

MASARYKOVA UNIVERZITA  
FAKULTA INFORMATIKY



# Správa a manažment rozsiahlych sietí

DIPLOMOVÁ PRÁCA

**Bc. Juraj Michálek**

Brno, jar 2006

## **Prehlásenie**

Prehlasujem, že táto diplomová práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

**Vedúci práce:** Mgr. David Rohleder

## **Pod'akovanie**

Ďakujem Petrovi Trškovi za nápady a postrehy. Ďakujem všetkým, ktorí mi pomohli radou alebo pripomienkou. Ďakujem vývojárom slobodného a open source softvéru za obrovské množstvo nástrojov, ktoré som využil pri vytvorení tejto práce.

## **Kľúčové slová**

PDCA, Webmin, Cfengine, YancaP, Borax, Netconf, konfigurácia

## Zhrnutie

Predkladaná práca prináša náhľad do problematiky manažmentu sietí. Je predstavený model PDCA a následne je aplikovaný na integráciu nástrojov pre zjednodušenie správy siete. Sú analyzované tri vhodné riešenia: Webmin, Cfengine a Yencap. Jadro predstavuje prezentácia riešenia Borax I. a II. Pri aplikácii Borax I. je prevedená analýza chýb, ktoré boli zanesené do návrhu. Následne riešenie Borax II. má za cieľ tieto nedostatky odstrániť s využitím systému na správu verzií Subversion a sieťového protokolu Netconf. Prínos práce spočíva v novom spôsobe integrácie riešení pre určených správu siete.

## Obsah

<b>1</b>	<b>Problematika budovania a konfigurácie sietí</b>	4
1.1	Životný cyklus siete	5
1.2	Preventívne verzus nápravné opatrenia	8
1.3	Spracovanie zmien	9
1.4	Zhrnutie	10
<b>2</b>	<b>Súčasná riešenia a ich vlastnosti</b>	11
2.1	Webmin	11
2.1.1	Plán nasadenia aplikácie Webmin	12
2.1.2	Realizácia nasadenia aplikácie Webmin	14
2.1.3	Údržba prostredníctvom aplikácie Webmin	14
2.1.4	Sumarizácia návrhov na zlepšenie	15
2.1.5	Zhrnutie k aplikácii Webmin	15
2.2	Cfengine	16
2.2.1	Cfengine koncept	16
2.2.2	Plán nasadenia Cfengine	18
2.2.3	Realizácia nasadenia Cfengine	19
2.2.4	Údržba prostredníctvom Cfengine	19
2.2.5	Sumarizácia návrhov na zlepšenie	20
2.2.6	Zhrnutie k Cfengine	20
2.3	YencaP	21
2.4	Zhrnutie	22
<b>3</b>	<b>Borax</b>	23
3.1	Architektúra	24
3.1.1	Komunikačná architektúra manažér, server, klient.	26
3.1.2	Komunikačné modely	27
3.1.3	Rozšíriteľnosť	28
3.1.4	Call flow	28
3.2	Borax I.	29
3.2.1	Detaily implementácie	30
3.2.2	Problematické oblasti	31
3.2.3	Vyhodnotenie Borax I.	31
3.3	Borax II.	32

---

3.4	<i>Ukladanie a verzorvanie konfigurácie</i> . . . . .	33
3.4.1	Použitie Subversion v projektoch . . . . .	33
3.4.2	Použitie Subversion ako úložisko pre Borax II. . . . .	35
3.4.3	Archivácia oprávnení v Subversion . . . . .	37
3.5	<i>Protokol Netconf</i> . . . . .	39
3.5.1	Použitie Netconf v Borax II. . . . .	40
3.5.2	Interoperabilita Borax II. a YencaP . . . . .	40
3.6	<i>PDCA a Borax</i> . . . . .	41
3.6.1	Plán nasadenia Borax . . . . .	42
3.6.2	Realizácia nasadenia Borax . . . . .	42
3.6.3	Údržba prostredníctvom Borax . . . . .	43
3.6.4	Sumarizácia návrhov na zlepšenie . . . . .	43
3.6.5	Vyhodnotenie Borax II. . . . .	43
3.7	<i>Porovnanie dostupných riešení</i> . . . . .	44

## Úvod

Pri správe rozsiahlejšej siete bez vhodných nástrojov je správca nútený postupovať mechanicky a neefektívne. Cieľom práce je identifikovať a integrovať nástroje určené pre správu siete, aby sa zredukovala prácnosť procesu správy. Ďalším problémom je správa siete rozloženej na veľkom území, kde sa nachádzajú lokálni administrátori, ktorí však nemajú dostatočné znalosti na riešenie zložitejších problémov. Druhým cieľom je integrácia a prípadné vytvorenie nástrojov, pomocou ktorých je možné riešiť vzdialené zásahy s minimálnymi prenosmi a konzultáciami.

V tejto práci sú popísané aspekty, ktoré súvisia s konfiguráciou rozsiahlejších sietí a so spôsobom, akým je možné vysporiadať sa s problémami, ktoré správa takejto siete prináša. Prvá kapitola predstavuje manažérsky pohľad na sieť. Je predstavený model PDCA, ktorý umožňuje použiť procesný prístup k riadeniu siete.

Druhá kapitola sa zameriava na dostupné softvérové riešenia. Z pohľadu riešenia problémov prostredníctvom softvérových nástrojov existuje niekoľko vhodných kandidátov, ktorí umožňujú zjednodušiť správu sietí a prípadne delegovať oprávnenia na ich správu medzi viacerých správcov. V práci sú zmapované nástroje väčšieho rozsahu ako Webmin a Cfengine, v ktorých sú zastúpené kľúčové vlastnosti potrebné pre správu siete.

V hlavnej časti práce je rozpracované riešenie problematiky správy sietí prostredníctvom nástroja Borax, ktorý som navrhol a implementoval. Postupne sú predstavené dve rôzne implementácie. Na prvej implementácii je poukázané na nedostatky, ktoré sa objavili počas vývoja softvérového nástroja. Sú k dispozícii odporúčania akým spôsobom sa vyhnúť problémom, ktoré sa počas implementácie objavili. Následne je k dispozícii popis a analýza druhej implementácie, ktorá je už postavená nad protokolom Netconf. Nakoniec je uvedené zhrnutie a porovnanie všetkých študovaných riešení.

## Kapitola 1

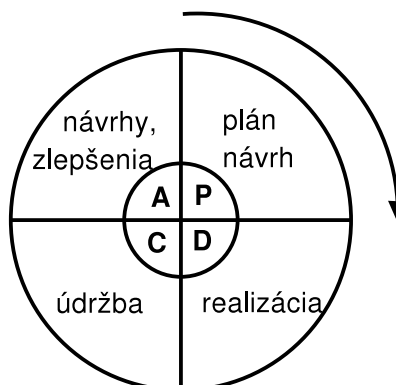
### Problematika budovania a konfigurácie sietí

V prvej kapitole bude definovaná terminológia využívaná v tejto práci. Pred tým, než pristúpime k samotnému popisu a riešeniu problémov, je nutné zdefinovať aký je význam termínov a slovných spojení, ktoré budú používané.

Za sieť budeme považovať sústavu zariadení schopných vzájomne komunikovať protokolom TCP/IP. Slovo zariadenie bude používané vo význame zariadenia, ktoré je možné konfigurovať a ktoré poskytuje v sieti určitú službu. Za takéto zariadenie môžeme považovať server, na ktorom bežia služby (napríklad služba elektronickej pošty). Ďalej týmto zariadením môže byť router. Konfiguráciou siete rozumieme proces, v ktorom je zanesené nastavenie do zariadení v sieti.

Ďalšou definíciou, na ktorú sa bude v práci odkazovať, je životný cyklus siete. Pojem životný cyklus siete je nutné definovať pre lepšie pochopenie a rozlíšenie rôznych situácií a okolností, ktoré súvisia s procesom konfigurácie siete. Ako základné fázy, pomocou ktorých je možné popísať životný cyklus siete, je možné prevziať osvedčené fázy, ktoré sú používané v oblasti vývoja softvérových produktov:

- návrh siete - v tejto fáze je zostavený návrh siete, je definovaná architektúra, ktorá bude použitá pri jej budovaní
- realizácia siete - sieť je budovaná a konfigurovaná podľa návrhu
- údržba siete - udržiavanie chodu siete, monitorovanie stavu a riešenie prípadných problémov
- návrh zlepšenia a úprav - na základe výsledkov a rozdielov medzi plánom a reálnym stavom siete, je možné navrhnúť vylepšenia, ktoré by zefektívni chod siete, táto fáza uzatvára jeden životný cyklus

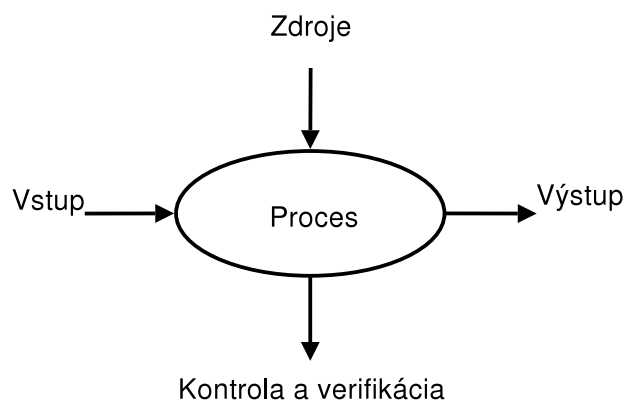


Obrázok 1.1: Mapovanie PDCA na životný cyklus siete

### 1.1 Životný cyklus siete

Fázy popísané v predchádzajúcom texte: návrh, realizácia, údržba, zlepšovanie; korešpondujú s modelom PDCA (Plan, Do, Check, Act). Tento model je označovaný ako Deming Wheel[1]. Cieľom modelu PDCA je vytvorenie podmienok pre neustále zlepšovanie sa, čo inými slovami môže v našom prípade znamenať zvyšovanie kvality siete a zlepšovanie služieb. Dôležitým aspektom PDCA je, že v súčasnej situácii máme len obmedzené znalosti a tým pádom nie je možné zvoliť „ideálny“ model siete. Architektúra siete a služby musia byť teda postupne zlepšované. Deming Wheel poskytuje jednoduchý návod ako takýto cieľ dosiahnuť. Proces, ktorý v našom prípade predstavuje jeden životný cyklus siete, je zostavený podľa zásad, modelu PDCA. Pre podporu takéhoto procesu je nutné mať k dispozícii vhodné nástroje, respektíve nástroje vhodným spôsobom nastavené. Korešpondencia medzi Deming Wheel a definíciou jedného životného cyklu je znázornená na obrázku 1.1.

Každá fáza cyklu vyžaduje špecifické zručnosti a skúsenosti. Pri väčšej sieti je žiadúce, aby jednotlivé fázy boli realizované ľuďmi so špecifickými znalosťami pre danú fázu. Na rozdiel od malých sietí nie je jednoducho možné, aby všetky štyri fázy boli realizované jedným človekom. Z pohľadu správy siete je zaujímavý prechod medzi jednotlivými fázami životného cyklu siete. Prechod z jednej fázy do druhej v podstate znamená, že riadenie prechádza od ľudí zodpovedných za realizáciu jednej fázy na ľudí zodpovedných za realizáciu fázy druhej. Z pohľadu manažmentu je nutné k týmto prechodom, respektíve presunom zodpovednosti pristupovať obozretne.



Obrázok 1.2: Model procesu

Pokúsme preto identifikovať jednotlivé prechody a popísať, čo je pre ne podstatné. Pre zjednodušenie práce použijeme abstrakciu, kedy jednu fázu budeme považovať za proces. Proces berie vstup, transformuje ho na výstup, pričom využíva dostupné zdroje a prevádza určité kontroly. Abstrakcia procesu je zachytená na obrázku 1.2. Takto definovaný proces sa bude využívať aj v ďalšom texte.

V nasledujúcom kroku prevedieme mapovanie životných fáz siete na procesy a identifikujeme jednotlivé atribúty procesu:

- návrh siete
  - vstup: požiadavky zmeny
  - výstup: návrh realizácie požiadaviek na zmeny
  - zdroje: aktuálny stav siete, finančné zdroje (rozpočet), znalosti
  - kontroly: plán nepresahuje rozpočet, v návrhu sú relevantné a nekonfliktné zmeny
  
- realizácia siete
  - vstup: súčasná infraštruktúra siete, návrh realizácie zmien v infraštruktúre
  - výstup: upravená infraštruktúra siete, upravené pracovné postupy a nástroje
  - zdroje: finančné zdroje, ľudské zdroje potrebné na realizáciu

- kontroly: dodržanie zámerov plánu, priebežná aktualizácia analýz rizík
- údržba siete
  - vstup: nová infraštruktúra, nové pracovné postupy a nástroje
  - výstup: monitoring, záznamy o stave siete, záznamy incidentov
  - zdroje: finančné zdroje, ľudské zdroje potrebné na údržbu a monitoring
  - kontroly: údržba nepresahuje rozpočet, anomálie fungovania siete sú zaznamenané
- návrh zlepšenia a úprav
  - vstup: záznamy o stavoch siete
  - výstup: požiadavka zmeny
  - zdroje: rozpočet, znalosti a skúsenosti z predchádzajúcich fáz
  - kontroly: v požiadavkách sú reflektované všetky kritické chyby a relevantné návrhy

Je dobré upozorniť, že uvedené štyri procesy sú definované mäkko, výstup jedného nemusí plne korešpondovať zo vstupom nasledujúceho procesu. Dôvodom k mäkkej definícii je fakt, že sa pohybujeme v oblasti manažmentu, ktorá sa nedá vyjadriť úplne exaktne. Samozrejme, že pri aplikácii uvedených modelov na konkrétne situácie, je možné jednotlivé atribúty procesov upresniť.

