VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Rozvoj řídícího software virtuální laboratoře počítačových sítí

Diplomová práce

2007

Jan Vavříček

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 6. května 2007 ....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla. Poděkování patří také celému vývojovému týmu kolem Virtlabu, který se podílí na jeho chodu, vývoji a nápadům do budoucnosti. Největší dík patří hlavnímu muži Virtlabu, Petru Grygárkovi, který tento projekt vymyslel.

Abstrakt

Smyslem této práce je implementace vhodného mapovacího mechanismu, umožňující v jednom časovém okně spouštět paralelní úlohy, pokud to umožňují požadované parametry. Součástí práce je přepracování webové aplikace pro potřeby nové koncepce virtuální laboratoře.

Klíčová slova: virtuální síťová laboratoř, Virtlab, distribuovaný systém, uživatelské prostředí, mapovací algoritmus, PHP, XML

Abstract

The purpose of this work is to implement mapping mechanism with capability to allow running parallel tasks when technical parameters support it. Part of this work is revision and rebuilding of web application for new concept of virtual laboratory.

Keywords: virtual network laboratory, Virtlab, distributed system, user interface, mapping algorithm, PHP, XML

Seznam použitých zkratek a symbolu

ASSSK	–	Automatizovaného Systému pro Správu Sít'ových Konfigurací
CSS	–	Cascading Style Sheets
DOM	–	Document Object Model
DTD	–	Document Type Definition
GUI	–	Graphic User Interface
HTML	–	Hypertext Markup Language
HTTP	–	HyperText Transfer Protocol
IP	–	Internet Protocol
LDAP	–	Lightweight Directory Access Protocol
MD5	–	Message Digest 5
MIME	–	Multipurpose Internet Mail Extensions
MPLS	–	Multi-Protocol Label Switching
PEAR	–	PHP Extension and Application Repository
PHP	–	PHP Hypertext Preprocessor
QinQ	–	tunelování 802.1q rámců v 802.1q rámcích
SAX	–	Simple API for XML
SHA	–	Secure Hash Algorithm
SQL	–	Simple Query Language
TAR	–	Tape ARchive
TCP	–	Transmission Control Protocol
UDP	–	User Datagram Protocol
UML	–	User Mode Linux
URL	–	Uniform Resource Locator
VLAN	–	Virtual Local Area Network
VoIP	–	Voice over Internet Protocol
XML	–	Extensible Markup Language

Obsah

1 Úvod **1**
1.1 Cíle práce 1

2 Nová koncepce Virtlabu **3**
2.1 Omezení původní verze Virtlabu 3
2.2 Základní principy distribuovaného systému 5
2.2.1 Logické topologie 5
2.2.2 Princip propojení fyzických zařízení 6
2.2.3 Aktivace topologií 6
2.3 Komponenty distribuovaného systému 7
2.3.1 Použitá terminologie 7
2.3.2 Hlavní části lokality 9

3 Webová aplikace **12**
3.1 Použití komponentového přístupu 12
3.1.1 Objektový model v PHP 12
3.2 Systém generování stránek 13
3.3 Databáze 18
3.3.1 Integritní omezení (kontrola cizích klíčů) 18
3.4 Multijazyčnost 20
3.5 Autentizace a autorizace uživatelů 20
3.5.1 Autentizace 22
3.5.2 Autorizace 23
3.6 Správa uživatelů 25
3.7 Úlohy 26
3.7.1 Import úlohy z archivu 26
3.8 Rezervace 27
3.8.1 Komunikace s rezervačním serverem 28
3.8.2 Systém kvót 29
3.9 Práce se soubory 30
3.10 Souborové skupiny 31

4 Konfigurace Virtlabu a její zpracování **32**
4.1 Načtení dat s použitím SAX 32
4.2 Validace XML 33
4.3 Získání informací z načtených dat 33

5 Mapovací algoritmus **35**
5.1 Popis vybavení 35
5.2 Popis logické topologie 36
5.3 Popis mapovacího algoritmu 39
5.3.1 Schopnost zařízení být prvkem logické topologie 39
5.3.2 Ohodnocení zařízení 40

5.3.3 Tabulkové mapování 40
5.3.4 Vlastní průběh algoritmu 42

6 Závěr **43**
6.1 Vývoj Virtlabu do budoucna 43
6.1.1 Zahrnutí jazyků do databáze 43
6.1.2 Správa uživatelů 44
6.1.3 Logování 44

7 Reference **45**

Přílohy **45**

A Co je třeba udělat, když ... **46**
A.1 Přidáváme novou stránku 46
A.2 Měníme přístupová práva na stránky 46
A.3 Měníme databázový stroj 46
A.4 Chceme mít na každé stránce stejné informace 46

B Použitá DTD **47**

C Popis PHP tříd **49**
C.1 virtlabWeb 49
C.2 virtlabWebPage 49
C.3 virtlabLanguage 50
C.4 virtlabParserTaskupload 50
C.5 virtlabSQL 51
C.6 virtlabReservations 52
C.7 virtlabWebFile 53
C.8 virtlabXmlParser 54
C.9 virtlabParserEquipment 55
C.10 virtlabParserTopology 58
C.11 virtlabMapping 60

D Podpůrné PHP funkce **61**
D.1 virtlabSupportFunctions.php.inc 61
D.2 virtlabValues.php.inc 62

E Struktura databáze **64**

Seznam tabulek

1 Popis tříd používaných webovou aplikací 13
2 Popis tabulek v databázi 18
3 SQL tabulka filegroup_members 64
4 SQL tabulka filegroups 64
5 SQL tabulka files 64
6 SQL tabulka reservation_users 64
7 SQL tabulka reservations 65
8 SQL tabulka reserved_devices 65
9 SQL tabulka task_categories 65
10 SQL tabulka task_classes 65
11 SQL tabulka tasks 66
12 SQL tabulka tasks_categories 66
13 SQL tabulka languages 66
14 SQL tabulka users 67

Seznam obrázků

1 Původní architektura 4
2 Distribuovaná architektura 6
3 Propojení fyzických zařízení 7
4 Průběh generování stránek 17
5 Vazby klíčů s tabulkách 19
6 Typy uživatelů identifikovaných aplikací 23

Seznam výpisu zdrojového kódu

1 Funkce __autoload – „magická funkce"PHP verze 5 14
2 Šablona sekce **body** 15
3 Sestavení stránky z jednotlivých sekcí 16
4 Získání řetězce v požadovaném jazyce 20
5 Zdrojový kód metody virtlabLanguage::CZE_text 20
6 Ukázka fungování session 21
7 Vytváření odkazů s identifikátorem sezení 22
8 PHP vlastnost – funkce v proměnné 22
9 Autorizace položek menu 24
10 Autorizace položek menu 24
11 Autorizace každé ze stránek 25
12 Import úlohy 27
13 Skript pro stáhnutí souboru z databázi 31
14 Vytvoreni SAX parseru 32
15 Validace XML 33
16 Hlavička XML 34
17 DTD dokumentu pro popis vybavení 37
18 DTD dokumentu pro popis topologie 38
19 DTD equipment 47
20 DTD topology 47
21 DTD taskupload 48
22 XML soubor 55
23 Výstup parseru (třídy) virtlabXmlParser 56

1 Úvod

Rozvoj a zájem o projekt virtuální síťové laboratoře (**Virtlab**), která byla vyvíjena v rámci diplomových prací Ing. Pavla Němce ([1]) a Ing. Romana Kubína ([2]), si vyžádal změnu přístupu[1].

Dříve byl Virtlab chápán jako samostatný systém, který běží na jednom místě (v jedné lokalitě). Jelikož zařízení v něm provozovaná jsou finančně nákladná, bylo potřeba zajistit jejich sdílení s jinými univerzitami. Virtlab byl přepracován do podoby distribuovaného systému, který ovšem při výpadku, nebo úplné absenci spojení s ostatními Virtlab-lokalitami, bude fungovat jako samostatný systém, který bude mít zachovanou veškerou funkcionalitu – jen bude uživatelům nabízet menší množství fyzických zařízení (zařízení jen ze své lokality).

Tato zásadní změna koncepce si vyžádala přepracování, nebo alespoň malou modifikaci, prakticky všech částí Virtlabu. Celá nová koncepce je popsána ve dvou diplomových pracech – v této a v diplomové práci [3] Tomáše Hrabálka.

Na analýze a návrzích nové koncepce Virtlabu, stejně jako na implementaci jednotlivých komponent systému, se podílelo značné množství lidí. Týmovou spoluprací se podařilo definovat nové části distribuovaného systému, určit a vymezit jejich činnost. Architektura a jednotlivé komponenty distribuovaného Virtlabu byly přesně definovány Ing. Petrem Grygárkem, Ph.D (vedoucí této diplomové práce). Podrobný popis nového systému je obsahem části 2 (strana 3)[2].

1.1 Cíle práce

Cílem této práce bylo navrhnout a implementovat oddělení logické identity síťových prvků v popisu úloh od jejich identity fyzické[3], implementovat vhodný mapovací mechanismus, umožňující v jednom časovém okně spouštět paralelní úlohy, pokud lze jejich logické síťové prvky namapovat na různé fyzické laboratorní prvky s požadovanými parametry.

Celá problematika zmiňovaného oddělení logické identity a mapovacího algoritmu je popsána v části 5 (strana 35). Informace o logické topologii a fyzickém vybavení jsou uloženy v dokumentech ve formátu XML. Část 4 (strana 32) se zabývá problematikou čtení informací z tohoto formátu ve skriptovacím jazyce PHP, který byl opětovně použit pro tvorbu webové aplikace distribuovaného Virtlabu.

Přebudování celé koncepce Virtlabu si vyžádalo změny v jeho webovém uživatelském rozhraní. Původním záměrem bylo přepracovat existující PHP skripty webové aplikace. Bohužel na nich bylo znát, že prošly několika postupnými editacemi a nové funkce do nich byly implementovány mnohdy nevhodným způsobem. Jejich přepracování pro potřeby nové distribuované koncepce by si tak vyžádalo nemalé úsilí a značné množ-

[1] popis fungování nedistribuované verze Virtlabu je v části 2 (strana 3)

[2] Tato část je společná pro obě výše zmíněné diplomové práce (tuto a práci Tomáše Hrabálka [3]) a byla vypracována společně oběma autory.

[3] Logickou identitou síťového prvku se myslí jeho role vrcholu síťové topologie. Naproti tomu je jako fyzická identita prvku označena jeho hmatatelné existence v reálném světě.

ství času. Jako efektivnější se tedy ukázalo předělání webové aplikace jako celku úplně od začátku. Z původní verze byla převzata jen část analýzy databáze a implementace autentizace uživatelů proti LDAP serveru školy.

Při tvorbě nového webového rozhraní byl kladen důraz na budoucí snadnou rozšiřitelnost a co největší modularitu celého systému. Byl vymyšlen systém generování stránek, který umožňuje snadné přidání nových funkcí a nových částí systému. Problematikou nové webové aplikace se zabývá část 3 (strana 12). Stručné návody jak webové rozhraní modifikovat jsou popsány v příloze A (strana 46).

2 Nová koncepce Virtlabu[4]

Projekt s názvem Virtuální síťová laboratoř (Virtlab) byl iniciován potřebou vzdáleně zpřístupnit specializovaná zařízení, která se nacházejí ve školních laboratořích pro výuku předmětů zabývajících se počítačovými sítěmi, zejména pro potřeby studentů, kteří nemají možnost osobně navštěvovat zmíněné laboratoře, a tak zajistit možnost dálkové práce s těmito zařízeními z Internetu.

Původní návrh a realizaci nedistribuované verze po softwarové stránce vytvořil Ing. Pavel Němec v rámci své diplomové práce Virtuální síťová laboratoř ([1]), kterou obhájil v roce 2005. Zařízení umožňující vzdálené propojování laboratorních prvků vytvořil Ing. David Seidl jako součást své diplomové práce Automatizovaný systém správy síťových konfigurací ([4]), která byla rovněž obhájena v roce 2005. V průběhu provozu Virtlabu se ukázala potřeba systém více zabezpečit a doplnit některé funkce. Proto byla virtuální laboratoř o tyto prvky vylepšena Ing. Romanem Kubínem v rámci diplomové práce Zajištění bezpečnosti a implementace nových prvků řídícího systému virtuální laboratoře ([2]) obhájené roku 2006.

Virtuální síťová laboratoř umožňuje jednoduchou vzdálenou konfiguraci a vzájemné propojení síťových prvků (změny síťové topologie) z libovolného počítače připojeného k Internetu a to i více uživatelům připojeným současně[5].

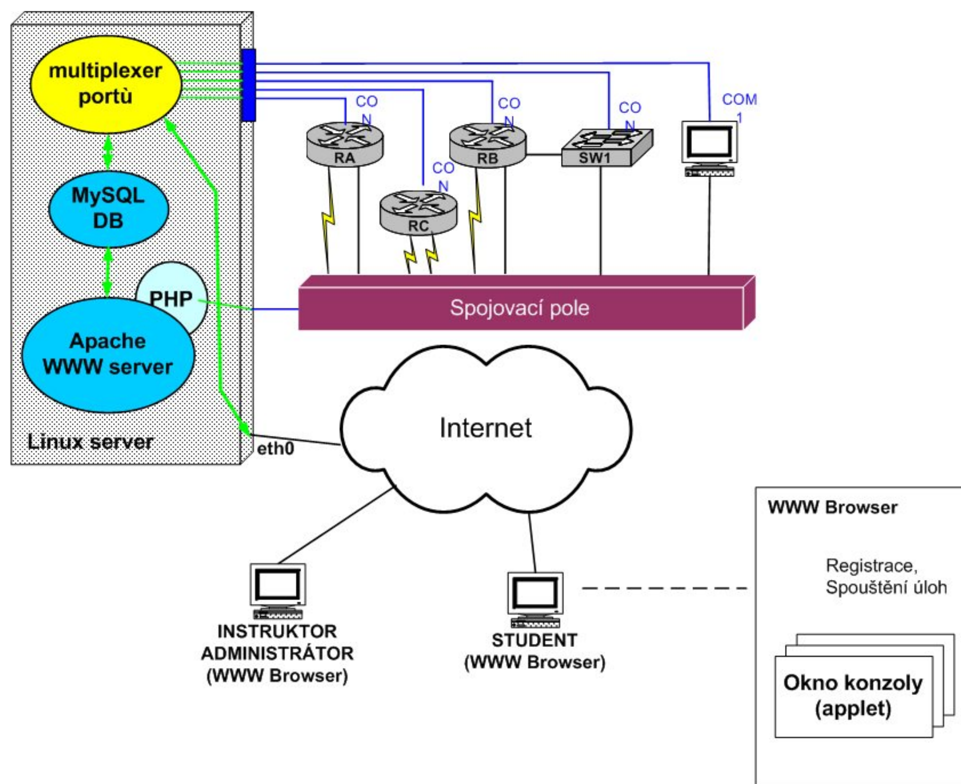
Distribuovaná virtuální síťová laboratoř, vytvořená v rámci této diplomové práce, umožňuje propojení více vzdálených lokalit (vzdálené propojení původních virtuálních laboratoří) tak, aby bylo možno vytvářet virtuální síťové topologie s použitím síťových prvků v zúčastněných lokalitách prostřednictvím tunelů vytvářených dynamicky skrze Internet. Navíc umožňuje tvorbu více topologií najednou, takže může několik uživatelů, z lokalit takřka z celého světa, naráz vytvářet a používat různé virtuální síťové topologie s využitím libovolných dostupných síťových prvků, aniž by je muselo zajímat, je-li tento prvek v jejich místní laboratoři nebo v cizí laboratoři na vzdáleném místě. A to vše navíc ovládají jednoduše prostřednictvím webového prohlížeče třeba z domu či ze zaměstnání.

2.1 Omezení původní verze Virtlabu

Omezením nejstarší verze Virtlabu byla pevná fyzická topologie. Jestliže bylo potřeba změnit fyzickou topologii, musel zodpovědný člověk jednotlivé prvky lokality ručně propojit do nově požadované topologie. Tento výrazně omezující faktor se podařilo překonat pomocí ASSSK-1 [4]. Toto zařízení dokáže propojovat sériové linky na základě příkazů, které jsou mu předávány. Zapojení určené fyzické topologie lze tedy jednoduše automatizovat. I když ASSSK-1 dokáže propojovat sériové linky i Ethernet, byla pro propojování Ethernetu časem použita jiná technika, aby se množství linek propojitelných

[4]Tato kapitola byla vytvořena kolektivně s Tomášem Hrabálkem, v jehož diplomové práci [3] je také tato část obsažena. Část 2.3.1 (strana 7) obsahuje definici jednotlivých pojmů, které se v distribuovaném Virtlabu objevují. Toto názvosloví je dílem Petra Grygárka [5], který souhlasil s jeho použitím.

[5]Na obrázku 1 (strana 4) je znázorněna architektura nedistribuovaného Virtlabu.



Obrázek 1: Původní architektura

pomocí ASSSK-1 nesnižovalo[6] – když pro propojování sériových linek nemáme žádné jiné řešení.

Zmíněné propojování ethernetu bylo časem realizováno technikou zvanou Q-in-Q na Cisco přepínačích řady 3500. Tato technika, jak už zkratka napovídá, je založena na technologii 802.1q, která umožňuje použití VLAN. Přidáním dvou bajtů dat do hlavičky ethernetového rámce lze na takzvané trunk lince identifikovat jednotlivé VLAN. V našem případě se ono přidání provádí ve skutečnosti dvakrát. Jedno označení mohou provádět propojovaná zařízení (musíme jim umožnit se propojovat ethernetovou trunk linkou), druhé označení provede zařízení, které propojuje jednotlivé prvky (vytvoří tunel simulující trunk linku), aby odlišilo jednotlivé toky dat, které odpovídají logickým spojům mezi propojovanými zařízeními.

Pro úplnost je třeba dodat, že musíme tunelovat i protokoly druhé vrstvy ISO modelu (STP, CDP, VTP), aby propojovaná zařízení nepoznala, že jsou propojena pomocí přepínače.

[6] ASSK-1 má jako každé fyzické zařízení jen konečný počet rozhraní. Ke kazdému z nich se může připojit sériová linka nebo Ethernet – nemohou ale fungovat zároveň. Propojování Ethernetových linek, by tak zmenšovalo množství propojitelných sériových linek.

I když byl vyřešen problém automatického zapojování topologií, stále bylo třeba vyřešit několik věcí. Jedním z omezujících faktorů je skutečnost, že v jeden okamžik může ve Virtlabu běžet daná úloha jen v jedné instanci, i kdyby byl dostatek prostředků pro více úloh současně. Odpovědnost za toto nese popis úlohy, ve které jsou definovány fyzické prvky, které se musejí použít. V popisech úloh se nemluví o směrovačích jako o logických prvcích s daným počtem rozhraní, ale jako o konkretním fyzickém prvku. Proto bylo potřeba definovat způsob popisu topologií na logické úrovni, popsat fyzické prvky každé lokality, při rezervaci úlohy zpracovat informace z těchto dokumentů a následně realizovat namapování jednotlivých logických prvků na prvky fyzické tak, aby byly splněny všechny požadavky topologie[7].

Dalším omezením současné verze je nemožnost sdílení prvků mezi několika Virtlab lokalitami. V realitě se objevují situace, kdy jedna z Virtlab lokalit zakoupí nějaké nákladné zařízení, ale toto zařízení nebude uživateli této lokality plně využíváno. V čase, kdy není používáno v lokalitě, v níž je nainstalováno, by jej ale plně mohla využít lokalita jiná, která si zakoupení stejného zařízení dovolit nemůže. Tento stav vyústil v návrh distribuovaného Virtlabu. Tento nápad s sebou přinesl řadu problémů, které bylo třeba vyřešit. Týkají se způsobu provádění rezervací, kdy je potřeba, aby o určité úloze v daný čas věděly všechny zúčastněné lokality, dále bylo třeba nalézt způsob propojení několika síťových prvků v různých lokalitách, jak provádět automatizované zapojování topologií napříč distribuovaným systémem, jak je automatizovaně konfigurovat či vzdáleně zpřístupnit uživatelům jejich ovládací konzole[8].

