VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

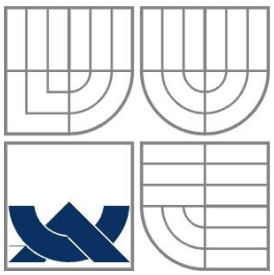
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

PODPORA XML ŠABLON V PHP RÁMCÍCH

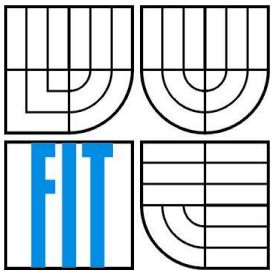
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE LUKÁŠ AMBROŽ
AUTHOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

PODPORA XML ŠABLON V PHP RÁMCÍCH

XML TEMPLATE SUPPORT IN PHP FRAMEWORKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUKÁŠ AMBROŽ
AUTHOR

VEDOUCÍ PRÁCE — Ing. RADEK BURGET, Ph.D.
SUPERVISOR

BRNO 2013

Abstrakt

Tato práce se zabývá problematikou definice šablon stránek v PHP rámcích. Jejím cílem je navrhnout a implementovat rozšíření pro Nette Framework, které umožní definovat šablony v jazyce XML. V rámci práce byly také porovnány různé přístupy k práci se šablonami v běžně používaných PHP rámcích a popsány možnosti jazyka XML, které je možné využít pro vlastní rozšíření. Vytvořená knihovna zahrnuje vlastní sadu XML značek, pomocí kterých lze šablony definovat v jazyce XML.

Abstract

This bachelor's thesis deals with template support in PHP frameworks. The aim is to design and implement an extension for Nette Framework, which allows defining templates in XML language. Thesis also provides a comparison of different approaches of template support in commonly used PHP frameworks. In addition it describes the possibilities of XML language that might be useful for own extension. Created library includes a set of special XML tags, which enable definition of templates in XML.

Klíčová slova

PHP, XML, XHTML, Nette Framework, Model-view-controller, šablony

Keywords

PHP, XML, XHTML, Nette Framework, Model-view-controller, templates

Citace

Ambrož Lukáš: Podpora XML šablon v PHP rámcích, bakalářská práce, Brno, FIT VUT v Brně, 2013

Podpora XML šablon v PHP rámcích

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Lukáš Ambrož
15. května 2013

Poděkování

Chtěl bych poděkovat vedoucímu mé bakalářské práce Ing. Radku Burgetovi, Ph.D. za odbornou pomoc a cenné rady při konzultacích.

Obsah

1 Úvod

Použití frameworků při vývoji webových aplikací v PHP je dnes zcela běžnou praxí. Díky množství funkcí, které obsahují, mohou vytváření vlastních programů významně usnadňovat. Mimo to ale také definují vlastní strukturu aplikace, která umožňuje oddělit její jednotlivé vrstvy. To souvisí zejména s využitím architektury Model-view-controller, kterou většina PHP rámců podporuje.

Při vytváření prezentační vrstvy je pak nutné definovat šablony reprezentující jednotlivé stránky nebo jejich části. Tyto šablony umožňují specifikovat dynamické části stránek, jejichž obsah je znám až v době vykonávání samotného programu. K tomu jednotlivé frameworky používají různé přístupy, například umožňují definovat šablony pomocí vlastních jazyků navržených pro tyto účely.

Cílem této práce bude navrhnout a implementovat rozšíření pro Nette Framework, které umožní šablony stránek definovat v jazyce XML. Použití tohoto jazyka může být výhodné zejména proto, že aplikací tohoto jazyka je přímo jazyk XHTML pro tvorbu stránek.

První část práce (kapitola 2) bude věnována frameworku Nette, kde budou vysvětleny základní pojmy a principy nutné pro realizaci vlastního rozšíření. Kapitola 3 pak porovnává různé přístupy, které jsou k dispozici pro práci se šablonami v běžně používaných rámcích. V následující části jsou popsány možnosti jazyka XML, které lze pro návrh vlastní knihovny využít. Poté již následuje návrh vlastního řešení a popis jeho implementace. V kapitole 7 je popsána ukázková aplikace vytvořená pro demonstraci použití vytvořeného rozšíření. V závěru práce jsou pak zhodnoceny dosažené výsledky a navržena další možná vylepšení.

2 Nette Framework

Protože je vlastní rozšíření implementované v rámci této práce určeno právě pro Nette, bude tomuto frameworku věnováno více prostoru v porovnání s ostatními PHP rámci. Nette Framework nabízí pro vytváření aplikací v PHP mnoho možností, jak jejich vývoj zjednodušit. Díky použití architektury Model-View-Presenter (MVP) umožňuje oddělení datové a prezentační vrstvy aplikace. Dále poskytuje prostředky pro práci s formuláři, databázemi atd. s důrazem na jejich bezpečnost. Obsahuje také vlastní systém pro definici šablon stránek – Latte. Díky aktivní komunitě okolo frameworku je k dispozici řada doplňků, které lze ve vlastních aplikacích použít. Následující kapitola čerpá z webových stránek Nette [1] a předpokládá framework ve verzi 2. Uvedené informace slouží spíše jako úvod do dané problematiky a věnují se především částem, které jsou potřebné pro realizaci vlastního řešení. Pro další podrobnosti jsou proto případní zájemci odkázáni na uvedené zdroje.

2.1 Struktura aplikace

Jak již bylo zmíněno v úvodu, základem pro vytváření aplikací v Nette je architektura MVP. Vrstva view je v Nette reprezentována šablonami, presenter pak view propojuje s modelem. Presenter v Nette představuje objekt, který podle příchozích požadavků od uživatele provádí příslušné akce v datové vrstvě aplikace a následně generuje odpovědi (např. HTML stránky). Na rozdíl od vrstvy controller v architektuře MVC však nemá na starosti žádné další akce, typicky v uživatelském rozhraní.

Presentery

Presenterů může být v aplikaci libovolný počet. Každý presenter daný PHP třídou (která je potomkem třídy `Nette\Application\UI\Presenter`) má pak své akce (metody), které již spolu s daným presenterem reprezentují konkrétní požadavky přijaté od uživatele. Příkladem může být presenter `Product` obsahující akci `show` pro zobrazení produktu apod. Každý presenter má pak vlastní sadu views, které již představují šablony pro vykreslení. Při běhu aplikace se nejprve vykonají požadované akce, podle kterých se pak vykreslují jim příslušející šablony.

Akce jsou v presenterech reprezentovány zvláštními metodami ve tvaru `action<Action>()`, kde `<Action>` je název příslušné akce. Tyto metody mohou provádět libovolné operace, například v modelové vrstvě aplikace.

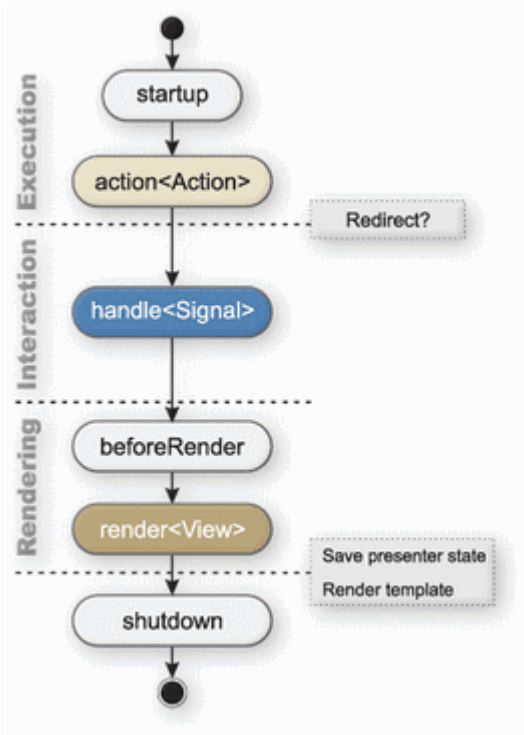
Po dokončení akce je na řadě vykreslení šablony (view). Před samotným vykreslením je ještě volána metoda `render<View>()`, která slouží k předání proměnných příslušné šabloně (v objektu presenteru je její objekt dostupný v proměnné `$template`), jak ukazuje příklad 2.1. Šablona pro

<View> je pak umístěna ve složce templates/<Presenter>/<view>.latte (soubor začíná malým písmenem).

Příklad 2.1 - Ukázka metody `renderDefault()` zajišťující registraci proměnných v objektu šablony.

```
public function renderDefault() {
    $this->template->title = 'Title';
}
```

Pokud není specifikována konkrétní akce, je použita akce s názvem `default`. Pro tuto akci se pak nejprve provede metoda `actionDefault()` a poté `renderDefault()`. Není-li tedy název view změněn ručně (lze pomocí `setView()`), vykoná se view se stejným jménem jako akce. Výše uvedené metody není nutné v presenteru vždy uvádět. Pokud neexistují, jednoduše se v rámci akce nic neprovede, resp. se šabloně nepředají žádná data. Pro názornost je posloupnost volání jednotlivých metod presenteru k dispozici na obrázku 2.1.