Prechod medzi jednotlivými fázami je kritický hneď z niekoľkých dôvodov. Ak má byť vykonaný prechod od jednej fázy k druhej, je nutné predať získané znalosti a informácie, ktoré sú relevantné pre beh v ďalšej fáze.

Prechod medzi návrhom siete a jej realizáciou spočíva v tom, že návrhy a plány sú predané skupine, ktorá sa postará o realizáciu a vybudovanie siete. Zaujímavejší prechod z fázy realizácie do údržby. Skupina, ktorá sa bude starať o beh siete musí prebrať určité množstvo znalostí od realizátorov siete.

Údržba je svojou povahou rutinná práca, tým pádom je nutné mať k dispozícii nástroje, ktoré umožnia túto rutinnú prácu realizovať a zredukujú prípadné problémy na minimálnu možnú mieru. Kľúčovou vecou pre fázu údržby je monitorovanie stavu siete a vedenie záznamov o zmenách, ktoré boli realizované. Zaznamenávanie zmien v konfigurácii siete je podstatné hlavne v prípade hľadania riešenia neočakávaného problému.

## 1.2 Preventívne verzus nápravné opatrenia

V oblasti manažmentu rizík býva venovaná veľká pozornosť preventívnym opatreniam. Úlohou preventívnych opatrení je zabrániť škodám, ktoré by mohli vzniknúť. Dobré rozpracovanie preventívnych opatrením je základom pre kvalitnú správu siete. Preventívne opatrenie môžeme chápať ako poisťku proti katastrofe.

Cieľom preventívneho opatrenia je znížiť možný dopad rizík na sieť. Riziko je definované pravdepodobnosťou výskytu problému násobená mierou jeho dopadu. Jednou z vhodných reprezentácií rizika je miera novej finančnej straty. Pokúsme sa identifikovať aspoň niektoré riziká majúce dopad na manažment siete. Nasledujúci zoznam môže poslúžiť ako základné vodítko pre určovanie rizík:

- nefunkčnosť siete - downtime
- poškodenie alebo nefunkčnosť služby poskytovanej sieťou
- poškodenie alebo zničenie zariadenia v sieti
- poškodenie záložných a zálohovacích mechanizmov
- útok na zariadenia
- únik informácií zo siete (konfigurácia, heslá, atď.)
- neprítomnosť administrátora
- podcenenie nákladov

Pre každú položku uvedenú v zozname je možné určiť množstvo príčin. Je nutné zobrať do úvahy tie, ktoré sú pre aktuálnu situáciu a stav siete rozhodujúce.

Ďalšími krokmi po identifikácii rizík je ich zotriedenie podľa výšky novej škody, vytvorenie plánu pre preventívne a nápravné opatrenia. V ďalšom kroku je nutné plán implementovať. Plán však nikdy nie je dokonalý a preto je nutné pravidelne prevádzať jeho revíziu a zlepšovanie. Je nutné odstrániť nepotrebné časti a pridať nové, ktoré sa objavili.

Všimnime si, že popísaný proces pre určovanie rizík opäť reflektuje proces zlepšovania - PDCA znázornený na obrázku 1.1. Poradie procesov je: identifikácia a hodnotenie rizík, tvorba plánu, implementácia, zhodnotenie a revízia.

Postavme proti sebe dva typy akcií, pomocou ktorých je možné sa vysporiadať s rizikami. Na jednej strane sú to preventívne opatrenia (preventive actions), ich cieľom je zabrániť vzniku nežiadúcej situácie. Na druhej strane sú to nápravné opatrenia (corrective actions), ktoré korigujú vzniknutý problém. U malých sietí alebo u sietí so zanedbaným manažmentom sa veľmi často môžeme stretnúť s tým, že existuje obrovské množstvo nápravných opatrení. Tento stav sa dá charakterizovať veľmi výstižne slovným spojením: „neustále sa hasia horiace problémy“. Takýto stav nie je dlhodobou udržateľný, vzhľadom na jeho náročnosť na zdroje (čas, peniaze, ľudia). Dôvod neudržateľnosti takéhoto stavu je jednoduchý. Stačí aby sa jednom okamihu zišiel dostatočný počet „požiarov“, ktoré nie je možné súčasne „uhasiť“.

Cieľom zlepšenia stavu siete musí byť presun od nápravných opatrení k preventívnym opatreniam. To znamená, že z pohľadu manažmentu je nutné venovať väčšiu pozornosť preventívnym opatreniam, nepodceňovať ich a prevádzať dostatočne často ich revíziu.

Na charakteristiku ceny preventívnych a opravných opatrení môžeme použiť veľmi známy cenový model z oblasti softvérového inžinierstva. Model hovorí, čím skôr je detekovaná chyba, tým je nižšia cena na jej opravu, pričom cena opravy nedetekovanej chyby rastie.

### 1.3 Spracovanie zmien

V tejto podkapitole sa budeme venovať štúdiu zmien a dopadov na implementáciu životného cyklu siete. Štúdium zmien je rozsiahla oblasť, či už na poli psychológie alebo na poli riadenia a manažmentu. My sa zameriame na aspekty, ktoré majú dopad na budovanie a údržbu siete.

V prvom rade je nutné sa pozrieť na psychologické dopady zmeny na ľudí. Dôvodom k tomuto kroku je fakt, že životný proces životného cyklu siete je implementovaný ľuďmi. Z pohľadu manažéra starajúceho sa o túto implementáciu, je kľúčové poznať faktory, ktoré sú zmenou ovplyvnené. Adaptáciu na zmenu výborne charakterizuje nasledujúcich niekoľko bodov:

1. odmietanie zmeny
2. hnev na okolie, pokus o násilné zastavenie zmeny
3. depresia, pocit straty starej situácie
4. prijatie zmeny, poznávanie novej situácie

5. objavenie kladných prvkov v zmene
6. plné prijatie zmeny

Na vysporiadanie sa s problémami, ktoré prináša zmena, má manažér k dispozícii nástroje, ktoré je možné použiť. Jedným z vhodných nástrojov je „Management Change Tool“ [2]. Je zostavený z niekoľkých sád textov usporiadaných do stromovej štruktúry. Manažér môže v tejto štruktúre identifikovať zmenu, ktorá ho zaujíma. Ku každému typu zmeny je pripojený krátky a hutný text, ktorý pojednáva o postupoch, akými je možné zmenu riešiť.

### 1.4 Zhrnutie

V tejto kapitole boli zadefinované základy terminológie používanej v ďalšom texte. Kapitola bola venovaná problematike manažmentu siete a zhodnoteniu niektorých aspektov, ktoré majú dopad na efektivitu nasadzovania nových riešení. Bol predstavený model PDCA, ktorého implementácia bude ukázaná v nasledujúcich kapitolách. Poslednou, aspoň okrajovo predstavenou časťou, bola problematika zmien a bol naznačený dopad na manažment siete.

## Kapitola 2

### SúčasnÉ riešenia a ich vlastnosti

V tejto kapitole budú analyzované dostupné riešenia, ktoré je možné využiť pre zjednodušenie konfigurácie siete a prípadne jej údržbu. Postupne budú rozobraté jednotlivé riešenia a vysvetlené koncepty, ktoré sú použité. Ku každému nástroju je uvedený spôsob, akým je možné integrovať ho do procesu správy siete.

U prvých dvoch nástrojov bude ich popis a integrácia vytvorená v súlade s princípom PDCA prezentovaným v predchádzajúcej kapitole. Cieľom takéhoto postupu je poskytnúť vodítko pre postup integrácie iných riešení do siete. Ako prvý nástroj bude predstavený Webmin, ktorý umožňuje prostredníctvom web rozhrania odtieniť administrátora od nutnosti priameho prístupu k systému. Druhým predstaveným nástrojom je Cfengine, ktorého hlavná sila spočíva vo vysoko úrovňovom jazyku slúžiacom na popis úloh, ktoré majú byť vykonané.

Posledným riešením, ktoré bude analyzované je Yencap. Jedná sa o open source implementáciu nástrojov podporujúcu protokol Netconf. Pochoopenie konceptov v predstavených nástrojoch a protokoloch je nutný predpokladom pre plné porozumenie textu v nasledujúcej kapitole.

#### 2.1 Webmin

Charakteristiku nástroja Webmi nájdeme v knihe The book of Webmin[6]. Citát z úvodnej kapitoly:

Webmin je webový nástroj určený na grafickú správu Unixového systému, napísaný Jamie Cameronom s použitím programovacieho jazyka Perl. Je navrhnutý tak, aby bol malý, funkčný a jednoducho rozšíriteľný. Webmin bol preložený do viac než 20 jazykov. Bol nasadený na všemožnom hardvéri a dodávatelia operačných systémov ho považujú za nástroj určený pre správu systému. Je jednoducho prenositeľný a je k nemu poskytovaná podpora pre viac než 35 rozličným Unixových operačných

systémov a Linuxových distribúcií. Je jednoducho rozšíriteľný o podporu nových vlastností a možností vďaka dostupnému a veľmi dobre zdokumentovanému aplikačnému rozhraniu.

Wemin[4] je nástroj pre unixové systémy, ktorý umožňuje správu systému prostredníctvom webového prehliadača. Webmin má za sebou veľkú komunitu vývojárov a používateľov. Aplikácia je zostavená zo sady CGI skriptov. Webmin je modulárny a je možné doň pridávať ďalšie moduly. Objavili sa aj pokusy integrovať API pre Python, ale integrácia nie je do dnešnej doby úplne hotová.

Webmin je primárne zameraný na správu jedného počítača, na ktorom beží a je určený aj pre začínajúcich administrátorov systému. Webmin obsahuje aj nástroje na správu viacerých počítačov v sieti, ktoré majú nainštalovanú aplikáciu Webmin. Prostredníctvom zabudovaného RPC komunikuje s ostatnými servermi a tým pádom je možné použiť jeden server na správu ostatných.

Jedna z dobrých vlastností aplikácie Webmin je možnosť rozdeliť oprávnenia pre nastavovanie jednotlivých modulov medzi viacerých administrátorov. Tým pádom sa o aplikácie a služby bežiace na serveri môže starať viac ľudí.

Webmin poskytuje web rozhranie prostredníctvom vlastného web servera na vlastnom porte (prevažne port 10000). Prístup na Webmin je možný prostredníctvom šifrovaného protokolu https.

### 2.1.1 Plán nasadenia aplikácie Webmin

Webmin je možné do procesu správy siete nasadiť v niekoľkých krokoch. Na vytvorenie postupu použijeme model PDCA prezentovaný v predchádzajúcej kapitole.

Prvým krokom je vytvorenie plánu, podľa ktorého bude Webmin nasadený. Stanovíme, aké servery alebo pracovné stanice bude možné pomocou Webmina spravovať. U každej stanice určíme, aké služby bude možné spravovať a aký administrátori budú pridelený na správu jednotlivých služieb. Do úvahy je nutné zobrať aktuálny stav siete a obmedzenia, ktoré stanovuje bezpečnostná politika siete. Pri tvorbe návrhu je vhodné položiť si niekoľko ďalších otázok:

1. Je žiadúce, aby mali administrátori prístup k aplikácii Webmin aj mimo siete?
2. Je nutné obmedziť prístup na Webmin na vybrané IP adresy? Ak áno, na aké?

3. Majú budúci administrátori dostatočné skúsenosti so správou prostredníctvom Webmina? Nie je nutné školenie?
4. Vyskytujú sa situácie, kedy majú viac než dvaja administrátori prístup rovnakej službe? Ak áno, akým spôsobom bude zaistené, aby si navzájom nezničili svoju prácu?
5. Aké kritéria určia úspešnosť a kvalitu nasadenia aplikácie Webmin?

Otázky 1. a 2. reflektujú aktuálny stav siete. Otázka 3. je dôležitá z hľadiska manažmentu a vyrovnanosti sa so zmenami. Ak neexistuje pozitívna odpoveď na túto otázku, je to indikácia, že pri prechode do fázy údržby (viď obrázok: 1.1) môžu nastať vážne problémy. Otázka 4. pokrýva oblasť problémov, ktoré môžu nastať. Dôvodom na položenie podobných otázok je odhalenie možných rizík a vytvorenia dostatočných opatrení, aby boli riziká eliminované. Otázku 5 nie je dobré podceňovať, aj keď sa to v mnohých prípadoch vytvárania plánov nasadenia nejakého riešenia deje. Bez dostatočnej odpovede, nie je návrh siete kompletný a nie je dobré prechádzať do fázy realizácie, pretože nie sú stanovené kritéria ako zmerať kvalitu. Vo fáze návrhov možných zlepšení tým pádom nie sú k dispozícii relevantné údaje. Vhodné kritériá ako určiť kvalitu riešenia môže byť napríklad:

- Doba riešenia problémov s konfiguráciou služieb na sieti klesla o ...
- Doba a náklady potrebné na zaškolenie nového administrátora klesla na ...
- Bolo zaznamenaných ... kritických incidentov spôsobenej neznalosťou Webmina.

Prezentovaný postup na vytvorenie návrhu siete je generický a pre potreby konkrétnej situácie je nutné ho doplniť o príslušné body. Vytvorenie plánu, resp. návrhu, sa môže javiť ako manažérska úloha. Túto fázu je však nutné previesť v úzkej spolupráci medzi manažmentom projektu a odborníkmi v danej oblasti. Napríklad, bez komunikácie s budúcimi administrátormi nie je možné relevantne odpovedať na otázku 3.