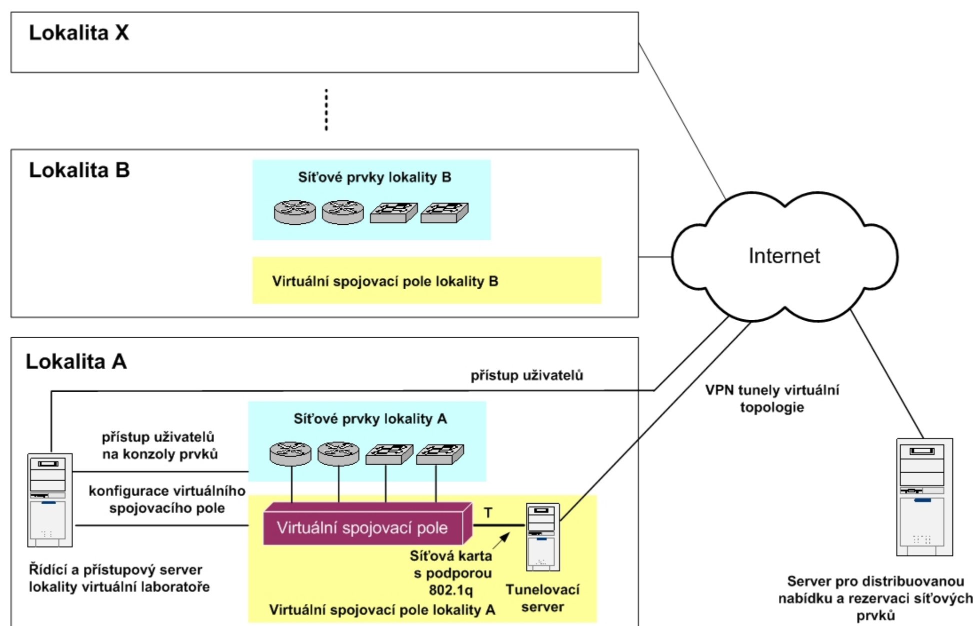
2.2 Základní principy distribuovaného systému

Jelikož cílem distribuovaného řešení je zpřístupnění síťových zařízení různých síťových laboratoří ve světě, je systém rozdělen na základní jednotky, které jsou nazvány lokality, a které reprezentují právě tyto jednotlivé síťové laboratoře. Tedy lokalita je místo, kde jsou fyzicky umístěna síťová zařízení, která se zpřístupňují uživatelům místním i z ostatních lokalit. Každá lokalita má svá pravidla, jež definují, která zařízení a ve kterém čase jsou k dispozici vybraným vzdáleným lokalitám. V každé lokalitě proto běží Rezervační server, který obhospodařuje práva uživatelů a lokalit a zabezpečuje zapůjčování prvků. Dále je zde spuštěn Konzolový server, Konfigurační server a webový server, který je určen vždy pro místní uživatele.

2.2.1 Logické topologie

Jestliže uživatel chce vytvořit jistou úlohu, musí vytvořit návrh žádané síťové topologie. Ten nepředepisuje přímo fyzická zařízení, ale určuje pouze typy síťových zařízení a způsob jejich propojení, proto jej nazýváme logická topologie. Teprve až je uživatelem zvolena logická topologie, zjišťuje se, je-li ji v daném čase možno realizovat, vyberou se vhodná síťová zařízení - a to jak lokální, tak i vzdálená, a prostřednictvím Rezervačních serverů se tyto prvky pro danou topologii vyhradí.

[7]Celá tato problematika je předmětem řešení této diplomové práce.
[8]Celá tato problematika je součástí diplomové práce Tomáše Hrabálka ([3])



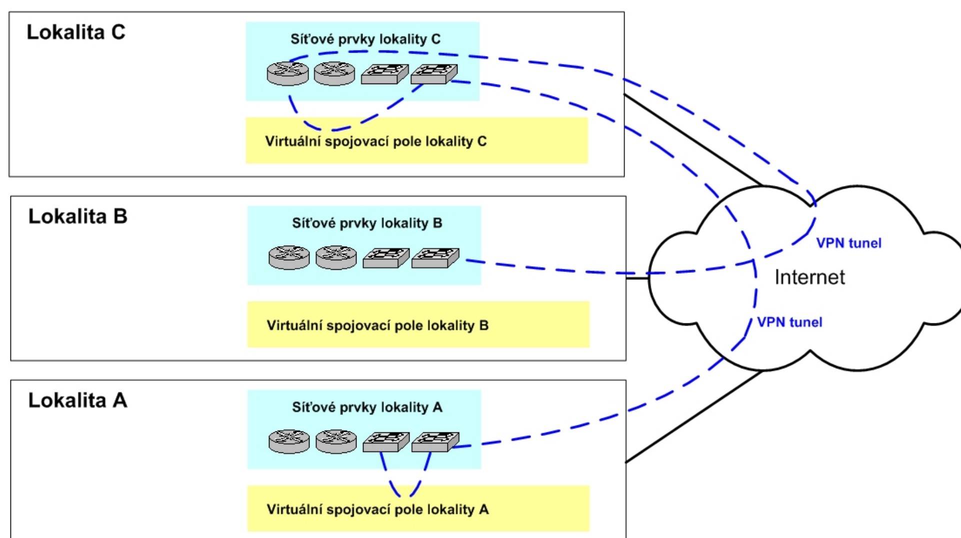
Obrázek 2: Distribuovaná architektura

2.2.2 Princip propojení fyzických zařízení

Ke spojení zařízení v rámci jedné lokality je určeno automatické spojovací zařízení - virtuální spojovací pole, které dokáže samočinně spojit libovolná sériová rozhraní i ethernetové porty. V současně době je používáno zařízení ASSSK-1, které umožňuje přímé propojení signálových vodičů sériových a ethernetových 10BaseT portů, a přepínač Cisco 3550, kterým se vytvářejí virtuální ethernetové segmenty pomocí technologie VLAN. Vzdálené propojení dvou segmentů distribuovaného virtuálního spojovacího pole má na starosti Tunelovací server, který ze dvou těchto různých segmentů, které používají různé lokální VLAN, udělá jeden logický ethernetový segment. Takto můžeme propojit například jeden ethernetový port na přepínači v jedné lokalitě s ethernetovým portem směrovače v jiné lokalitě, přičemž tato zařízení pracují, jakoby byla spojena přímo přes ethernetový bridge, jehož existence je pro protokoly přes něj běžící utajena.

2.2.3 Aktivace topologií

Při započetí rezervace vyvolá Rezervační server prostřednictvím Aktivačního skriptu generování konfigurací topologie pro virtuální spojovací pole a tunelovací servery. Poté odešle soubory s vygenerovanou konfigurací všem konfiguračním serverům v lokalitách podílejících se na rezervaci. Tyto pak zajistí jejich nahrání na fyzické spojovací prvky, respektive Tunelovací server, v dané lokalitě.



Obrázek 3: Propojení fyzických zařízení

2.3 Komponenty distribuovaného systému

2.3.1 Použitá terminologie

Lokalita Lokalita je jedna lokální, autonomní instance Virtlabu schopná samostatného provozu i spolupráce s jinými lokalitami prostřednictvím připojení k volnému Internetu. Spolupráce spočívá jednak v nabízení svých laboratorních prvků pro logické topologie požadované uživateli z jiných lokalit (tím vznikají distribuované topologie pomocí propojovacích tunelů) a jednak v používání laboratorních prvků nabízených jinými lokalitami pro logické topologie požadované uživateli vlastní lokality. Lokality jsou pojmenovávány textovými řetězci[9].

Lokalita obsahuje:

- laboratorní prvky lokality (mohou volitelně zahrnovat jeden či více serverů simulujících stanice (Xen [10]) a Cisco 7200 (Dynamips [11]))
- řídící server lokality (Virtlab server)
- konzolový server
- segment virtuálního spojovacího pole
- konfigurační server spojovacího pole
- rezervační server

[9] kódování UTF-8

Laboratorní prvky Laboratorní prvky jsou síťové prvky v jednotlivých lokalitách, připojené k lokálnímu segmentu virtuálního spojovacího pole, které jsou k dispozici pro práci studentů na jednotlivých úlohách. Může jít o fyzické síťové prvky nebo prvky simulované (XEN [10], Dynamips [11]). Laboratorní prvky jsou globálně pojmenovávány ve tvaru `jmeno@lokalita`[10]. Vlastnosti jednotlivých laboratorních prvků jsou popsány v XML.

Uživatelé Uživatelé jsou zaváděni v jednotlivých lokalitách. Mají jména jednoznačná v rámci lokalit. Globálně jednoznačné jméno uživatele vznikne spojením se jménem domovské lokality a má tvar `jmeno@lokalita`. Uživatelé se autentizují v lokalitě, do které náleží, kde jsou také definována jejich práva či role v systému. V rámci lokality mohou být definovány lokální skupiny uživatelů. Přiřazení uživatele požadujícího sestavení úlohy do skupiny uživatelů může ovlivnit práva na výběr prvků při mapování logické topologie úlohy na fyzickou (toto mapování sestavuje mapovací algoritmus běžící v lokalitě uživatele, takže má definice skupin uživatelů k dispozici).

Logická topologie úlohy Logická topologie úlohy je popis požadavků na laboratorní prvky žádaných pro řešení určité úlohy včetně popisu požadavků na jejich vzájemné propojení. Každý laboratorní prvek je v popisu zastoupen jedním logickým laboratorním prvkem. Logická topologie je popsána v XML. [11]

Fyzická topologie úlohy Fyzickou topologií úlohy rozumíme soubor laboratorních prvků-zařízení (i z různých lokalit) namapovaných algoritmem mapování na jednotlivé logické prvky odpovídající logické topologii úlohy, včetně přiřazení fyzických rozhraní laboratorních prvků ke spojům logické topologie. Fyzická topologie může být distribuovaná, tedy obsahovat laboratorní prvky z různých lokalit a spoje mezi nimi realizovat prostřednictvím propojovacích tunelů. Spoje uvnitř lokality jsou realizovány spojovacím polem.

Úloha Úlohou rozumíme definici zadání úkolu pro uživatele popisující mimo jiné logickou topologii úlohy. Úloha může dále obsahovat i vzorové řešení v podobě konfigurace jednotlivých zařízení.

Spuštění úlohy Spuštěním úlohy rozumíme v čase vymezené propojení laboratorních prvků pro práci studentů podle požadavků popsaných logickou topologií úlohy. Časový interval sestavení úlohy (timeslot) je chápán obecně a není vázán na žádný fixní časový rastr. Zdrojem logické topologie může být buďto tabulka předdefinovaných úloh nebo GUI (u topologie na přání studenta).

[10] kódování UTF-8

[11] V rámci logické topologie mluvíme o vrcholech (logický prvek topologie) a hranách-linkách (propojení dvou logických prvků). Terminologie byla převzata z Teorie grafů.

Timeslot Timeslotem je nazýván časový úsek rezervovaný studentem pro řešení určité úlohy. Začátek ani konec není oproti předchozí verzi Virtlabu vázán žádným pevným časovým rastrem. Prvních 5 minut timeslotu je vyhrazeno na vymazání předchozí konfigurace z laboratorních prvků použitých v úloze a spojení fyzické topologie, student během nich nemůže přistupovat na laboratorní prvky.

Virtuální spojovací pole Distribuovaný spojovací systém, který je založený na technologii VLAN a tunelování VLAN (802.1q rámců) pomocí UDP (vlastní enkapsulační formát – viz [3]) přes volný Internet. Spojování laboratorních síťových prvků v lokalitách se děje jejich zařazováním do stejných VLAN (příp. VLAN tunelů QinQ při spojování trunků) na přepínačích Cisco 3550, tunely přes Internet jsou zajišťovány vlastním SW – tunelovacím démonem běžícím na tunelovacím serveru.

Řídící skripty virtuálního spojovacího pole Řídící skripty virtuálního spojovacího pole na základě konfiguračních souborů a textového popisu propojení požadované fyzické topologie, který je vyvořen mapovacím algoritmem, vygenerují konfigurační příkazy pro všechny segmenty virtuálního spojovacího pole (včetně tunelovacích serverů) ve všech lokalitách. Textový popis propojení fyzické topologie obsahuje v jednotlivých řádcích dvojice jmen fyzických laboratorních prvků a jejich rozhraní, které mají být propojeny. Skripty spouští před zahájením úlohy Aktivátor konfigurací lokality, jejíž uživatel rezervaci úlohy vyžádal. Vygenerované konfigurační příkazy jsou zaslány do segmentů virtuálních spojovacích poli všech lokalit zúčastněných v distribuované topologii dané úlohy prostřednictvím jejich Konfiguračních serverů.

2.3.2 Hlavní části lokality[12]

Rezervační server Původní verze virtuální síťové laboratoře umožňovala jednotlivým uživatelům, aby si mohli zarezervovat určitý čas, kdy budou s Virtlabem exkluzivně pracovat a nemohlo tedy dojít ke konfliktu, kdy více uživatelů současně zde bude chtít najednou systém používat. Nevýhodou bylo, že rezervace mohly být uskutečněny pouze v rámci přesně daných timeslotů, což neumožňovalo přílišnou flexibilitu v čase. Dále virtuální laboratoř neumožňovala spouštění více úloh najednou a proto při návrhu distribuované varianty bylo třeba počítat s tím, že bude moci najednou pracovat i více uživatelů na různých úlohách a to dokonce z více lokalit. Bylo proto nutné navrhnout nový rezervační systém, který bude umět nejen rezervovat uživateli nějaký čas, ale bude přímo zprostředkovávat rezervaci fyzických síťových zařízení v libovolné lokalitě distribuované virtuální laboratoře.

Aby ovládací software (PHP skript) věděl, které síťové prvky z celého distribuovaného systému může uživateli poskytnout, bylo třeba implementovat i funkci vyhledávání volných zařízení v systému. A v neposlední řadě vznikl požadavek na to, aby určité

[12]Součástí této diplomové práce je řídící server – ostatní části jsou řešeny v diplomové práci Tomáše Hrabálka [3]

prvky byly zpřístupněny daným lokalitám jen po určitý čas, nejlépe v týdenním časovém plánu.

Rezervační server je démon, který běží vždy na jednom serveru v každé lokalitě distribuovaného systému a tedy v celém systému běží tolik instancí, kolik je definováno lokalit. Tento server má vždy určeno jméno své lokality, jména vzdálených lokalit a IP adresy jejich rezervačních serverů. Dále má ke každé lokalitě uveden seznam síťových prvků místní lokality, které může vzdálené lokalitě poskytnout k zarezervování a to podle daného týdenního rozvrhu.

Když ovládací software chce zarezervovat určité prvky, pošle rezervačnímu serveru dotaz, které prvky jsou globálně v celém distribuovaném systému pro něj v určeném čase k dispozici. Rezervační server žádost zpracuje a odešle i do vzdálených lokalit. Každá lokalita má přiložen XML soubor s popisem vybavení místní laboratoře. Z něj rezervační server vybere nezarezervované a povolené prvky a odešle výsledný XML soubor tazateli. Rezervační server, který rozdistribuovával dotaz ovládacího software, poskládá všechny přijaté XML popisy vybavení do jediného souboru, který vrátí tazateli. Není-li k dispozici žádný síťový prvek, je vrácen platný soubor, ovšem bez zařízení.

Ovládací software XML soubor zpracuje, vybere nejvhodnější fyzické prvky a může dále žádat o zarezervování seznamu síťových prvků v daném čase. K tomu vygeneruje unikátní rezervační id ve tvaru `pořadové_číslo@lokalita`. Rezervační server žádost rozdělí a pošle každé vzdálené lokalitě v žádosti jen ty prvky, které jí patří. Aby se eliminovaly konflikty, místní rezervační server požádá vzdálené rezervační servery, aby onu rezervaci považovali za dočasnou. Povede-li se dočasná rezervace u všech žádaných lokalit, je všem rezervace potvrzena trvale. Může ovšem dojít k tomu, že si někdo před námi již daný síťový prvek zarezervoval a není možné rezervaci u některých z rezervačních serverů provést. Pak k potvrzení samozřejmě nedojde a po dané časové prodlevě vyprší dočasné rezervace.

Konzolový server Konzolový server zprostředkovává přístup k sériovým nebo telnetovým konzolím síťových zařízení prostřednictvím TCP protokolu. Je využíván Java Appletem, který běží ve webovém ovládacím rozhraní, což umožňuje uživateli jednoduchý přístup na síťová zařízení. Zároveň slouží i jako proxy server, který zprostředkovává přístup k zařízením, která jsou připojena ke vzdáleným konzolovým serverům, kam nemá místní uživatel přímý přístup. Také prostřednictvím speciálního PHP skriptu ověřuje, jsou-li požadavky uživatele oprávněné a tím konzoly prvků zabezpečuje od neautorizovaného přístupu.

Tunelovací server Při návrhu distribuované varianty virtuální síťové laboratoře se ukázala potřeba propojovat dvě vzdálená zařízení do jednoho virtuálního ethernetového segmentu. A to dynamicky, podle potřeby uživatele, kdy ještě navíc bylo potřeba počítat s tím, že v každé lokalitě může být zařízení, které má být do virtuálního ethernetového segmentu, připojeno do jiné VLAN. Při zkoumání možných řešení nebyl nalezen žádný vhodný existující software, a proto bylo přistoupeno k implementaci vlastního řešení.

Tunelovací server je démon běžící na zvláštním serveru, jež je součástí Distribuované virtuální laboratoře a jeho úkolem je zajistit virtuální propojení různých VLAN mezi jednotlivými lokálními virtuálními laboratořemi a také různých VLAN v rámci jedné lokality. Takto lze zajistit, aby síťová zařízení, která jsou připojena na tunelované VLAN, byla zapojena jakoby v jediném zdánlivém ethernetovém segmentu, přestože je každé zařízení v jiné lokalitě a případně v jiné VLAN.

Konfigurační server Konfigurační server je jednoduchý síťový démon, jehož úkolem je příjem konfiguračních souborů pro jednotlivé spojovací prvky lokálního segmentu virtuálníhé spojovacího pole a také nahrání těchto konfigurací do příslušných zařízení. Tímto dochází k aktivaci žádané síťové topologie.

Aktivátor konfigurací Úkolem aktivačního skriptu je zajistit spuštění skriptu generujícího konfigurace pro spojovací pole, dále skriptu pro vytvoření konfigurace pro tunelovací servery a nakonec spuštění skriptu, který tyto konfigurace pošle určeným konfiguračním serverům. Aktivační skript je spuštěn před začátkem dané úlohy rezervačním serverem.

Řídící server Řídící server lokality je vlastně jediným místem, kde běžný uživatel přichází s Virtlabem do styku. Kromě řadových uživatelů jsou v systému rozlišování i administrátoři, správci úloh a tutoři. Webový server umožňuje uživatelům si na určitou dobu zarezervovat logickou topologii, kterou si vyberou. Řídící server komunikuje s rezervačním serverem, který mu nabízí fyzická zařízení, která jsou k dispozici. Seznam zařízení a logická topologie, jsou vstupními argumenty mapovacího procesu, který pro jednotlivé vrcholy logické topologie vybere adekvátní fyzická zařízení a vytvoří data, které slouží dále k vygenerování konfigurací spojovacích prvků.

Konzolový applet Klíčovou součástí webového rozhraní uživatelů je applet, jehož úkolem je zajistit uživateli vzdálený přístup k sériovým, případně telnetovým konzolám síťových prvků. Konzolový applet přijímá data od konzolového serveru (CServer) a zpět mu odesílá vstupy od uživatele.

3 Webová aplikace

Webové uživatelské rozhraní zaujímá v celém systému Virtlab důležitou roli, jelikož je jedinou částí, která je viditelná pro běžné uživatele a je vstupní branou do virtuální laboratoře počítačových sítí. Webovou aplikaci však netvoří jen dynamicky generované HTML stránky, ale také části, které komunikují s rezervačním serverem lokality[13], umožňují načítání archivů s úlohami[14], čtou data z XML souborů[15] a provádějí mapování prvků logické topologie na prvky fyzické[16].

Proti předchozí verzi webového rozhraní bylo přepracováno prakticky vše. Jediná část, která zůstala stejná, je výsledný design webových stránek, který se ukázal jako velmi efektivní a dostačující. Design kopíruje vzhled systému KatIS[17], který svojí efektivitu prokazuje neustále.

Nově navržená struktura a koncepce se skládá z několika logických komponent, které jsou v následujících kapitolách vysvětleny a popsány.

Pro tvorbu dynamických webových stránek byl použit, stejně jako v předchozích verzích, jazyk PHP. Primárně použitým databázovým strojem zůstalo MySQL. Stručný popis struktury databáze je v části 3.3 (strana 18), podrobný popis jednotlivých tabulek a schéma vazeb klíčů mezi nimi je v příloze E (strana 64).

3.1 Použití komponentového přístupu

Při tvorbě jednotlivých částí webové aplikace byl kladen velký důraz na pozdější snadnou rozšířitelnost a jednoduchou modifikaci jednotlivých komponent. Tyto požadavky nejlépe splňuje komponentárně orientovaný přístup.

V tabulce 1 (strana 13) je zobrazen seznam tříd, které webové rozhraní používá, s krátkým popisem jejich funkce. Podrobný popis jednotlivých tříd je v příloze C (strana 49). Obecný popis funkcí poskytovaných třídami je zmíněn v jednotlivých kapitolách, pojednávajících o příslušných částech webové aplikace.

Jednotlivé třídy jsou komponentami, ze kterých se webová aplikaci skládá. Některé vytvářejí HTML stránky, které tvoří rozhraní aplikace, jiné zprostředkovávají komunikaci s dalšími servery, ...

