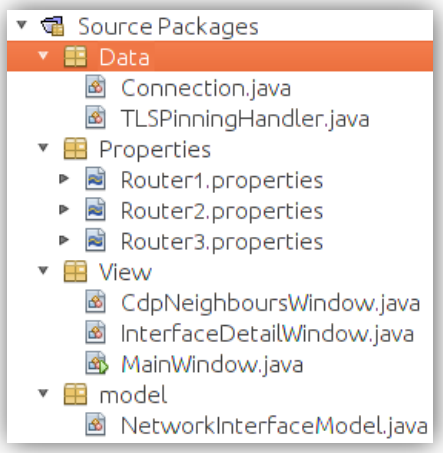


Obrázek 11 - Topologie 3node

5.2. ONEPK APLIKACE

Jakmile máme připravenou a spuštěnou virtuální topologii, můžeme se pustit do vývoje vlastní aplikace. Jak bylo již řečeno dříve, máme na výběr celkem ze tří programovacích jazyků. V této práci byl zvolen programovací jazyk Java a to

z důvodu znalosti programovacího jazyka, ale vše záleží pouze na osobní preferenci programátora. Samotná aplikace je rozdělena do několika tříd (Obrázek 12), kde každá z nich má na starost pouze určité úkoly.



Obrázek 12 - Rozdělení tříd do jednotlivých balíčků

Jak zobrazuje Obrázek 12, každý balíček plní v aplikaci určitou úlohu. Balíček *Data* se stará o spojení se síťovým elementem, získávání dat z něho a plnění těchto dat do balíčku *Model*, dále jsme si vytvořili balíček *View*, obsahující všechna okna aplikace a operace, které v nich můžeme provádět a poslední balíček *Properties* obsahuje složku s informacemi o jednotlivých zařízeních, ze nichž získáváme hostname daných zařízení a cestu k pinning souborům.

5.2.1. ZOBRAZENÍ VŠECH ROZHRANÍ A JEJICH STAVŮ

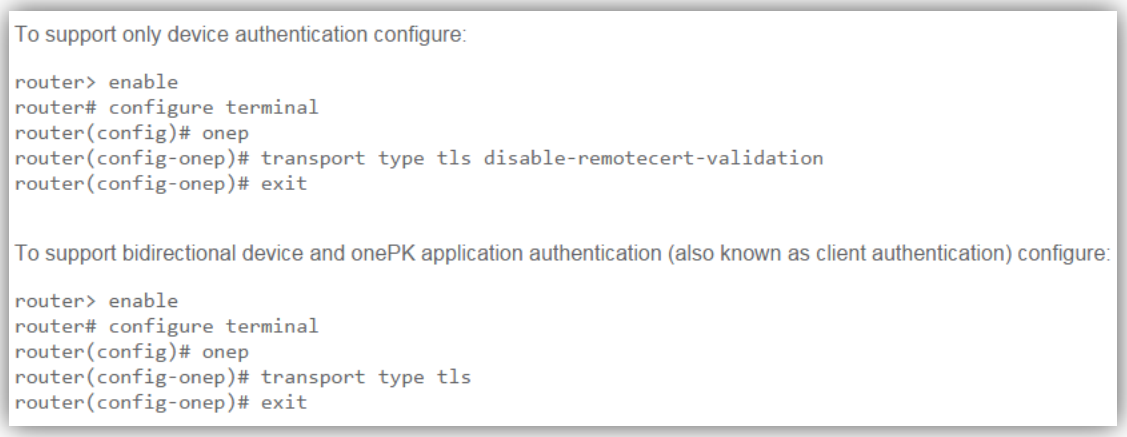
Ze všeho nejdříve je potřeba se pomocí aplikace připojit k požadovanému elementu. K tomu je zapotřebí znát IP adresu nebo hostname daného zařízení a identifikační údaje k přihlášení, kterými jsou přihlašovací jméno a heslo. Potřebné údaje, kromě přihlašovacího jména a hesla, nalezneme ve složce Properties vytvořené aplikace. Každé zařízení má svůj soubor, např.: *Router1.properties*, ve kterém nalezneme následující informace:

- **Hostname** - celý název zařízení, v našem případě to může být např.: router1.3node.example.com
- **Pinning file** - soubor, do kterého se ukládá přihlašovací certifikát TLS, pokud chceme, aby si aplikace pamatovala, že jsme přijali certifikát a už

se nás neptala na jeho opakované přijetí (více o TLS bude zmíněno níže)

- **Truststore** - soubor, ve kterém se ukládá seznam certifikátů, kterým se rozhodneme důvěřovat v našem připojení (v našem případě TLS)

TLS (Transport Layer Security) se doporučuje k bezpečné komunikaci mezi zařízením a aplikací. Poskytuje šifrovanou komunikaci, která využívá autentizaci pomocí certifikátů. V naší aplikaci byla využita jednosměrná autentizace, kde si síťové zařízení generuje vlastní certifikát. K plné funkčnosti je nejprve nutné nastavit směrovač tak, aby tento certifikát vygeneroval. Jak uvádí návod [26], směrovač nastavíme řadou příkazů v konfiguračním módu zařízení (Obrázek 13). Při samotném nastavování si však musíme dát pozor na to, o jaký typ komunikace v rámci TLS se jedná. Pokud se bude jednat o komunikaci mezi zařízeními, použijeme první sadu příkazů a v případě komunikace zařízení s OnePK aplikací je třeba využít druhou sadu.