V prípade prvého nasadenia Webmina alebo nasadenia modulov, ktoré nie sú plne otestované, je nutnosťou mať vždy k dispozícii zálohu konfigurácie. Tento aspekt môže byť ošetrený napríklad pravidelnou automatickou zálohou. S hotovým plánom je možné prejsť do fázy realizácie.

### 2.1.2 Realizácia nasadenia aplikácie Webmin

Fáza realizácie nasadenia by mala byť uskutočnená podľa plánu a relevantné odchýlky od plánu by mali byť zaznamenané a uložené na neskoršie spracovanie. V tejto časti rozoberiem, ako môže vyzerat' jedno konkrétne nasadenie aplikácie Webmin a s čím je nutné počítať.

Samotnú inštaláciu Webmina na cieľové zariadenie (server/počítač) je možné realizovať prostredníctvom balíčku z Linuxovej distribúcie, alebo použitím zdrojových kódov. Podrobných postup inštalácie je možné nájsť v dokumentácii [4, 6, 5] a nie je ho preto nutné celý dopodrobna rozoberať.

Po inštalácii a spustení aplikácie je používateľovi dostupné administratívne web rozhranie, ktoré však obsahuje len veľmi malú funkcionálnosť a malý počet modulov určených na správu jednotlivých služieb. Rozhranie je prostredníctvom protokolu https dostupné na porte 10000 výhradne z lokálneho počítača (URL: <https://localhost:10000>). Takouto konfiguráciou sa snažia autori Webminu minimalizovať bezpečnostné riziko, ktoré by mohlo vzniknúť inštaláciou Webmina a ponechaním ho bez ďalšej konfigurácie.

Podľa plánu, ktorý bol stanovený v predchádzajúcej kapitole sa pridávajú moduly pre potrebné služby do Webmina a zmení sa nastavenie tak, aby bolo v súlade s bezpečnostnou politikou siete. Pre úplnosť je nutné uviesť, že pre Webmin existuje viac než 100 štandardných modulov a viac než 350 modulov od iných dodávateľov, slúžiacich pre správu rôznych služieb a nastavení. Je však nutné upozorniť, že nie všetky moduly dosahujú vynikajúcu kvalitu a pred ich nasadením je ich vhodné testovať.

### 2.1.3 Údržba prostredníctvom aplikácie Webmin

V poradí treťou fázou v procese nasadenia Webminu na správu siete je údržba. Táto fáza predpokladá, že administrátori boli dostatočne zaškolení. Ich úlohou je pomocou nástroja Webmin monitorovať a udržiavať konfiguráciu siete. Je nutné, aby boli evidované všetky anomálie a nedostatky, ktoré sa v procese údržby objavujú.

V prípade, že sú na správu nasadené moduly, ktoré nemajú dostatočnú kvalitu alebo novo vyvinuté moduly, je nutné zvláštnu pozornosť venovať výsledkom práce s nimi. U takýchto modulov je nutné mať k dispozícii zálohu stavu konfigurácie, pretože neotestovaný a nestabilný modul môže spôsobiť závažné zmeny v prípade, že sa objaví chyba.

### 2.1.4 Sumarizácia návrhov na zlepšenie

Poslednou fázou, ktorá uzatvára cyklus PDCA je spracovanie informácií, ktoré boli získané pri monitoringu siete. V tejto fáze prebehne vyhodnotenie kvality nasadeného riešenia. Jednak sa vyhodnotia kritéria, stanovené vo fáze návrhu a zosumarizuje sa, do akej miery boli splnené a kde je možné previesť ďalšie zlepšenia.

Zber dostatočných dát je u Webminu problémový, pretože Webmin nie je monitorovací nástroj. Ako podporné nástroje, ktoré je možné použiť pre zber údajov sú Logwatch[7, 8] a Nagios[9]. Opäť je však nutné zvážiť všetky aspekty súvisiace s ich implementáciou do siete, podobne ako sme to spravili pri aplikácií Webmin.

### 2.1.5 Zhrnutie k aplikácii Webmin

Relatívna jednoduchosť použitia a nasadenia Webmina má však aj svoje negatívne stránky. Pokúsme sa zhrnúť niekoľko faktov o Webminovi, ktoré hovoria proti jeho používaniu.

Aplikácia Webmin pracuje priamo pod oprávnením používateľa „root“, čo predstavuje bezpečnostné riziko. Webmin sa tým pádom stáva nástrojom, ktorý má neobmedzený prístup k celému systému. Pri plánovaní nasadenia Webmina je nutné zvážiť, či táto skutočnosť nie je v rozpore s otázkou bezpečnosti siete.

Webmin sa stal nechválne známy hlavne vďaka nie celkom korektné napísaným konfiguračným modulom, ktoré za určitých okolností boli schopné zničiť aktuálnu konfiguráciu bez možnosti návratu. Začínajúci administrátori boli často nemilo prekvapení, keď po nainštalovaní aplikácie a niekoľkých operáciách v aplikácii zmizli ich pôvodné nastavenia alebo v horšom prípade došlo k poškodeniu dát.

S právomocami rastie zodpovednosť, toto platí aj v prípade nasadenia aplikácie Webmin. Autori modulov sa ich síce snažia navrhnuť tak, aby v prípade kritických operácií bol používateľ včas upozornený a prípadne aby operáciu odvolal.

Tak ako každý použitý nástroj v sieti, aj Webmin obsahuje chyby. Základným preventívnym opatrením voči chybe spôsobenej Webminom je záloha. Ďalším preventívnym opatrením u nových modulov, typicky u modulov starých len niekoľko mesiacov, je testovanie. Pred nasadením modulu na systém, ktorý má byť jeho prostredníctvom spravovaný, je nutné tento modul dôkladne otestovať. Obrovskou výhodou modulov pre Webmin je dostupnosť zdrojového kódu. Takže detekovaná chyba môže byť

opravená.

Pri integrácii Webmina do siete odporúčam previesť niekoľko PDCA cyklov. Najskôr otestovať Webmina v malom a postupne rozširovať jeho nasadzovanie. Takýto iteratívny postup umožní detekovať chyby už v začiatkových cykloch nasadenia a tým pádom znížiť náklady na celkové nasadenie.

Ukážky práce s niektorými modulmi v aplikácii Webmin je možné nájsť v prílohe.

### 2.2 Cfengine

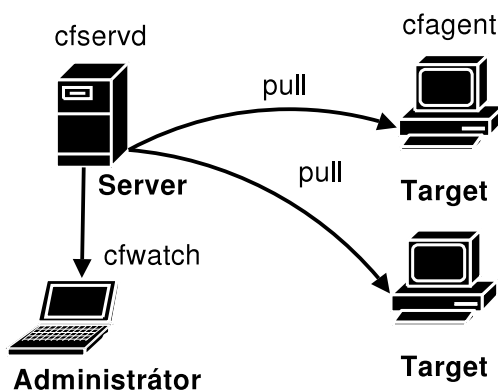
Cfengine[10] je nástroj, ktorý bol vyvinutý na Oslo University College v Nórsku. Prvotnou motiváciou tohoto projektu bolo zjednodušiť konfiguráciu a zjednotiť nástroje na konfigurovanie počítačov v sieti. Sieť obsahujúca väčšie množstvo počítačov sa veľmi rýchlo bez správnych nástrojov stane nespravovateľná. Ak sa ku konfigurácii na každom stroji pristupuje jednotlivo bez patričného systému tvorby konfigurácie, veľmi rýchlo dôjde k zneprehľadneniu nastavení. Administrátor napríklad na vyriešenie jednej úlohy vytvorí skript, ktorý ponechá na stroji, kde ho spustil. Takýmto spôsobom sa môže stať, že na každom počítači v sieti sa nachádza iná sada nástrojov. Čo je však horšie, tieto nástroje často nemajú jednotnú syntax a pravdepodobne ani dokumentáciu. Väčšina znalostí o tom, ako pracovať s takouto sieťou je v hlave administrátora, takže v prípade neprítomnosti administrátora sa sieť stáva nespravovateľnou. Ak sa má navyše zaškoliť nový pracovník do role administrátora, musí absorbovať netriviálne množstvo poznatkov o sieti a nástrojoch od pôvodného administrátora. Takýto prístup je veľmi neefektívny.

Cfengine sa s takýmto problémom vysporiadáva pomocou centralizovanej správy konfigurácie. Konfigurácia sa nachádza na serveri, odkiaľ je distribuovaná na klientov prostredníctvom programu cfagent.

#### 2.2.1 Cfengine koncept

Cfengine pozostáva z nasledujúcich komponentov [11]:

- cfagent - konfiguračný engine, ktorý komunikuje cez sieť pomocou zasielania požiadaviek na kopírovanie zo vzdialeného zdroja. Táto komponenta vykonáva väčšinu práce súvisiacej s konfiguráciou na základe pravidiel špecifikovaných v súbore cfengine.conf.



Obrázok 2.1: Cfengine model

- cfservd - služba, ktorá pracuje zároveň ako súborový server a správca úloh pre cfagent. Démon autentifikuje a spracováva požiadavky prichádzajúce zo siete na základe pravidiel špecifikovaných v súbore cfservd.conf.
- cfrun - jednoduchý inicializačný program umožňujúci spúšťanie cfagent na väčšom množstve počítačov. Nie je možné pomocou neho povedať programu cfagent, čo má robiť. Jediné, čo má tento program povolené je zavolať cfagent s konfiguračným súborom, ktorý už cfagent má k dispozícii. Tento program môže byť volaný hocikým, nevyžaduje špeciálne oprávnenia. Zamykací mechanizmus cfengine chráni tento program pred zneužitím.
- cfwatch - tento program (nie je súčasťou distribúcie Cfengine, jeho implementácia je ponechaná iným) má poskytovať grafické používateľské rozhranie, ktoré umožňuje sledovať konfiguráciu na klientoch, na ktorých beží cfagent a sledovať výstupy z prebiehajúcej konfigurácie.

Princíp fungovania Cfengine v sieti je jednoduchý. Na serveri sú uložené jednotlivé konfiguračné súbory. Cfagent, ktorý je spustený na klientovi si tieto súbory vyžiada a vykoná príslušné operácie. Fungovanie Cfengine je zachytené na obrázku 2.1.

Odporúčané nastavenie pre cfagent je, že cfagent je volaný v pravidelných intervaloch napríklad prostredníctvom programu cron. Používateľ môže cfagent volať priamo prostredníctvom príkazu.

Podstatnou vlastnosťou Cfenginu, ktorá ho odlišuje od skriptov, je vysoko úrovňový jazyk s relatívne jednoduchou syntaxou, pomocou ktorého je možné popísať úlohy súvisiace s aplikovaním konfigurácie.

### 2.2.2 Plán nasadenia Cfengine

Pri nasadení Cfengine opäť využijeme model PDCA (Plan, Do, Check, Act). Budeme postupovať podobne ako pri aplikácii Webmin. Prvým krokom je vytvorenie plánu. Musíme určiť aké zariadenia budú spravované a z akého servera budú získavať údaje o konfigurácii. Na cieľových zariadeniach bude musieť byť nasadený program cfagent a na serveri pobeží program cfserverd.

Pri integrácii bezpečnostnej politiky siete do návrhu, je dobré zvážiť niekoľko faktov. Vzhľadom na to, že agenti pracujú na princípe sťahovania informácií zo servera (pull model), na zariadení, kde cfagent beží nie je nutné mať otvorený port. Vráťme sa k aplikácii Webmin. Na každom stroji, kde Webmin bežal bolo nutné mať otvorený port, čo predstavovalo určité bezpečnostné riziko. Architektúra Cfengine vyžaduje otvorenie jediného portu. Cfserverd bežiaci na serveri otvára port, na ktorom počúva požiadavky od klientov (cfagent).

Cfengine podporuje centralizovanú správu. Hlavnú pozornosť zameriame preto na realizáciu centrálného servera. Pre každú službu, ktorá má byť konfigurovaná na cieľovom zariadení je nutné vytvoriť príslušné konfiguračné skripty. Preto vytvoríme zoznam služieb, skriptov a stanovíme spôsob testovania konfiguračných skriptov.

Posledným krokom pri tvorbe plánu je nutné určiť kritéria, na základe ktorých bude vyhodnotená kvalita a úspešnosť nasadenia Cfengine. Za základ môžeme zobrať kritériá uvedené v kapitole o aplikácii Webmin. Tie je vhodné doplniť o kritéria špecifické pre Cfengine:

- Doba potrebná na konfiguráciu nového zariadenia pridaného do siete klesla na ...
- Doba potrebná na nahradenie starého zariadenia novým a jeho konfiguráciu klesla na ...
- Množstvo skriptov používaných pri nastavovaní zariadení v sieti kleslo na ...

Postup vytvorenia plánu je nutné upraviť podľa potrieb aktuálnej situácie. Cfengine sám o sebe nepodporuje verzovanie konfigurácie, preto je nutné vytvoriť podporný plán pre archiváciu a zálohovanie. Existuje rozšírenie

pre Cfengine, ktoré umožňuje tento nástroj integrovať spolu s verzovacím systémom CVS[18].

