VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Reimplementace řídící aplikace systému Virtlab s použitím moderních webových technologií

Reimplementation of Virtlab System Control Application Using Modern Web Technologies

2010 Martin Grames

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

V Ostravě 6. května 2010 .....

Rád bych na tomto místě poděkoval členům Virtlabu a zejména svému vedoucímu diplomové práce - Ing. Petru Grygárkovi, Ph.D.

Abstrakt

Tato diplomová práce se zabývá reimplementací webové řídící aplikace distribuované virtuální laboratoře počítačových sítí. Zahrnuje analýzu současné řídící aplikace a velkou částí se věnuje výběru vhodného aplikačního frameworku a nástroje pro práci s datovou vrstvou. Několik kandidátů je důkladně prozkoumáno a jsou u nich shrnuty jejich přednosti a nedostatky. Vybraný aplikační framework a knihovna poskytující vhodnou vrstvu pro práci s daty jsou následně použity k implementaci nového řešení řídící aplikace. Reimplementovaná aplikace již od základu využívá principů objektově orientovaného programování a pracuje s moderními technologiemi.

Klíčová slova: Virtlab, webová aplikace, PHP framework, datová vrstva

Abstract

This thesis deals with the reimplementation of control web application in distributed virtual networking laboratory. It includes analysis of current control application and large part is devoted to selecting an appropriate application framework and tool for working with the data layer. Several candidates are thoroughly inspected and there are summarized their strengths and weaknesses. Selected application framework and library that provides a suitable layer for working with data are then used to implement a new solution of control application. Reimplemented application, from the base already, uses the principles of object-oriented programming and it works with modern technologies.

Keywords: Virtlab, web application, PHP framework, data layer

Seznam použitých zkratek a symbolů

CSS	–	Cascading Style Sheets
DFD	–	Data Flow Diagram
HTML	–	Hyper Text Markup Language
OOP	–	Object-oriented programming
ORM	–	Object-relational mapping
PHP	–	PHP: Hypertext Preprocessor
SQL	–	Structured Query Language

Obsah

Seznam obrázků

1 Úvod

Distribuovaná virtuální laboratoř počítačových sítí umožňuje zpřístupnění reálných síťových prvků, kterými laboratoř disponuje, uživatelům skrze internet. Uživatelé si rezervují úlohy na určitý čas. V daném čase pak s úlohami pracují, nastavují síťové prvky, testují je apod. Všechny tyto úkony jsou prováděny skrz řídící webovou aplikaci, která postupem času narostla a je již složité do ní implementovat novou funkcionalitu. Stávající aplikace není vhodně rozdělena do funkčních celků a nerespektuje principy objektově orientovaného programování.

Tato diplomová práce řeší zmíněné nedostatky reimplementací celé aplikace s důrazem na použití OOP a obecně metod softwarového inženýrství. Nejdříve je potřeba provést analýzu stávající aplikace, poté vyberu vhodný programovací jazyk. Následuje výběr frameworku, vhodného pro tento systém a používající moderní technologie. Pak je ještě potřeba nalézt vhodný nástroj pro práci s datovou vrstvou, který bude umožňovat případný přesun na jinou databázi bez změn v kódu. Také bude tato práce sloužit jako dokumentace nového řešení řídící aplikace pro další vývojáře, kteří budou na projektu pracovat.

2 Popis distribuované virtuální laboratoře

2.1 Distribuovaná virtuální laboratoř (Virtlab)

Vznik projektu virtuální laboratoře byl iniciován v roce 2005 v diplomové práci Pavla Němce. Jednalo se však pouze o nedistribuovanou variantu, která neumožňovala dynamické spojování topologií. To mohlo být realizováno až díky diplomové práci Davida Seidela, který navrhl zařízení ASSSK pro automatické spojování rezervací. Zabezpečení virtuální laboratoře pak bylo dopracováno v diplomové práci Romana Kubína.

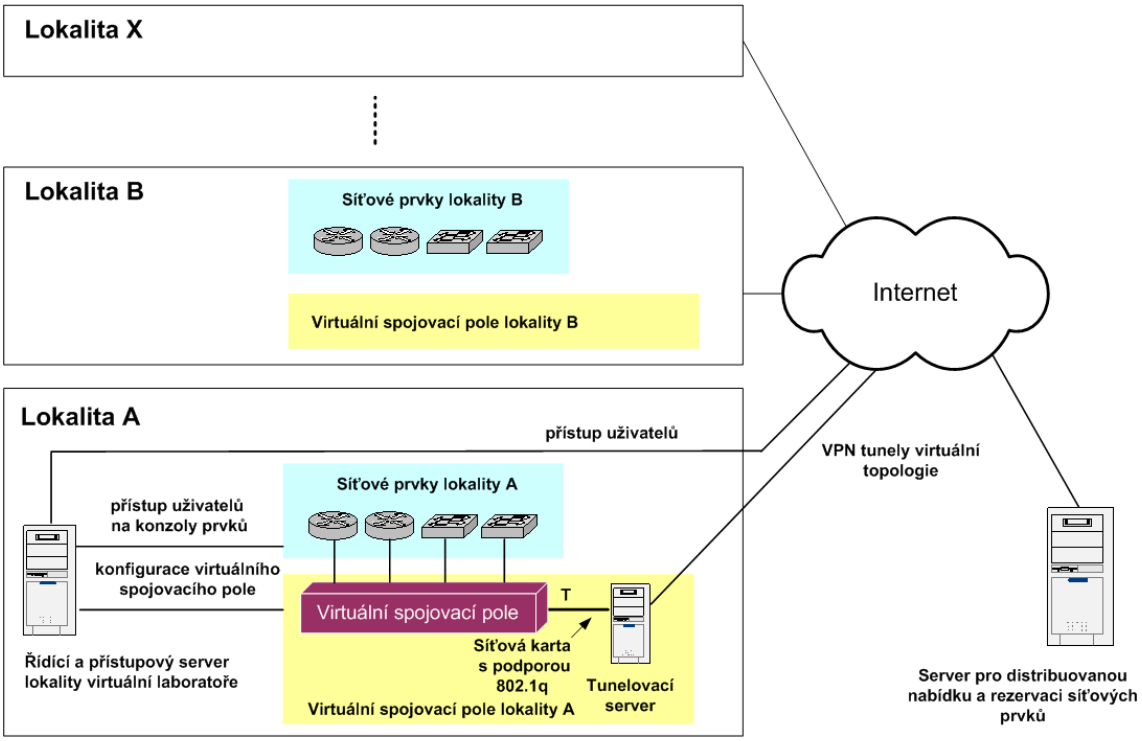
Jelikož je každá laboratoř tvořena nákladnými zařízeními a tato zařízení nejsou vždy plně využita, byla přijata myšlenka vytvořit virtuální laboratoř distribuovaně. Distribuovanost obnáší existenci více lokalit virtuální laboratoře, které si vzájemně poskytují zařízení vhodná do virtuálních topologií. Architektura distribuované virtuální laboratoře byla popsána v diplomových pracích Jana Vavříčka a Tomáše Hrabálka. Taktéž bylo v tomto roce vyvinuto modernizované zařízení pro automatické spojování konfigurací nazvané ASSSK2. Vývoj tohoto zařízení je popsán v diplomové práci Petra Sedláře.

Jako další přínosy pro virtuální laboratoř můžeme uvést například implementaci sledování provozu na VLAN popsanou v bakalářské práci Radka Nováka, modernizaci jedné z klíčových komponent virtuální laboratoře - Tunelovacího serveru popsané v bakalářské práci Václava Bortlíka nebo implementaci automatických testů popsanou v diplomové práci Zdeňka Filipce. Veškerý soupis diplomových prací lze nalézt na oficiálních stránkách projektu www.virtlab.cz.

2.2 Architektura distribuované virtuální laboratoře

Celá distribuovaná virtuální laboratoř se skládá z tzv. lokalit. Lokalitou se rozumí soubor softwarových a jiných technických prostředků, které tvoří samostatnou funkční jednotku virtuální laboratoře. Tato funkční jednotka samozřejmě může komunikovat s dalšími lokalitami a využívat jejich zařízení. Každá lokalita je tvořena třemi důležitými částmi:

- **Řídící webová aplikace** - jedná se o webovou aplikaci, se kterou komunikují uživatelé virtuální laboratoře a skrze kterou si rezervují své úlohy, přistupují na zařízení, píšou si mezi sebou e-maily apod.
- **Serverová část** - jedná se o mezivrstvu mezi webovou aplikací a zařízeními. Zajišťuje přeposílání datových toků mezi jednotlivými lokalitami, umožňuje přístupy na zařízení a taktéž spravuje a půjčuje zařízení pro rezervace.
- **Zařízení** - jedná se o reálné síťové prvky nebo zařízení nutné pro běh virtuální síťové laboratoře. Například zařízení ASSSK nebo MOXA karty.



Obrázek 1: Architektura distribuované virtuální laboratoře

2.3 Řídící webová aplikace

Řídící webová aplikace je jediným místem, kde přichází uživatel do styku se systémem Virtlab. K této aplikaci se přistupuje přes běžný webový prohlížeč, jen je potřeba mít nainstalovánu Javu, aby bylo možno pracovat s virtuálními prvky přes konzoli realizovanou pomocí Java appletu. Řídící aplikace obsahuje:

- správu uživatelů dané lokality
- správu úloh
- rezervaci úloh na zvolený čas uživateli dané lokality
- přístup uživatelů dané lokality ke konzolám laboratorních prvků spuštěné úlohy
- další administraci lokality

Řídící webová aplikace je v současné době realizována ve skriptovacím jazyku PHP běžícím na serveru Apache a využívající databázi MySQL. Tato diplomová práce se zabývá reimplementací řídící aplikace.

3 Analýza

3.1 Definice aktérů

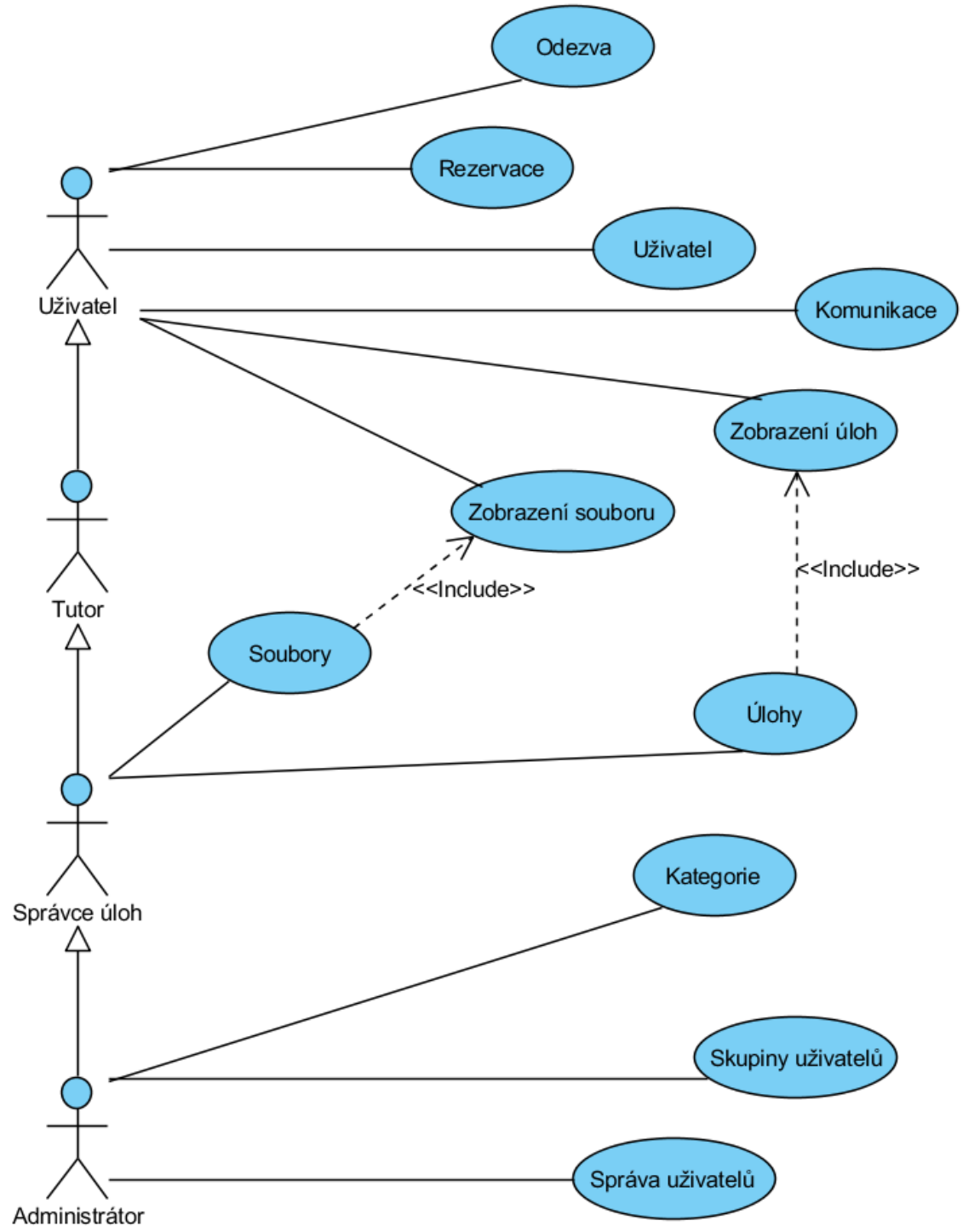
Systém rozlišuje tyto uživatelské role:

- **uživatel** - běžný uživatel (většinou student), zobrazuje si úlohy, ty si dále rezervuje a pracuje na nich
- **tutor** - přistupuje k běžícím úlohám jako tutor - tzn. najednou s běžným uživatelem ovládá stejné zařízení
- **správce úloh** - modifikuje jednotlivé úlohy a s nimi související data
- **administrátor** - uživatel s plným přístupem do všech částí systému