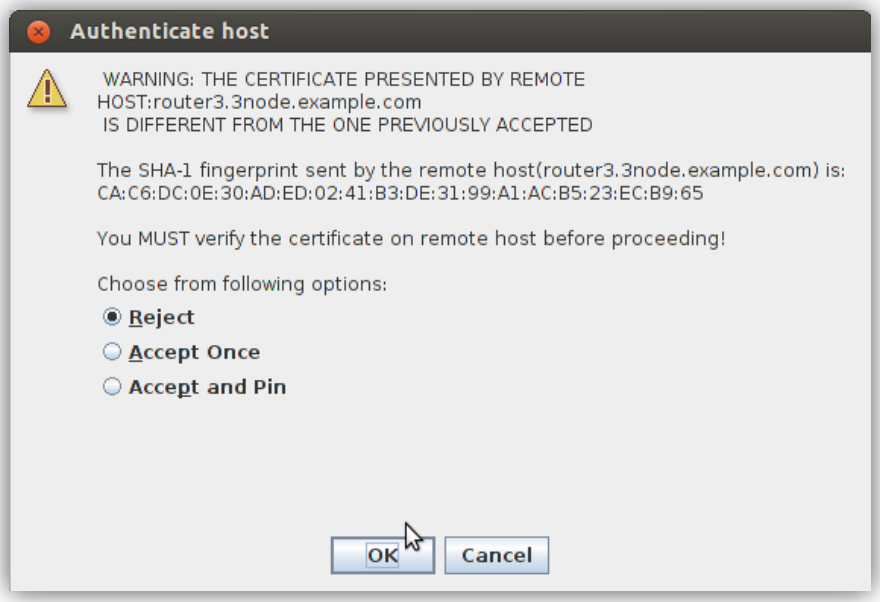
Obrázek 13 - Nastavení TLS na směrovači

Dále je potřeba před samotným připojením povolit TLS přímo v naší aplikaci a získat instanci síťového elementu, na který se chceme připojit. Instanci dostaneme pomocí jména elementu (hostname), které bylo získáno ze souboru *Properties* daného zařízení. Poté se lze připojit na požadované zařízení. Připojení vyžaduje bezpečnostní certifikát, který je vygenerován samotným zařízením.

Jakmile je certifikát vygenerován, musí si ho aplikace zkopírovat někam, kde

bude dosažitelný a bude s ním moci pracovat. Poté si ho aplikace buď porovná s již dříve zmíněným *Pinning file* a pokud nebude v pinning file bude uživateli nabídnuto dialogové okno, kde si bude moci vybrat mezi těmito možnostmi (Obrázek 14) :

- **Reject** - Po odmítnutí certifikátu se spojení se zařízením neuskuteční a aplikace je ukončena
- **Accept once** - Přijme certifikát a aplikace bude spojena se zařízením
- **Accept and pin** - Přijme certifikát a uloží do pinning souboru



Obrázek 14 - Možnosti přijetí certifikátu

Jakmile jsme připojení k danému elementu, je možné získat seznam všech rozhraní na něm a pro tyto účely byla vytvořena metoda *getAllInterfaces()* vracející seznam všech rozhraní příslušného zařízení

```
public List<NetworkInterface> getAllInterfaces() {
        List<NetworkInterface> lstInterface = null;
        try {
            networkElement = getNetworkElement();
            lstInterface = networkElement.getInterfaceList(new
InterfaceFilter());
        } catch (Exception e) {
            getLogger().error(e.getLocalizedMessage(), e);
        }
        return lstInterface;
                                    }
```

Pro pohodlnější práci s rozhraními byla vytvořena modelovací třída *NetworkInterfaceModel*, kde se pro každé rozhraní budou ukládat všechny potřebné informace do globálních proměnných a když budou potřeba, jsou jednoduše dostupné z této třídy metodami *get* těchto proměnných. V následující ukázce kódu vidíme proměnné této metody, ze kterých je možné získat všechny potřebné informace pro aplikaci.

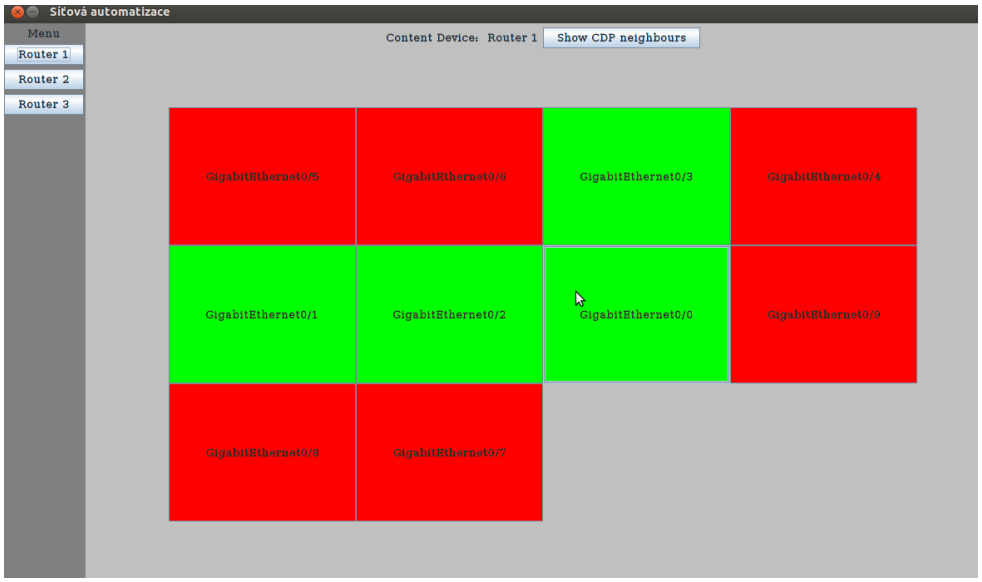
```
public class NetworkInterfaceModel {

    private String name;
    private String linkStateStatus;
    private String protocolStateStatus;
    private String ipAddress;
    private int prefixLength;
```