### 2.2.3 Realizácia nasadenia Cfengine

Realizácia by mala prebiehať podľa stanoveného plánu. Je nutné nainštalovať server a vytvoriť skripty potrebné na konfiguráciu služieb a následne na jednotlivé cieľové zariadenia nainštalovať aplikáciu cfagent. K Cfengine existuje kvalitná a prepracovaná dokumentácia v publikácii Cfengine concepts [11]. Vzhľadom na to, že Cfengine je produkt, ktorý sa neustále vyvíja, je vhodné pri jeho nasadzovaní venovať pozornosť online dokumentácii vo forme Wiki. Jedná sa o Cfwiki[13], do ktorej prispieva množstvo administrátorov a používateľov Cfengine.

Na popis konfiguračných úloh, ktoré majú byť vykonané prostredníctvom aplikácie cfagent, bol pre Cfengine vyvinutý vlastný vysoko úrovňový jazyk. Jazyk je optimalizovaný na čo najjednoduchšie popisovanie úloh súvisiacich s konfiguráciou. Krátka ukážka inštrukcií pre Cfagent[11]:

```
# Comment...
control:
    actionsequence = ( links )
links:
    sun4::
        /bin -> /usr/bin
        # other links
    osf::
        # other links
```

Cfagent v tomto prípade vytvorí symbolický odkaz /bin smerujúci na adresár /usr/bin. K Cfengine existuje množstvo skriptov, ktoré sú už otestované a voľne dostupné. Tak ako v prípade Webmin, aj tu je nutné podotknúť, že nové alebo špecifické skripty je nutné pred nasadením dôkladne otestovať.

### 2.2.4 Údržba prostredníctvom Cfengine

Proces údržby Cfengine pozostáva hlavne z monitorovania a sledovania stavu siete. Veľkou posilou pre tento proces je nástroj na monitorovanie a detekovanie anomálií správania sa siete. Jedná sa o cdenvd[12]. Nástroj postupne zbiera informácie o bežiacich službách o ich stave. Jeho cieľom je detekovať anomáliu, ktorá sa vyskytla v dlhšom časovom období. Takáto

anomália môže indikovať útok alebo preťaženie servera. Autori uvádzajú, že na získanie dostatočného množstva informácií o monitorovanom stave, je nutné mať k dispozícii záznamy minimálne z 15 dní behu zariadenia.

Cfengine spracováva odchýlky v chovaní systému. Administrátor má potom k dispozícii prehľad o problémoch, ktoré sa môžu veľmi pomaly propagovať sieťou. Je možné použiť podporný nástroj cfevgraph, ktorý výsledky zberu dát zanesie na graf a odchýlky od „normálneho stavu“ sú jednoduchšie pozorovateľné. Ukážku tohoto nástroja je možné nájsť v prílohe.

### 2.2.5 Sumarizácia návrhov na zlepšenie

V tomto kroku je nutné zhodnotiť úspešnosť a kvalitu nasadenia Cfengine. Okrem hodnotenia kritérií, ktoré boli stanovené pri vytváraní plánu je v prípade Cfengine vhodné zamerať pozornosť na oblasť konfiguračných skriptov. Hodnotenie by malo byť zamerané na oblasti, kde boli potrebné najväčšie zásahy do skriptov. Vysoký počet zásahov indikuje prítomnosť problémov a nestability riešenia.

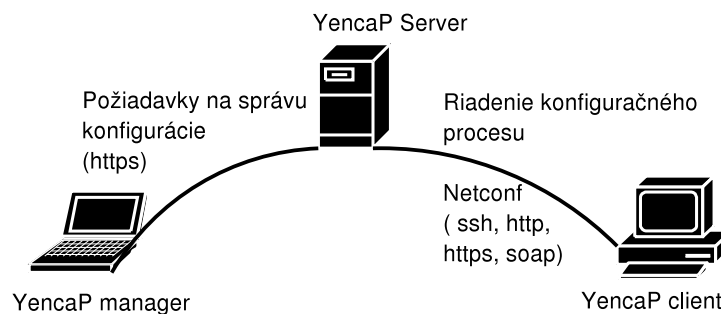
Pokiaľ sa na implementácii skriptov podieľalo viac ľudí, je vhodné zistiť, kto mal s tvorbou skriptov a údržbou riešenia problémy a akého charakteru tieto problémy boli. Na základe týchto informácií je možné vytvoriť návrh na zlepšenie situácie.

Týmto krokom sme uzavreli cyklus PDCA, ktorý by mal viesť k zlepšeniu kvality siete. V ďalšom cykle je opäť nutné previesť analýzu požiadaviek na zmenu, vytvoriť plán, zrealizovať ho, zmerať a vyhodnotiť ho.

### 2.2.6 Zhrnutie k Cfengine

Cfengine je vynikajúci nástroj na automatizáciu úloh súvisiacich s konfiguráciou zariadení v sieti. Cfengine využíva vlastný jazyk na popis úloh, ktoré sa majú vykonávať. Jazyk je špecifický pre Cfengine. Autori sa snažili zo začiatku držať idey, že jazyk bude čo najjednoduchší. Pri rozširovaní jazyka o novú funkcionálnosť však použili konštrukty, ktorých význam nie je zrejmý zo zdrojového kódu. Jazyk obsahuje konštrukty z jazyka C a C++, zmiešané s konštrukciami z jazykov ako Perl a Scheme. Pochopenie jazyka vyžaduje určité množstvo času. Všeobecne môžeme povedať, že implementácia vlastného jazyka je náročná a nie je možné potom dobre využiť nástroje, ktoré sú dostupné pre iné jazyky - ako zvyrazňovanie syntaxe, či overenie korektnosti syntaxe. Navyše jazyk sa problematicky integruje s ďalšími nástrojmi.

Aj napriek uvedeným problémom vychádzajúcich z návrhu jazyka sa



Obrázok 2.2: Model aplikácie YencaP

ukázal Cfengine ako stabilné riešenie používané rozsiahlou skupinou používateľov. Cfengine je neustále vyvíjaný. Na základe požiadaviek od administrátorov autori momentálne pracujú na novej verzii Cfengine3. Hlavným vylepšením tejto verzie je integrácia API pre C++ a ďalšie jazyky.

Cfengine je nástroj orientovaný na skripty. Je ho vhodné nasadiť v prípade, že administrátori a správcovia siete majú dostatočné skúsenosti so systémami typu Unix.

### 2.3 YencaP

Skupina nástrojov označovaná ako YencaP[14], sebe zjednocuje koncepty z nástrojov Webmin a Cfengine. Výsledkom je koncept, ktorý umožňuje prostredníctvom web rozhrania riadiť z centrálného servera konfiguráciu siete. Samotná konfigurácia je implementovaná podobne ako je to v prípade nástroja Cfengine. Na každom z cieľových strojov beží malý program YencaP Client, ktorý získava konfiguráciu od servera a aplikuje ju.

Podstatným rozdielom oproti Cfengine, je implementácia protokolu Netconf [16], pomocou ktorého je konfigurácia riadená. Tento protokol bude popísaný v nasledujúcej kapitole v súvislosti s riešením Borax. Je nutné povedať, že prvotnou motiváciou vzniku YencaP je práve implementácia protokolu Netconf. Architektúra YencaP je zachytená na obrázku 2.2.

Vzhľadom na to, že protokol Netconf je nový a momentálne stále podlieha zmenám, implementácia YencaP je momentálne len v začiatkoch. YencaP je možné rozširovať podobným spôsobom ako aplikáciu Webmin a síce použitím modulov. V súčasnosti sú však implementované len moduly pre: Asterisk, nastavovanie sieťových rozhraní, RBAC modul, Route modul, BGP Modul, RIP modul. K dispozícii je popis aplikačného rozhrania, takže je možné YencaP rozširovať o novú funkcionálnosť. Jazyk použitý na im-

plementáciu je Python, čo v sebe ukrýva ďalšie možnosti pre rozširovanie funkcionality.

### 2.4 Zhrnutie

V tejto kapitole boli vytvorené analýzy dvoch riešení Webmin a Cfengine. Podarilo sa úspešne preukázať, že model PDCA prezentovaný v úvode práce, je možné použiť pri implementácii týchto nástrojov do siete. Zároveň bola vytvorená šablóna, podľa ktorej je možné postupovať pri implementácii iných riešení do siete. Z pohľadu manažéra je tým pádom k dispozícii teoretický nástroj, ktorý mu umožní: vytvoriť plán riešenia, zrealizovať ho, previesť fázu údržby a monitoringu a následne získať podklady pre zlepšenie a skvalitnenie práce.

Zároveň boli v analýze spomenuté niektoré koncepty tvoriace základ pre ďalšiu kapitolu venovanú implementácii riešenia na skvalitnenie správy siete - Borax.

## Kapitola 3

### Borax

Táto kapitola obsahuje hlavné jadro práce. V kapitole budú postupne predstavené dve riešenia Borax I. a Borax II. Prvé riešenie Borax I. je historicky staršie a bolo vytvorené bez znalosti nástroja Cfengine a protokolu Netconf. V jednoduchosti sa dá povedať, že Borax I. je nástroj podobný Webminu, ktorý vie distribuovať úlohy po sieti. Vzhľadom na to, že riešenie Borax I. obsahovalo nedostatky, ktoré zabraňovali rozšíreniu o nové vlastnosti, bol vytvorený nový návrh nástroja.

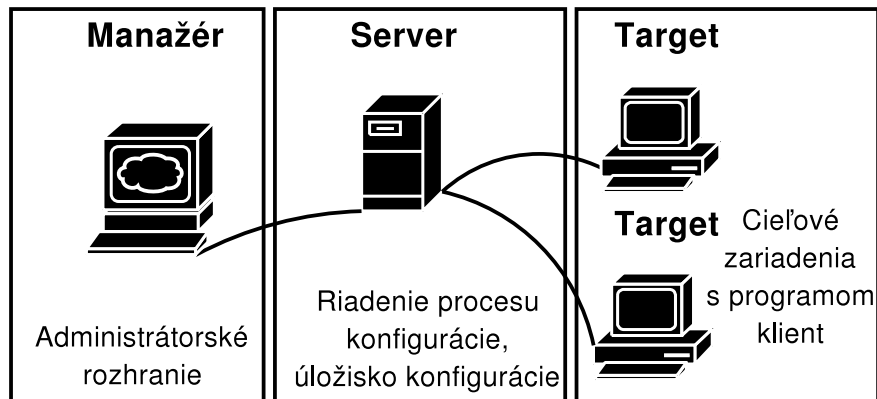
Borax II. je navrhnutý v súlade s myšlienkami, ktoré sú implementované vrámci riešenia ako Cfengine a Yencap. Hlavným prínosom riešenia je podpora verzovania konfigurácie.

V závere kapitoly sú zhrnuté výsledky, ktoré boli dosiahnuté počas implementácie riešení Borax I., II. Tiež je k dispozícii tabuľka, ktorá navzájom porovnáva všetky riešenia a nástroje spomínané v tejto práci.

V ďalšom texte sa budú vyskytovať pojmy, ktoré budú uvažované v nasledujúcom význame:

- manažér - rozhranie pomocou ktorého používateľ vytvára konfiguráciu (používané synonymum: administratívne rozhranie)
- server - služba bežiaci na počítači, ktorá zabezpečuje centrálnu údržbu konfigurácie.
- target - označenie cieľového zariadenia, na ktoré sa aplikuje konfigurácia (používané synonymum: cieľové zariadenie)
- klient - softvérový nástroj bežiaci na cieľovom zariadení, slúžiaci na aplikovanie konfigurácie

Pre lepšiu predstavu významu pojmov je vhodné použiť zjednodušenú schému na obrázku 3.1 zachytávajúcu architektúru systému. Voľba architektúry je rozobratá v nasledujúcej podkapitole.



Obrázok 3.1: Schematické zachytenie architektúry

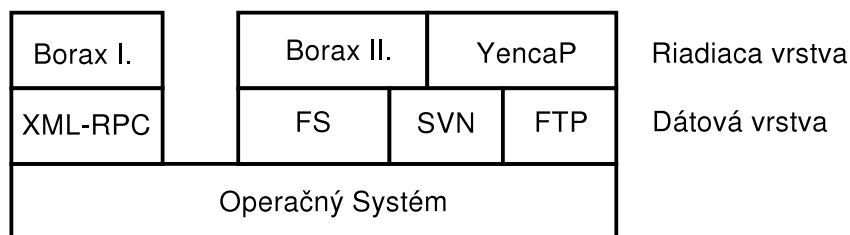
### 3.1 Architektúra

V tejto časti rozoberieme koncepty, ktoré sú základom pre vybudovanie systému pre konfiguráciu siete. Koncepty sa vzťahujú hlavne na riešenie Borax II. Pri riešení Borax I. ich použijeme na vyhodnotenie dôvodov problematickej implementácie.

Naším cieľom je navrhnúť architektúru, ktorá umožní vybudovať systém umožňujúci konfiguráciu zariadení v sieti. Za základ zoberme koncept vrstiev, ktorý sa osvedčil napríklad pri sieťovom modeli ISO/OSI. Rozdelíme riešenie do troch vrstiev. Základnou vrstvou je operačný systém. Nad ním postavíme dátovú vrstvu, ktorej účelom bude starosť o samotné konfiguračné dáta. Do tejto vrstvy môžeme zaradiť súborový systém (FS), FTP alebo úložisko dát dostupné prostredníctvom siete. Tieto dve vrstvy dokážu fungovať samostatne, ale nedokážu riadiť tok konfigurácie. Preto nad ne postavíme riadiacu vrstvu. Sem patria nástroje schopné usmerniť konfiguráciu prostredníctvom komunikačných alebo signalizačných protokolov.

Pre lepšiu predstavu je vhodné uviesť obrázok 3.2. Obrázok znázorňuje niekoľko možných usporiadaní, dôvody vedúce k voľbe konkrétnych komponentov v uvedených vrstvách budú upresnené neskôr.