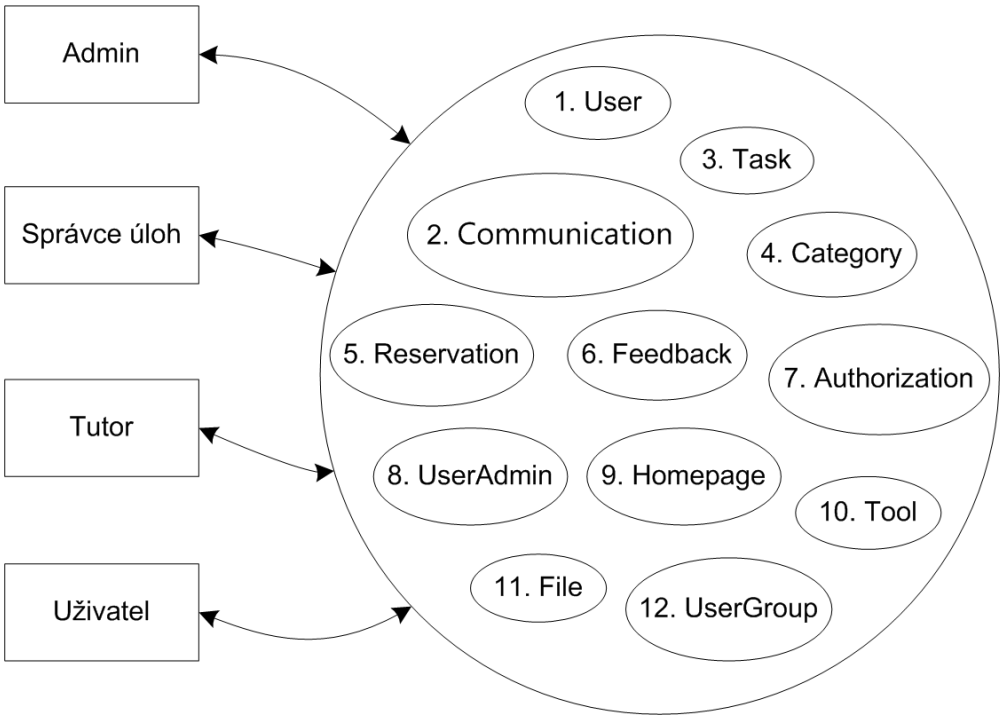
Přístup uživatelů s jednotlivými rolemi k funkcím systému znázorňuje případ užití na obrázku 2.

3.2 Stávající funkce systému

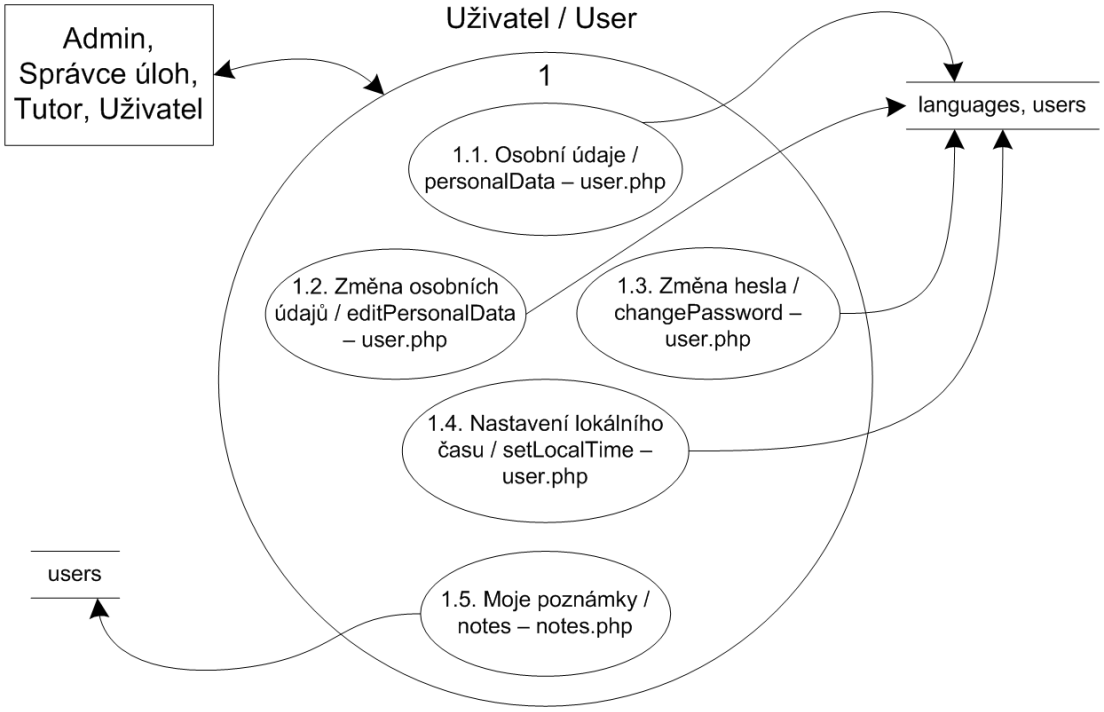
Jednotlivé funkce stávajícího systému zobrazují následující DFD (Data Flow Diagram). U každé funkce je popsán její název, pak navrhovaný název metody v novém systému a soubor, ve se kterém funkce nachází nyní (v aktuální aplikaci určené k reimplementaci).