Po získání všech dostupných rozhraní na daném zařízení vytvoříme seznam (ArrayList) třídy *NetworkInterfaceModel*, který metodou *getNetworkInterfaceInfo* naplníme hodnotami ze seznamu Síťových rozhraní (*NetworkInterface*). Samotná metoda je vcelku obsáhlá a z našeho pohledu nijak zajímavá, takže zde nebude ukázána (jedná se pouze o získávání hodnot z instancí jednotlivých rozhraní a jejich ukládání do třídy *NetworkInterfaceModel* a to pouze z důvodu jednodušší manipulace s těmito hodnotami)

Jakmile máme připravenou metodu pro získání seznamu třídy *NetworkInterfaceModel*, můžeme přejít k zobrazení těchto informací uživateli. Aplikace má několik oken rozdělených podle informací, které jsou skrze tato okna poskytovány uživateli. Hlavní okno aplikace naskočí ihned po spuštění aplikace a umožňuje vybrat zařízení, ke kterému se chceme připojit. Po připojení zobrazuje všechna dostupná rozhraní daného zařízení a s pomocí tzv. naslouchačů, kterými

se bude tato práce zabývat níže, zobrazuje pomocí barev aktuální stav daného rozhraní (zelená -> zapnuto, červená -> vypnuto).



Obrázek 15 - Hlavní okno aplikace

Jak bylo řečeno dříve, požadované hodnoty získáme ze seznamu všech rozhraní podle vytvořené modelovací třídy. Metoda *getInterfaceInfo* vrací tento seznam, ze kterého pak metodou *get(int i)*, kde i je číslo daného rozhraní, získáme instanci třídy *NetworkInterfaceModel*, která zastupuje námi vybrané rozhraní a jejíž proměnné jsou připravené a jednoduše dostupné přes metody *get*. Tyto proměnné byly již vypsány výše.

```
public List<NetworkInterfaceModel> lstInter;

lstInter = connection.getInterfaceInfo();
```

Jakmile získáme tuto instanci, máme všechny potřebné informace o rozhraní, které potřebujeme a můžeme tyto hodnoty předat do textových polí a popisků. Tuto instanci vybraného rozhraní společně s instancí třídy connection je potřeba předat třídě *InterfaceDetailWindow*, která se postará o samotné zobrazení detailních informací a práci s rozhraním. Detailní okno ukazuje Obrázek 16.

Ke správné funkčnosti aplikace je nutné si uvědomit, že změny stavu i IP adresy, či vypnutí rozhraní, lze dosáhnout jak naší aplikací, tak použitím příkazů CLI přímo na daném zařízení. Proto bylo třeba k instanci připojeného elementu

v aplikaci přidat naslouchače (ang. Listener) událostí. Jelikož sledujeme v našem zařízení status rozhraní a jeho adresy, byly přidány naslouchače právě pro tyto dvě události. Oba naslouchače byly k elementu přidány jako anonymní vnitřní třídy implementující metodu *handleEvent*, která se spustí po výskytu dané události a umožní nám na danou událost reagovat. Jelikož postup přidání naslouchače k instanci síťového elementu zde byl vysvětlen, v následujícím kódu bude uveden pouze obsah metody *handleEvent*, která v případě vypnutí rozhraní jeho tlačítko vybarví červeně a v případě zapnutí zeleně.

```
if
(lstBtnInterface[i].getText().equals(ise.getInterface().getName())
&&
ise.getInterface().getStatus().getLinkState().toString().equals("ONE
P_IF_STATE_ADMIN_DOWN")) {
      lstBtnInterface[i].setBackground(Color.red);
}
if
(lstBtnInterface[i].getText().equals(ise.getInterface().getName())
&&
ise.getInterface().getStatus().getLinkState().toString().equals("ONE
P_IF_STATE_OPER_UP")) {
      lstBtnInterface[i].setBackground(Color.green);
}
```

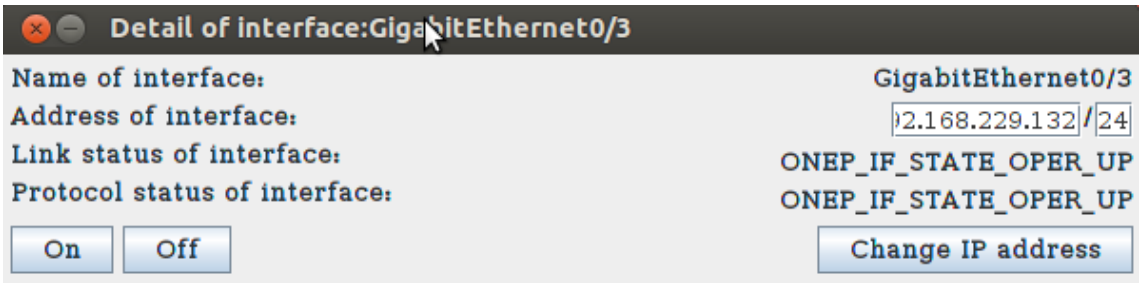
Naprosto stejně bude přidán naslouchač změny IP adresy, který bude měnit hodnoty v seznamu rozhraní hodnotami získanými právě z naslouchače změny IP adresy.

```
for(int i = 0; i<lstInter.size();i++){
if(lstInter.get(i).getName().equals(iace.getInterface().getName()))
   lstInter.get(i).setIpAddress(iace.getNewAddr().getHostAddress());
   lstInter.get(i).setPrefixLength(iace.getNewPrefixLength());
   detailWindow.setNetworkInterfaceModel(lstInter.get(i));
   detailWindow.setValuesToLabels();
   }
}
```