Rozloženie architektúry do troch vrstiev nám umožňuje ku každej vrstve pristupovať samostatne. To nám umožní oddeliť ich funkcionality a v prípade potreby vytvoriť pomocné funkcie suplujúce inú vrstvu (stubs). Jedná sa hlavne o možnosť existencie dátovej vrstvy bez existencie riadiacej vrstvy. Tento fakt je veľmi dôležitý pri integrácii riešenia do už existujúcej siete, ako si ukážeme v podkapitole venovanej nasadeniu riešenia „PDCA



Obrázok 3.2: Vrstvy

a Broax''

V predchádzajúcej časti práce boli predstavené riešenia Webmin, Cfengine a YencaP. Každé z nich má odlišnú architektúru a koncepty, na ktorých je postavené. Určíme vlastnosti významné pre riešenie, ktoré by sa malo stať nástrojom pre konfiguráciu siete.

- Jednoduchosť a rozšíriteľnosť. Túto dvojicu vlastností je nutné uviesť na prvé miesto. Ich porušenie indikuje problémy v dizajne aplikácie. Preto je nutné pred každým krokom, ktorý rozširuje alebo upravuje funkcionality, položiť si otázku: Je riešenie naďalej jednoduché a je ho možné rozšíriť?<sup>1</sup>
- Komunikačná architektúra manažér, server, klient. U riešenia Cfengine sme sa stretli s architektúrou server – klient. Z nášho pohľadu môžeme povedať, že sa jedná o dátovú vrstvu. O riadenie tejto vrstvy sa stará administrátor. Na riadenie môžeme použiť napríklad web rozhranie, pomocou ktorého je možné zadávať parametre konfigurácie. Toto rozhranie označíme ako manažér.
- Prenos konfigurácie nad nedôveryhodnou sieťou. Konfiguračné súbory môžu obsahovať citlivé informácie. Pri sieťach rozložených na väčších územiach určite narazíme na to, že komunikácia prechádza cez nedôveryhodné úseky siete. Logickou požiadavkou je, vytvoriť dostatočné bezpečnostné opatrenia, aby nedošlo k úniku alebo poškodeniu informácií.
- Separácia spravovaných služieb medzi viacerých administrátorov. Z manažérskeho pohľadu je toto kľúčová vlastnosť, ktorá umožňuje dis-

1. U softvérových riešení sa veľmi často stáva, že je pridávaná nová funkcionality, bez premýšľania o tom, či nové funkcie nie sú v rozpore s pôvodnou myšlienkou riešenia. Tento problém veľmi pekne vystihuje Zen of Python - v prílohe.

tribuovanie a rozdeľovanie úloh. Má dopad na veľkosť siete, na rýchlosť úprav a samozrejme v neposlednej rade na kvalitu.

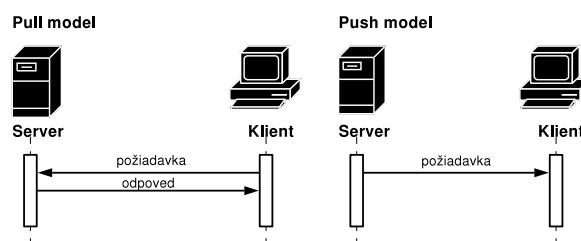
- Verzovanie konfigurácie. Tento podporný proces je bežnou praxou v softvérovom inžinierstve, ale v oblasti sietí nie je až tak dobre využívaný. Z manažérskeho pohľadu sa jedná o funkciu umožňujúcu návrat k predchádzajúcim stavom siete. Zároveň umožňuje zaznamenávať históriu zmien.
- Plánovanie konfigurácie. Nie každú zmenu je nutné vykonať okamžite. U niektorých zmien je podstatné ich presné načasovanie. Administrátori väčšinou pracujú na „živých“ systémoch a zmeny konfigurácie prevádzajú. V prípade, že je zmena rozsiahlejšia a vyžaduje viac krokov, tak sa výrazne zvyšuje riziko novej chyby. Plánovanie konfigurácie umožňuje presun k lepšie spravovateľnej sieti.
- Podpora pre atomickú konfiguráciu a rollback. Je možné, že počas aplikovania novej konfigurácie dôjde k chybe, ktorej riešenie by zabralo príliš veľa času. Logickou požiadavkou je preto mať mechanizmus, ktorý umožní vrátenie sa k predchádzajúcemu stabilnému stavu. Toto je možné za podpory systému na správu verzii veľmi dobre implementovať.

### 3.1.1 Komunikačná architektúra manažér, server, klient.

Skúsme podrobnejšie rozobrať model manažér, server, klient. Táto komunikačná architektúra je sklbením myšlienok z architektúr Cfengine a Webmin. Cfengine umožňuje distribuovanie konfigurácie zo servera na veľké množstvo klientov. Webmin umožňuje odtienenie administrátora od „nepodstatných“ zložiek systému a umožňuje rozdeliť privilégia pre správu systému medzi viacerých administrátorov.

Základnými dvoma komponentami je server a klient. Server udržiava konfiguráciu klientov, autentifikuje požiadavky, generuje odpovede pre klientov, prípadne zasiela informácie klientovi, čo má vykonať. Klient sa autentifikuje voči serveru, požaduje informácie a konfiguráciu. Manažérske rozhranie je prostredie, ktorým je konfigurácia prezentovaná voči administrátorovi. Rozhraní pre konfiguráciu jednej služby môže byť viac. Tým sa napríklad môže zabezpečiť, že jednu serverovú službu môže spravovať viacej administrátorov.

Dobrým príkladom využitia manažérskeho rozhrania môžu byť emailové aliasy. Klientom je zariadenie, na ktorom beží SMTP služba prijíma-



Obrázok 3.3: Pull a push model

júca e-maily pre niekoľko domén. Každú doménu vlastní iný používateľ. Účelná požiadavka je, aby si každý s používateľov mohol spravovať vlastnú doménu a nebolo mu nutné dávať administrátorský prístup na zariadenie SMTP službou. V takúto úlohu je možné riešiť nasledovne. Manažérske rozhranie umožní administrátorovi domény nastaviť si vlastné aliasy. Táto konfigurácia je uložená na server, odkiaľ si ju klientské zariadenie môže vyzdvihnúť a konfiguráciu aplikovať.

Realizácia manažérskeho rozhrania prostredníctvom web aplikácie je ľahko pochopiteľná aj pre laika, server si udržuje informáciu o stave konfigurácie (prípadne jej verzii) a dovoľí klientovi len zmeny, na ktoré má oprávnenie. Aplikovanie konfigurácie je možné časovo naplánovať. Administrátor domény nemusí mať vôbec prístup na zariadenie s SMTP službou.

### 3.1.2 Komunikačné modely

Pre rozpracovaním ďalej témy je nutné upozorniť na dva základné komunikačné modely s ktorými sa v sieti stretne. Dôvodom k ich rozlišovaniu je rôzny dopad na bezpečnosť a správanie sa voči firewallom. Jedná sa o nasledujúce modely (znázornené na obrázku 3.3):

- Pull model – komunikáciu klient iniciuje klient a vyžiada si údaje zo servera, ktorý mu zašle odpoveď.
- Push model – komunikáciu iniciuje server.

Pull model je využívaný v aplikácii Cfengine, kde server hrá pasívnu rolu (napr. file server) a aktívne nezasahuje do konfigurácie. Všetky zmeny konfigurácie sú ponechané na klientovi. Push model je aplikovaný v programe Webmin, kde zmenu ktoré sú vykonané vo web rozhraní sú automaticky premietané do konfiguračných súborov na klientovi.

Z bezpečnostného hľadiska je zaujímavý Pull model. V tomto prípade je nutné, aby bol server viditeľný pre všetkých klientov, ktorý s ním potrebujú komunikovať. Naopak klienti nemusia byť vôbec viditeľný pre server. Tento model môžeme prirovnať k situácii, kedy používateľ si pozerá stránku na serveri. Server nemá informáciu o stave klientov a klienti môžu byť zamaskovaní za firewallmi a nedostupný z vonkajšej siete.

Push model naopak vyžaduje, aby server videl klientov. Tento prístup umožňuje, aby konfigurácia zo servera bola na klienta umiestnená okamžite.

Každý z uvedených modelov má iné vlastnosti, záleží na konkrétnej situácii, ktoré z týchto vlastností sú relevantné. Komunikačná architektúra manažér, server, klient, predstavuje kombináciu oboch modelov. A síce medzi manažérom a serverom je to model push a pull. Medzi serverom a klientom je to model pull.

### 3.1.3 Rozšíriteľnosť

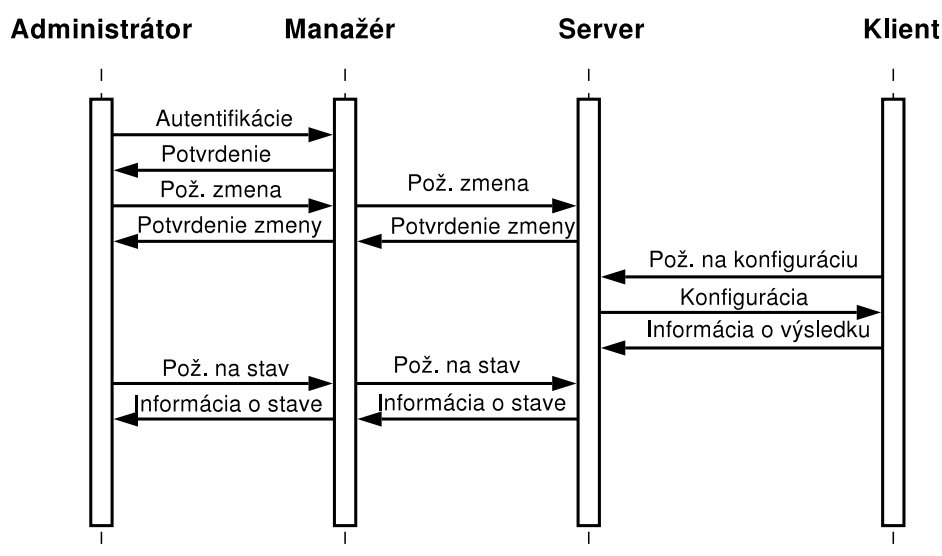
V tejto časti sa pozrieme, akým spôsobom je možné zabezpečiť rozšírenie aplikácie. Naším cieľom je zvoliť model, ktorý nám umožní čo najflexibilnejšie pridávať a odoberať funkcionality z aplikácie. Požiadavka pridávania funkcionality je zrejme, v dobe vývoja neboli známe všetky vlastnosti, ktoré by mal softvér spĺňať. Postupom času sa ukázalo, že je potrebné pridať nové vlastnosti.

Opačný smer k pridávaniu funkcionality je jej odoberanie. Dôvod tejto požiadavky už nie je natoľko jasný. Odoberanie funkcionality umožňuje prispôbiť riešenie na potreby aktuálnej situácie. Napríklad pokiaľ máme zariadenie s malou kapacitou pamäte, ktoré sa stará len o filtrovanie sieťovej prevádzky, je zbytočné, aby program aplikácie bežiaci na tomto zariadení obsahoval funkcionality pre konfiguráciu web servera.

Jednou z možností ako zabezpečiť rozšíriteľnosť je pridávať funkcionality priamo do kódu aplikácie. Takéto riešenie však pre smer odoberania funkcionality absolútne neschodné. Omnoho lepší a praxou odskúšaný model rozšíriteľnosti aplikácie sa ukázalo používanie modulov. Tento princíp je úspešne aplikovaný v riešení Webmin. S vhodnými programovacími nástrojmi, je možné princíp modulov jednoducho implementovať.

### 3.1.4 Call flow

Modelovanie situácií prostredníctvom call flow je veľmi účinná metóda, pomocou ktorej je možné jednoducho vizualizovať tok správ. Metódu je



Obrázok 3.4: Call flow modelujúci jeden z možných scenárov.

vhodné využiť hlavne pri systémoch pozostávajúcich z niekoľkých komunikujúcich komponent. Pomocou call flow je možné veľmi jednoducho overiť správnosť implementácie. Zároveň je možné vytvárať testy aplikácie nezávisle na jej implementácii.

Pokúsme sa vytvoriť návrh implementácie komunikačnej architektúry manažér, server, klient. Cieľom tohoto kroku je ukázať, akým spôsobom sa budú jednotlivé komponenty chovať. Vytvoríme jednoduchý call flow diagram zachytávajúci volania medzi komponentami.

Do diagramu chceme zachytiť jednu zo situácií, ktorá pri správe siete nastáva. Scenár je nasledovný: administrátor sa prihlási na manažérske rozhranie a zmení konfiguráciu vybranej služby, klient si vyžiada novú konfiguráciu, aplikuje ju a zašle informáciu serveru o stave. Ako posledný krok administrátor si vyžiada zobrazenie stavu konfigurácie. Call flow diagram je zachytený na obrázku 3.4.

Uvedený call flow použijeme ako hodnotení vlastností riešenia Borax v nasledujúcom texte.

### 3.2 Borax I.

V predchádzajúcom texte sme stanovili architektúru systému pre zjednodušenie správy a konfigurácie sietí. Pozrime sa na príklad implementácie. Bude

predstavené riešenie Borax I., ktoré do veľkej miery zohľadňuje popísanú architektúru. Cieľom tejto podkapitoly je poukázať na chyby, ktoré môžu vzniknúť pri návrhu a realizácii aplikácie.