3.1.1 Objektový model v PHP

S vývojem skriptovacího jazyka PHP se měnila i úroveň implementace objektového modelu v něm. Zatím poslední verze PHP[18] má již objektový model implementován v dostatečné míře.

[13] viz část 3.8.1 (strana 28)
[14] viz část 3.7.1 (strana 26)
[15] viz část 4 (strana 32)
[16] viz část 5 (strana 35)
[17] http://katis.cs.vsb.cz
[18] v současné době verze 5.2

třída	popis
virtlabLanguage	zajišťuje multijazyčnost webové aplikace (viz část 3.4 na straně 20)
virtlabParserTaskupload	zajišťuje extrakci informaci z importovaného archivu s úlohou (viz část 3.7.1 na straně 26). Konkrétně z XML souboru v něm obsaženém (práce s XML je popsána v části 4 na straně 32).
virtlabReservations	zajišťuje komunikaci s rezervačním serverem lokality (viz část 3.8.1 na straně 28)
virtlabSQL	zajišťuje komunikaci s databázovým strojem, který obsahuje všechna data (viz část 3.3 na straně 18)
virtlabWeb	zajišťuje mechanismus generování stránek (viz část 3.2 na straně 13)
virtlabWebFile	zajišťuje vkládání a extrakci souborů uložených v databázi (viz část 3.9 na straně 30)
virtlabWebPage	zajišťuje mechanismus generování stránek (viz část 3.2 na straně 13)

Tabulka 1: Popis tříd používaných webovou aplikací

Práci s objekty v PHP verze 5 velmi usnadňuje funkce `__autoload`[19], která (pokud je ve skriptu definována) se provede při konstrukci objektu zatím nedefinované třídy. Této funkci je jako jediný argument předán název této nedefinované třídy. Pokud budeme třídy ukládat po jedné do souboru se stejným názvem, usnadníme si načítání všech potřebných definic tříd. Na výpisu 1 (strana 14) je zobrazen kód použitý k načítání souborů s definicí třídy. Z výpisu je vidět, že webová aplikace má definice tříd uloženy v adresáři, který je zapsán v proměnné `$dir_class`. Soubor má stejné jméno jako v něm definovaná třída a příponu `php.inc`. Spojení této vlastnosti PHP a systému generování stránek má za následek, že funkce `__autoload` je definovaná pouze jednou a v žádném ze skriptů již není žádný ze souborů s definicí třídy načítán.

3.2 Systém generování stránek

Analýzou předchozí verze Virtlabu se ukázalo jako velmi nevýhodné vytvářet webové stránky způsobem, že si každá z nich vykreslovala celé okno (každá řešila, jestli je uživatel přihlášený, vykreslovala menu a všechny další činnosti nutné k vytvoření celkového vzhledu).

[19]Uvozující znaky jsou dvě podtržítka (stejně jako u konstruktoru). Funkce, jejichž jména začínají dvěma podtržítky, si PHP vyhrazuje pro sebe, a proto by se neměly používat pro uživatelské funkce. Funkce s tímto začátkem PHP označuje jako „magické metody“ – což jsou konstruktor, destruktor, funkce pro serializaci a deserializaci objektu, . . .

```
function __autoload($jmeno_tridy)
{
    global $dir_class;
    require_once($dir_class . $jmeno_tridy . ".php.inc");
}
```

Výpis 1: Funkce __autoload – „magická funkce" PHP verze 5

Situace byla částečně usnadněna použitím funkcí, které tyto rutinní záležitosti (např.: zjištění, zda-li je uživatel přihlášen) řešily a byly definovány v souboru, který byl vkládán do každého skriptu. Abychom byli schopni webovému rozhraní, které je vlastně tím nejdůležitějším místem virtuální laboratoře a jediným přístupovým místem pro uživatele, zaručit udržitelný rozvoj, snadnou integraci nových funkcí a jednoduchou správu, musel být navrhnut systém jiný.

Byl vytvořen systém „dvojího generování", který je postaven na co nejjednodušším principu, který využívá vkládání stránek do skriptu[20]. První krok generování sestaví výslednou HTML stránku z těchto sekcí:

head jde o úplný začátek HTML stránky, takže zde najdeme značky <!DOCTYPE>, <html>, <head> – značka head zde není uzavřena. V této sekci se vkládají odkazy na soubory s kaskádovými styly, které definují vzhled stránek

headbody obsahuje značky </head> a <body>. V této sekci se vkládají javaskripty, které se umisťují hned za <body>

body tato sekce generuje vlatní tělo HTML stránky, které se ale skládá z více částí, které se vloží v druhém kroku

tail obsahuje konec celého HTML - značky </body> a </html>

Jak už bylo naznačeno, v některých sekcích se vkládají do HTML další části. V sekci **head** to jsou například kaskádové styly, v sekci **headbody** javaskripty. Asi nejdůležitější sekcí je **body**, která vytváří celkový vzhled výsledné stránky.

Z původní verze byl převzat vzhled stránek, který je odvozen od systému KatIS, který je na škole používán. Stránka je rozdělena na dvě hlavní časti – menu na levé straně a tělo stránky na pravé. Menu se dále skládá z loga Virtlabu[21] v horní části a samotných odkazů na jednotlivé části webu pod ním. Plocha vlastní stránky je rozdělana na tři části: hlavičku, tělo[22] a patičku. V současné chvíli nejsou ve Virtlabu hlavičky ani patičky využívány, ale pokud by například bylo potřeba v každé stránce na konec vložit nějaké informace (např. o licenci, . . .), lze to provést velmi jednoduchým a rychlým způsobem.

Výpis 2 (strana 15) zobrazuje soubor, používaný jako definice sekce **body**.

[20]Toto vkládání využívá standardní funkce require_once, které vloží v místě svého volaní soubor jí určený. Od funkce include (ekvivalentu stejnojmenné funkce z jazyka C) se liší zpracováním chyb – viz [16]

[21]v současné době žádné konkrétní logo nemá, proto je zde umístěna fotografie ASSSK-1

[22]nezaměňovat s HTML značkami head a body – jde o kontejnerovou značku div

```
<div id="menu">
    <div id="logo"></div>
    <div id="menu_list">
        <?php require($_part["menu"]); ?>
    </div>
</div>
<div id="main">
    <div id="header">
        <?php require($_part["header"]); ?>
    </div>
    <div id="body">
        <?php require($_part["body"]); ?>
    </div>
    <div id="footer">
        <?php require($_part["footer"]) ; ?>
    </div>
</div>
```

Výpis 2: Šablona sekce **body**

Z výpisu je patrné, z jakých částí se tělo stránky skládá. Samotné pozicování prvků ve stránce je vyřešeno v externím CSS souboru. Příkazy pro PHP, které se ve výpisu objevují, budou vysvětleny následně.

Aby byla usnadněna práce s generováním kódu jednotlivých sekcí, byla vytvořena pro tyto účely třída `virtlabWebPage`. Objekt této třídy při svém vzniku vyžaduje cestu k souboru se šablonou sekce[23], volitelným parametrem jsou libovolná data, poskytnutá šabloně, a přístupná v ní přes proměnnou `$_part`. Detailní popis této třídy je na stránce 49.

Jelikož se výsledná stránka skládá z několika sekcí, byla vyrobena „obalující" třída `virtlabWeb`, do které se uloží jednotlivé definice sekcí a následně se najednou vygeneruje celá stránka. Výpis zdrojového kódu 3 (strana 16) ukazuje, jak ve výsledku vypadá veškerý kód, který je potřeba napsat, pokud potřebujeme ve Virtlabu vyrobit jednu stránku[24].

Ve výpisu je použit reálný kód, generující stránku pro výpis úloh v systému. Nyní si popišme význam jednotlivých řádků[25]:

`$all = new virtlabWeb()` vlastní vytvoření objektu virtlabWeb

`case "15"` bude vysvětleno dále, zatím jen naznačme, že se jedná o hodnotu proměnné, která volí, jaká stránka se má zobrazit, jelikož uživatel komunikuje pouze se stránkou `index.php`

[23]takovou šablonou je například výpis 2 (strana 15)

[24]Pokud půjdeme do důsledků, tak velikost nezbytného kódu je ještě menší, protože např. všechny stránky používají hlavní soubor s CSS styly. Tyto stejné hodnoty můžeme nastavit ještě před samotnou vícenásobnou podmínkou a kód pro jednu stránku bude řádově o polovinu menší.

[25]proměnné `$dir_*` obsahují cesty v souborovém systému k adresáři s požadovanými soubory

```
...
$all = new virtlabWeb();
...
case "15"://tasks-list
    $css = array("css" => array($dir_css . "virtlab_style.css",
                                $dir_css . "task.css"));
    $all->AddPage($dir_page . "head.php.inc", $css);
    $script = array("script" => $dir_java . "login_form_check.js");
    $all->AddPage($dir_page . "headbody.php.inc", $script);
    $body = array("menu"   => $dir_virt . "menu.php",
                  "header" => $dir_page . "empty",
                  "body"   => $dir_virt . "tasks-list.php",
                  "footer" => $dir_page . "empty");
    $all->AddPage$dir_page . "body.php.inc", $body);
    $all->AddPage($dir_page . "tail.php.inc");
    break;
...
$all->PrintPages();
```

Výpis 3: Sestavení stránky z jednotlivých sekcí

`$css = array("css"...` tvorba pole, v němž jsou odkazy na jednotlivé CSS soubory používané stránkou. V tomto případě je jich několik. Soubor `virtlab_style.css` obsahuje jen obecné definice vzhledu (pozicování menu, definice písem, ...), kdežto soubor `task.css` má v sobě definovány styly elementů, použitých pouze v této jedné stránce.

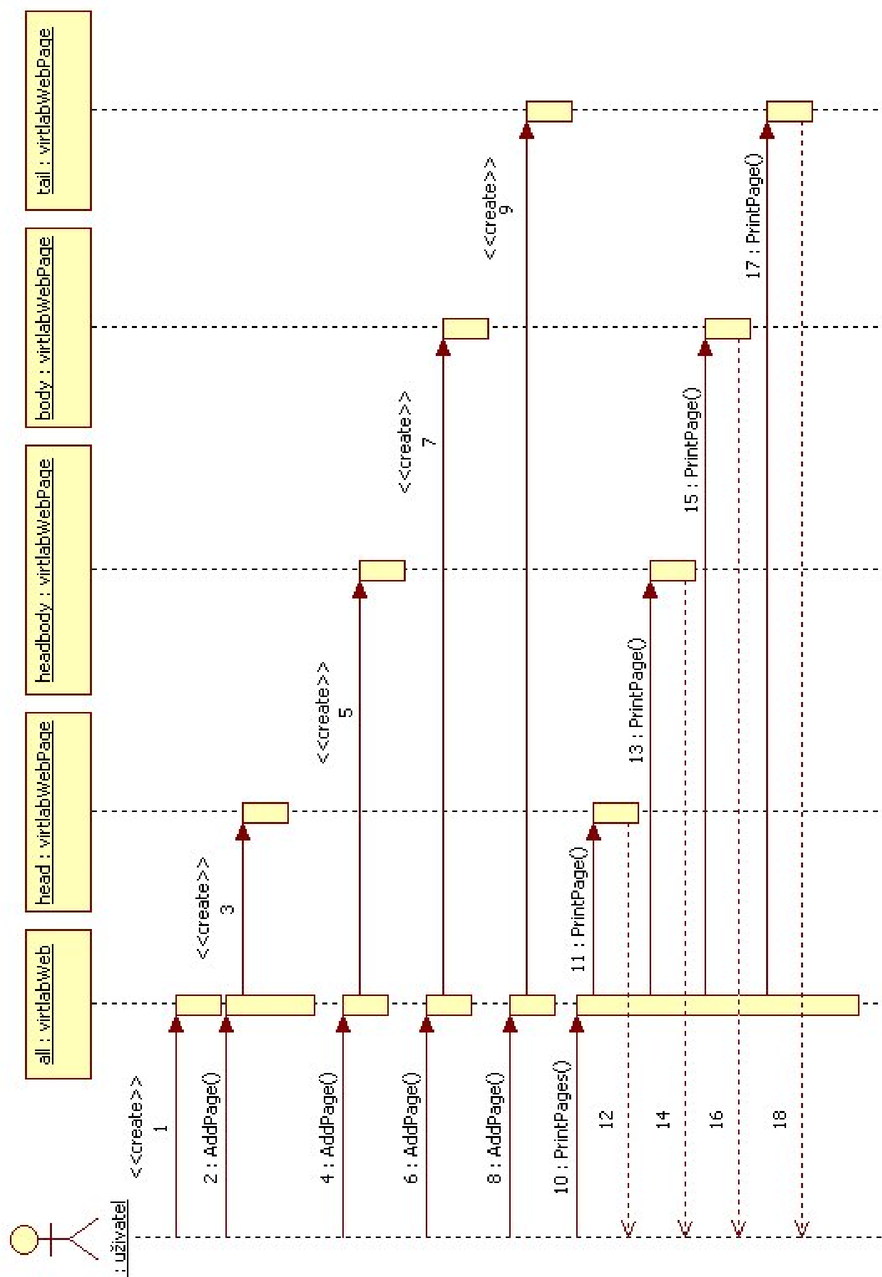
`$all->AddPage...` přidání sekce do hlavního objektu. Prvním argumentem funkce je cesta k souboru se šablonou, druhým volitelným parametrem jsou data pro šablonu.

`$body = array...` tento řádek souvisí s výpisem 2 (strana 15) šablony sekce **body**. Do jednotlivých buněk pole se ukládají cesty k souborům, tvořící tělo HTML stránky. Pokud pomineme přidávání CSS souborů nebo javaskriptů, tak jediným místem, odlišujícím od sebe stránky je kontejner[26] *body*, určující soubor se skriptem, který realizuje samotnou funkci stránky (v našem výpisu jde o zobrazení úloh v systému).

`$all->PrintPages()` zpracuje všechny stránky v pořadí, v jakém byly do objektu `$all` vloženy a vygeneruje tak všechna data HTML stránky. Tento příkaz je posledním příkazem souboru `index.php`

Celý průběh generování stránky z jednotlivých sekcí je zobrazen na obrázku 4 (strana 17). Ukázka 3 (strana 16) ukazuje kód, který je potřeba pro „vytvoření“ jedné stránky. Jak je možné vytušit z konstrukce **case "15":**, stránky se vybírají pomocí proměnné – konkrétně jde o proměnnou `page`, která se předává webovému prohlížeči metodou GET (např.: `index.php?page=15`).

[26] HTML značka `div` se často označuje jako kontejner



Obrázek 4: Průběh generování stránek

3.3 Databáze

Nezbytnou součástí každé webové aplikace je databáze, která obsahuje všechna potřebná data. Virtlab není vyjímkou. Jako databázový server byl vybrán MySQL. Kvůli používání techniky kontroly cizích klíčů musel být v MySQL použit typ tabulek InnoDB, který tyto funkce podporuje. Do databáze se ukládá prakticky vše potřebné, včetně souborů popisujících jednotlivé úlohy – tato vlastnost je popisována v části 3.9 (strana 30). Databázový server by se mohl stát uložištěm také dalších dat a informací, které se objevily v průběhu implementace, ale již nemohly být zahrnuty v tomto vývojovém cyklu – o těchto možnostech budoucího rozvoje pojednává část 6.1 (strana 43).

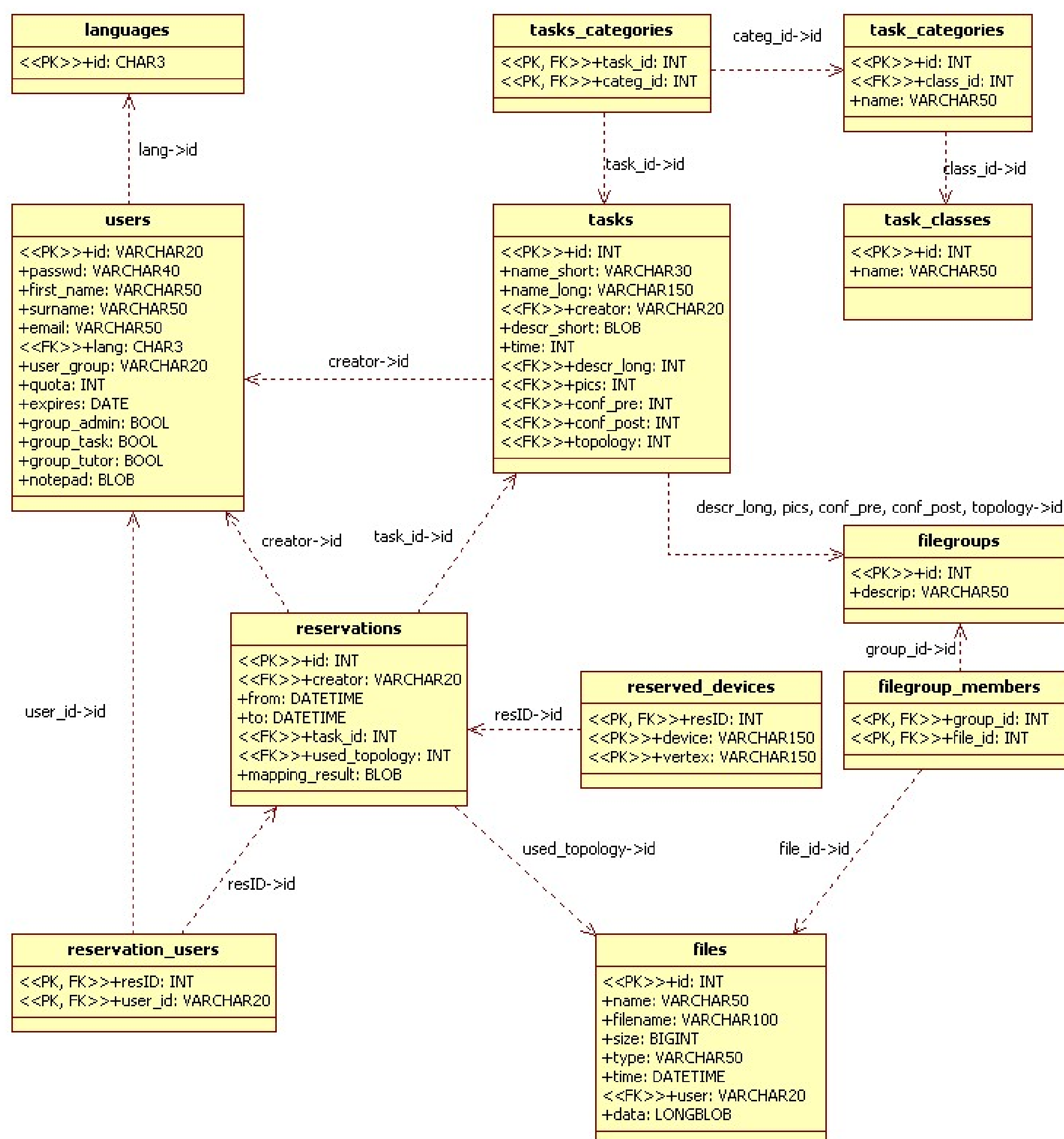
tabulka	**popis**
filegroup_members	vazební tabulka mezi `files` a `filegroups`
filegroups	obsahuje definice skupin souborů – viz část 3.10 (strana 31)
files	obsahuje soubory vkládané do webové aplikace – viz část 3.9 (strana 30)
languages	seznam použitelných jazyků, pro preferovaný jazyk uživatele – viz část 3.4 (strana 20)
reservations	obsahuje seznam rezervaci v lokalitě – viz část 3.8 (strana 27)
reservation_users	obsahuje uzivatele, kteří spolupracují na nějaké rezervaci s tvůrcem rezervace
reserved_devices	obsahuje výsledek mapování logických prvků na fyzické pro jednotlivé rezervace
task_categories	obsahuje definice jednotlivých kategorií pro třídy, podle kterých se kategorizují úlohy
task_classes	obsahuje definice tříd kategorií
tasks	obsahuje seznam úloh v systému – viz část 3.7 (strana 26)
tasks_categories	vazební tabulka mezi `task_categories` a `tasks`
users	obsahuje seznam uživatelů

Tabulka 2: Popis tabulek v databázi

3.3.1 Integritní omezení (kontrola cizích klíčů)

V návrhu databáze pro potřeby Virtlabu je použito techniky kontroly cizích klíčů. Cizí klíč je sloupec databázové tabulky, na který se odkazuje jiný sloupec jiné tabulky, nebo i stejné tabulky. Množina hodnot odkazujícího se sloupce musí být podmnožinou hodnot sloupce, který je klíčem. Vytváří se tak reference – odkaz. Podmínka shody se kontroluje při všech operacích nad databází. Pokud dojde ke změně hodnoty, na kterou je odkazo-

váno, musí dojít i ke změně odkazující se buňky tabulky[27]. Všechny tabulky jsou tímto způsobem provázány, jak je ukázáno na obrázku 5 (strana 19). V části E (strana 64) jsou popsány další podrobnosti, týkající se databázových tabulek.