Obrázek 2.1 - Životní cyklus presenteru. Převzato z [1].

Ve většině aplikací je nutné, aby některá data byla společná pro více presenterů. Může se jednat například o menu nebo jiné podobné položky, které mají být k dispozici na všech stránkách aplikace. Aby se tato data nemusela načítat v každém presenteru, je možné definovat třídu reprezentující společného předka všech presenterů – například `BasePresenter`. Tato třída může

definovat metodu `beforeRender()` specifikující operace, které se mají vykonat před každým vykreslením stránky (viz obrázek 2.1). Může jít například o přiřazení společných dat šabloně apod.

Výchozí presenter a akce aplikace, která se má vykonat při příchodu na domovskou stránku, lze v rámci aplikace nastavit (ve výchozím stavu je to `HomepagePresenter`). Zápis odkazů na jednotlivé akce pak definuje nastavení routeru. Protože pro potřeby práce bude vždy použit zápis odkazů ve speciálním tvaru, jehož použití bude vysvětleno v části věnující se šablonám, není nutné se nyní těmito detaily zabývat.

Adresářová struktura aplikace

Základní adresářová struktura pro Nette aplikace je znázorněna v příkladu 2.2. Zmínku ještě zaslouží složka `www` obsahující soubory veřejně dostupné z webu. To je například soubor `index.php` pro danou aplikaci, obrázky, CSS dokumenty a další. Pokud nechceme tento adresář vždy uvádět v URL pro přístup k aplikaci, je nutné na něj namířit kořenovou složku webového serveru. Případně je možné také obsah adresáře přesunout o úroveň výše do složky `app`, a v souboru `index.php` pak správně nastavit cesty.

Příklad 2.2 - Adresářová struktura aplikace. Převzato z [1].

```
app/                - Hlavní adresář aplikace
    config/
    model/          - Třídy modelové vrstvy
    presenters/     - Třídy presenterů
    router/
    templates/      - Šablony
    bootstrap.php   - Zaváděcí soubor
libs/               - Knihovny
    Nette/
log/
temp/               - Dočasné soubory – např. cache
test/
www/                - Složka přístupná z webu
```

2.2 Šablony

Pro definici šablon stránek používá Nette vlastní jazyk – Latte. Ten definuje pro zápis šablony dva druhy speciálních značek – makra a tzv. n:makra. Makra jsou konstrukce uzavřené ve složených závorkách, které umožňují zápis akcí v šablonách. Příkladem může být makro `{if}` umožňující

zápis podmínky. N:makra jsou pak alternativním zápisem pro každé párové makro. Zapisují se jako atributy HTML elementů, např. `<div n:if="…">`. Ukázku lze vidět v příkladu 2.3. V šablonách jsou také k dispozici proměnné přiřazené objektu dané šablony.

Příklad 2.3 - Ukázka ekvivalentního zápisu pomocí makra a n:makra v Latte.

```
{if $test}<p>Hello!</p>{/if}
<p n:if="$test">Hello!</p>
```

Nette poskytuje pro práci se šablonami také tzv. helpery, což jsou pomocné funkce umožňující formátování dat před jejich výpisem v šablonách. Výpis proměnné se v Latte provádí jejím zápisem do složených závorek, např. `{$title}`. Helpery se pak na vypisované objekty dají aplikovat následujícím způsobem: `{$title|capitalize}`. Případné parametry helperu lze oddělit uvnitř zápisu dvojtečkami nebo čárkami, například `{$title|truncate:10}`. Helpery je také možné za sebou řetězit. Při výpisu proměnných v šablonách Latte dále automaticky převádí znaky mající v HTML zvláštní význam na odpovídající entity. Tento převod navíc správně zajišťuje i pro různé části dokumentu, kde je množina speciálních znaků odlišná.

Latte obsahuje mj. také makra pro vykreslování formulářů vytvořených v rámci Nette frameworku, definici bloků, dědičnost šablon apod. Protože jsou principy těchto možnosti vysvětleny v rámci návrhu vlastního rozšíření v kapitole 5, nebudou nyní dále rozebírány. Přehled všech podporovaných maker a helperů je k dispozici na stránkách frameworku [1].

Příklad 2.4 - Ukázka použití Latte.

```
<h1>{$title|capitalize}</h1>
{if $test}
    {form 'loginForm'}
        <div>
            {label login}Login:{/login}
            {input login}
        </div>
    {/form}
{/if}
```

Zápis odkazů

Latte navíc obsahuje i zvláštní makra umožňující zápis odkazů na základě zadané akce presenteru. Jedná se o makro `{link Presenter:action}` a také atribut n:href, který lze použít v elementu odkazu následovně: `<a n:href="Presenter:action">Odkaz</a>`.

2.3 Formuláře

Pro práci s formuláři poskytuje Nette vlastní rozhraní. Formuláře jsou zde reprezentovány objekty, kterým lze pomocí metod přidávat jednotlivé formulářové prvky. Nette dále umožňuje formulářovým prvkům přidávat validační pravidla, která zajišťují ověření vyplněných dat při odeslání formuláře. Tato validace pak probíhá jak na straně klienta pomocí JavaScriptu, tak na straně serveru. Třída reprezentující formulář také poskytuje metody pro získání dat z odeslaného formuláře. Ukázka definice formuláře je uvedena v příkladu 2.5.

Příklad 2.5 - Ukázka definice formuláře. Převzato z [1].

```
$form->addText('name', 'Jméno')
     ->setRequired('Zadejte prosím jméno');
$form->addText('age', 'Věk:')
     ->addRule(Form::INTEGER, 'Věk musí být číslo');
$form->addSubmit('send', 'Odeslat');
```

V Nette je také možné prvky sdružovat do skupin. K tomu lze použít kontejnery, které jsou samy prvkem formuláře umožňujícím prvky dále vnořovat, nebo skupiny, které formulářové prvky seskupují pouze vizuálně pro účely vykreslení.

Vykreslování formulářů

Formuláře lze vykreslovat několika způsoby. Prvním způsobem je vykreslení celého formuláře automaticky. Při tomto způsobu je použit tzv. renderer formuláře, který definuje způsob výsledného formátování. Každý formulář má přiřazen výchozí renderer, který lze změnit pomocí metody `setRenderer()`.

Formuláře je také možné vykreslovat po jednotlivých prvcích. V tomto případě jsou zvlášť vykresleny popisky formulářových prvků a zvlášť jejich vstupní pole, jak ukazuje příklad Příklad 2.6.

Příklad 2.6 - Manuální vykreslování formulářů.

```
echo $form['name']->label;
echo $form['name']->control;
```

2.4 Komponenty

Komponenty usnadňují znovupoužití určitých vizuálních částí stránky. Může se jednat například o seznamy produktů, které chceme v rámci stránky vícekrát zobrazit, ale pokaždé s jinými parametry. V Nette jsou reprezentovány zvláštními třídami, které také umožňují jejich vykreslení v rámci šablony.

3 Šablony stránek v jiných rámcích

Následující kapitola uvádí stručný přehled různých přístupů k práci se šablonami v některých běžně používaných PHP rámcích a dalších technologiích.

3.1 Symfony

Tvorba aplikací ve frameworku Symfony je založena na architektuře Model-view-controller. Podobně jako v Nette i zde controller zpracovává příchozí požadavky, na základě kterých nakonec zajišťuje vykreslení příslušných šablon. Pro zápis šablon Symfony používá vlastní jazyk se jménem Twig. Kromě něj lze ale také šablony zapisovat čistě v jazyce PHP. Následující část vychází z informací dostupných na stránkách frameworku [2].

Twig definuje dva základní typy syntaxe. Zápis ohraničený v `{{...}}` slouží k výpisu proměnných nebo výrazů, zatímco zápis `{%...%}` slouží k provádění akcí v šablonách (například cyklů).

Pro úpravu vypisovaných výrazů lze použít filtry, které mají podobný význam i zápis jako helpery v Latte: `{{ title|upper }}`.

K dispozici je také sada funkcí, usnadňujících použití šablon. Jedná se například o funkci `cycle()`, kterou lze použít v cyklech pro výpis prvku zadaného pole na základě čísla aktuálního průchodu cyklu (viz příklad 3.1).

Pro zápis akcí v šablonách jako jsou cykly, podmínky a další, Symfony používá zvláštní tagy zapsané v `{%...%}`. Ukázka použití šablon je znázorněna v příkladu 3.1.