Prvá verzia nástroja Borax vychádzala z potreby vytvoriť riešenie, ktoré umožní konfigurovanie služieb na serveroch prostredníctvom web rozhrania. Ako modelová serverová služba, pre ktorú bola vytvorená prvá verzia Boraxu, bola elektronická pošta. Úlohou bolo vytvoriť konfiguračný nástroj, ktorý by umožňoval definovanie emailových aliasov pre domény hostované na serveroch. Riešenie malo umožňovať rozdelenie právomocí na spravovanie aliasov pre jednotlivé domény.

Komunikačná architektúra bola stanovená na manažér, server, klient. Na začiatku sa spravili príslušné analýzy a následne bola implementácia realizovaná.

### 3.2.1 Detaily implementácie

Konkrétna implementácia bola zostavená nasledovne. Na cieľovom stroji, kde sa nachádzala bežiacia služba pre doručovanie pošty Courier Mail Transfer Agent, bol v intervaloch piatich minút spúšťaný malý klient s názvom Sentinel. Tento klient kontaktoval server s konfiguráciou, autentifikoval sa a prostredníctvom XML-RPC si zistil, či došlo k nejakej zmene, ktorú by mal replikovať zo serveru na target. Pokiaľ zistil, že má byť prevedená zmena, vyžiadal si od servera zoznam emailových aliasov, ktorých sa zmena dotýkala. Uložil ich do príslušného konfiguračného súboru a previedol príslušnú operáciu (makealiases), aby upozornil Courier-MTA na existenciu novej konfigurácie. Skript bol implementovaný v Pythone a na komunikáciu so serverom prostredníctvom XML-RPC je použitá knižnica Xmlrpc-lib. Dovolím si podotknúť že táto knižnica umožňuje v jazyku Python prevádzať veľmi flexibilne implementáciu komunikácie so vzdialenými objektami.

Na strane serveru udržiavajúceho konfiguráciu bežal aplikačný server Zope starajúci sa o komunikáciu s klientom a administrátorom. Konfiguračné údaje boli ukladané do relačnej databázy. Ako databázový server bol zvolený MySQL.

Cieľ vytvoriť nástroj zjednodušujúci správu emailových aliasov sa podarilo dosiahnuť v krátkom čase. Momentálne je riešenie nasadené na troch serveroch. Borax I. je aktívne využívaný na správu mail aliasov pre 13 rôznych domén, ktoré sú rozdelené medzi 5 administrátorov. Podrobnosti k tomuto riešeniu ako relačný model DB a implementačné detaily je možné nájsť v prílohe.

### 3.2.2 Problematické oblasti

Aj napriek úspešnej implementácii sa od ďalšieho rozširovania Borax I. upustilo. Bolo identifikovaných niekoľko problematických častí, ktoré boli dôsledkom nie celkom korektného návrhu Borax I. Prvým problémom bol návrh klienta Sentinel, ktorý slúžil na ukladanie konfigurácie na cieľovom stroji. Sentinel bol úzko zameraný na jednu službu a komunikačný protokol bol úzko špecifický, čo znamená, že pokiaľ by mal byť Sentinel rozšírený na ďalšiu službu, bolo nutné ho v podstate kompletne pre túto službu prepísať. Len pre informáciu je vhodné doplniť, že Borax II. využíva na komunikáciu medzi serverom a klientom protokol Netconf, ktorý je na účel konfigurácie navrhnutý.

Druhým úzkym hrdlom bolo ukladanie konfigurácie na serveri. Ako úložisko bola zvolená MySQL databáza, ktorá na tento účel nie je príliš vhodná. V konečnom dôsledku databázový systém suploval súborový systém a s počtom služieb, ktoré boli implementované by rástol počet tabuliek. Ukázalo sa, že implementácia podpory sledovania histórie a verzovania konfigurácii je príliš náročná.

Vyššie uvedené závery boli vyvozené ako dôsledok úvah a analýz možností rozšírenia Borax I. Modely a dokumenty súvisiace s analýzou rozšírenia Borax I. je možné nájsť v prílohe. Tieto materiály môžu byť prínosné pre ľudí, ktorí sa zaoberajú analýzou a implementáciou podobného riešenie.

### 3.2.3 Vyhodnotenie Borax I.

Pozrime sa na rozpory medzi architektúrou navrhnutou v prvej časti tejto kapitoly a stavom, do ktorého sa dostala implementácia v prípade riešenia Borax I. Za základ si zoberme kritéria navrhnuté pre architektúru aplikácie určenej na správu siete:

- Jednoduchosť. Nebola dodržaná, implementácia ukladania informácií do databázy vynucovala prítomnosť databázovej logiky, takže systém nedokázal bez databázového stroja pracovať.
- Rozšíriteľnosť. Nebolo dodržané. Implementácia klienta Sentinel bola spravená účelovo bez možnosti rozšírenia. Na serverovej strane vznikol problém v rozširovaní databázy.
- Komunikačná architektúra manažér, server, klient. Bolo dodržané. Ukázalo sa že model je odolný voči výpadkom spojenia aj presunom zariadení medzi segmentami v sieti.

- Prenos konfigurácie nad nedôveryhodnou sieťou. Bolo dodržané. Dokonca v jednom testovanom prípade, je dokonca konfigurácia prenášaná zo servera v jednej krajine na cieľovej zariadenie v inej krajine. Pričom komunikácia prechádza cez niekoľkých rôznych sietí.
- Separácia spravovaných služieb medzi viacerých administrátorov. Bolo dodržané. Implementácia sítě nemá úplne jednoducho vyriešené pridávanie nových administrátorov do manažérskeho rozhrania, avšak s aplikáciou mohlo pracovať niekoľko administrátorov nezávisle na sebe.
- Verzovanie konfigurácie. Nebolo dodržané. Pôvodne sa predpokladalo, že na verzovanie konfigurácie bude využitá databáza, čo sa následne ukázalo ako neschodná cesta.
- Plánovanie konfigurácie. Čiastočne dodržané. Administrátor si mohol nadefinovať novú konfiguráciu do databázy a ľubovoľne ju editovať. Následne keď bol hotový mohol zvoliť aplikovanie konfigurácie, ktorá sa do piatich minút zreplikovala na klienta.
- Podpora pre atomickú konfiguráciu a rollback. Nedodržané. Táto vlastnosť je závislá na existencii verzovacieho systému, ktorý v systéme Borax I. chýbal

Na základe stanovenej architektúry sme previedli porovnanie s implementáciou Borax I. Ako vidíme, nie všetky body sú dodržané. Tieto nedostatky sa stali východiskom pre návrh a implementáciu riešenia Borax II. Na porovnanie úspešnosti implementácie Borax II. použijeme opäť porovnanie vlastností riešenia voči navrhnutéj architektúre.

### 3.3 Borax II.

V ďalšom texte kapitoly si predstavíme riešenie Borax II., ktoré bolo motivované problémami zistenými pri vývoji riešenia Borax I. Hlavnou myšlienkou, ktorá od seba tieto dve riešenia odlišuje a ktorá sa ukázala ako prínosná, je maximálne využitie už existujúcich nástrojov a protokolov. Separácia architektúry riešenia Borax II: operačný systém, dátová vrstva a riadiaca vrstva; umožnilo kvalitatívny posun dopredu.

V prvej časti budú predstavené systémy pre správu verzií. Následne bude ukázané, akým spôsobom je možné rozšíriť verzovací systém Subversion, tak aby zodpovedal potrebám pre archiváciu konfigurácie. Návrh som

vypracoval na základe konzultácii a odporúčaní odborníkov, ktorý tento systém používajú už dlhšiu dobu.

Ďalšou podstatnou zmenou oproti Boraxu I. je použitie protokolu Netconf na riadenia procesu konfigurácie. Tento protokol je momentálne v stave „IETF draft“. Má za sebou históriu a ukazuje sa ako veľmi jednoduchý a plne použiteľný na svoj účel. Použitie protokolu Netconf ďalej umožňuje začlenenie inetoperability Boraxu a ďalších systémov do zoznamu podporovaných vlastností. Tento fakt bude dokumentovaný na spolupráci medzi riešením Borax a YencaP.

### 3.4 Ukladanie a verzovanie konfigurácie

Pri vývoji rozsiahlejšieho softvéru je podmienkou existencia verzovacieho systému. Bežne sa stane, že vývojár sa dostane do situácie, kedy sa potrebuje vrátiť k predchádzajúcej verzii editovaného súboru, prípadne potrebuje vedieť, ako vyzeral daný súbor pred pol rokom. Pri konfigurovaní služieb na serveri je administrátor vo veľmi podobnej situácii ako vývojár softvéru. Podporný proces pre správu verzii je označovaný ako „Version control“.

Administrátori pracujú veľmi často na živom systéme a to hlavne v prípade, že sa vyskytne chyba, ktorú je nutné vo veľmi krátkej dobe odstrániť. bežne sa stane, že nie je vôbec vedený záznam o zásahu a ani o zmenách, ktoré administrátor vykonal. Pôvodná konfigurácia je nenávratne stratená. Prípadne administrátor si skopíruje konfiguráciu do nejakého adresára ako zálohu, pre prípad, že by z nej nejaký súbor potreboval. Takáto konfigurácia zostane na počítači uložená, prípadne pri ďalšom zásahu pribudne ďalšia v inom adresári, ktorú tam pridal druhý administrátor. Tu je vidieť veľkú podobnosť vo svete vývojárov, kde je tento problém riešený práve verzovacím systémom.

V súčasnej dobe je k dispozícii niekoľko rôznych verzovacích systémov. V tabuľke 3.2 je možné nájsť porovnanie verzovacích systémov CVS[18], ClearCase[19], Git[20] a Subversion[21]. Verzovacích systémov na trhu je samozrejme omnoho viac, než je zachytené v tabuľke. Pre naše potreby boli vybrané len relevantné.

#### 3.4.1 Použitie Subversion v projektoch

V tejto podkapitole bude predstavený verzovací systém Subversion, ktorý sa používa pri vývoji softvéru na správu verzii. Popíšeme, akým spôsobom sa Subversion používa pre softvérové projekty. Podrobnosti k fungovaniu Subversion je možné nájsť v publikácii Version Control with Subversion[22],

	CVS	ClearCase	Git	Subversion
licencia	GNU/GPL	komerčná IBM	GNU/GPL	Apache, BSD-style
decentralizácia	nie	áno	áno	s použitím SVK
komunikačný protokol	CVS	MVFS	http, ssh, rsync, git	svn, ssh, https (DeltaV, WebDav)
podporované OS	Linux, BSD, Unix, Windows	Linux, Unix, Windows	Linux, BSD, Unix	Linux, BSD, Unix, Windows
väzba na jazyky	Perl, Python, Java, C	Perl, Python, Java, C	Perl, C++	Python, Ruby, Perl, Java, C...
uviedenie systému	1989	pred 1990	2005	2004

Tabuľka 3.1: Porovnanie verzovacích systémov

ktorá je priebežne dopĺňaná a aktualizovaná. Táto publikácia je dostupná aj v elektronickej podobe. V nasledujúcom texte uvedieme spôsob, ako má byť rozložená adresárová štruktúra, aby ju bolo možné úspešne využiť na verzovanie konfigurácie zariadení v sieti.

Pri vytvorení úložiska (repository) v Subversion, je nutné zvoliť adresárovú štruktúru (directory layout), do ktorej budú súbory ukladané. V rámci jedného úložiska môže existovať viacero projektov. Pre každý projekt je vhodné zvoliť jednoznačné meno, ktoré ho reprezentuje. Do adresára s menom projektu autori Subversion odporúčajú umiestniť nasledujúcu štruktúru[22]:

- trunk - Hlavný adresár, v ktorom sa nachádzajú aktuálne verzie súborov. Tu prebieha hlavná časť vývoja. Nie je garantované, že posledná revízia musí byť stabilná.
- branches - Adresár na ukladanie vetiev projektu. Napríklad pri implementácii novej vlastnosti do programu, ktorá má rozsiahly dopad na aplikáciu, je vhodné pre tieto zmeny vytvoriť vetvu, na ktorej budú zmeny prevádzané. Po dokončení implementácie a otestovaní, tieto zmeny môžu byť prenesené (merge) do hlavnej vývojovej vetvy v adre-

sári trunk.

- tags - Adresár na ukladanie vetiev projektu, ktoré však už nie sú menené. Väčšinou sa používa na uloženie vetvy, ktorá je stabilná a má otestované vlastnosti.

Takáto štruktúra je výhodná aj z hľadiska riadenia prístupu k úložisku. Subversion umožňuje definovať oprávnenia na zápis a čítanie na úrovni adresárov. Najbežnejšie riešenie, ktoré sa využíva pri open source projektoch je:

- Anonym má právo čítať v celom repository. Ktokoľvek môže prísť a získať si kód projektu.
- Vývojár má právo zápisu do adresára trunk a branches. Môžu tým pádom vytvoriť vlastnú vývojovú vetvu, prípadne prenášať kód medzi vetvami.
- Správcovia projektu majú právo zápisu do adresára tags. Jedná sa o ľudí, ktorí sa starajú o dohľadovanie a vývoj projektu. Zápis do adresára tags umožňuje správcovi projektu označiť verziu, ktorá je stabilná.

Popísaný model je osvedčený a používaný v mnohých projektoch a veľa vývojárov je zvyknutých na jeho dodržiavanie. Subversion tento model nevynucuje. Je možné zvoliť ľubovoľný iný model, ktorý rešpektuje potreby realizovaného projektu.