Obrázek 5: Vazby klíčů s tabulkách

[27]Existují dva typy změny hodnoty: při přejmenování a při smazání – označováno jako **ON UPDATE** a **ON DELETE**. Událostí, které se mají s odkazující se buňkou stát při změně, je několik a mohou se lišit u různých databázových serveru – hlavní jsou: provést změnu taky (**CASCADE**), odstranit hodnotu (**SET NULL**), nedělat nic (**NO ACTION**).

3.4 Multijazyčnost

S přechodem Virtlabu na novou distribuovanou koncepci, kde mohou být jednotlivé lokality od sebe geograficky i velmi vzdálené, bylo potřeba zajistit multijazyčnost webové aplikace.

Multijazyčnost Virtlabu není zahrnuta v zadání této diplomové práce, nicméně i tak byla do nové webové aplikace zapracována. Jazykově závislé jsou všechny texty objevující se v GUI – popisy tlačítek, hlavičky tabulek, položky menu, názvy stránek, chybové hlášení javaskriptů, ...

Multijazyčnost webové aplikace zajišťuje třída `virtlabLanguage`, která je popsaná v příloze C.3 (strana 50). Její fungování je velice triviální: na základě jednoznačného identifikátoru slova nebo fráze je vrácen textový řetězec v jazyce definovaném při konstrukci objektu této třídy. Ukázka 4 (strana 20) zobrazuje volání metody, která toto provádí.

```
$_lang = new virtlabLanguage('cze');

print($_lang->Text('page-unauth'));
//vytiskne: Na tuto stranku nemate pristup!
```

Výpis 4: Získání řetězce v požadovaném jazyce

Zjištění odpovídajícího jazykového řetězce je v současné době implementována jako vícenásobná podmínka (*switch*) – část kódu je zobrazena ve výpisu 5 (strana 20). Tato implementace má několik omezujících faktorů, které budou vyřešeny převedením jazyků do databáze – více v sekci 6.1.1 (strana 43).

```
private function CZE_text($identifier) {
    switch($identifier) {
        case 'alert-diffpass':   return 'Zadana hesla se neshoduji!';
        case 'alert-login':      return 'Musite zadat uzivatelske jmeno!';
    ...
        case 'file-update':      return 'Aktualizovat';
        case 'file-fromdisk':    return 'Soubor z disku';
        default:
            return 'Text neuveden';
    } //switch
} //function
```

Výpis 5: Zdrojový kód metody `virtlabLanguage::CZE_text`

3.5 Autentizace a autorizace uživatelů

Základní potřebou každé aplikace, kterou v jeden okamžik může využívat několik uživatelů současně, je umět od sebe tyto uživatele rozlišit a každému z nich nabízet pro něj relevantní informaci, i když ji všichni požadují na stejném místě. Pokud tento obecný popis konkretizujeme na případ webové aplikace Virtlabu, dojdeme ke zjištění, že musíme

vyřešit identifikaci uživatelů přistupujících k jedné webové stránce. Musíme rozlišovat jednotlivá „sezení–relace“ – **session**[28].

V předchozích verzích webové aplikace Virtlabu byla pro identifikaci jednotlivých sezení použita vlastní implementace. Tato koncepce byla zavržena a byla implementována podpora sezení, která je součástí jazyka PHP.

Na ukázce 6 (strana 21) je zobrazen kód jednoduché stránky, která od sebe dokáže odlišit jednotlivé uživatele. Když se připojí nový uživatel, je uložen čas tohoto vstupu. Při dalším vstupu na tuto stránku se zobrazuje už jen uložený čas.

```
session_start();

if ( !isset( $_SESSION['entering_time'] ) )
    $_SESSION['entering_time'] = time();

print( $_SESSION['entering_time'] );
```

Výpis 6: Ukázka fungování session

Hlavním příkazem ve výpisu 6 (strana 21) je `session_start()`. Tento příkaz zapne v PHP podporu sezení – musí být volán jako první příkaz ve skriptu. Tímto příkazem začnou být přístupná data v superglobální[29] proměnné `$_SESSION`.

Z výpisu ale není patrné, jak dokáže PHP jednotlivé uživatele rozlišit. Rozlišení se děje na základě jednoznačného identifikátoru, který je skriptu předáván. Tento identifikátor je výstupem z hash funkce[30], kterou si PHP vyhodnocuje. Jeho předání mezi prohlížečem uživatele a serverem je možno provádět dvěma způsoby: pomocí cookies a metodou GET[31].

Jelikož je možné, aby měl uživatel ve svém webovém prohlížeči zakázáno ukládání cookies a nechceme jej k zapnutí nutit, byl zvolen způsob předávání přes URL řádek – tedy metodou GET. Identifikátor sezení se serveru předá jako část HTTP dotazu – jako proměnná předávaná metodou GET. Celý URL řádek pak vypadá takto (je zahrnuta i proměnná `page`, která volí, jaká stránka se má zobrazit – viz strana 16):
`index.php?page=23&sid=414625eae02f6c3027f8dafd3ce8883f40fee290`.

Název proměnné, která přenáší identifikátor sezení, není zvolen náhodně – je přesně definován v konfiguračním souboru PHP[32]. Jméno proměnné musí být takto přesně defi-

[28] pod pojmem sezení chápejme anglické označení session, které se ale těžko podrobuje pravidlům české mluvnice, proto budeme používat tento překlad

[29] tímto výrazem se v PHP označují proměnné, které jsou přístupné ve všech kontextech a neměly by být parametrem jazykového konstruktu `global`. Dalšími superglobálními proměnnými jsou například `$_GET`, `$_POST`, které obsahují data přijatá skriptem příslušnými metodami protokolu HTTP.

[30] v konfiguračním souboru PHP (`php.ini`) lze nastavit, jestli se má jednat o hash funkci MD5 nebo SHA-1 (konfigurační direktiva `hash_function` – tato i všechny dále uvedené direktivy patří do kontextu `session`), a kolik bitů výsledku funkce se použije na jeden symbol (direktiva `hash_bits_per_character`) – klasicky se používají 4 bity (identifikátor pak odpovídá přepisu hodnoty v šestnáctkové soustavě), ale počet jde zvýšit až na 6 bitů.

[31] konfigurační direktiva `session.cookie`

[32] standardně se tato proměnná jmenuje `PHPSESSION` – konfigurační direktiva `session.name`

nováno, aby příslušné obslužné funkce pro sezení (jako například zmíněná `session_start`) věděly, se kterou proměnnou pracovat.

Pokud používáme předávání přes metodu GET, musíme ve skriptech zajistit, aby všechny hypertextové odkazy (samozřejmě jen ty, kde chceme zajistit funkce pro sezení) obsahovaly výše zmíněnou proměnnou s příslušným identifikátorem. Pro tyto účely je v PHP definována konstanta **`SID`**, která příslušnou část URL řádku obsahuje. Ve výpisu 7 (strana 22) je zobrazen způsob, jakým vypíšeme odkaz s identifikátorem pro sezení[33].

```
session_start();
...
print('<a href="index.php?page=23&' . SID . '">odkaz</a>');
```

Výpis 7: Vytváření odkazů s identifikátorem sezení

Výše zmiňovaný postup i varianta s použitím cookies řeší předání identifikátoru sezení. Žádným z těchto způsobů se nepřenáší samotná data–proměnné jednotlivých sezení. Data proměnných v sezení jsou uložena v dočasných souborech na serveru a nedochází tak k jejich přenosu přes síť, což by v případě nezabezpečeného spojení mohlo být bezpečnostní riziko.

3.5.1 Autentizace

Z předchozí verze webové aplikace systému Virtlab byl převzat způsob autentizace uživatelů. V současné chvíli[34] existují dva způsoby, jak ověřovat uživatele – proti lokální databázi Virtlabu, nebo proti LDAP serveru školy.

Databáze obsahuje tabulku `users`, která ve sloupci `passwd` ukládá zahashované, uživatelem nastavené heslo. Jako hashovací algoritmus je v současné době použit SHA-1. Změna algoritmu pro nashování hesel je triviální, jelikož název této funkce je uložen v proměnné `$db_hash`, která se nachází v hlavním konfiguračním souboru `settings.php` a další operace s hesly s touto skutečností počítají. Aby tato skutečnost měla nějaký funkční význam, je třeba využít vlastnosti PHP, kdy lze proměnnou s uloženým názvem funkce použít jako funkci samotnou. Příklad techniky „funkce v proměnné" je ve výpisu 8 (strana 22).

```
print( sin(1.23) );       //vytiskne 0.94248
$funkce = "sin";
print( $funkce(1.23) );  //vytiskne 0.94248
```

Výpis 8: PHP vlastnost – funkce v proměnné

[33]pro proměnnou `page` použijeme ve výpisu stejnou hodnotu, jaká byla použita výše v ukázce vzhledu URL řádky

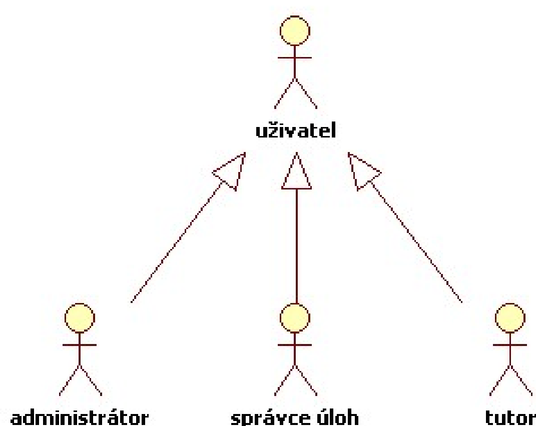
[34]přidat další způsob autentizace (např.: proti RADIUS serveru) obnáší pouze vytvoření příslušné PHP funkce, které realizuje spojení s příslušnou ověřující entitou a vrací informace, zda-li se uživatelské jméno a heslo shodují nebo ne, a dále určení řetězce, který bude v databázi identifikovat tento způsob ověřování

Aby bylo možné rozlišit jednotlivé způsoby autentizace uživatelů, byl zvolen stejný způsob jako v předchozí verzi webové aplikace. Do sloupce tabulky, kam se za normálních okolností ukládá hash hesla, je uložen řetězec identifikující způsob autentizace. který ale za žádných okolností nesmí odpovídat žádné hash-hodnotě. Pro implementovanou autentizace přes LDAP server je tento řetězec `--LDAP--`.

3.5.2 Autorizace

Jako většina webových aplikací, také Virtlab rozlišuje několik typů uživatelů, kteří mají přístup k různým částem aplikace – webové stránky. Na obrázku 6 (strana 23) je ukázáno, že kromě běžného uživatele systém rozlišuje ještě administrátora, správce úloh a tutora.

Jedině uživatel, který má nastavena práva administrátora, má přístup na všechny stránky aplikace. Ostatní typy uživatelů mají přístup omezen.



Obrázek 6: Typy uživatelů identifikovaných aplikací

Podle nastavení těchto přístupových práv se jednotlivým uživatelům generuje i rozdílné menu, které je zobrazeno v levé části stránky. Menu se liší jen v množství položek v nich zobrazených – ve výpisu 9 (strana 24) je zobrazena část skriptu, která generuje uživatelské menu. Ve výpisu objevíme i příkazy pro vypsání textu v nastaveném jazyce – viz část 3.4 (strana 20).

Funkce, která zajišťuje autorizaci uživatele pro jednotlivé položky menu, je `Authorization`. Tato funkce je metodou třídy `virtlabWeb` – podrobný popis této třídy je v příloze C.1 (strana 49). Část této funkce je zobrazena ve výpisu 10 (strana 24), kde je vidět jednoduchost implementace. Funkce na základě vstupního identifikátoru a práv přihlášeného uživatele[35] vrací logickou hodnotu, jestli má být „vstup“ povolen, nebo má být zamítnut.

[35]práva jsou uložena v proměnné `$_SESSION`, která je pro každého uživatele samostatná

Výše zmíněnými postupy je zabráněno, aby se v menu zobrazovaly odkazy na stránky, kam není uživatel oprávněn vstupovat. Nic ale nebrání uživateli, aby si sám upravil URL řádek prohlížeče a vyžádal si libovolnou stránku[36]. Pro zamezení tomuto typu neoprávněného vstupu obsahuje každý skript stránky na svém začátku kód, který je zobrazen ve výpisu 11 (strana 25). Každému uživateli, který není přihlášen, nebo nemá na stránku přístup (ve výpisu jde o stránku s přístupovým identifikátorem `page-reser-mine`), je zobrazeno varování, že na stránku nemá přístup a provádění skriptu je ukončeno.

```
...
<div class="menu_title"><?php print($_lang->Text("word-main")); ?></div>
<ul>
    <li><a href="index.php?page=0&<?php echo SID; ?>">
        <?php print($_lang->Text("menu-main")); ?></a></li>
...
<?php
    if ( $all->Authorization("menu-users")) {
        print("<div class=\"menu_title\">" .$_lang->Text("word-users") . "</div>\n");
        print("<ul>\n");
        if ( $all->Authorization("menu-users-edit"))
            print("<li><a href=\"index.php?page=4&" .SID. "\">" .$_lang->Text("menu-edit"). "</a
                ></li>\n");
    ...
        print("</ul>\n");
    } // menu-users
...
```

Výpis 9: Autorizace položek menu

```
public function Authorization($page_id) {
    switch($page_id) {
    ...
        case 'menu-tasks-list':
        case 'page-tasks-list':
            return TRUE; //prava pro vsechny
    ...
        case 'menu-tasks-import' :
        case 'page-tasks-import' :
            if ($_SESSION['rights']['admin'] || $_SESSION['rights']['task']) return TRUE;
            else return FALSE; //jen pro administratory a spravce uloh
    ...
        case 'menu-tasks-delete':
        case 'page-tasks-delete':
            if ($_SESSION['rights']['admin']) return TRUE;
            else return FALSE; //jen pro administratory
    ...
```

Výpis 10: Autorizace položek menu

[36] ať už hodnotu argumentu `page` jen uhádl, nebo ji zahlédl při práci správce a zapamatoval si ji

```
// pristup k globalnim objektum trid virtlabWeb a virtlabLanguage
global $all, $_lang;

if (!$_SESSION["logged"] || !$all->Authorization("page-reser-mine")) {
    print("<div class=\"headline error\">" . $_lang->Text("page-unauth") . "</div>\n");
    return;
} // if-unauthorized
```

Výpis 11: Autorizace každé ze stránek

3.6 Správa uživatelů

Ve webové aplikaci Virtlabu je implementována základní správa uživatelů, která umožňuje uživatele vytvářet, mazat a editovat. Uživatelé jsou uloženi v databázi a pouze oni se mohou k webové aplikaci přihlásit. Možnosti autentizace uživatele jsou popsány v části 3.5.1 (strana 22).

U jednotlivých uživatelů jsou evidovány tyto údaje:

přihlašovací jméno slouží jako jednoznačný identifikátor uživatele. Tímto jménem se uživatel přihlašuje k webové aplikaci.

křestní jméno uživatele – má jen informační charakter

příjmení uživatele – má jen informační charakter

e-mail uživatele – zatím má jen informační charakter, ale v budoucnu na něj budou uživateli posílány informace o změně jeho rezervací – viz část 6.1.2 (strana 44)

preferovaný jazyk uživatele – určuje jazyk webového rozhraní aplikace pro daného uživatele

heslo – viz část 3.5.1 (strana 22)

uživatelská skupina určuje příslušnost uživatele k nějaké skupině – zatím má jen informační charakter, ale v budoucnu bude implementována správa uživatelských skupin – viz část 6.1.2 (strana 44)

kvóta určuje množství času, které může uživatel použít pro rezervace – viz část 3.8.2 (strana 29)

platnost vymezuje dobu, do které je uživatelský účet platný – zatím má jen informační charakter, ale v budoucnu bude implementována podpora pro tento údaj – viz část 6.1.2 (strana 44)

práva: administrátor určuje úroveň uživatelských práv – viz část 3.5.2 (strana 23)

práva: správce úloh určuje úroveň uživatelských práv – viz část 3.5.2 (strana 23)

práva: tutor určuje úroveň uživatelských práv – viz část 3.5.2 (strana 23)

3.7 Úlohy

Úlohy patří k základním částem Virtlabu. V prvních verzích Virtlabu úlohy obsahovaly popis propojení prvků a popis úlohy, kterou si má student vyzkoušet.

V současné implementaci se pojetí definice úloh lehce pozměnilo – bylo rozšířeno v souvislosti s novým rozšířením webové aplikace o práci se soubory, popsaným v části 3.9 (strana 30), a rozšíření o skupiny souborů, popsaným v části 3.10 (strana 31).

Definice úlohy nyní obsahují tyto informace:

krátký název slouží jako zkratkové označení úlohy a je používáno při výpisech (příklad: OSPF-u1, RIPv2-u5, ...)

dlouhý název slouží jako podrobnější název úlohy (příklad: směrování protokolem RIP verze 2)

stručný popis má v jednoduchosti nástínit, co si má uživatel v úloze vyzkoušet

časová náročnost informuje uživatele, kolik času ja na úlohu potřeba – má pouze informační charakter

skupina souborů: zadání má uživateli posloužit jako podrobný popis a návod k úloze

skupina souborů: obrázky mají ilustrovat logické topologie použité v úloze, nebo jinak napomoci uživateli. Jde o normální skupinu souboru, jejíž jedinou odlišností je, že se je prohlížeč bude snažít zobrazit jako obrázky (tedy pomocí značky `img`).

skupina souborů: předkonfigurace obsahuje konfiguraci síťových prvků, která v nich bude nastavena na začátku úlohy – tato vlastnost není v současnosti implementována

skupina souborů: cílová konfigurace má uživateli posloužit jako ukázka konfigurace jednotlivých prvků, které měl dosáhnout na konci úlohy

skupina souborů: logická topologie obsahuje skupinu logických topologií, které může uživatel pro úlohu použít

Úlohy mohou být vytvořeny prostřednicvím webového rozhraní, nebo mohou být importovány z archivu – tato možnost se v minulosti osvědčila, jako velmi užitečná pro účely offline přípravy úloh, a proto byla implementována znova. Blíže o importu úlohy pojednává následující část.

3.7.1 Import úlohy z archivu

Ačkoliv jde úlohy vytvářet pomocí webového rozhraní Virtlabu, možnost úlohu přímo importovat v jednom kroku se ukázala jako velice důležitá. Tvůrce úloh nemusí být neustále k aplikaci připojen a může úlohy vytvářet „offline“, a pak je nahrát všechny najednou.

Předchozí implementace pracovala s importovaným archivem na úrovni souborového systému – soubory byly ukládány do adresářů, k rozbalení archivu se využívaly programy instalované v operačním systému. Změna koncepce celého Virtlabu, jak je popsána v části 2 (strana 3), a ukládání souborů v databázi (část 3.9 na straně 30) si vyžádalo kompletní přepracování celého mechanismu importu.

Jelikož už nebylo potřeba pracovat na úrovni souborového systému (kvůli implementace ukládání souborů do databáze), bylo potřeba najít nástroj na práci s archivy v PHP. Jako ideální se ukázala třída `Archive_Tar`, která pochází z „knihovny“ PEAR[37]. Tato třída má implementovanou základní manipulaci[38] s archivy TAR – a to bez komprese, nebo s použitím komprese GZip a BZ2[39].

Stejně jako v předchozí verzi webové aplikace, i v tomto případě bylo potřeba popsat obsah importovaného archivu, aby se aplikaci usnadnila práce s ním. Byl zvolen XML formát souboru, který je snadno zpracovatelný aplikací. Pro extrakci údajů potřebných pro import byla vyrobena třída `virtlabParserTaskupload`, která je popsaná v části C.4 (strana 50). Ve výpisu 12 (strana 27) je zobrazena část skriptu starající se o import úlohy.

```
    // trida potrebna pro praci s archivy TAR
    require_once("Archive/Tar.php");

    switch($_POST['type']) {    //na zaklade formulare se vybere typ archivu
        case 'tar' : $archive = new Archive_Tar($_FILES['file']['tmp_name']); break;
        case 'tgz' : $archive = new Archive_Tar($_FILES['file']['tmp_name'],'gz'); break;
        case 'bz2': $archive = new Archive_Tar($_FILES['file']['tmp_name'],'bz2'); break;
        default: return;
    } //switch
...
    //extrakce XML souboru popisujiciho archiv
    $xml = $archive->extractInString("task.xml");
...
    //vytvoreni objektu pro extrakci dat z XML
    $taskparser = new virtlabParserTaskupload($nowsxml, 0);
...
```

Výpis 12: Import úlohy

3.8 Rezervace

Rezervace jsou asi nejdůležitější částí celého systému Virtlab. Realizují vzdálený přístup uživatelů na konzoly síťových prvků. Kvůli přechodu na novou koncepci, která je popsaná v části 2 (strana 3), musely právě rezervace projít nesložitějším přepracováním.