Příklad 3.1 - Ukázka zápisu šablon ve frameworku Symfony. Převzato z [2].

```
{% for i in 0..10 %}
    <div class="{{ cycle(['odd', 'even'], i) }}">
        <p>{{ content }}</p>
    </div>
{% endfor %}
```

Obecně pak Twig poskytuje podobné možnosti při práci se šablonami jako Latte (včetně dědičnosti šablon atd.). Přesto ale lze nalézt některé zajímavé odlišnosti. Jedná se například o vykreslování formulářů, kde je možné kombinovat manuální vykreslování prvků s automatickým. Nejprve jsou ručně vykresleny požadované prvky a zbytek formuláře lze poté vykreslit automaticky pomocí funkce `form_rest()`. Podobně jako Latte také Symfony zajišťuje ošetřování vypisovaných proměnných a výrazů převodem speciálních HTML znaků na entity. Není zde však podpora pro

automatické rozpoznání částí dokumentů s rozdílnými požadavky na ošetřovaná data a ve výchozí podobě se hodnoty ošetřují vždy pro použití v HTML.

3.2 CakePHP

Stejně jako dříve zmíněné rámce i CakePHP využívá pro vytváření aplikací architekturu Model-view-controller. Na rozdíl od nich však tento framework používá pro zápis šablon přímo jazyk PHP. Podrobné informace ke CakePHP lze nalézt na jeho internetových stránkách [3], ze kterých tato podkapitola vychází.

Framework definuje v rámci prezentační vrstvy několik částí:

- Views - Části vykreslované stránky, které přísluší konkrétním akcím v rámci aplikace. Jsou reprezentovány příslušnými šablonami stránek.
- Elements – Menší znovupoužitelné části kódu, které se nachází ve views.
- Layouts – Reprezentují společnou část vykreslovanou pro množinu views, které jsou v ní umístěny. Tato část je stejně jako views dána konkrétní šablonou.
- Helpers – Třídy, které zajišťují logiku pro některé části views, jako jsou například formuláře.

Pro zápis akcí v šablonách je tedy možné použít přímo konstrukce jazyka PHP. CakePHP navíc poskytuje metody, pomocí kterých lze realizovat možnosti známé z dříve uvedených rámců, například definici a vkládání bloků apod. V ukázce 3.2 je uveden příklad šablony obsahující vkládání bloků.

Příklad 3.2 - Ukázka zápisu šablony v CakePHP. Převzato z [3].

```
<h1><?php echo $this->fetch('title'); ?></h1>
<?php echo $this->fetch('content'); ?>
<div class="actions">
    <h3>Related actions</h3>
    <ul>
    <?php echo $this->fetch('sidebar'); ?>
    </ul>
</div>
```

3.3 Další technologie

Protože má vlastní rozšíření implementované v rámci této práce umožnit definici šablon stránek v jazyce XML, jsou v této části pro ukázku uvedeny ještě některé další technologie, které umožňují zápis šablon pomocí prostředků dostupných v XML.

3.3.1 Facelets

Tato technologie [4] představuje webový framework, který umožňuje tvorbu prezentační vrstvy ve webových aplikacích vytvořených v JavaServer Faces. Obsah této podkapitoly vychází také z oficiální příručky [5].

Šablony stránek jsou zde reprezentovány XML soubory, kde je XHTML kód stránky doplněn o speciální značky z vlastních knihoven této technologie. Ty pak umožňují zápis akcí v šablonách a od XHTML tagů jsou rozlišeny použitím XML jmenných prostorů.

Propojení s aplikační logikou je v šablonách realizováno pomocí tzv. Expression Language. Jedná se o zvláštní způsob zápisu výrazů v šablonách, který umožňuje přístup k modelové vrstvě aplikace a provádění základních operací nad touto vrstvou. V šablonách se zapisuje uvnitř `#{…}`.

Knihovny značek pak sdružují tagy provádějící různé druhy akcí v šablonách. K dispozici jsou také knihovny funkcí pro použití v Expression Language, které usnadňují například formátování vypisovaných dat nebo jiné operace.

Protože má vlastní rozšíření Nette frameworku s touto technologií společné některé základní rysy, byl tento přístup v mnohém inspirativní. Ukázka šablony je uvedena v příkladu 3.3.

Příklad 3.3 - Ukázka zápisu Facelets. Převzato z [5].

```
<ui:define name="content">
    <h:graphicImage value="#{resource['images:wave.med.gif']}"/>
    <h:outputText value="You are in the Main Content Section"/>
</ui:define>

<ui:insert name="content">Main Content</ui:insert>
```

3.3.2 PHPTAL

Druhou ukázkou, která využívá pro zápis šablon pouze možností XML (resp. XHTML), je knihovna PHPTAL [6]. Ta umožňuje veškeré akce v šablonách zapisovat ve vlastních XML atributech, které jsou součástí XHTML značek. Přístup k datům aplikace je pak realizován podobně jako ve Facelets použitím zvláštní syntaxe uvnitř vlastních atributů. Ukázka šablony je v příkladu 3.4.

Příklad 3.4 - Příklad zápisu šablony pomocí knihovny PHPTAL. Převzato z [6].

```
<div class="item" tal:repeat="value values">
    <div class="title">
      <span tal:condition="value/hasDate"
            tal:replace="value/getDate"/>
      <a tal:attributes="href value/getUrl"
         tal:content="value/getTitle"/>
    </div>
    <div id="content" tal:content="value/getContent"/>
</div>
```

4 Jazyk XML

Následující podkapitoly uvádí stručný přehled těch částí jazyka XML, které jsou podstatné pro návrh vlastního rozšíření. Jedná se o problematiku XML schémat, jmenných prostorů a pak také o možnosti zpracování XML v PHP.

4.1 Schémata

Následující informace vychází z knihy Jiřího Koska [7] a také tutoriálu od stejného autora [8]. Jazyk XML umožňuje vytvářet vlastní sady značek – tzv. aplikace jazyka (nebo také vlastní značkovací jazyk). Takovou aplikací je například také jazyk XHTML pro popis webových stránek. Vlastní značkovací jazyk (jeho schéma) je pak možné jednoznačně a formálně popsat.

Standard XML obsahuje pro tyto účely nástroj DTD, který však má některá omezení. Nepodporuje například datové typy nebo jmenné prostory. Proto postupně vzniklo několik jazyků pro popis schématu XML dokumentů, které tyto nedostatky řeší. Všechny navíc na rozdíl od DTD využívají pro popis schématu přímo jazyk XML. Nejpoužívanějšími jazyky jsou například W3C XML Schema, Relax NG, RNC nebo Schematron.

Jazyky pro popis schématu dokumentů umožňují specifikovat, které atributy a elementy lze v dokumentu používat, jak je lze vzájemně kombinovat nebo co mohou obsahovat. Významem této specifikace může být například formální definice značkovacích jazyků, validace apod.

4.2 Jmenné prostory

Uvedené informace čerpají z dokumentu [9]. Jmenné prostory v XML jsou mechanismem, který umožňuje v jednom dokumentu kombinovat více sad značek. Jednotlivé sady lze odlišit pomocí prefixů, pro které je nutné deklarovat příslušný jmenný prostor. Deklarace jmenného prostoru se uvádí pomocí atributu `xmlns` typicky v kořenovém elementu dokumentu nebo v elementu, kde je daný prefix poprvé použit. Jako jednoznačný identifikátor jmenného prostoru se používá URI. To slouží pouze pro pojmenování, nikoli pro vyhledání dalších informací. Lze také deklarovat výchozí jmenný prostor, který bude použit pro značky bez uvedeného prefixu. Jmenné prostory lze aplikovat jak na elementy, tak i na atributy. Atributy bez uvedeného prefixu ale nepatří žádnému jmennému prostoru (ani výchozímu). Jednoduchá ukázka použití se nachází v příkladu 4.1.

Příklad 4.1 - Ukázka použití jmenných prostorů v XML.

```
<html xmlns:n="namespace-id">
    <div>
        <n:tag></n:tag>
    </div>
</html>
```

4.3 Zpracování XML v PHP

PHP poskytuje pro zpracování XML dokumentů několik rozhraní, jak uvádí [7], [10] a [11]. Jejich přehled je uveden v rámci této podkapitoly.

Rozhraní SAX

Tento přístup je založen na řízení pomocí událostí. Definujeme funkce, které jsou volány, jakmile parser narazí na příslušný typ uzlu XML dokumentu (začátek elementu, obsah elementu apod.). Zpracování staví na tzv. push modelu – parser předává informace do aplikace sám voláním definovaných funkcí.

Výhodami použití tohoto způsobu jsou především rychlost a nízká paměťová náročnost, což vychází z faktu, že je dokument načítán postupně. Proto se hodí pro zpracování velkých dokumentů.

V PHP je toto rozhraní přístupné v rámci rozšíření XML Parser.

Rozhraní XMLReader

Rozhraní využívá podobný způsob zpracování jako SAX (postupné načítání), ale s tím rozdílem, že staví na tzv. pull modelu. Data jsou tedy předávána aplikaci v okamžiku, kdy o ně sama požádá voláním příslušné metody rozhraní.