#### 3.4.2 Použitie Subversion ako úložisko pre Borax II.

V predchádzajúcej kapitole boli prezentované možnosti použitia Subversion v softvérových projektoch. Pre správne využitie Subversion ako úložiska pre konfiguráciu, je nutné spraviť niekoľko úvah a úprav pôvodného modelu. Vytvoríme úpravy, ktoré nepoškodzujú architektúru Subversion a pritom poskytujú širšie možnosti pre ukladanie konfigurácie. Ako je naznačené na obrázku 3.2, je možné Subversion použiť ako úložisko konfigurácie bez nutnosti prítomnosti nadradenej riadiacej vrstvy Borax II. Tým pádom môžeme použiť Subversion samostatne. Ako si ukážeme, toto je kľúčový fakt, veľmi podstatný pri integrácii do už existujúcich sietí.

V nasledujúcom kroku sa pokúsime previesť model určený pre správu projektov v Subversion previesť na model správy konfigurácie. Cieľom je vytvoriť také mapovanie, aby bola zachovaná logika modelu pre správu projektov.

V prípade konfigurácie služieb na zariadeniach v sieti nehovoríme o projektoch. Slovo projekt môžeme dať v takom prípade do rovnosti so slovom služba, ktorú chceme nakonfigurovať. Namiesto projektov budeme v našom prípade hovoriť o službe poskytovanej v sieti. Takou službou v sieti môže byť SMTP. Po krátkej úvahe ale zistíme, že takéto pomenovanie neposkytuje potrebnú granularitu pre rozlišovanie konfigurácie. Konfigurácia služby SMTP v sieti predstavuje konfiguráciu procesov, ktoré túto službu realizujú. Preto namiesto uvedeného mapovania na služby použijeme mapovanie na mená procesov poskytujúcich službu. Ako mená procesov teda môžeme použiť označenie ako apache2, courier, zope, clamav a podobne. V praxi to znamená, že pre každú službu bude v koreňovom adresári úložiska vytvorený samostatný adresár nesúci meno služby.

Ďalšou časťou, ktorú je nutné vhodným spôsobom premapovať z oblasti verzovania projektov je rozloženie adresárov trunk, branches a tags. Aj v tomto prípade budeme rešpektovať logiku odporúčaného rozloženia adresárov. Jedinú úpravu, ktorú je nutné vykonať pre potreby ukladanie konfigurácie, je priradenie doplňujúceho významu adresárov. Je nutné si uvedomiť, že chceme mať k dispozícii štruktúru, umožňujúcu nakonfigurovať procesy poskytujúce službu v sieti. Služba môže byť poskytovaná viacerými procesmi bežiacimi na rôznych zariadeniach.

- trunk - Sem je ukladaná všeobecná konfigurácia pre proces. Od týchto súborov je potom odvodená špecifická konfigurácia pre jednotlivé zariadenia v sieti.
- branches - Sem sa ukladá špecifická konfigurácia procesu určená pre konkrétne zariadenie. Ako meno adresára je vhodné zobrať plné DNS meno.
- tags - Do tohoto adresára je vhodné ukladať odladené verzie konfigurácie, ktoré sa ďalej nebudú meniť. Jedná sa o adresár s archívami.

Pre lepšie pochopenie rozloženia je vhodné uviesť príklad. Zoberme do úvahy sieť, ktorá pozostáva z troch počítačov Angel1, Angel2, Challenger. Angel1 a Angel2 majú slúžiť ako poštové servery. Ako poštového démona použijeme Courier. Challenger slúži ako adresárový server so službou LDAP. Doména pre tieto servery je earthsearch.org. Štruktúra úložiska vyzerá nasledovne:

```
/
courier/
  trunk/
    courier-config-file
  branches/
    angel1.earthsearch.org/
    angel2.earthsearch.org/
  tags/
    frozen-01-angel1.earthsearch.org/
    frozen-02-angel1.earthsearch.org/
ldap/
  trunk/
    ldap-config
  branches/
    challenger.earthsearch.org/
```

Do takto navrhutej štruktúry je možné ukladať konfiguráciu. Zastavme sa pri systéme voľby mena vetvy v adresári branches. Existuje niekoľko možností ako zvoliť meno adresára. Prvou možnosťou je zvolenie mena rovnakého ako meno zariadenia zanesené v DNS záznamoch. Táto možnosť je priamočiara a v prípade zmeny IP adresy zariadenia. Druhou možnosťou je použiť ako meno IP adresu jedného z rozhraní konfigurovaného zariadenia. Takýto prístup síce nevyžaduje žiadny DNS záznam, avšak implikuje zvýšenú réžiu pri presunoch zariadení medzi segmentami siete. Prípadne treťou možnosťou je využiť vlastné pomenovanie. V tom prípade je nutné, aby systém návrhy mien bol známy a aby sa predišlo duplikácii a kolízii mien.

### 3.4.3 Archivácia oprávnení v Subversion

Subversion je navrhnuté platformovo nezávisle a s úložiskom dát je možné pracovať prostredníctvom rôznych klientov. Cieľom Subversion je poskytnúť kvalitný systém pre správu verzií. Pre naše potreby verzovania konfigurácie však potrebujeme viac než len rôzne verzie textových alebo binárnych konfiguračných súborov. Pri konfigurácii zariadenia, na ktorom beží viacero rôznych služieb je nutné, aby sa boli do verzovacieho systému ukladané aj informácie o oprávneniach na súbory.

Momentálne existujú dve použiteľné riešenia umožňujúce ukladanie a načítavanie oprávnení do Subversion. Jedná sa o asvn[23] a fsvs (File System VerSioning)[24]. Obe riešenia využívajú vlastnosť „Properties“, jedná

sa o meta dáta, ktoré je možné ku každému verzovanému objektu priložiť. Konceptuálne sú vlastnosti (properties) riešené jednoducho. K menu vlastnosti je priradená hodnota. Meno vlastnosti pozostáva z dvoch častí oddelených dvojbodkou. Prvá časť predstavuje menný priestor (namespace<sup>2</sup>) a druhá časť predstavuje samotné meno premennej (napr.: svn:executable - indikuje spustiteľnosť súboru). Hodnotou môže byť text, ale aj súbor. Ak je verzovaným objektom súbor s obrázkom, je možné ako jeho vlastnosť pridať obrázok s náhľadom. Veľmi dôležitou vlastnosťou týchto meta dát je, že tak ako obsah súboru, aj oni podliehajú verzovaniu.

Nástroj asvn je malý nástroj napísaný ako shell skript. Vznikol v roku 2003. Jedná sa o jednoduché obalenie nástroja svn volaného z príkazového riadku. Pri práci s oprávneniami môžeme rozlíšiť dva smery: ukladanie a načítavanie informácií zo Subversion. Ukladanie prebieha tak, že skript k verzovanému súboru pridá meta dáta vo forme textového reťazca zachytávajúceho oprávnenia na súbor. Pri načítavaní súboru z úložiska, sú tieto meta dáta analyzované a príslušné oprávnenia sú aplikované na súbor. Do reťazca priradeného k vlastnosti file:permissions, sú zachytené: práva, vlastník, skupina, typ. Nevýhodou riešenia je, že skript je viazaný na prostredie operačného systému Linux a podporuje len základný model unixových oprávnení.

Komplexnejší program fsvs vychádza z rovnakého princípu a síce obaľuje nástroje pre prácu s úložiskom, vznikol v roku 2005. Fsvs je implementovaný v jazyku C a umožňuje ukladať nasledujúce informácie: čas, vlastníka, skupinu a oprávnenia. Je silne viazaný na operačný systém Linux.

Pre podporu oprávnení v nástroji Borax je použitý podobný princíp ako pri nástroji asvn. S tým rozdielom, že informácie o oprávnení nie je ukladaná do jedného reťazca, ale je rozdelené medzi viac vlastností (meta dáta). Dôvodom k rozdeleniu je, že jednotlivé oprávnenia môžu byť spravované samostatne a navyiac je možné jednoducho rozšíriť podporované modely oprávnení. Pre potreby definovania viacerých vlastností je vytvorený samostatný menný priestor „borax“, čím sa zamedzí kolízii mien vlastností z iných nástrojov.

Pre podporu unixových oprávnení sú zadefinovaná schéma nasledujúcich vlastností:

```
borax:model    = unix
borax:uid      = uid/user_name
borax:gid      = gid/group_name
```

2. Použitie namespace nie je v Subversion povinné, je však silne doporučované. Dôvodom je prechádzanie kolíziám mien. Používa sa rovnaký princíp ako pri XML.

```
borax:access = -rwxrwxrwx
borax:atime  = date time
borax:ctime  = data time
borax:mtime  = date time
```

Vlastnosť model určuje o aký model oprávnení sa jedná, ku každému modelu je určený zoznam vlastností a formát ich zápisu. Zápis ostatných vlastností vychádza zo syntaxe GNU nástroja `stat`[28] pre určovanie atribútov súboru. Dôležitou požiadavkou, ktorú je nutné zohľadniť pri tvorbe schém pre zachytenie atribútov ďalších modelov oprávnení, je čitateľnosť záznamov pre človeka. Administrátor musí byť schopný s minimálnou námahou prečítať všetky zachytené vlastnosti.

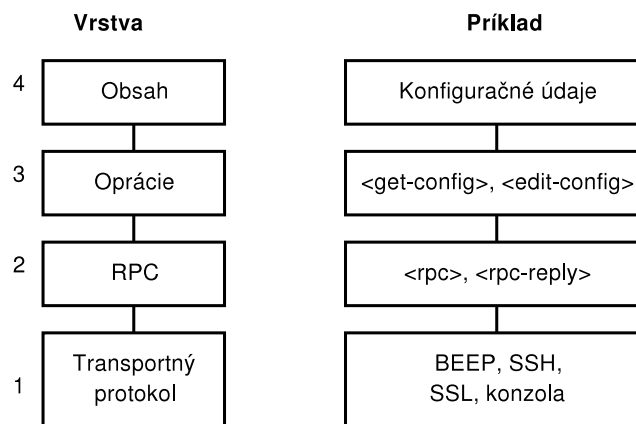
### 3.5 Protokol Netconf

Netconf protokol má slúžiť na distribuovanie konfigurácie v sieti, jedná sa o protokol postavený na XML využívajúci už existujúce vrstvy ako SSH, SOAP a ďalšie. Na tvorbe tohoto protokolu sa podieľajú ľudia z firiem ako Siemens, Cisco a Nokia. Netconf je v dobe písania tejto práce špecifikovaný ako draft a v decembri 2006 sa očakáva podanie špecifikačných dokumentov pre Netconf ako návrh na štandard. V dokumente NETCONF Configuration Protocol[17], je protokol charakterizovaný nasledovne:

Protokol Netconf definuje jednoduchý mechanizmus, prostredníctvom ktorého je možné spravovať zariadenia v sieti, je možné ním prenášať konfiguračné údaje. Umožňuje aplikovanie novej konfigurácie a manipuláciu s ňou. Protokol umožňuje zariadeniam definované aplikačné rozhranie. Aplikácia môže využívať toto rozhranie na priame prijímanie všetkých alebo časti konfiguračných informácií.

Netconf je možné konceptuálne rozdeliť do štyroch vrstiev znázornených na obrázku 3.5. Význam vrstiev je nasledovný:

1. Vrstva transportného protokolu umožňuje komunikáciu medzi klientom a serverom. Netconf je možné tým pádom postaviť nad protokolmi ako https a podobne.
2. Jednoduchá vrstva umožňujúca oddelenie RPC funkcionality od transportnej vrstvy.
3. Operačná vrstva definuje množinu operácií, ktoré je možné prostredníctvom RPC vyvolať.



Obrázok 3.5: Netconf vrstvy

4. Poslednou vrstvou je vrstva, ktorá udržiava konfiguračné údaje. Formát a obsah údajov, je už mimo rozsahu protokolu Netconf. Konkrétne vhodné formáty pre ukladanie môžu byť predmetom ďalšieho výskumu.