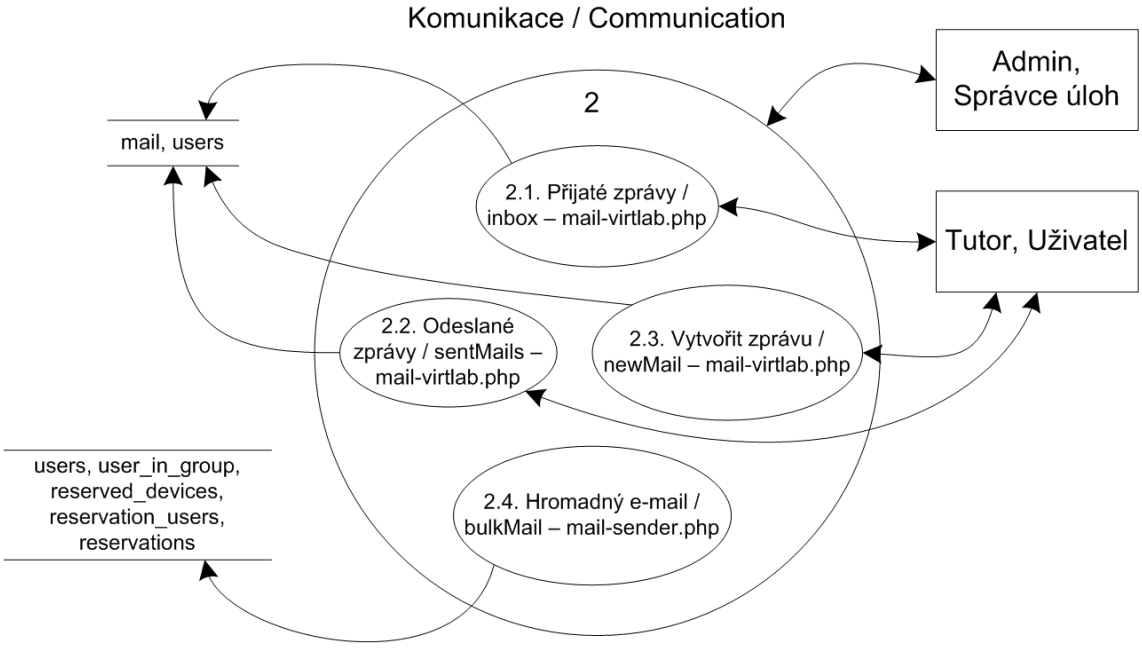
Obrázek 2: Use case



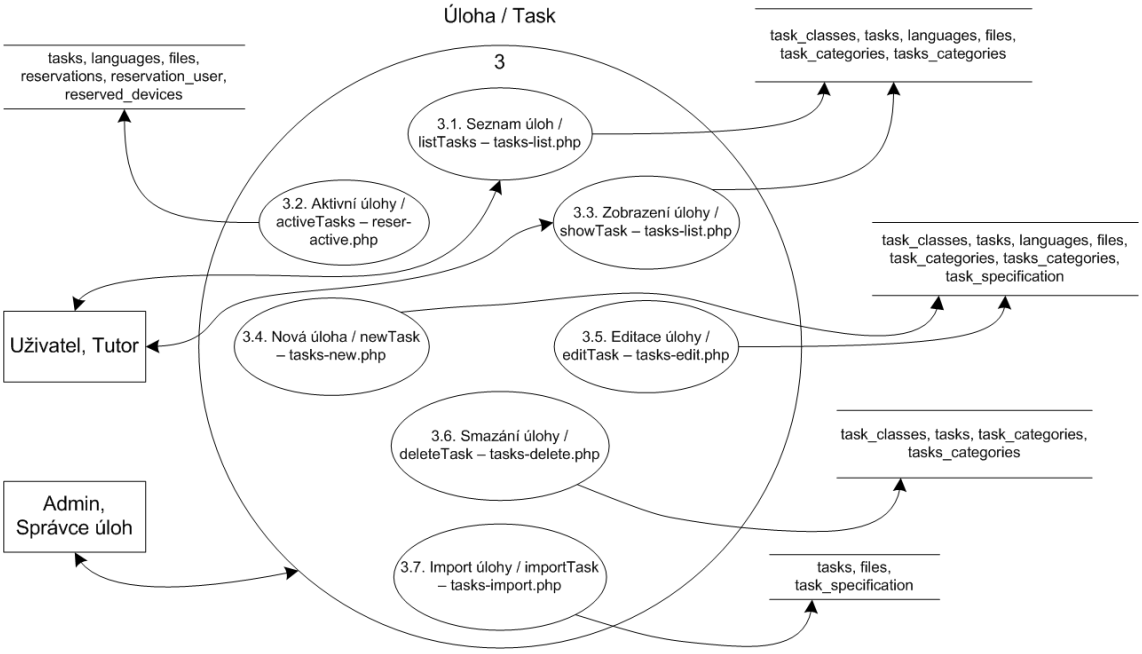
Obrázek 3: DFD nulté úrovně



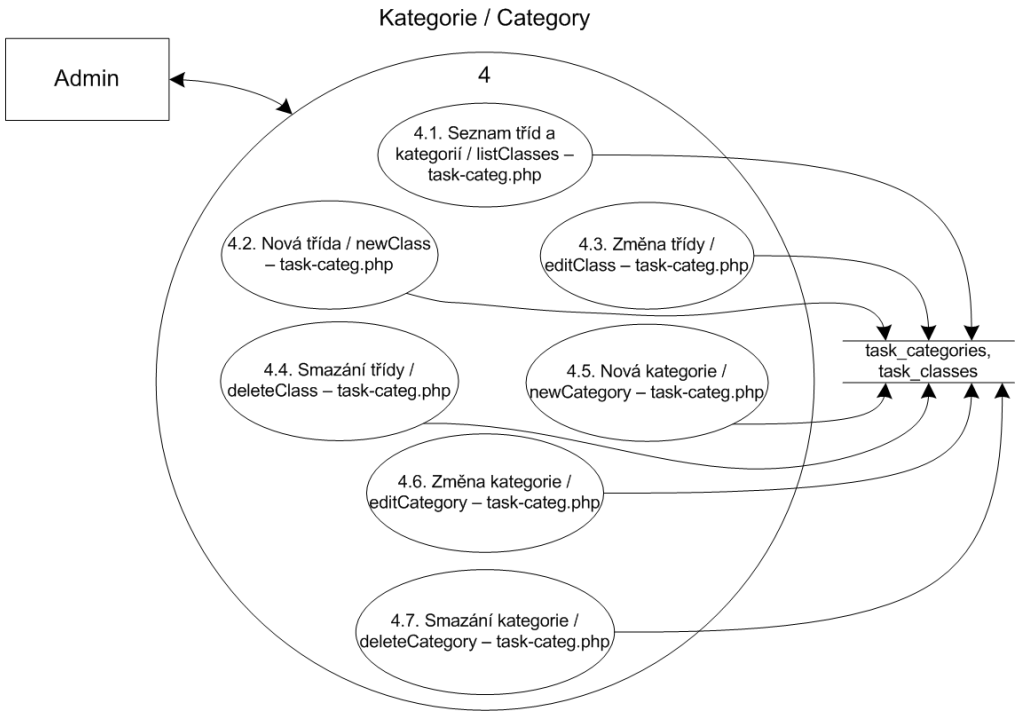
Obrázek 4: DFD 1



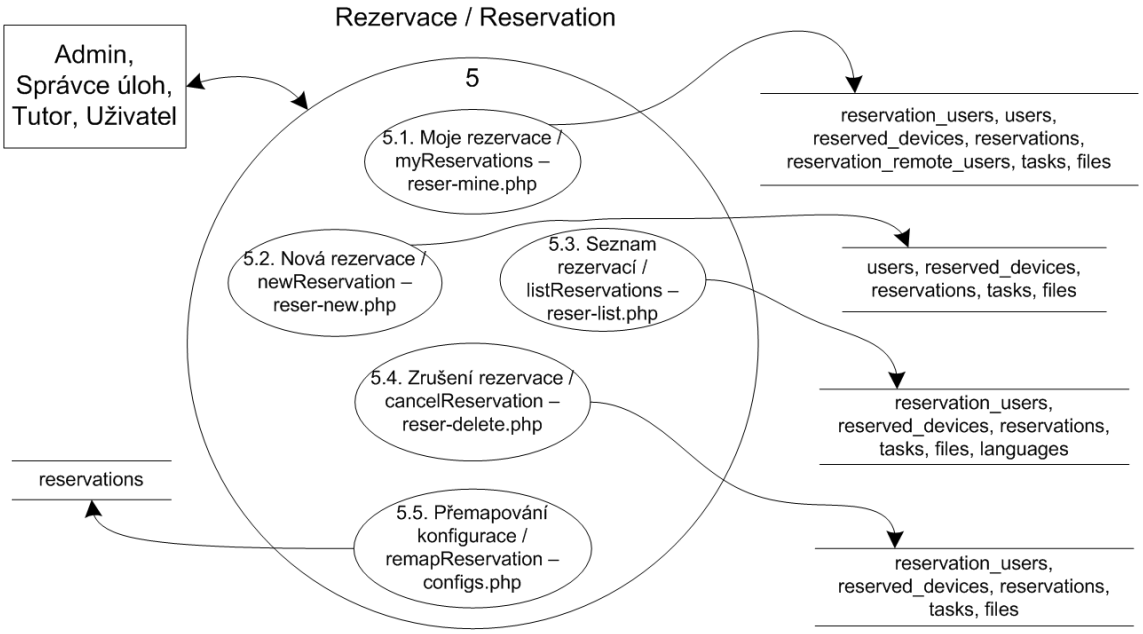
Obrázek 5: DFD 2



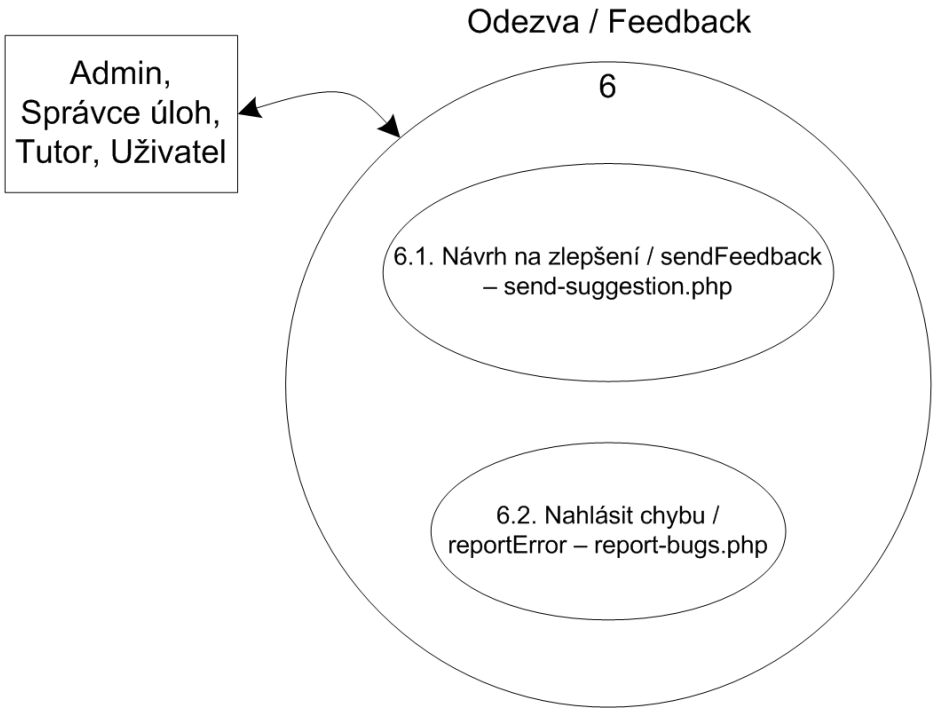
Obrázek 6: DFD 3



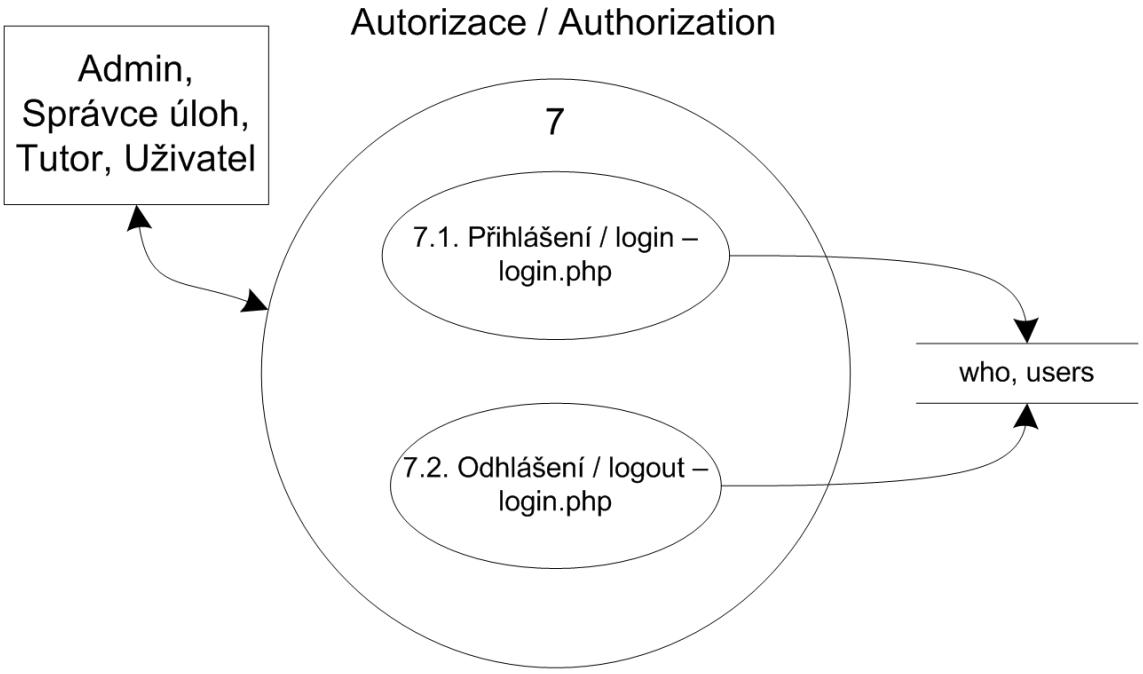
Obrázek 7: DFD 4



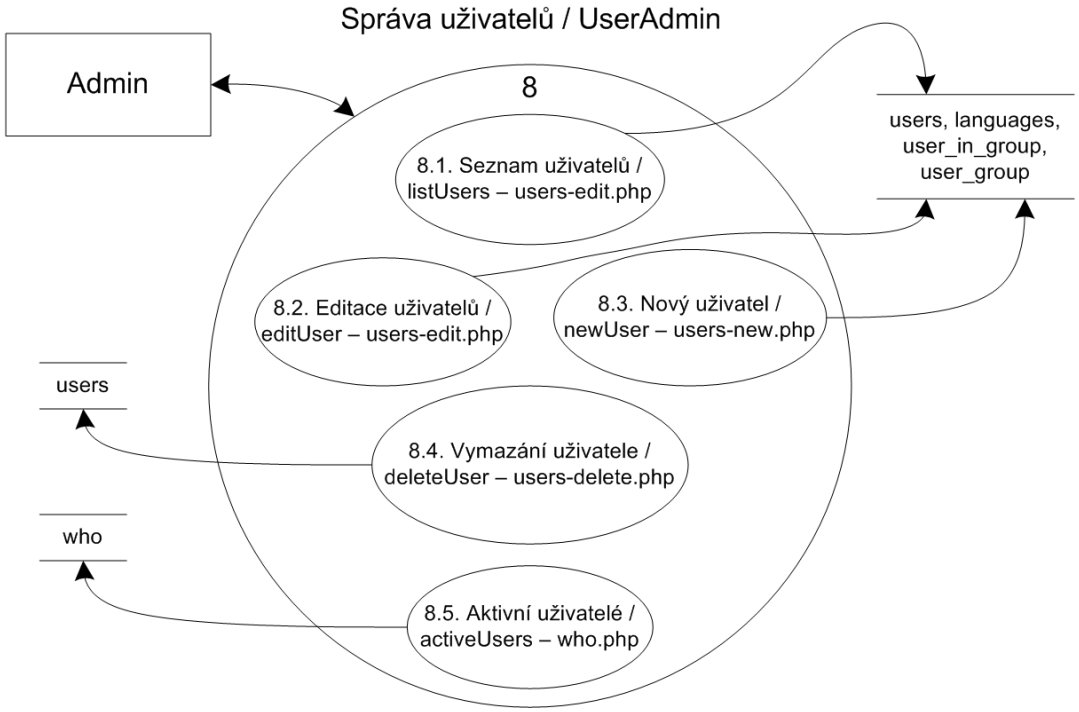
Obrázek 8: DFD 5



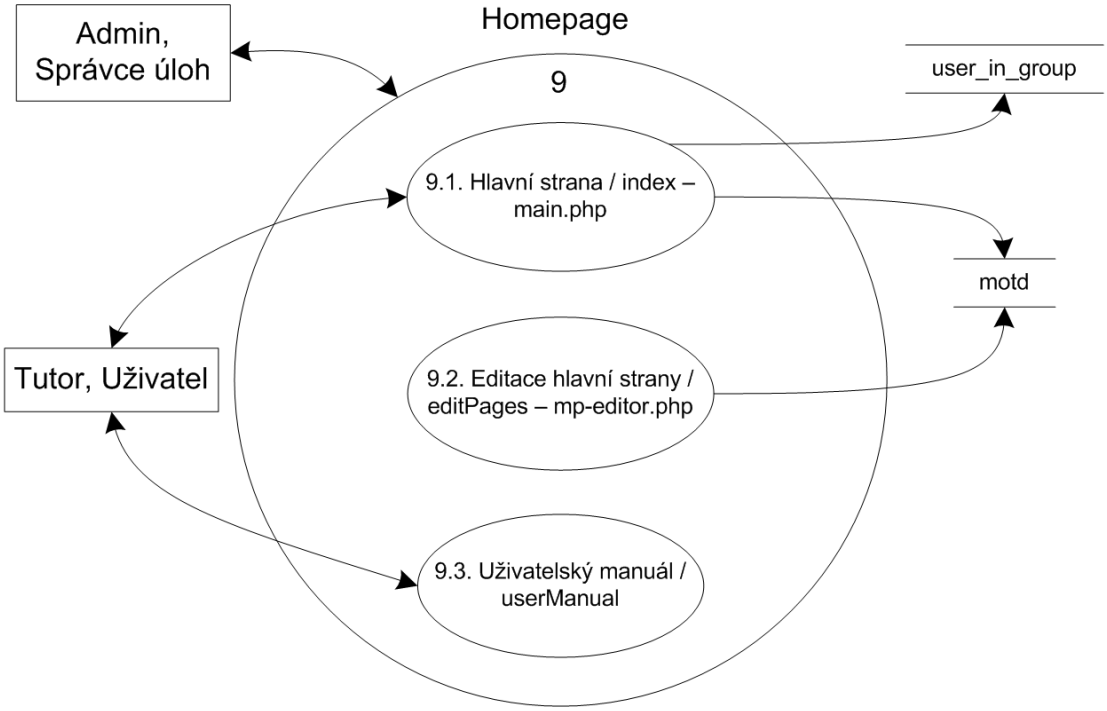
Obrázek 9: DFD 6



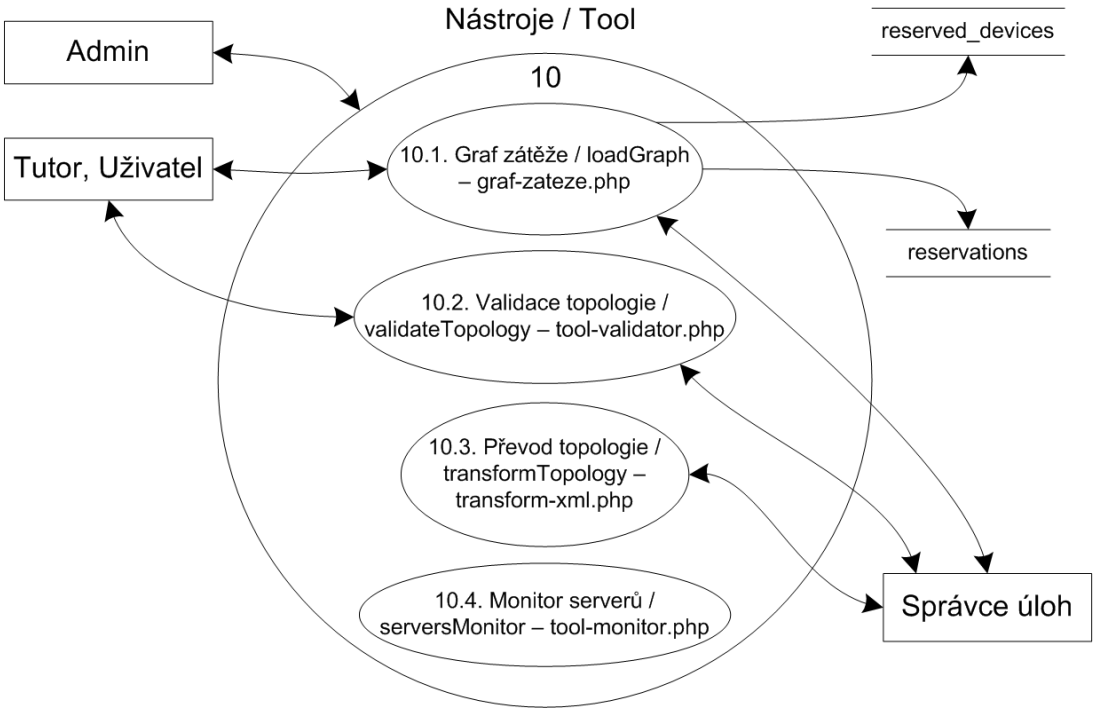
Obrázek 10: DFD 7



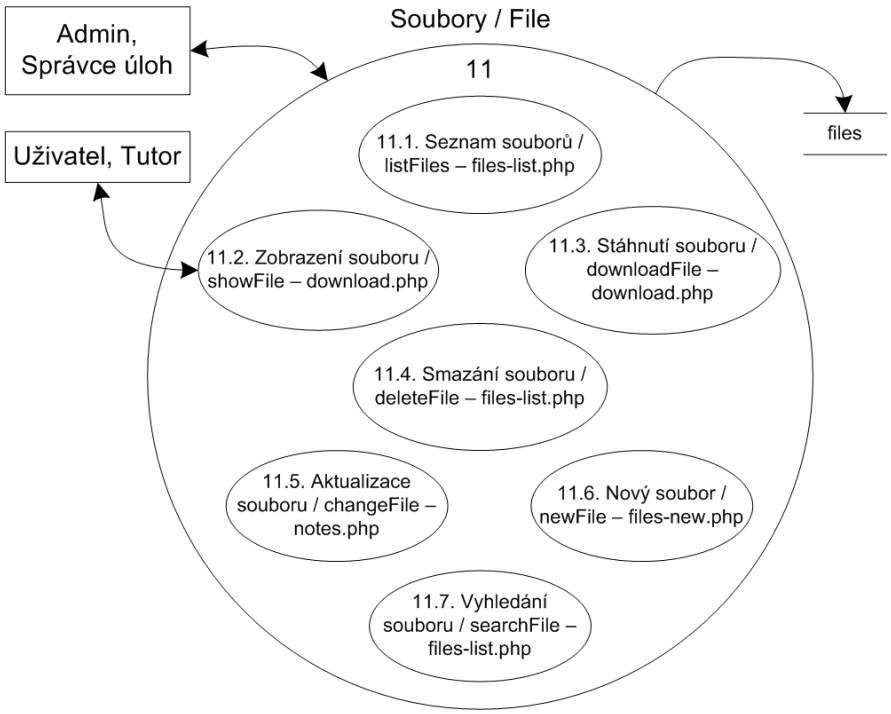
Obrázek 11: DFD 8



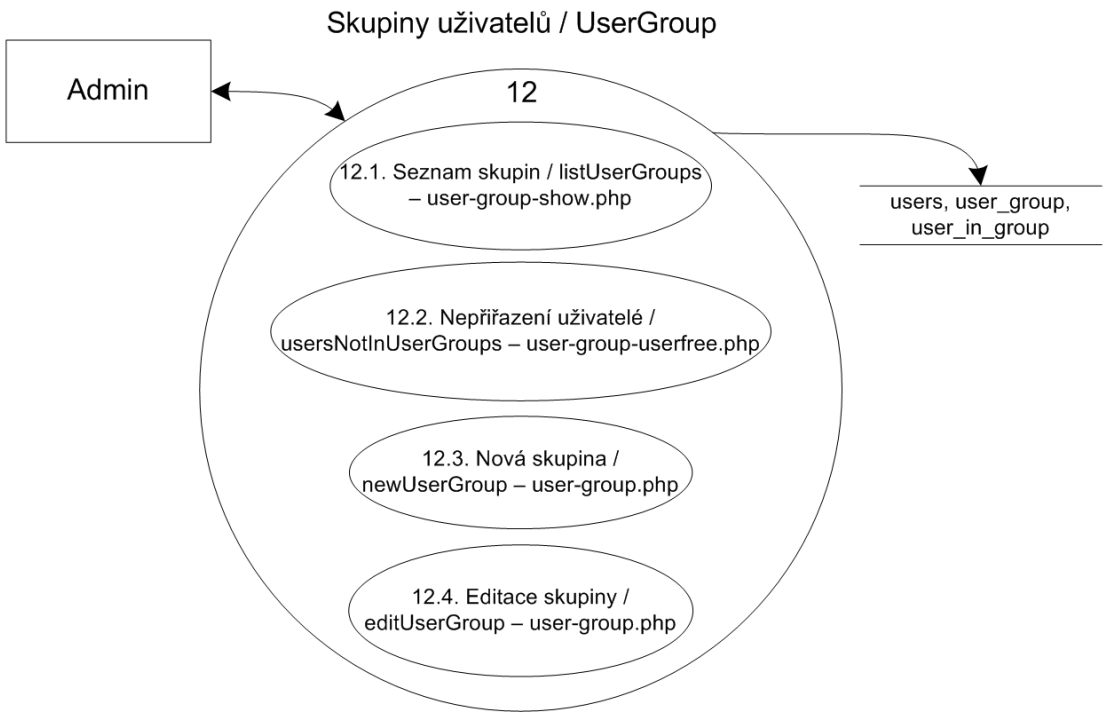
Obrázek 12: DFD 9



Obrázek 13: DFD 10



Obrázek 14: DFD 11



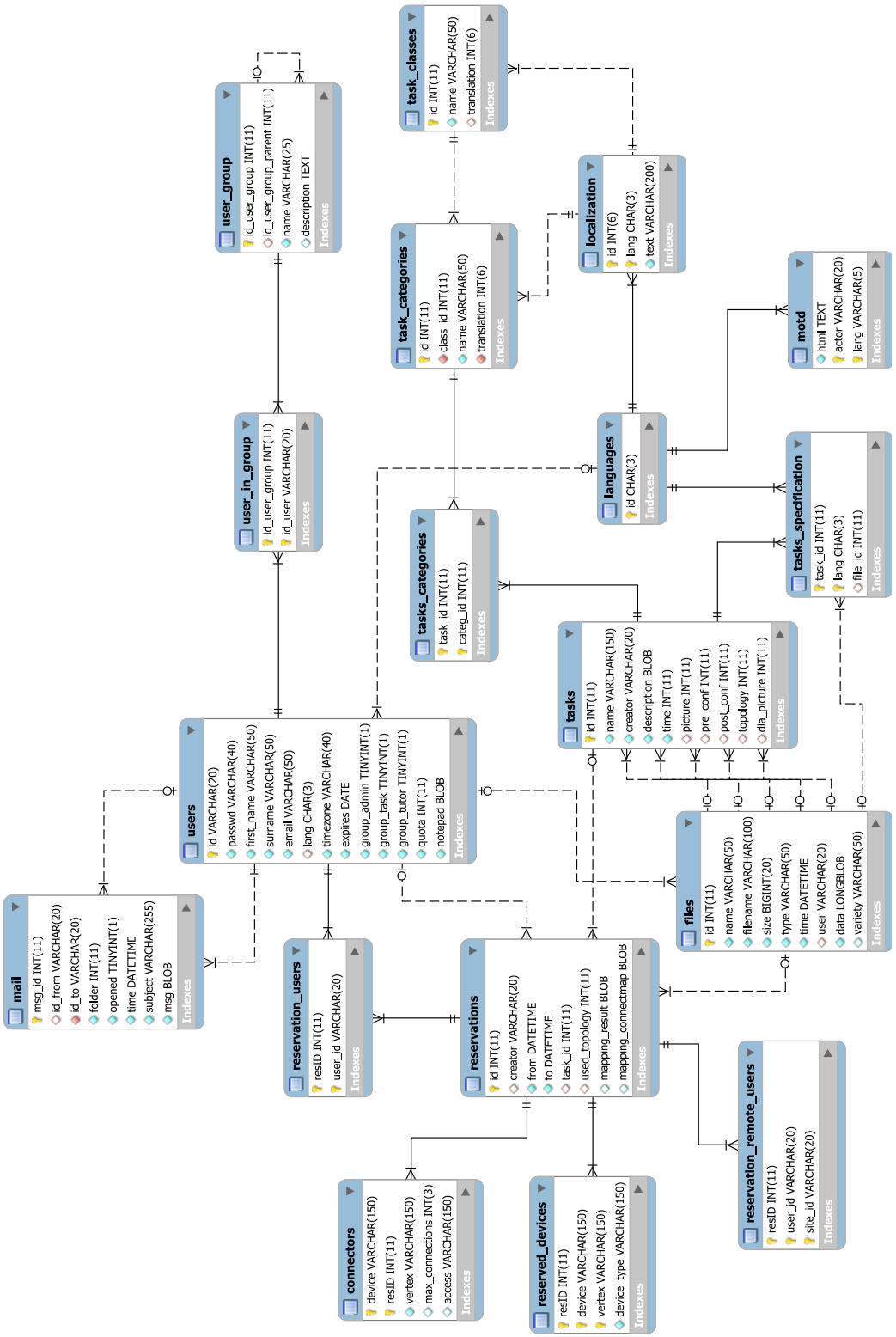
Obrázek 15: DFD 12

3.3 Analýza datové vrstvy

Databáze Virtlabu obsahuje tyto tabulky:

- **files** - soubory použité v úlohách a rezervacích, většinou obrázky topologií, ukázky konfigurací nebo popis zadání
- **languages** - seznam jazyků
- **localization** - překlad různých názvů v databázi do více jazyků
- **mail** - komunikace mezi uživateli systému
- **motd** - Message Of The Day - zprávy zobrazující se na hlavní straně po přihlášení
- **reservation_remote_users** - spolupracovníci v rezervaci z jiných lokalit
- **reservation_users** - spolupracovníci v rezervaci ze stejné lokality
- **reservations** - rezervace na daný čas
- **reserved_devices** - zařízení potřebná pro rezervaci
- **task_categories** - kategorie pro úlohy
- **task_classes** - třídy úloh
- **tasks** - úlohy
- **tasks_categories** - přiřazení úloh do kategorií
- **tasks_specification** - specifikace úloh dle jazyka
- **user_group** - uživatelské skupiny
- **user_in_group** - přiřazení uživatelů do skupin
- **users** - uživatelé a jejich práva

Jednotlivé vztahy mezi tabulkami popisuje ER diagram na obrázku 16.



Obrázek 16: ER diagram databáze

4 Výběr frameworku

4.1 Požadavky na framework

Ještě než přišla na řadu volba frameworku, bylo potřeba vybrat vhodný skriptovací jazyk pro tvorbu webové aplikace. Protože v souvislosti s webovými aplikacemi se dnes stále nejvíce[1] používá jazyk PHP a je tudíž léty prověřený, byl zvolen právě tento. Také byl brán ohled na to, aby mohl být systém dále udržován a rozšiřován dle potřeby dalšími vývojáři. Výběrem skriptovacího jazyka PHP je tato podmínka určitě splněna, protože je jednoduché nalézt programátora se znalostí jazyka PHP.

PHP je velmi rozšířený skriptovací jazyk používaný pro tvorbu dynamických internetových stránek a proto je k dispozici i velké množství frameworků. Při bližším pohledu je většina frameworků v základu stejných a odlišují se jen způsobem, jak jsou již známé funkce implementovány a jakou nabízí framework přidanou hodnotu ve formě dalších rozšiřujících komponent.

Všechny mnou zkoumané frameworky nabízejí tyto funkce, které stojí za povšimnutí:

- použití architektury Model-View-Controller[4]
- validace formulářů (ošetření chyb na vstupu)
- použitím komponenty pro práci s databází nikdy neztrácíme možnost napsat ručně celý SQL dotaz
- autentizace a autorizace uživatelů
- routování - mapování URL na jednotlivé controllery a akce v nich
- caching - ukládání dat do vyrovnávací paměti (např. pro snížení zátěže databáze při složitých dotazech), cache se obvykle zapíná ručně pro vybrané objekty nebo proměnné

Jelikož základní požadavky splňují všechny frameworky, tak jsem se při průzkumu zaměřil spíše na přidanou hodnotu a jak se celkově s frameworkem pracuje. Hledal jsem takový, který bude vše potřebné již obsahovat a pro základní práci nebude potřeba hledat a instalovat další komponenty. Zkoumal jsem proto tyto body:

- šablonovací systém - je opravdu nepřehledné, když se mezi sebou míchá HTML a PHP. Také nesmíme zapomenout, že na vytvoření webové aplikace se podílí více profesí - minimálně programátor a grafik. Grafik/kodér[2] se má primárně starat o vizuální podobu a neměl by se učit dynamickému chování naší aplikace - proto je potřeba oddělit HTML od PHP formou jednoduše pochopitelného šablonovacího systému.

[1] dle statistik[3] z února 2010 pro české domény je nejvíce zastoupeno PHP (38% domén) a .NET (14% domén), ostatní jazyky používané na webových stránkách mají zanedbatelný podíl

[2] kodér je označení pro člověka, který vytváří HTML a CSS kód (většinou má jako podklad grafiku rozdělenou do vrstev)

- rozumná práce s datovou vrstvou - skoro všechny frameworky podporují postupné skládání dotazů objektovým způsobem, pomocí kterého se zpřehlední SQL dotaz a také se předejde případným bezpečnostním rizikům v podobě SQL injection
- podpora transakcí - je výhodné, pokud mohu umístit transakci do bloku try-catch a nemusím toto řešit pouze na úrovni databáze přes SQL dotaz
- profiler - v rámci optimalizace aplikace je potřebný profiler, který vypisuje poslední spuštěné dotazy na databázi a také čas potřebný ke zpracování těchto dotazů

4.2 Seznam kandidátů

U každého posuzovaného frameworku je uveden graf získaný z testů PHP frameworků na serveru Root[5] (uvádím data bez použití eAcceleratoru).

4.2.1 Kohana [6]

Vychází z CodeIgniteru[12], přepsáno do OOP a vyvíjeno komunitou (CodeIgniter je vytvářen firmou, ale je poskytován zdarma).
Klady:

- pro práci s datovou vrstvou lze použít ORM
- obsahuje profiler (zobrazuje čas zpracování, využitá paměť, doba trvání SQL dotazů a počet ovlivněných řádků v databázi)

Zápory:

- chybí podpora transakcí
- nepoužívá šablonovací systém

Ukázka práce s datovou vrstvou:

```
$post = ORM::factory('blog_post', 1);

echo $post->user->username;
```

4.2.1.1 Podpora více jazyků

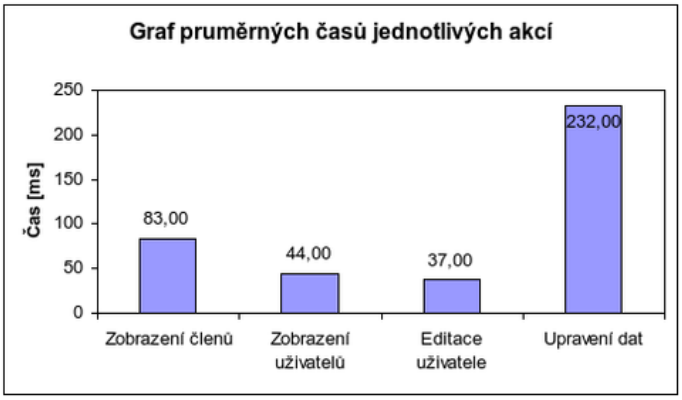
Jednotlivé texty jsou uloženy v poli v příslušném jazykovém souboru.