5.2.2. SPRÁVA ROZHRANÍ

Jedním z hlavních úkolů aplikace byla možnost správy těchto rozhraní, takže jakmile máme instanci požadovaného rozhraní i připojení, můžeme se pustit do jejich správy. Jak vidíme na Obrázek 16, dané rozhraní můžeme ovládat pomocí

tlačítek vypnout, zapnout a změnit IP adresu.



Obrázek 16 - Okno pro správu jednotlivých rozhraní

Pro změnu statusu rozhraní slouží následující metoda, která přijímá pouze název rozhraní a proměnnou typu boolean. Pomocí názvu rozhraní vyhledá odpovídající rozhraní na zařízení a následně umožňuje nad tímto rozhraním zavolat metodu *networkInterface.shutdown()* a tím ovládat, jestli dané rozhraní bude zapnuté.

```
public void changeStatusOfInterface(boolean status, String name) {
        try {
            NetworkInterface networkInterface =
networkElement.getInterfaceByName(name);
            networkInterface.shutdown(status);
        } catch (Exception ex) {
            System.out.println(ex);
        }
```

Této metodě předáme pouze název rozhraní, které chceme vypnout a proměnnou typu boolean (pouze TRUE/FALSE), která určuje, jestli chceme dané rozhraní vypnout (FALSE) nebo zapnout (TRUE).

Pokud chceme změnit IP adresu daného rozhraní, musíme znát následující hodnoty proměnných:

- **Název rozhraní**
- **IP adresu**
- **Prefix**

Jak vidíme níže, všechny potřebné hodnoty lze získat pomocí příkazu *InetAddress.getByName().* Tato metoda přijímá pouze textový řetězec (z důvodu přítomnosti teček v IP adrese nelze použít celočíselnou hodnotu), do které vložíme

proměnnou typu string, obsahující IP adresu rozhraní, které chceme aktualizovat.

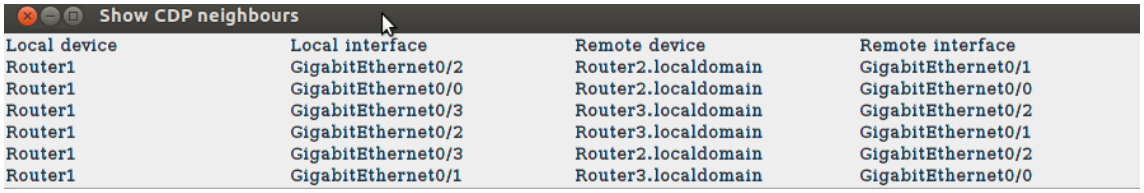
```
public void changeIpOfInterface(String name, String address, int
prefixLength) {
        try {
            NetworkInterface networkInterface =
networkElement.getInterfaceByName(name);
            InetAddress add = InetAddress.getByName(address);

networkInterface.updateAddress(OnepAddressScopeType.ONEP_ADDRESS_IPv4
_PRIMARY, add, prefixLength);
        } catch (Exception ex) {
            System.out.println(ex);
        }
    }
```

Opět získáme instanci rozhraní pomocí jeho názvu a nad ní zavoláme metodu *updateAddress(),* která v konstruktoru přijímá hodnotu určující typ adresy (IPv4 nebo IPv6), již zmíněnou IP adresu a prefix, které jsou získány z textových polí okna pro detail rozhraní (Obrázek 16). Ovšem jelikož *updateAdress* vyžaduje proměnnou typu *InetAddress* a z textového pole pro adresu získáme String, musí být získaná hodnota přetypována do již zmíněného typu *InetAddress*. Nejjednodušší varianta přetypování je ukázána výše v ukázce kódu a konkrétně se jedná o šedě zvýrazněný řádek.

5.2.3. ZOBRAZENÍ SOUSEDNÍCH ZAŘÍZENÍ

Pro správu sítě je velice důležité vědět, jaká zařízení máte připojená k tomu síťovému prvku, se kterým právě pracujete a skrze která rozhraní jsou mezi sebou navzájem propojena. Pro správnou konfiguraci rozhraní na obou stranách spojení je třeba znát název všech rozhraní, která chceme konfigurovat. Oba názvy lze snadno získat pomocí tabulky (Obrázek 17), která všechny potřebné informace získává pomocí protokolu CDP.



Obrázek 17 - Okno pro zobrazení sousedních zařízení

Cisco Discovery Protocol (dále jen CDP) je protokol společnosti Cisco, operující na vrstvě 2 síťového modelu OSI. Je nezávislý na přenosovém médiu a umožňuje

zjistit, jaké síťové elementy jsou přímo připojená k zařízení, se kterým právě pracujeme. Přímo připojená zařízení jsou taková, která mezi sebou sdílí jedno přenosové médium. Pomocí tohoto protokolu tedy bohužel nelze zjistit celou topologii sítě, ale pouze zařízení vzdálená 1 HOP od našeho směrovače [4].