Z tohoto důvodu je tento způsob přehlednější než předchozí rozhraní, přičemž zůstávají zachovány zmíněné výhody plynoucí z postupného načítání dokumentu.

Rozhraní DOM

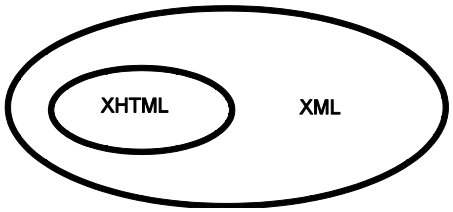
Dokument je v tomto případě reprezentován jako stromová struktura. Díky tomu jej lze snadno procházet, modifikovat jednotlivé uzly apod. Dokument nemusíme procházet od začátku do konce, ale můžeme se v něm pohybovat podle potřeby. Uplatní se v případech, kdy je nutné s dokumentem provádět náročnější práci. Jelikož je při tomto způsobu vstupní XML načteno celé do paměti, nehodí se pro zpracování objemných dokumentů. V PHP je toto rozhraní dostupné v rozšíření DOM.

Knihovna SimpleXML

Funguje na podobném principu jako DOM, ale velmi zjednodušeně, je vhodná pro menší dokumenty.

5 Návrh vlastního rozšíření

Následující kapitola popisuje návrh rozšíření pro Nette Framework, které umožní definici šablon stránek v jazyce XML. Prostor bude věnován především formátu šablon a způsobu práce s nimi. Pro zápis vlastních akcí v šablonách budou definovány zvláštní XML značky, které budou od XHTML značek rozlišeny pomocí jmenných prostorů v XML. Knihovna bude zároveň podporovat snadné použití částí Nette frameworku jako jsou formuláře nebo komponenty a také tvorbu aplikací založených na presenterech, které využívají MVC architekturu frameworku.



Obrázek 5.1 - Vztah XHTML a XML

Použití jazyka XML pro definici šablon stránek může přinést několik výhod. V prvé řadě se jedná o rozšířený jazyk s přesně definovanou syntaxí. Aplikací jazyka XML je navíc také přímo jazyk XHTML pro tvorbu webových stránek (jejich vztah je znázorněn na obrázku 5.1). Odpadá tedy nutnost použití nového jazyka pro definici šablon jako je například Latte. Další výhodou jsou rozsáhlé možnosti, které jsou pro tvorbu a zpracování XML dokumentů k dispozici. Například pro implementaci vlastního rozšíření tak lze použít ten nejvhodnější parser pro daný účel z několika, které PHP nabízí. Poslední podstatná výhoda vychází z již zmíněné příbuznosti s jazykem XHTML. Jelikož jsou vlastní značky umístěny v jiném jmenném prostoru než značky XHTML, lze samotné šablony bez nutnosti zpracování frameworkem zobrazit v prohlížeči, který značky nepatřící do jmenného prostoru XHTML ignoruje. Díky tomu je možné šablony snadněji testovat.

Rozšíření bude integrováno do frameworku stejným způsobem jako Latte. Knihovna tedy bude pracovat jako filtr, který zdrojový kód šablony v jazyce XML převede na její PHP kód, který je následně vykonán. Na celé práci se šablonami v rámci frameworku se tedy změní pouze jejich zápis.

Při návrhu způsobu zápisu vlastních šablon jsem se zaměřil také na jednoduchost použití, které jsem se snažil dosáhnout především využíváním již známé syntaxe (zejména pro zápis výrazů) a také jejího konzistentního používání.

5.1 Formát šablon

Pro použití šablon bude nejprve nutné definovat několik jejich základních částí. Jedná se o výrazy, které umožňují propojení datové nebo aplikační vrstvy aplikace se šablonami. Dále je to pak množina

pomocných funkcí, které lze snadno zapsat ve výrazech šablony, a pomocí kterých lze například vypisovaná data zpracovat před výstupem. Poslední částí jsou knihovny XML značek, které umožňují zápis akcí v šablonách (například cykly apod.). Jak již bylo zmíněno dříve, vlastní značky jsou od XHTML značek odděleny jejich umístěním do zvláštního jmenného prostoru. V případě vlastního rozšíření budou definovány dva jmenné prostory. První obsahuje základní knihovnu značek, druhý je pak použit pro vlastní značky definované uživatelem.

Šablony stránek musí být vždy správně strukturované (well-formed) XML dokumenty. Například tedy musí být vždy uveden jeden kořenový element, i v případě, že se jedná pouze o šablonu vkládanou ze souboru do jiné šablony. V každé šabloně je také nutné uvést deklaraci všech použitých jmenných prostorů.

Soubory se šablonami se ukládají s příponou `.phtml`, která je použita kvůli automatickému vyhledávání šablon v Nette. Soubory musí být rovněž uloženy v kódování UTF-8, stejně jako všechny ostatní zdrojové texty použité s frameworkem.

5.1.1 Výrazy

Výrazy umožňují propojení šablon s dalšími částmi aplikace, typicky s proměnnými, které jsou v šabloně registrovány a které mají být vypsány. Ve vlastních šablonách se výrazy ohraničují složenými závorkami a mohou se vyskytovat pouze v obsahu elementů nebo atributů. Uvnitř výrazů je možné použít libovolný výraz zapsaný v PHP, musí se však vždy jednat o výraz, jehož hodnota je následně použita podle kontextu výrazu (nelze tedy zapsat například příkaz nebo jakékoli jiné konstrukce).

Příklad 5.1 - Ukázky výrazů

```
{$iterator->isOdd ? "odd" : "even"}
{array('item' => $value)}
```

Způsob vyhodnocení výrazu určuje kontext, ve kterém je výraz použit. Jedná-li se o atribut některé ze značek vlastní knihovny, je hodnota výrazu použita podle příslušného významu (např. pro vyhodnocení podmínky apod.). Ve všech ostatních případech je hodnota výrazu vypsána. Pokud je v obsahu atributů některé ze značek z vlastní knihovny výraz obklopen dalším obsahem, dojde ve výsledku ke konkatenaci výrazů s okolními řetězci. Je-li ve stejném případě zadán pouze textový obsah, bude se při zpracování značky považovat za řetězcovou konstantu. Protože v rámci celého zdrojového textu šablony slouží složené závorky pouze k ohraničení výrazů, je nutné je pro jejich použití v dokumentu nebo uvnitř výrazu zapsat pomocí jejich zdvojení jako `{{` nebo `}}`. Ve výrazech je také možné používat všechny standardní funkce z PHP.

5.1.2 Pomocné funkce

Ve výrazech jsou k dispozici také některé pomocné funkce frameworku Nette, které mohou například usnadňovat provádění testů v podmínkách, zajišťovat výpis odkazů na základě zadaných akcí presenterů apod. Dále je zde přístup k helperům Nette, které jsou v šabloně zaregistrovány. Jelikož však všechny tyto funkce nemohou být definovány v místě, ve kterém je kód výsledné šablony v PHP vykonáván, bylo nutné je nějakým způsobem v tomto prostoru zpřístupnit, ale zároveň zajistit jejich pohodlný a co nejkratší zápis v PHP. Proto byly navrženy dvě třídy – `h` pro helpery a `f` pro další funkce, které obsahují statické metody reprezentující požadované funkce. V případě helperů jsou k dispozici i funkce pro správné ošetřování výstupu podle typu dokumentu (např. `escapeHtml()`).

Příklad 5.2 - Ukázka použití helperu a pomocné funkce pro vygenerování textu odkazu na základě zadané akce presenteru ve výrazech

```
{h::capitalize($string)}
{f::link('Product:show')}
```

5.1.3 Komentáře

V XML šablonách je možné používat standardní komentáře, které jsou následně vygenerovány i na výstupu. Pro zápis komentářů, které mohou komentovat například kód šablony a není žádoucí, aby se objevily i na výstupu, byla navržena syntaxe začínající sekvencí `<!---` místo `<!--`.

Příklad 5.3 - Komentář, který nebude ve výstupu po zpracování šablony

```
<!--- Komentář, který nebude ve výstupu -->
```

5.1.4 Knihovna základních značek

V rámci vlastního rozšíření byly navrženy XML značky reprezentující základní akce v šablonách, které jsou při vytváření aplikací využívajících Nette Framework potřebné, ale i takové, které přímo systém Latte nepodporuje. V této části následuje jejich stručný přehled a vysvětlení. Ukázka možností použití značek se pak nachází na v kapitole 7 věnované ukázkové aplikaci. Název XML jmenného prostoru pro tuto skupinu značek je `"xmltmpl-core-tags"` a v této práci bude používána s prefixem `n:`. Atributy značek zapsané kurzívou jsou pro daný tag volitelné.

Řídicí struktury

`<n:if test="{$test}"></n:if>`

Tělo elementu je vykonáno na základě výsledku vyhodnocení obsahu atributu `test`.

