České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Katedra matematiky
Obor: Inženýrská informatika
Zaměření: Tvorba software

Systém pro generování konfiguračních souborů
System for generating of configuration files

Diplomová práce

Vypracoval: Jan Ehrenberger
Vedoucí práce: Ing. Tomáš Oberhuber
Rok: 2011

Před svázáním místo téhle stránky vložíte zadání práce s podpisem děkana (bude to jediný oboustranný list ve Vaší práci) !!!!

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne .....

.....
Jan Ehrenberger

Poděkování

Děkuji Ing. Tomáši Oberhuberovi za vedení mé diplomové práce a za podnětné návrhy, které ji obohatily. Dále bych chtěl poděkovat Davidu Fabianovi za spolupráci na projektu Freeconf.

Jan Ehrenberger

Název práce:
Systém pro generování konfiguračních souborů

Autor: Jan Ehrenberger

Obor: Inženýrská informatika
Druh práce: Diplomová práce

Vedoucí práce: Ing. Tomáš Oberhuber
Katedra matematiky, Fakulta jaderná a fyzikálně inženýrská, České vysoké učení technické v Praze

Abstrakt: Tato práce se zabývá projektem Freeconf, který se snaží o zjednodušení nastavení programů operačních systémů typu Open Source. Konfigurované programy jsou popsány sadou souborů ve formátu XML, jež tvoří takzvaný konfigurační balík. Pomocí nástroje s grafickým uživatelským rozhraním pak může uživatel z konfiguračního balíčku snadno vygenerovat soubory nastavení jednotlivých programů s požadovanými hodnotami. První část práce se věnuje zpracování konfigurace webového serveru Apache a implementaci změn v projektu, které si konfigurace serveru vyžádala. Druhá část je věnována návrhu rozšíření konfigurace pomocí spustitelných skriptů.

Klíčová slova: konfigurace Apache webový server skriptování

Title:
System for generating of configuration files

Author: Jan Ehrenberger

Abstract: This work deals with project Freeconf which tries to simplify configuration proces of applications in Open Source operating systems. Applications are described by a set of files in XML format which forms so called configration package. Using a tool with graphical user interface which is part of project Freeconf one can easily generate configuration files of individual applications with desired values. First part of this work deals with configuration of Apache web server and describes implementation of changes that were necessary for support of server's configuration. Second part focuses on design of enhancing configuration with executable scripts.

Key words: configuration Apache web server scripting

Obsah

Úvod

Ve světě operačních systémů unixového typu jsou velmi rozšířeným prostředkem pro správu nastavení programu textové konfigurační soubory. Tento způsob ukládání nastavení má mnoho praktických výhod. Vyniká zejména snadnou editovatelností uživatelem, který může konfigurační soubory upravovat libovolným textovým editorem. Lze je jednoduše generovat i zpracovávat automatizovanými skripty bez nutnosti používat speciální systémové knihovny. Textové soubory jsou také odolnější proti drobným změnám formátu a tedy i snadněji přenositelné mezi jednotlivými verzemi programu.

Bohužel tento přístup má i své stinné stránky. Formáty konfiguračních souborů pro jednotlivé programy se mezi sebou značně liší. Uživatelé a správci systému si všechny možnosti konfigurace zpravidla nepamatují a jsou tak často nuceni sahat k dokumentaci. Ruční zadávání hodnot nastavení do konfiguračních souborů sebou také nese riziko lidské chyby. Překlep při editaci se pak může projevit například nečekaným chováním systému, nebo pádem programu.

Typickým protikladem textových konfiguračních souborů je binární registr nastavení operačního systému Windows. Je centralizovaný a spravovaný operačním systémem. Všechny programy k němu mohou přistupovat voláním systémových funkcí. Uživatelé jej mohou měnit pouze prostřednictvím dialogů konfigurovaných programů, nebo systémovým nástrojem regedit. Přenést nastavení mezi různými verzemi operačního systému nebo různými počítači je pak velmi komplikované, ne-li nemožné.

Cílem systému pro generování konfiguračních souborů, který nazýváme *Freeconf*, je vytvořit nadstavbu nad již existujícím systémem textových konfiguračních souborů. Jde o jakousi mezivrstvu, která poskytuje jednotné a univerzální rozhraní ke konfiguraci operačního systému. Zároveň do něj ale nijak nezasahuje, pouze přidává nové funkce. Díky tomu je možné vytvořit sadu nástrojů, které s nastavením systému pracují, aniž by se staraly o formát a umístění jednotlivých konfiguračních souborů.

V této práci jsme se pokusili zjistit, zda je náš systém schopen popsat všechna nastavení webového serveru Apache. Jde o poměrně rozsáhlou aplikaci s netriviální konfigurací a tak si vyžádala mnohé změny v návrhu projektu. Těmto změnám je věnována kapitola 3. Výsledné rozhraní konfigurace jsme v kapitole 5 porovnali se dvěma existujícími nástroji na konfiguraci serveru Apache.

Dalším úkolem, kterému jsme se věnovali, bylo rozšíření projektu o podporu spustitelných skriptů. Ty by měly být schopné za běhu měnit vlastnosti konfigurace a automaticky detekovat některé hodnoty. Nastavení programů by se tak mohlo samo

přizpůsobovat cílovému systému. Je to další krok na cestě k snazší a uživatelsky přívětivější konfiguraci operačních systémů. Návrh podpory spustitelných skriptů je obsahem kapitoly 4.

Kapitola 1

Seznámení s projektem Freeconf

1.1 Konfigurační systémy

Nejdříve se podíváme na některé existující konfigurační systémy, na které se budeme v práci odkazovat, a krátce si popíšeme jejich nejvýznamnější vlastnosti.

1.1.1 Webmin

Webmin je aplikace pro dálkovou správu serverů s operačními systémy typu Unix. Aplikace se spustí jako malý webový server na cílovém počítači. Uživatel se k němu pak může připojit vzdáleně pomocí webového prohlížeče. Celá konfigurace tedy probíhá přes webové stránky. Webmin takto umožňuje provádět většinu běžných administračních úkonů. Podporuje například správu uživatelských účtů, nastavení DNS[1], sdílení souborů, webového serveru Apache a mnoho dalšího. Nastavení programů se ukládá přímo do jejich konfiguračních souborů, hodnoty zobrazované aplikací Webmin jsou tedy vždy aktuální.

Poněkud horší je rozšiřitelnost aplikace o konfiguraci nových programů. K tomu je totiž potřeba znalost programování, CGI[2] skriptů, programovacího jazyka Perl a interních knihoven aplikace Webmin.

1.1.2 Konfigurační nástroje distribucí operačního systémů GNU Linux

Operační systém GNU Linux je k dispozici v mnoha podobách a verzích. Ucelený instalační balík operačního systému a přidružených aplikací se obvykle označuje jako distribuce. Většina těchto distribucí také obsahuje konfigurační nástroje, které

[1] DNS (Domain Name System) je hierarchický systém doménových jmen. Jeho hlavním úkolem jsou vzájemné převody doménových jmen a IP adres v síti.

[2] CGI (Common Gateway Interface) je protokol, který webovému serveru umožňuje přesměrovat požadavek od klienta na externí aplikaci.

umožňují nastavení nejvýznamnějších aplikací a služeb operačního systému pomocí GUI[3]. V kapitole 5.2 si představíme jeden z těchto nástrojů, který slouží k nastavení webového serveru Apache v distribuci Fedora.

1.1.3 Linux Kernel Config

Tento konfigurační nástroj slouží k nastavení voleb jádra operačního systému Linux před překladem ze zdrojových souborů do strojového kódu. Ačkoli se jedná o úzce specializovaný nástroj, je v mnoha rysech podobný nástrojům projektu Freeconf. Konfigurační volby jádra jsou popsány speciálně formátovaným textovým souborem, který lze načíst pomocí několika konfiguračních aplikací. K dispozici je například aplikace pro konfiguraci z GUI nebo aplikace pro zadávání voleb v textovém režimu. Také řešení závislostí mezi jednotlivými volbami je v jádru Linux řešeno podobně, jako v projektu Freeconf.

1.1.4 Freeconf

Projekt Freeconf, jemuž se věnujeme v této práci, tvoří sada znovupoužitelných konfiguračních nástrojů a jednotné rozhraní pro popsání nastavení nejrůznějších programů. V projektu se snažíme o splnění následujících cílů:

1. **Jednoduchost**: Chceme eliminovat zbytečné a opakující se kroky, které se vyskytují během konfigurace. Zároveň chceme zachovat i celkovou jednoduchost a přehlednost řešení.

2. **Sjednocení**: Konfigurační proces různých aplikací chceme sjednotit a zpřístupnit pomocí unifikovaného rozhraní.

3. **Použití XML**: V celém návrhu se snažíme maximálně využívat vlastností technologie XML[4].

4. **Rozšiřitelnost**: Proces rozšíření projektu o popis konfigurace nového programu by měl být snadný, rychlý a měl by se obejít bez znalosti programování.

5. **Schopnost popsat co nejvíce formátů konfigurace**: Naše řešení by mělo být schopné popsat co možná největší počet různých konfiguračních formátů.

6. **Přenositelnost**: Konfiguraci by mělo být možné přenést z jednoho počítače na jiný bez nutnosti velkých změn. Zároveň chceme, aby byly nástroje projektu schopné fungovat na různých operačních systémech.

7. **Škálovatelnost**: Projekt Freeconf by měl být schopen popsat konfiguraci jak malých programů tak i rozsáhlých systémů.

[3] GUI (Graphical User Interface) značí grafické uživatelské rozhraní.

[4] XML(eXtensible Markup Language) je obecný standardizovaný značkovací jazyk. Více o něm je možné nalézt v dokumentaci [16].

8. **Nezasahovat do operačního systému**: Nechceme zasahovat do operačního systému ani jiných programů. Projekt Freeconf by měl fungovat čistě jen jako nadstavba nad existujícími programy.

9. **Konfigurace pomocí GUI**: Konfigurace by měla probíhat z GUI bez nutnosti přímé editace konfiguračních souborů.

Ve zbytku této kapitoly se v hrubých rysech seznámíme s projektem Freeconf. Popíšeme si stav projektu v době začátku psaní této práce. Veškeré změny provedené ve struktuře projektu a formátech souborů, které byly provedeny v rámci této práce, budou podrobně rozebrány v dalších kapitolách.

1.2 Základní pojmy

Zde je seznam základních pojmů, jejichž znalost budeme potřebovat pro porozumění následujícího textu:

- **cílový program**: Program, který nastavujeme pomocí projektu Freeconf.
- **koncový uživatel**: Uživatel, který nastavuje cílový program.
- **konfigurační aplikace**: Aplikace projektu Freeconf, prostřednictvím které koncový uživatel nastavuje cílový program. V textu je někdy totéž označováno jako klientská aplikace.

1.3 Klíčová slova

Základní částí konfigurace v projektu Freeconf je klíčové slovo. Reprezentuje nejmenší jednotku konfigurace, jako například zapnutí/vypnutí určité vlastnosti programu, cesta k souboru, atd. Klíčové slovo je identifikováno svým názvem a jeho hodnotu zadává uživatel.

Klíčová slova jsou sdružena do logických celků zvaných sekce. Každá sekce také může obsahovat další (vnořené) sekce a nastavení programu tak lze organizovat do stromové struktury. Sekci, která není obsažena v žádné jiné sekci nazýváme kořenová sekce. Taková je v konfiguraci právě jedna.

K jednoznačné identifikaci klíčového slova tedy potřebujeme znát jeho název a název sekce, v které je obsaženo. K identifikaci této sekce pak potřebujeme znát i název jí nadřazené sekce a takto lze pokračovat až ke kořenové sekci, která je v konfiguraci vždy jen jedna a je tedy jednoznačně dána. Toto se velmi podobá organizaci souborů do adresářů v souborových systémech. Na klíčové slovo ve stromě konfigurace se tedy odkazujeme podobně jako na soubor v souborovém systému, pomocí textového řetězce, který nazýváme cestou ke klíčovému slovu:

```
/Kořenová sekce/Vnořená sekce/.../Klíčové slovo
```

1.3.1 Typy

V současnosti jsou podporovány čtyři typy klíčových slov:

bool

Typ *bool* představuje booleovskou hodnotu. Klíčová slova tohoto typu mohou nabývat jednoho ze dvou stavů: zapnuto (`yes`) a vypnuto (`no`).

number

Používá se k uložení číselné hodnoty s plovoucí desetinnou čárkou. Klíčová slova tohoto typu mají několik vlastností, pomocí kterých lze snadno omezit rozsah povolených hodnot:

- **min**: Minimální povolená hodnota čísla.
- **max**: Maximální povolená hodnota čísla.
- **step**: Krok číselného klíče. Určuje, o jaký krok může uživatel číslo zvětšit nebo zmenšit.
- **precision**: Počet desetinných míst čísla. Pokud je tato vlastnost nastavena na 0, dostaneme celé číslo.
- **print-sign**: Umožňuje vynutit vypisování znaménka + nebo − před číselnou hodnotou.
- **leading-zeros**: Určuje počet znaků, na které se bude číselná hodnota zarovnávat při výpisu. Zarovnání bude provedeno znakem „0“.

string

Klíčová slova typu *string* se používají k reprezentaci libovolné hodnoty, kterou lze zapsat jako nějakou posloupnost znaků. Například jméno souboru, emailová adresa, popiska, atd. U slov tohoto typu lze definovat následující vlastnosti:

- **regexp**: Regulární výraz, kterému musí hodnota vyhovovat.
- **data**: Jméno seznamu řetězců. Pokud je tato vlastnost nastavena, omezuje rozsah povolených hodnot na hodnoty obsažené v seznamu. Více o seznamech hodnot nalezneme v kapitole 1.4.7.

fuzzy

Klíčová slova typu *fuzzy* mohou nabývat jedné z předdefinovaných logických hodnot. To nám umožňuje implementovat logiku na obecně větší než dvoustavové množině zapnuto/vypnuto.

U klíčových slov tohoto typu je třeba nastavit jednu povinnou vlastnost:

- **data**: Jméno seznamu fuzzy hodnot. Každá hodnota v seznamu má definovaný stupeň příslušnosti do fuzzy množiny.

Více o fuzzy hodnotách lze nalézt v práci [5].

1.3.2 Obecné vlastnosti

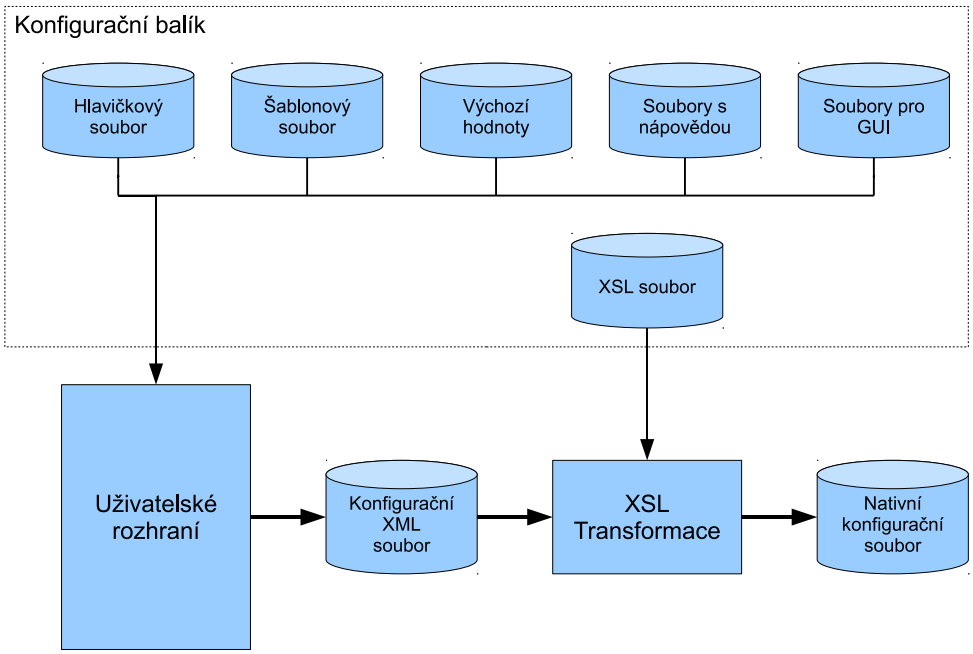
Pro klíčová slova všech typů lze také definovat tři obecné vlastnosti. Všechny mají pouze dva stavy: zapnuto/vypnuto.

- **mandatory**: Klíče se zapnutou vlastností `mandatory` označujeme jako *povinné klíče*. Jinak jsou označovány jako *nepovinné*. U povinných klíčů musí být vždy vyplněna jejich hodnota. Pokud hodnota vyplněna není, dostane se klíč do takzvaného nekonzistentního stavu a setrvá v něm, dokud uživatel hodnotu nevyplní. Sekce obsahující nekonzistentní klíč nebo sekci je také považována za nekonzistentní. Konfigurační aplikace projektu Freeconf vynucuje vyplnění hodnot nekonzistentních klíčů zvýrazněním červenou barvou a nemožností uložit konfiguraci.
- **active**: Klíče se zapnutou vlastností `active` označujeme jako *aktivní klíče*. Zbylé klíče jsou *neaktivní*. Aktivní klíče jsou při konfiguraci zobrazeny uživateli. Neaktivní klíče zobrazeny nejsou.
- **enabled**: Klíče se zapnutou vlastností `enabled` označujeme jako *editovatelné klíče*. Ostatní klíče jsou *needitovatelné*. Hodnotu editovatelných klíčů může uživatel měnit, hodnotu needitovatelných klíčů změnit nemůže.

Více o obecných vlastnostech klíčových slov a rozdílech mezi nimi lze nalézt v práci [7].

1.4 Konfigurační balík

Struktura všech nastavení konfigurovaného programu je v projektu Freeconf popsána sadou souborů ve formátu XML, která se nazývá *konfigurační balík*. Obsahuje informace o logické struktuře konfigurace, závislostech klíčových slov, výchozí hodnoty, popis uživatelského rozhraní, v němž by měla být zobrazena konfigurace, a nápovědu ve více jazycích.



Obrázek 1.1: Konfigurační proces

Struktura konfiguračního balíku a jeho role v konfiguračním procesu je znázorněna na obrázku 1.1. Na základě informací z konfiguračního balíku je vytvořeno grafické uživatelské rozhraní (zkráceně GUI), pomocí kterého může koncový uživatel měnit nastavení cílového programu. Po ukončení konfigurace uživatelem dojde k zapsání všech nastavení do konfiguračního XML souboru. Z toho je pak pomocí XSL transformace[5] vytvořen nativní konfigurační soubor ve formátu, kterému rozumí cílový program. Použití XSL transformace je zde klíčové, neboť nám umožňuje snadno vytvořit konfigurační soubor v libovolném textovém formátu.

1.4.1 Hlavičkový soubor

V následující ukázce můžeme vidět hlavičkový soubor z konfiguračního balíku webového serveru Apache.

```
<freeconf-header>
  <content>
    <template>apache_template.xml</template>
    <default-values>apache_default.xml</default-values>
    <help>apache_help.xml</help>
    <output>apache_conf.xml</output>
    <transform>apache_transform.xsl</transform>
```

[5] XSL (eXtensible Stylesheet Language) je jazyk umožňující popsat, jak se mají XML soubory formátovat a převádět. Více o XSL transformacích nalezneme v dokumentaci [17].

```
      <native-output>apache2.conf</native-output>
      <dependencies>apache_dependencies.xml</dependencies>
      <lists>apache_lists.xml</lists>
      <gui-template>apache_gui_template.xml</gui-template>
      <gui-label>apache_gui_label.xml</gui-label>
    </content>
    <settings>
      <output-defaults>yes</output-defaults>
    </settings>
  </freeconf-header>
```

Uvnitř elementu `content` se nachází seznam odkazů na soubory obsažené v balíku. Element `settings` obsahuje globální nastavení balíku. Nyní si objasníme význam jednotlivých záznamů:

template

Jméno šablonového souboru. Podrobněji si ho popíšeme v kapitole 1.4.2.

default-values

Jméno souboru s výchozími hodnotami, více se o něm dočteme v kapitole 1.4.5.

help

Jméno souboru s nápovědou a popiskami pro uživatelské rozhraní. Soubory s nápovědou jsou umístěné v podadresáři `L10n` adresáře konfiguračního balíku. Soubor může mít více jazykových verzí, přičemž aktuální verze se hledá v podadresáři, jehož jméno je odvozeno od systémové proměnné `LANG`.

Například v systému, kde je výchozím jazykem čeština v kódování UTF-8, se soubor s nápovědou bude hledat v podadresáři `L10n/cz_CZ.UTF-8`.

transform

Jméno souboru s XSL transformací. Více o tomto souboru najdeme v kapitole 1.4.6.

output

Jméno, případně i cesta ke konfiguračnímu souboru ve formátu Freeconf. Cesta k souboru může obsahovat zástupný znak $, který se nahradí cestou k adresáři konfiguračního balíku a $\sim$, který značí domovský adresář uživatele.

native-output

Jméno a umístění nativního konfiguračního souboru. Toto je obecně libovolný textový soubor ve formátu, kterému rozumí konfigurovaný program. Cesta k souboru může také obsahovat zástupné znaky $ a ∼, stejně jako u konfiguračního souboru ve formátu Freeconf.

dependencies

Obsahuje jméno souboru se závislostmi. Jeho detailnější popis najdeme v práci [5].

lists

Element `lists` obsahuje jméno souboru se seznamy hodnot. Soubor se postupně hledá v těchto adresářích:

1. uživatelský adresář: ∼`/.freeconf/lists`
2. adresář konfiguračního balíku: podadresář `lists`
3. systémový adresář: `/usr/share/freeconf/lists`

Více o seznamech hodnot najdeme v kapitole 1.4.7.

gui-template

Jméno šablonového souboru popisujícího GUI. Ukázku a popis tohoto souboru nalezneme v kapitole 1.4.9.

gui-labels

Jméno souboru s popiskami GUI. Je umístěn v podadresáři `L10n`, stejně jako soubory s nápovědou.

1.4.2 Šablonový soubor

Šablonový soubor popisuje logickou strukturu nativního konfiguračního souboru. Zde jsou definovány sekce, klíčová slova a jejich vlastnosti.

Ukázka části šablonového souboru webového serveru Apache:

```
<freeconf-template>
  <!-- Název kořenové sekce -->
  <section-name>apache-config</section-name>
```

```
  <!-- Definice sekce NameVirtualHost -->
  <section>
    <!-- Název sekce -->
    <section-name>NameVirtualHost</section-name>

    <!-- Definice klíče typu string -->
    <string>
      <!-- Název klíče -->
      <entry-name>Address</entry-name>

      <!-- Obecné vlastnosti klíče -->
      <mandatory>no</mandatory>
      <active>yes</active>
    </string>

    <!-- Definice klíče typu number -->
    <number>
      <entry-name>Port</entry-name>

      <!-- Obecné vlastnosti klíče -->
      <mandatory>no</mandatory>
      <active>yes</active>

      <!-- Vlastnosti klíče typu number -->
      <properties>
        <!-- Hodnota je celé číslo -->
        <precision>0</precision>
        <!-- Minimální hodnota -->
        <min>0</min>
        <!-- Maximální hodnota -->
        <max>65535</max>
        <!-- Krok -->
        <step>1</step>
      </properties>
    </number>
  </section>

  <!-- ... -->
</freeconf-template>
```

Jak vidíme z ukázky, sekce definujeme XML elementem `section`. Název sekce se vyplňuje do elementu `section-name`.

Klíče definujeme elementem se stejným názvem, jaký je název typu klíče. Tedy například klíčové slovo typu *string* zapíšeme do XML elementu `string`. Název klíče vyplníme do elementu `entry-name`. Následuje seznam obecných vlastností klíčového

slova. Zde je možné uvést vlastnosti *mandatory* a *active*. Obecná vlastnost *enabled* se do šablonového souboru nezapisuje a je možné ji změnit pouze pomocí závislostí. Pokud je třeba nastavit specifické vlastnosti typu klíče, uvádí se mezi elementy `properties`.

1.4.3 Soubor s nápovědou

Systém Freeconf může uživateli nabídnout nápovědu obsahující detailní popis jednotlivých klíčových slov a jejich hodnot. Tyto informace jsou uloženy právě v souborech s nápovědou. Kromě detailního popisu obsahuje také krátké popisky klíčových slov a sekcí. Popisky by měly být co nejvýstižnější. Slouží k tomu, aby byl uživatel schopen rychle odhadnout, jaký je význam daného klíčového slova, aniž by musel číst dlouhý text nápovědy.