Samotné zjištění sousedních zařízení se provádí následovně. Nejprve je potřeba vytvořit novou instanci třídy *Topology*, které do konstruktoru předáme instanci síťového prvku, se kterým se právě pracuje a jako druhý parametr konstruktoru třída *Topology* přijímá typ topologie (protokol, kterým prozkoumá topologii). V tomto případě aplikace využívá protokolu CDP. Následně získáme seznam všech spojení pro dané zařízení a uložíme ho do proměnné *lstEdge*.

```
//zjištění topologie a jejího grafu
Topology topology = new Topology(connection.getNetworkElement(),
TopologyType.CDP);
Graph graph = topology.getGraph();
//získání všech všech hran grafu
List<Edge> lstEdge = graph.getEdgeList(Edge.EdgeType.UNDIRECTED);
```

Každé spojení má na obou koncích připojené zařízení. Proto je potřeba projít celý seznam *lstEdge* a pro každý prvek v tomto seznamu vypsat počáteční a koncové zařízení i s názvy portů, skrze které jsou spojeny. Zařízení, ze kterého vysíláme požadavek, získáme pomocí metody *getHeadNodeConnector()* a zařízení na druhé straně přenosového média pomocí metody *getTailNodeConnector()* obdobně nalezneme i názvy portů pro příslušné médium a tyto hodnoty vypíšeme do jednotlivých popisků v tabulce. Realizaci vypsání jednotlivých spojení je ukázána níže.

```
if (lstEdge.size() > 0) {
	for (Edge edge : lstEdge) {
		headNodeConnector = edge.getHeadNodeConnector();
		tailNodeConnector = edge.getTailNodeConnector();
		headNode = edge.getHeadNode();
		tailNode = edge.getTailNode();

		lstLblData.add(new JLabel(headNode.getName()));
		lstLblData.add(new JLabel(headNodeConnector.getName()));
		lstLblData.add(new JLabel(tailNode.getName()));
		lstLblData.add(new JLabel(tailNodeConnector.getName()));
	}
}
```

5.3. SHRNUTÍ VÝSLEDKŮ

Vývoj vlastní aplikace v prostředí OnePK je velmi usnadněn všemi poskytovanými informacemi přímo ve virtuálním stroji *„All-in-one-VM“*, pokud vývojář zná dobře zvolený programovací jazyk a drží se informací, které lze jednoduše získat přímo z plochy virtuálního stroje, není samotný vývoj aplikace nijak velký problém.

Jsou zde poskytnuty návody na určité operace. Dostupné jsou skrze ikonu na ploše pod názvem *„onePK Tutorials“* ukazující, jak implementovat určité operace. A to od základních, jako připojení k elementu, které je potřeba u všech následujících operací, až po pokročilé operace jako nastavení QoS politiky na elementu atd. Veškeré návody jsou po krocích popsány a do detailu vysvětleny. Jednotlivé operace, které jsou zde popsané, mají i své funkční ukázky ve složce *„/home/cisco/onePK-sdk-1.2.1.194/java/tutorials/src/main/java/com/cisco/onep/tutorials“*, kde se nachází ukázkový kód, přehledně okomentovaný a u každé části kódu je vysvětlena její funkce.

Pokud některé informace nejsou v návodech obsaženy, je možné je získat z dokumentace, která je rovněž připravena na ploše ve složce *„onePK-Documentation“*. Při vývoji aplikace jsem se setkal s problémem po opakovaném připojení na zařízení, kde aplikace zůstala připojena k původnímu síťovému elementu a navíc se připojila k zařízení, které bylo nově zvoleno. A podrobnosti o vytváření a ukončování připojení mezi aplikací a síťovým elementem byly dohledány právě v dokumentaci pro Java API.

Je tedy vidět, že i při výskytu nějakého problému je celkem snadné dohledat řešení, aniž bychom museli dlouze prohledávat internet a OnePK se tak ukazuje jako velice mocný nástroj schopný implementovat řešení, které právě potřebujeme, bez větších problémů. Nicméně je potřeba držet na vědomí, že OnePK je stále relativně nová technologie a může se stát, že některý náš nápad nebude zatím schopno realizovat, proto je potřeba si před samotnou implementací zjistit, jestli náš problém má v OnePK možné řešení.

6. ZÁVĚRY A DOPORUČENÍ

Jak již bylo řečeno v úvodu, v dnešní době rozsáhlých topologií sítí a stále se zvětšujících požadavků na poskytované služby, což je důsledkem zvyšujícího se počtu mobilních zařízení (mobily, tablety, notebooky, atd.), je na správce sítě kladen stále větší důraz na úspory v nákladech na správu sítě a času potřebného k její údržbě.

Cílem této práce tedy bylo prozkoumat oblast počítačové sítě a zjistit v ní možné využití síťové automatizace tak, aby to správcům ušetřilo jak čas, tak i peníze a to realizovat tento nápad pomocí technologie OnePK od společnosti Cisco a tím si technologii vyzkoušet. Pro realizaci automatizace byla vybrána oblast správy sítě a její konfigurace.