`<n:select></n:select>`

Umožňuje specifikovat několik elementů `<n:when>` (viz dále), z nichž může být vykonán nejvýše jeden v závislosti na vyhodnocení podmínky uvedené v elementu, případně lze zadat i část `<n:otherwise>`, která se provede, není-li splněna žádná z podmínek. `<n:select>` může obsahovat pouze tyto dva typy značek.

`<n:when test="{$test}"></n:when>`

Element pro značku `<n:select>`, jehož obsah může být vykonán podle pravdivosti uvedené podmínky.

`<n:otherwise></n:otherwise>`

Element v `<n:select>`, jehož obsah je vykonán, není-li splněna žádná z podmínek `<n:when>`.

`<n:foreach items="{$array}" var="name"></n:foreach>`

Prochází tělo elementu pro každý prvek pole zadaného atributem `items` (může být zadáno také funkcemi jako `range()` nebo lze zadat i objekt implementující rozhraní `Traversable`). Hodnota prvku v aktuálním průchodu je k dispozici v proměnné, jejíž název je zadán atributem `var` (očekává se pouze název, který může být případně zadán i výrazem, nikoli přímo daná proměnná). Uvnitř cyklu je také k dispozici proměnná `$iterator` typu `Nette\Iterators\CachingIterator`, pomocí které lze zjišťovat další užitečné informace o průchodu.

`<n:while test="{$test}"></n:while>`

Prochází tělo elementu, dokud je platná zadaná podmínka.

`<n:continue if="{$test}"/>`

Příkaz `continue` pro použití v těle cyklů, lze specifikovat i podmínku, za které bude vykonán.

`<n:break if="{$test}"/>`

Příkaz `break`, použití je obdobné jako u `continue`.

Vykreslování formulářů

Formuláře definované pomocí frameworku Nette lze podobně jako v Latte i zde vykreslovat pomocí zvláštních značek jak celé automaticky pomocí nastaveného rendereru formuláře, tak manuálně po jednotlivých prvcích. Navíc je však k dispozici i podpora pro automatické vykreslování skupin prvků nebo kontejnerů formuláře.

```
<n:form form="{$form}" attr="{array('class' => 'login')}"/>
```

Umožňuje vykreslení celého formuláře automaticky pomocí nastaveného rendereru. Pomocí atributu `form` lze zadat přímo objekt formuláře, který má být vykreslen, nebo název komponenty, která reprezentuje požadovaný formulář. Způsob, kterým je formulář zadán, je pak rozpoznán při zpracování šablony. Oproti Latte lze také specifikovat XHTML atributy vykresleného formuláře jejich zadáním ve tvaru pole v atributu `attr`.

```
<n:form form="{$form}" attr="{$array}"></n:form>
```

Párová varianta značky je základem pro manuální vykreslení prvků formuláře. Jednotlivé prvky formuláře (viz značky dále) musí být uvedeny uvnitř tohoto tagu, který pouze specifikuje formulář, jehož části se budou vykreslovat. Způsob zápisu značky je pak stejný jako u nepárové varianty.

```
<n:formErrors/>
```

Značka pro vykreslení chyb při odesílání formuláře.

```
<n:formGroup name="name"/>
```

Tento element zajišťuje vykreslení skupiny prvků definované ve formuláři.

```
<n:formContainer name="name"/>
```

Vykreslí zadaný kontejner ve formuláři.

```
<n:formContainer name="name"></n:formContainer>
```

Párový tag pro vykreslování kontejneru formuláře slouží k ručnímu vykreslení jeho jednotlivých prvků. Podobně jako párová značka `form` tedy pouze definuje kontejner, do kterého patří následně vykreslované části.

```
<n:input name="name" attr="{$array}"/>
```

Značka pro vykreslení vstupního pole formuláře včetně formulářových prvků jako jsou odesílací tlačítka, elementy `select`, `textarea` a další.

```
<n:label name="name" attr="{$array}"/>
```

Vykreslí popisek pro zadané pole formuláře.

Definice bloků, dědičnost šablon

Následující značky umožňují definovat části šablon pro jejich pozdější použití. Podobným způsobem jako v Nette je také možné využívat dědičnost šablon.

```
<n:include src="file.phtml" args="{$array}"/>
```

Značka umožňuje vložení šablony zadané názvem souboru v atributu `src`. Ve vložené šabloně jsou k dispozici všechny proměnné a bloky, které byly definovány v původní šabloně. V té jsou pak následně k dostupné i bloky a proměnné z vloženého souboru. Volitelný atribut `args` slouží k předání argumentů, které jsou poté ve vkládané šabloně k dispozici jako proměnné. Argumenty jsou předávány jako asociativní pole, kde klíčem prvku je název proměnné.

```
<n:define name="name"></n:define>
```

Definuje blok, jehož obsah pak může být vložen na jiných místech v šabloně. Uvnitř bloku jsou dostupné všechny proměnné a bloky, které jsou definovány vně, proměnné definované v bloku jsou však pouze lokální. Uvnitř bloku nelze definovat nový blok. Blok také umožňuje vložení sebe sama (viz další značky), čehož lze využít například při vykreslování víceúrovňových seznamů nebo nabídek.

```
<n:insert name="name" args="{$array}"/>
```

Vloží blok s případnými zadanými argumenty.

```
<n:insert name="name" args="{$array}"></n:insert>
```

Párová značka pro vložení bloku umožňuje navíc uvnitř definovat obsah, který bude použit v případě, že zadaný blok neexistuje.

```
<n:extends layout="file.phtml"></n:extends>
```

Tato značka slouží k použití dědičnosti šablon. Bloky definované v rámci dědičnosti musí být uvedeny uvnitř elementu a vždy nahrazují bloky se stejným názvem, které jsou definovány později. Použít lze i vícenásobnou dědičnost. Soubor rozšiřované šablony není nutné uvádět v případě, kdy chceme použít výchozí šablonu layoutu. Pokud je v šabloně uvedena značka `<n:extends>`, je veškerý obsah souboru kromě definice bloků a proměnných ignorován. Díky tomu je možné definované bloky obklopit dalšími XHTML elementy a samostatnou šablonu zobrazit v prohlížeči pro její testování.

Další značky

```
<n:control name="name" mode="mode" args="{$array}"/>
```

Ve značce pro vykreslení komponenty lze kromě názvu uvést také způsob jejího vykreslení daný atributem `mode`. Příklad 5.4 pro názornost ukazuje zápis značky a jemu odpovídající kód v PHP. V atributu `args` lze specifikovat argumenty pro vykreslení komponenty.

Příklad 5.4 - Ukázka použití značky pro vykreslení komponenty a odpovídajícího zápisu v PHP

```
<n:control name="product" mode="small"/>
$control->getComponent('product')->renderSmall();
```

```
<n:set var="name" value="{$value}"/>
```

Tento element slouží k definici proměnné se zadaným jménem v šabloně nebo k přiřazení její nové hodnoty.

```
<n:execute stmt="{var_dump($var);}" if="{$test}"/>
```

V šabloně lze zadat příkaz v jazyce PHP, který bude při vykonávání kódu šablony proveden. Za příkazem musí být vždy uveden středník, kterým je také možné oddělit více příkazů. Volitelně lze specifikovat podmínku, v závislosti na které bude příkaz vykonán.

```
<n:remove></n:remove>
```

Poslední element knihovny slouží k odstranění obsahu uvnitř značky při zpracování šablony. Toho je možné využít například při testování samostatných šablon, kdy ještě nemáme k dispozici data z aplikace a potřebujeme je nahradit vlastním obsahem.

5.1.5 Definice vlastních značek

Vlastní knihovna bude kromě použití základních značek umožňovat také definovat značky vlastní. Tato možnost spočívá v definici PHP kódu pro vytvářené značky, který bude pro danou značku použit při generování vlastního kódu šablony. Pro vlastní značky bude v rámci knihovny k dispozici zvláštní XML jmenný prostor, jehož název je `"xmltmpl-user-tags"` a v této práci je používán s prefixem `m:`. Konkrétní způsob definice bude popsán na konci kapitoly 6.2.

5.1.6 Další vlastnosti

Při návrhu syntaxe šablon bylo nutné brát v potaz také některá omezení, která vyplývají z možností jazyka XML. Jedná se zejména o možnosti použití některých znaků, které mají v XML zvláštní význam, jako jsou například znaky `<` a `&`. Další omezení vyplývají z použití XML parseru pro

zpracování vlastních šablon, který při zpracování dokumentu například převádí XML entity na odpovídající znaky a také interpretuje některé části dokumentu, které mohou být určeny pro výsledné XHTML (například deklarace typu dokumentu nebo XML deklarace). Aby bylo chování vlastní knihovny konzistentní, je formát šablon přizpůsoben XML parseru. Šablonami jsou tedy XML dokumenty, které jsou před vygenerováním samotného výstupu zpracovány parserem, a s těmito skutečnostmi je nutné při jejich zápisu počítat.