Obsah souboru coffee.php:

```
$lang = array
(
    'cup'    => 'Coffee goes in here',
    'beans'  => 'Coffee is created from beans.',
    'java'   => 'Place where coffee comes from, not the programming language',
);
```

Použití:

```
echo Kohana::lang('coffee.cup');
echo Kohana::lang('coffee.beans');
echo Kohana::lang('coffee.java');
```



Obrázek 17: Výsledky testů pro framework Kohana. Zdroj: [5]

4.2.2 Zend [7]

Klady:

- obsahuje několik modulů pro práci s PDF, LDAP, SOAP, službami Google, generování čárového kódu a další
- propracovaný profiler
- podpora transakcí
- obsáhlá dokumentace, velká komunita

Zápory:

- dle testů nejpomalejší framework
- šablonovací systém není součástí frameworku, je doporučováno si stáhnout Smarty

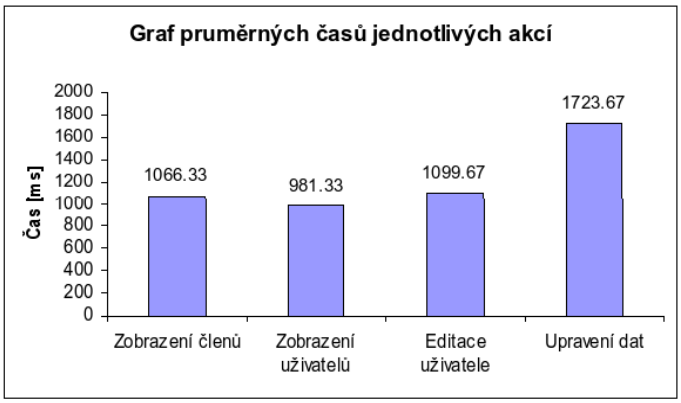
Ukázka práce s datovou vrstvou:

```
$select = $db->select()
              ->from(array('p' => 'products'),
                     array('product_id'))
              ->join(array('l' => 'line_items'),
                     'p.product_id = l.product_id',
                     array('line_items_per_product' => 'COUNT(*)'))
              ->group('p.product_id');
```

4.2.2.1 Podpora více jazyků Lokalizace je vyřešena komplexně - datum, čas, formát čísel... Soubory s jednotlivými jazykovými řetězci mohou být v několika formátech - ini, xml, ts (formát knihovny QT) nebo přímo výstup gettextu - mo soubor.

View pak může vypadat třeba takto:

```
$translate = new Zend_Translate('gettext', '/my/path/source-de.mo', 'de');
$translate->addTranslation('//my/path/fr-source.mo', 'fr');
print $translate->("Example")."\n";
print "=======\n";
print $translate->("Here is line one")."\n";
printf($translate->("Today is the %1\$s") . "\n", date("d.m.Y"));
print "\n";
$translate->setLocale('fr');
print $translate->("Fix language here is line two") . "\n";
```



Obrázek 18: Výsledky testů pro framework Zend. Zdroj: [5]

4.2.3 Nette [8]

Klady:

- vylepšuje samotné PHP (všechny třídy dědí z Nette\Object, který upravuje nekonzistentní generování chyb, přidává gettery a settery, reflexi atd.)
- skvěle vyřešené ladění aplikace (při vypisování chyby ukazuje zdrojový kód s označeným řádkem, na kterém je chyba)
- obousměrný router (definice URL jen na jednom místě, View se při tvorbě odkazu ptá routeru na URL)
- podle napsaných pravidel se u formulářů generuje validace jak na straně serveru, tak i u klienta (Javascript)
- podpora transakcí

- velmi dobrý profiler
- obsahuje šablonovací systém
- velmi aktivní komunita
- velký důraz na bezpečnost celé aplikace

Ukázka práce s datovou vrstvou (pomocí externí knihovny Dibi[14] od stejného autora):

```
$res = dibi::select('product_id')->as('id')
        ->select('title')
        ->from('products')
        ->innerJoin('orders')->using('(product_id)')
        ->orderBy('title')
        ->execute();
```

Ukázka šablony:

```
<h2>Table „{$table}“</h2>

<p><a href="{plink default}">Return to home</a></p>

<table class="grid" border="1">
<tr>
    {foreach $columns as $column}
    <th>{$column}</th>
    {/foreach}
</tr>

{foreach $rows as $num => $row}
<tr{!=$num % 2 ? ' class="alt"' : ''}>

    {foreach $row as $value}
    <td>{=substr($value, 0, 100)}</td>
    {/foreach}

</tr>
{/foreach}
</table>
```

4.2.3.1 Podpora více jazyků K lokalizaci je možné použít doplněk GettextTranslator - Gettextový překladač, který pracuje s binárními soubory *.mo* (překlad je definován v souborech *.po* a ty jsou pak programem Gettext[13] přeloženy do binární podoby). Nette má podporu lokalizací v šablonách i formulářích, jen se mu předá objekt tohoto překladače (případně jiné komponenty nebo vlastní třídy řešící překlad).

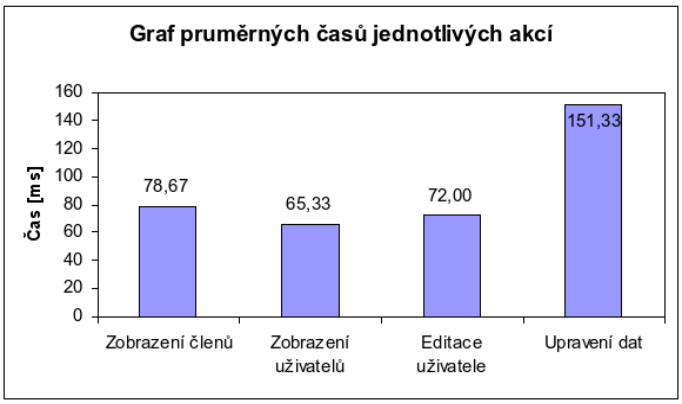
Lokalizace formuláře:

```
$translator = new GettextTranslator('form.cs.mo', 'cs');
$form = new Form;
$form->setTranslator($translator);
```

Lokalizace řetězců v šabloně:

```
{? $template->setTranslator(new GettextTranslator('template.cs.mo'))}

{_'Hello! Welcome to our page!'}
{_'There is %d unreaded comment in thread %s.', $number, 'Changelog'}
{_'In thread %2$s is %1$d unreaded comment.', $number, 'Changelog'}
```



Obrázek 19: Výsledky testů pro framework Nette. Zdroj: [5]

4.2.4 FUSE [9]

Klady:

- obsahuje ORM
- podpora transakcí
- má šablonovací systém

Zápory:

- nedostatečná dokumentace
- neobsahuje profiler
- lze použít jen s databází MySQL
- dle testů velmi pomalá úprava dat (zřejmě špatně naimplementované ORM)

Ukázka práce s datovou vrstvou:

```
FUSE::require_model('Company');

$company = new Company();
$company->id = 100;
```

```
$iterator = $company->franchises->fetch_all();

while ($franchise = $iterator->next()) {
    echo "Franchise id is: {$franchise->id}, name is: {$franchise->name}<br/>\n";
}
```

Ukázka šablony:

```
<{ITERATOR all_news}>

News title: <{title}><br />
News date: <{date}><br />
News entry: <{entry}><br />

<{/ITERATOR}>
```

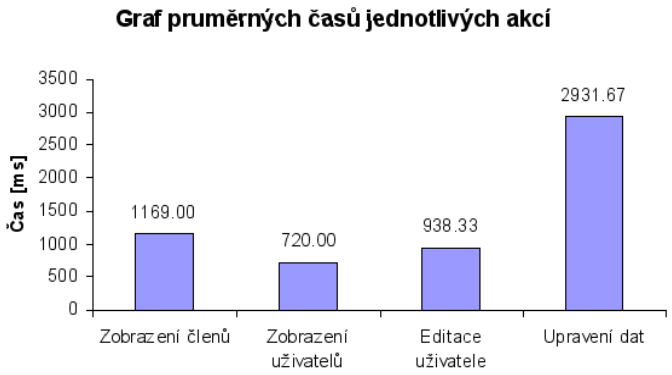
4.2.4.1 Podpora více jazyků

Řetězce jsou uloženy v daném jazykovém souboru v poli:

```
$widget_messages['id_invalid'] = 'The ID %1 is NOT valid';
$widget_messages['id_valid'] = 'The ID %1 is valid';
```

Možné použití:

```
if ( $this->id && is_numeric($this->id) ) {
    $this->message = "widget-id_valid %{$this->id}%";
}
else {
    $this->message = "widget-id_invalid %{$this->id}%";
}
```



Obrázek 20: Výsledky testů pro framework FUSE. Zdroj: [5]

4.2.5 PRADO [10]

Klady:

- ORM s podporou několika databází
- podpora transakcí
- šablonovací systém (styl zápisu připomíná .NET)

Zápory:

- nedostatečná dokumentace
- nemá profiler
- dle testu pomalé

Ukázka práce s datovou vrstvou:

```
$criteria  = new TActiveRecordCriteria;
$criteria->Condition = 'username = :name AND password = :pass';
$criteria->Parameters[':name'] = 'admin';
$criteria->Parameters[':pass'] = 'prado';
$criteria->OrdersBy['level'] = 'desc';
$criteria->OrdersBy['name'] = 'asc';
$criteria->Limit = 10;
$criteria->Offset = 20;

$finder  = UserRecord::finder();
$finder->find($criteria);
```

Ukázka šablony:

```
<com:TView ID="WinView">
<h2>You Win!</h2>
<p>The word was: <%= $this->Page->Word %>.</p>
<p><com:TLinkButton Text="Start Again" OnClick="startAgain" /></p>
</com:TView>
```

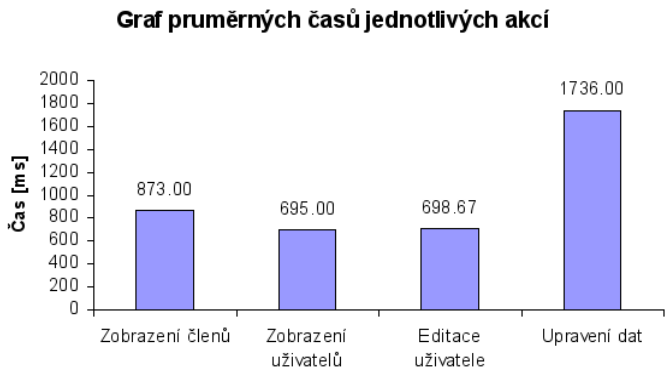
4.2.5.1 Podpora více jazyků Řetězce jsou uloženy v XML nebo v databázi. Lokalizace řeší kromě překladu textů i datum, čas, formát čísel nebo podobu výběrového seznamu (např.: 1 - One Apple, 2 - Two Apples, ...) a další.

V šabloně se může objevit třeba toto:

```
<com:TTranslate Text="Goodbye" />

<com:TLabel Text="<%[ Hello World! ]%>" />

<com:TNumberFormat Type="currency" Culture="en US" Currency="EUR" Value="100" />
```



Obrázek 21: Výsledky testů pro framework PRADO. Zdroj: [5]

4.2.6 CakePHP [11]

Klady:

- používá ORM
- také má obousměrný router
- validační pravidla definované v modelu lze využít při tvorbě formulářů

Zápory:

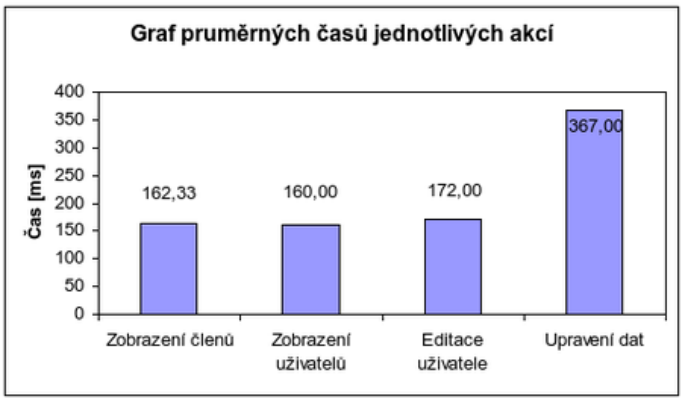
- nemá profiler
- podporu pro transakce je nutno dopsat
- neobsahuje šablonovací systém

```
$condition = array(
    'conditions' => array('Model.Active =' => 1),
    'fields' => array('DISTINCT Model.Client', 'Model.Job', 'Model.Description'),
    'order' => array('Model.Client'),
    'group' => array('Model.Client', 'Model.Job', 'Model.Description'));

$this->Model->find('all', $condition);
```

4.2.6.1 Podpora více jazyků CakePHP také řeší lokalizaci přes Gettext, překlad se provádí voláním funkce __():

```
<h2><?php __('Posts') ?></h2>
```



Obrázek 22: Výsledky testů pro framework CakePHP. Zdroj: [5]

4.3 Vyhodnocení

Začnu od nejslabšího frameworku - FUSE. Tady převládají zápory - ORM je spíše na obtíž, protože špatnou implementací velmi znatelně zpomaluje práci s daty. PRADO vidím spíše jako pokus přepsat ASP.NET do PHP - nedokážu si ovšem představit, že bych ručně psal definici formuláře v této syntaxi (při práci v .NET lze celý formulář naklikat v návrháři, ale tady je potřeba si vše pamatovat a ručně psát tento kód do šablony). Chabou dokumentaci k PRADO se snaží trochu zachránit tutoriály, ale bohužel v nich nenalezneme vše. Zend nabízí opravdu velkou škálu funkcí, ale bohužel se z něho stal pomalý moloch. CakePHP a Kohana vypadají už použitelně, na škodu je absence šablonovacího systému. Nejen proto si nakonec vybírám Nette - přijde mi totiž hodně inovativní a svými myšlenkami vyčnívá mezi desítkami PHP frameworků. V následující kapitole se pokusím popsat hlavní přednosti Nette a myslím, že mi dáte za pravdu.