V první části práce je vysvětlena automatizace jako pojem, její základní princip a možná využití v reálném světě. Dále popisuje zařízení, se kterými se můžeme setkat, a poté popisuje, jak je realizována konfigurace sítě a její správa. Do jakých skupin se správa sítě dělí, jak probíhá komunikace mezi správcem a zařízením, které je spravováno a popis jednotlivých protokolů realizujících tuto komunikaci.

Další část je věnována nástrojům, umožňujícím automatizování síťových úkonů se zaměřením na síťový operační systém. Každý systém nám nabízí různá řešení, jak automatizaci realizovat a v této části jsou jednotlivé technologie popsány, vysvětleny, porovnány mezi sebou a ukázány jejich schopnosti a přednosti.

Poslední část práce je zaměřená na implementování úloh právě pomocí OnePK ve virtuálním prostředí All-in-one-VM, což bylo cílem celé práce. Je zde popsán vývoj aplikace, která má několik základních funkcí, souvisejících s konfigurací, či správou sítě. Je zaměřená na konfiguraci směrovačů a jejich rozhraní.

OnePK se tedy prokázalo jako velice mocný nástroj nabízející široké možnosti práce s prvky v síti a je limitován především znalostmi vývojáře. Jelikož je ale onePK stále relativně nový, je nutno podotknout že pár věcí stále nefunguje, či není implementováno a je potřeba tyto chyby odstranit. Práce s OnePK je při znalosti jednoho z nabízených programovacích jazyků vcelku jednoduchá a skrývá veliký potenciál pro vývoj aplikací pro zařízení od společnosti Cisco.

Jako další možnosti zkoumání oblasti síťové automatizace by bylo možné srovnání s technologiemi pro automatizaci ostatních operačních systémů (Junos, RouterOS), nebo implementace složitějších úloh v onePK, kde jsou možnosti téměř neomezené.

7. ZDROJE

1. SCHIFFMAN, Mike. *How to Use Network Automation to Save Time and Money* [online]. [cit. 20.2.2014]. Dostupný na WWW: http://www.eweek.com/c/a/IT-Infrastructure/How-to-Use-Network-Automation-to-Save-Time-and-Money/
2. SINGH, Harsh. *Automating Network Operations for Greater Agility* [online]. [cit. 20.2.2014]. Dostupný na WWW: http://www.enterprisenetworkingplanet.com/datacenter/automating-network-operations-for-greater-agility.html
3. ROUSE, Alien V.. *Understand the role of scripting in network administration* [online]. [cit. 20.2.2014]. Dostupný na WWW: http://www.techrepublic.com/article/understand-the-role-of-scripting-in-network-administration/
4. NETACAD. *Cisco Networking Academy* [online]. 2007 [cit. 2015-04-03]. Dostupné z: www.netacad.com
5. TECHTARGET. Network automation overview [online]. 2010 [cit. 2014-07-15]. Dostupné z: http://searchnetworking.techtarget.com/tutorial/Network-automation-overview
6. NETOMATA. *Netomata Config Generator (NCG)* [online]. [cit. 2014-07-17]. Dostupné z: http://www.netomata.com/tools/ncg
7. ALTERPOINT. *Alterpoint* [online]. [cit. 2014-07-17]. Dostupné z: http://www.alterpoint.com/
8. SOLARWINDS. *Network Configuration Manager* [online]. [cit. 2014-07-17]. Dostupné z: http://www.solarwinds.com/network-configuration-manager.aspx
9. CLEMM, Alexander. *Network management fundamentals*. Indianapolis, IN: Cisco Press, c2007, xxiii, 510 p. ISBN 15872013725500.

10. TECHTARGET. *Command line interface (CLI)* [online]. 2005 [cit. 2014-07-21]. Dostupné z: http://searchwindowsserver.techtarget.com/definition/command-line-interface-CLI

11. CISCO. *Using the Command-Line Interface in Cisco IOS Software* [online]. 2010 [cit. 2014-07-22]. Dostupné z: http://www.cisco.com/c/en/us/td/docs/ios/iproute_bfd/configuration/guide/15_1/irb_15_1_book/usingios.html

12. JUNIPER. *Junos network operating system* [online]. [cit. 20.2.2014]. Dostupný na WWW: http://www.juniper.net/us/en/products-services/nos/junos/

13. CALL, Curtis. DAY ONE: APPLYING JUNOS OPERATIONS AUTOMATION. *Day One Library* [online]. 2009 [cit. 2015-03-23]. Dostupné z: https://jncie.files.wordpress.com/2010/02/7100109-en.pdf

14. JUNIPER. *Applications in the Junos Architecture* [online]. 2012 [cit. 2015-03-26]. Dostupné z: http://www.juniper.net/techpubs/en_US/junos12.2/topics/concept/sdk-architecture-overview.html

15. MIKROTIK. *MikroTik RouterOS* [online]. [cit. 20.2.2014]. Dostupný na WWW: http://www.mikrotik.com/pdf/what_is_routeros.pdf

16. MIKROTIK. *Scripting* [online]. 2012 [cit. 2015-03-26]. Dostupné z: http://wiki.mikrotik.com/wiki/Manual:Scripting

17. MIKROTIK. *API* [online]. 2012 [cit. 2015-03-26]. Dostupné z: http://wiki.mikrotik.com/wiki/Manual:API

18. CISCO. C*isco Internetwork Operating System (Cisco IOS)* [online]. [cit. 20.2.2014]. Dostupný na WWW: http://www.cisco.com/c/en/us/support/docs/ios-nx-os-software/ios-software-releases-110/13178-15.html

19. BLAIR, Arvind DURAI a John LAUTMANN. *Tcl scripting for Cisco IOS*.

Indianapolis, Ind.: Cisco Press, c2010, xvi, 294 p. ISBN 978-158-7059-452.