Použití znaků < a &

Použití uvedených znaků je nutné zejména ve výrazech, například chceme-li použít operátory pro porovnání. Jelikož se ale jedná o znaky mající v XML dokumentu speciální význam a nelze je v obsahu atributů použít, je nutné je nahradit odpovídajícími XML entitami.

XML deklarace a typ dokumentu

XML deklarace včetně typu dokumentu jsou vždy při zpracování šablon vygenerovány automaticky a není nutné je v šablonách uvádět. Jako typ dokumentu je vždy použita přechodová verze XHTML (transitional).

XML entity

Při zpracování šablony jsou XML entity převedeny na odpovídající znaky. Chceme-li, aby byl daný znak zapsán jako entita i na výstupu, je nutné jej například v případě znaku `&` zapsat jako `&amp;`.

Obsah atributů

Posledním omezením je způsob, jakým XML parser zpracovává hodnoty atributů. Konkrétně jde o atributy, jejichž hodnoty jsou uzavřeny v apostrofech. Pokud jsou v tomto případě v obsahu atributu uvozovky (například ohraničující řetězec ve výrazu), jsou automaticky převedeny na odpovídající entity. Díky tomu již poté není možné rozpoznat, zda byly v šabloně původně zapsány uvozovky nebo pouze entity. Výraz obsahující uvozovky pak tedy nelze jednoznačně interpretovat. Z tohoto důvodu je nutné vždy v těchto případech hodnotu atributu uzavřít do uvozovek a případný řetězec ve výrazu do apostrofů. Situace je znázorněna v příkladu 5.5.

Příklad 5.5 - Ukázka dvou možností zápisu totožného atributu, z nichž je podporována pouze první, jelikož druhý případ nelze správně zpracovat

```
attr="{array('class' => 'odd')}"
attr='{array("class" => "odd")}'
```

6 Implementace rozšíření

Implementace vlastní knihovny umožňující definici šablon stránek v jazyce XML pro Nette Framework vychází z návrhu šablon popsaného v předchozí kapitole. Způsob realizace rozšíření se odvíjí od možností, které jsou pro zpracování šablon v Nette k dispozici. Protože framework poskytuje kompletní architekturu pro tvorbu MVC aplikací, je podpora šablon jeho nezbytně nutnou součástí. Díky obecnému návrhu způsobu práce se šablonami ve frameworku je dobře možné tuto část rozšířit.

Před vykreslením samotné šablony umožňuje Nette aplikovat na její zdrojový text množinu filtrů, které je možné v aplikaci libovolně registrovat. Takovým filtrem je také Latte, které je ve frameworku k dispozici ve výchozím nastavení. Latte pak pouze převádí vlastní způsob zápisu šablon na kód šablony v jazyce PHP, který je následně vykonán. Celý tento mechanismus je součástí vykreslování šablon, jejichž podpora je ve frameworku integrována. Propojení šablon se zbytkem aplikace je tak zcela nezávislé na způsobu jejich zápisu, který lze díky filtrům libovolně změnit.

Vlastní rozšíření tedy bude realizováno jako filtr, který obdrží zdrojový kód šablony v jazyce XML a následně jej převede na výsledný kód, kterým bude šablona vykreslena. Během převodu je pak k dispozici pouze zmíněný zdrojový text, žádné jiné reference na další části aplikace dostupné nejsou. Převod se rovněž provádí vždy jen při prvním vykreslení zadaného souboru, poté je výsledek uložen do cache a není tak nutné až do další případné změny šablony aplikaci filtru znovu provádět. Po převodu šablony je pak její výsledný kód vyhodnocen funkcí `eval()`. Předtím jsou ještě do dané oblasti platných identifikátorů pomocí funkce `extract()` převedeny všechny proměnné, které jsou v šabloně registrovány. Díky tomu je možné tyto proměnné ve zdrojových textech šablon používat jako lokální.

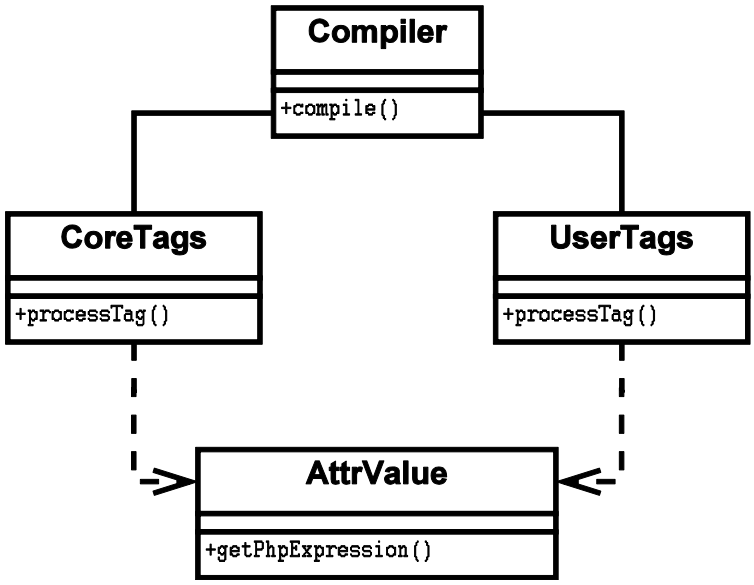
Následující podkapitoly budou věnovány popisu implementace vlastního filtru a později pak také možnostem jeho použití.

6.1 Popis implementace

Důležitou součástí implementace je XML parser, pomocí kterého lze zdrojový soubor šablony zpracovat. Pro zpracování souboru šablony postačuje postupný průchod dokumentem a čtení jeho částí. Proto je vhodné použití parseru typu SAX, který navíc disponuje vyšší rychlostí a menší paměťovou náročností. Pro potřeby aplikace byla zvolena knihovna `XMLReader`, která proti extenzi `XML Parser` nabízí přívětivější rozhraní. Zároveň podporuje zpracování všech součástí XML dokumentů potřebných pro vlastní rozšíření, jako například jmenných prostorů.

Vlastní kód knihovny tvoří několik tříd. Hlavní z nich, která poskytuje rozhraní filtru a řídí zpracování šablony je třída `Compiler`. Zpracování základních značek a generování jejich PHP kódu

obstarává třída `CoreTags`, pro vlastní značky funguje obdobně třída `UserTags`. Třída `AttrValue` pak zajišťuje zpracování výrazů v šablonách. Dále je součástí rozšíření třída `TempStorage`, která slouží jako pomocné úložiště dat ve vygenerovaném PHP kódu šablon. Třídy `h` a `f` pak zajišťují přístup k pomocným funkcím ve výrazech. Vztah hlavních tříd programu je znázorněn na diagramu 6.1.



Obrázek 6.1 - Diagram tříd pro základní třídy knihovny

6.1.1 Zpracování šablon

Jako první je při zpracování šablony volána metoda `compile()` třídy `Compiler`. Zde jsou postupně načítány jednotlivé části XML dokumentu způsobem, který umožňuje knihovna `XMLReader`. Ta postupně předává jednotlivé uzly zdrojového souboru, jako jsou elementy, komentáře, obsah elementů a další. Každý typ uzlu je pak zpracován zvláštní metodou ve třídě `Compiler`, například pro elementy je to metoda `processElement()`. Díky rozhraní `XMLReader` je pak možné pro každý uzel XML souboru zjistit další informace, například URI identifikující XML jmenný prostor pro právě zpracovávaný element. Díky tomu pak lze správně rozlišit mezi XHTML elementy a značkami vlastní knihovny. Výsledkem zpracování všech uzlů je poté zdrojový kód šablony v PHP.

Zpracování výrazů

Jak již bylo uvedeno dříve v kapitole věnované návrhu vlastního rozšíření, výrazy se mohou v šablonách vyskytovat pouze v obsahu atributů nebo elementů. Výskyt v jiných částech, například přímo v elementech mezi jednotlivými atributy, by měl vliv na validnost dané XML šablony, protože složené závorky ohraničující výrazy se v těchto částech vyskytovat nemohou. Výrazy se tedy vždy

nacházejí v rámci jednoho zpracovávaného uzlu XML šablony – obsahu elementu nebo atributu. Před samotným zpracováním výrazů je ještě nutné zkontrolovat správnost jejich zápisu a po něm pak zajistit převod případných zdvojených složených závorek umožňujících zápis závorek mimo význam ohraničení výrazů. Tyto operace zajišťují metody `validateText()` a `unescapeText()` třídy `Compiler`. V rámci knihovny je pak zpracování výrazů závislé na kontextu jejich použití.