Jak již bylo řečeno dříve, v popisu hlavičkového souboru, nápověda může být dostupná ve více jazykových verzích.

Ukázka nápovědy pro klíčové slovo `Address`, které nastavuje IP adresu, na níž bude server přijímat požadavky pro virtuální hostitele:

```
<freeconf-help>
<section name="apache-config">
  <section name="NameVirtualHost">
    <!-- Nápověda ke klíčovému slovu Address -->
    <entry name="Address">
      <label>Adresa</label>
      <help>IP adresa (a volitelně i port), na které bude server
            přijímat dotazy na virtuální hostitele. Můžete zadat
            přímo IP adresu a nebo název počítače v síti. Pokud
            zadáte adresu typu IPv6, musí být uzavřena v hranatých
            závorkách. Pokud chcete, aby server přijímal dotazy na
            všech rozhraních, zadejte hvězdičku (*).
      </help>
    </entry>
    <!-- ... -->
  </section>
  <!-- ... -->
</section>
</freeconf-help>
```

1.4.4 Konfigurační soubor ve formátu Freeconf

V konfiguračním souboru ve formátu Freeconf jsou uloženy aktuální hodnoty nastavení programu. V zásadě se jedná o XML soubor, kde elementy `entry` zastupují klíčová slova a elementy `section` zastupují sekce. Struktura souboru částečně kopíruje strukturu šablonového souboru, neboť obsahuje stejné sekce, podsekce a

klíčová slova. Ale místo definice klíčových slov ukládá právě jejich hodnoty a další informace potřebné pro vygenerování nativního konfiguračního souboru. Tedy i nápovědu klíčových slov, kterou lze do nativního konfiguračního souboru doplnit pro větší srozumitelnost.

Následující ukázka obsahuje konfigurační soubor vygenerovaný z konfigurační struktury dané šablonovým souborem v kapitole 1.4.2:

```
<section name="apache-config">
  <section name="NameVirtualHost">
  <entry name="Address">
    <value>*</value>
    <help>IP adresa (a volitelně i port), na které bude server
          přijímat dotazy ...
    </help>
  </entry>
  <entry name="Port">
    <value>80</value>
    <help>...</help>
  </entry>
  <!-- ... -->
  </section>
</section>
```

1.4.5 Soubor s výchozími hodnotami

Soubor s výchozími hodnotami má stejný formát jako konfigurační soubor ve formátu Freeconf až na to, že jsou zde uvedeny pouze hodnoty klíčových slov. Načítá se automaticky po startu systému Freeconf, ještě před načtením konfiguračního souboru. Pokud tedy konfigurační soubor nějaké klíčové slovo ještě neobsahuje (uživatel pro něj ještě nezadal hodnotu), použije se pro něj právě výchozí hodnota.

1.4.6 Soubor s XSL transformací

Nastavení máme uloženo v XML souboru ve formátu Freeconf, kterému ale konfigurovaný program nerozumí. Potřebujeme někde definovat jakým způsobem převést konfiguraci z XML do nativního textového formátu. Místem pro tuto definici je právě soubor s XSL transformací. Kdybychom například narazili na program, který by ukládal nastavení v podobě `Klíčové slovo = hodnota`, přičemž na každém řádku by bylo jedno klíčové slovo, stačila by nám k zapsání nativního konfiguračního souboru jednoduchá transformace:

```
<xsl:stylesheet
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     version="1.0">
```

```
<!-- Výstup je v textovém formátu -->
<xsl:output method="text" indent="no"/>

<!-- Zpracování klíčových slov -->
<xsl:template match="entry">
  <!-- Vypsat jméno klíčového slova -->
  <xsl:value-of select="@name" />
  <!-- Oddělovač klíčového slova a hodnoty -->
  <xsl:text> = </xsl:text>
  <!-- Vypsat hodnotu klíčového slova -->
  <xsl:value-of select="value" />
  <!-- Vypsat nový řádek -->
  <xsl:text>&#x0A;</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

Více o XSL transformacích je možné nalézt v dokumentaci [17].

1.4.7 Soubor se seznamy hodnot

Zde jsou uloženy definice seznamů hodnot. Seznamy máme dvojího typu:

1. **Seznamy řetězců**: Seznam textových hodnot, kterých může nějaký řetězec nabývat. Seznam tohoto typu je reprezentovaný XML elementem `string-list`.
2. **Seznamy fuzzy hodnot**: Seznam hodnot které jsou přípustné pro konfigurační záznam typu *fuzzy*. Seznam tohoto typu je reprezentovaný XML elementem `fuzzy-list`.

Každý seznam má unikátní jméno uložené v atributu `name`. Na toto jméno se odkazuje šablonový soubor prostřednictvím elementu `data` u klíčových slov. Samotné hodnoty jsou ohraničeny elementy `value`.

Ukázka seznamu řetězců ze souboru `code-pages.xml`:

```
<freeconf-lists>
  <string-list name="windows-cp">
    <value>CP1250</value>
    <value>CP1251</value>
    <value>CP1255</value>
  </string-list>
</freeconf-lists>
```

1.4.8 Popisky seznamů hodnot

Také seznamy hodnot k sobě mohou mít definovanou nápovědu, která je uložena ve zvláštním souboru podle jazyka. Tento soubor k hodnotám v seznamu přiřazuje krátkou popisku, která se uživateli zobrazí na místo samotné hodnoty. Volitelně je také možné definovat detailnější popis každé hodnoty.

Soubory s nápovědou pro seznamy hodnot se hledají podle aktuálního jazyka v podadresáři `L10n` adresáře, ve kterém byl nalezen XML soubor seznamu. Název souboru s nápovědou je shodný s názvem souboru seznamu hodnot. Takže například anglické popisky k seznamu `/usr/share/freeconf/lists/code-pages.xml` se budou hledat v cestě `/usr/share/freeconf/lists/L10n/en/code-pages.xml`.

Ukázka nápovědy k seznamu hodnot `code-pages.xml`:

```
<freeconf-lists-help>
  <string-list name="windows-cp">
    <entry>
      <value>CP1250</value>
      <label>Windows-1250</label>
      <help>Kódová stránka Windows pro střední a východní Evropu.</help>
    </entry>
    <entry>
      <value>CP1251</value>
      <label>Windows-1251</label>
      <help>Kódová stránka Windows pro cyrilici.</help>
    </entry>
    <entry>
      <value>CP1255</value>
      <label>Windows-1255</label>
      <help>Kódová stránka Windows pro hebrejštinu.</help>
    </entry>
  </string-list>
</freeconf-lists-help>
```

1.4.9 Šablonový soubor pro GUI

Šablonový soubor GUI popisuje, jakým způsobem mají být zobrazeny záznamy z šablonového souboru v grafické konfigurační aplikaci. Lze zde definovat rozměry okna aplikace, rozdělení záznamů do záložek, grafické ikony a mnoho dalšího.

Podívejme se nyní na ukázku šablonového souboru pro GUI z konfiguračního balíku serveru Apache:

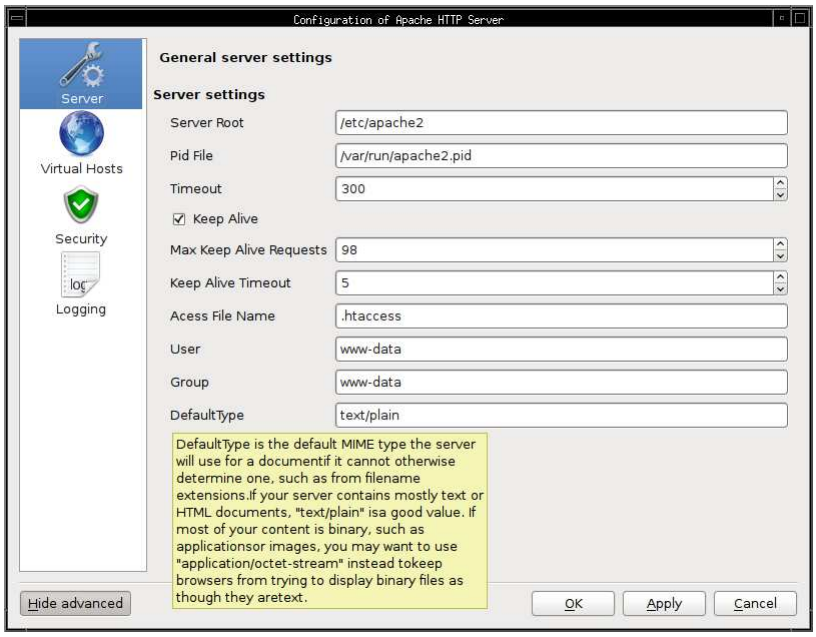
```
<freeconf-gui-template>
  <window>
    <!-- Vlastnosti okna konfigurační aplikace -->
```

```
    <min-width>640</min-width>
    <min-height>480</min-height>
    <max-width>800</max-width>
    <max-height>600</max-height>
  </window>

  <!-- Definice nové záložky -->
  <tab>
    <!-- Jméno záložky -->
    <tab-name>server-tab</tab-name>
    <!-- Ikona záložky -->
    <tab-icon>categories/preferences-other</tab-icon>
    <!-- Definice grafické sekce -->
    <tab-section>
      <!-- Jméno grafické sekce -->
      <tab-section-name>server-options</tab-section-name>
      <!-- Odkazy na záznamy z šablonového souboru -->
      <import-template-entry>
        /apache-config/ServerRoot
      </import-template-entry>
      <import-template-entry>
        /apache-config/LockFile
      </import-template-entry>

      <!-- ... -->
    </tab-section>
  </tab>
</freeconf-gui-template>
```

Nejprve jsme nastavili rozměry okna konfigurační aplikace. Dále jsme definovali novou záložku pro nastavení serveru a vytvořili v ní grafickou sekci. Toto je jiná sekce, než jaké známe ze šablonového souboru. Grafické sekce slouží k organizaci položek v GUI. Obvykle bývají nějak opticky odděleny, například rámečkem. Do nové grafické sekce následně vložíme klíčová slova, na která se zde odkazujeme pomocí cesty. Jak bude zobrazena záložka definovaná v ukázce vidíme na obrázku 1.2.

1.4.10 Popisky GUI

V tomto souboru jsou uvedeny popisky použité pro záznamy v šabloně GUI. Definuje se zde titulek okna aplikace, popisky jednotlivých záložek a grafických sekcí. Popisky mohou být dostupné ve více jazykových verzích.



Obrázek 1.2: Grafický klient

1.5 Dělení projektu

Projekt Freeconf dělíme na dvě vrstvy: výkonnou a aplikační. Výkonná vrstva je tvořena výkonnou knihovnou, která obsahuje veškerou logiku pro práci s konfiguračním balíkem. Aplikační vrstva je tvořena klientskou aplikací, která zajišťuje komunikaci s uživatelem. Dále do aplikační vrstvy řadíme aplikaci návrháře balíků, která má za cíl usnadnit tvorbu konfiguračních balíků.

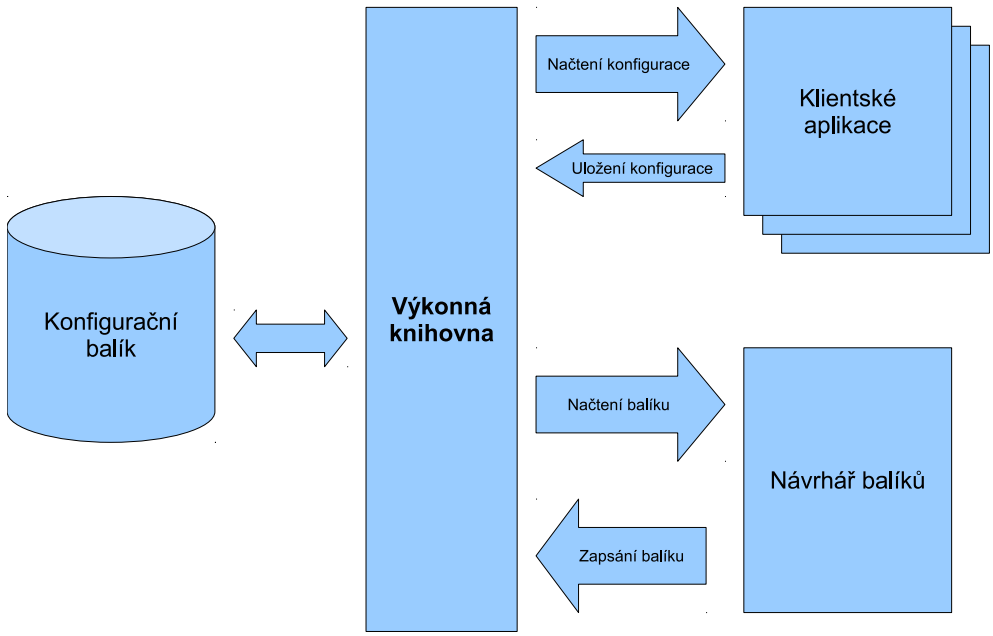
1.5.1 Výkonná knihovna

Výkonná knihovna poskytuje třídy a funkce, které obsahují veškerou logiku pro práci s konfiguračními balíky. V současné době je napsána v jazyce Python[6] a čítá něco přes 7000 řádek zdrojového kódu. Je zde kód pro načítání a ukládání konfiguračního balíku, vytváření nativních konfiguračních souborů, zpracování závislostí a mnoho dalšího. Knihovna také definuje rozhraní pro připojení klientské aplikace a návrháře balíků.

Rozhraní klientské aplikace poskytuje funkce, které jsou potřeba pro koncového uživatele. Umožňuje načtení balíku, změny konfiguračních hodnot a zápis výsledných konfiguračních souborů. Rozhraní pro návrháře navíc poskytuje funkce, jež mohou měnit vnitřní strukturu konfiguračního balíku. Přes toto rozhraní lze tedy například vytvořit nové klíčové slovo a nastavit jeho vlastnosti.

Role výkonné knihovny je znázorněna na obrázku 1.3. Jak vidíme, výkonná knihovna

[6]Python je dynamický objektově orientovaný programovací jazyk. Umožňuje rychlou tvorbu rozsáhlých počítačových programů. Více o tomto jazyce nalezneme v dokumentaci [11].



Obrázek 1.3: Role výkonné knihovny

zprostředkovává jakousi vrstvu, která odděluje klientské aplikace a návrháře balíků od samotné práce se soubory konfiguračního balíku.

1.5.2 Klientská aplikace

Klientská aplikace je cílena na koncového uživatele projektu Freeconf, kterému umožňuje provádět konfiguraci cílového programu. Po načtení konfiguračního balíku pro daný program se vykreslí GUI s ovládacími prvky, kterými je možné měnit nastavení. Po ukončení konfigurace a uložení nastavení dojde k vygenerování výsledného konfiguračního souboru pro cílový program.

V současné době existuje jen jedna referenční implementace klientské aplikace v jazyce Python využívající grafickou knihovnu Qt[7]. Jak tato aplikace vypadá vidíme na obrázku 1.2. V budoucnu může takových aplikací vzniknout více. Dalo by se uvažovat například o klientské aplikaci, která by byla přístupná jako HTML[8] stránka přes webový prohlížeč a umožnila tak i konfiguraci ze vzdáleného počítače. Užitečná by také mohla být klientská aplikace s rozhraním pro textový terminál pro případy, kdy grafické uživatelské prostředí není z nějakého důvodu dostupné.

[7] Qt multiplatformní knihovna pro vytváření programů s GUI. Více informací o této knihovně lze nalézt v dokumentaci [10].

[8] HTML(HyperText Markup Language) je značkovací jazyk pro hypertext. Je jedním z jazyků pro vytváření stránek v systému World Wide Web.

1.5.3 Návrhář balíků

Ruční editace XML souborů konfiguračního balíku je časově velmi náročná činnost. Ačkoli jsou XML soubory dobře čitelné, jejich zápis je zdlouhavý. Navíc jsou informace v konfiguračním balíku rozděleny do několika XML souborů, které jsou spolu vzájemě provázány.

A právě aplikace návrháře balíků si klade za cíl maximálně zjednodušit editaci souborů konfiguračního balíku. Je zde možné vytvořit nový balík i upravovat již existující konfigurační balíky. Do balíku lze vkládat nové klíče a sekce a měnit jejich vlastnosti, včetně nápovědy. Vše probíhá v přehledném GUI.

Dále se již budeme věnovat jen popisu výkonné knihovny. Zájemci o konfigurační aplikaci a návrháře balíků naleznou podrobnější informace v práci [7].

1.6 Základy implementace

1.6.1 Názvosloví

Třídy ve výkonné knihovně jsou pojmenovány podle jejich funkce a místa ve struktuře. Z názvu je pak možné ihned odvodit jejich funkci. Všechny třídy, se kterými se mohou uživatelé knihovny setkat, začínají předponou `Fc`, aby bylo jasné, že patří do knihovny projektu Freeconf. Vnitřní třídy a pomocné třídy knihovny tuto předponu mít nutně nemusí. Třídy ze základní struktury navíc používají tyto zkratky:

- T = Template – Třída je z šablonového stromu.
- C = Config – Třída je z konfiguračního stromu.
- K = Keyword – Třída reprezentuje klíčové slovo.
- S = Section – Třída je sekce, tj. může obsahovat odkazy na další klíčová slova a sekce.
- M = Multiple – Třída je vícenásobná sekce (viz kapitola 3.1).
- G = GUI – Třída je ze stromu pro GUI.

1.6.2 Hierarchie

Jádro výkonné knihovny tvoří tři stromy objektů, jež reprezentují prvky v XML souborech balíku. Jednou z nich je šablonový strom, který odpovídá struktuře klíčových slov, tak jak byla načtena ze šablonového souboru. Obsahuje názvy klíčových slov, jejich popisky a vlastnosti. Další důležitou strukturou je konfigurační strom, který odpovídá klíčovým slovům v konfiguračním souboru a v souboru s výchozími hodnotami. Obsahuje aktuální hodnoty klíčových slov, odkazy na příslušné objekty v šablonovém stromu a funkce pro aktualizaci závislostí. A nakonec grafický strom

objektů reprezentuje grafické prvky, které se načítají z šablony GUI. Obsahuje názvy prvků, jejich popisky, vlastnosti a odkazy na příslušné objekty konfiguračního stromu.

Podrobnější popis všech tříd a jejich vlastností nalezneme v práci [5].

Kapitola 2

Webový server Apache

2.1 Historie

2.1.1 CERN httpd

Počátky vývoje webových serverů sahají až do roku 1990, kdy byl ve švýcarském CERNu spuštěn první webový server na světě. Nesl název CERN httpd, kde *httpd* je zkratkou pro „HTTP daemon“. Jeho konfigurace sestávala z jednoho souboru s názvem `httpd.conf`. Měla jednoduchý formát ve tvaru:

```
# Komentář
KlíčovéSlovo  Hodnota

# Definice sekce
Sekce Parametr {
  # další klíčová slova
}
```

Některá klíčová slova v konfiguraci zůstala až do dnešní doby a nalezneme je i v serveru Apache. Jsou to například `ServerRoot`, `PidFile` nebo `ErrorLog`.

2.1.2 NCSA HTTPd

Vývoj webového serveru NCSA HTTPd začal v roce 1993 v NCSA (National Center for Supercomputing Applications) na Americké univerzitě ve státu Illinois. Původně mělo jít o malý a jednoduchý server, který by byl snazší na použití a konfiguraci než, v té době již poměrně komplexní, CERN httpd. Tyto vlastnosti vedly k jeho masovému rozšíření. Ve své době byl používán na více než 95% webových serverů v Internetu. Po verzi 1.5 byl pro nezájem vývojářů opuštěn a začal být postupně nahrazován modernějším serverem Apache.

Konfigurace NCSA HTTPd je již hodně blízká konfiguračním souborům serveru Apache. Formát zápisu klíčových slov je stejný jako u serveru CERN httpd. Jen

sekce jsou kvůli větší přehlednosti označeny počáteční a koncovou značkou ve tvaru `<Sekce` *`parametry`*`> ... </Sekce>` podobně jako v XML.

2.1.3 Apache

První veřejná verze serveru Apache s označením 0.6.2 byla vydána v dubnu roku 1995. Jako základ nového serveru posloužil kód stávajícího serveru NCSA HTTPd, který byl rozšířen o řadu uživatelských záplat („patchů“). Od toho je také odvozen jeho název, jako zkratka anglického slovního spojení „A patchy server“.

V prosinci roku 1995 vyšla verze 1.0 a v roce 2002 byla vydána verze 2.0, ve které byl všechen původní kód serveru NCSA HTTPd odstraněn. Nicméně pro zpětnou kompatibilitu zůstala konfigurace takřka nezměněna.

2.2 Proč právě Apache?

Zde je na místě otázka, proč jsme si vybrali právě webový server Apache? Jistě by se našlo nepřeberné množství dalších programů, jejichž konfiguraci bychom se mohli snažit popsat.

Jedním z důvodů bylo, že jde o poměrně rozsáhlý počítačový program s dlouhou historií a s velkým množstvím konfiguračních voleb. Pokud se nám podaří vyřešit všechny záludnosti nastavení takového složitého programu, pak popis konfigurace ostatních jednodušších programů již nebude představovat velký problém. Vývoj projektu Freeconf totiž probíhá iteračním způsobem s krátkými vývojovými cykly. V každém kroku vývoje se snažíme mít funkční systém schopný popsat co největší množinu programů. Nové funkce do návrhu přidáváme až tehdy, když nás k jejich použití donutí konfigurace některého existujícího programu. A právě nastavení serveru Apache obsahuje mnoho nových prvků, které jsme si chtěli vyzkoušet. Jde zejména o použití vícenásobných sekcí, rozdělení konfigurace do více než jednoho souboru a podporu zásuvných modulů.

Dalším důvodem bylo, že jde o velmi oblíbený a rozšířený webový server. V květnu roku 2011 na něm bylo provozováno odhadem 63% webových stránek v Internetu. Je dostupný zcela zdarma a běží na většině operačních systémů. Pokud by se nám podařilo popsat konfiguraci serveru Apache a přehledně ji graficky zobrazit, usnadnili bychom tím práci mnoha začínajícím administrátorům a vývojářům webových stránek.

V této práci jsme se pokusili vytvořit konfigurační balík, který by popsal základní konfiguraci serveru z výchozí instalace tak, aby byl schopen fungovat. Zaměřili jsme se také na vyřešení všech problémů, které by v průběhu konfiguračního procesu mohly nastat.

2.3 Konfigurace

Cílem této podkapitoly je popsat syntaxi konfiguračních souborů serveru Apache. Během výkladu se také zaměříme na zapsání struktury souborů do konfiguračního balíčku systému Freeconf.

2.3.1 Základní struktura

Nastavení webového serveru Apache je popsáno sadou několika konfiguračních souborů, které jsou na operačních systémech typu Unix obvykle uloženy v cestě `/etc/apache2`. Konfigurační soubory mají příponu `.conf` a obsahují nastavení jednotlivých klíčových slov. Každé klíčové slovo je zde na novém řádku, hodnota je od klíčového slova oddělena jednou nebo více mezerami. Řetězcové hodnoty jsou ohraničeny uvozovkami. Booleovskou hodnotu *pravda* zapisujeme jako `On`, což vyjadřuje, že daná konfigurační volba je zapnuta. Booleovská hodnota vyjadřující *nepravdu*, nebo vypnutí konfigurační volby, se zapisuje jako `Off`. Komentáře v souboru začínají znakem # a pokračují do konce řádky. Jednoduchou ukázku vyjmutou z hlavního konfiguračního souboru `apache2.conf` vidíme zde:

```
# ServerRoot: Určuje, kde jsou konfigurační soubory serveru.
ServerRoot "/etc/apache2"

# KeepAlive: Povolení nebo zákaz trvalých spojení (klient může
# v jednom spojení provést víc požadavků než jeden). Trvalá
# spojení lze zakázat nastavením volby na "Off".
KeepAlive On
```

Základní klíčová slova v takovémto formátu jsme schopni vygenerovat velice snadno. Využijeme k tomu předpřipravenou šablonu, jejíž autorem je pan Ing. Fabián, představenou v práci [6] na straně 21. Šablona je dobře parametrizována, a tak nastavením několika hodnot dostaneme XSL transformaci, která je schopna generovat kýžený formát:

```
<!-- Oddělovač názvu klíčového slova a jeho hodnoty je mezera -->
<xsl:variable
  name="key-value-separator" xml:space="preserve"> </xsl:variable>
<!-- Komentáře začínají znakem # -->
<xsl:variable name="comment-sequence">#</xsl:variable>
<!-- Vyjádření booleovské hodnoty pravda -->
<xsl:variable name="boolean-yes">On</xsl:variable>
<!-- Vyjádření booleovské hodnoty nepravda -->
<xsl:variable name="boolean-no">Off</xsl:variable>
```

Kompletní XSL transformaci balíku Apache naleznete v SVN repozitáři projektu (viz příloha A).

2.3.2 Sekce

V konfiguračních souborech serveru Apache se kromě jednoduchých záznamů vyskytují také záznamy sdružené do sekcí. Zde je ukázka použití sekce `Directory`, která umožňuje nastavit způsob zobrazení a přístupová práva k danému adresáři:

```
<Directory "/usr/share/doc/">
  Options Indexes FollowSymLinks
  Order deny,allow
  Deny from all
  Allow from 127.0.0.0/255.0.0.0 ::1/128
</Directory>
```

Tento kus konfiguračního souboru nám říká, že na adresáři `/usr/share/doc` je povoleno vypisování obsahu a budou se následovat symbolické odkazy. Přístup k adresáři budou mít povoleni pouze uživatelé, jejichž IP adresa je ve tvaru 127.x.x.x. Ostatní uživatelé budou odmítnuti.

Z kapitoly 2.3.1 již víme, že v konfiguračním balíku projektu Freeconf jsme schopni sdružovat záznamy do sekcí. Zdálo by se tedy, že nadefinovat sekci `Directory` a jí podřízené záznamy bude triviální operací. Jenže není tomu tak. Podívejme se ještě jednou a obecněji na formát zápisu sekce v konfiguračních souborech serveru Apache:

```
<Sekce [parametry]>
  ... nastavení klíčových slov, případně další sekce ...
</Sekce>
```

Zádrhel je právě v tom, že sekce zde může mít volitelné parametry. V projektu Freeconf jsme s něčím takovým nepočítali. Sekce z pohledu projektu Freeconf je jen jakýsi kontejner na další klíčová slova a sekce. Naštěstí máme k dispozici velice silný nástroj v podobě XSL transformací, do kterých je možné zapsat i poměrně komplikované podmínky. Můžeme se tedy k problému postavit tak, že parametry sekce jsou jen další klíčová slova. Jediný rozdíl mezi těmito parametry a ostatními klíčovými slovy je ve způsobu, jakým se zapisují do nativního konfiguračního souboru. Pro každou sekci s parametry bude tedy potřeba nadefinovat výjimku do transformačního XSL souboru. Hodnoty klíčových slov, které zastupují parametry se vypíší hned za název sekce. Ostatní klíčová slova se vypíší v souladu s transformací definovanou v 2.3.1.

Pro sekci `Directory` z předchozího příkladu tedy v šablonovém souboru nadefinujeme povinné klíčové slovo `Path` typu řetězec.