[37]PEAR je uložiště funkcí, tříd a aplikací napsaných v jazyce PHP – viz http://pear.php.net

[38]podporuje vytváření, procházení, extrakci a přidávání souborů do archívu

[39]pro obě varianty je potřeba, aby v PHP byly nainstalovány příslušná rozšíření, která tyto komprese implementují. Bývají součástí standardní instalace.

Aby bylo možno vyřešit systém rezervací mezi lokalitami v nové distribuované koncepci, musel být vyroben **rezervační server**[40].

Rezervační servery komunikují jednak mezi sebou, a jednak s webovou aplikací ve své lokalitě. Druhý uvedený způsob komunikace bude popsán v následující části 3.8.1 (strana 28).

Webová aplikace, kromě komunikace s rezervačním serverem své lokality, obsahuje také správu již existujících rezervací. V souvislosti s tím musel být implementován nový systém uživatelských kvót, který je popsán v 3.8.2 (strana 29).

Ve webové aplikaci jsou evidovány jen ty rezervace, které byly vytvořeny uživateli příslušné lokality. Rezervace se dají vytvořit na základě logické topologie uvedené u nějaké úlohy, nebo jako „topologie na přání", kdy je uživatelem přímo nahrán soubor s logickou topologií. S každou rezervací je evidováno mnoho dalších informací:

identifikátor rezervace je jedinečný v rámci lokality a je potřebný pro komunikaci s rezervačním serverem

tvůrce rezervace je jediný uživatel, který může rezervaci zrušit a přihlašovat ke své rezervaci další kolegy z řad uživatelů

trvání (od – do) určuje čas, ve kterém je rezervace pro tvůrce a jeho kolegy přístupná. V tomto čase mohou vzdáleně přistupovat k síťovým prvkům. Prvních 5 minut od počátku rezervace jsou ještě prvky nedostupné – tento čas je vyhrazen pro realizování topologie.

úloha , ke které se rezervace váže (databázový odkaz na ní). Pokud ale jde o rezervaci, která vznikla jako „topologie na přání", tak tato informace není uvedena.

topologie, ke které se rezervace váže (databázový odkaz na ní). Pokud ale jde o rezervaci, která vznikla jako „topologie na přání", tak tato informace není uvedena.

kolegové jsou uživatelé, kteří mohou na úloze v době rezervace spolupracovat. Ti mohou z rezervace odstoupit, ale zpět je může vrátit jen tvůrce rezervace.

výsledek mapování eviduje, který vrchol logické topologie byl namapována na jaké fyzické zařízení – viz část 5 (strana 35)

3.8.1 Komunikace s rezervačním serverem

Komunikační protokol mezi webovou aplikací a rezervačním serverem je implementovám v metodách třídy `virtlabReservations`, která je popsaná v příloze C.6 (strana 52).

Kompletní vytvoření rezervace probíhá v několika krocích a vyžaduje několik výměn zpráv s rezervačním serverem. Průběh komunikace s rezervačním serverem při tvorbě rezervace je následující:

[40] jeho popis a implementace je součástí diplomové práce Tomáše Hrabálka [3]

1. je vyžádán seznam zařízení dostupných v čase, který uživatel uvedl – rezervační server si podle své konfigurace případně vyžádá i zařízení z jiných lokalit
2. na seznam zařízení se mapovací algoritmus, který je popsán v části 5 (strana 35), pokusí namapovat zadanou logickou topologii
3. mapováním vzniklý seznam potřebných fyzických zařízení je u rezervačního serveru zarezervován – pokud jsou některá zařízení z jiných lokalit, pokusí se rezervační server tyto prvky rezervovat u rezervačních serverů příslušných lokalit
4. rezervačnímu serveru jsou předána data vzniklá mapovacím algoritmem, která popisují, jak mají být jednotlivá zařízení propojena
5. pokud každý z předešlých kroků proběhl v pořádku, tak je rezervace zaevidována a v příslušný čas bude uživateli k dispozici

3.8.2 Systém kvót

Pro zajištění rovného přístupu k Virtlabu pro všechny uživatele bylo potřeba navrhnout systém kvót. Ten by měl ohlídat jednotlivé uživatele, aby nemohli celý systém obsadit jen sami pro sebe. Kvótou rozumíme množství času, které může uživatel použít v definovaném časovém období.

V předchozí verzi Virtlabu měl uživatel přidělen určitý počet timeslotů (elementárních časových úseků), které mohl vypotřebovat na úlohy během jednoho kalendářního týdne, ať si je rezervoval na libovolný týden v budoucnosti. V novém týdnu mu byla kvóta opět navýšena na určenou hodnotu.

Uživatelům jsme chtěli umožnit rezervovat si úlohu na libovolnou dobu – i několik měsíců dopředu. Pokud bychom výše popsaný systém aplikovali teď, znamenalo by to identifikovat, do kterého kalendářního týdne (nebo jiného období) konkrétní rezervace spadá, jestli nezasahuje i do týdne vedlejšího. Pokud si uvědomíme všechny detaily tohoto řešení, zjistíme, že by implementace byla poměrně komplikovaná. Proto byl vymyšlen systém nový.

Systém klouzavého okna

V systému je definován spojitý časový úsek (označme ho jako A), který systém prohledává. Dále je definováno množství času, které uživatel může použít na rezervace úloh (označme ho jako B). Aby celý systém mohl fungovat, je potřeba, aby $A > B$. V reálném provozu si můžeme pod těmito proměnnými představit hodnoty, jako $A = 1$ týden (chápan jako 7 dní, 168 hodin, ...) a $B = 10$h.

Během rezervace úlohy na konkrétní čas dochází ke kontrole, zda-li uživatel může rezervaci provést, nebo už má množství času v „A-okolí“ zamýšlené rezervace vyčerpáno. Kontrola probíhá následovně:

- čas začátku ($Start$) i konce ($Stop$) rezervace se převede na Unix timestamp[41]. Tato veličina je definována jako počet sekund od 1.1.1970 0:00. Jedná se vlastně o prostor přirozených čísel, který je omezen pouze konečnou pamětí pro uložení tohoto čísla. Proto operace s časem (např.: doba mezi dvěma daty) lze provádět triviálními aritmetickými operacemi.

- najdeme datum, které je uprostřed rezervace:

$$Middle = \frac{(Stop - Start)}{2} + Start$$

- budeme zkoumat dobu od

$$Middle - \frac{A}{2}$$

  do

$$Middle + \frac{A}{2}$$

  kolik času má uživatel v tomto intervalu rezervováno a jestli již nepřekročil množství času B. Pokud jej překročil, tak mu nebude dovoleno vytvořit požadovanou rezervaci.

Výhodou tohoto systému je zabránění uživateli ve spoření času na nějaké vzdálené období, kde by uživatel mohl obsadit najednou velký časový prostor. Odpadají problémy s operacemi s klasickým kalendářem (přestupné roky, různý počet dnů v měsíci, ...). Hodnoty A a B může mít definován každý uživatel zvlášť. Tímto způsobem můžeme některým uživatelům zvýšit množství času (pokud si to nějakým způsobem zaslouží), nebo naopak snížit.

3.9 Práce se soubory

Již v části 3.7.1 (strana 26) bylo nastíněno, že nová verze webové aplikace Virtlabu nepracuje se soubory na úrovni souborového systému, ale ukládá si veškeré soubory do databáze. Tento přístup sebou nese několik výhod i nevýhod.

Výhody:

- jednoduchost zálohování celého systému – ten se skládá „jen" ze skriptů webového rozhraní a databáze (skripty jsou pevné)

- přístup k souborům lze zajistit všem systémům (běžících na jiných strojích v jiných částech světa), které umí komunikovat s databázovým serverem

- jednoduchost zabezpečení přístupu k souborům – přístup k jednotlivým souborům je totožný se zabezpečením přístupu na jednotlivé stránky systému

- uživateli se neodhaluje žádná adresářová struktura – vyšší bezpečnost

[41] viz http://en.wikipedia.org/wiki/Unix_time a http://www.unixtimestamp.com

Nevýhody:

- náročnější implementace manipulačních funkcí – funkce pro práci se souborovým systémem jsou součástí PHP
- větší nároky na databázový stroj

Manipulaci se soubory v databázi zajišťuje třída `virtlabWebFile`, která je popsaná v části C.7 (strana 53).

Drobnou obtíží při práci se soubory v databázi je jejich zobrazování (pokud jde o obrázky) a stahování. Pro tyto operace je potřeba vyrobit speciální skript, který bude měnit hlavičku HTTP zprávy, jdoucí ze serveru, aby byl správně uveden MIME typ a případně jméno souboru.

V ukázce 13 (strana 31) je zobrazená část skriptu, který toto provádí. Tím nejdůležitějším v ní je funkce `Header`, která ovlivňuje právě hlavičku HTTP zprávy. Její první dvě použití v ukázce nastavují MIME typ dat a jejich délku v bajtech. Pokud je v HTTP hlavičce uvedená část `Content-Disposition:...`, prohlížeč při jejím načtení automaticky nabídne uživateli uložení dat do souboru s názvem, který je u tohoto atributu uveden dále.

```
...
$file = new virtlabWebFile($sql_server, $sql_user, $sql_pass, $sql_db, $file_table);
$type = "";
$size = 0;
$name = "";
$data = $file->Get($_GET["file"], $type, $size, $name);

Header("Content-Type: " .$type);
Header("Content-Length: $size");
if ($_GET["save"]) Header('Content-Disposition: attachment; filename="' .$name. '"');

print($data);
...
```

Výpis 13: Skript pro stáhnutí souboru z databázi

Ve webové aplikace ji implementována základní správa souboru: procházení jejich seznamem, zobrazení, stažení, vymazání, ...

3.10 Souborové skupiny

Po implementaci souborů v databázi se ukázalo jako nezbytné definovat skupiny souborů. Skupiny totiž řeší dekompozici vazby **M:N**, která vzniká mezi soubory a jejich použitím v úlohách. V jednotlivých položkách úlohy[42] je třeba evidovat obecně **N** souborů, což nejde v přímo v databázi realizovat, proto musely byt implementovány skupiny souborů. Ve webové aplikaci je implementovaná základní správa těchto souborových skupin: vytvoření, vymazání.

[42] viz 3.7 (strana 26)

4 Konfigurace Virtlabu a její zpracování

V několika místech systému Virtlab bylo zapotřebí načítat strukturovaná data, která se dále zpracovávají. Pro tyto data byl zvolen formát XML. Tento formát je dobře zpracovatelný programově a lehce rozšířitelný.

Systém využívá tento formát na třech místech:

popis fyzických prvků je jednou ze dvou nejdůležitejších částí nové systému Virtlab (co se dat týká). Nad těmito daty se provádí mapování[43] logických prvků topologie na prvky fyzické. Tato data jsou webovou aplikací získána od rezervačního serveru[44].

popis logické topologie je tou druhou důležitou částí. Je také druhým vstupním údajem pro algoritmus mapování.

popis archivu pro import úlohy je nezbytný, aby webová aplikace mohla správně zapsat data o úloze do databáze.

4.1 Načtení dat s použitím SAX

V PHP jsou implemenentovány dva základní způsoby, kterými se načítají informace z dat ve formátu XML: SAX a DOM. První z nich (SAX) představuje postupné sériové zpracování XML, kdy jsou pro jeho jednotlivé části (otevírací značka s atributy, data značky, uzavírací značka) volány uživatelem definované funkce. Druhý způsob (DOM) převede celé XML na strom objektů, které jsou vzájemně propojeny stejným způsobem, jako jsou v XML do sebe vnořovány jednotlivé značky. Každý ze způsobů má svoje pro a proti. Nicméně pro potřeby webové aplikace se ukázal jako nejvhodnější SAX[45].

```
function __construct() {
    $this->xml_obj = xml_parser_create();
    xml_set_object($this->xml_obj,$this);
    xml_set_character_data_handler($this->xml_obj, 'dataHandler');
    xml_set_element_handler($this->xml_obj, 'startHandler', 'endHandler');
}

private function startHandler($parser, $name, $attribs) { ... }
private function dataHandler($parser, $data) { ... }
private function endHandler($parser, $name) { ... }
```

Výpis 14: Vytvoreni SAX parseru

[43]viz část 5 (strana 35)

[44]viz část 3.8.1 (strana 28)

[45]V době, kdy začala implementace zpracování XML ve Virtlabu, se používalo PHP verze 4, které mělo DOM implementováno jako nestandardní rozšíření. SAX je implementován přímo v PHP již velmi dlouhou dobu a jde tedy o ověřené řešení. Až později bylo rozhodnuto o přechodu na PHP verze 5, který DOM již používá podle standardů. Nicméně SAX parser byl již implementován a byly vytvořeny „parsery druhé úrovně" – viz část 4.3 (strana 33).

Na výpisu 14 (strana 32) je zobrazeno vytvoření SAX parseru. Poslední dva řádky konstruktoru v ukázce nastavují parser, aby používal pro zpracování dat funkce: `dataHandler` pro data značky, `startHandler` pro otevírací značku a `endHandler` pro uzavírací značku.

Parser SAX sám o sobě nepřevádí XML data na nějaké dále zpracovatelné informace – jen dovoluje programátorovi, aby si nadefinoval ony tři obslužné funkce a sám si tak vymyslel, jakým způsobem s daty naloží.

Pro použití ve webové aplikaci se ukázalo z hlediska dalšího zpracování jako nejjednodušší vytvořit z XML jedno pole, které odpovídá strukturou stromu XML. Stručný nástin způsobu, jak je takovéto pole vytvořeno, je v popisu třídy `virtlabXmlParser` (která zajišťuje zpracování XML dat) v příloze C.8 (strana 54). Výpis 22 (strana 55) zobrazuje jednoduchý XML soubor, který po zpracování odpovídá poli zobrazenému ve výpisu 23 (strana 56).

4.2 Validace XML

Správným načtením dat z XML máme zaručeno jen to, že jsou data **syntakticky** správná. Pokud ale informace dále zpracováváme, je potřeba mít zaručeno, že jsou XML data **validní** – tzn.: XML je vytvořen způsobem, jaký potřebujeme ,a určili jsme to pomocí jiného souboru – DTD.

DTD definují způsob, kterým můžeme určit, jaká data budou v XML uložena – jakou budou mít strukturu a jaké můžou mít jednotlivé elementy hodnotu. Samotná validace XML musí proběhnout před vlastním zpracováním parserem[46].

Samotné validování dat je zobrazeno ve výpisu 15 (strana 33). Odkaz na samotný DTD dokument, proti kterému se XML ověřuje, musí byt uveden v hlavičce XML[47] – viz výpis 16 (strana 34). Jednotlivá DTD používaná webovou aplikací jsou zobrazena ve výpisech 21 (strana 48), 18 (strana 38) a 17 (strana 37).

```
$dom = new DOMDocument;
$dom->load($path);
if (!( $dom->validate()))
    die($path . ' document is invalid !\n');
print('document is VALID');
```

Výpis 15: Validace XML

4.3 Získání informací z načtených dat

V části 4.1 (strana 32) je popsán způsob, jakým se ze „surových“ XML dat vyrobí mnohorozměrné pole, kde jsou jednotlivé informace zapsány – příklad pole je na výpis 23 (strana 56).

[46]pokud nejsou data ve formátu, jaký požadujeme, tak proč se s nimi vůbec zabývat

[47]V PHP je možno validovat jen pomocí DTD uvedeného v hlavičce XML dokumentu. Nejde explicitně určit DTD soubor, proti kterému validace probíhá.

```
<?xml version="1.0" encoding="utf−8" ?>
<!DOCTYPE virtual_topology SYSTEM "topology.dtd">

<virtual_topology>
 ...
</virtual_topology>
```

Výpis 16: Hlavička XML

Takovéto pole může být značně komplikované a může dosahovat značného množství vnoření. Používat ve všech funkcích, které potřebují data uložená v XML, tyto dlouhé identifikátory[48] by bylo značně komplikované a v případě úpravy způsobu získávání dat z XML by muselo všude dojít k jejich přepsání. Z těchto důvodů byly navrženy „parsery druhé úrovně“, které poskytují přístup k datům, uložených v poli získaném parserem XML.

Jednotlivé třídy, které tyto parsery obsahují, jsou popsány v částech: C.4 (strana 50), C.9 (strana 55) a C.10 (strana 58).

[48] po přidání všech indexů

5 Mapovací algoritmus

Mapovací algoritmus je po přepracování webové aplikace druhou inovací v systému Virtlab, která byla určena v rámci této diplomové práce. Realizuje propojení logického popisu topologie úlohy s fyzickými prvky a jejich rozhraními. Kromě abstrakce topologie na logickou úroveň přináší dále možnost paralelního běhu několika úloh, pokud to dovolují požadované parametry a množství dostupných fyzických zařízení.

Algoritmus musí umět najít vhodnou podmnožinu fyzických zařízení ze všech dostupných, která splňuje všechny požadavky kladené popisem logické topologie. Kladené požadavky mají komplexní charakter – může jít o počet rozhraní zařízení, verzi operačního systému, ... – viz část 5.2 (strana 36).

Aby mohlo být vůbec nějaké mapování realizováno, je potřeba kromě popisu logické topologie i popis fyzických prvků, které mohou být potenciálně použity. Popis vybavení musí obsahovat údaje, které samozřejmě odrážejí možné požadavky kladené logickou topologií – viz část 5.1 (strana 35).

Mapování je spuštěno vždy při tvorbě rezervace ve webové aplikaci. Data popisující logickou topologii jsou získána z databáze (pokud jde o běžnou rezervaci úlohy), nebo přímo od uživatele (pokud je o „topologii na přání“). Data o fyzických prvcích jsou získána od rezervačního serveru, který eviduje obsazenost zařízení – viz část 3.8.1 (strana 28).

5.1 Popis vybavení

Popis vybavení je označení pro strukturovaný popis vlastností fyzických prvků, který slouží jako jeden ze vstupů pro mapovací algoritmus. Jde o soupis vlastností a „schopností“ jednotlivých fyzických prvků. Informace jsou strukturovany ve formátu XML – zpracování toho formátu ve webové aplikaci Virtlabu je věnována část 4 (strana 32).

Dokument v XML popisující vybavení je validován proti DTD, které je uvedeno ve výpisu 17 (strana 37). Tímto DTD je ověřena syntaktická správnost XML dokumentu popisujícího vybavení. Význam jednotlivých značek a atributů je tento[49]:

equipment je kořenový element celého dokumentu

device je značka reprezentující jeden fyzický síťový prvek

device(type) tento atribut určuje typ zařízení – v současné době jsou implementovány typy `router` a `switch`

device(name) tento atribut určuje název zařízení. Jméno je ve formátu `název@lokalita` – viz 2 (strana 3). Jméno zařízení není důležité jen pro mapovácí algoritmus, ale objevuje se i v jiných částech systému – ve webové aplikaci, rezervačním serveru, ...

device(serial_number) seriové číslo v současné době není funkčně využíváno. Má informační charakter a mělo by sloužit pro evidenci vybavení.

[49] V seznamu jsou vypsány jednotlivé značky dokumentu. Popisované atributy jsou napsány v závorce u jména značky, ke které patří.

device(platform) tento atribut určuje platformu – produktovou řadu zařízení

os verze operačního systému (případně firmwaru) zařízení

interfaces tento element je „zastřešujícím“ pro všechny elementy `interface`

interface tento element reprezentuje jedno síťové rozhraní zařízení

interface(technology) určuje technologii síťového rozhraní – v současné době jsou implementovány typy `serial` a `ethernet`

interface(ether_type) určuje druh ethernetu (pokud je technologie rozhraní `serial`, tak nemusí být zadán) – v současné době jsou implementovány typy `legacy`, `fast` a `gigabit`

interface(connect_group) je identifikátor „skupiny propojitelnosti“. Jen rozhraní se stejným identifikátorem mohou být propojena. Tento atribut má zamezit mapovacímu algoritmu v projení dvojice rozhraní, která fyzicky nemohou být propojena z důvodu vnitřní implementace virtuálního spojovacího pole – nejde ovšem o problém propojit seriové rozhraní s ethernetový rozhraním (to je vyřešeno atributem `technology`).

interface(name) určuje název rozhraní, jak je uvedeno v operačním systému zařízení (např.: fa0/1/2, s0, . . .)

max_bps udává maximální možnou rychlost seriového rozhraní (rychlost je uvedena v bitech za sekundu) – pokud je tento element uveden u rozhraní technologie `ethernet`, je hodnota ignorována

int_feature tímto elementem se vyjmenují vlastnosti rozhraní, které jsou nestandardní pro tento typ rozhraní

special tento element je „zastřešujícím“ pro všechny elementy `feature`

feature tímto elementem se vyjmenují vlastnosti, které nejsou u tohoto typu zařízení běžné (např.: MPLS, VoIP, . . .)