Pokud je výraz použit v atributech vlastních značek knihovny, je výraz zpracován ve smyslu použití dané značky, například pro vyhodnocení podmínky v elementu `<n:if>`. K tomuto vyhodnocení však dochází až při vykonávání již vygenerovaného kódu šablony, a proto je nutné výrazy správně vkládat do tohoto kódu. To platí zejména pro případy, kdy je výraz v atributu obklopen dalším obsahem, jak bylo popsáno v části 5.1.1, nebo je v obsahu atributu zadán pouze text bez výrazu. V těchto situacích je nutné zajistit převod zadaného obsahu na odpovídající PHP výraz. Tento převod zajišťuje třída `AttrValue` pomocí metody `getPhpExpression()`, která ze zadané hodnoty atributu vytvoří odpovídající výraz v PHP pomocí regulárních výrazů. Ukázka převodu je znázorněna v příkladu 6.1. V případě některých atributů může být nutné akceptovat v jeho hodnotě pouze textový obsah neobsahující výrazy. Jedná se především o atributy, jejichž hodnotu je nutné znát již při generování kódu šablony. Pro tyto účely obsahuje třída `AttrValue` metodu `getString()`, která vrací pouze zadaný obsah atributu nebo `NULL`, obsahuje-li hodnota atributu výrazy.

Příklad 6.1 - Ukázka použití výrazu v obsahu atributu a odpovídajícího PHP kódu vygenerovaného ve výsledné šabloně

```
src="{$name}.phtml"
$name.'.phtml'
```

Ve všech ostatních případech použití výrazů je hodnota výrazu vypsána v daném místě dokumentu. Ke zpracování tohoto případu slouží metoda `processExpressions()` třídy `Compiler`, která všechny výskyty výrazů v zadaném obsahu uzlu nahradí jejich výpisem pomocí konstrukce `echo()` ve výsledném kódu šablony.

Zpracování vlastních značek

Značky knihovny jsou zpracovány třídou `CoreTags` způsobem popsaným v této části. Podobně pracuje také třída `UserTags` určená pro vlastní značky. Pokud zpracování šablony ve třídě `Compiler` narazí na značku z XML jmenného prostoru knihovny, předá ji příslušné třídě pro její zpracování voláním metody `processTag()` dané třídy. Kromě názvu značky je vždy předán i příznak, zda se jedná o značku prázdnou, otevírací nebo uzavírací, a také pole atributů značky, které obsahuje ke každému atributu jeho hodnotu ve formě objektu třídy `AttrValue`.

Konkrétní třída (např. `CoreTags`) pak obsahuje metody pro zpracování jednotlivých značek. Pro vyhledání metody, která zpracuje zadanou značku, slouží asociativní pole, které ke každému názvu podporované značky (klíč v poli) obsahuje pole názvů metod určených pro jednotlivé typy elementů (v pořadí prázdný, otevírací, uzavírací). Princip je pro názornost uveden v příkladu Příklad 6.2.

Příklad 6.2 - Ukázka způsobu mapování značek na příslušné metody pro jejich zpracování

```
$tags['foreach'] = array(NULL, 'tagForeach', 'tagEndForeach');
```

Není-li některý typ elementu podporován, je místo názvu metody uvedena hodnota `NULL`. Po vyhledání správné metody pro zpracování značky je pak tato metoda volána pomocí funkce `call_user_func()`, která umožňuje volání funkce na základě specifikace jejího názvu a případných argumentů. Jako argumenty jsou předány atributy značky, které tak má každá metoda k dispozici pro generování PHP kódu šablony. V příkladu 6.3 je ukázka metody pro zpracování otevírací značky `<n:if>`.

Příklad 6.3 - Ukázka metody pro zpracování otevírací značky `<n:if>`

```
private function tagIf($attrs)
{
    // Ověření zadaných atributů
    if (!array_key_exists('test', $attrs))
        return false;

    // Vygenerování kódu pro výslednou šablonu
    $res = "<?php if (".$attrs['test']->getPhpExpression()."): ?>";
    $this->compiler->appendOutput($res);

    return true;
}
```

6.1.2 Generování PHP kódu šablon

Následující odstavce popisují řešení některých specifických problémů, kterými se bylo nutné při návrhu generovaného kódu šablon zabývat. Pro některé značky také bylo potřeba navrhnout pomocné datové struktury pro dočasné ukládání informací během vykreslování již zpracovaných šablon. Většina těchto mechanismů vychází ze způsobu, jakým generuje PHP kód šablon v Nette také Latte. Konkrétní příklady generovaného kódu jsou uvedeny v kapitole 7 věnující se ukázkové aplikaci.

Třída `TempStorage`

Pro ukládání veškerých dočasných informací spojených s vlastním rozšířením během vykonávání kódu šablon slouží třída `TempStorage`. Poskytuje například zásobníky formulářů nebo iterátorů v cyklech `foreach`. Zásobníky bylo nutné použít v případech, kdy je nutné umožnit vnořování stejných typů značek, které zároveň vyžadují uložení nějakých dat. Jedná se například o značku `<n:foreach>`, která v těle cyklu definuje proměnnou `$iterator`, a při vnoření cyklů je nutné zajistit, aby na každé úrovni vnoření byla k dispozici právě proměnná příslušící danému cyklu.

Formuláře

Při manuálním vykreslování prvků formuláře je nutné mít v PHP kódu šablony pro jejich vykreslení k dispozici referenci na objekt formuláře, do kterého dané prvky patří, definovaný ve značce `<n:form>`. Pro její uložení slouží objekt třídy `TempStorage`, který je v šabloně k dispozici.

Bloky

Bloky jsou v PHP kódu šablon reprezentovány funkcemi, stejně jako v Latte. To umožňuje jednoduše realizovat jejich vložení, kdy stačí definovanou funkci zavolat. Jelikož mají být ve vloženém bloku k dispozici všechny proměnné definované v místě vložení bloku, jsou nejprve získány pomocí funkce `get_defined_vars()`, a následně předány funkci reprezentující blok. Ta poté na svém začátku pomocí funkce `extract()` vloží předané proměnné do svého lokálního prostoru.

Při definici bloků je dále nutné rozlišit mezi případem, kdy je blok definován vícekrát v rámci dědičnosti (má se použít blok definovaný v hierarchii dědičnosti jako první), a případem, kdy je blok přepsán ve vloženém souboru. Proto se ukládají definované bloky také do objektu třídy `TempStorage` v šabloně a nastavují se patřičné příznaky, které zmíněné případy rozliší. Třída `TempStorage` pak poskytuje metodu, která automaticky vrátí název bloku, který se má použít, na základě nastavených příznaků.

Vkládání šablon a dědičnost

Vkládání šablon je realizováno podobným způsobem jako dědičnost. Při vykreslení šablony je v jejím PHP kódu vytvořen objekt třídy `Nette\Templating\FileTemplate` reprezentující vkládanou nebo rozšiřovanou šablonu, kterému jsou předány proměnné a bloky definované v aktuální šabloně. Vytvořená šablona je následně vykreslena pomocí své metody `render()`, čímž je zajištěno vložení jejího obsahu.

6.2 Použití rozšíření

Zdrojové soubory knihovny se na přiloženém CD nachází v adresáři `XmlTmpl` (viz obsah CD v příloze). Všechny třídy kromě `h` a `f` pro použití ve výrazech jsou také umístěny ve jmenném prostoru `XmlTmpl`.

Pro použití rozšíření je nutné jeho adresář nejprve zkopírovat do požadovaného umístění ve vlastní aplikaci. Při použití adresářové struktury Nette frameworku to může být například složka `libs` pro knihovny třetích stran.

Jedná-li se o aplikaci využívající architekturu frameworku, není dále nutné zdrojové soubory knihovny načítat v samotném programu díky automatickému načítání tříd (v případě, že je správně nastaveno). V opačném případě je nutné načíst soubor knihovny s názvem `Compiler.php`, například pomocí konstrukce `require`.

Posledním krokem je nastavení knihovny jako filtru pro zpracování šablon. V Nette aplikaci je nejvhodnějším místem `BasePresenter` (předek všech presenterů v programu), kde lze registraci provést přidáním metody `templatePrepareFilters()`, jak ukazuje příklad 6.4.

Příklad 6.4 - Ukázka registrace filtru pomocí metody `templatePrepareFilters()`

```
public function templatePrepareFilters($template)
{
    $template->registerFilter(new XmlTmpl\Compiler());
}
```

Další možností je registrace filtru přímo v objektu šablony, viz ukázka 6.5 (obsahuje navíc i registraci helperů, která je nutná při manuálním použití šablon).

Příklad 6.5 - Registrace filtru včetně helperů přímo v objektu šablony

```
$template->onPrepareFilters[] = function($template) {
    $template->registerFilter(new XmlTmpl\Compiler());
};
$template->registerHelperLoader('Nette\Templating\Helpers::loader');
```

Nyní je již možné používat šablony v definovaném XML formátu. Práce se šablonami v aplikaci pak zůstává nezměněna.

Definice vlastních značek

Vlastní značky jsou v knihovně umístěny v XML jmenném prostoru `"xmltmpl-user-tags"` a v rámci této práce jsou používány s prefixem `m:`. Možnost definice vlastních značek vychází ze způsobu implementace základní knihovny značek.