```
<section>
  <section-name>Directory</section-name>
  <string>
    <entry-name>Path</entry-name>
    <mandatory>yes</mandatory>
```

```
    </string>

    ...
  </section>
```

A do souboru s XSL transformací přidáme výjimku pro sekci Directory následujícím způsobem:

```
<!-- Zpracování sekcí -->
<xsl:template match="section">
  <xsl:choose>
    <!-- Výjimka pro sekci Directory -->
    <xsl:when test="@name = 'Directory'">
      <!-- Začátek sekce -->
      <xsl:text>&lt;</xsl:text>
      <!-- Název sekce -->
      <xsl:value-of select="@name" />
      <!-- Cesta k adresáři, bude obalena uvozovkami -->
      <xsl:text> "</xsl:text>
      <xsl:value-of select="entry[@name = 'Path']/value" />
      <xsl:text>"&gt;</xsl:text>
      <!-- Odřádkování -->
      <xsl:call-template name="new-lines"/>

      <!-- Zde zavoláme šablony pro zpracování všech klíčových slov
           s výjimkou klíčového slova Path -->
      <xsl:apply-templates select="*[@name != 'Path']"/>

      <!-- Konec sekce -->
      <xsl:text>&lt;/</xsl:text>
      <xsl:value-of select="@name" />
      <xsl:text>&gt;</xsl:text>
      <xsl:call-template name="new-lines"/>
    </xsl:when>

    <!-- ... Zde bude zpracování ostatních sekcí ... -->
  </xsl:choose>
</xsl:template>
```

Nevýhodou tohoto přístupu je, že pro každou další sekci, která se může vyskytnout v konfiguračních souborech jsme nuceni přidat podobnou výjimku do XSL transformace. Dalším možným řešením toho problému by bylo přidat podporu sekcí s parametry přímo do výkonné knihovny projektu Freeconf. Nicméně tento způsob zápisu sekcí je specifický pouze pro server Apache. Nikde jinde jsme se s ním zatím nesetkali. Navíc takových sekcí je v konfiguraci jen omezené množství. Celkem jsme jich v konfiguračních souborech napočítali 9. Pokud tedy problém můžeme vyřešit

přímo v XSL transformaci, pak by rozšiřování výkonné knihovny kvůli několika málo záznamům znamenalo jen zbytečnou komplikaci celého návrhu.

2.3.3 Složené záznamy

Další anomálií na kterou jsme v konfiguraci serveru Apache narazili, jsou záznamy, jejichž hodnota nese více než jednu informaci. Pro jednoduchost budeme takové záznamy označovat pojmem složené záznamy. Vše bude jasnější, když se podíváme na ukázku z konfigurace. Vezměme si například klíčové slovo `Options`, které jsme již dříve viděli použít v sekci `Directory`:

```
# Directory options
Options Indexes FollowSymLinks
```

Na první pohled vypadá záznam celkem obyčejně. Definice začíná názvem klíčového slova, následuje hodnota `Indexes FollowSymLinks`. Když se ale podíváme do dokumentace serveru Apache, zjistíme, že hodnota klíčového slova `Options` je dále strukturována. V podstatě se jedná o seznam předem definovaných přepínačů. Přepínače, které je zde povoleno použít jsou:

- `Indexes` - Vypíše obsah adresáře v případě, že není nalezen soubor `index.html`.
- `ExecCGI` - Povolí spouštění CGI skriptů, které dynamicky generují obsah webových stránek.
- `FolowSymLinks` - Umožní následovat symbolické odkazy.
- `SymlinksIfOwnerMatch` - Umožní následovat symbolické odkazy pouze pokud se shoduje vlastník odkazu a cílového souboru.
- `Includes` - Povolí předzpracování webových stránek na straně serveru pomocí maker SGML.
- `IncludesNOEXEC` - Stejné jako předchozí volba, až na to, že spouštění externích programů není povoleno.
- `MultiViews` - Povolí vyhledávání souborů podle masky, pokud dotaz na stránku vede na neexistující soubor.
- `All` - Zapne všechny volby, kromě `MultiViews`. Toto je výchozí chování.

Přitom každý přepínač může být použit pouze jednou.

Nejjednodušším řešením by bylo nadefinovat klíčové slovo jako řetězec a formátování jeho hodnoty popsat do nápovědy. Bude pak na uživateli, aby správně zadal názvy jednotlivých přepínačů. Pokud uživatel zadá název nějakého přepínače chybně, nebo zadá přepínače ve špatném formátu, způsobí to vytvoření neplatné konfigurace a server Apache nebude fungovat správně nebo se vůbec nespustí. Navíc je takový

postup v rozporu s naší snahou o maximální zjednodušení konfigurace pro uživatele. To je jistě nežádoucí stav, a tak bude třeba zvolit jiný přístup.

Uvědomili jsme si, že klíčové slovo `Options` jen sadou několika přepínačů, z nichž každý je buď zapnutý, nebo vypnutý. Jde tedy vlastně o sadu booleovských hodnot, které jsou sdružené do jednoho logického celku. V konfiguračním balíku budeme tedy chápat direktivu `Options` jako sekci a jednotlivé přepínače jako klíčová slova typu `bool`. Příslušný úsek konfiguračního souboru vidíme na následující ukázce:

```
<section>
  <section-name>Options</section-name>
  <bool><entry-name>FollowSymlinks</entry-name></bool>
  <bool><entry-name>ExecCGI</entry-name></bool>
  <bool><entry-name>Includes</entry-name></bool>
  <bool><entry-name>IncludesNOEXEC</entry-name></bool>
  <bool><entry-name>Indexes</entry-name></bool>
  <bool><entry-name>MultiViews</entry-name></bool>
  <bool><entry-name>SymLinksIfOwnerMatch</entry-name></bool>
</section>
```

Pozorný čtenář si už možná všiml, že jsme zde vynechali volbu `All`. Je to proto, že podle dokumentace dosáhneme stejného efektu zapnutím všech voleb s výjimkou volby `MultiViews`. Výhodu tohoto řešení spatřujeme především v tom, že uživatel přímo uvidí, které volby jsou aktivovány.

Teď už nám zbývá jen přidat výjimku do transformačního XSL souboru tak, aby se záznam `Options` správně zapsal do nativního konfiguračního souboru. Na rozdíl od výjimky pro sekci `Directory` zde již ale nebudeme volat šablony na zbývající klíčová slova. Pro každé klíčové slovo pouze otestujeme, zda je zapnuté a pokud ano, vypíšeme jeho název.

```
<xsl:when test="@name = 'Options'">
  <!-- Výjimka pro sekci Options -->
  <xsl:value-of select="@name" />
  <!-- Zpracování seznamu voleb -->
  <xsl:for-each select="entry">
    <xsl:value-of select="$key-value-separator" />
    <xsl:if test="value = 'yes'">
      <!-- Vypíšeme název zapnuté volby -->
      <xsl:value-of select="@name" />
    </xsl:if>
  </xsl:for-each>
  <xsl:call-template name="new-lines"/>
</xsl:when>
```

Zpracování výpisu složeného záznamu se nám podařilo zapsat na několik málo řádek transformačního souboru. Podobných záznamů, které lze v projektu Freeconf popsat

pomocí sekcí, jsme v konfiguraci serveru Apache nalezli více. Pro každý takový záznam bylo třeba přidat výjimku do XSL transformace, protože zápis jejich hodnot není obecně stejný.

Až dosud jsme si při konfiguraci serveru Apache vystačili se stávajícími vlastnostmi konfiguračních balíků, výrazně nám pomohla možnost definovat vlastní XSL transformaci. Nicméně další problémy, na které jsme v konfiguraci serveru narazili a které popíšeme v následujících podkapitolách, si již vyžádaly zásahy do návrhu projektu Freeconf. Zde uvedeme pouze problémy samotné. Nutné úpravy projektu naleznete v samostatné kapitole 3 na straně 40.

2.3.4 Vícenásobné sekce

V kapitole 2.3.2 jsme rozebírali zápis sekce `Directory`, která má jeden klíč jménem `Path` použitý jako parametr. Pokud bychom sekci nechali definovanou tak, jak jsme si ukázali v úryvku šablonového souboru, znamenalo by to, že uživatel by ji mohl zadat pouze jednou. Jenže konfigurace Apache umožňuje zadat omezení na více než jeden adresář, tedy sekce `Directory` se v konfiguračních souborech může vyskytovat mnohokrát, pokaždé s jinými parametry.

V terminologii projektu Freeconf bychom řekli, že sekci v šablonovém stromu může odpovídat více instancí sekce ve stromu konfiguračním. Takové sekce nazýváme vícenásobné. Podpora vícenásobných sekcí již v projektu Freeconf kdysi byla, nicméně při přepisu knihovny do programovacího jazyka Python byla kvůli zjednodušení z návrhu vypuštěna. Novému návrhu vícenásobných sekcí je věnována kapitola 3.1.

Dalším příkladem vícenásobné sekce v konfiguraci serveru Apache by mohl být složený záznam `Listen`, který definuje na jaké IP adrese a na jakém portu má webový server očekávat požadavky. Těchto dvojic (adresa a port) je možné definovat více, přičemž vždy by se v konfiguraci měla vyskytovat alespoň jedna. To je celkem rozumný požadavek, vždyť k čemu by nám byl webový server, když bychom se k němu nemohli odnikud připojit? Nový návrh vícenásobných sekcí by tedy měl pokrýt i tato omezení.

2.3.5 Vnořené záznamy a sekce

V průběhu pročítání dokumentace serveru Apache jsme několikrát narazili na záznamy a sekce, které se mohou vyskytovat jak na nejvyšší úrovni, tak i vnořené do některé ze sekcí. Záznam má v každém místě jiný význam, který je závislý na kontextu použití, nicméně syntaxe jeho zápisu je stále stejná. Vše bude jistě jasnější na ukázce z reálné konfigurace.

Vezměme například vícenásobnou sekci `Files`. Tato sekce umožňuje nastavit vlastnosti a omezení na množinu souborů, která je definována regulárním výrazem. Pokud do konfigurace umístíme následující text, server nepovolí přístup k souborům, které se jmenují `private.html`, ať už jsou v jakémkoli adresáři.

```
<Files private.html>
  Order allow,deny
  Deny from all
</Files>
```

Když ale stejný kód umístíme do sekce `Directory`, bude omezení přístupu k souborům platit jen v rámci daného adresáře a všech jeho podadresářů:

```
<Directory "/www/web">
<Files private.html>
  Order allow,deny
  Deny from all
</Files>
</Directory>
```

Podobné chování dostaneme, i když sekci `Files` umístíme do sekce `VirtualHost`. V takovém případě bude omezen přístup k souborům `private.html` jen v rámci daného virtuálního hostitele.

V konfiguračním balíku projektu Freeconf má ale každý záznam a sekce své pevné umístění v konfiguračním stromě. Znamenalo by to tedy zapsat definici sekce `Files` celkem třikrát: na nejvyšší úroveň, do sekce `Directory` a do sekce `VirtualHost`. A vše se ještě zkomplikuje, povíváme-li se na dokumentaci sekce `Directory`. Tu lze totiž také zapsat jak do nejvyšší úrovně konfigurace, tak i do sekce `VirtualHost`. Zopakovat by se musela nejen definice v šablonovém souboru, definice výchozích hodnot, ale také text nápovědy. Takové řešení je sice pro uživatele konfiguračního balíku přijatelné, nicméně by znamenalo značnou komplikaci pro vývojáře balíku. Malá změna v definici sekce by si vyžádala úpravy na mnoha místech v konfiguračních XML souborech.

V této práci jsme se pokusili implementovat koncept odkazů v šablonovém stromu, který tyto problémy řeší. Na jeho návrh a implementaci se podíváme v kapitole 3.2.

2.3.6 Více konfiguračních souborů

Další vlastnost konfigurace serveru Apache, kterou projekt Freeconf zatím nedokázal popsat, je rozdělení záznamů do více nativních konfiguračních souborů. Běžná instalace serveru obsahuje výchozí soubor s obecnou konfigurací `apache2.conf`, soubor se seznamem portů `ports.conf`, soubory s nastavením virtuálních hostitelů a soubory s nastavením aktivních zásuvných modulů.

Jako první se načítá soubor `apache2.conf`. Další soubory se pak načítají pomocí direktivy `Include`. Tedy například na soubor `ports.conf` vede odkaz:

```
Include ports.conf
```

Uživatel tak může konfiguraci serveru rozdělit do libovolného množství konfiguračních souborů a je jen na něm, jak je pojmenuje. Nicméně takové rozšíření by znamenalo poměrně drastický zásah do projektu Freeconf. Proto jsme se rozhodli o podporu trochu skromnějšího scénáře, kdy výstupní soubory jsou pevně definované v konfiguračním balíku. Zachováme tím přehlednost konfigurace. Například oddělení konfigurace portů, na které jsou administrátoři serverů zvyklí, zůstane. Zároveň ale nebudeme zatěžovat uživatele definicí vlastních konfiguračních souborů, které by konfiguraci jen znepřehlednily. Navíc tato zjednodušená úprava bude znamenat jen malý zásah do návrhu projektu Freeconf, jak uvidíme v kapitole 3.3.

2.3.7 Zásuvné moduly

Poslední novinkou, která se vyskytuje v konfiguraci serveru Apache jsou takzvané zásuvné moduly tvořené binárními knihovnami. Ty se načítají při spouštění serveru a mohou jej doplňovat o nové funkce. Lze tak přidat například podporu pro šifrování přenosů dat pomocí SSL, a nebo podporu pro dynamicky generované webové stránky psané v programovacím jazyce PHP.

Každý takový zásuvný modul může přidávat i podporu pro nové konfigurační klíče, které umožňují jeho nastavení. Nové klíče se obvykle zapisují do odděleného konfiguračního souboru, přičemž tyto soubory se pak načítají z `apache2.conf` direktivou `Include`.

I zde bude třeba rozšířit návrh projektu Freeconf a umožnit uživateli vkládat do konfigurace rozšiřující moduly. Každý takový modul bude moci přidat nové konfigurační klíče a sekce včetně výchozích hodnot, nápovědy, XSL transformace a definice grafického rozhraní. Více o tomto rozšíření se dozvíme v kapitole 3.4.

Kapitola 3

Změny v projektu pro webový server Apache

V této části se budeme věnovat změnám v návrhu projektu Freeconf, které jsou nutné pro konfiguraci webového serveru Apache.

3.1 Vícenásobné sekce

V kapitole 2.3.4 jsme narazili na problém výskytu vícenásobných sekcí v konfiguraci. Pro klasické sekce platí, že každé sekci v šablonovém stromě odpovídá jedna sekce ve stromě konfiguračním. U vícenásobných sekcí může jedné sekci v šablonovém stromě odpovídat $0...N$ sekcí ve stromě konfiguračním. Sekci v šablonovém stromě budeme nazývat definice sekce. Její obraz v konfiguračním stromě pak nazveme instancí sekce. Situace je znázorněna na obrázku 3.1.

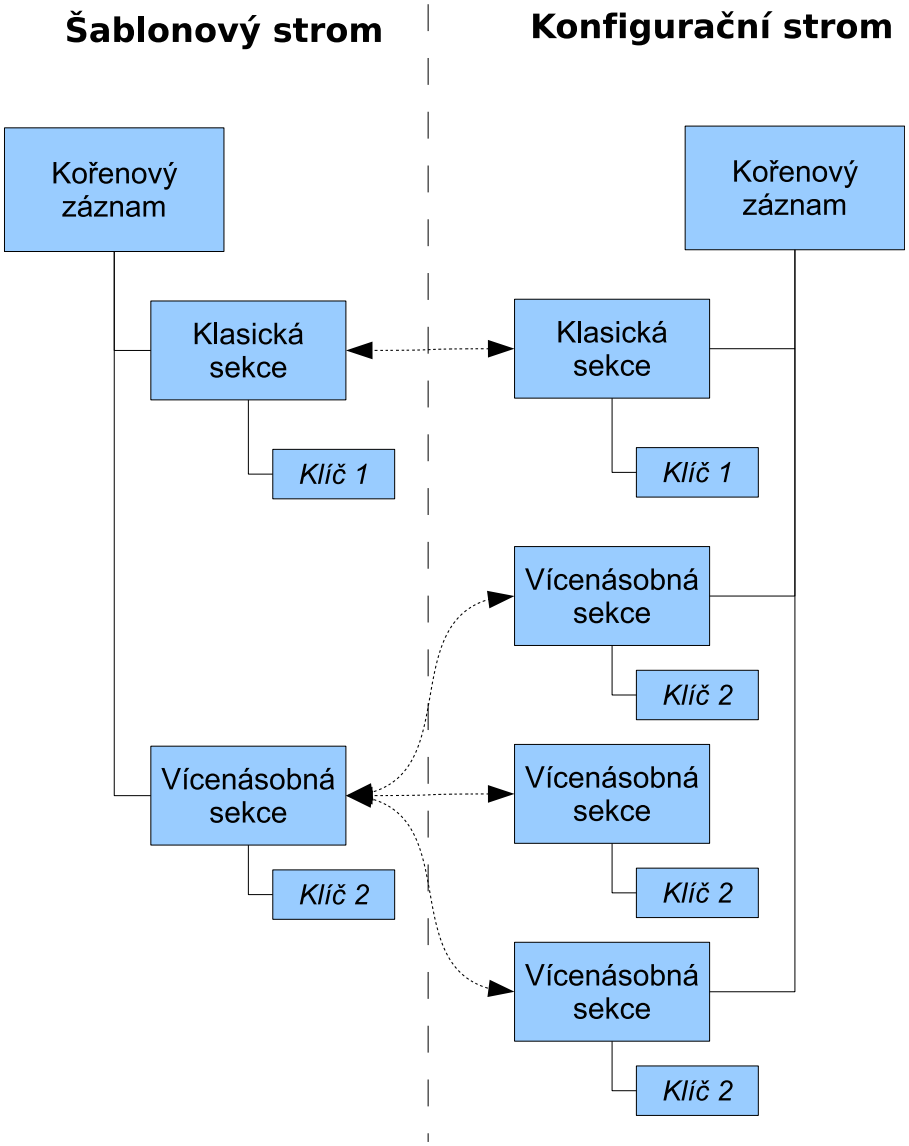
Cílem je, aby uživatel mohl, například po stisknutí tlačítka, vložit další instanci vícenásobné sekce do konfigurace a mohl upravovat hodnoty jejích klíčových slov. Samozřejmě by také měl mít možnost vícenásobnou sekci smazat.

Z kapitoly 2.3.4 také vyplývá, že počet instancí vícenásobné sekce může být zdola omezen například jedničkou. Obecně budeme předpokládat, že počet instancí sekce může být omezen jak zdola, tak i shora. Pokud se uživatel pokusí překročit zadané limity, akce se neprovede a uživateli se zobrazí varování.

3.1.1 Definice

Nejvýznamnější změnou v konfiguračním balíku bude zápis sekce v šablonovém souboru. Zde musíme nějakým způsobem označit zda daná sekce je vícenásobná či nikoliv. U vícenásobných sekcí bychom také měli mít možnost zadat horní a dolní limit na počet jejich instancí.

Nakonec jsme došli k následujícímu zápisu: Zápis klasických sekcí ponecháme beze změny a vícenásobné sekce označíme XML značkou `<multiple/>` (což je v jazyce



Obrázek 3.1: Rozdíl vícenásobné a klasické sekce

Obrázek 3.2: Zobrazení vícenásobné sekce

XML zkratka pro výraz `<multiple></multiple>`). Pokud bude potřeba zadat nějaké vlastnosti specifické pro vícenásobnou sekci, například limity na počet instancí, zapíšeme tyto vlastnosti mezi značky `multiple`. Vše si ukážeme na příkladu vícenásobné sekce `Listen` z konfigurace serveru Apache:

```
<section>
  <section-name>Listen</section-name>
  <multiple>
    <!-- Minimální počet instancí -->
    <min>1</min>
    <!-- Maximální počet instancí, smyšlená konstanta je zde pouze
         pro ilustraci -->
    <max>256</max>
  </multiple>
  <!-- Definice klíčových slov -->
  <!-- ... -->
</section>
```

3.1.2 Zobrazení v GUI

V předchozích odstavcích jsme si již řekli, že bychom chtěli, aby byl uživatel schopen upravovat seznam instancí vícenásobných sekcí. Naše konfigurační aplikace by měla umožňovat přidávat a odebírat sekce, měnit pořadí sekcí a upravovat jejich obsah.

Na obrázku 3.2 vidíme cílový stav zobrazení vícenásobné sekce. Uprostřed je patrný seznam instancí vícenásobné sekce. Po pravé straně pak nalezneme tlačítka na přidání a odebrání sekce a na změnu pořadí prvků v seznamu. Pokud vybereme prvek ze seznamu a dvakrát na něj poklikneme, nebo stiskneme tlačítko Enter, zobrazí se editační okno sekce, kde můžeme upravovat hodnoty jednotlivých klíčů. Povšimněme si červeně podbarveného záznamu. Tímto způsobem jsou zvýrazněny instance sekce, které jsou v nekonzistentním stavu (viz kapitola 1.3.2).

Šablonový soubor GUI

V původním stavu šablonový soubor umožňoval odkazovat se pouze na klíče z konfiguračního stromu. Odkázat se na celou sekci nebylo možné. Pokud tedy chtěl vývojář balíku zobrazit celou jednu sekci v GUI, musel do šablonového souboru pro GUI vypsat zvlášť každý její klíč.

Zde ovšem narazíme na problém, jakým způsobem se odkázat na vícenásobnou sekci? Odkazovat se přímo na klíče uvnitř vícenásobné sekce zjevně nemůžeme, neboť klíče uvnitř vícenásobné sekce v konfiguračním stromě vůbec nemusí existovat, nebo tam jsou, ale víckrát než jednou. Proto do šablonového souboru pro GUI zavádíme nový element `<import-template-section>`, který do GUI vloží sekci a všechny její podřízené prvky, jako jsou klíče a další sekce. Podívejme se na příklad použití:

```
<import-template-section>
  /apache-config/Directory
</import-template-section>
```

Tímto krokem zjednodušíme práci vývojářům konfiguračních balíků, protože už v definici GUI nemusí vypisovat každý klíč sekce zvlášť. Také si tím připravíme půdu pro zobrazování vícenásobných sekcí. Pokud totiž konfigurační aplikace zjistí, že vkládaná sekce je vícenásobná, zobrazí seznam s tlačítky, který jsme viděli na obrázku 3.2.

Při návrhu zobrazení vícenásobné sekce jsme narazili na problém, jakým způsobem od sebe odlišit jednotlivé instance vícenásobné sekce. Ideální by bylo u každé instance v seznamu vypsat nějaký její jednoznačný identifikátor. Zde se inspirujeme u vícenásobných sekcí v konfiguraci webového serveru Apache, kde každá sekce má svůj parametr, který ji i jednoznačně identifikuje. Například pro adresář (sekce `Directory`) je to cesta v souborovém systému.

U vícenásobných sekcí tedy zavedeme nový pojem: primární klíč. Jde o klíč, jehož hodnota pokud možno jednoznačně identifikuje instanci vícenásobné sekce. Tato hodnota se pak také zobrazí v GUI v seznamu instancí vícenásobné sekce. Primární klíč budeme definovat ve vlastnostech elementu `import-template-section` jako jméno klíče v sekci následujícím způsobem:

```
<import-template-section primary="Path">
  /apache-config/Directory
</import-template-section>
```

Pokud pro sekci z nějakého důvodu není primární klíč definovaný, použije se první klíč sekce.

Do budoucna by bylo možné uvažovat i o podpoře více primárních klíčů. Mohly by totiž nastat situace, kdy jeden klíč k identifikaci sekce nestačí a je třeba použít kombinace více klíčů.

3.1.3 Změny ve výkonné knihovně

V této kapitole se podíváme na nejvýznamnější změny ve výkonné knihovně, které si vynutila implementace vícenásobných sekcí.

Nové vlastnosti

Abychom mohli úspěšně načíst nové vlastnosti popisující vícenásobné sekce ze šablonového souboru, musíme nejprve rozšířit třídu `FcTSEntry`, která reprezentuje sekci v šablonovém stromu. Nové vlastnosti třídy jsou:

- **multiple**: Booleovská hodnota. Říká nám, zda sekce je vícenásobná či nikoliv. Výchozí hodnotou je nepravda.
- **multipleMin**: Celé číslo reprezentující dolní hranici počtu instancí sekce. Lze ji nastavit pouze pokud vlastnost `multiple` je pravdivá. Pokud je nastavena na 0, pak počet instancí není omezen.
- **multipleMax**: Celé číslo reprezentující horní hranici počtu instancí sekce. Chová se stejně jako vlastnost `multipleMin`.

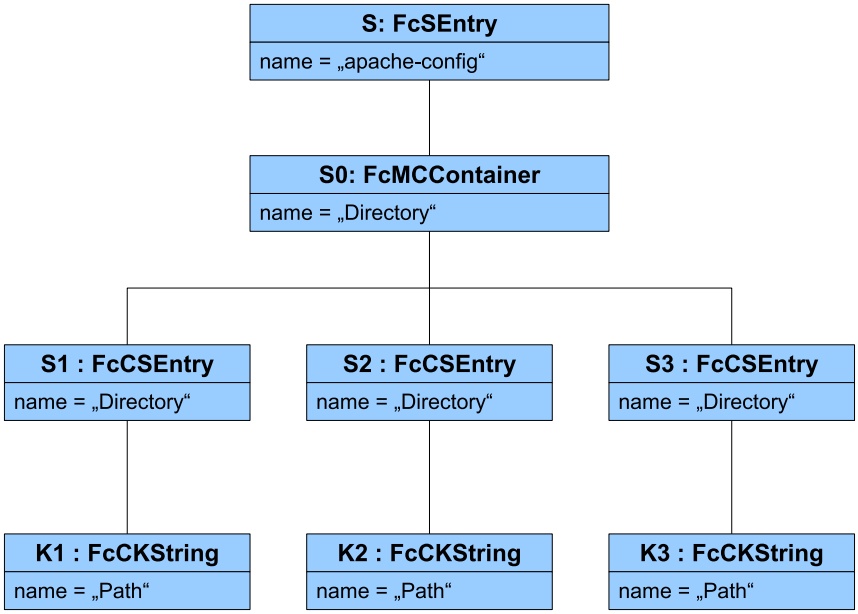
Kontejner vícenásobných sekcí

Nyní je potřeba vyřešit, jakým způsobem budeme vícenásobné sekce ukládat do konfiguračního stromu. Po zvážení několika možností jsme nakonec dospěli k obdobnému řešení, jaké bylo použito už v původní výkonné knihovně `libfreeconf`.

Základem tohoto řešení je nová třída `FcMCContainer` odvozená od třídy `FcCSEntry` (ta reprezentuje sekce v konfiguračním stromě). Objekt třídy `FcMCContainer` budeme do konfiguračního stromu vkládat namísto objektu klasické sekce. Na diagramu 3.3 je znázorněno, jak to bude vypadat v případě, že do konfiguračního stromu vložíme 3 instance vícenásobné sekce `Directory` z konfigurace serveru Apache. Instance sekce `Directory` jsou označeny symboly `S1`, `S2` a `S3`. Za pozornost stojí, že kontejnerový objekt má stejné jméno jako vícenásobná sekce.

Objekty typu `FcMCContainer` jsou tedy jakési virtuální sekce nebo kontejnery, jejichž obsahem jsou instance vícenásobné sekce. Tímto elegantním řešením zajistíme, že i když v konfiguraci nebude uložena ani jedna instance vícenásobné sekce, stále budeme mít v konfiguračním stromě zástupný objekt typu `FcMCContainer`, i když samozřejmě prázdný. Ten nám říká, že zde je vícenásobná sekce. Díky tomu v klientské aplikaci snadno zobrazíme seznam instancí sekce s ovládacími tlačítky, jak bylo vidět na obrázku 3.2. Přidání instance vícenásobné sekce se pak rovná vložení sekce do našeho kontejnerového objektu.

Drobnou nevýhodou řešení s pomocí kontejneru je, že musíme ošetřit všechna místa v kódu, kde se prochází konfigurační strom. Tedy například při načítání vícenásobných sekcí z konfiguračního souboru musíme nejprve zjistit, zda v konfiguračním



Obrázek 3.3: Použití objektu typu `FcMCContainer`

stromě existuje příslušný kontejner. Pokud ne, vložíme do konfiguračního stromu objekt typu `FcMCContainer` a až pak můžeme vkládat jednotlivé instance vícenásobné sekce.

Abychom si usnadnili časté testování typu sekce v konfiguračním stromu, přidáme do třídy `FcCSEntry` novou vlastnost `isMultipleContainer`, která nám řekne, zda daný objekt je klasická sekce nebo kontejner vícenásobných sekcí.

Vložení instance

Vkládání nových instancí vícenásobné sekce ošetříme ve funkci `createCEntry` třídy `FcTEntry`, která má na starost vytváření objektů v konfiguračním stromě z jejich vzoru v šablonovém stromě. Musíme zde ošetřit zejména situaci zmíněnou v předchozí kapitole, kdy pro vícenásobné sekce musíme vytvořit objekt kontejneru. Měli bychom zde také otestovat, zda počet instancí vícenásobné sekce nepřekročí zadané limity. Podívejme se tedy, jak toho ve funkci `createCEntry` dosáhneme:

```
def createCEntry (self, centry_parent):
  centry = None
  if centry_parent != None and self.multiple == False:
    # Existuje už záznam ?
    centry = centry_parent.findCEntry(self.name)

  if centry == None:
```

```
    if self.type == FcTypes.SECTION and self.multiple == True:
      if centry_parent != None and
         centry_parent.isMultipleEntryContainer():
        # Kontrola počtu instancí
        n = len(centry_parent.entries)
        if self.multipleMax and n + 1 > self.multipleMax:
          raise FcMultipleError()
        # Vytvoření záznamu
        centry = self._makeCEntry()
      else:
        # Vytvoření kontejneru
        centry = FcMCContainer(self)
    else:
      centry = self._makeCEntry()

    if centry_parent != None:
      centry_parent.append(centry)
  return centry
```

Jak vidíme, funkce má jeden parametr `centry_parent`. Ten určuje, do které konfigurační sekce se má nový objekt vložit. Pokud je tento parametr nastavený na *None*(prázdná hodnota), pak nový objekt do žádné sekce vložen nebude. To se hodí zejména při vytváření kořenového prvku konfiguračního stromu.

Funkce testuje, zda konfigurační objekt ve stromu existuje. Pokud ano, vrátí se nalezený objekt a funkce skončí. Tuto kontrolu je ale pro vícenásobné sekce potřeba vypnout, neboť můžeme mít více instancí jedné sekce.

Následuje kód na vytvoření nového konfiguračního záznamu. Když vytváříme vícenásobnou sekci a vkládáme ji do kontejneru, provede se kontrola na počet instancí sekce a pokud je vše v pořádku, vytvoříme nový záznam přetíženou metodou `_makeCEntry`. Jestliže kontejner neexistuje, vytvoříme ho.

Výsledkem funkce je nový objekt v konfiguračním stromě.

Při vytváření instance vícenásobné sekce uživatelem z GUI musíme ještě na čerstvý objekt zavolat novou metodu `fill`. Ta rekurzivně projde podřízené záznamy sekce v šablonovém stromě a vytvoří jejich obraz ve vícenásobné sekci. Toto je ošetřeno ve třídě `FcGUICommunicator`, která je zodpovědná za komunikaci klientské aplikace a výkonné knihovny.

Odebrání instance

Odebrání instance vícenásobné sekce ošetříme ve třídě `FcMCContainer`. Zde přetížíme funkci `disconnect`, která se volá, pokud chceme odstranit objekt z konfiguračního stromu. Ve funkci nejprve zkontrolujeme, zda odebráním záznamu neporušíme limit na minimální počet instancí vícenásobné sekce. Pokud ano, vyvoláme výjimku. Pokud je vše v pořádku, zavoláme původní metodu `disconnect`:

```
def disconnect(self, entry):
  # Kontrola počtu instancí
  n = len(self.entries)
  if self.templateBuddy.multipleMin and
     n - 1 < self.templateBuddy.multipleMin:
    raise FcMultipleError()
  # Odebereme záznam
  super(FcMCContainer, self).disconnect(entry)
```

Výchozí hodnoty

Zavedení vícenásobných sekcí si vyžádalo změnu vztahu šablonového a konfiguračního stromu. Dříve totiž platilo, že každému objektu v šablonovém stromě odpovídá jeden objekt v konfiguračním stromě, přičemž stromy byly vzájemně provázané. Z objektu v konfiguračním stromě jsme se mohli dostat pomocí vlastnosti `templateBuddy` na příslušný objekt v šablonovém stromě. A naopak z objektu v šablonovém stromě vedla vazba přes vlastnost `configBuddy` na konfigurační objekt. Šlo tedy o oboustrannou vazbu typu 1:1 a na mnoha místech kódu výkonné knihovny se s tím počítalo.

Ovšem vícenásobné sekce toto pravidlo porušily. Nyní je vztah šablonového a konfiguračního stromu typu 1:N. Musíme tedy přepracovat vazební vlastnosti a vše co s tím souvisí. Je zřejmé, že vlastnost `templateBuddy`, která vede z objektu v konfiguračním stromě na jeho vzor v šablonovém stromě, můžeme ponechat. Ale vlastnost `configBuddy` vedoucí druhým směrem je třeba zrušit. Z šablonového stromu se již nelze jednoznačně odkázat na příslušný konfigurační objekt.

Jednou z věcí, kterou je třeba kvůli této změně přepracovat, je umístění výchozích hodnot. Ty se totiž ukládaly přímo do klíčů v konfiguračním stromě. To ale nyní naráží na dva problémy:

1. Při nastavování výchozích hodnot z grafického návrháře se používala právě vazba `configBuddy` šablonového objektu, přes níž se přistupovalo k výchozí hodnotě klíče.

2. Klíč, který je ve vícenásobné sekci, v konfiguračním stromě neexistuje, dokud není vložena alespoň jedna instance sekce. Při načítání souboru s výchozími hodnotami by tedy mohla nastat situace, kdy bychom neměli výchozí hodnotu kam uložit.

Rozhodli jsme se tedy, že výchozí hodnoty přesuneme do šablonového stromu do třídy `FcTKEntry`, kam nyní logicky lépe zapadají. Z objektů v konfiguračním stromě jsou výchozí hodnoty nadále dostupné přes vazební vlastnost `templateBuddy` jako: `templateBuddy.defaultValue`. Spolu s tím bylo potřeba přesunout i funkce `convertValue` a `getValueRepr`, které se používají při nastavování a vypisování výchozích hodnot.

Závislosti

Dalším problémem, který bude potřeba vyřešit ve vztahu k vícenásobným sekcím jsou závislosti mezi jejich klíčovými slovy. Vezměme například hypotetickou závislost v jejíž podmínce je zadán výraz `/apache-config/Directory/Path = "/tmp"`. Taková závislost zjevně nemá smysl, pokud ve vícenásobné sekci `Directory` není ani jedna instance. Naopak, pokud by v sekci bylo instancí více, tak bez nějaké dodatečné informace nevíme vzhledem ke které instanci se má podmínka vyhodnotit.

Právě kvůli těmto logickým problémům jsme zatím závislosti vícenásobných sekcí do výkonné knihovny neimplementovali. Navíc v konfiguraci serveru Apache zatím nebyly potřeba. Jestliže se při použití současné verze výkonné knihovny pokusíme zapsat závislost na nějakém klíčovém slově uvnitř vícenásobné sekce, dostaneme chybu podobnou této:

```
Multiple entry Directory inside path /apache-config/Directory/Path!
```

Pokud bychom v budoucnu přeci jen potřebovali doplnit podporu závislostí vícenásobných sekcí, dalo by se uvažovat o dvou pravidlech, které by nejednoznačnosti řešily:

1. Klíčová slova uvnitř vícenásobné sekce mohou záviset na hodnotách klíčových slov v aktuální instanci sekce a na hodnotách klíčových slov v nadřazených sekcích nebo v sekcích, které nejsou vícenásobné.
2. Klíčová slova mimo vícenásobnou sekci nemohou záviset na hodnotách slov uvnitř vícenásobné sekce, neboť taková podmínka nemusí dávat smysl.

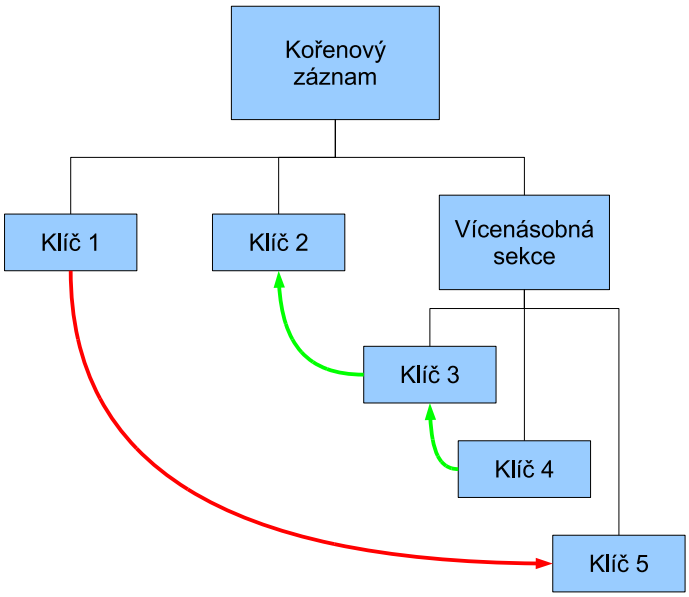
Aplikace pravidel je znázorněna na obrázku 3.4. Povolené závislosti jsou reprezentovány zelenými šipkami, zakázané závislosti jsou reprezentovány červenými šipkami. Vidíme, že `Klíč 4` ve vícenásobné sekci závisí na záznamu `Klíč 3` a neboť i ten je ve stejné vícenásobné sekci, je tato závislost povolena. Pokud `Klíč 3` závisí na záznamu `Klíč 2`, který není ve vícenásobné sekci, je tato závislost také v pořádku. Zakázána je ovšem závislost vnějších klíčů na hodnotách klíčů uvnitř vícenásobné sekce. Tedy `Klíč 1` nemůže záviset na záznamu `Klíč 5`.

3.2 Odkazy v šablonovém souboru

Cílem odkazů je využít jednu definici klíčového slova nebo sekce víckrát než jednou.

3.2.1 Šablonový soubor

Jedinou věcí, kterou je třeba v konfiguračním balíku změnit je formát šablonového souboru. Ten rozšíříme o nový typ klíče s názvem `reference`. Podívejme se nyní na ukázku zápisu odkazu z konfiguračního balíku serveru Apache:



Obrázek 3.4: Závislosti ve vícenásobné sekci

```
<reference>
  <entry-name>Files</entry-name>
  <target>/apache-config/Files</target>
</reference>
```

Jak vidíme, u odkazu je třeba definovat název a cestu k cíli.

Předpokládejme, že odkaz může mířit pouze na sekci nebo klíčové slovo, které bylo v šablonovém souboru definováno dříve. Nelze se tedy nejprve odkázat na klíčové slovo a až poté ho definovat. Díky tomu nám k vyhodnocení odkazů bude stačit jen jeden průchod šablonovým souborem, tedy doba zpracování šablonového souboru se výrazně nezvýší.

3.2.2 Změny ve výkonné knihovně

Třída FcTReference

Nejdůležitější změnou v knihovně PyFC je nová třída šablonového stromu `FcTReference`. Třída funguje jako proxy objekt a většinu volání přesměrovává na svůj cíl. Některé vlastnosti a metody bychom ale přesměrovávat neměli, jsou to:

- **name**: U odkazu definujeme jeho vlastní název.
- **parent**: Odkaz na nadřazenou sekci musí být pochopitelně lokální, abychom mohli objekt správně zařadit do šablonového stromu.

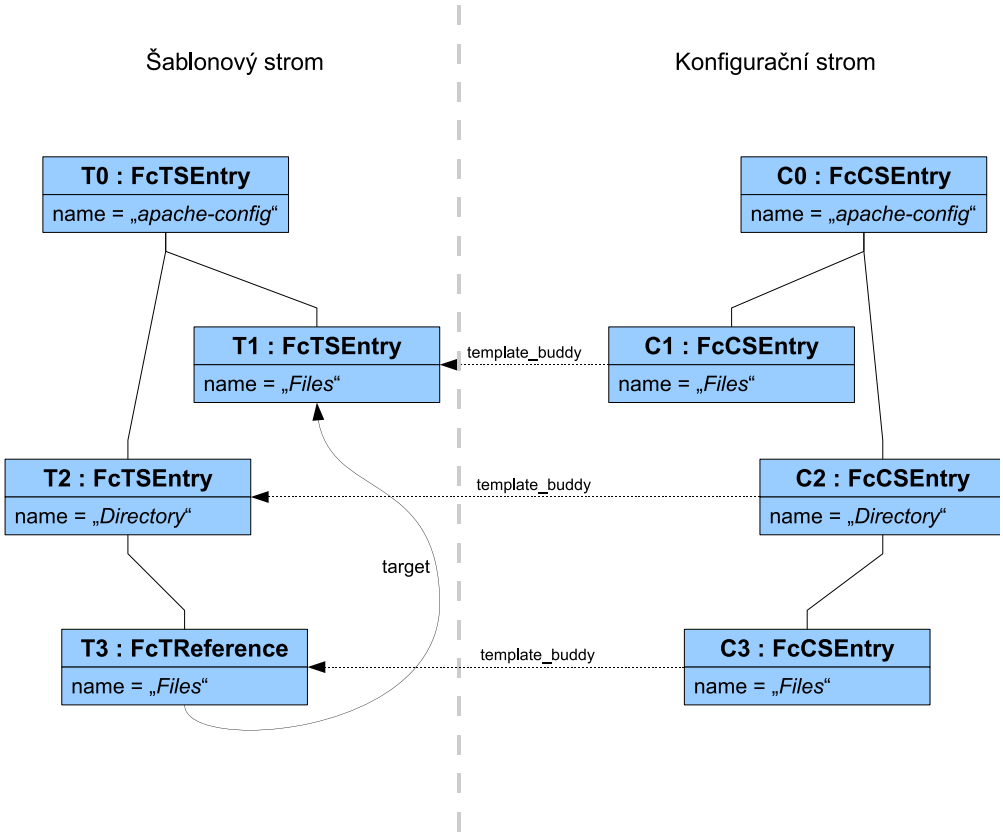
- **active**: Aktivita odkazu je lokální vlastnost, lze ji totiž nastavovat i pomocí závislostí.
- **mandatory**: Ani tuto vlastnost přesměrovávat nebudeme ze stejného důvodu, jaký jsme uvedli u vlastnosti `active`.
- **group**: Odkaz může být definovaný v jiné skupině záznamů než jeho cíl.
- **output()**: Tato metoda vrací XML popis záznamu tak, jak má být uložen do šablonového souboru. Metodu v této třídě přetížíme tak, aby vracela XML definici odkazu.

Vše ostatní přesměrujeme na cílový objekt. V následující ukázce vidíme ty nejzajímavější části kódu třídy:

```
class FcTReference(FcTEntry):
  def __init__(self, proxy = None):
    FcTEntry.__init__(self, proxy)
    self.target = None

  def isReference(self):
    return True
  @property
  def type(self):
    return self.target.type
  def isSection(self):
    return self.target.isSection()
  def isKeyword(self):
    return self.target.isKeyword()

  def keyLabel(self, language = ""):
    return self.target.keyLabel(language)
  def setKeyLabel(self, language, label):
    raise AttributeError("Read only attribute!")

  def _makeCEntry(self):
    return self.target._makeCEntry()

  def __getattr__(self, name):
    return getattr(self.target, name)
```

V konstruktoru nastavíme vlastnost `target`, do které se při načítání šablonového souboru vyplní odkazovaný šablonový objekt. Dále definujeme novou funkci `isReference`, podle které můžeme v šablonovém stromě rozlišit reference od ostatních objektů.

Následuje přesměrování důležitých funkcí a vlastností na cílový objekt. Všimněme si funkce `setKeyLabel`, která běžně nastavuje popisku klíčového slova nebo sekce.



Obrázek 3.5: Použití objektu typu FcTReference

Zde tato funkce vyvolá výjimku `AttributeError`, protože u odkazů je tato vlastnost potřeba pouze pro čtení.

A nakonec nadefinujeme funkci `__getattr__`, jejímž úkolem je přesměrovat zbývající vlastnosti a funkce. Zde jsme chytře využili vlastnosti jazyka Python. `__getattr__` je totiž interní funkce jazyka, která se volá pokaždé, když přistupujeme k prvku objektu, který jazyk Python nezná. Tedy například `obj.a` způsobí volání funkce `obj.__getattr__("a")`. Přetížením této funkce pak můžeme všechna zbývající volání přesměrovat na cílový objekt.

Obraz v konfiguračním stromě

Oproti ostatním typům klíčů nemá třída `FcTReference` svůj obraz v konfiguračním stromě. Při vytváření konfiguračního stromu dojde k přesměrování volání privátní funkce `_makeCEntry`, která je zodpovědná za vytvoření konfiguračního objektu příslušného typu. Pokud tedy odkaz `/apache-config/Directory/Files` míří například na sekci `/apache-config/Files`, pak v konfiguračním stromě bude na místo odkazu vložena přímo instance sekce `Files`. Popsaná situace je znázorněna na diagramu 3.5.

3.3 Skupiny záznamů

Pojmenované skupiny záznamů jsme do projektu Freeconf přidali jako odpověď na potřebu zapisovat z jednoho konfiguračního balíku do více nativních konfiguračních souborů. Princip je jednoduchý: záznamy dělíme do pojmenovaných skupin podle toho, do kterého výstupního souboru se mají zapsat.

3.3.1 Změny v konfiguračním balíku

Hlavičkový soubor

Skupiny záznamů definujeme v hlavičkovém souboru konfiguračního balíku elementem `entry-group`. Zadána by zde měla být alespoň jedna skupina, přičemž každá má unikátní jméno, které zadáváme do atributu `name`. Pro skupinu lze nastavit cestu k nativnímu konfiguračnímu souboru, soubor s XSL transformací a zda se mají do konfiguračního souboru vypisovat i výchozí hodnoty, pokud nebyly změněny (parametr `output-defaults`, více o něm je možné nalézt v práci [7] na straně 68).

Definici skupin si ukážeme na hlavičkovém souboru serveru Apache:

```
<freeconf-header>
<content>
  <template>apache_template.xml</template>
  <default-values>apache_default.xml</default-values>
  <help>apache_help.xml</help>
  <output>$PACKAGE/apache_conf.xml</output>
  <lists>apache_lists.xml</lists>
</content>
<!-- Hlavní konfigurační soubor -->
<entry-group>
  <native-output>$PACKAGE/apache2.conf</native-output>
  <transform>apache_default_transform.xsl</transform>
  <output-defaults>yes</output-defaults>
</entry-group>
<!-- Soubor se seznamem portů -->
<entry-group name="ports">
  <native-output>$PACKAGE/ports.conf</native-output>
  <transform>apache_default_transform.xsl</transform>
  <output-defaults>yes</output-defaults>
</entry-group>
</freeconf-header>
```

Zde jsme vytvořili dvě skupiny, jednu výchozí a jednu určenou pro seznam portů. U výchozí skupiny není třeba uvádět jméno, bude totiž nastaveno na *„default“*. Z toho jasné, že záznam `entry-group` bez argumentu má smysl do hlavičkového souboru zadávat pouze jednou.

Další změny, které ve formátu hlavičkového souboru jsou:

- Elementy `native-output` a `transform` lze již zapisovat pouze uvnitř elementu `entry-group`, kde definují nativní konfigurační soubor skupiny. Jejich umístění v hlavním elementu `content` by totiž už nedávalo smysl.
- Zrušili jsme element `settings` a jeho obsah přesunuli do nového elementu `entry-group`. Parametr `output-defaults`, který zde jako jediný mohl být, má nyní lokální platnost vzhledem ke skupině, ve které je uveden.

Šablonový soubor

Když už máme v balíku nadefinovanou množinu konfiguračních souborů v nativním formátu, můžeme jejím prvkům přiřadit jednotlivé klíče konfigurace. Toho nejlépe dosáhneme rozšířením šablonového souboru o novou vlastnost `group`. Ta bude určovat jméno skupiny, do které klíčové slovo patří.

Abychom vlastnost `group` nemuseli zapisovat ke každému klíčovému slovu, zavedeme ještě dvě pravidla:

1. Pokud je jméno skupiny uvedeno v definici sekce, propaguje se do hloubky na každé podřízené klíčové slovo a sekci.
2. Jestliže u klíčového slova není jméno skupiny uvedeno a není uvedeno ani pro některou z nadřazených sekcí, nastaví se automaticky na jméno výchozí skupiny (tedy *„default“*).

A nakonec ukázka z šablonového souboru:

```
<section>
  <section-name>Listen</section-name>
  <!-- Skupina: Platí pro všechna podřízená klíčová slova -->
  <group>ports</group>

  <!-- Klíčová slova -->
  <!-- ... -->
</section>

<number>
  <entry-name>Port</entry-name>
  <!-- Skupina: Platí pouze pro toto klíčové slovo. -->
  <group>ports</ports>
</number>
```

3.3.2 Změny ve výkonné knihovně

Třída FcGroup

Výkonnou knihovnu projektu Freeconf je třeba rozšířit zejména o novou třídu, kterou pojmenujeme `FcGroup` a uložíme ji do nového souboru `group.py`. Nadefinujeme do ní vše, co musíme uložit při načítání skupiny z hlavičkového souboru. Její základní vlastnosti a funkce tedy budou:

- **name**: Jméno skupiny.
- **transformFile**: Cesta k souboru s transformací.
- **nativeFile**: Cesta k nativnímu konfiguračnímu souboru skupiny.
- **outputDefaults**: Booleovská hodnota. Určuje, zda se do nativního konfiguračního souboru mají vypisovat i hodnoty, které se neliší od výchozích.
- **transform(config_root)**: Funkce vygeneruje nativní konfigurační soubor skupiny záznamů. Jako parametr `config_root` přebírá kořen konfiguračního stromu.

Aby bylo možné skupiny snáze dohledat, uložíme je do proměnné `groups` v objektu, který reprezentuje načtený balík. Proměnná `groups` je typu slovník a jménu skupiny přiřazuje její objekt.

Rozšíření FcTEntry

Jak název napovídá, třída `FcTEntry` je základovou třídou pro všechny objekty v šablonovém stromu, tedy klíčová slova a sekce. Proto se perfektně hodí pro uložení nové vlastnosti `group` z šablonového souboru. Nebudeme zde ovšem ukládat název načtený z definice klíčového slova nebo sekce. Místo toho při načítání šablonového souboru dohledáme objekt příslušné skupiny, který vznikl při načítání hlavičkového souboru, a uložíme odkaz na něj.

Zde také vyřešíme požadavky ze strany 53 na propagaci skupin dovnitř sekcí a na výchozí skupinu. Propagování skupiny do podřízených záznamů uvnitř sekcí provedeme v přístupových funkcích vlastnosti `group`:

```
@property
def group (self):
  if self.__group == None and self.parent != None:
    # Propagace skupiny z nadřazeného záznamu
    return self.parent.group
  return self.__group

@group.setter
def group (self, group):
  self.__group = group
```

Skutečný odkaz na skupinu jsme uložili do soukromé proměnné `__group`. Při získávání hodnoty vlastnosti `group` nejprve zkontrolujeme, zda je nastavena privátní proměnná a pokud ano, vrátíme její hodnotu. V případě, že nastavena není, vrátíme hodnotu skupiny nadřazené sekce.

Řešení druhého požadavku na výchozí skupiny je již triviální. Postačí, když u kořenové sekce napevno nastavíme vlastnost `group` na výchozí skupinu. Díky tomu se nám automaticky zpropaguje do všech záznamů, pro které nebyla skupina explicitně definována v šablonovém souboru.

Zápis nativních konfiguračních souborů

Jak bylo uvedeno výše, zápis nativních konfiguračních souborů probíhá ve funkci `transform` třídy `FcGroup`. Zde ale vznikne malý problém. Generování nativních souborů totiž funguje tak, že na konfigurační soubor ve formátu XML pustíme zadanou XSL transformaci. Jenže konfigurační soubor ve formátu XML stále ukládáme pouze jeden, zatímco nativních konfiguračních souborů může být více, každý s jinou podmnožinou klíčů.

Samozřejmě bychom mohli rozdělit i konfigurační soubor ve formátu XML a hodnoty klíčů ukládat pro každou skupinu zvlášť. To by nám ovšem znepřehlednilo konfigurační balík dalšími zbytečnými soubory. Máme totiž ještě jednu možnost: Konfigurační XML soubory se záznamy pro skupiny vygenerovat pouze dočasně, nejlépe do operační paměti počítače. Po zapsání nativního konfiguračního souboru dočasný XML dokument opět smažeme.

Naštěstí nám knihovna `libxslt`, která je pro transformace použita, umožňuje provést XSL transformaci i nad řetězcem uloženým v proměnné. Stačí tedy vybrat klíče dané skupiny a jejich hodnoty ve formátu XML zapsat do proměnné, kterou pak předhodíme transformační funkci.

Tato změna nám také dovolí snížení velikosti konfiguračního souboru ve formátu XML, který je třeba ukládat na disk. Původně se totiž do tohoto souboru ukládal i text nápovědy klíčových slov, aby bylo možné vygenerovat nápovědu do komentářů v nativním konfiguračním souboru. Tento text bývá obvykle mnohonásobně delší, než samotná hodnota klíčového slova, a tak jen zbytečně nafukuje velikost konfiguračního XML souboru. Navíc máme tutéž informaci dostupnou i v souborech s nápovědou a stejný text byl proto v balíku uložen dvakrát. Těmto duplicitám můžeme nyní jednoduše zabránit. Stačí, když text nápovědy budeme generovat jen do XML dokumentu v paměti, který se pak použije pro XSL transformaci. Při ukládání XML konfiguračního souboru na disk již můžeme zapsat pouze nezbytné informace o stavu konfigurace.

Rozdíl bude nejlépe patrný na ukázce. Zde je úryvek konfigurace v XML formátu se všemi informacemi tak, jak se vygeneruje do paměti:

```
<entry name="ServerTokens">
  <value>Full</value>
  <help>This directive configures what you return as the Server HTTP
```

```
responseHeader. The default is &apos;Full&apos; which sends
information about the OS-Type and compiled in modules.</help>
  <type>string</type>
</entry>
```

A zde je tatáž konfigurace v podobě, v jaké se zapíše do konfiguračního souboru na disku:

```
<entry name="ServerTokens">
  <value>Full</value>
</entry>
```

Úspora místa je patrná na první pohled.

3.4 Rozšiřující moduly

Cílem rozšiřujících modulů (dále jen modulů) je přidat do již existujícího konfiguračního balíku nové klíče a vše co s nimi souvisí. Moduly by mělo být možné snadno nainstalovat i odinstalovat, neměly by tedy měnit žádný ze stávajících konfiguračních souborů v balíku.

3.4.1 Návrh modulu

Během návrhu jsme si nejprve sepsali požadavky na soubory a atributy balíku, které by moduly mohly rozšiřovat:

1. Šablonový soubor: Lze přidávat nové klíče a sekce. Měnit vlastnosti stávajících klíčů je zakázáno.
2. Soubor s nápovědou: Je třeba definovat popisky a nápovědu pro nové klíče.
3. Výchozí hodnoty: Nové klíče potřebují výchozí hodnoty.
4. Konfigurační soubor ve formátu XML: Aktuální hodnoty nových klíčů je třeba uložit do odděleného konfiguračního souboru. Zde jsme nejprve předpokládali, že klíče modulu budou zapsány do konfiguračního XML souboru hlavního balíku. V průběhu prvotní implementace se ale ukázalo, že takové řešení není nejvhodnější, neboť při odstranění nebo přesunutí modulu nám v hlavním konfiguračním XML souboru zůstávaly záznamy pro neexistující klíčová slova.
5. XSL transformace: Musíme mít možnost rozšířit stávající transformaci o zpracování nových klíčů a sekcí.
6. Nové skupiny záznamů: Modul může požadovat, aby jeho konfigurační hodnoty byly ukládány do odděleného nativního konfiguračního souboru s novou XSL transformací.

7. Seznamy hodnot: Lze nadefinovat seznamy hodnot pro nové klíče.
8. Závislosti: Nové klíče mohou záviset na hodnotách stávajících klíčů a naopak.
9. Šablonový soubor pro GUI: Bude potřeba definovat zobrazení nových klíčů a sekcí.
10. A samozřejmě budeme také potřebovat popisky nového šablonového souboru pro GUI.

Z toho vidíme, že modul má schopnost rozšířit takřka všechny soubory v konfiguračním balíku. Proto jsme se rozhodli zapisovat moduly stejným způsobem jako konfigurační balíky. Formáty souborů zůstanou až na jednu výjimku nezměněny.

Tyto konfigurační balíky modulů uložíme do adresáře `plugins` (anglický termín pro rozšiřující moduly), který umístíme do hlavního konfiguračního balíku. Odtud pak budou moduly načítány výkonnou knihovnou. Tímto způsobem také získáme snadnou cestu, jak moduly nainstalovat a odinstalovat. Instalace modulu do konfiguračního balíku se rovná přesunutí modulu do adresáře `plugins` v konfiguračním balíku. K odinstalování modulu nám pak postačí modul z tohoto adresáře smazat či přesunout jinam.

Hlavičkový soubor

Jedinou změnu formátu v balíku modulu oproti standardnímu konfiguračnímu balíku provedeme v hlavičkovém souboru `header.xml`, a to kvůli požadavku číslo 5. Ten říká, že bychom v modulu měli být schopni rozšířit stávající XSL transformaci. Na přesný způsob rozšíření transformace se podíváme v kapitole 3.4.2. Zde si pouze ukážeme, jak upravíme hlavičkový soubor, aby toto rozšíření podporoval. Do něho musíme zapsat cestu k souboru, který rozšíří stávající XSL transformaci. A jak víme z kapitoly 3.3.1, cesta k souboru s XSL transformací je součástí definice skupiny. Přidáme tedy podporu pro modifikaci obsahu skupiny v hlavičkovém souboru modulu následujícím způsobem:

```
<freeconf-header>
  <content>
    <template>ssl_template.xml</template>
    <default-values>ssl_default.xml</default-values>
    <help>ssl_help.xml</help>
    <output>$PLUGIN/ssl_conf.xml</output>
    <gui-template>ssl_gui_template.xml</gui-template>
  </content>
  <change-group name="default">
    <add-transform>ssl_default_transform.xsl</add-transform>
  </change-group>
</freeconf-header>
```

Příklad je z rozšiřujícího modulu SSL pro webový server Apache. Přidali jsme nový element `change-group`, který modifikuje existující skupinu `default` z konfiguračního balíku serveru Apache. Jméno skupiny zapisujeme opět do atributu `name`. Uvnitř tohoto elementu definujeme vlastnosti skupiny, které chceme změnit. Zatím budeme podporovat pouze rozšíření transformace o nová pravidla ze souboru, jehož jméno je zadané v elementu `add-transform`.

Podpora nahrazování proměnných

V dřívějších verzích formátu konfiguračního balíku existovala možnost odkázat se pomocí zástupných znaků na adresář konfiguračního balíku (znak $) a na uživatelský adresář (znak ∼). Tyto speciální znaky se pak automaticky nahradily správnými cestami v souborovém systému. Nahrazování bylo podporováno v cestě ke konfiguračnímu souboru ve formátu XML a k nativnímu konfiguračnímu souboru. S příchodem zásuvných modulů by bylo vhodné se podobným způsobem odkázat i na cestu k aktuálnímu zásuvnému modulu, což by znamenalo zavedení dalšího zástupného znaku.

Při vývoji jsme ovšem zjistili, že takovéto speciální znaky se špatně pamatují. Zejména proto, že neexistuje zřejmá logická vazba mezi zástupným znakem a adresářem, který zastupuje. Rozhodli se tedy, že zrušíme podporu zástupných znaků a nahradíme je odkazy na proměnné, podobně jako to funguje například v příkazovém řádku operačního systému GNU/Linux. Odkaz na proměnnou zapisujeme jako `$`*`NÁZEV`* nebo `${`*`NÁZEV`*`}`, kde *NÁZEV* je jméno proměnné.

V cestách k souborům se budou automaticky nahrazovat veškeré systémové proměnné definované v prostředí programu a následující proměnné specifické pro projekt Freeconf:

- **$PACKAGE**: Cesta ke konfiguračnímu balíku.
- **$PLUGIN**: Cesta k aktuálnímu rozšiřujícímu modulu. Pokud aktuální rozšiřující modul není definován, má tato proměnná stejný význam jako `$PACKAGE`. To nastane například odkážeme-li se na proměnnou `$PLUGIN` z hlavičkového souboru hlavního konfiguračního balíku.

Na domovský adresář uživatele se lze odkázat systémovou proměnnou `$HOME`.

Nyní si ukážeme jednoduchou funkci v jazyce Python, která provádí samotné nahrazování systémových proměnných. Jejím povinným argumentem je textový řetězec, v němž budeme nahrazovat proměnné. Jako volitelný argument lze předat seznam dodatečných proměnných. V něm se budou předávat specifické proměnné projektu Freeconf.

```
def expandVariables(string, variables = {}):
  for match in re.finditer(r"\$\{?(\w*)\}?", string):
    var = match.group(1)
```

```
    val = None
    if var in variables:
      val = variables[var]
    else:
      val = getEnv(var)
    if val == None:
      # Proměnná nenalezna
      val = ""
    # Nahrazení proměnné v řetězci
    string = string.replace(match.group(0), val)
  return string
```

Ve funkci nejprve nalezneme všechny shluky znaků ve tvaru `$NÁZEV` nebo `${NÁZEV}`. Název proměnné může obsahovat pouze alfanumerické znaky a podtržítka. Dále se pro nalezené proměnné pokusíme dohledat jejich hodnotu. Nejprve prohledáme zadaný seznam specifických proměnných a když neuspějeme, pokusíme se proměnnou nalézt v systému. Pokud ani tam nebyla proměnná nalezena, nahradíme ji prázdným řetězcem.

3.4.2 Rozšíření XSL transformace

V kapitole 3.4.1 jsme si řekli, že modul by měl být schopen rozšířit stávající transformaci svým vlastním XSL souborem. Zde by mohly být například výjimky ve zpracování sekcí či složených konfiguračních záznamů v modulu podobné těm, na které jsme narazili v konfiguraci serveru Apache.

Naštěstí je jazyk XSL na podobný scénář připraven a umožňuje nám do sebe vkládat jednotlivé transformační soubory direktivou `include`. Její zápis je následující:

```
<xsl:include href="cesta k XSL souboru" />
```

Na místo `<xsl:include ... />` se pak do transformace vloží pravidla z odkazovaného XSL souboru.

Při spojování XSL transformací budeme tedy postupovat následovně: Výkonná knihovna vytvoří pro každou skupinu záznamů dočasnou XSL transformaci, do které pomocí direktivy `include` vloží odkazy na všechny transformační soubory skupiny. Jako první bude vložen odkaz na XSL transformaci konfiguračního balíku. Následovat budou odkazy na transformační XSL soubory definované v modulech.

Tím, že odkazy na transformace modulů vložíme až za hlavní transformaci balíku, budeme moci přidávat výjimky do stávajících pravidel balíku, a nebo je dokonce předefinovat. Pokud je totiž pro jeden XML element pravidel v XSL souboru více, platí vždy až to poslední.

Podívejme se nyní na ukázku z XSL souboru `ssl_default_transform.xsl` výchozí skupiny modulu SSL, který přidává výjimku pro složený záznam `SSLOptions`:

```
<?xml version="1.0"?>
<xsl:stylesheet
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     version="1.0">

<xsl:template match="section[@name = 'SSLOptions']">
  <!-- Výjimka pro sekci SSLOptions -->
  <xsl:value-of select="@name" />
  <!-- Zpracování seznamu voleb -->
  <xsl:for-each select="entry">
    <xsl:value-of select="$key-value-separator" />
    <xsl:if test="value = 'yes'">
      <!-- Vypíšeme název zapnuté volby -->
      <xsl:value-of select="@name" />
    </xsl:if>
  </xsl:for-each>
  <xsl:call-template name="new-lines"/>
</xsl:template>

</xsl:stylesheet>
```

Klíčovou roli v této šabloně hraje atribut `match` elementu `xsl:template`. V něm určíme, že chceme zpracovat pouze sekci s názvem `SSLOptions`. Kdybychom podmínku na název nezadali, nová šablona by se aplikovala na všechny sekce balíku Apache a to by způsobilo vygenerování neplatné konfigurace.

3.4.3 Změny ve výkonné knihovně

Podpora rozšiřujících modulů si vyžádala celou řadu změn zdrojového kódu výkonné knihovny. Nebudeme zde rozebírat všechny změny, neboť to by bylo mimo rozsah této práce. Popíšeme si pouze ty nejvýznamnější úpravy. Všechny změny kódu je možné nalézt v SVN repozitáři projektu. Více o něm se dočtete v příloze A.

Třídy na zpracování balíku a modulů

Nejvíce času nám zabraly úpravy třídy `FcPackage` v souboru knihovny `package.py`. Zde se soustředí kód pro řízení operací prováděných nad konfiguračním balíkem. Jde zejména o načítání a ukládání balíku. Uchovávají se zde také odkazy na všechny datové struktury balíku: šablonový, konfigurační a grafický strom objektů, závislosti mezi klíčovými slovy, seznam skupin balíku, dostupné seznamy hodnot a aktuálně dostupné jazyky.

Pro práci s moduly jsme vytvořili novou třídu `FcPlugin`, která je zodpovědná za načítání konfigurace modulů. Jak jsme viděli v předchozích odstavcích, rozšiřující moduly se svou strukturou velice blíží konfiguračnímu balíku. Proto bylo možné

velkou část kódu třídy `FcPackage` využít i pro načítání modulů. Společnou funkcionalitu obou tříd jsme upravili, aby byla použitelná jak pro balíky tak pro moduly, a přesunuli do základové třídy `FcPackageBase`. Jde zejména o následující metody:

- **addGroup(group)**: Vloží novou skupinu do balíku nebo modulu.
- **removeGroup(name)**: Odstraní skupinu daného jména z balíku nebo modulu.
- **expandFileName(path)**: Nahradí proměnné v cestě `path`. Proměnná `PLUGIN` bude nahrazena správnou hodnotou v závislosti na tom, zda jde o balík, či modul. Zpracování ostatních proměnných se neliší.
- **loadPacakgeBase()**: Načítá soubory konfiguračního balíku nebo modulu. Postupně zde dojde k načtení hlavičkového souboru, seznamů hodnot, šablonového souboru, souborů s nápovědou, výchozích hodnot, konfiguračního souboru, souboru se závislostmi a v nakonec souborů s definicí uživatelského rozhraní. Před voláním této metody je třeba správně nastavit datové struktury třídy `FcPackageBase`. Díky nim pak metoda `loadPacakgeBase` rozezná, odkud má číst soubory balíku (resp. modulu) a kam má uložit načtené hodnoty.
- **writeOutput()**: Zapíše aktuální hodnoty klíčů balíku (resp. modulu) do konfiguračního XML souboru.
- **transform()**: Zapíše nativní konfigurační soubory pro všechny skupiny balíku nebo modulu.

Abychom omezili větvení kódu ve společných metodách třídy `FcPackageBase`, deklarovali jsme si následující abstraktní vlastnosti. Ty bude třeba nadefinovat zvlášť pro třídu konfiguračního balíku a zvlášť pro třídu modulu:

- **availableLists**: Vrátí seznam aktuálně dostupných seznamů hodnot. V konfiguračním balíku vlastnost vrátí seznamy hodnot balíku. V modulu musí tato vlastnost vrátit jak seznamy hodnot balíku tak i lokální seznamy hodnot modulu.
- **availableGroups**: Vrátí seznam aktuálně dostupných skupin. Chování této vlastnosti se liší pro balík a modul podobně jako tomu bylo u vlastnosti `availableLists`.

Do základové třídy jsme ještě přidali abstraktní metodu `isPlugin`, která vrátí pravdu, pokud instance třídy reprezentuje modul, a nepravdu, pokud zastupuje konfigurační balík.

Dále bylo nutné vyrovnat se s pomocnými datovými strukturami původní třídy `FcPackage`: `PackagePaths`, `PackageTrees` a `PackageData`. Struktura `FcPackage` obsahuje cesty k nejrůznějším částem souborů, které jsou potřeba pří načítání konfiguračního balíku. Struktura `PackageTrees` zase obsahuje reference na jednotlivé

stromy objektů balíku. A nakonec struktura `PackageData` sdružuje ostatní informace uložené v balíku jako například skupiny, seznamy hodnot nebo závislosti.

Původně byly tyto struktury nadefinované na nejvyšší úrovni v souboru `package.py` hned vedle třídy `FcPackage`. Nicméně pro lepší využití dědičnosti jsme se rozhodli zapsat definici struktur do samotné třídy `FcPackageBase` jako podtřídy `Paths`, `Trees` a `Data`. Třídy balíku a pluginu pak mohou snadno upravit jejich chování odvozením nové třídy stejného jména.

Největší rozdíly jsou v pomocné třídě `Paths`, která odpovídá původní struktuře `PackagePaths`. Zde uložené cesty k adresářům a souborům jsou totiž pro balík a modul značně odlišné, jako například cesta k hlavičkovému souboru. Ale některé informace jsou zase téměř stejné, jako například seznam cest k systémovým adresářům projektu Freeconf. Toho snadno dosáhneme dědičností.

Podívejme se nyní na definici podtřídy `Paths` ve třídě `FcPackageBase`:

```
class FcPackageBase:
  class Paths:
    def __init__ (self):
      self.packageName = ""
      self.packageDir = ""
      # Dostupné adresáře
      self.freeconfDirs = []
      self.helpDirs = {}
      self.listDirs = []
      self.defaultValuesDirs = []
      # Cesty k souborům balíku
      self.listFiles = {}
      self.templateFile = FcFileLocation()
      self.dependenciesFile = FcFileLocation()
      self.helpFile = FcFileLocation()
      self.defaultValuesFile = FcFileLocation()
      self.outputFile = FcFileLocation()
      self.guiTemplateFile = FcFileLocation()
      self.guiLabelFile = FcFileLocation()

    @property
    def mainDir(self):
      return self.packageDir
    @property
    def headerFileFullPath(self):
      return self.mainDir + '/header.xml'

    def fill(self, path):
      raise NotImplementedError # Abstraktní funkce

  # Zbytek třídy FcPackageBase ...
```

V konstruktoru deklarujeme jméno balíku, cestu k němu a seznamy dostupných adresářů, kde se budou hledat soubory. Dále jsme vytvořili proměnné, do kterých se zapíší aktuální cesty k souborům balíku nebo modulu jako struktura typu `FcFileLocation`. Ta obsahuje dvě položky: název souboru a plnou cestu k jeho aktuálnímu umístění. Tyto struktury budou naplněny ve funkci `loadPacakgeBase` hned po načtení hlavičkového souboru.

Dále jsme zavedli pomocnou vlastnost `mainDir`, která vrátí hlavní cestu k balíku nebo modulu. Její použití vidíme hned v následující vlastnosti `headerFileFullPath`, kde se konstruuje cesta k hlavičkovému souboru.

A nakonec jsme nadeklarovali abstraktní funkci `fill`. Ta nám poslouží k naplnění seznamů dostupných adresářů. Jejím argumentem je cesta k balíku nebo modulu. Přetížení podtřídy `Paths` a zmíněnou funkci si ukážeme ve třídě konfiguračního balíku `FcPackage`. Jen upozorním, že pro snadnější pochopení jsme zde funkci `fill` poněkud zjednodušili a vyzdvihli nejdůležitější části, v aktuální podobě výkonné knihovny obsahuje mnohem více kódu.