20. WHEELER, Bert. *Tcl/Tk 8.5 Programming Cookbook* [online]. 2011 [cit. 2014-07-23]. Dostupné z: https://www.packtpub.com/sites/default/files/2985OS-Chpater-1-The-Tcl-Shell.pdf

21. CISCO. *Cisco IOS Embedded Event Manager (EEM)* [online]. [cit. 20.2.2014]. Dostupný na WWW: http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-embedded-event-manager-eem/datasheet_c78-692254.html

22. CISCO. *Writing Embedded Event Manager Policies Using Tcl* [online]. [cit. 20.2.2014]. Dostupný na WWW: http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/eem/configuration/xe-3s/Writing_Embedded_Event_Manager_Policies_Using_Tcl.pdf

23. CISCO. *Real-Time Event Detection and Response and Task Automation Using Cisco IOS Embedded Event Manager* [online]. 2009 [cit. 2014-07-26]. Dostupné z: http://www.cisco.com/c/en/us/products/collateral/switches/catalyst-6500-series-switches/white_paper_c11-567132.html

24. SCHIFFMAN, Mike. *Cisco Blog* [online]. [cit. 20.2.2014]. Dostupný na WWW: https://blogs.cisco.com/security/ciscos-onepk-part-1-introduction/

25. *DEVNET. All-in-One Virtual Machine* [online]. 2014 [cit. 2014-08-15]. Dostupné z: https://developer.cisco.com/site/networking/one/onepk/sdk-and-docs/all-in-one-vm/

26. DEVNET. *Getting Started with onePK Version 1.2* [online]. 2014 [cit. 2014-08-15]. Dostupné z: https://developer.cisco.com/site/networking/one/onepk/discover/getting-started/

8. PŘÍLOHY

I. KONFIGURAČNÍ SOUBOR VIRTUÁLNÍ TOPOLOGIE

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
<topology xsi:schemaLocation="http://www.cisco.com/VIRL
http://cide.cisco.com/vmmaestro/schema/virl.xsd" schemaVersion="0.3"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.cisco.com/VIRL">
    <node vmImage="/usr/share/vmcloud/data/images/vios.ova"
    location="188,263" subtype="vios" type="SIMPLE"
    name="router1">
        <extensions>
            <entry type="String" key="bootstrap
            configuration">/home/cisco/vmcloud-example-
            networks/3node/router1.con</entry><entry
            type="String" key="import
            files">/home/cisco/vmcloud-example-
            networks/3node/router1.p12</entry>
        </extensions>
        <interface name="GigabitEthernet0/0"/>
        <interface name="GigabitEthernet0/1"/>
        <interface name="GigabitEthernet0/2"/>
        <interface name="GigabitEthernet0/3"/>
    </node>
    <node vmImage="/usr/share/vmcloud/data/images/vios.ova"
    location="488,319" subtype="vios" type="SIMPLE"
    name="router2">
        <extensions>
            <entry type="String" key="bootstrap
            configuration">/home/cisco/vmcloud-example-
            networks/3node/router2.con</entry><entry
            type="String" key="import
```

```
            FILES">/HOME/CISCO/VMCLOUD-EXAMPLE-
            NETWORKS/3NODE/ROUTER2.P12</ENTRY>
        </EXTENSIONS>
        <INTERFACE NAME="GIGABITETHERNET0/0"/>
        <INTERFACE NAME="GIGABITETHERNET0/1"/>
        <INTERFACE NAME="GIGABITETHERNET0/2"/>
</NODE>
<NODE VMIMAGE="/USR/SHARE/VMCLOUD/DATA/IMAGES/VIOS.OVA"
LOCATION="371,407" SUBTYPE="VIOS" TYPE="SIMPLE"
NAME="ROUTER3">
        <EXTENSIONS>
            <ENTRY TYPE="STRING" KEY="BOOTSTRAP
            CONFIGURATION">/HOME/CISCO/VMCLOUD-EXAMPLE-
            NETWORKS/3NODE/ROUTER3.CON</ENTRY>
            <ENTRY TYPE="STRING" KEY="IMPORT
            FILES">/HOME/CISCO/VMCLOUD-EXAMPLE-
            NETWORKS/3NODE/ROUTER3.P12</ENTRY>
        </EXTENSIONS>
        <INTERFACE NAME="GIGABITETHERNET0/0"/>
        <INTERFACE NAME="GIGABITETHERNET0/1"/>
        <INTERFACE NAME="GIGABITETHERNET0/2"/>
</NODE>
<NODE LOCATION="374,520" TYPE="SEGMENT" NAME="VMC_LAN_1"/>
<NODE LOCATION="671,235" TYPE="ASSET" NAME="ETH1">
        <INTERFACE NAME="NONE0"/>
        <INTERFACE NAME="NONE1"/>
</NODE>
<NODE LOCATION="722,161" TYPE="SEGMENT" NAME="LAN_EX"/>
<CONNECTION DST="/TOPOLOGY/NODE[2]/INTERFACE[1]"
SRC="/TOPOLOGY/NODE[1]/INTERFACE[1]"/>
<CONNECTION DST="/TOPOLOGY/NODE[3]/INTERFACE[1]"
SRC="/TOPOLOGY/NODE[1]/INTERFACE[2]"/>
<CONNECTION DST="/TOPOLOGY/NODE[4]"
SRC="/TOPOLOGY/NODE[1]/INTERFACE[3]"/>
```