4.4 Nette Framework

Různé inovace tohoto frameworku začínají už u samotného jazyka PHP - tento jazyk totiž nebyl od začátku objektový a podpora OOP se začala nabalovat postupem vývoje. Objekty jsou tedy jen takové rozšíření a PHP se vůči nim chová nekonzistentně při hlášení chyb - některé závažné chyby jsou hlášeny pouze jako Notice. Navíc při chybě se normálně zobrazí stránka a nepošle se HTTP 500, takže vyhledávače tyto špatně vygenerované stránky zaindexují. Tyto a další věci právě Nette v základu řeší, navíc přidává podporu getterů, setterů a extension method (známé z C#), dále vylepšuje reflexi a události.

Při práci s různými třídami frameworku je není potřeba ručně volat pomocí `require` - v Nette je obsažen RobotLoader, který automaticky načítá třídy až ve chvíli, kdy jsou skutečně potřeba. Navíc se tímto odhalí konflikty názvů a připadáte si jako v kompilovaném jazyce.

Jedna z nejčastějších věcí při vývoji složitějších systému je psaní formulářů a v Nette je práce s formuláři implementována velmi dobře. Zápis v kódu je velmi jednoduchý a podle něj je vygenerováno správné HTML s ohledem na přístupnost, dále je dle pravi-

del vygenerována validace na straně serveru i klienta. Javascriptová validace na straně klienta oznamuje chyby ve špatně vyplněném formuláři okamžitě a působí na uživatele lépe než opětovné načítání stránky s popisem chyby. Validace na straně klienta se musí většinou psát ručně a neposkytuje ji skoro žádný framework, proto má v tomto směru Nette velký náskok.

Oblast ladění je v Nette podchycena naprosto výborně - obsahuje třídu Nette\Debug, která při chybě zobrazuje kompletní backtrace, zdrojový kód se zvýrazněným řádkem s chybou a také výpis všech proměnných. Podobnou knihovnu neobsahuje žádný framework, kterými jsem se zabýval. V produkčním prostředí se samozřejmě takto chyby nevypisují a jsou logovány na server nebo posílány e-mailem.

Za zmínku určitě stojí i šablonovací systém. Ten používá podobnou syntaxi jako řada známých šablonovacích systémů, např. Smarty. Nejdůležitější vlastnost tvoří kontextově-sensitivní escapování - všechny proměnné vkládané do šablony se automaticky escapují (znaky mající v daném kontextu speciální význam se převádění na jiné odpovídající sekvence) podle kontextu, v jakém jsou vkládány (HTML, JavaScript, URL, CSS a XML). Používání různých proměnných v šablonách je tak bezpečné a nemůže dojít k útoku typu Cross-site scripting (XSS).

Dále framework obsahuje podporu AJAXu, sessions jako jmenné prostory, autentizace a autorizace, cache a tak dále, je toho ještě hodně. Pro více informací vám doporučuji seriál na Root[15] nebo přímo stránky Nette[8].

Také je příjemné, že Nette i Zend jsou modulární, takže není problém použít třídu ze Zendu v Nette, pokud bude potřeba doplnit funkcionalitu o PDF, LDAP, SOAP atd.

O uchycení tohoto mladého frameworku na trhu svědčí časté přednášky (v listopadu 2009 přednášel David Grudl i na VŠB-TU Ostrava) a školení pořádaná přímo autorem a také nasazování Nette do provozu.

4.4.1 Kdo používá Nette Framework

- Mladá fronta, a. s. (www.mfplus.cz, www.e15.cz)
- Internet Info, s.r.o. (www.root.cz, www.mesec.cz)
- Václav Klaus (www.klaus.cz)
- VLTAVA-LABE-PRESS, a.s.
- a spousta dalších webových stránek...

Stránky používající Nette Framework lze poznat tak, že posílají v HTTP hlavičce atribut `X-Powered-By: Nette Framework`

5 Datová vrstva

5.1 Požadavky na datovou vrstvu

Jelikož Nette Framework je nezávislý na datové vrstvě, tak bylo potřeba také vybrat vhodný nástroj pro práci s databází. Především byl kladen důraz na objektový přístup a vyhnutí se přímému psaní SQL dotazů, proto byl výběr zúžen pouze na ORM nástroje (ORM - Object Relational Mapping - metoda mapování relační databáze na objekty).

Mezi další požadavky patřila hlavně univerzálnost napojení na různé databáze, což většina ORM nástrojů splňuje. Dále byla zohledněna syntaxe při práci s objekty - zejména jakým způsobem se získávají data a jak se realizují vazby mezi tabulkami.

5.2 Seznam ORM nástrojů

Vybral jsem pár představitelů ORM pro PHP. Ve skutečnosti jsem jich prozkoumal více, ale většinou jsem narazil na nedostatečnou dokumentaci nebo nepropracovanou funkcionalitu.

5.2.1 Doctrine [16]

- výborná dokumentace
- umí generovat modely ze stávající databáze
- podle nastavení modelu jsou data validována (při nastavení atributu *email* je kontrolována i existence domény přes DNS !), možno napsat i vlastní validátory
- k dotazování se používá DQL (Doctrine Query Language), což je jen objektové psaní SQL dotazů:

```
$q = Doctrine_Query::create()
    ->select('u.username, p.*')
    ->from('User u')
    ->leftJoin('u.Phonenumbers p');
```

- jen pro jednoduché věci (což je škoda) lze použít *Magic Finders*:

```
$users = $userTable->findByName("Jack");
```

- obsahuje i profiler
- podpora transakcí

Ukázka modelu:

```
class User extends Doctrine_Record
{
    public function setTableDefinition ()
    {
```

```
        $this->hasColumn('username', 'string' 255);
        $this->hasColumn('password', 'string', 255);
    }

    public function setPassword($password)
    {
        return $this->_set('password', md5($password));
    }
}
```

Ukázka vložení nového záznamu:

```
$product = new Product();
$product->name = 'žehlička';
$product->price = 150;
$product->categoryId = 2;
$product->save();
```

5.2.2 Propel [17]

- popis modelu pomocí XML, pak se podle něj vygeneruje SQL a PHP

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>

<database name="bookstore" defaultIdMethod="native">
 <table name="author" description="Author Table">
  <column name="author_id" type="integer" required="true" primaryKey="true" autoIncrement
      ="true" description="Author Id"/>
  <column name="first_name" type="varchar" size="128" required="true" description="First
      Name"/>
  <column name="last_name" type="varchar" size="128" required="true" description="Last
      Name"/>
 </table>
</database>
```

- validace dat (definice v XML)
- podpora transakcí
- syntaxe psaní dotazů na databázi mě moc nenadchla:

```
$c = new Criteria();
$c->add(AuthorPeer::FIRST_NAME, "Karl");
$c->add(AuthorPeer::LAST_NAME, "Marx", Criteria::NOT_EQUAL);

$authors = AuthorPeer::doSelect($c);


$c = new Criteria(AuthorPeer::DATABASE_NAME);

$c->addJoin(AuthorPeer::ID, BookPeer::AUTHOR_ID, Criteria::INNER_JOIN);
$c->addJoin(BookPeer::PUBLISHER_ID, PublisherPeer::ID, Criteria::INNER_JOIN);
```

```
$c->add(PublisherPeer::NAME, 'Some Name');

$authors = AuthorPeer::doSelect($c);
```

Ukázka vložení nového záznamu:

```
$author = new Author();
$author->setFirstName("Leo");
$author->setLastName("Tolstoy");

$book = new Book();
$book->setTitle("War & Peace");
$book->setIsbn("0140444173");
$book->setAuthor($author);
$book->save();
```

5.2.3 EZPDO [18]

- poslední verze z roku 2007
- model nelze vygenerovat
- podpora transakcí
- pro dotazování zavádí svůj jazyk EZOQL:

```
$m = epManager::instance();
$books = $m->find("from Book as b where b.title like ? order by b.uuid desc", $title);
```

- pro jednodušší věci lze použít i takovéto vyhledávání:

```
$book2find = $m->create('Book');
$book2find->author = "ezpdo4php";

$books = $m->find($book2find);
```

Ukázka modelu:

```
class Author extends Base {
    /**
     * Name of the author
     * @var string
     * @orm char(64)
     */
    public $name;

    /**
     * Books written by the author
     * @var array of Book
     * @orm has many Book
```

```
     */
    public $books = array();

    /**
     * Constructor
     * @param string $name author name
     */
    public function __construct($name = '') {
        parent::__construct();
        $this->name = $name;
    }
}
```

5.2.4 RedBean [19]

- velice zajímavá funkčnost, kterou jsem viděl poprvé - není potřeba psát žádné modely nebo tabulky v databázi, prostě se v kódu napíše nová třída, zadají jednotlivé atributy a vše se automaticky vygeneruje, dokonce se mění typy sloupců a přidávají sloupce další podle toho, jaká data do třídy uložím:

```
// vygeneruje se nová tabulka BlogPost
R::gen("BlogPost");

$post = new BlogPost();
$post->title = " Title ";
$post->description = "description";
// vytvoří dva nové sloupce v tabulce a uloží záznam
$post->save();
```

- moc hezké, člověk ani nemusí vědět, co je databáze. Ale nejsem si jistý, jestli by se to dalo použít pro naše účely - modely sice lze napsat ručně, ale dle mého názoru hrozí, že to bude měnit sloupce v databázi pod rukama (je možné, že tomu lze nějak zabránit, do hloubky jsem to nestudoval)
- takto vypadá vyhledávání:

```
$currentPost = new BlogPost( $id );
$comments = $currentPost->getRelatedComment();

// a trochu složitější :
$customer = new Customer;
$customer->setGender('f');
$customer->setAge(50);
$oldladies = Customer::find( $customer, array( "age"=>">","gender"=>"=" ) );
```

5.2.5 CoughPHP [20]

- modely jsou generované podle databáze
- podpora validace dat a transakcí

- obsahuje profiler

```
// tabulka author má atribut most_popular_book_id, což je cizí klíč k˘tabulce book
// dotaz udělám jednoduše:
$book = $author->getMostPopularBook_Object();
// pak přistupuju k˘atributům takto:
echo $book->getTitle();

// takto vypadá složitější dotaz many-to-many:
$book = Book::constructByKey(1);
$joins = $book->getAuthoredBook_Collection();
$joins->sortByMethod('getAuthorSortOrder', SORT_ASC);
foreach ($joins as $join) {
    $author = $join->getAuthor_Object();
}
```

5.2.6 Ormion [21]

- postaveno nad Dibi[14], takže dědí její pozitiva jako je podpora několika databází, transakcí, profiler atd.
- využívá novinek PHP 5.3
- velmi jednoduchá syntaxe:

```
// vyhledání záznamů podle určitých hodnot
$articles = Article :: findAll (array(
    "author" => 4,
    "allowed" => true,
));

// nebo alternativně
$articles = Article ::findAllByAuthorAndAllowed(4, true);

// vytvoření záznamu
$page = new Page;
$page->name = "Nová";
$page->url = "nova";
$page->text = "Nová stránka";
$page->save();
```

- s kolekcí lze dále pracovat podobně jako s třídou DibiFluent:

```
$articles = Article :: findAll ()
    ->where("[author] = %i", 4)
    ->orderBy("[date] DESC")
    ->limit(10)
    ->offset(0);
```

5.3 Vyhodnocení

Bohužel pro PHP neexistuje mnoho použitelných ORM nástrojů, spíše se čekalo na PHP 5.3, které nabízí lepší práci s objekty (např. jmenné prostory). Velmi dobře vypadá Doctrine, ale práce s modely není dle mých představ - toto řeší verze 2.0 (postavená na PHP 5.3), která je v současné době ve vývoji. Další reálný kandidát byl CoughPHP, byl mi doporučen členem Virtlabu. Po větším testování jsem však narazil na velké problémy při práci s kolekcemi a musel jsem od používání CoughPHP upustit. Navíc se vývoj zastavil ke konci roku 2008 a další verze již zřejmě nebude. Rozhodl jsem se tedy pro novou knihovnu Ormion, která je postavena nad databázovou vrstvou Dibi (od autora Nette Frameworku). Dibi již dobře znám a díky této nadstavbě je práce s databází velmi jednoduchá.

5.4 Ormion

Ormion je jednoduchá komponenta rozšiřující Dibi, na stránce o doplňcích k Nette[21] lze nalézt dostačující dokumentaci.

Základní myšlenka u ORM je taková, že pro každou tabulku v databázi máme vytvořenou třídu. Model aplikace tedy tvoří třídy odpovídající daným tabulkám, v Ormionu vypadá takováto třída jednoduše:

```
class Article extends Ormion\Record
{
    protected static $table = "articles";
}
```

Třídu pouze podědíme od knihovny Ormion a do proměnné *table* napíšeme název tabulky v databázi. Toto pro základní funkčnost modelu stačí, při rozšiřování pouze dopisujeme další vlastní metody. Při prvním použití modelu se z databáze načte struktura tabulky a uloží se do konfiguračního souboru.

Výběr dat probíhá tak, že nad modelem *Article* zavoláme metodu *find* pro získání jednoho záznamu (dostaneme objekt *Article*) nebo *findAll* pro celou kolekci záznamů (dostaneme objekt OrmionCollection). Pokud chceme k výběru dat aplikovat nějakou podmínku, pouze za *find* nebo *findAll* dopíšeme *By* a jméno atributu. Například, když chceme všechny články od autora s ID 4, tak zavoláme:

```
Article::findAllByAuthor(4)
```

Někdy se však stává, že je potřeba napsat složitější dotaz a v tom jednoduché ORM zaostává. Proto Ormion dovoluje s jeho objekty pracovat jako s rozhraním DibiFluent (objektové skládání dotazu) a přidat tak k výběru dat dodatečné podmínky. Ukázka s objektovým skládáním dotazu je již obsažena na předchozí straně.

6 Implementace

6.1 Architektura

Aplikace používá architekturu MVC (Model-View-Controller), která je známá již od roku 1979. Spočívá v rozdělení aplikace do třech logických částí tak, aby je šlo upravovat samostatně a dopad změn byl na ostatní části co nejmenší.

Model reprezentuje data, v našem případě propojení s databází a vrstvu nad ní. Je použita technologie ORM, proto každý model je tvořen samostatnou třídou a představuje tabulku v databázi. Každý řádek v tabulce je vrácen jako objekt s určitými vlastnostmi.

View zobrazuje uživatelské rozhraní aplikace. Je tvořen šablonami, které generují HTML kód zobrazovaný v prohlížeči uživatele.

URL router je v MVC architektuře součástí Controlleru, já ho uvádím zvlášť z důvodu větší důležitosti v Nette Frameworku – zde je totiž použit oboustranný router – nejen že zpracovává příchozí HTTP požadavky a dle nastavených pravidel je nasměruje na konkrétní Action v daném Controlleru, ale funguje i opačným směrem. To znamená, že View se URL routeru dotazuje, jakou má použít HTTP adresu, aby směrovala na Action, na kterou chceme vytvořit odkaz ze šablony. URL adresy jsou tak definovány pouze na jednom místě a nevzniká zbytečná redundance.

Controller se stará o aplikační logiku. Jak už bylo řečeno, konkrétní Action je volána z URL routeru. Následně probíhá spolupráce s Modelem (získání nebo úprava dat) a nakonec se podle šablony ve View vykreslí stránka.

Jak již bylo řečeno, architektura MVC je již velmi stará. Od té doby se některé věci změnily a jsou chápány jinak, proto Nette Framework používá místo MVC architekturu MVP. Controller byl tedy nahrazen Presenterem, vysvětlení tohoto kroku se můžete dočíst v seriálu na serveru Zdroják[15] ve článku „MVC & MVP“. Dále v textu již budu používat pouze označení Presenter.

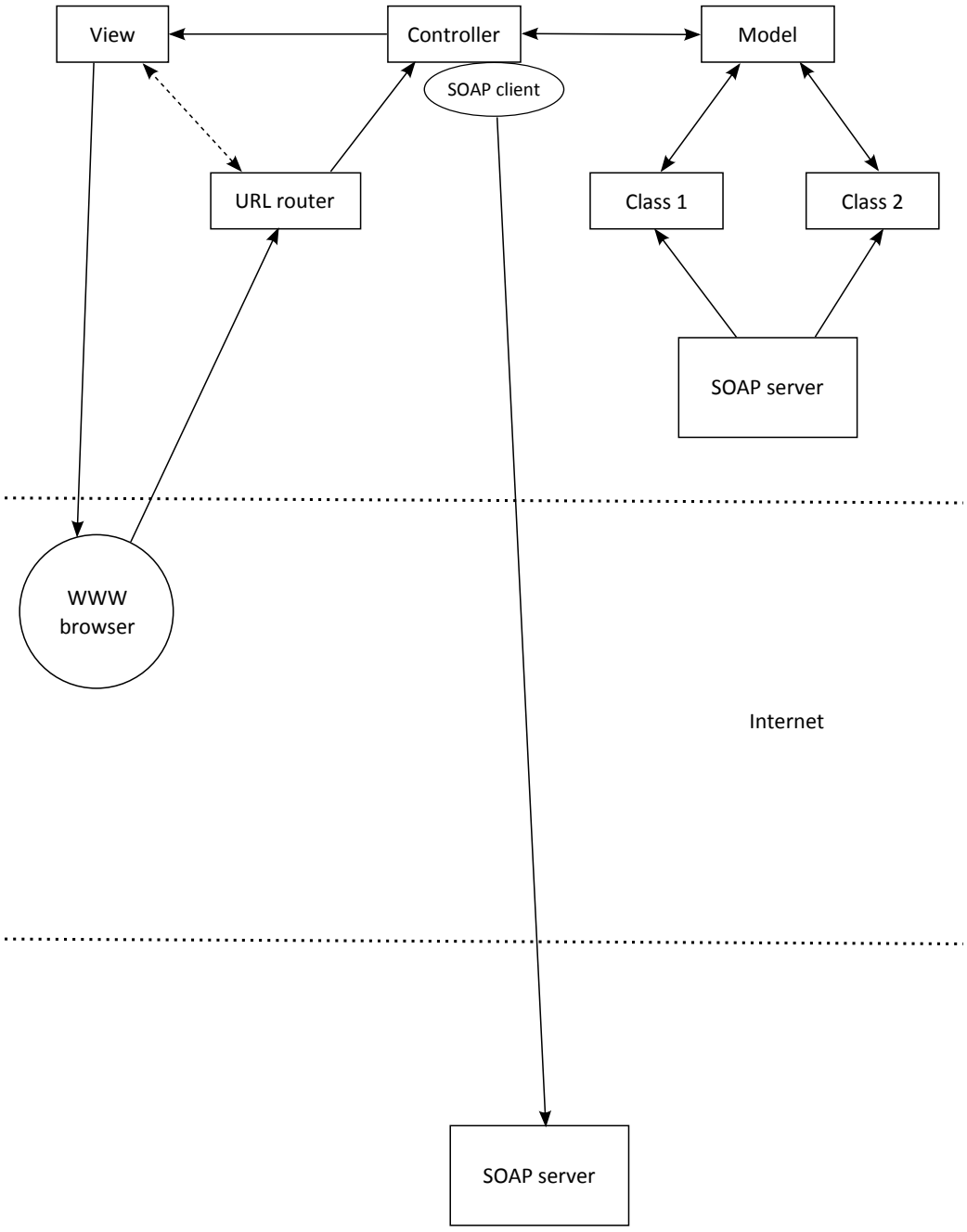
6.2 Propojení se vzdálenou lokalitou

Vzdálené lokality spolu komunikují pomocí technologie SOAP.

SOAP server v našem případě slouží jako forwarder – pouze přesměrovává požadavky na konkrétní lokální třídu k tomu určenou a provádí transformaci parametrů do dat předávaných patřičné metodě. Tyto třídy, zajišťující vzdálenou funkcionalitu, komunikují s Modelem stejně jako Presenter a jsou uvedeny ve společném jmenném prostoru. Aby mohla být lokální funkcionalita poskytnuta i vzdálené lokalitě, tak se kód zajišťující danou funkcionalitu nepíše přímo do Presenteru, ale je rozdělen do dalších tříd nebo začleněn do Modelu.

SOAP klient je přímo součástí kódu, kdy se konkrétní požadavky volají z Presenteru (případně z jiné třídy volané Presenterem) a využívají pouze třídu zajišťující SOAP komunikaci.

Architekturu aplikace i s názorným propojením se vzdálenou lokalitou ukazuje obrázek 23.



Obrázek 23: Architektura aplikace

6.3 Adresářová struktura aplikace

Pro základní orientaci v aplikaci (projektu) je nejdříve nutné znát rozdělení jednotlivých částí do adresářů:

- **app** - vlastní aplikační logika
  - **locale** - soubory s překlady do více jazyků
  - **log** - záznamy o chybách a další informace, ***adresář musí mít nastaveny práva pro zápis***
  - **models** - Model (datová vrstva)
  - **modules** - pomocné třídy pro práci s uživatelskými skupinami, zpracování topologie v XML apod.
  - **presenters** - Presenter
  - **temp** - vyrovnávací paměť, ***adresář musí mít nastaveny práva pro zápis***
  - **templates** - View, soubory s šablonami
  - *ACL.php* - definice uživatelských rolí a jednotlivých práv
  - *bootstrap.php* - zaváděcí soubor celé aplikace
  - *config.ini* - konfigurace aplikace, nastavení přístupu k databázi a dalších proměnných
- **libs** - knihovny - Nette, Dibi, Texy, Ormion a další doplňky
- *.htaccess* - pomocí *mod_rewrite* jsou směrovány všechny požadavky na *index.php*
- *index.php* - nastavení adresářů a předání řízení souboru *bootstrap.php*
- *verify.php* - ověření uživatele a rezervace pro rezervační server

Na přiloženém CD je navíc umístěn soubor se SQL příkazy pro převod současné databáze na formát používaný v nové aplikaci. Při přechodu na novou řídící aplikaci je díky tomu možné zachovat stávající data.

6.4 Rozdělení do funkčních celků

Jednotlivé funkce jsou rozděleny do Presenterů podle analýzy (kapitola 3.2):

- **BasePresenter.php** - z této třídy dědí všechny Presentery, před zpracováním požadavku kontroluje zda je uživatel přihlášen a generuje navigační menu
- **AuthPresenter.php** - autentizace, přihlášení do systému a také odhlášení
- **CategoryPresenter.php** - kategorie, dle DFD 4 (obr. 7)
  - renderDefault - seznam tříd (listClasses)

  - renderNewClass
  - renderEditClass
  - renderDeleteClass
  - renderCategories
  - renderNewCategory
  - renderEditCategory
  - renderDeleteCategory
- **FilePresenter.php** - soubory, dle DFD 11 (obr. 14)
  - renderDefault - seznam souborů (listFiles)
  - renderShowFile
  - renderDownloadFile
  - renderNewFile
  - renderEditFile
  - renderDeleteFile
- **HomepagePresenter.php** - homepage, dle DFD 9 (obr. 12)
  - renderDefault - hlavní strana (index)
  - renderEditor - v DFD jako editPages
- **ReservationPresenter.php** - rezervace, dle DFD 5 (obr. 8)
  - renderDefault - seznam rezervací (listReservations)
  - renderMy
  - renderShow
  - renderNew
  - renderActive
  - renderApplet
  - renderDeviceReload
  - renderDelete
- **TaskPresenter.php** - úlohy, dle DFD 3 (obr. 6)
  - renderDefault - seznam úloh (listTasks)
  - renderShowTask
  - renderNewTask
  - renderEditTask
  - renderDeleteTask

- **UserAdminPresenter.php** - správa uživatelů, dle DFD 8 (obr. 11)
  - renderDefault - seznam uživatelů (listUsers)
  - renderNewUser
  - renderEditUser
  - renderDeleteUser
- **UserGroupPresenter.php** - skupiny uživatelů, dle DFD 12 (obr. 15)
  - renderDefault - seznam skupin (listUserGroups)
  - renderShow
  - renderNew
  - renderEdit
  - renderDelete
- **UserPresenter.php** - uživatel, dle DFD 1 (obr. 4)
  - renderDefault - změna osobních údajů, hesla a nastavení času
  - renderNotes

Pro přidání nové funkce/stránky se vytvoří metoda *renderNazev* v příslušném Presenteru a do adresáře *app/templates/NazevPresenteru* se vytvoří šablona *Nazev.phtml*. V případě nového funkčního celku se vytvoří nový Presenter, který dědí z třídy *BasePresenter* a v *app/templates* se vytvoří nový adresář podle jména Presenteru.

Při doplňování stránek není potřeba měnit mapování adres v URL routeru. Standardně se používá adresa ve tvaru *presenter/action/id* a v případě jiných parametrů než *id* se tyto přidávají do adresy za otazník. Konfigurace routeru se upravuje jen když je potřeba mapovat nějakou stránku na speciální tvar adresy (aby vybraná důležitá stránka měla dobře zapamatovatelnou adresu).

6.5 Formuláře

Formuláře se vytvářejí jako komponenty. Slouží k tomu metoda *createComponent*, která se píše do Presenteru. Odeslání formuláře nebo kliknutí na jiné tlačítko vyvolá událost - do definice formuláře se pomocí *onSubmit* nebo *onClick* určují metody, které se mají zavolat, až tato událost nastane. Po každém úspěšném zpracování formuláře je potřeba provést přesměrování na jinou stránku. Toto je velmi důležité - pokud by nedošlo k přesměrování, mohl by uživatel v prohlížeči pomocí tlačítka Zpět nebo Obnovit vyvolat opětovné zpracování formuláře se stejnými hodnotami a takovéto chování aplikace je nepřípustné.

Po dopsání definice formuláře a jeho obslužné metodě stačí v šabloně napsat `{widget nazevFormulare}` a formulář se na tomto místě vykreslí. Definice formuláře a validačních pravidel je jednoduchá, bližší popis naleznete v dokumentaci k Nette Frameworku.

Hodnota	Konstanta	Popis
0	EMERGENCY, EMERG	system is unusable
1	ALERT	action must be taken immediately
2	CRITICAL, CRIT	critical conditions
3	ERROR, ERR	error conditions
4	WARNING	warning conditions
5	NOTICE	normal but significant condition
6	INFO	informational
7	DEBUG	debug-level messages

Tabulka 1: Úrovně logování v komponentě Logger

6.6 Zobrazování zpráv

Po každém zpracování formuláře se zobrazí uživateli zpráva informující o úspěšném provedení akce či chybě. K tomuto účelu je v Nette implementována následující metoda, která se používá v Presenteru:

```
$this->flashMessage('Zpráva uživateli', 'info');
```

Druhý parametr definuje styl, jakým se zpráva zobrazí - v současné aplikaci jsou definovány tyto styly: *info*, *done*, *warning* a *error*.

6.7 Logování

Pokud chceme nějaké zprávy logovat na disk pro administrátora systému, tak je k tomuto účelu obsažena v aplikaci komponenta Logger[22]. Před použitím musíme získat její instanci:

```
$logger = Environment::getService('Nette\Logger');
$logger->logMessage(ILogger::CRIT, 'Zpráva, která se uloží do logu (souboru)');
```

Úrovně logování vycházejí ze známého nástroje syslog a jsou popsány v tabulce 1.

6.8 Lokalizace

K lokalizaci je používán nástroj gettext[13]. V adresáři *app/locale* jsou umístěny soubory s překlady (.po), ty lze upravovat pomocí multiplatformního editoru Poedit[23]. V tomto editoru lze tyto soubory i jednoduše kompilovat do binární podoby (.mo), která je pak použita k překladu aplikace. Pro posbírání všech textů z aplikace a vygenerování *po* souboru se používá GettextExtractor[24] - skript *generate_locale.php*. Po spuštění tohoto skriptu se prozkoumají všechny soubory v adresáři *app* a uloží se řetězce, které se nacházejí ve formulářích, šablonách nebo jako parametr v metodě *translate* - tedy všechny řetězce, které se budou překládat do zvoleného jazyka.

Zapnutí lokalizace pro prvky formuláře:

```
// getTranslator je metoda BasePresenteru
$translator = $this->getTranslator();
$form->setTranslator($translator);
```

Překlad na jiném místě v Presenteru:

```
$translator->translate('Notes has been updated.')
```

Zapnutí lokalizace pro šablony je provedeno v *BasePresenteru* v metodě *beforeRender*:

```
$this->template->setTranslator($translator);
```

Lokalizace textu v šabloně pak vypadá takto:

```
<p>{_'Are you sure that you want to delete task "%s"?', $name}</p>
```

6.9 Dokumentace k frameworku

Toto byl pouze stručný úvod pro základní orientaci v aplikaci, podrobnější informace lze nalézt v dokumentaci k Nette Frameworku[8]. Jako první krok a základní orientaci v Nette doporučuji seriál na serveru Zdroják od autora frameworku - Začínáme s Nette Framework[15]. Komunita okolo Nette je velká a proto velkou pomoc při hledání různých řešení poskytuje fórum Nette Frameworku, které má již přes 26 000 příspěvků. Na stránkách Nette se nachází i tutoriály a API dokumentace, která určitě přijde vhod.

7 Příklad doplnění stránky do aplikace

Pomocí tohoto jednoduchého návodu si ukážeme, jakým způsobem se rozšiřuje funkcionalita aplikace. Příklad popisuje vytvoření nové stránky s jednoduchým výpisem prvních deseti uživatelů.

Novou stránku pojmenujeme *showFirstUsers* a zařadíme ji mezi stávající správu uživatelů. Vybereme tedy stávající Presenter *UserAdminPresenter* a vytvoříme zde novou metodu *renderShowFirstUsers*. V obsahu metody bude jen jednoduché načtení prvních deseti uživatelů a uložení výsledku do proměnné v šabloně:

```
public function renderShowFirstUsers()
{
    $this->template->users = User::findAll()->orderBy('id')->limit(10);
}
```

Při práci s kolekcí z Modelu je důležité vědět, že data nejsou načítány z databáze ihned, protože je možno dále specifikovat další požadavky, které se promítnou do výsledného dotazu. Dotaz na databázi je spuštěn až při použití *foreach* nad kolekcí nebo zavolání jedné z „fetchovacích" (*fetch, fetchAll, fetchSingle, fetchColumn, fetchPairs, fetchAssoc*) či agregačních (count, getMin, getMax, getAvg, getSum) funkcí.

Dále je potřeba vytvořit šablonu, ve které zobrazíme výsledky. V adresáři *app/templates/UserAdmin* vytvoříme soubor *showFirstUsers.phtml*:

```
{block #content}

<h1>{block #title}{_'First users'}{/block}</h1>

<ul>
{foreach $users as $user}
    <li>{$user->first_name} {$user->surname}</li>
{/foreach}
</ul>
```

Značka `{block #content}` znamená, že následující kód bude vložen do bloku *content* v hlavní šabloně *@layout.phtml*. Podobně obsah nadpisu H1 je vložen do bloku *title* a bude použit pro titulek stránky. Následuje už jen vypsání dat, které jsme získali v Presenteru.

Jako poslední krok vložíme odkaz na novou stránku do hlavního menu aplikace. Menu se vytváří v *BasePresenteru* v metodě *createComponentNavigation*. Sekce „Uživatelé" nyní vypadá takto:

```
$users = $navigation->get('Users');
if ($user->isAllowed('UserAdmin'))
    $users->add('Browse', $this->link('UserAdmin:default'));
if ($user->isAllowed('UserAdmin'))
    $users->add('New user', $this->link('UserAdmin:newUser'));
if ($user->isAllowed('UserGroup'))
    $users->add('User Groups', $this->link('UserGroup:default'));
```

Můžete si všimnout, že odkazy se vytvářejí voláním metody *Presenteru* s parametrem ve tvaru *Presenter:action*. Tyto údaje se pošlou URL routeru a ten nám vrátí patřičnou URL adresu.

Pod výše zmíněný blok kódu vložíme odkaz na naši novou stránku bez ověřování oprávnění:

```
$users->add('First users', $this->link('UserAdmin:showFirstUsers'));
```

Pokud bychom chtěli na stránku odkázat z jiné stránky, vložíme do patřičné šablony následující odkaz:

```
<a href="{link UserAdmin:showFirstUsers}">First users</a>
```

Nyní je vše hotovo a po kliknutí na námi vytvořený odkaz se zobrazí nová stránka se seznamem deseti uživatelů.

8 Provoz, nasazení, testování

K testovacímu provozu řídící webové aplikace byl použit virtualizovaný Virtlab - lokalita virtualizovaná pomocí software XEN, která slouží pro vývojáře a kopíruje funkcionalitu reálného Virtlabu. Jak ukazuje obrázek 24, je možné na této virtuální lokalitě realizovat i jednoduché propojení prvků router - pc.

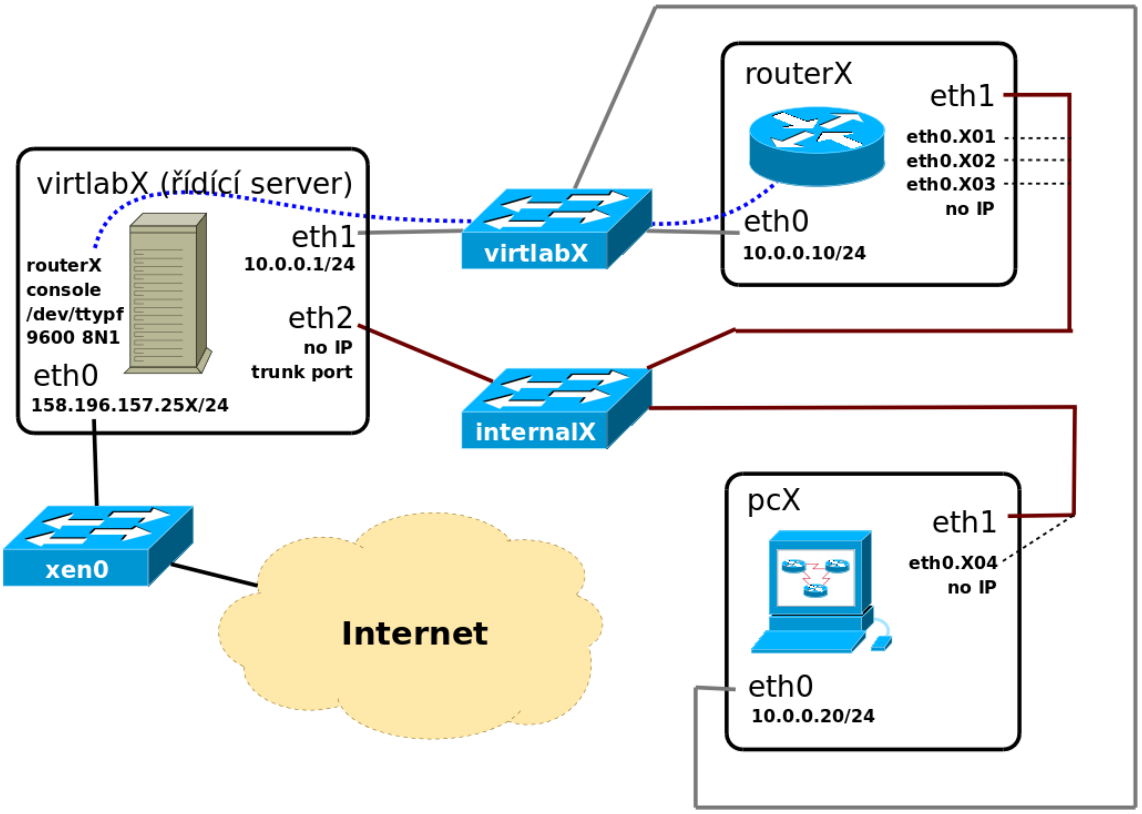
Přístup k testovacímu provozu aplikace je možný z vnitřní sítě VŠB-TUO a to na adrese: https://praha.dvirtlab.net/new/

Chod webové aplikace zajišťuje následující software: linuxová distribuce Debian 5.0 Lenny, webový server Apache 2.2.9, PHP 5.3.2 a databáze MySQL 5.0. Jako testovací data byl použit obsah databáze Virtlabu, který je v provozu a pracují na něm studenti.

Nová webová aplikace obsahuje základní funkcionalitu, takže je možné otestovat tyto po sobě jdoucí akce:

- vytvoření nového uživatele
- vytvoření úlohy + nahrání souborů se specifikací úlohy, ukázkovou konfigurací, definicí topologie apod.
- vytvoření rezervace z úlohy
- přístup k prvkům rezervace přes konzoli (Java applet)

Testování aplikace prováděli členové Virtlabu, ohlášené chyby a nedostatky byly obratem opraveny.



Obrázek 24: Schéma propojení instancí v rámci jedné virtuální lokality

9 Závěr

Volba Nette Frameworku se ukázala jako velmi dobrá a ani při práci s datovou vrstvou jsem neshledal u vybraného nástroje Ormion žádné zásadní nedostatky. Základní funkcionalita řídící webové aplikace Virtlabu byla implementována s použitím moderních technologií a nasazena na testovací virtuální stroj, kde byla následně otestována uživateli. Také byl vytvořen soubor s SQL příkazy, který převádí starou databázi aplikace na nový formát - je tak možno veškeré stávající data použít v nové aplikaci, což výrazně usnadní přechod.

Na tuto hotovou reimplementaci navázal ve své diplomové práci Roman Krhovjak, který dále rozšířil práci s uživatelskými skupinami a spolupráci se vzdálenou lokalitou.

10 Reference

[1] VONDRÁK, Ivo. *Úvod do softwarového inženýrství*. Ostrava, 2002. 74 s.

[2] ŠARMANOVÁ, Jana. Databázové a informační systémy. Ostrava, 2007. 124 s. ISBN 978-80-248-1499-5.

[3] VRÁNA, Jakub. *PHP triky* [online]. 5.2.2010 [cit. 2010-04-29]. Verze PHP v ČR – únor 2010. Dostupné z WWW: <http://php.vrana.cz/verze-php-v-cr-unor-2010.php>.

[4] Model-View-Controller In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 5 August 2003, 23 April 2010 [cit. 2010-04-27]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Model-View-Controller>.

[5] DANĚK, Petr. Velký test PHP frameworků. *Root.cz : informace nejen ze světa Linuxu* [online]. 11. 9. 2008, [cit. 2010-04-27]. Dostupný z WWW: <http://www.root.cz/clanky/velky-test-php-frameworku-2008/>.

[6] *Kohana : The Swift PHP Framework* [online].Kohana Team, c2010 [cit. 2010-04-27]. Dostupné z WWW: <http://kohanaframework.org>.

[7] *Zend Framework* [online].Zend Technologies Ltd., c2010 [cit. 2010-04-27]. Dostupné z WWW: <http://framework.zend.com>.

[8] *Nette Framework* [online].Nette Foundation, 27.4.2010 [cit. 2010-04-27]. Dostupné z WWW: <http://nettephp.com>.

[9] *Fuse : PHP MVC Framework* [online]. 2009 [cit. 2010-04-27]. Dostupné z WWW: <http://www.phpfuse.net>.

[10] *PRADO PHP Framework* [online].PRADO Group, c2010 [cit. 2010-04-27]. Dostupné z WWW: <http://pradosoft.com>.

[11] *CakePHP : the rapid development php framework* [online].Cake Software Foundation, Inc., c2010 [cit. 2010-04-27]. Dostupné z WWW: <http://cakephp.org>.

[12] *CodeIgniter : Open source PHP web application framework* [online].EllisLab, Inc., c2010 [cit. 2010-04-27]. Dostupné z WWW: <http://codeigniter.com>.

[13] GNU Project [online].Free Software Foundation, Inc., 2010/01/31 [cit. 2010-04-27]. Gettext. Dostupné z WWW: <http://www.gnu.org/software/gettext/gettext.html>.

[14] *Dibi : Database Abstraction Library for PHP 5* [online].Nette Foundation, 25.2.2010 [cit. 2010-04-27]. Dostupné z WWW: <http://dibiphp.com>.

[15] GRUDL, David. Začínáme s Nette Framework. *Zdroják : tvorba webových stránek a aplikací* [online]. 30. 6. 2009, [cit. 2010-04-27]. Dostupný z WWW: <http://zdrojak.root.cz/serialy/zaciname-s-nette-framework/>.

[16] WAGE, Jonathan H., et al. *Doctrine : PHP Object Relational Mapper* [online]. 2010 [cit. 2010-04-27]. Dostupné z WWW: <http://www.doctrine-project.org>.

[17] *Propel ORM* [online].Propel project, March 2010 [cit. 2010-04-27]. Dostupné z WWW: <http://www.propelorm.org>.

[18] *EZPDO* [online]. May 9th, 2007 [cit. 2010-04-27]. Dostupné z WWW: <http://www.ezpdo.net>.

[19] *RedBean* [online]. 2010 [cit. 2010-04-27]. Dostupné z WWW: <http://www.redbeanphp.com>.

[20] BUSH, Anthony, et al. *CoughPHP : Collection Handling Framework* [online]. c2009 [cit. 2010-04-27]. Dostupné z WWW: <http://coughphp.com>.

[21] MAREK, Jan. *Nette Framework* [online].Nette Foundation, c2010 [cit. 2010-04-27]. Ormion. Dostupné z WWW: <http://addons.nettephp.com/cs/ormion>.

[22] SMITKA, Jan; PECKA, Martin. *Nette Framework* [online].Nette Foundation, c2010 [cit. 2010-04-27]. Logger. Dostupné z WWW: <http://addons.nettephp.com/cs/logger>.

[23] SLAVÍK, Václav. *Poedit* [online]. 2010-03-22 [cit. 2010-04-27]. Dostupné z WWW: <http://www.poedit.net>.

[24] KLÍMA, Karel. *Nette Framework forum* [online]. 22. 10. 2009 [cit. 2010-04-27]. Kompletní lokalizace Nette aplikací – GettextExtractor v2. Dostupné z WWW: <http://forum.nettephp.com/cs/2798-kompletni-lokalizace-nette-aplikaci-gettext extractor-v2>.