5.2 Popis logické topologie

Popis logické topologie je označení pro strukturovaný popis topologie logických zařízení, která budou mapovacím algoritmem namapovány na zařízení fyzické, tak, aby byly splněny všechny požadavky touto topologií kladené. Informace jsou opět strukturovany ve formátu XML – zpracování toho formátu ve webové aplikaci Virtlabu je věnována část 4 (strana 32).

```
<!ELEMENT equipment (device*)>
<!ELEMENT device      (os, interfaces ?, special ?)>
<!ELEMENT os          (#PCDATA)>
<!ELEMENT interfaces  ( interface +)>
<!ELEMENT interface   (max_bps?, int_feature*)>
<!ELEMENT max_bps     (#PCDATA)>
<!ELEMENT int_feature (#PCDATA)>
<!ELEMENT special     ( feature+)>
<!ELEMENT feature     (#PCDATA)>

<!ATTLIST device
          type           (router | switch)           #REQUIRED
          name           CDATA                       #REQUIRED
          serial_number  NMTOKEN                     #REQUIRED
          platform       CDATA                       #REQUIRED>
<!ATTLIST interface
          technology     ( serial | ethernet)        #REQUIRED
          ether_type     (legacy | fast | gigabit)   #IMPLIED
          connect_group  NMTOKEN                     #REQUIRED
          name           CDATA                       #REQUIRED>
```

Výpis 17: DTD dokumentu pro popis vybavení

XML dokument popisující logickou topologii je validován proti DTD, které je uvedeno ve výpisu 18 (strana 38). Tímto DTD je ověřena syntaktická správnost XML dokumentu popisujícího logickou topologii[50]. Význam jednotlivých značek a atributů je tento[51]:

virtual_topology je kořenový element celého XML

edge je značka popisující jednu linku, která spojuje dva prvky topologie

edge(technology) určuje technologii linky – v současné době jsou implementovány typy `serial` a `ethernet`

edge(name) určuje název linky. Tento název má sloužit pro lepší orientaci v topologii – je také využíván mapovacím algoritmem, neboť jednoznačně identifikuje linku.

edge(ether_type) určuje druh ethernetu (pokud je technologie rozhraní `serial`, tak nemusí být zadán) – v současné době jsou implementovány typy `legacy`, `fast` a `gigabit`

vertex tato značka označuje jeden vrchol logické linky-hrany v topologii

vertex(name) určuje jméno vrcholu logické topologie. Pod tímto názvem bude uživateli zpřístupněn fyzický prvek, který mapovací algoritmus vybere.

vertex_detail tato značka slouží pro popis požadavků, které jsou kladeny na vrchol logické topologie

[50]Terminologie u popisu logické topologie byla převzata z teorie grafů, proto zde mluvíme o vrcholech a hranách – tyto termíny z pohledu počítačových sítí představují síťové zařízení a linku, která je spojuje.

[51]V seznamu jsou vypsány jednotlivé značky dokumentu. Popisované atributy jsou napsány v závorce u jména značky, ke které patří.

vertex_detail(type) určuje typ vrcholu logické topologie – v současné době jsou implementovány typy `router` a `switch`

vertex_detail(name) udává jméno vrcholu logické topologie. Údaj spojuje touto značkou popisovaný vrchol s vrcholem uváděným v některých linkách (`edge`).

poss_platform tento element je „zastřešujícím" pro všechny elementy `platform`

platform tato značka představuje požadovanou platformu (produktovou řadu) fyzického prvku, který bude namapován na tento vrchol logické topologie. Hodnota této značky je považována za **regulární výraz**!

os určuje požadovanou verzi operačního systému fyzického prvku, který bude namapován na tento vrchol logické topologie. Hodnota této značky je považována za **regulární výraz**!

vertex_feature tímto elementem se vyjmenují vlastnosti, které nejsou u fyzických zařízení požadovaného typu běžné (např.: MPLS, VoIP, ...) – hodnota této značky je v popisu fyzických zařízení porovnávána se značkou `feature`

min_bps udává minimální požadovanou rychlost seriové linky (rychlost je uvedena v bitech za sekundu) – pokud je tento element uveden u linky technologie `ethernet`, je hodnota ignorována

edge_feature tímto elementem se vyjmenují vlastnosti linky, které jsou nestandardní pro tento typ – hodnota této značky je v popisu fyzických zařízení porovnávána se značkou `int_feature`

```
<!ELEMENT virtual_topology  (edge*, vertex_detail*)>
<!ELEMENT edge              (vertex, vertex, min_bps?, edge_feature*)>
<!ELEMENT vertex            EMPTY>
<!ELEMENT vertex_detail     (poss_platforms?,os?, vertex_feature*)>
<!ELEMENT poss_platforms    (platform+)>
<!ELEMENT platform          (#PCDATA)>
<!ELEMENT os                (#PCDATA)>
<!ELEMENT vertex_feature    (#PCDATA)>
<!ELEMENT min_bps           (#PCDATA)>
<!ELEMENT edge_feature      (#PCDATA)>

<!ATTLIST vertex
          name        IDREF                           #REQUIRED>
<!ATTLIST vertex_detail
          type        (router | switch)               #REQUIRED
          name        ID                              #REQUIRED>
<!ATTLIST edge
          technology  ( serial | ethernet)            #REQUIRED
          name        CDATA                           #REQUIRED
          ether_type  (legacy | fast | gigabit)       #IMPLIED>
```

Výpis 18: DTD dokumentu pro popis topologie

5.3 Popis mapovacího algoritmu

Mapovací algoritmus určuje, které fyzické zařízení[52] bude fungovat jako který logický prvek topologie[53]. Musí brát ohled na „schopnosti" fyzického zařízení a požadavky logické topologie kladené na toto zařízení – snaží se přitom vybrat fyzické zařízení, které svými schopnostmi přesahuje požadavky co nejméně[54].

V celém algoritmu je několik částí-funkcí, které se používají opakovaně, jelikož značná část algoritmu je rekurzivní. Tyto části algoritmu jsou popsány v následujících částech diplomové práce. Ony funkce jsou metody třídy `virtlabMapping`, která je popsána v příloze C.11 (strana 60).

5.3.1 Schopnost zařízení být prvkem logické topologie

Na začátku celého algoritmu je potřeba zjistit, který fyzický prvek může být kterým logickým prvkem. Toto ověření musí prozkoumat všechny požadavky, které logická topologie na fyzický prvek klade, a porovnat je se schopnostmi zařízení. Postup ověření je následující (musí být splněny všechny body postupu, aby fyzický prvek mohl být použit jako daný vrchol topologie):

1. porovnání typu zařízení s typem vrcholu logické topologie
2. porovnání platformy zařízení s požadovanou platformou uvedenou v popisu logické topologie (požadavek na platformu je považován za regulární výraz)
3. porovnání verze operačního systému s požadavkem logické topologie na verzi (požadavek na verzi operačního systému je považován za regulární výraz)
4. porovnání nestandardních vlastností fyzického prvku s požadavkem na nestandardní vlastnosti vrcholu logické topologie
5. ověření dostatečného počtu rozhraní (pokud nemá zařízení dostatek rozhraní, aby mohla pokrýt potřeby topologie, není potřeba zkoumat podrobnosti jednotlivých rozhraní – kontroluje se i typ rozhraní)
6. zjištění způsobilostí rozhraní pro všechny požadované linky (počítáno přes všechny linky, na niž vrchol v logické topologii leží, a všechna rozhraní zařízení). Je potřeba zjistit, které rozhraní může být součástí jaké linky, na které vrchol leží.
   (a) porovnání speciálních vlastností rozhraní a linky
   (b) porovnání technologie – u linek typu `serial` se porovnává maximální rychlost rozhraní fyzického prvku a požadovaná minimální rychlost linky, u linek typu `ethernet` se porovná typ ethernetu (např. na lince, která má mít rychlost `fast`, může být rozhraní `gigabit`, ale ne `legacy`)

---

[52] viz část 5.1 (strana 35)

[53] viz část 5.2 (strana 36)

[54] Můžeme si to ilustrovat na příkladu, kdy ve vybavení máme tři směrovače – jeden se dvěma rozhraními, druhý se třemi s třetí s osmi. Topologie požaduje zařízení se dvěma rozhraními. Žádné další požadavky nejsou. Mapovací algoritmus vybere směrovač se dvěma rozhraními.

- pokud se pro nějakou linku nenajde ani jedno možné rozhraní, je celé porovnávání ukončeno

Výše popsaný postup je implementován metodou `Availability` ze třídy `virtlabMapping` – viz příloha C.11 (strana 60). Pokud v některém z kroků nejsou podmínky splněny, funkce skončí a vrátí chybovou hodnotu. Pokud vše proběhne v pořádku, je vráceno pole, které udává, jaká rozhraní mohou být použita na jednotlivých linkách logické topologie. Tato část mapovacího algoritmu trvá nejdelší dobu, jelikož množství ověřování odpovídá součinu počtu vrcholů logické topologie a počtu fyzických zařízení. Některé z těchto ověření jsou ukončena záhy, pokud se neshoduje například typ síťového prvku, jiné ale trvají déle a ověřují všechna rozhraní na linkách daného vrcholu.

5.3.2 Ohodnocení zařízení

Abychom mohli vybrat zařízení, které jen minimálně přesahuje svými schopnostmi potřeby kladené logickou topologií, musíme mít mechanismus ohodnocení jednotlivých zařízení. Současný systém je poměrně jednoduchý – za každé rozhraní, nestandardní vlastnost, . . . je zařízení přiřazen určitý počet bodů (přesný počet je uveden v příloze D.2). Body tak představují jakousi cenu zařízení, která ovlivňuje proces výběru zařízení – jsou preferována zařízení s co nejnižší cenou.

Výše popsaný postup je implementován metodou `Evaluate` ze třídy `virtlabMapping` – viz část C.11 (strana 60).

5.3.3 Tabulkové mapování

Jádrem mapovacího algoritmu je jednoduchý problém výběru možností z tabulky. Představme si tabulku, kde řádky představují jednotlivé vrcholy logické topologie, sloupce zase představují fyzická zařízení. Buňky tabulky jsou vyplněny tehdy, pokud zařízení (indexující sloupec) může být daným vrcholem (indexující řádek) logické topologie. V každém sloupci a v každém řádku tak může být několik vyplněných buněk. Úkolem této funkce je vybrat vyplněné buňky tak, aby každý vrchol (řádek) měl právě jednu a vybraných zařízení (sloupců) bylo tolik, kolik je vrcholů – tzn., že každému vrcholu bude přiřazeno právě jedno unikátní zařízení.

Funkce musí brát ohled také na „cenu" zařízení a vybírat prvky od „nejlevnějšího". Implementovat tuto vlastnost není příliš komplikované, nejprve si ale musíme uvědomit, jak bude ona tabulka implementována. Tabulka je implementována jako dvojrozměrné pole, kde prvním indexem jsou vrcholy logické topologie, druhým indexem pak jednotlivá zařízení. Hodnoty prvků pole pak jsou ceny odpovídajícího zařízení. V dvojrozměrném poli jsou samozřejmě jen ony vyplněné buňky tabulky. Pokud si nyní seřadíme vnitřní pole vzestupně podle ceny a budeme zařízení vybírat od začátku, máme implementovánu onu vlastnost preference levnějších zařízení.

Průběh funkce je následující (funkce je rekurzivní):

- Jako prvotní vstup funkce slouží data vyprodukovaná funkcí `Availability`. Její výsledek udává, která zařízení mohou být kterým vrcholem logické topologie a tvoří tak vlastně tabulku, popisovanou výše.
- Funkce rekurzivně zjišťuje, jestli je namapování možné – tuto informaci nám sdělí návratová hodnota (`0` nebo `1`). V případě kladného výsledku je funkcí vrácen výsledek tabulkového mapovaní.

1. Ve vstupní tabulce se zjistí, jestli všechny řádky obsahují alespoň jeden prvek (jednu vyplněnou buňku) – pokud by neobsahovaly, znamená to, že pro nějaký vrchol není dosavadní[55] mapovaní možné a vratí se hodnota `0`
2. pokud vstupní tabulka obsahuje už jen jednu položku vnějšího pole (jeden nenamapovaný vrchol – tabulka má jen jeden řádek), vezme se první prvek vnitřního pole (kvůli úvodnímu setřídění, to bude zařízení s nejnižší možnou cenou). Příslušná dvojice vrchol-zařízení se uloží do výsledku a funkce vrátí `1`.
3. Projdou se všechny vrcholy, jestli některý nemá už jen jedno možné zařízení na namapování (jediný sloupec u některého z řádku)
   (a) Příslušná dvojice vrchol-zařízení (se všemi výskyty zařízení) se odstraní z tabulky (pomocí funkce `MatrixClear` – viz část D.1) a na zbytek tabulky se spustí funkce na tabulkové mapování (zde dochází ke zmíněné rekurzi), přičemž se zjišťuje návratová hodnota této funkce. Kladný výsledek (reprezentován `1`) znamená, že výběr dvojice vede k úspěšnému namapování a dvojice se tady uloží do výsledku a funkce je ukončena s hodnota `1`. Záporný výsledek znamená (reprezentován `0`), že namapování zbytku se nepovedlo, a jelikož tato dvojice je jediná možná (pro daný vrchol neexistuje jiné možné zařízení), je funkcí vrácena `0`.
4. Pokud má každý vrchol ve vstupní tabulce několik možných zařízení, nezbývá nic jiného, než projít postupně celou tabulku – respektive stačí projít pouze první (nebo libovolně jiný) řádek, protože pokud se nám mapování pro něj nepodaří, je celé tabulkové mapování neúspěšné. Každou dvojici vrchol-zařízení odstraníme z tabulky (pomocí funkce `MatrixClear` – viz část D.1), přičemž si původní tabulku zachováme. Na zbytek tabulky se pokusíme spustit funkci pro tabulkové mapování (zde dochází ke zmíněné rekurzi), přičemž opět kontrolujeme návratovou hodnotu. Kladný výsledek (reprezentován `1`) znamená, že výběr dvojice vede k úspěšnému namapování a dvojice se tady uloží do výsledku a funkce je ukončena s hodnota `1`. Záporný výsledek znamená (reprezentován `0`), že namapování zbytku tabulky se nepovedlo, a pokračuje se dále v procházení dvojic.
5. Pokud projdeme celý řádek a žádné mapování se nezdařilo (pro každou dvojici byla vrácena `0`), tak funkce končí a vrací `0`.

[55] musíme si uvědomit, že jde o rekurzivní funkci

Výše popsaný postup je implementován metodou `Matrix_mapping` ze třídy `virtlab-Mapping` – viz příloha C.11 (strana 60).

5.3.4 Vlastní průběh algoritmu

Celý mapovací algoritmus využívá jednotlivé funkce popsané výše. I když by se mohlo zdát, že výše popsané funkce řeší celý proces mapování, objevila se v průběhu analýzy jedna problematická část – jde o atribut **interface(connect_group)** – viz část 5.1 (strana 35).

Problém vězí v tom, že porovnání „skupiny propojitelnosti“ je možné až po skončení mapování – pokud není na některé lince shoda těchto hodnot, je potřeba spustit mapování znovu, ale evidovat si obě rozhraní a linku, na kterých došlo k nesouladu skupiny propojitelnosti. Dochází tedy k další rekurzi v algoritmu.

Průběh celého mapovacího algoritmu je následující:

1. je vyrobeno pole, které obsahuje data funkce `Availability`, která je spuštěna pro všechny dvojice vrchol-zařízení (tato dvojice je zároveň dvojicí indexů, které pole indexují). Pokud může být zařízení daným vrcholem, pak je hodnotou dvojrozměrného pole další dvojrozměrné pole, které popisuje, která rozhraní mohou být na kterých linkách. Pokud nemůže, pak je hodnota buňky dvojrozměrného pole číslo určující důvod, proč nemůže.

2. pro prvky dvojrozměrného pole, kde funkce `Availability` určila použitelná rozhraní na jednotlivých linkách, jsou zařízení ohodnocena funkcí `Evaluate`. Tento výsledek je pro jednotlivé vrcholy seřazen vzestupně podle ohodnocení zařízení.

3. nad tímto dvojrozměrným polem ohodnocení je spuštěna funkce `Matrix_mapping`, která určí, které zařízení bude kterým vrcholem

4. funkce `Matrix_mapping` je opětovně použita pro určení, které rozhraní bude na které lince (zde jde opět o tabulkové mapování, stejně jako v případě vrchol-zařízení)

5. na všech linkách se ověřuje shoda skupiny propojitelnosti rozhraní na nich ležících. Pokud je zjištěna neshoda, algoritmus pokračuje znova od bodu 2 – nejprve s odebráním jednoho rozhraní (voleno náhodně) z původního pole vzniklého v bodu 1, pokud se mapování opět nepovede, je odstraněno druhé rozhraní, pokud ani to nevede k úspěšnému mapování, odstraní se celé zařízení. Při všech těchto odebráních se kontroluje, jestli mají všechna zařízení ještě nějaká rozhraní na všech potřebných linkách – v případě odebrání celého zařízení se kontroluje, jestli nebylo posledním rozhraním pro některý vrchol. V těchto případech by mapování skončilo nezdarem.

Výše popsaný postup je implementován metodou `Map` ze třídy `virtlabMapping` – viz část C.11 (strana 60).

6 Závěr

V této práci jsem se snažil navrhnout a vytvořit webovou aplikaci Virtlabu takovým způsobem, aby byla v budoucnu snadno rozšířitelná o další funkce a další vývoj nemusel začínat opět od začátku. Mapovací algoritmus bylo potřeba vymyslet celý.

I přes značnou složitost se mapovací algoritmus během testování ukázal jako velice spolehlivý. Byl testován na velkých a komplikovaných topologiích, které obsahovaly i desítky fyzických prvků a ještě mnohem více linek, které je propojovaly. Během těchto testů se objevily jen drobné programátorské chyby, ale algoritmus jako celek se osvědčil.

Když jsem stál před problémem, jak celou webovou aplikaci postavit, vymyslel jsem systém generování stránek popsaný v části 3.2 (strana 13). Ten jsem vytvářel s ohledem na co nejjednodušší budoucí modifikace a co nejširší možnosti využití.

Další části systému jsem tvořil s podobnými ohledy – na snadnou modifikovatelnost a nenákladné budoucí rozšíření. Některé části systému se mohou zdát těžkopádné (například řešení multijazyčnosti systému), ale je to malá daň za jednoduchost budoucích modifikací. A budoucí rozšiřování je pro Virtlab velice důležité, protože prakticky neustále se objevují nové nápady na jeho vylepšení.

Jelikož je Virtlab projektem, na kterém se podílí značné množství lidí, bylo potřeba najít nástroj umožňující jednoduchou výměnu nápadů, informací a dokumentů. Ideálním nástrojem pro tuto distribuci informací je technologie wiki – konkrétně MediaWiki [12] (na tomto systému je postavena encyklopedie Wikipedia [13]). Mým úkolem bylo přizpůsobit MediaWiki pro potřeby Virtlab-týmu – výsledek je možno vidět na adrese `http://www.cs.vsb.cz/vl-wiki`.

6.1 Vývoj Virtlabu do budoucna

Již v průběhu implementace se objevilo několik oblastí, které by měly být přepracovány k větší obecnosti a lepší modifikovatelnosti. Některé z nich se podařilo zapracovat, ale některé svojí náročností překročily rozsah této práce a budou implementovány až následně. Neimplementované nápady jsou popsány v následujících částech.

6.1.1 Zahrnutí jazyků do databáze

Jako velmi důležité se ukázalo implementovat podporu pro více jazyků, nejen pro češtinu. Nicméně v současné době je implementace značně nevyhovující. Počet identifikátorů textového řetězce roste a je nutné kopírovat je do jednotlivých funkcí odpovědných za jednotlivé jazyky.

Za ideální řešení považuji, ukládat textové řetězce do databáze, kde jednotlivé propojení identifikátorů s textovým řetězcem některého jazyka zařídí klíče. Také bude možné implementovat různé úrovně identifikátorů – některé jsou totiž „systémové“ a některé spíše „uživatelské“, a tak je potřeba mít mechanismus, který zaručí, že systémové identifikátory bude moci měnit pouze administrátor. Ostatní identifikátory může měnit uživatel s nižším oprávněním.

6.1.2 Správa uživatelů

I když je ve webové aplikaci implementována základní správa uživatelů, je potřeba tuto správu rozšířit o další funkce. V současné době není nijak využívána položka uživatelské skupiny, ani čas expirace účtu. Bylo by velmi užitečné implementovat komponentu pro zasílání elektronické pošty uživatelům (při jimi určených událostech).