```
class FcPackage (FcPackageBase):
  class Paths(FcPackageBase.Paths):
    def __init__(self):
      FcPackageBase.Paths.__init__(self)

    def fill(self, path):
      self.packageDir = path
      homeDir = getEnv('HOME')
      # Systémové adresáře projektu Freeconf
      self.freeconfDirs = [
        homeDir + '/.freeconf',
        '/usr/local/share/freeconf',
        '/usr/share/freeconf'
      ]

      # Naplní seznam možných umístění seznamů hodnot
      self.listDirs = [
        dir + '/lists'
          for dir in [self.packageDir] + self.freeconfDirs
      ]

      # Naplnění dalších seznamů adresářů ...

  # Zbytek třídy FcPackage ...
```

Jak vidíme, ve funkci sestavíme seznam systémových adresářů a ten pak použijeme při naplnění seznamu možných umístění souborů se seznamy hodnot. Ty se mohou vyskytovat jak v systémových adresářích tak v podadresáři `lists` konfiguračního balíku.

V rozšiřujících modulech ale podle požadavku číslo 7. v kapitole 3.4.1 máme možnost nadefinovat vlastní seznamy hodnot. Ty se ukládají do podadresáře `lists` v cestě modulu. Tento problém jsme vyřešili odvozením podtřídy `Paths` ve třídě modulu `FcPackage`:

```
class FcPlugin(FcPackageBase):
  class Paths (FcPackageBase.Paths):
    def __init__ (self, paths):
      FcPackageBase.Paths.__init__(self)
      # Cesta k modulu
      self.pluginDir = ""
      # Inicializace z cest balíku
      self.homeDir = paths.homeDir
      self.packageName = paths.packageName
      self.packageDir = paths.packageDir
      self.listDirs = paths.listDirs
      self.scriptDirs = paths.scriptDirs
      self.freeconfDirs = paths.freeconfDirs
      self.defaultValuesDirs = self.defaultValuesDirs

    @property
    def mainDir(self):
      return self.pluginDir

    def fill(self, path):
      self.pluginDir = path
      # Rozšíří možná umístění seznamů hodnot
      self.listDirs.insert(0, self.pluginDir + '/lists')

      # Rozšíření ostatních seznamů adresářů ...

  # Zbytek třídy FcPlugin ...
```

Konstruktor podtřídy `Paths` zde bere jako argument instanci `Paths` konfiguračního balíku. Z ní se nainicializují seznamy adresářů. Tyto seznamy pak rozšíříme v metodě `fill`, kde budou doplněny o lokální cesty modulu. Celá funkcionalita je opět znázorněna na seznamu adresářů možných umístění seznamů hodnot. Jak vidíme, lokální adresář `lists` modulu vkládáme na první místo seznamu, kde bude mít nejvyšší prioritu.

Rozšíření třídy FcGroup

Do rozhraní třídy `FcGroup`, které jsme si představili v kapitole 3.3.2, jsme doplnili novou metodu `includeTransform` s parametrem file typu `FcFileLocation`. Tato metoda vloží do skupiny XSL transformaci modulu.

Sestavení dočasné transformace, jejíž princip jsme popsali v kapitole 3.4.2, provedeme v metodě `transform` poměrně triviálním kódem:

```
xslString = '<?xml version="1.0" ?>
             <xsl:stylesheet
               xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
               version="1.0">'
# Vložíme odkaz na hlavní XSL soubor balíku
xslString +=
  '<xsl:include href="%s"/>' % (self.transformFile.fullPath,)
# Vložení odkazů na XSL soubory modulů
for i in self.includedTransformFiles:
  xslString += '<xsl:include href="%s"/>' % (i.fullPath,)
# Ukončení transformace
xslString += '</xsl:stylesheet>'
```

Dočasnou XSL transformaci tedy ukládáme pouze do řetězce v paměti, podobně jako jsme to udělali s dočasným konfiguračním souborem ve formátu XML v kapitole 3.3.2 na straně 55.

Rozšíření FcTEntry

Aby bylo možné rozhodnout, do kterého modulu či balíku patří dané klíčové slovo nebo sekce, rozšířili jsme šablonový strom o novou vlastnost `package`. Tu jsme umístili do třídy `FcTEntry`, která je základovou třídou pro všechny objekty v šablonovém stromu. Hodnotou nové vlastnosti je reference na konfigurační balík nebo rozšiřující modul. Musí tedy jít o instanci třídy odvozené z `FcPackageBase`.

Kapitola 4

Podpora skriptování

4.1 Úvod

Při zpracování konfigurace programů projektem Freeconf se čas od času vyskytnou požadavky na zavedení nových, a často dosti specifických, funkcionalit. Je vždy na pečlivém uvážení vývojářů, zda podporu požadované funkce do projektu doplnit či odmítnout. Pokud jde o dostatečně obecný požadavek, který by se mohl vyskytnout i v konfiguraci jiných programů, obvykle se snažíme jej do projektu zařadit. Například díky konfiguraci serveru Apache jsme přidali hned 4 nové funkce.

Mohou ale nastat situace, kdy požadovaná funkcionalita je příliš specifická na to, aby mohla být zařazena do návrhu projektu. Přesto ji vývojář konfiguračního balíku musí nějak vyřešit.

Odpovědí na tyto problémy je právě podpora spustitelných skriptů. Základní myšlenka je taková, že v daných situacích při běhu konfigurační aplikace dojde k zavolání skriptu[1] nebo jiného spustitelného programu, který určí vývojář balíku. Tento program pak může na základě libovolné logiky modifikovat konfiguraci balíku bez zásahu uživatele.

4.2 Příklady

Podívejme se nyní na několik konkrétních příkladů, kdy by se podpora spustitelných skriptů mohla hodit.

[1] Skript je počítačový program napsaný v interpretovaném programovacím jazyce, který je navržen s ohledem na snadné zvládnutí jazyka a rychlý vývoj krátkých programů. Takové programovací jazyky nazýváme skriptovací jazyky.

4.2.1 Autodetekce výchozích hodnot

Výchozí hodnoty zajišťují, že konfigurace je v konzistentním stavu i při prvním spuštění konfigurační aplikace. To ale vůbec nemusí znamenat, že je nastavení programu správné. Výchozí hodnoty jsou totiž pevně dané vývojářem balíku. Mohou ale nastat situace, kdy výchozí hodnota je silně závislá na nastavení systému uživatele a je potřeba ji ze systému získat.

Například v konfiguraci jádra operačního systému Linux se vyskytují klíče, které nastavují typ procesoru. V současnosti je uživatel musí vyplnit ručně, nicméně dalo by se uvažovat o jednoduchém skriptu, který by typ procesu zjistil automaticky a uživateli nabídl detekované hodnoty.

Pro demonstraci této vlastnosti skriptů zde použijeme ještě jednodušší příklad, a to nastavení jména serveru v konfiguraci Apache. To je uloženo v konfiguračním klíči `ServerName`. Jako výchozí hodnotu bychom rádi použili jméno počítače, na kterém je spuštěna konfigurační aplikace. V operačním systému GNU/Linux lze jméno počítače zjistit příkazem:

```
uname -n
```

Výstup tohoto příkazu tedy potřebujeme přesměrovat do výkonné knihovny projektu Freeconf a uložit jako výchozí hodnotu klíče `ServerName`.

4.2.2 Výběr tabulky v databázi

Další případ, který jsme při návrhu podpory skriptování uvažovali je proces výběru jména tabulky v databázi. Na to lze narazit například v konfiguraci e-mailového serveru `Courier`, který může používat tabulku v databázi MySQL jako zdroj informací o aktivních e-mailových schránkách. Podívejme se na úryvek z konfiguračního souboru, který připojení k databázi nastavuje:

```
# Jméno databázového uživatele
MYSQL_USERNAME mail
# Heslo pro přístup k serveru
MYSQL_PASSWORD Heslo
# Jméno databáze v MySQL
MYSQL_DATABASE maildb
# Jméno tabulky se seznamem uživatelů
MYSQL_USER_TABLE users
```

Uživatel zde musí nejprve zadat jméno a heslo pro připojení k databázovému serveru. Poté vybere vlastní databázi a požadovanou tabulku. Protože databází je v systému přeci jen omezené množství, bylo by vhodné, aby konfigurační aplikace uživateli nabídla seznam dostupných databází. Stejně tak i seznam tabulek v má jen omezený počet položek, které navíc závisí na hodnotě klíčového slova `MYSQL_DATABASE`. Tady

si ovšem se závislostmi projektu Freeconf nevystačíme, neboť k zjištění dostupných tabulek je třeba se nejprve připojit k nastavené databázi a spustit na ní dotaz vypisující tabulky. Něco takového je mimo schopnosti současné implementace závislostí a není ani v plánu takovou funkcionalitu doplnit.

Potřebovali bychom tedy skript, který by se spustil po změně klíčových slov `MYSQL_USERNAME` a `MYSQL_PASSWORD`, načetl seznam databází a nastavil jej jako seznam hodnot klíčového slova `MYSQL_DATABASE`. Po výběru databáze by pak skript načetl i seznam dostupných tabulek a nastavil jej jako seznam hodnot klíče `MYSQL_USER_TABLE`.

4.2.3 Propojení s jinými konfiguračními systémy

Ne všechny aplikace své nastavení ukládají do textových konfiguračních souborů. V úvodu jsme zmínili například registr operačního systému Windows, kde se nastavení programů ukládají do sady binárních souborů, ke kterým aplikace přistupují přes systémové funkce. Konfigurační systém gconf zase k ukládání hodnot používá adresářovou strukturu, kde adresáře hrají roli sekcí a klíče jsou uloženy do souborů ve formátu XML. A další programy mohou k ukládání nastavení využívat například databázi.

Také propojení s těmito konfiguračními systémy by nám podpora spustitelných skriptů měla umožnit.

4.3 Možné nevýhody

Koncept podpory spustitelných skriptů s sebou oproti vestavěné funkcionalitě knihovny přináší jisté nevýhody. Vývojáři konfiguračního balíku musí při využití skriptování zastat i práci programátorů, což může zpomalit vývoj balíku. Zde bychom jim mohli částečně ušetřit práci vytvořením knihovny v nejpoužívanějších skriptovacích jazycích, která by podporovala rozhraní projektu Freeconf. Také by se dalo uvažovat o vytvoření znovupoužitelné sady skriptů na nejběžnější úkoly.

Další nevýhodu skriptování spatřujeme v možném prodloužení doby odezvy konfigurační aplikace. Spuštění a ukončení externího skriptu / programu znamená jistou režii navíc.

Z uvedených důvodů by vývojáři konfiguračních balíků měli k použití skriptů přistupovat teprve tehdy, když všechny ostatní postupy, jako XSL transformace nebo vestavěné závislosti, selžou.

4.4 Požadavky

Na základě předchozích příkladů jsme sestavili sadu požadavků, z nichž by měl návrh podpory skriptů vycházet. Nejprve uvedeme obecné požadavky, které jsme se snažili dodržet:

1. Rozhraní komunikace výkonné knihovny se skripty by mělo probíhat ve formátu XML podobně, jako funguje například protokol SOAP[2] pro komunikaci s webovými službami.

2. Rozhraní by mělo být co nejvíce podobné stávajícím souborům balíku ve formátu XML, aby si na něj vývojáři balíku snadno zvykli.

3. Objem komunikace a velikost zpráv mezi výkonnou knihovnou a skripty bychom rádi drželi co nejmenší, aby zpracování zpráv zabralo co nejméně času.

4.4.1 Co mohou skripty ovlivňovat?

Další skupina požadavků, které jsme si stanovili, nám říká, do jakých částí konfigurace dovolíme spustitelným skriptům zasahovat. Nejprve uvedeme ty části do konfigurace, jež je potřeba měnit v příkladech uvedených v kapitole 4.2.

1. **Výchozí hodnoty**: Příklad 4.2.1 vyžaduje nastavení výchozí hodnoty na detekovanou hodnotu.

2. **Hodnoty klíčových slov**: Při vyčítání nastavení z jiného konfiguračního systému v příkladu 4.2.3 bychom měli být schopni přímo nastavit hodnoty klíčových slov.

3. **Seznamy řetězců**: Pokud má mít uživatel možnost vybrat tabulku z načteného seznamu, musí být skript schopen předefinovat hodnoty v existujícím seznamu řetězců.

Obecně budeme předpokládat, že skripty by měly být schopné nastavovat všechny vlastnosti klíčových slov, které dávají smysl, včetně rozsahu přípustných hodnot, aktivity a povinnosti.

4.4.2 Kdy spouštět skripty?

Z uvedených příkladů vidíme, že skripty je třeba spouštět v nejrůznějších situacích.

1. **Po spuštění**: Automatická detekce výchozích hodnot by měla být provedena co nejdříve po spuštění konfigurační aplikace. Takový okamžik nastane po načtení konfiguračního balíku.

2. **Po uložení**: Pokud chceme konfigurační hodnoty zapsat i do nějakého externího konfiguračního systému, bylo by vhodné to udělat po zapsání aktuálního stavu konfigurace na disk.

[2]SOAP (Simple Object Access Protocol) je protokolem pro výměnu zpráv založených na XML přes síť, hlavně pomocí HTTP. Tvoří základní vrstvu komunikace mezi webovými službami.

3. **Po změně hodnoty klíčového slova**: V příkladu 4.2.2 potřebujeme dynamicky reagovat na akce uživatele. Po změně hodnoty klíčového slova `MYSQL_DATABASE` je třeba spustit skript na načtení aktuálního seznamu tabulek.

Tlačítka obnovy

Stav operačního systému se v průběhu konfigurace může měnit. Uživatel může například změnit název počítače a bude očekávat, že nová hodnota se objeví i v konfigurační aplikaci systému Freeconf. My bohužel nemáme kontrolu nad operačním systémem a nepřetržité sledování změn nastavení systému by bylo velice náročné na zdroje a mimo rozsah působení aplikace Freeconf. Rozhodli jsme se proto, že do konfigurační aplikace přidáme možnost definovat tlačítko „Obnovit“, jehož stisknutí způsobí spuštění zadaných skriptů a obnovu jimi generovaných hodnot. Pokud uživatel uvidí, že detekované hodnoty nesouhlasí s nastavením systému, stiskem tlačítka tyto hodnoty obnoví.

Tlačítko obnovy by se mělo zobrazit u klíčových slov, jež to vyžadují. Mohli bychom ho tedy zobrazit například u klíčového slova `ServerName` z příkladu 4.2.1. Takové tlačítko budeme nazývat lokální tlačítko obnovy. Lokální proto, že se zobrazuje přímo u klíčových slov.

Mohou ale nastat i situace, kdy by uživatel potřeboval najednou obnovit všechny hodnoty generované skripty. Vyžadovat, aby uživatel v takovém případě postupně aktivoval všechna lokální tlačítka obnovy by se zřejmě nesetkalo s velkým ohlasem. Proto rozšíříme konfigurační aplikaci ještě o jedno tlačítko obnovy, které způsobí spuštění všech potřebných skriptů. Takové tlačítko budeme nazývat globální tlačítko obnovy.

4.4.3 Jaké informace předat?

Spustitelné skripty ke svému rozhodnutí mohou vyžadovat znalost aktuálního stavu konfigurace. Například skript pro načtení seznamu dostupných tabulek z databáze jistě potřebuje znát hodnoty klíčových slov, které se týkají nastavení spojení s databází. A skript, který by ukládal nastavení do externího konfiguračního systému z příkladu 4.2.3, dokonce vyžaduje znalost hodnot všech klíčových slov.

Skriptu tedy budeme předávat aktuální hodnoty klíčových slov. Aby nedocházelo ke zbytečnému přenosu hodnot, které nejsou potřeba, budeme po tvůrci skriptu požadovat, aby specifikoval, která klíčová slova skript obdrží. Předávání dalších informací skriptu nyní nebudeme uvažovat. Nicméně návrh by měl být natolik obecný, aby jejich případné doplnění do komunikačního protokolu nepředstavovalo velký problém.

4.5 Návrh rozhraní

Nyní přistoupíme k návrhu samotného komunikačního rozhraní mezi spustitelnými skripty a výkonnou knihovnou projektu Freeconf tak, aby vyhovoval všem požadavkům uvedeným v předchozí kapitole.

4.5.1 Úpravy balíku

Hlavičkový soubor konfiguračního balíku rozšíříme o volitelný element `script`. Do něj budeme zapisovat název definičního souboru skriptu. Podobně jako element `lists` i element `script` může být v hlavičkovém souboru uveden víckrát.

Jak jsme viděli v příkladu v kapitole 4.2.3, skripty mohou být použity k zápisu konfigurace i do jiného cíle, než do nativního konfiguračního souboru. Z toho vyplývá, že v hlavičkovém souboru již není nutné definovat nativní konfigurační soubor ani XSL transformaci. A jelikož definice těchto dvou souborů byla hlavní náplní elementu `entry-group`, který jsme zavedli v kapitole 3.3.1 pro pojmenované skupiny záznamů, nebudeme vyžadovat ani zadání tohoto elementu. Pokud tedy nativní konfigurační soubor není uveden, předpokládá se, že zápis nastavení konfigurované aplikace provede skript.

Podívejme se nyní na ukázku zjednodušeného hlavičkového souboru konfigurace serveru Apache s odkazem na soubor skriptu:

```
<freeconf-header>
<content>
  <template>apache_template.xml</template>
  <default-values>apache_default.xml</default-values>
  <help>apache_help.xml</help>
  <output>${PACKAGE}/apache_conf.xml</output>
  <lists>apache_lists.xml</lists>
  <script>uname.xml</script>
</content>
</freeconf-header>
```

Jak je patrné z ukázky, definiční soubor skriptu je ve formátu XML. V tomto souboru by měly být uvedeny veškeré informace, které výkonná knihovna potřebuje znát předtím, než začne skript používat. Definiční soubory skriptu se budou hledat v adresáři `scripts` v konfiguračním balíku.

4.5.2 Definice skriptu

Jak jsme již řekli, definiční soubor skriptu budeme zapisovat ve formátu XML. Podívejme se nyní pro snadnější představu na krátkou ukázku, jak by takový soubor zhruba mohl vypadat:

```
<freeconf-script>
  <!-- Cesta ke skriptu: -->
  <path>CESTA</path>
  <!-- Parametry skriptu: -->
  <argument>Parametr 1</argument>
  <argument>Parametr 2</argument>
  <argument>...</argument>

  <!-- Na základě jakých událostí se skript spustí: -->
  <!-- Po načtení balíku -->
  <on-load />
  <!-- Při stisknutí tlačítka obnovy -->
  <on-refresh />

  <!-- Zpřístupnění obsahu vybraných klíčových slov: -->
  <access>
    <entry>Cesta ke klíčovému slovu</entry>
    <!-- ... -->
  </access>
</freeconf-script>
```

Podobně jako ostatní XML soubory konfiguračního balíku uvedeme i definiční soubor skriptu speciálním elementem, který jej odliší od ostatních. Jeho název bude `freeconf-script`.

Následuje nejdůležitější část definice, a tou je cesta k samotnému skriptu a volitelně i jeho parametry. Cestu ke skriptu zadáme do elementu `path`. V cestě lze použít odkazy na proměnné z kapitoly 3.4.1.

Parametr skriptu uvedeme do elementu `argument`. Pokud skript vyžaduje parametrů více, uvedeme element `argument` jednoduše víckrát. Parametry budou skriptu při jeho spuštění předány v pořadí, v jakém jsou uvedeny v definičním souboru.

Spouštěče

Na základě požadavků v kapitole 4.4.2 nadefinujeme události, při kterých se skript spustí. Tyto události budeme nazývat „spouštěče“. Do definičního souboru lze zapsat následující spouštěče:

- **on-load**: Skript se spustí hned po načtení konfiguračního balíku.
- **on-save**: Skript se spustí po uložení nastavení programu.
- **on-entry-change**: Skript se spustí po změně hodnoty jednoho z daných klíčových slov. Klíčová slova definujeme uvnitř elementu `on-entry-change` pomocí elementů `entry`. Jako hodnotu elementu `entry` uvedeme plnou cestu ke klíčovému slovu, které se má sledovat. Pokud je třeba, aby se skript spustil po změně

libovolného klíčového slova, nebudeme po vývojáři balíku vyžadovat vypsání seznamu všech klíčových slov. Místo toho stačí uvést element `all-entries`, který zastoupí všechna klíčová slova v konfiguraci.

- **on-refresh**: Skript se spustí po stisknutí globálního tlačítka „Obnovit“.
- **on-entry-refresh**: Skript se spustí po stisknutí lokálního tlačítka „Obnovit“. Do tohoto elementu je třeba zadat seznam všech klíčových slov, u nichž se má toto tlačítko zobrazit. Formát seznamu klíčových slov je stejný jako v případě obsahu elementu `on-entry-change`.

Pokud v definičním souboru není uveden žádný spouštěč, skript bude sice v knihovně definován, ale nikdy se nespustí. Takový stav bychom tedy mohli považovat za nepřípustný a definiční soubory bez spouštěče ignorovat.

Nastavení přístupu

Z kapitoly 4.4.3 víme, že skript ke své činnosti může vyžadovat znalost hodnot některých klíčových slov. Seznam těchto klíčových slov budeme zapisovat do volitelného elementu `access`. Formát seznamu je stejný jako v předchozích případech, tedy `entry` definuje cestu k jednomu klíčovému slovu a `all-entries` značí všechna klíčová slova.

Pořadí spouštění

Při použití více skriptů v jednom konfiguračním balíku může nastat situace, kdy pro jednu událost (například `on-load`) je potřeba spustit více skriptů. Nepočítáme však s tím, že by aplikace se skripty měla pracovat paralelně, a tedy nebudeme provádět všechny skripty najednou. Proto musíme jasně definovat pořadí, v jakém se skripty budou spouštět.

Pokud není uvedeno jinak, spustí se skripty ve stejném pořadí, v jakém jsou uvedeny v hlavičkovém souboru. Co když ale potřebujeme, aby se skript pro nějaký spouštěč spustil jako první a pro jiný spouštěč jako poslední v pořadí?

Při řešení tohoto problému se inspirujeme u inicializačních skriptů v operačním systému Linux. Tento operační systém může pracovat v jedné z několika úrovní běhu („run level“), přičemž pro každou úroveň lze definovat sadu spouštěcích a ukončovacích skriptů. Pořadí skriptu je zde dáno celým číslem z intervalu $< 0; 99 >$. Je zaručeno, že skripty s nižším pořadovým číslem se spustí dříve než skripty s vyšším číslem.

Proto tedy zavedeme nový atribut spouštěčů: `order`. Jeho hodnotou může být celé číslo z intervalu $< 0; 99 >$. Pokud pak dojde na nějakém spouštěči ke konfliktu, spustí se dříve skript s nižším pořadovým číslem. Definice pořadí pro uvedený příklad může vypadat například takto:

```
<!-- Při načítání spustit skript jako první -->
```

```
  <on-load order="0" />
  <!-- Při ukládání spustit skript jako poslední -->
  <on-save order="99" />
```

Pokud u spouštěče nebude pořadí explicitně určeno, bude parametr `order` automaticky nastaven na výchozí hodnotu 50.

Ukázka definice

Nyní se podíváme, jak takový definiční XML soubor skriptu vypadá v celku. Nejprve si ukážeme soubor popisující skript `uname.sh` pro příklad z kapitoly 4.2.1:

```
<freeconf-script>
  <path>${PACKAGE}/bin/uname.sh</path>
  <argument>/apache-config/ServerName</argument>
  <argument>-n</argument>

  <!-- Spouštěče -->
  <on-load />
  <on-refresh />
</freeconf-script>
```

Kořenový element nám říká, že jde o definiční soubor skriptu. Následuje element `path`, který ukazuje na umístění skriptu. Ten jsme uložili do adresáře `bin` v konfiguračním balíku. Jako parametr skriptu jsme uvedli název klíčového slova, které má nastavit. Zbývající parametry skript předá programu `uname`. Parametr `-n` tedy způsobí vypsání jména počítače.

V seznamu událostí spouštěčů jsme uvedli načtení balíku a stisk globálního tlačítka obnovy.

Abychom si ukázali také použití ostatních elementů, podíváme se ještě na jeden příklad. Tentokrát půjde o návrh definičního souboru skriptu `get_table_list.sh` pro příklad 4.2.2, který získá seznam dostupných tabulek.