Definice se provádí pomocí metody `registerUserTag()` třídy `Compiler`. Tato metoda přijímá jako první parametr název definované značky a dále pak funkce pro zpracování jednotlivých typů elementů. Jako první se uvádí funkce pro zpracování prázdného elementu, následuje funkce pro zpracování otevírací značky a nakonec funkce pro značku uzavírací. Jako jednotlivé funkce lze registrovat callbacky nebo anonymní funkce v PHP.

Funkce také mohou volitelně přijímat jako parametr pole atributů (funkce pro uzavírací značku může také obdržet atributy zadané v otevírací značce), které bude jako klíče obsahovat jejich názvy a jako hodnoty objekty třídy `AttrValue`. Uvnitř funkce tedy lze použít metody této třídy například pro vygenerování PHP výrazu ze zadané hodnoty atributu.

Každá funkce pak musí vracet řetězec obsahující PHP kód, který je pro danou značku vygenerován do výsledné šablony.

Příklad 6.6 - Ukázka definice vlastní značky `<m:if>` s atributem `test`

```
function tagIf($attrs) {
    return "<?php if (".$attrs['test']->getPhpExpression()."): ?>";
}
function tagEndIf() {
    return "<?php endif; ?>";
}
$compiler->registerUserTag('if', NULL, 'tagIf', 'tagEndIf');
```

7 Ukázková aplikace

Pro demonstraci základních možností použití a způsobu fungování knihovny byla v rámci práce vytvořena také ukázková aplikace. Jedná se o jednoduchou knihu návštěv obsahující formulář pro zadávání nových příspěvků na jedné stránce a jejich zobrazení na druhé. Pro lepší demonstraci některých částí knihovny aplikace využívá MVC architekturu frameworku. Příspěvky jsou ukládány v XML souboru na straně serveru, aby bylo možné aplikaci jednoduše nainstalovat a používat.

Zdrojové soubory programu jsou k dispozici také na přiloženém CD. Pro zprovoznění aplikace stačí její adresář přesunout na server.

Struktura aplikace

Nejdůležitější části aplikace se nachází v adresáři `app`. Ve složce model je k dispozici třída `MsgStorage` reprezentující datový model aplikace. Tato třída zajišťuje potřebné operace s XML souborem pro ukládání a načítání příspěvků, pro účely demonstrace však není podstatná. V adresáři `presenters` je umístěn hlavní presenter aplikace `HomepagePresenter`. Šablony jsou umístěny ve složce `templates`, kde se nachází šablona layoutu (pro dědičnost šablon) a v adresáři `Homepage` pak šablony reprezentující konkrétní views presenteru.

`HomepagePresenter` definuje dvě akce. První pro zobrazení příspěvků (`default`) a druhou pro vykreslení formuláře umožňujícího příspěvek přidat (`addMessage`). Těmto akcím také odpovídají šablony ve složce `Homepage`. Zdrojový soubor presenteru ještě obsahuje definici komponenty pro formulář a metodu pro zpracování jeho odeslání.

Vygenerovaný kód šablon je možné prohlížet po jejich prvním zpracování a uložení do cache v adresáři `temp/cache/_Nette.FileTemplate`.

Příklady

Pro demonstraci následují některé ukázky ze šablon aplikace a také části generovaného PHP kódu. Proměnná `$_temp` ve vygenerovaných šablonách reprezentuje objekt třídy `TempStorage` pro dočasné ukládání informací.

Příklad 7.1 - Ukázka použití pomocných funkcí pro práci s odkazy. Atribut `class` je nastaven v závislosti na tom, zda odkaz vede na aktuálně zobrazenou stránku.

```
<a href="{f::link('Homepage:')}"
   class="{f::isLinkCurrent('Homepage:') ? 'current' : ''}">
Zobrazit zprávy</a>
```

Příklad 7.2 - Příklad výpisu prvků pole v cyklu `foreach` s využitím proměnné `$iterator`.

```
<n:foreach items="{$messages}" var="msg">
    <div class="{$iterator->odd ? 'odd' : 'even'}">
        <h2>{$msg['subject']}</h2>
        <p class="from">{$msg['from']} ({$msg['email']})</p>
        <p>{$msg['text']}</p>
    </div>
</n:foreach>
```

Příklad 7.3 - Ukázka vykreslení části formuláře zadaného pomocí názvu komponenty s nastavením atributu `id`.

```
<n:form form="msgForm" attr="{array('id' => 'msgForm')}">
    <table>
        <tr>
            <td><n:label name="email"/></td>
            <td><n:input name="email"/></td>
        </tr>
    </table>
</n:form>
```

Příklad 7.4 - Příklad základních částí kódu, které jsou vygenerovány pro formulář `'msgForm'` z Příklad 7.3.

```
// Test, zda byl formulář zadán jako objekt nebo název komponenty
$_temp->pushForm(is_object('msgForm') ? 'msgForm' :
    $_control['msgForm']);
// Nastavení zadaných atributů
$_temp->topForm()->getElementPrototype()
    ->addAttributes(array('id' => 'msgForm'));
// Vykreslení částí formuláře
$_temp->topForm()->render('begin');
$_tmp = $_temp->topForm();
echo $_tmp['email']->label->render();
echo $_tmp['email']->control->render();
$_temp->popForm()->render('end');
```

Příklad 7.5 - Realizace vložení bloku s názvem `'content'` v PHP kódu šablon. Metoda `getBlockFunction()` třídy `TempStorage` vrací název funkce reprezentující daný blok způsobem popsaným v části 6.1.2.

```
call_user_func($_temp->getBlockFunction('content'),
    get_defined_vars());
```

8 Závěr

Výsledkem mé práce je rozšíření pro Nette Framework, které umožňuje definovat šablony stránek v jazyce XML. Pro zápis zvláštních akcí, které jsou v šablonách nutné, byly navrženy zvláštní XML značky. Propojení šablon s modelovou vrstvou aplikace je realizováno pomocí výrazů, jejichž podobu bylo nutné v rámci řešení také navrhnout. Vytvořená knihovna základních značek vychází z možností frameworku, které je nutné podporovat, byly ale také přidány některé vlastní značky, například pro vykreslování skupin formulářových prvků. Pro uživatele je dále k dispozici možnost definice vlastních značek pomocí PHP. Knihovna je pak realizována jako filtr, který obdrží šablonu v jazyce XML a převede ji na odpovídající reprezentaci v PHP. Pomocí ní je šablona následně vykreslena. V rámci práce byla také vytvořena ukázková aplikace, která demonstruje možnosti knihovny.

Implementace byla úspěšně otestována v prostředích Windows a Linux pro PHP ve verzi 5.3 a 5.4.

Pro další vývoj rozšíření byla navržena některá další vylepšení. V prvé řadě může jít o rozšíření knihovny značek o další užitečné elementy. Jedná se například o značky usnadňující vykreslování tabulek, které byly v původním návrhu. Dále by mohlo jít o optimalizaci generovaného zdrojového kódu šablon a také některých částí knihovny. Také by mohla být přidána podpora pro automatické ošetřování vypisovaných hodnot, jako je tomu v Latte.

Literatura

[1] *Nette Framework* [online]. [cit. 2013-05-12]. Dostupné z: http://nette.org/cs/

[2] *Symfony* [online]. [cit. 2013-05-12]. Dostupné z: http://symfony.com/

[3] *CakePHP* [online]. [cit. 2013-05-12]. Dostupné z: http://cakephp.org/

[4] Facelets. In: *Wikipedia* [online]. 2001- [cit. 2013-05-12]. Dostupné z: http://cs.wikipedia.org/wiki/Facelets

[5] *The Java EE 6 Tutorial* [online]. [cit. 2013-05-12]. Dostupné z: http://docs.oracle.com/javaee/6/tutorial/doc/index.html

[6] *PHPTAL* [online]. [cit. 2013-05-12]. Dostupné z: http://phptal.org/

[7] KOSEK, Jiří. *PHP a XML*. 1. vyd. Praha: Grada, 2009, 367 s. ISBN 978-80-247-1116-4.

[8] XML schémata. [online]. [cit. 2013-05-12]. Dostupné z: http://www.kosek.cz/xml/schema/index.html

[9] Namespaces in XML 1.0 (Third Edition): W3C Recommendation 8 December 2009. [online]. [cit. 2013-05-12]. Dostupné z: http://www.w3.org/TR/REC-xml-names/

[10] VASWANI, Vikram. *XML and PHP*.

[11] *PHP* [online]. [cit. 2013-05-12]. Dostupné z: http://php.net/

Příloha A: Obsah CD

Přiložené CD má následující strukturu:

- doc/ – projektová dokumentace
- examples/ – adresář obsahující ukázkovou aplikaci
- Nette/ – Nette Framework
- thesis/ – text bakalářské práce
- XmlTmpl/ – zdrojové soubory vlastního rozšíření