6.1.3 Logování

U konzolového serveru se velmi osvědčilo logování provozu. Myslím, že implementace logování pro webovou aplikaci by rovněž nebylo naškodu. Logování systému, kde v jednom okamžiku bude pracovat několik uživatelů, si vyžádá jiný přístup než přímý zápis do souboru – není možné, aby několik procesů najednou zapisovalo do jednoho souboru a záznam nebyl zpřeházený.

Jako ideální řešení vidím použít SQL databázi, která řeší problém souběžného zápisu několika procesů a umožňuje vzdálenou kontrolu logů.

V konfiguračním souboru pro webovou aplikaci se určí úroveň logování. Nejnižší úroveň by mohla logovat jen zásadní operace (komunikace s rezervačním serverem, mazání uživatelů, úloh, ...). Nejvyšší úroveň by evidovala veškerý pochyb uživatelů po webovém rozhraní. Každý záznam logu by měl obsahovat čas události (s co největší přesností), uživatele který událost způsobil, session id (důvod: stejný uživatel se přihlasí během krátké doby ze dvou míst/prohlížečů), uroveň zprávy (odpovídá úrovni logování), text záznamu.

7 Reference

[1] P. Němec, *Virtuální sít'ová laboratoř*, diplomová práce, VŠB-TU FEI, Ostrava, 2005

[2] R. Kubín, *Zajištění bezpečnosti a implementace nových prvků řídícího systému virtuální laboratoře*, diplomová práce, VŠB-TU FEI, Ostrava, 2006

[3] T. Hrabálek, *Podpora vytváření virtuálních topologií ve virtuální laboratoři počítačových sítí s využitím technologie tunelování*, diplomová práce, VŠB-TU FEI, Ostrava, 2007

[4] D. Seidl, *Systém pro automatizovanou správu sít'ových konfigurací*, diplomová práce, VŠB-TU FMMI, Ostrava, 2005

[5] P. Grygárek, *Terminologie distribuovaného Virtlabu*, VŠB-TU FEI, http://www.cs.vsb.cz/vl-wiki/index.php/Virtlab:DistrTerminologie

[6] P. Grygárek, D. Seidl, P. Němec, *Zpřístupnění prvků laboratoře počítačových sítí pro praktickou výuku prostřednictvím Internetu*, Sborník konference Technologie pro e-vzdělávání, ČVUT, Praha, ISBN 80-01-03274-4, 2005

[7] P. Grygárek, *Zkušenosti z nasazení virtuální laboratoře počítačových sítí a další směry jejího rozvoje*, Sborník semináře Technologie pro e-vzdělávání, FEL ČVUT, Praha, ISBN 80-01-03512-3, 2006

[8] D. Seidl, P. Grygárek, *Systém pro automatizovanou správu sít'ových topologií*, Seminář Opensource řešení v sítích 3, SLU Karviná, 2005

[9] P. Grygárek, D. Seidl, P. Němec, *Virtuální sít'ová laboratoř pro CNAP*, Výroční konference Cisco Networking Academy Program, Brno, 2005

[10] *The Xen virtual machine monitor*, University of Cambridge, http://www.cl.cam.ac.uk/research/srg/netos/xen/

[11] *Cisco 7200 Emulator*, University of Technology of Compiegne, France, http://www.ipflow.utc.fr/index.php/Cisco_7200_Simulator

[12] *MediWiki*, http://www.mediawiki.org

[13] *Wikipedia – the free encyclopedia*, http://www.wikipedia.org

[14] kolektiv autorů, *PHP: programujeme profesionálně*, Computer Press, ISBN 8072263102, 2001

[15] M. Snížek, *CSS pro zelenáče*, Neocortex, ISBN 8086330141, 2004

[16] kolektiv autorů, *Dokumentace PHP*, http://www.php.net/manual/cs/

A Co je třeba udělat, když . . .

Tato příloha má sloužit jako malá příručka k budoucím modifikacím nové webové aplikace. Z vlastního textu diplomové práce nemusejí být jednotlivé kroky modifikace zcela zřejmé.

A.1 Přidáváme novou stránku

Nejprve je potřeba v souboru `index.php` přidat část kódu, která stránku bude generovat – jde o část, která je v ukázce 3 (strana 16), náležící konstrukci `case` mnohonásobné podmínky. K tomu přísluší i přidělení čísla identifikující stránku – viz část 3.2 (strana 13).

Pokud chceme mít na stránku odkaz z menu, je potřeba upravit soubor `virtlab/menu.php` – to obnáší vytvoření identifikátoru pro přístup a identifikátoru zobrazeného řetězce (aby byla zaručena multijazyčnost) – viz ukázka 9 (strana 24). Tyto identifikátory je pak třeba zapsat do příslušných souborů: pro přístup do `class/virtlabWeb.php.inc` – viz ukázka 10 (strana 24), řetězec do `class/virtlabLanguage.php.inc` – viz ukázka 5 (strana 20). Dále je žádoucí, aby stránka obsahovala na začátku kód zabraňující neoprávněnému vstupu – viz ukázka 11 (strana 25).

A.2 Měníme přístupová práva na stránky

Je třeba přepsat metodu `Authorization` třídy `virtlabWeb`, která je popsána v části C.1 (strana 49). O přístupových právech také pojednává část 3.5.2 (strana 23).

A.3 Měníme databázový stroj

Je třeba přepsat metody třídy `vistlabSQL`, která je popsána v části C.5 (strana 51).

A.4 Chceme mít na každé stránce stejné informace

Pokud jsou tyto informace „menšího“ charakteru, je možno využít nepoužívané `div` kontejnery `header` a `footer`, které se nacházejí ve všech stránkách a jsou v současnosti nepoužívané. To obnáší upravit příslušné části generující stránku – ukázka 3 (strana 16) zobrazuje tuto část kódu. Také bude nejspíš potřeba upravit hlavní CSS styl `virtlab_style.css`, aby nově přidaná část stránky byla správně pozicována.

Pokud by bylo potřeba kompletně předělat vzhled webového rozhraní, musí se vytvořit nová šablona pro sekci `body` – viz ukázka 2 (strana 15) a pak příslušným způsobem upravit generování stránky (jako to bylo popsáno v předchozím odstavci).

B Použitá DTD

```
<!ELEMENT equipment (device*)>
<!ELEMENT device     (os, interfaces ?, special ?)>
<!ELEMENT os         (#PCDATA)>
<!ELEMENT interfaces ( interface +)>
<!ELEMENT interface  (max_bps?, int_feature*)>
<!ELEMENT max_bps    (#PCDATA)>
<!ELEMENT int_feature (#PCDATA)>
<!ELEMENT special    (feature+)>
<!ELEMENT feature    (#PCDATA)>

<!ATTLIST device
          type          (router | switch)           #REQUIRED
          name          CDATA                       #REQUIRED
          serial_number NMTOKEN                     #REQUIRED
          platform      CDATA                       #REQUIRED>
<!ATTLIST interface
          technology    ( serial | ethernet)        #REQUIRED
          ether_type    (legacy | fast | gigabit)   #IMPLIED
          connect_group NMTOKEN                     #REQUIRED
          name          CDATA                       #REQUIRED>
```

Výpis 19: DTD equipment

```
<!ELEMENT virtual_topology  (edge*, vertex_detail*)>
<!ELEMENT edge              (vertex, vertex, min_bps?, edge_feature*)>
<!ELEMENT vertex            EMPTY>
<!ELEMENT vertex_detail     (poss_platforms?,os?, vertex_feature*)>
<!ELEMENT poss_platforms    (platform+)>
<!ELEMENT platform          (#PCDATA)>
<!ELEMENT os                (#PCDATA)>
<!ELEMENT vertex_feature    (#PCDATA)>
<!ELEMENT min_bps           (#PCDATA)>
<!ELEMENT edge_feature      (#PCDATA)>

<!ATTLIST vertex
          name       IDREF                      #REQUIRED>
<!ATTLIST vertex_detail
          type       (router | switch)          #REQUIRED
          name       ID                         #REQUIRED>
<!ATTLIST edge
          technology ( serial | ethernet)       #REQUIRED
          name       CDATA                      #REQUIRED
          ether_type (legacy | fast | gigabit)  #IMPLIED>
```

Výpis 20: DTD topology

```
<!ELEMENT task              (longname, description, group_descrip, group_pics, group_preconf, group_postconf,
     group_topology)>
<!ELEMENT longname          (#PCDATA)>
<!ELEMENT description       (#PCDATA)>
<!ELEMENT group_descrip ( file *)>
<!ELEMENT group_pics        ( file *)>
<!ELEMENT group_preconf ( file *)>
<!ELEMENT group_postconf (file*)>
<!ELEMENT group_topology (file*)>
<!ELEMENT file              EMPTY>

<!ATTLIST task
          name     CDATA #REQUIRED
          time     CDATA #IMPLIED>
<!ATTLIST group_descrip
          name          CDATA        #REQUIRED
          exists        (yes | no)   #REQUIRED>
<!ATTLIST group_pics
          name          CDATA        #REQUIRED
          exists        (yes | no)   #REQUIRED>
<!ATTLIST group_preconf
          name          CDATA        #REQUIRED
          exists        (yes | no)   #REQUIRED>
<!ATTLIST group_postconf
          name          CDATA        #REQUIRED
          exists        (yes | no)   #REQUIRED>
<!ATTLIST group_topology
          name          CDATA        #REQUIRED
          exists        (yes | no)   #REQUIRED>
<!ATTLIST file
          name          CDATA        #REQUIRED
          filepath      CDATA        #REQUIRED
          exists        (yes | no)   #REQUIRED>
```

Výpis 21: DTD taskupload

C Popis PHP tříd

C.1 virtlabWeb

Popis

Třída sestavující výslednou stránku z několika sekcí, které jsou vytvořeny pomocí šablony a dat do ní vložených. Podrobnosti o generování stránek jsou v části 3.2 (strana 13). Tato třída také obsahuje funkci (`Authorization`), které ověřuje přístupová práva na jednotlivé stránky Virtlabu podle nastavení každého uživatele – jak práva pro zobrazení stránky, tak práva na zobrazení příslušných položek v menu (i když ve většině případů se obě práva budou řídit stejnými hodnotami, obecně jde o dvě různé věci) – viz část 3.5.2 (strana 23).

Proměnné

private $pages = array() pole objektů `virtlabWebPage`, které tvoří jednotlivé sekce výsledné HTML stránky. Sekce jsou generovány v pořadí, v jakém byly do pole uloženy.

Metody

public function AddPage($template_file, $parts=NULL) touto funkcí se do objektu vkládají jednotlivé sekce – prvním atributem je soubor se šablonou, druhým (volitelným) je proměnná s daty, která může šablona využívat (přes proměnnou `$_part`)

public function PrintPages() funkce provede vygenerování celé HTML stránky

public function Authorization($page_id, $rights=NULL) funkce sloužící k autorizování uživatele pro přístup na jednotlivé stránky. Prvním agrumentem funkce je textový identifikátor elementu, pro který ověřujeme přístupová práva. Druhý argument není zatím implementován – může posloužit k dodatečnému vložení informací. V současném okamžiku se přístupová práva uživatele zjišťují z dat v session.

C.2 virtlabWebPage

Popis

Třída, která vygeneruje jednu sekci HTML kódu – tedy podle jedné šablony. Je „základním kamenem" pro generování stránek - viz část 3.2 (strana 13)

Proměnné

private $template = '' proměnná sloužící k uložení cesty k souboru se šablonou

private $parts = array() ukládá data, která budou poskytnuta šabloně (přes proměnnou `$_part`)

Metody

function __construct($template_file, $parts=NULL) konstruktor třídy (v PHP verze 5). Prvním vstupním argumentem je cesta k souboru se šablonou. Druhým nepovinným argumentem jsou data pro šablonu.

public function PrintPage() vygeneruje HTML data podle vložené šablony

C.3 virtlabLanguage

Popis

Tato třída zajišťuje multijazyčnost Virtlabu. Na základě identifikátoru vrátí text v jazyce, který byl nastaven při konstrukci objektu. O multijazyčnosti pojednává část 3.4 (strana 20).

Proměnné

private language = '' identifikátor jazyka, který byl určen při konstrukci objektu

Metody

function __construct($language) konstruktor třídy (v PHP verze 5). Argumentem je identifikátor použitého jazyka. Zatím jsou implementovány jen hodnoty `cze` a `eng`.

public function CZE_text($identifier) obsahuje definici textu v češtině

public function ENG_text($identifier) obsahuje definici textu v angličtině

public function Text($identifier) tato metoda je volána z PHP skriptů a na základě vnitřní proměnné objektu je vrácen text v příslušném jazyce

C.4 virtlabParserTaskupload

Popis

Tato třída pomáhá s extrakcí dat z XML souboru, který popisuje data potřebná pro import úlohy. DTD tohoto souboru je ve výpisu 21 (strana 48).

Proměnné

private $parsed = array() obsahuje pole, které je výstupem parseru `virtlabXmlParser` a ze kterého jsou extrahována potřebná data

Metody

function __construct($file, $is_file = 0) konstruktor třídy (v PHP verze 5). Argument `$is_file` určuje, zda-li je argument `$file` cestou k XML souboru, nebo zda jde přímo o XML data.

public function getShortname() vrací krátký název úlohy

public function getTime() vrací informační časovou náročnost úlohy

public function getLongname() vrací dlouhý název úlohy

public function getShortdescription() vrací krátký popis úlohy

private function getGroup($group) vrací příslušnou část naparsovaného pole, ve které je popsaná `$group` skupina souborů. Povolené skupiny jsou: `descrip`, `pics`, `preconf`, `postconf` a `topology`.

public function getGroupExists($group) vrátí logickou hodnotu, zda-li skupina souborů existuje

public function getGroupName($group) vrátí jméno skupiny `$group`

public function getGroupFileCount($group) vrátí počet souborů ve skupině `$group`

private function getGroupFile($group, $fileidx) vrátí příslušnou část naparsovaného pole, ve kterém je popsán soubor s indexem `$fileidx` ze skupiny `$group`

public function getGroupFileName($group, $fileidx) vrátí jméno souboru s indexem `$fileidx` ze skupiny `$group`

public function getGroupFilePath($group, $fileidx) vrátí cestu k souboru, v rámci archívu, s indexem `$fileidx` ze skupiny `$group`

public function getGroupFileExists($group, $fileidx) vrátí logickou hodnotu, zda-li uvedený soubor existuje

C.5 virtlabSQL

Popis

Tato třída slouží jako zapouzdření nad SQL databází. Všechny skripty webové aplikace pracují s databází přes objekt této třídy. V současné verzi jsou implementovány funkce pro práci s databázovým serverem MySQL. Přechod na jiný databázový stroj si vyžádá pouze přepsání metod této třídy – což by u většiny SQL serverů neměl být problém, jelikož implementované funkce jsou elementárního charakteru.

Proměnné

private $dbserver ukládá IP adresu databázového serveru

private $dbuser ukládá databázového uživatele pro přístup k databázi

private $dbpassword ukládá heslo pro databázového uživatele

Metody

function __construct($sql_server, $sql_user, $sql_pass) konstruktor třídy (v PHP verze 5). Argumenty jsou: adresa databázového serveru, jméno databázového uživatele pro přístup a jeho heslo.

public function connect($die = 0) funkce provede spojení s databází, identifikátor spojení je funkcí vrácen. Volitelný argument `$die` určuje, zda-li má při chybě funkce ukončit další provádění PHP skriptů.

public function select_db($name, $die = 0) funkce v databázovém serveru vybere pracovní databázi `$name`

public function query($query, $db_name = 0, $die = 0, $whole = 0, $close = 0) tato funkce provede SQL dotaz `$query`. Výsledek dotazu je vrácen. Další argumenty umožňují, aby nemusel byt použit žádný jiný příkaz – funkce se spojí se serverem (je-li `$whole` nastaveno na `1`), vybere databázi `$db_name`, provede dotaz `$query` a ukončí spojení s databází (je-li `$close` nastaveno na `1`). Volitelný argument `$die` určuje, zda-li má při chybě funkce ukončit další provádění PHP skriptů.

public function fetch_assoc(&$resource) funkce z výsledku `$resource` vytáhne jeden řádek a vrátí ho jako asociativní pole – indexy odpovídají jménům databázových sloupců.

public function create_assoc($resource) funkce z výsledku `$resource` vyrobí dvojrozměrné pole. První index je celočíselný (odpovídá pořadí řádku dat v odpovědi), druhý index je textový, odpovídající jménům databázových sloupců.

public function num_row($resource) funkce vrátí počet řádků výsledku `$resource`

public function lastid() funkce vrátí hodnotu sloupce, který je nastaven jako *autoincrement*, vytvořeného posledním dotazem **INSERT**.

C.6 virtlabReservations

Popis

Tato třída implementuje komunikaci klient-server s rezervačním serverem. O této komunikaci pojednává část 3.8.1 (strana 28).

Proměnné

private $host = '' ukládá adresu rezervačního serveru

private $port = '' ukládá TCP port, pro komunikaci s rezervačním serverem

private $locality = '' ukládá název místní lokality

private $distrib = '' ukládá řetězec použitý pro atribut `DISTIB` během komunikace s rezervačním serverem

private $line_length = 255 interní proměnná, určující po kolika bytech se má z komunikačního streamu číst

Metody

function __construct($host, $port, $locality, $distrib) konstruktor třídy (v PHP verze 5). Argumenty mají tento význam: adresa rezervačního serveru, TCP port rezervačního serveru, lokalita, řetězec pro atribut `DISTRIB` – viz [3].

private function connect() funkce se spojí s rezervačním serverem, je vrácen identifikátor spojení

public function GetOffer() funkce si od rezervačního serveru vyžádá kompletní seznam vybavení a ten je funkcí vrácen

public function GetOffer_time($from, $to) funkce si od rezervačního serveru vyžádá seznam zařízení dostupných v čase od `$from` do `$to` a ten je funkcí vrácen

public function Reserve($resID, $devices, $from, $to) funkce u rezervačního serveru zaeviduje rezervaci s identifikátorem `$resID` v čase od `$from` do `$to`, pro pole zařízení `$devices`

public function Cancel($resID) funkce zruší rezervaci s identifikátorem `$resID`

public function Attach($resID, $connections) funkce předá rezervačnímu serveru data `$connections` vzniklé mapovacím algoritmem, která určují, jak mají být zařízení propojena pro rezervaci `$resID`

C.7 virtlabWebFile

Popis

Tato třída zajišťuje správu souborů uložených v tabulce databáze – viz část 3.9 (strana 30).

Proměnné

private $sqler = NULL ukládá odkaz na objekt typu `virtlabSQL`

private $db = '' ukládá název pracovní databáze

private $table = '' ukládá název tabulky se soubory

Metody

function __construct($sql_serv, $sql_usr, $sql_pas, $sql_db, $sql_tab) konstruktor třídy (v PHP verze 5). Argumenty mají tento význam: adresa databázového serveru, databázový uživatel pro přístup, jeho heslo, pracovní databáze, tabulka se soubory.

public function Set($name, $filename, $size, $type, $user, $data) funkce uloží soubor do tabulky pod jménem `$name`, jako zdrojový název souboru nastaví `$filename`, velikost souboru v bytech `$size`, MIME typ souboru `$type`. Jako tvůrce souboru označí uživatele `$user`. Data souboru jsou v argumentu `$data`.

public function Get($identif, &$type, &$size, &$name) funkce vrátí soubor s identifikátorem `$identif`. Do argumentů `$type`, `$size` a `$name` jsou vložený příslušné hodnoty. Data souboru jsou vráceny funkcí.

public function Get_info($identif) funkce vrátí pole informací o souboru s identifikátorem `$identif`

public function Delete($identif) funkce vymaže soubor s identifikátorem `$identif`

public function Change($identif, $filename, $size, $type, $user, $data) funkce provede změnu souboru s identifikátorem `$identif`. Argumenty mají stejný význam jako u funkce `Set`.

C.8 virtlabXmlParser

Popis

Tato třída zajišťuje převod XML souborů na asociativní pole metodou SAX.

Tvorba pole probíhá na principu „zásobníku“. Postupně se při otevíracích značkách tvoří pole, která mají jen položku `attribs` vyplněnou polem atributu. Pokud se zpracovávají data značky, tak se vloží do položky pole `content` posledního prvku (pole) „zásobníku“. Funkce pro obsluhu uzavírací značky vyzvedne poslední položku „zásobníku“ a vloží ji do předchozí položky zásobníku pod indexem `child`. Tímto postupem se postupně tvoří strom polí, které obsahuje data z XML.

Ve výpisu 22 (strana 55) jsou zobrazeny XML data a výstup tohoto parseru je ve výpisu 23[56] (strana 56) – pro výpis pole je použit upravený výstup PHP funkce **`print_r`**.