```
<freeconf-script>
  <path>${PACKAGE}/bin/get_table_list.sh</path>

  <on-entry-change>
    <entry>/courier-config/MYSQL_DATABASE</entry>
  </on-entry-change>

  <access>
    <entry>/courier-config/MYSQL_USERNAME</entry>
    <entry>/courier-config/MYSQL_PASSWORD</entry>
  </access>
</freeconf-script>
```

Tento skript se spustí pokaždé, když uživatel změní hodnotu klíčového slova `MYSQL_DATABASE`. K dispozici bude mít název databáze, jméno uživatele a přístupové heslo. Tyto hodnoty budou skriptu předány na standardní vstup ve formátu, který si popíšeme v následující kapitole.

4.5.3 Vstup skriptu

Komunikace se spustitelnými skripty bude probíhat pomocí standardního vstupu a výstupu. Je to jeden z nejjednodušších způsobů komunikace se spuštěným procesem a je podporován na většině operačních systémů. Po spuštění skriptu předá výkonná knihovna skrz standardní vstup zprávu obsahující veškeré informace, které skript potřebuje znát ke svému běhu. Následovně skript zpracuje svůj vstup, vykoná potřebné operace (například získání jména počítače), výsledek zapíše na standardní výstup a ukončí se. Výkonná knihovna poté zpracuje výstup skriptu, podle kterého provede změny v konfiguraci.

V kapitole 4.4 jsme si již řekli, že komunikace by měla probíhat pomocí zpráv ve formátu XML. Aby si skript a výkonná knihovna vzájemně rozuměli, musí mít předávané zprávy jasně definovanou strukturu. Jako kořen vstupní zprávy použijeme element `freeconf-script-input`. Podle něj skript pozná, že jde o vstupní zprávu od výkonné knihovny. Pokud by od knihovny obdržel jiný kořenový element, jde zřejmě o chybu a měl by skončit odesláním chybové zprávy, kterou si popíšeme v kapitole 4.5.6.

Aby byl skript informován o události, která způsobila jeho spuštění, zařadíme do vstupu element popisující tuto událost. Název elementu bude stejný, jako název příslušného spouštěče v definičním souboru. Jestliže je tedy skript spuštěn po načtení konfiguračního balíku, vložíme do vstupní zprávy značku `<on-load/>`.

V případě spouštěčů, jejichž parametrem je seznam klíčových slov, tedy `on-entry-change` a `on-entry-refresh`, předáme skriptu i název a hodnotu klíčového slova, u kterých byla tato událost vyvolána.

Zbytek vstupní zprávy bude obsahovat seznam klíčových slov a hodnot, které mají být skriptu zpřístupněny. Klíčová slova budeme zapisovat do elementu `entry`. Ten zde bude mít jeden jediný parametr `path`, jež bude obsahovat název a plnou cestu klíčového slova. Jako hodnota elementu bude uvedena hodnota klíčového slova.

Podívejme se nyní na ukázku vstupní zprávy, jakou by mohl obdržet skript `get_table_list.sh`, jehož definiční soubor jsme viděli v předchozí kapitole.

```
<freeconf-script-input>
  <on-entry-change>
    <entry path="/courier-config/MYSQL_DATABASE">maildb</entry>
  </on-entry-change>

  <entry path="/courier-config/MYSQL_USERNAME">mail</entry>
  <entry path="/courier-config/MYSQL_PASSWORD">password</entry>
</freeconf-script-input>
```

Tato zpráva informuje skript, že byl spuštěn po změně hodnoty klíčového slova `/courier-config/MYSQL_DATABASE`. Následují hodnoty důležitých klíčových slov: jméno uživatele a heslo pro přístup k databázi.

4.5.4 Výstup skriptu

Výstupní zprávu označíme kořenovým elementem `freeconf-script-output`. Pokud knihovna obdrží jakýkoliv jiný kořenový element, nahlásí chybu a výstup skriptu nebude zpracován.

Formát obsahu výstupní zprávy skriptu by nám měl poskytovat prostředky k ovlivnění všech částí konfigurace, jež jsme specifikovali v kapitole 4.4.1.

Nastavení klíčových slov

Vlastnosti klíčových slov budeme nastavovat uvnitř elementu jménem `entry`, který jsme pro práci s klíči v návrhu již několikrát použili. Jako parametr `path` uvedeme plnou cestu ke klíčovému slovu, jehož vlastnosti se mají změnit. Obsahem elementu pak bude popis těchto vlastností.

Výchozí hodnotu klíčového slova nastavíme jako obsah elementu `default`. Hodnotu klíče lze změnit obdobným způsobem v elementu `value`.

Zápis dalších vlastností klíčových slov je stejný jako v šablonovém souboru. Tedy například povinnost klíče nastavíme elementem `mandatory`, aktivitu změníme elementem `active`. Zápis vlastností specifických pro typ klíčového slova, které se v šablonovém souboru zapisují do elementu `properties`, budeme i zde zadávat do elementu `properties`. Změnit nebude možné pouze vlastnosti, které jsou v konfiguračním balíku pevně dané, tedy typ klíčového slova a jeho skupinu.

Následující ukázka obsahuje možný výstup skriptu `uname.sh`, jehož definiční soubor jsme navrhli v kapitole 4.5.2. Ten nastaví výchozí hodnotu klíčového slova `ServerName` na jméno počítače:

```
<freeconf-script-output>
  <entry path="/apache-config/ServerName">
    <default>Pracovní stanice 1</default>
  </entry>
</freeconf-script-output>
```

Ještě poznamenejme, že změny vlastností klíčových slov provedené skriptem by měly být pouze dočasné, platné v rámci běhu konfigurační aplikace. Po skončení konfigurační aplikace se z provedených změn na disk zapíší pouze hodnoty klíčových slov do konfiguračního souboru ve formátu XML. Ostatní změny včetně výchozích hodnot by neměly měnit obsah XML souborů konfiguračního balíku.

Nastavení obsahu seznamů hodnot

K nastavení obsahu seznamů hodnot použijeme stejný způsob zápisu, v jakém jsou seznamy uloženy v konfiguračním balíku. Seznam řetězců tedy uvedeme elementem `string-list` a seznam fuzzy hodnot elementem `fuzzy-list`. Atributem elementů je parametr `name`, jež obsahuje jméno seznamu. Následuje seznam hodnot, z nichž každá je označena elementem `value`. U seznamů fuzzy hodnot je ještě třeba elementu `value` nastavit parametr `grade`, který určuje stupeň příslušnosti hodnoty do fuzzy množiny. Více o seznamech fuzzy hodnot naleznete v práci [5]. Jak to celé vypadá vidíme v následujícím příkladu:

```
<freeconf-script-output>
  <string-list name="mysql_table_list">
    <value>table_1</value>
    <value>table_2</value>
    <value>table_3</value>
  </string-list>

  <entry path="/courier-config/MYSQL_USER_TABLE">
    <properties>
      <data>mysql_table_list</data>
    </properties>
  </entry>
</freeconf-script-output>
```

Zde jsme definovali seznam hodnot `mysql_table_list`, který obsahuje dostupné tabulky v databázi. Následně jsme tento seznam použili při změně vlastností klíčového slova `MYSQL_USER_TABLE`.

Z úvodní kapitoly 1.4.8 víme, že k seznamu hodnot patří také soubory s vícejazyčnými popiskami a nápovědou k jednotlivým hodnotám. Zde přitom o popiskách ještě nepadlo ani slovo. Je to proto, že vícejazyčným seznamem popisků bychom návrh rozhraní pro skripty jen zbytečně komplikovali. Seznamy hodnot generované skriptem budou zejména seznamy nějakých systémových hodnot, ke kterým skript nápovědu ani znát nebude.

Pokud by přeci jen bylo v budoucnu třeba nastavovat popisky seznamů, dalo by se uvažovat o úpravě zápisu seznamu hodnot do podoby, v jaké jsou uloženy v souboru s nápovědou k seznamům hodnot. Výkonná knihovna by pak mohla předpokládat, že popisky jsou uvedeny v aktuálním jazyce. Aktuální jazyk by mohl skript zjistit například ze systémové proměnné `LANG`.

4.5.5 Vícenásobné sekce

Vícenásobné sekce, které jsme zavedli v kapitole 3.1, bude třeba zohlednit i při návrhu rozhraní pro spustitelné skripty. Zamysleme se, co by se při současném stavu návrhu stalo, kdybychom se ze skriptu pokusili upravit hodnotu klíče, jež leží ve

vícenásobné sekci. Mějme hypotetický konfigurační balík `test`, v něm jednu vícenásobnou sekci `V` a v ní jeden klíč typu číslo jménem `K`. Nechť má sekce `V` dvě instance, hodnota klíče `K` v první instanci je 0 a ve druhé instanci 1. Představme si, že skript `S.sh` má změnit hodnotu klíče `K` ve druhé instanci sekce na hodnotu 2. To zřejmě není možné, neboť zatím nemáme žádný způsob, jak jednoznačně identifikovat instanci vícenásobné sekce v konfiguraci.

Potřebovali bychom tedy upravit formát zápisu cest v konfiguračním stromě tak, aby nám umožňoval odkazovat se na instance vícenásobných sekcí. Toho dosáhneme jednoduše: pomocí indexování. Každou instanci vícenásobné sekce lze totiž jednoznačně identifikovat kladným celým číslem podle toho, v jakém pořadí je uložena v konfiguraci. První přidané sekci přiřadíme číslo 1, druhé přiřadíme číslo 2, atd. Index instance sekce do cesty zapíšeme mezi závorky [a] za název sekce následujícím způsobem:

```
/test-config/V[2]/K
```

Takto vyjádřená cesta už jednoznačně míří na požadovaný klíč `K` z druhé instance sekce `V`.

Změna hodnoty klíče

Změnu klíče v instanci vícenásobné sekce nám tedy umožní indexování v cestě ke klíči. Je jasné, že výkonná knihovna bude muset kontrolovat, zda index označuje existující instanci. Pokud by hodnota indexu v cestě byla větší, než je aktuální počet instancí sekce, dojde k chybě.

Skriptu bychom tedy měli nějakým způsobem předat informaci o tom, jaké instance vícenásobných sekcí existují, aby je mohl správně adresovat. Toho docílíme tak, že i vstupní zprávy odeslané výkonnou knihovnou budou obsahovat adresy vícenásobných sekcí v rozšířeném formátu. Zde rozlišíme dva případy:

1. V seznamu klíčových slov předaných v bloku spouštěče (tj. u záznamů v elementech `on-entry-change` a `on-entry-refresh`) předáme pouze ty instance klíčových slov, ve kterých došlo ke změně, nebo u kterých bylo stisknuto tlačítko obnovy. Za změnu hodnoty klíče budeme považovat i vytvoření klíče v nové instanci sekce.

2. V seznamu klíčových slov, které byly v definičním XML souboru uvedeny v bloku `access`, předáme hodnoty klíčových slov ze všech instancí vícenásobných sekcí.

Podívejme se tedy na ukázku k předchozímu příkladu. Skript `S.sh` byl spuštěn na základě změny hodnoty klíčového slova `K` v druhé instanci sekce `V`. Na svůj vstup tedy dostane zprávu:

```
<freeconf-script-input>
  <on-entry-change>
    <entry path="/test-config/V[2]/K">1</entry>
  </on-entry-change>
</freeconf-script-input>
```

Pokud by skript `S.sh` byl spuštěn po načtení konfiguračního balíku (na základě spouštěče `on-load`) a měl by zpřístupněny hodnoty klíče `K`, obdrží následující zprávu:

```
<freeconf-script-input>
  <entry path="/test-config/V[1]/K">0</entry>
  <entry path="/test-config/V[2]/K">1</entry>
</freeconf-script-input>
```

Změnu hodnoty klíče `K` ve druhé instanci sekce `V` tedy skript provede odesláním zprávy:

```
<freeconf-script-output>
  <entry path="/test-config/V[2]/K">
    <value>2</value>
  </entry>
</freeconf-script-output>
```

Ještě poznamenejme, že způsob zápisu cest v definičním souboru skriptu se nemění. Zde totiž ještě nemáme informace o přesném počtu instancí sekce a nemůžeme je tedy pomocí indexů jednoznačně adresovat. Cesty uvedené v definičním souboru budou označovat vždy všechny instance vícenásobné sekce a to i ty, které ještě nebyly vytvořeny. Definiční soubor k poslední uvedené vstupní zprávě tedy může vypadat například takto:

```
<freeconf-script>
  <path>${PACKAGE}/bin/S.sh</path>

  <on-load/>

  <access>
    <entry>/test-config/V/K</entry>
  </access>
</freeconf-script>
```

Při implementaci podpory skriptování do výkonné knihovny by bylo nejlepší, aby se cesty v definičním souboru vyhodnocovaly vzhledem k šablonovému stromu, neboť zde se nastavuje obecné chování klíčových slov ke skriptu ve všech instancích vícenásobných sekcí. Cesty ve vstupních a výstupních zprávách skriptu by se pak měly vyhodnocovat vzhledem ke konfiguračnímu stromu, protože ty již označují konkrétní hodnoty klíčových slov.

Vytvoření instance sekce

Další operací, kterou potřebujeme pro práci s vícenásobnými sekcemi, je vytvoření nové instance sekce a její naplnění konfiguračními hodnotami. Formát výstupní zprávy skriptu tedy rozšíříme o nový element `new-entry`, jež vytvoří novou instanci vícenásobné sekce. Cestu k vícenásobné sekci uvedeme, podobně jako u záznamů klíčových slov, do parametru `path`. Obsah elementu bude tvořit seznam klíčových slov uvnitř nové instance sekce a nastavení jejich vlastností. Ty budeme zapisovat stejně jako u klasických klíčových slov.

Pokud bychom tedy v našem příkladu chtěli vložit novou instanci sekce `V` s hodnotou klíčového slova `K` nastavenou na 3, bude výstupní zpráva skriptu vypadat takto:

```
<freeconf-script-output>
  <new-entry path="/test-config/V">
    <entry path="K">
      <value>3</value>
    </entry>
  </new-entry>
</freeconf-script-output>
```

Povšimněme si, že u vícenásobné sekce `V` jsme neuvedli index instance. Je to proto, že vkládáme novou instanci a její index je tedy zatím neznámý. A dále u klíčového slova `K` neuvádíme plnou cestu, ale cestu relativní k vícenásobné sekci.

Odebrání instance sekce

Vzhledem k tomu, že jsme již skriptům umožnili vytvářet nové instance vícenásobných sekcí, měli bychom jim také umožnit instance sekcí odebírat. Formát výstupní zprávy tedy rozšíříme ještě o jeden nový element: `remove-entry`, který odebere jednu instanci vícenásobné sekce. Jeho jediný parametr `path` bude obsahovat cestu k instanci. Cesta musí být uvedena celá včetně indexu, který jednoznačně určí odebíranou instanci.

Následující výstupní zpráva našeho demonstračního skriptu způsobí odebrání první instance vícenásobné sekce `V`:

```
<freeconf-script-output>
  <remove-entry path="/test-config/V[1]" />
</freeconf-script-output>
```

4.5.6 Chyby

Běh spustitelného skriptu samozřejmě nemusí být vždy úspěšný. Prakticky kdykoliv během vykonávání skriptu může nastat chyba nebo nějaký neočekávaný stav, na

který skript nemusí být připraven, a v důsledku toho není schopen pokračovat. V takovém případě by měl informovat výkonnou knihovnu o svém neúspěchu a skončit.

To, jakým způsobem skript skončil, se dozvíme z jeho návratového kódu. Návratový kód je celé číslo (obvykle od 0 do 255), které program předá při svém ukončení volajícímu programu. Používá se jako standardní prostředek pro signalizaci chyb. Pokud je návratový kód roven 0, volající strana předpokládá, že program skončil úspěšně. Jakákoliv jiná hodnota obvykle značí chybu. I my se v návrhu budeme držet této zažité konvence. Více o návratových kódech je možné nalézt v knize [14].

Pokud je tedy návratový kód skriptu roven 0, skončil úspěšně a výkonná knihovna může zpracovat XML zprávu z jeho standardního výstupu. V případě chyby bude návratový kód skriptu větší než 0. Díky tomu se výkonná knihovna dozví o problému a výstup skriptu nezpracuje.

Přesný důvod chyby a další detailní informace o problému by skripty měly zapisovat na svůj standardní chybový výstup. Výkonná knihovna pak může tyto informace načíst a zobrazit uživateli.

4.6 Příklad skriptu

Nyní se podíváme na to, jak by takový spustitelný skript mohl vypadat. Představíme zde jednoduchý skript `uname.sh` k příkladu 4.2.1. Jeho definiční XML soubor jsme viděli v kapitole 4.5.2.

Skript je psaný v jazyce unixového shellu `Bash`. Z definičního souboru skriptu víme, že má přebírat dva parametry. Jméno klíčového slova, jehož výchozí hodnotu budeme měnit, a parametry programu `uname`. Cílem je danému klíčovému slovu nastavit výchozí hodnotu na text, který vrátí program `uname`. Kód skriptu je následující:

```
#!/bin/bash

# Klíčové slovo je první a povinný parametr
KLICOVE_SLOVO="$1"
if [[ -z "$KLICOVE_SLOVO" ]]
then
  # Odeslání chyby a konec skriptu
  echo "Chybí klíčové slovo!" >&2
  exit 1
fi
# Odebereme první parametr
shift

# Načteme vstupní zprávu
# V tomto skriptu nás její obsah nezajímá, proto ji můžeme zahodit
cat >/dev/null
```

```
# Zavoláme program uname a předáme mu zbývající parametry
HODNOTA="$(uname "$@" )"
if [[ $? -ne 0 ]]
then
  # Odeslání chyby a konec skriptu
  echo "Program uname skončil s chybou!" >&2
  exit 2
fi

# Odešleme výstupní zprávu
cat <<KONEC
<freeconf-script-output>
  <entry path="$KLICOVE_SLOVO">
    <default>$HODNOTA</default>
  </entry>
</freeconf-script-output>
KONEC

# Úspěšný konec skriptu
exit 0
```

Ve skriptu nejprve získáme první parametr a zkontrolujeme, zda byl vůbec nastaven. Pokud ne, odešleme informaci o chybě na standardní chybový výstup a skript ukončíme chybovým návratovým kódem. Dále se provádí načtení vstupní zprávy skriptu. Vzhledem k tomu, že skript ke svému běhu žádné další informace, kromě těch předaných v parametrech, nepotřebuje, zprávu můžeme zahodit. Následuje zavolání programu `uname` a získání jeho výstupu, který uložíme do proměnné `HODNOTA`. Poté zkontrolujeme, zda program skončil s návratovým kódem 0. Pokud ne, vypíšeme informaci o chybě a skončíme.

Závěrečnou část skriptu tvoří naformátování a odeslání výstupní XML zprávy. Do ní vložíme získanou hodnotu. Zde jsme v rámci zjednodušení vynechali ošetření speciálních znaků, které by se mohly v hodnotě vyskytnout. Do zprávy ve formátu XML totiž nelze zapsat všechny znaky tak jak jsou. Některé mohou mít speciální význam. Takovými znakem je například < a je nutné jej zakódovat do podoby, v jaké nebude pro XML parser představovat problém. Náhradu znaku < v XML tvoří sekvence znaků `<`. V ukázkovém skriptu předpokládáme, že k výskytu takových znaků ve výstupu programu `uname` nedojde.

A nakonec skript ukončíme s návratovým kódem 0.

4.7 Poznámky k implementaci

4.7.1 Spuštění skriptu z jazyka Python

Jednou z nejdůležitější akcí, kterou budeme muset při implementaci podpory spustitelných skriptů udělat, je samotné spuštění skriptu z výkonné knihovny, odeslání vstupní zprávy a načtení výstupní zprávy. V programovacím jazyce Python toho dosáhneme snadno za pomoci třídy `Popen` z modulu `subprocess`. Konstruktor třídy `Popen` přebírá jméno programu a seznam jeho argumentů jako pole, následuje sada nejrůznějších parametrů. Zajímat nás budou zejména parametry `stdin`, `stdout` a `stderr`, které nastavují způsob předání standardního vstupu, výstupu a chybového výstupu. My tyto parametry nastavíme na hodnotu `subprocess.PIPE`, která způsobí, že pro komunikaci se spuštěným programem budou použity roury[3]. Podrobný popis třídy `Popen` a všech parametrů konstruktoru nalezneme v dokumentaci jazyka Python [11].

Po vytvoření objektu třídy `Popen` zavoláme metodu `communicate` a předáme jí vstupní zprávu. Výstup metody tvoří seznam, kde první položkou je řetězec obsahující standardní výstup programu, druhou položku tvoří řetězec se standardním chybovým výstupem. Návratový kód programu získáme z atributu `returncode`.

Podívejme se nyní, jak lze pomocí třídy `Popen` na několika málo řádcích zdrojového kódu spustit skript `uname.sh` z předchozí kapitoly:

```
# Načtení modulu subprocess
from subprocess import Popen, PIPE

# Vstupní zpráva
input_msg = "<freeconf-script-input />"

# Vytvoření objektu pro volání skriptu
p = Popen(
  ["uname.sh", '/apache-config/ServerName', '-n'],
  stdin=PIPE, stdout=PIPE, stderr=PIPE
)