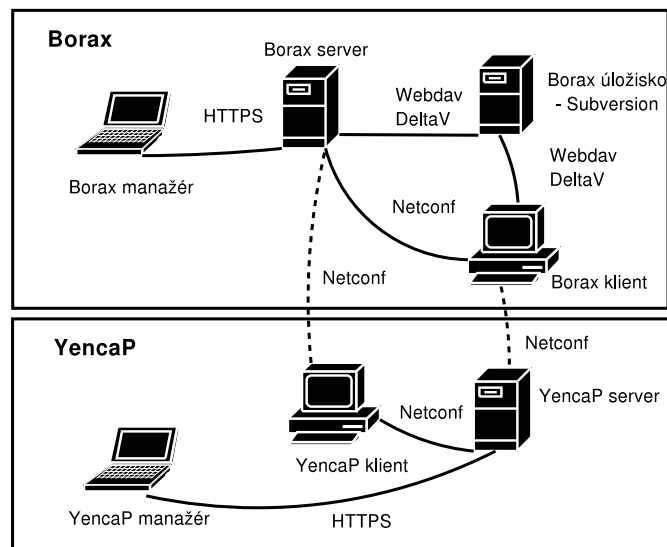
Pre ilustráciu je vhodné uviesť príklad požiadavky zaslanej pomocou protokolu Netconf. XML zápis v ukážke znamená, že konfigurácia uzatvorená značkou „config“ bude uložená do súboru „/etc/courier/sizelimit“:

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <copy-config>
    <target>
      <url>file:///etc/courier/sizelimit</url>
    </target>
    <source>
      <config>
        100000
      </config>
    </source>
  </copy-config>
</rpc>
```

### 3.5.1 Použitie Netconf v Borax II.

### 3.5.2 Interoperabilita Borax II. a Yencap

Vzhľadom na to, že riešenie Borax II aj Yencap používajú na komunikáciu rovnaký protokol, je do určitej miery previesť ich integráciu. Obidve riešenia sú momentálne vo vývojovej fáze. Vzájomá spolupráca riešení môže



Obrázok 3.6: Spolupráca medzi Borax - YencaP

byť veľkým prínosom pre testovanie a optimalizáciu komunikácie. Jedná sa o situácie, kde je využívaný na komunikáciu protokol Netconf a síce:

- YencaP server – Borax klient
- Borax server – YencaP client

Koncept vzájomnej spolupráce dvoch rôznych riešení je naznačený na obrázku 3.6. Interoperabilita je zaujímavá hneď z niekoľkých hľadísk. Borax má implementované niektoré rozdielne vlastnosti než YencaP. Tou hlavnou je integrácia Subversion ako zdroja, z ktorého môže Borax klient načítavať konfiguráciu. Ďalšou zaujímavou oblasťou je sledovanie do akej miery je dodržaná špecifikácia protokolu Netconf. Dodržanie špecifikácie je dôležité a do hlavne z dôvodu, že sa môžu (a to je aj zámerom Netconf) objaviť zariadenia od tretích strán, ktoré budú podporovať Netconf a bude ich nutné integrovať do siete.

### 3.6 PDCA a Borax

Na začiatku práce bol spomenutý model PDCA (Plan, Do, Check, Act). Tento model bol následne aplikovaný na integráciu riešení Webmin a Cfengine do siete. Aby sme zachovali konzistentnosť práce, použijeme rovnakú metodiku práce aj na Borax II. Takto ukážeme ďalšiu možnosť implementácie

PDCA do praxe a poskytneme vodítko pre riadenie nasadenia riešenia Borax<sup>3</sup>.

Budeme postupovať v štyroch krokoch. Vytvoríme plán, doplníme detaily k jeho realizácii. Doplníme informácie dotýkajúce sa údržby a cyklus uzatvoríme poukázaním na spôsob, akým je možné získať závery pre zlepšenie fungovania.

### 3.6.1 Plán nasadenia Borax

Prvým krokom pri integrácii riešenia Borax do siete je vytvorenie plánu. Budeme postupovať podobne ako v prípade aplikácie Cfengine. Stanovíme si, na ktorom zariadení v sieti pobeží Borax server. Určíme zariadenia, kam je nutné nainštalovať Borax klient, aby načítaval konfiguráciu zo servera.

Pri harmonizácii nového riešenia a bezpečnostnej politiky, môžeme zobrať do úvahy fakt, že všetka komunikácia medzi manažérskym rozhraním, serverom a klientom je šifrovaná. Pri prístupe na úložisko implementované pomocou subversion je tým pádom vhodné použiť protokol https.

Ďalej je nutné stanoviť kritéria, na základe ktorých môžeme hodnotiť úspešnosť a prínos integrácie riešenia Borax do siete. Môžeme využiť kritéria stanovené pre aplikácie Webmin a Cfengine, prípadne doplniť ďalšie ako:

- Počet situácií, kedy bolo nutné použiť návrat k prechádzajúcej verzii konfigurácie bol ...
- Počet situácií, kedy bol úspešne aplikovaný automatický roll back konfigurácie bol ...

### 3.6.2 Realizácia nasadenia Borax

Samotnú realizáciu odporúčam prevádzať v dvoch krokoch. Tu sa využije vlastnosť jednotlivých vrstiev popísaných v architektúre. Prvým krokom je uloženie už existujúcej konfigurácie do úložiska pomocou Borax nástrojov. Tým pádom riadiaca vrstva je nahradená administrátorom. Akonáhle je konfigurácia uložená v Subversion, je možné prísť k druhej časti a pridať k dátovej vrstve riadiacu vrstvu. To znamená spojenie Borax klientov so serverom. Ako finálny krok pri realizácii by malo byť napojenie Borax server na manažérske rozhranie a zdefinovanie prístupov k jednotlivým službám.

---

3. Veľmi podobný postup je možné použiť aj na integráciu riešenia YencalP.

### 3.6.3 Údržba prostredníctvom Borax

Tak ako Webmin ani Borax nie je monitorovací nástroj. Tým pádom je vhodné na podporu a zber informácií o systémoch použiť nástroje ako Logwatch a Nagios.

Vo fáze údržby je vhodné venovať pozornosť úložisku konfigurácie na serveri. Aj napriek tomu, že Subversion je dimenzované na veľké projekty, je vhodné sledovať výkon úložiska. Tu je dobré poznamenať, že autori odporúčajú používať na takéto účely ako back end k Subversion FSFS a nie BerkelyDB.

### 3.6.4 Sumarizácia návrhov na zlepšenie

V tomto kroku je nutné vyhodnotiť kritériá stanovené v pláne a zosumarizovať informácie o výkone a chybách celého riešenia. Týmto krokom uzatvoríme jeden cyklus nasadenia. V ďalšom cykle je nutné zohľadniť zistené kritické chyby a implementovať navrhované vylepšenia pre zlepšenie výkonu.

### 3.6.5 Vyhodnotenie Borax II.

Tak ako v prípade riešenia Borax I., aj teraz použijeme porovnanie riešenia proti architektúre definovanej na začiatku tejto kapitoly:

- Jednoduchosť. Dodržaná. Borax II. do veľkej miery využíva už existujúce osvedčené vrstvy a nástroje. Tým pádom sa neduplikuje implementácia. Dobrým príkladom je využitie systému pre správu verzii Subversion.
- Rozšíriteľnosť. Dodržaná. Manažérske rozhranie, server aj klient je možné pomocou modulov rozšíriť o novú funkcionality. Moduly je možné taktiež odoberať a tým pádom vyoptymalizovať riešenie pre potreby dané situáciou.
- Komunikačná architektúra manažér, server, klient. Dodržané. Výsledky zodpovedajú riešeniu Borax I.
- Prenos konfigurácie nad nedôveryhodnou sieťou. Dodržané. Boli použité protokoly podporujúce šifrovanie.
- Separácia spravovaných služieb medzi viacerých administrátorov. Dodržané.

- Verzovanie konfigurácie. Dodržané. V tomto prípade bolo nutné rozšíriť význam adresárovej štruktúry úložiska v Subversion. Taktiež bolo nutné pridať definíciu nových vlastností (properties) potrebných pre ukladanie konfigurácie. Tieto vlastnosti sú udržiavané spolu s verzovanými objektami a prostredníctvom nástrojov pre Subversion sú dostupné.
- Plánovanie konfigurácie. Dodržané. Manažérske rozhranie podporuje stanovenie času, kedy bude ktorá verzia konfigurácie aplikovaná.
- Podpora pre atomickú konfiguráciu a rollback. Čiastočne dodržané. Dôvodom je problematické detekovanie chyby. V niektorých situáciách dochádza k problémom, ktoré nie je možné súčasnou verziou skriptov odstrániť a je nutné previesť nápravu ručne. S počtom testovaných situácií sa však darí tento problém postupne eliminovať.

### 3.7 Porovnanie dostupných riešení

V tejto kapitole sme si predstavili všeobecnú architektúru pre vytvorenie systému určeného na správu siete. Bola analyzovaná implementácia riešenia Borax I. a boli vyvodené závery súvisiace s problémami pri rozširovaní. Následne bolo predstavené druhé riešenie Borax II. Rozšírenie tohoto riešenia spočívalo v pridaní systému pre správu verzií Subversion a použitie protokolu Netconf na riadenie distribúcie konfigurácie.

Pre dokreslenie situácie, je vhodné uviesť porovnanie rôznych riešení a ich vlastností. Porovnanie sa nachádza v tabuľke 3.2.

	Webmin	Cfengine	YencaP	Borax I.	Borax II.
web rozhranie	áno	nie	áno	áno	áno
komunikácia server-target	zápis do súborov, rpc		Netconf	https, xml-rpc	Netconf
komunikácia manažér-server	https	priama úprava súborov	https	https	https
verzovanie konfigurácie	nie	nie	nie	nie	áno - Subversion
delegovanie právomocí	áno	áno	áno	áno	áno
server push	áno	nie	áno	nie	áno
klient pull	nie	áno	áno	áno	áno
jazyk	Perl, Python, C, C++	C	Python	Python	Python
podporované platformy	Linux	nezávislé	Linux	Linux	nezávislé
služby	väčšina služieb	väčšina služieb	route, interface, BGP, RIP, Asterisk	Courier-MTA	Courier-MTA, Apache
server dokáže zistiť stav klienta	áno	áno	áno	nie	áno

Tabuľka 3.2: Porovnanie nástrojov pre správu a konfiguráciu siete.

## Záver

Predkladaná práca priniesla rozobratie problematiky správy sietí z pohľadu manažmentu. Bol navrhnutý použitie modlu PDCA (Plan, Do, Check, Act), ktorý by mal byť aktívne využívaný pri integrácii nových riešení do siete. PDCA je model, ktorého cieľom je dosiahnutie neustáleho zlepšovania. V súvislosti so zavádzaním zmien do siete bolo nutné spomenúť aj faktory, ktorými zmena ovplyvňuje chovanie ľudí. Čo má priamy dopad na manažment siete ako celku.

Od teoretickej oblasti zameranej na manažment sa práca presunula k analýze troch riešení. U prvých dvoch bola dôsledne ukázaný spôsob použitia PDCA pri implementácii riešenia do siete. Prvým predstaveným riešením bol Webmin, ktorý umožňuje spravovať sieť prostredníctvom web rozhrania. Druhým riešením bol Cfengine, ktorý je zameraný hlavne na automatizáciu procesu správy siete. U oboch riešení boli vybrané kľúčové koncepty, ktoré boli podstatné pre ďalšiu prácu. Tretím riešením predstaveným na záver bolo riešenie podporujúce protokol Netconf, jednalo sa o nástroj Yencap.

Tretia kapitola práce bola venovaná riešeniu Borax, ktoré som navrhol a implementoval. Prvým krokom bolo predstavenie generickej architektúry systému pre správu siete. Bola rozobratá implementácia Borax I. a bolo poukázané na nedostatky. Borax I. bol konfrontovaný s popisom generickej architektúry. Následne bolo predstavené riešenie Borax II. a jeho komponenty, ktoré považujem za podstatné rozšírenie. Jedným z rozšírení bolo Subversion a druhým použitie protokolu Netconf. Nakoniec boli všetky spomínané riešenia porovnané na základe svojich vlastností.

Práca splnila svoj účel. Boli identifikované riešenia zjednodušujúce správu rozsiahlej siete, bola vytvorená implementácia jedného z riešení a taktiež bola ukázaný postup, ktorý je možné využiť pri manažmente siete.

## Literatúra

- [1] Nancy R. Tague: Quality toolbox. ASQ Quality Press, Milwaukee, Wisconsin, 2005.
- [2] Change Management Toolbook: <http://www.change-management-toolbook.com/>
- [3] PDCAuditing – Quality Management in the 21st Century: <http://www.pdcauditing.com>
- [4] Webmin: <http://www.webmin.com/>
- [5] Jamie Cameron: Managing Linux Systems with Webmin. Pearson Education Inc., United States of America, 2004.
- [6] Joe Cooper: The Book of Webmin. No Starch Press, United States of America, 2003.
- [7] Logwatch: <http://www.logwatch.org/>
- [8] Kirk Bauer: Automating UNIX and Linux Administration. Apress Inc., 2003.
- [9] Nagios: <http://www.nagios.org/>
- [10] Cfengine: <http://www.cfengine.org/>
- [11] Mark Burgess: Cfengine concepts. Faculty of Engineering, Oslo University College, Norway, 2005.
- [12] Mark Burgess: Anomaly detection with cfenvd. Faculty of Engineering, Oslo University College, Norway, 2005.
- [13] Cfengine Wiki: [http://cfwiki.org/cfwiki/index.php/Main\\_Page](http://cfwiki.org/cfwiki/index.php/Main_Page)
- [14] EnSuite platform, YencaP: <http://libresource.inria.fr/projects/ensuite>
- [15] Vincent Cridlig, Radu State: YencaP Documentation. Madynes Research team, LORIA-INTRIA Lorraine, Francúzsko, 2005.

- [16] Netconf: <http://www.ietf.org/html.charters/netconf-charter.html>
- [17] Netconf IETF Draft: <http://www.ietf.org/internet-drafts/draft-ietf-netconf-prot-12.txt>
- [18] CVS: <http://www.nongnu.org/cvs/>
- [19] ClearCase: <http://www-306.ibm.com/software/awdtools/clearcase/>
- [20] Git: <http://git.or.cz/>
- [21] Subversion: <http://subversion.tigris.org/>
- [22] Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato: Version Control with Subversion. O'Reilly, 2004.
- [23] Asvn: <http://svn.collab.net/repos/svn/trunk/contrib/client-side/asvn>
- [24] FSVS: <http://fsvs.subversion.tigris.org/>
- [25] RFC 1925 – The Twelve Networking Truths:  
<http://www.faqs.org/rfcs/rfc1925.html>
- [26] RFC 2518 - HTTP Extensions for Distributed Authoring – WEBDAV:  
<http://www.faqs.org/rfcs/rfc2518.html>
- [27] RFC 3253 – Versioning Extensions to WebDAV (Web Distributed Authoring and Versioning): <http://www.faqs.org/rfcs/rfc3253.html>
- [28] GNU Coreutils: <http://www.gnu.org/software/coreutils/>