[56] výpis může zprvu působit chaoticky, ale je to jen strukturou. Např.: hodnota `minor` verze OS je uložená v poli pod indexem: [0]['child'][0]['child'][0]['child'][1]['content']

Proměnné

private $xml_obj = NULL obsahuje odkaz na interní PHP objekt – SAX parser

private $valid = 0 rozhoduje, zda-li mají XML data přijít validací (v PHP verze 5)

public $output = array() výstupní pole parseru – zobrazeno ve výpisu 23 (strana 56)

Metody

function __construct($validition = 0) konstruktor třídy (v PHP verze 5). Volitelným argumentem je určení, zda-li má proběhnout validace XML.

private function startHandler($parser, $name, $attribs) funkce pro SAX – pro otevírací značku

private function dataHandler($parser, $data) funkce pro SAX – pro data značky

private function endHandler($parser, $name) funkce pro SAX – pro uzavírací značku

public function parse($path) funkce spustí parsování souboru s adresou `$path`

public function parse_data($data) funkce spustí parsování řetězce `$data` s XML daty

```
<?xml version="1.0" encoding="utf−8" ?>
<equipment>
   <device type="router" name="r1">
      <os>
         <major>12</major>
         <minor>8</minor>
      </os>
  </device>
</equipment>
```

Výpis 22: XML soubor

C.9 virtlabParserEquipment

Popis

Tato třída pomáhá s extrakcí dat z XML souboru, který popisuje vybavení. DTD tohoto souboru je ve výpisu 19 (strana 47).

Proměnné

parsed = array() veškerá data získaná z XML souboru parserem `virtlabXmlParser`

cached_devices = 0 určuje, jestli se mají data o laboratorních prvcích cacheovat v proměnné `devices`

```
Array
(0 => Array
    (
        [name] => EQUIPMENT
            [content] =>
            [child] => Array
                (0 => Array
                    (
                        [name] => DEVICE
                        [attribs] => Array( 'TYPE' => router, 'NAME => r1 )
                        [content] =>
                        [child] => Array
                            (0 => Array
                                (
                                    [name] => OS
                                    [content] =>
                                    [child] => Array
                                        (
                                            [0] => Array( 'name' => MAJOR, 'content' => 12 )
                                            [1] => Array( 'name => MINOR, 'content => 8 )
                                        )
                                )
                            )
                    )
                )
    )
)
```

Výpis 23: Výstup parseru (třídy) `virtlabXmlParser`

devices = array() cache dat o laboratorních prvcích

Metody

function __construct($file, $is_file = 0, $cache_dev = 1) konstruktor třídy (v PHP verze 5). Argumenty mají tento význam: `$is_file` určuje, zda-li je `$file` cestou k souboru, nebo jde přímo o XML data. Argument `$cache_dev` nastavuje, jestli se mají data o laboratorních prvcích uložit do proměnné `$devices`, ze které se při potřebě načítají, nebo se vždy čtou z původního výstupu třídy `virtlabXmlParser` uloženém v `$parsed`.

public function getDevices() vrátí pole s kompletními daty o laboratorních prvcích

public function getDevicesCount() vrátí počet laboratorních prvků

public function getDevice($index) vrátí určený laboratorní prvek (odpovídající úsek z naparsovaných dat), podle parametru `$index`, který odpovídá jejímu indexu v celkovém poli všech laboratorních prvků

public function getDeviceType($index) vrátí typ zadaného laboratorního prvku

public function getDeviceName($index) vrátí název zadaného laboratorního prvku

public function getDeviceSerial($index) vrátí sériové číslo laboratorního prvku

public function getDevicePlatform($index) vrátí platformu laboratorního prvku

public function getDeviceByName($name, &$index) vrátí požadovaný laboratorní prvek (určený jeho názvem). Do parametru `$index` se uloží jeho index

public function getDeviceOS($name, $variant = 1) vrátí verzi OS laboratorního prvku. Parametr `$variant` určuje jestli má být výstup upravený nebo v podobě, jako v původním poli.

public function getDeviceInterfaces($name) vrátí rozhraní zadaného laboratorního prvku

public function getDeviceInterfacesCount($name) vrátí počet rozhraní zadaného laboratorního prvku

public function getDeviceInterface($name, $index, $variant = 1) vrátí určené rozhraní (určené parametrem `$index`) určeného laboratorního prvku `$name`

public function getDeviceInterfaceTechnology($name, $index) vrátí technologii rozhraní laboratorního prvku

public function getDeviceInterfaceEthertype($name, $index) vrátí druh ethernetu rozhraní. Pokud je technologie `serial`, vrací `NULL`.

public function getDeviceInterfaceMaxbps($name, $index) vrátí maximální rychlost rozhraní. Pokud je technologie `ethernet`, vrací `NULL`.

public function getDeviceInterfaceFeatures($name, $index) vrátí speciální vlastnosti zadaného rozhraní

public function getDeviceFeatures($name) vrátí speciální vlastnosti laboratorního prvku

public function getDevicesFeatures() vrátí všechny speciální vlastnosti všech laboratorních prvků – i s duplicitami

public function getDevicesByType($type) vrátí jména laboratorních prvků daného typu

public function getDevicesTypes() vrátí typy laboratorních prvků, které se ve vybavení vyskytují

public function getDeviceInterfacesFeatures($name) vrátí speciální vlastnosti všech rozhraní daného laboratorního prvku

public function getDevicesInterfacesFeatures() vrátí speciální vlastnosti všech rozhraní všech laboratorních prvků

public function getDevicesList() vrátí seznam názvů všech laboratorních prvků

public function getDeviceInterfaceName($name, $index) vrátí jméno zadaného rozhraní

C.10 virtlabParserTopology

Popis

Tato třída pomáhá s extrakcí dat z XML souboru, který popisuje logickou topologii. DTD tohoto souboru je ve výpisu 20 (strana 47).

Proměnné

parsed = array() veškerá data získaná z XML souboru parserem `virtlabXmlParser`

cached_edges = 0 určuje, jestli se mají data o spojích cacheovat v proměnné `edges`

cached_vertexes = 0 určuje, jestli se mají data o vrcholech logické topologie cacheovat v proměnné `vertexes`

edges = array() cache dat o linkách logické topologie

vertexes = array() cache dat o vrcholech logické topologie

Metody

function __construct($file, $is_file = 0, $cache_ed = 1, $cache_ve = 1) konstruktor třídy (v PHP verze 5). Argumenty mají tento význam: `$is_file` určuje, zda-li je `$file` cestou k souboru, nebo jde přímo o XML data. Argument `$cache_ed` nastavuje, jestli se mají data o linkách logické topologie uložit do proměnné `edges`, ze které si při potřebě načítají, nebo se vždy čtou z původního výstupu třídy `virtlabXmlParser` uloženém v `parsed`. Argument `$cache_ve` funguje stejně jako `$cache_ed`, ale pro jednotlivé vrcholy logické topologie.

public function getEdges() vrátí část původního pole, která popisuje jen linky virtuální topologie

public function getEdgesCount() vrátí počet linek v topologii

public function getEdge($index) vrátí určenou linku (podle parametru `$index`, který odpovídá jejímu indexu v celkovém poli všech linek)

public function getEdgeName($index) vrátí název zadané linky

public function getEdgeByName($name, &$index) vrátí požadovanou linku (určenou jejím názvem). Do parametru `$index` se uloží její index.

public function getEdgeTechnology($index) vrátí technologii linky – množina návratových hodnot odpovídá definici atributu `technology` značky `edge` v XML souboru s popisem topologie.

public function getEdgeEthertype($index) vrátí druh ethernetu linky. Pokud je technologie `serial`, vrací `NULL`.

public function getEdgeMinbps($index) vrátí požadovanou minimální rychlost linky. Pokud je technologie `ethernet`, vrací `NULL`.

public function getEdgeVertexes($index) vrátí zařízení určené linky

public function getEdgesByVertex($name) vrátí linky, na kterých leží zadané zařízení. Pokud na žádné, vrací `NULL`.

public function getEdgeFeatures($index) vrátí speciální vlastnosti linky. Pokud nejsou zadány, vrací `NULL`.

public function getEdgesFeatures() vrátí všechny speciální vlastnosti všech linek v topologii

public function getEdgesList($variant = 1) vrátí seznam linek v topologii. Parametr `$variant` určuje jestli má být výstup upravený nebo v podobě, jako v původním poli.

public function getVertexes() vrátí část původního pole, která popisuje jen zařízení virtuální topologie

public function getVertexesCount() vrátí počet zařízení v topologii

public function getVertex($index) vrátí určené zařízení podle indexu

public function getVertexName($index) vrátí název zařízení podle indexu

public function getVertexByName($name) vrátí požadované zařízení (určené jménem)

public function getVertexPlatforms($name, $variant = 1) vrací zadané platformy zařízení. Parametr `$variant` určuje jestli má být výstup upravený nebo v podobě, jako v původním poli.

public function getVertexOS($name, $variant = 1) vrátí zadanou verzi OS. Parametr `$variant` určuje jestli má být výstup upravený nebo v podobě, jako v původním poli.

public function getVertexType($name) vrátí typ zařízení

public function getVertexesByType($type) vrátí zařízení určeného typu

public function getVertexesTypes() vrátí všechny typy zařízení v topologii

public function getVertexFeatures($name) vrátí speciální vlastnosti zařízení

public function getVertexesFeatures() vrátí všechny speciální vlastnosti všech zařízení v topologii

public function getVertexesList($variant = 1) vrátí seznam zařízení v topologii

C.11 virtlabMapping

Popis

Tato třída implementuje vlastní namapování fyzických prvků na prvky v logické topologii, tak aby byly splněny všechny podmínky. Popis mapovacího algoritmu je v části 5 (strana 35).

Proměnné

equipment odkaz na objekt `virtlabParserEquipment`, který poskytuje data o laboratorních prvcích

topology odkaz na objekt `virtlabParserTopology`, který poskytuje data o logické topologii

Metody

function __construct(virtlabParserEquipment $eq, virtlabParserTopology $top) konstruktor třídy (v PHP verze 5). Jako parametry očekává objekty jednotlivých parseru – virtuální topologie a vybavení.

public function Evaluate($device) vrátí vypočtenou hodnotu ceny (tu ovlivňuje typ fyzického prvku, počet rozhraní, . . .) zadaného fyzického prvku. Nastavení konstant třídy `virtlabValues` ovlivní výslednou hodnotu.

public function DevicesValue() funkce vrátí pole všech fyzických prvků s jejich vypočtenou cenou

public function Availability($device, $vertex) funkce zjistí, zda-li může být zadaný fyzický prvek, zařízením v logické topologii. Pokud ano, vrátí pole s určením, která rozhraní mohou být použita, na kterých linkách logické topologie. Pokud ne, vrátí číslo chyby – definováno ve třídě `virtlabValues`.

private function Matrix_mapping($matrix, &$vysledek) rekurzivní funkce, která se snaží mapovat. Ve dvojrozměrné poli `$matrix` je uloženo, který vrchol logické topologie může být realizován jakými fyzickými prvky. Případný výsledek mapovaní je uložen do proměnné `$vysledek`. Tato funkce je psána obecně, takže je ve druhém kroku znovu použita na mapovaní *LINKA–ROZHRANÍ*.

public function Map() funkce obstarávající celý algoritmus mapování

D Podpůrné PHP funkce

Pro různé části nové implementace Virtlabu bylo třeba vytvořit pomocné funkce, které jsou velmi různorodého charakteru.

D.1 virtlabSupportFunctions.php.inc

function ClearWhitespaceInXML($data)

Pomocí regulárního výrazu z XML dat odstraní „bílé znaky", které jsou mezi jednotlivými značkami – vznikají při editaci XML lidmi – pro ty je přehlednější uspořádání dat pomocí odřádkování a tabulátorů. Výsledné XML je vráceno.

function Unique($array)

Z pole se odstraní duplicitní prvky. Výsledné pole je vráceno.

function array_delete(&$array, $value)

Z pole `$array` je odstraněn prvek `$value`. Pokud byl prvek odstraněn, je vrácena hodnota `1` a změněné pole ve vráceno v argumentu `$array`, pokud odstraněn nebyl je vrácena hodnota `0`. Pokud vstupní argument `$array` není pole, je vrácena hodnota `NULL`.

function array_porovnej($array1, $array2)

Funkce porovná obě vstupní pole a vrátí prvky, které jsou pouze v poli `$array1`. Tato funkce bere ohled i na duplicitní prvky polí.

function DoubleArrayItems($array)

Funkce zdvojí prvky pole. Indexy původního pole jsou odstraněny.

function MatrixClear(&$matice, $radek, $sloupec)

Funkce odstraní z pole `$matice`, které musí být dvojrozměrným polem řádek s indexem `$radek` a sloupec s indexem `$sloupec` – odstraní hodnoty `$matice[$radek][*]` a `$matice[*][$sloupec]`.

function VerifyQuota($user, $from, $to, $window, $credit, $excluded_res=NULL)

Tato funkce implementuje „systém klouzavého okna", který je popsán v části 3.8.2 (strana 29). Pro uživatele `$user` v čase od `$from` do `$to` zjistí, zda-li je jeho kvóta `$credit` dostatečný pro rezervaci úlohy – ta je vymezena dobou od `$from` do `$to`. Argument `$window` určuje velikost okna, pro které funkce kvótu uživatele zkoumá – viz 3.8.2 (strana 29). Argument `$excluded_res` je pole, jehož prvky jsou identifikátory rezervací, které nemají být do výpočtu zahrnuty.

D.2 virtlabValues.php.inc

Tento soubor obsahuje několik konstant, které využívá algoritmus mapování popsaný v části 5 (strana 35). Prvních 10 položek se používá pro ohodnocení zařízení, záporné hodnoty odpovídají chybovým návratovým hodnotám funkce `virtlabMapping::-Availability` viz část C.11 (strana 60), zbylé konstanty jsou výchozími hodnotami vlastností prvků, které nemusejí být zadány.

const DeviceFeature = 125 počet bodů[57] za každou uvedenou vlastnost zařízení

const InterfaceTechnologySerial = 100 počet bodů za sériové rozhraní

const InterfaceTechnologyEthernet = 20 počet bodů za ethernetové rozhraní

const InterfaceFeature = 30 počet bodů za každou uvedenou vlastnost rozhraní

const EthertypeMultiplerLegacy = 1 násobitel bodů za ethernetové rozhraní, pokud jde o 10Mb ethernet

const EthertypeMultiplerFast = 2 násobitel bodů za ethernetové rozhraní, pokud jde o 100Mb ethernet

const EthertypeMultiplerGigabit = 5 násobitel bodů za ethernetové rozhraní, pokud jde o 1000Mb ethernet

const bpsLower = 0.8 násobitel bodů za sériové rozhraní, pokud je jeho maximální rychlost menší než **`defaultMaxbps`**

const bpsBigger = 1.2 násobitel bodů za sériové rozhraní, pokud je jeho maximální rychlost vyšší než **`defaultMaxbps`**

const prefer = 300 pokud je zařízení z preferované lokality, je mu odpočten tento počet bodů

const badType = -1 zařízení nejsou stejného typu

const badPlatform = -3 požadovaná platforma se neshoduje s platformou zařízení

const badOS = -10 požadovaná verze OS nesouhlasí s OS zařízení

const noDeviceFeature = -11 jsou požadováný speciální vlastnost, které zařízení nemá

const VertexDeviceMismatch = -13 zadané zařízení nemůže být zadaným vrcholem

const notEnoughInterfaces = -14 zařízení nemá dostatek rozhraní, aby pokrylo potřeby kladené logickou topologií

[57] tato hodnota nemá žádnou jednotku, jde jen o jistou „cenu" zařízení – čím je zařízení speciálnější, má více rozhraní, ... tím má vyšší tuto hodnotu. Mapování vybírá zařízení od „nejlevnějších" ke „dražším"

const defaultMaxbps = 64000 pokud není v XML dokumentu (pro popis zařízení) definována hodnota `maxbps` u seriového rozhraní, použije se tato hodnota

const defaultEthertype = "fast" pokud není v XML dokumentu (pro popis zařízení) definována hodnota `ethertype`, u ethernetového rozhraní, použije se tato hodnota

E Struktura databáze

sloupec	typ	integritní omezení	komentář
group_id	int(11)	cizí klíč `filegroup(id)`	identifikátor skupiny souborů
file_id	int(11)	cizí klíč `files(id)`	identifikátor souboru

Tabulka 3: SQL tabulka `filegroup_members`

sloupec	typ	integritní omezení	komentář
id	int(11)		jednoznačný identifikátor
descrip	varchar(50)		textový identifikátor skupiny

Tabulka 4: SQL tabulka `filegroups`

sloupec	typ	integritní omezení	komentář
id	int(11)		jednoznačný identifikátor
name	varchar(50)		textový identifikátor souboru
filename	varchar(100)		jméno souboru na disku
size	bigint(20)		velikost souboru
type	varchar(50)		MIME typ souboru
time	datetime		datum poslední změny souboru
user	varchar(20)	cizí klíč `users(id)`	identifikátor uživatele, který soubor nahral, nebo změnil
data	longblob		binární data souboru

Tabulka 5: SQL tabulka `files`

sloupec	typ	integritní omezení	komentář
resID	int(11)	cizí klíč `reservations(id)`	identifikátor rezervace
user_id	varchar(20)	cizí klíč `users(id)`	identifikátor uživatele

Tabulka 6: SQL tabulka `reservation_users`

sloupec	typ	integritní omezení	komentář
id	int(11)		jednoznačný identifikátor rezervace
creator	varchar(20)	cizí klíč `users(id)`	identifikátor tvůrce rezervace
from	datetime		čas začátku rezervace
to	datetime		čas konce rezervace
task_id	int(11)	cizí klíč `tasks(id)`	identifikátor úlohy
used_topology	int(11)	cizí klíč `files(id)`	identifikátor souboru s použitou topologií
mapping_result	blob		výsledek mapovacího algoritmu

Tabulka 7: SQL tabulka `reservations`

sloupec	typ	integritní omezení	komentář
resID	int(11)	cizí klíč `reservations(id)`	identifikátor rezervace
device	varchar(150)		název fyzického zařízení
vertex	varchar(150)		název logického zařízení

Tabulka 8: SQL tabulka `reserved_devices`

sloupec	typ	integritní omezení	komentář
id	int(11)	cizí klíč `tasks(id)`	jednoznačný identifikátor kategorie
class_id	int(11)	cizí klíč `task_classes(id)`	identifikátor třídy, pod kterou kategorie náleží
name	varchar(50)		itentifikátor textu

Tabulka 9: SQL tabulka `task_categories`

sloupec	typ	integritní omezení	komentář
id	int(11)		jednoznačný identifikátor třídy
name	varchar(50)		itentifikátor textu

Tabulka 10: SQL tabulka `task_classes`

sloupec	typ	integritní omezení	komentář
id	int(11)		jednoznačný identifikátor úlohy
name_short	varchar(30)		krátký název úlohy – zkratka
name_long	varchar(150)		dlouhý název úlohy
creator	varchar(20)	cizí klíč `users(id)`	identifikátor tvůrce úlohy
descr_short	blob		krátký textový popis úlohy
descr_long	int(11)	cizí klíč `filesgroups(id)`	identifikátor skupiny souborů s popisem úlohy
time	int(11)		informativní složitost úlohy (v minutách)
pics	int(11)	cizí klíč `filesgroups(id)`	identifikátor skupiny souborů s obrázky k úloze
conf_pre	int(11)	cizí klíč `filesgroups(id)`	identifikátor skupiny souborů s předkonfiguracemi
conf_post	int(11)	cizí klíč `filesgroups(id)`	identifikátor skupiny souborů s ukázkovými výslednými konfiguracemi
topology	int(11)	cizí klíč `filesgroups(id)`	identifikátor skupiny souborů s topologiemi úlohy

Tabulka 11: SQL tabulka `tasks`

sloupec	typ	integritní omezení	komentář
task_id	int(11)	cizí klíč `tasks(id)`	identifikátor úlohy
categ_id	int(11)	cizí klíč `task_categories(id)`	identifikátor kategorie

Tabulka 12: SQL tabulka `tasks_categories`

sloupec	typ	integritní omezení	komentář
id	char(3)		třínakový ISO identifikátor jazyka

Tabulka 13: SQL tabulka `languages`

sloupec	**typ**	**integritní omezení**	**komentář**
id	varchar(20)		jednoznačný identofokátor uživatele
passwd	varchar(40)		hash hesla uživatele, nebo identifikátor autentizace přes LDAP
first_name	varchar(50)		křestní jnémo uživatele
surname	varchar(50)		přijmení uživatele
email	varchar(50)		e-mailová adresa
lang	char(3)	cizí klíč `languages(id)`	preferovaný jazyk GUI
user_group	varchar(20)		uživatelská skupina
expires	datetime		datum vypršení platnosti
group_admin	tinyint(1)		přístupová práva – administrátor
group_task	tinyint(1)		přístupová práva – správce úloh
group_tutor	tinyint(1)		přístupová práva – tutor
quota	int(11)		kvóta uživatele
notepad	blob		poznámkový blok uživatele

Tabulka 14: SQL tabulka `users`