# Spuštění skriptu, předání vstupní zprávy a získání výstupu
(output_msg, error_msg) = p.communicate(input_msg)

# Vypsání výsledku
print "Návratový kód:", p.returncode
print "Výstup skriptu:\n", output_msg
# Pokud došlo k chybě, vypíšeme i chybový výstup skriptu
if p.returncode > 0:
  print "Chybový výstup skriptu:\n", error_msg
```

[3] Roury jsou standardním prostředkem pro meziprocesovou komunikaci v operačních systémech typu Unix. Více o této problematice je možné nalézt například v knize [14]

4.7.2 Změny ve výkonné knihovně

Vzhledem k rozsahu této práce se nám již bohužel nepodařilo skriptování do výkonné knihovny projektu Freeconf implementovat. Proto v této kapitole uvedeme orientační seznam změn, které bude v knihovně potřeba vykonat, aby byla podpora spustitelných skriptů plně funkční.

- V souboru `script.py` je základ třídy `FcScript`, jež má reprezentovat objekt skriptu ve výkonné knihovně. Tuto třídu bude potřeba rozšířit o všechny atributy skriptu, které mohou být nastaveny v XML definičním souboru, a o akce, které mohou být se skriptem vykonány. Zejména pak spuštění skriptu, ošetření chyb, naformátování vstupní zprávy a načtení výstupní zprávy.

- Bylo by vhodné definovat třídu reprezentující vstupní zprávu skriptu, kterou jsme navrhli v kapitole 4.5.3. Tato třída by měla obsahovat identifikátor spouštěče, jeho parametry a seznam aktuálních hodnot klíčových slov. Z objektu třídy pak bude vyrobena vstupní zpráva skriptu ve formátu XML.

- Podobně bylo by vhodné definovat i třídu, jež by reprezentovala výstupní zprávu skriptu navrženou v kapitole 4.5.4. Objekty této třídy by vznikaly načtením výstupní XML zprávy skriptu a měly by tedy obsahovat všechny informace uvedené ve zprávách.

- Dále bude potřeba kód, který aplikuje všechny změny popsané v objektu výstupní zprávy skriptu. Měl by tedy být schopen měnit vlastnosti klíčových slov v šablonovém stromě, měnit hodnoty klíčových slov v konfiguračním stromě a také vytvářet a rušit instance vícenásobných sekcí. Klientská aplikace by měla být informována o všech těchto změnách mechanismem zpráv, který se v knihovně používá při aplikaci závislostí. Více o tomto mechanismu je možné nalézt v práci [5].

- Načtení definičního XML souboru skriptu jsme již částečně implementovali v souboru `script_file.py`. Dochází zde k vytvoření objektu třídy `FcScript` a jeho naplnění hodnotami z definičního souboru. V současnosti podporuje načtení jména a argumentů skriptu a spouštěč `on-load`. Zpracování tedy bude třeba rozšířit o všechny zbývající vlastnosti z kapitoly 4.5.2.

- Rozšířit načítání hlavičkového souboru balíku tak, aby podporovalo nový element `script` z kapitoly 4.5.1.

- Skripty bude potřeba navázat na objekty, ve kterých mohou vzniknout události, při nichž se skripty mají spouštět. Tedy například objekt konfiguračního balíku bude muset obsahovat seznam skriptů, které se mají spustit po načtení balíku a další seznam skriptů, které se budou spouštět při ukládání balíku.

- Rozšířit GUI klientské aplikace o tlačítka „Obnovit“.

- Upravit GUI klientské aplikace tak, aby bylo schopné zpracovat změnu klíčového slova typu řetězec z jednoduchého textového pole na výběr ze seznamu

řetězců a naopak. Tato situace může nastat u výběru jména tabulky z příkladu v kapitole 4.2.2.

- Podpora spustitelných skriptů může být aktivní jen, pokud je výkonná knihovna použita z klientské aplikace. Při použití výkonné knihovny z grafického návrháře balíku je spouštění skriptů nežádoucí, neboť by zasahovalo do struktury konfigurace vytvářené vývojářem balíku.

Kapitola 5

Porovnání konfigurace webového serveru Apache

Poté, co se nám podařilo popsat základ konfigurace webového serveru Apache, jsme se rozhodli porovnat výsledek s některými již existujícími nástroji na konfiguraci serveru Apache z GUI. Vybrali jsme dvě konfigurační aplikace, se kterými se uživatel při konfiguraci Apache pravděpodobně setká nejčastěji. První aplikací je software na dálkovou správu serverů *Webmin*. Druhou testovanou aplikací je konfigurační nástroj *system-config-httpd* z distribuce operačního systému Linux Fedora.

V každé konfigurační aplikaci jsme se pokusili nastavit webový server tak, aby poskytoval přístup k jednoduché testovací stránce na místní adrese počítače `localhost`. Port, na kterém měl server čekat požadavky jsme nastavili na 7000. Vždy jsme vycházeli z výchozího nastavení.

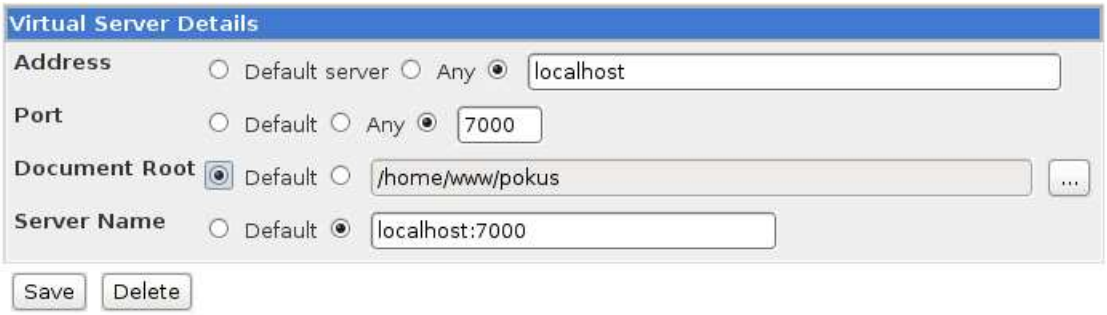
5.1 Webmin

Konfigurační nástroj jsme si již v hrubých rysech představili v kapitole 1.1.1. Zde se zaměříme pouze na nastavení webového serveru Apache. Pro porovnání s naší konfigurační aplikací jsme použili poslední dostupnou verzi aplikace Webmin 1.570 staženou z webových stránek na adrese `http://webmin.com`.

5.1.1 Výhody

Vlastní konfigurace je poměrně jednoduchá. V záložce „Apache Webserver“ jsou všechna nastavení serveru uspořádána do několika tematických oddílů. Nastavení se načítají přímo z konfiguračního souboru serveru. To může být užitečné v případě, že chceme změnit nastavení již existujícího serveru, který byl doposud konfigurován ručně nebo jiným konfiguračním systémem. Zaujala nás také možnost přímo zobrazit aktuální podobu konfiguračního souboru, případně jej rovnou editovat.

Dalším zajímavým prvkem v konfiguraci je přepínač výchozích hodnot, který se



Obrázek 5.1: Část nastavení virtuálního hostitele v aplikaci Webmin

zobrazuje u každé volby. Jak tento přepínač vypadá vidíme na obrázku 5.1. Pokud je zaškrtnutý, použije se pro volbu výchozí hodnota. To je užitečné například, když se nám nové nastavení neosvědčilo a chceme se vrátit k přednastaveným hodnotám serveru. Použitou výchozí hodnotu pak ale bohužel nevidíme. Pokud ji chceme zjistit je nutné se uchýlit k dokumentaci, kde jsou výchozí hodnoty voleb uvedeny.

Na obrázku 5.1 vidíme u volby *Document Root* tlačítko s textem „...“. Po jeho stisknutí se nám zobrazí adresářová struktura, ze které můžeme snadno vybrat adresář, jež má webový server použít. To je také jistě užitečná funkcionalita která uživatelům usnadňuje konfiguraci.

Po dokončení editace je možné nové nastavení přímo z webového rozhraní aplikovat na server Apache, případně jej zastavit a nebo spustit.

K dispozici je také záznam změn, kam se zapisují všechny změny provedené v konfiguraci. V případě náhlých problémů s chováním serveru po změně konfigurace je tak možné dohledat, které změny to mohly způsobit.

5.1.2 Nevýhody

V průběhu editace voleb nám zde poněkud chyběla nápověda k jednotlivým nastavením, která se zde nezobrazuje a je třeba ji dohledávat v dokumentaci. To je sice malý, leč nepříjemný nedostatek, jež zejména nezkušeným administrátorům komplikuje práci. Pokud se již administrátor ve volbách orientuje, není chybějící nápověda na závadu.

Na závažnější problém jsme narazili při zadávání hodnoty portu, na kterém má webový server očekávat požadavky. Všimli jsme si, že lze položka není dostatečně ošetřena na nevalidní vstup. Aplikace sice vynucuje zadání číselné hodnoty a při zadání textového řetězce logicky nahlásí chybu. Nicméně při zadání příliš velké hodnoty portu (například 70000[1]) lze konfiguraci bez problémů uložit. Komplikace nastanou až v okamžiku kdy se pokusíme nastavený server spustit. To se nepodaří a server skončí s chybou *Invalid address or port*.

[1] Maximální hodnota portu je 65535, neboť síťový protokol TCP k uložení čísla portu používá 16 bitů.

Tyto chybějící kontroly jsou zřejmě dány cílovou skupinou aplikace Webmin, kterou tvoří zejména administrátoři serverů a takový problém snadno odhalí i bez kontrolních mechanismů konfigurační aplikace.

5.2 Konfigurační nástroj distribuce Fedora

Další aplikací, kterou jsme k porovnání použili, byl nástroj system-config-httpd z distribuce operačního systému Linux Fedora verze 16. Nástroj je integrovaný do grafického prostředí GNOME a umožňuje přehlednou editaci nejčastějších voleb webového serveru Apache.

5.2.1 Výhody

Výhody tohoto nástroje spatřujeme zejména v přehlednosti konfigurace. Například nastavení vícenásobných sekcí jsou zde logicky zorganizována a rozdělena do záložek, jak je vidět na obrázku 5.2. Klientská aplikace projektu Freeconf tato nastavení zobrazí do jednoho okna, neboť záložky jsou zatím podporovány pouze v hlavním okně, nikoli však v oknech vícenásobných sekcí.

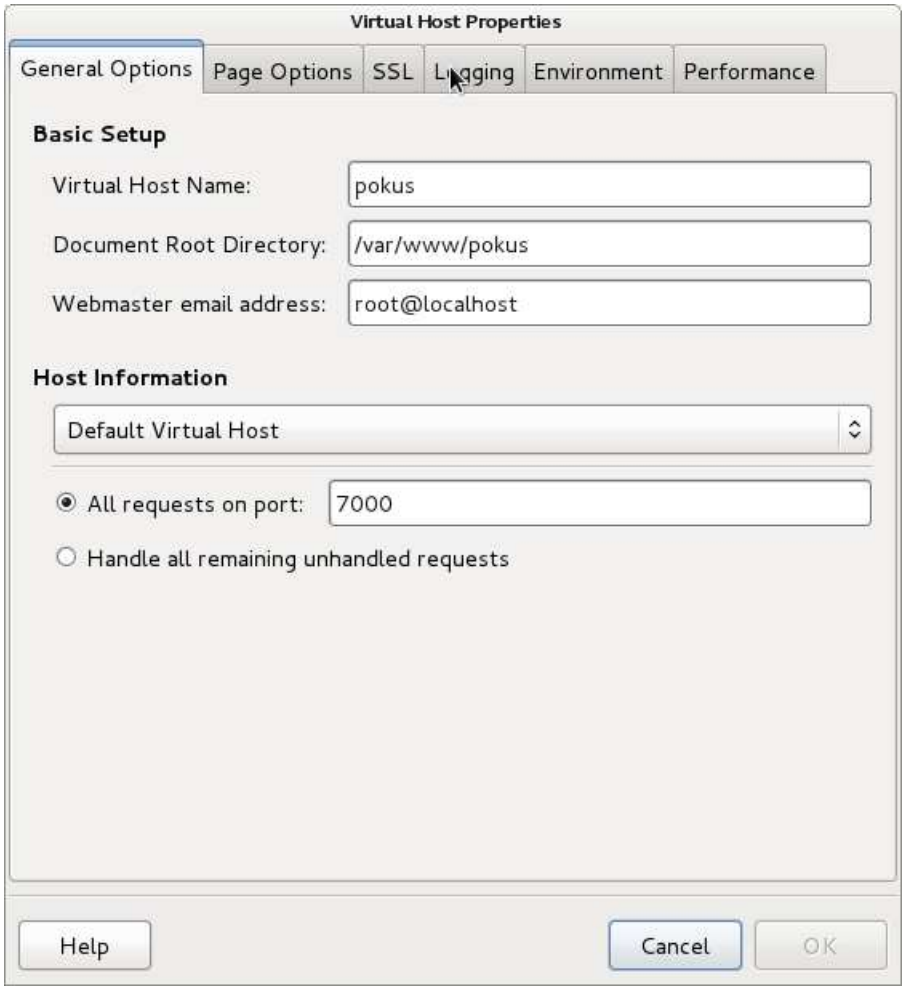
Na rozdíl od aplikace Webmin má již tento nástroj integrovanou nápovědu. Ta se zobrazí po kliknutí na tlačítko „Help“ v novém okně webového prohlížeče. Každé okno je popsáno svojí webovou stránkou. Co zde ale chybí je přímá vazba nápovědy ke konfiguračním volbám. Při hledání informací k jedné konkrétní volbě tak uživatel musí procházet poměrně dlouhý text a hledat v něm relevantní informace.

5.2.2 Nevýhody

I zde jsme narazili na problémy s ošetřením vstupních hodnot. Zadání příliš vysokého čísla portu v konfiguraci virtuálního hostitele není nijak kontrolováno a tak uložení konfigurace opět způsobí chybu při spouštění webového serveru.

5.3 Klientská aplikace projektu Freeconf

Konfigurační balík serveru Apache v současné době neobsahuje zdaleka všechny volby serveru, neboť jsme do balíku zařadili jen nezbytné volby k nastavení serveru a k ověření nových funkcí projektu Freeconf. V tomto ohledu je tedy naše konfigurační aplikace jednoznačně v nevýhodě. Při porovnávání jsme se proto zaměřili zejména na obecné vlastnosti konfigurace, které nelze změnit pouhou úpravou konfiguračního balíku.



Obrázek 5.2: Nastavení virtuálního hostitele v aplikaci system-config-httpd

5.3.1 Výhody

Zjistili jsme, že oproti uvedeným aplikacím v projektu Freeconf lépe zvládáme nápovědu. Díky tomu, že pro každé klíčové slovo máme text s jeho popisem, jsme schopni nápovědu zobrazit ihned u konfigurované volby a ušetřit tak čas, který by jinak uživatel musel vynaložit na čtení dokumentace. I ošetření vstupů od uživatele, zejména pak rozsahů číselných hodnot, zvládá aplikace projektu Freeconf lépe. V šablonovém souboru konfiguračního balíku máme široké možnosti omezení přípustné množiny vstupních hodnot.

V žádné z testovaných aplikací jsme také nenarazili na obdobu zpracování nekonzistencí. Tím klientská aplikace jasně a přehledně signalizuje, které hodnoty je třeba vyplnit, aby byla konfigurace funkční.

5.3.2 Co zatím chybí

V průběhu testování konfiguračních aplikací serveru Apache jsme narazili na mnoho zajímavých prvků, které projekt Freeconf zatím nepodporuje. Některé z nich by ale mohly být užitečné a jejich doplnění do projektu Freeconf by uživatelům usnadnilo konfiguraci. Navrhované změny jsou seřazeny sestupně podle odhadované náročnosti implementace:

1. Zaznamenávání změn, podobně jako to umí aplikace Webmin, by mohlo pomoci při hledání problémů v konfiguraci. Stačilo by zaznamenávat změny hodnot klíčových slov, zejména pak čas události, cestu ke klíčovému slovu, jeho původní hodnotu a novou hodnotu.

2. Další inspirací z aplikace Webmin je způsob práce s výchozími hodnotami. Klíčová slova, která používají výchozí hodnotu jsou zvýrazněna pomocí přepínače. Uživatel tak na první pohled rozezná klíčová slova, jejichž hodnota byla změněna. V průběhu konfigurace se také lze kdykoliv vrátit k výchozí hodnotě klíče. Zvýraznění výchozích hodnot bychom mohli dosáhnout například vhodným podbarvením konfiguračních polí v klientské aplikaci. Návrat k výchozí hodnotě by mohla klientská aplikace realizovat například pomocí položky v menu, které by se otevřelo po stisknutí pravého tlačítka myši nad daným klíčovým slovem.

3. Obě testované aplikace pracovaly přímo s konfiguračními soubory serveru Apache. Při spuštění z nich načítaly hodnoty a po ukončení konfigurace hodnoty do souborů opět zapsaly. První krok v projektu Freeconf úplně chybí, neboť konfigurační hodnoty se primárně ukládají do konfiguračního XML souboru a z něj se následně generuje nativní konfigurační soubor. Nicméně s podporou skriptování, kterou jsme navrhli v kapitole 4, by bylo možné tento nedostatek vyřešit. Stačilo by pro webový server Apache vytvořit skript, který by po spuštění klientské aplikace načetl nastavení serveru Apache přímo z jeho konfiguračních souborů a podle něj nastavil příslušná klíčová slova.

4. V nástroji system-config-httpd nás zaujala přehledná organizace klíčových slov ve vícenásobné sekci do záložek. Klientská aplikace projektu Freeconf zatím klíčová slova ve vícenásobné sekci zobrazuje pod sebe do jednoho okna. Abychom byli schopni klíčová slova v instanci sekce lépe organizovat, bude třeba rozšířit šablonový soubor GUI o popis zobrazení vícenásobných sekcí a definici záložek v sekcích. Samozřejmě bude také nutné upravit třídy objektů grafického stromu a klientskou aplikaci, aby toto rozšíření podporovala.

5. Projekt Freeconf zatím neumožňuje vzdálený přístup ke konfiguraci počítače. To by mohla řešit klientská aplikace, ke které by uživatel přistupoval přes webový prohlížeč tak, jak je navrhováno v kapitole 1.5.2.

Závěr

Cílem práce bylo navrhnout a implementovat podporu pro konfiguraci programů s rozšiřujícími moduly a s více než jedním konfiguračním souborem. Dále jsme měli otestovat nové vlastnosti projektu na konfiguraci webového serveru Apache. Posledním úkolem naší práce bylo navržení rozšíření systému tak, aby podporoval dynamické konfigurování pomocí spustitelných skriptů.

V první kapitole jsme si vysvětlili základní princip vytváření konfiguračních souborů v projektu Freeconf a představili jsme si jednotlivé soubory konfiguračního balíku.

Ve druhé kapitole jsme si představili webový server Apache a rozebrali jeho konfiguraci. přičemž jsme se snažili nalézt problémy, které by se při konfiguraci pomocí projektu Freeconf mohly vyskytnout. Kromě očekávaných nedostatků v podobě chybějící podpory rozšiřujících modulů a více konfiguračních souborů jsme narazili na vícenásobné sekce a vnořené záznamy, se kterými výkonná knihovna projektu Freeconf doposud nedovedla pracovat.

Všechny problémy nalezené ve druhé kapitole se nám podařilo vyřešit a potřebná rozšíření implementovat do výkonné knihovny. Přidali jsme podporu pro vícenásobné sekce a rozšiřující moduly. Rozdělení nastavení do více konfiguračních souborů jsme vyřešili rozdělením klíčových slov do předem definovaných skupin. Každá skupina klíčových slov pak může být zapsána do jiného výstupního souboru. Problém s vnořenými záznamy jsme vyřešili zavedením odkazů v šablonovém souboru.

Dále jsme se věnovali návrhu rozšíření konfigurace pomocí spustitelných skriptů. Nejprve jsme stanovili reálné situace, kdy by podpora spustitelných skriptů mohla být užitečná. Při návrhu jsme se snažili, aby výsledné rozšíření bylo schopné dané situace vyřešit. Také jsme ale usilovali o maximální obecnost celého konceptu spustitelných skriptů tak, aby případné rozšíření o další funkce nepředstavovalo velký problém. Bohužel nám už nezbyl čas na implementaci navrženého rozšíření. Proto jsme alespoň sestavili seznam změn, které bude potřeba ve zdrojových kódech projektu vykonat.

V poslední kapitole jsme se věnovali srovnání naší konfigurační aplikace se dvěma již existujícími aplikacemi na správu nastavení webového serveru Apache pomocí grafického uživatelského rozhraní. Zjistili jsme, že konfigurační aplikace projektu Freeconf lépe zvládá ošetření vstupních hodnot a poskytuje dostupnější nápovědu pro uživatele. Narazili jsme také na několik nedostatků, které by bylo vhodné v konfigurační aplikaci vyřešit.

Všechny cíle vytyčené na začátku práce se nám podařilo splnit. Pokryli jsme problém

konfigurace webového serveru Apache a projekt Freeconf jsme rozšířili o několik nových funkcí. Tím jsme rozšířili množinu aplikací, jejichž konfiguraci je projekt schopen popsat a přiblížili jsme jej tak reálnému nasazení. Dále jsme navrhli podporu dynamického rozšíření konfigurace pomocí spustitelných skriptů. Věříme, že implementace návrhu povede ke snazší a uživatelsky přívětivější konfiguraci. Klientská aplikace bude moci dynamicky reagovat na akce uživatele a na aktuální nastavení operačního systému. Podpora spustitelných skriptů by nám také měla umožnit propojení projektu Freeconf s dalšími konfiguračními systémy.

Ještě poznamenejme, že pravidelně aktualizovanou dokumentaci projektu Freeconf lze nalézt na internetové adrese:

`http://geraldine.fjfi.cvut.cz/ oberhuber/doku-wiki-freeconf/doku.php`

Seznam použitých zdrojů

[1] Apache Software Foundation: *Apache HTTP Server Version 2.2 Documentation* [online]. 2011. Dostupné na `http://httpd.apache.org/docs/2.2/`

[2] Aulds, Charles: *Linux - administrace serveru Apache*. 1. vyd. Praha: Grada Publishing, 2003. 535s. ISBN 80-247-0640-7.

[3] Double Precision, Inc.: *Dokumentace IMAP serveru Courier* [online]. 2008. Dostupné na `http://www.courier-mta.org/imap/documentation.html`

[4] Ehrenberger, J.: *Systém pro konfiguraci operačních systémů typu Open Source*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta jaderná a fyzikálně inženýrská, Praha, 2009.

[5] Ehrenberger, J.: *Systém pro konfiguraci operačních systémů*. Výzkumný úkol, České vysoké učení technické v Praze, Fakulta jaderná a fyzikálně inženýrská, Praha, 2010.

[6] Fabian, D.: *Systém pro konfiguraci operačních systémů typu Open Source*. Výzkumný úkol, České vysoké učení technické v Praze, Fakulta jaderná a fyzikálně inženýrská, Praha, 2009.

[7] Fabian, D.: *Systém pro zjednodušené generování konfigurací*. Diplomová práce, České vysoké učení technické v Praze, Fakulta jaderná a fyzikálně inženýrská, Praha, 2011.

[8] NCSA: *NCSA HTTPd* [online]. 1998. Dostupné na `http://web.archive.org/web/20071029000128/http://hoohoo.ncsa.uiuc.edu/`

[9] Netcraft: *Průzkum rozšíření webových serverů* [online]. 2011. Dostupné na `http://news.netcraft.com/archives/2011/05/02/may-2011-web-server-survey.html`

[10] Nokia Corp.: *Dokumentace Qt* [online]. 2011. Dostupné na `http://doc.qt.nokia.com/latest/`

[11] Python Software Foundation *Python Documentation* [online]. 2009. Dostupné na `http://www.python.org/doc/`

[12] Richard J. Simon, Michael Gouker, Brian C. Barnes. *Win32 API - průvodce vývojáře*. 1. vyd. Brno: UNIS Publishing, 1997. 1417 s. ISBN 80-86097-06-4.

[13] Richard Stones, Neil Matthew. *Linux programujeme profesionálně*. 1. vyd. Praha: Computer Press, 2001. 1079 s. ISBN 80-7226-532-6.

[14] Richard Stones, Neil Matthew. *Linux začínáme programovat*. 1. vyd. Praha: Computer Press, 2000. 897 s. ISBN 80-7226-307-2.

[15] W3C: *CERN httpd* [online]. 1999. Dostupné na `http://www.w3.org/Daemon/`

[16] W3C: *Extensible Markup Language (XML) 1.0* [online]. 2008. Dostupné na `http://www.w3.org/TR/2008/REC-xml-20081126/`

[17] W3C: *XSL Transformations (XSLT) Version 2.0* [online]. 2007. Dostupné na `http://www.w3.org/TR/xslt20/`

Příloha A

Získání zdrojového kódu projektu

Veškeré zdrojové kódy projektu Freeconf jsou umístěny v SVN repozitáři na jednom ze serverů Fakulty jaderné a fyzikálně inženýrské. K jejich získání stačí spustit následující příkazy:

```
# Získání kódu výkonné knihovny
mkdir PyFC
svn co https://bimbo.fjfi.cvut.cz/repos/freeconf/trunk/src/\
PyFC PyFC

# Získání kódu klientské aplikace
mkdir PyQtFreeconf
svn co https://bimbo.fjfi.cvut.cz/repos/freeconf/trunk/src/\
PyQtFreeconf PyQtFreeconf

# Získání kódu návrháře balíků
mkdir FreeconfDesigner
svn co https://bimbo.fjfi.cvut.cz/repos/freeconf/trunk/src/\
FreeconfDesigner FreeconfDesigner
```

Aplikace projektu Freeconf předpokládají, že výkonná knihovna je umístěna v podadresáři `PyFC` v některém z adresářů, které jsou prohledávány interpreterem jazyka Python. To je i adresář, odkud je program spouštěn. Proto nejjednodušší cestou ke zprovoznění aplikací projektu je vytvoření symbolických odkazů na kód knihovny pomocí následující sady příkazů:

```
ln -s $PWD/PyFC PyQtFreeconf/PyFC
ln -s $PWD/PyFC FreeconfDesigner/PyFC
```

Ke konfiguraci vybraných cílových programů ještě potřebujeme stáhnout jejich konfigurační balíky a uložit je do podadresáře `.freeconf` v domovském adresáři uživatele. Také balíky jsou umístěny v repozitáři a lze je získat spuštěním příkazů:

```
mkdir -p ~/.freeconf/packages
svn co https://bimbo.fjfi.cvut.cz/repos/freeconf/trunk/packages\
~/.freeconf/packages
```

Spuštění klientské aplikace pak provedeme jednoduše z adresáře PyQtFreeconf:

```
cd PyQtFreeconf
python main.py <název konfiguračního balíku>
```