```
        <CONNECTION DST="/TOPOLOGY/NODE[4]"
        SRC="/TOPOLOGY/NODE[2]/INTERFACE[2]"/>
        <CONNECTION DST="/TOPOLOGY/NODE[4]"
        SRC="/TOPOLOGY/NODE[3]/INTERFACE[2]"/>
        <CONNECTION DST="/TOPOLOGY/NODE[6]"
        SRC="/TOPOLOGY/NODE[3]/INTERFACE[3]"/>
        <CONNECTION DST="/TOPOLOGY/NODE[6]"
        SRC="/TOPOLOGY/NODE[5]/INTERFACE[1]"/>
        <CONNECTION DST="/TOPOLOGY/NODE[6]"
        SRC="/TOPOLOGY/NODE[1]/INTERFACE[4]"/>
        <CONNECTION DST="/TOPOLOGY/NODE[5]/INTERFACE[2]"
        SRC="/TOPOLOGY/NODE[2]/INTERFACE[3]"/>
</TOPOLOGY>
```

II. KONFIGURAČNÍ SOUBOR SMĚROVAČE

```
VERSION 15.3
SERVICE TIMESTAMPS DEBUG DATETIME MSEC
SERVICE TIMESTAMPS LOG DATETIME MSEC
NO SERVICE PASSWORD-ENCRYPTION
!
HOSTNAME ROUTER1
!
BOOT-START-MARKER
BOOT-END-MARKER
!
!
!
NO AAA NEW-MODEL
MMI POLLING-INTERVAL 60
NO MMI AUTO-CONFIGURE
NO MMI PVC
MMI SNMP-TIMEOUT 180
!
IP CEF
```

```
NO IPV6 CEF
IPV6 MULTICAST RPF USE-BGP
!
MULTILINK BUNDLE-NAME AUTHENTICATED
!
USERNAME CISCO PRIVILEGE 15 PASSWORD 0 CISCO123
!
REDUNDANCY
!
INTERFACE GIGABITETHERNET0/0
IP ADDRESS 10.10.20.110 255.255.255.0
DUPLEX AUTO
SPEED AUTO
NO SHUTDOWN
!
INTERFACE GIGABITETHERNET0/1
IP ADDRESS 10.10.30.110 255.255.255.0
NO SHUTDOWN
DUPLEX AUTO
SPEED AUTO
!
INTERFACE GIGABITETHERNET0/2
IP ADDRESS 10.10.10.110 255.255.255.0
NO SHUTDOWN
DUPLEX AUTO
SPEED AUTO
!
INTERFACE GIGABITETHERNET0/3
IP ADDRESS DHCP
NO SHUTDOWN
DUPLEX AUTO
SPEED AUTO
!
IP FORWARD-PROTOCOL ND
```

```
!
!
NO IP HTTP SERVER
NO IP HTTP SECURE-SERVER
!
LINE CON 0
LINE AUX 0
LINE VTY 0 4
TRANSPORT INPUT ALL
!
ONEP
TRANSPORT TYPE TLS LOCALCERT DEMOTP DISABLE-REMOTECERT-VALIDATION
START
!
END
!
CONTROL-PLANE
!
!
LINE CON 0
LINE AUX 0
LINE VTY 0 4
TRANSPORT INPUT ALL
!
ONEP
TRANSPORT TYPE TLS LOCALCERT DEMOTP DISABLE-REMOTECERT-VALIDATION
START
!
END
```

III. PŘILOŽENÉ CD

Přiložené CD obsahuje kompletní projekt *„OnepkApp“*, psaný v programu *Netbeans*, a to včetně:

- Jednotlivých *.java* souborů pro ukázku vypracovaného kódu ve složce „OnepkApp\src\“
- Vygenerované dokumentace javadoc, která byla vygenerována do složky s cestou „OnepkApp\dist\javadoc\“
- Spustitelného souboru .jar ve složce „OnepkApp\dist\“

Je však nutno podotknout, že aplikace sama o sobě nebude fungovat, neboť byla vypracována v prostředí *„All-in-one-VM“*, které umožňuje vytvoření virtuálních zařízení. Bez těchto zařízení bohužel není možné aplikaci otestovat.

FIM UHK
UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Rokitanského 62, 500 03 Hradec Králové, tel: 493 331 111, fax: 493 332 235

Zadání k závěrečné práci

Jméno a příjmení studenta: **Miloš Gábrle**

Obor studia: Aplikovaná informatika
Jméno a příjmení vedoucího práce: **Vladimír Soběslav**

Název práce:
Automatizace síťových prvků

Název práce v AJ:
Network Automation

Podtitul práce:

Podtitul práce v AJ:

Cíl práce: Cílem bakalářské práce je analyzovat technologie a systémy pro automatizaci úloh na aktivních síťových prvcích a implementovat je do vybraných úloh z oblasti automatizace provozu síťových prvků.

Osnova práce:
1. Úvod
2. Cíle a metodika práce
3. Automatizace úloh
4. Nástroje pro automatizaci
5. Implementace vybraných úloh
6. Závěry a doporučení
7. Zdroje

Normal 0 21 false false false CS X-NONE X-NONE

Projednáno dne: 15. 10. 2013

Podpis studenta

Podpis vedoucího práce