VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

RÁMEC PRO TVORBU KOMPONENT V PROSTŘEDÍ NETTE

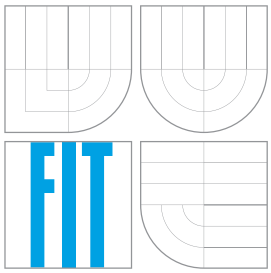
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL KVITA

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

RÁMEC PRO TVORBU KOMPONENT V PROSTŘEDÍ NETTE

A COMPONENT DESIGN FRAMEWORK IN NETTE ENVIRONMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL KVITA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK BURGET, Ph.D.

BRNO 2013

Abstrakt

Tato práce se zabývá návrhem a implementací rámce pro zjednodušení tvorby, správy a použití komponent v českém PHP frameworku Nette a jeho šablonovacím jazyce Latte. Projekt je realizován jako knihovna za využití systému rozšíření Nette. Knihovna příslušně využívá značkovací jazyk XML a interní konfigurační formát Nette nazvaný NEON. Součástí textu je i analýza částí Nette Frameworku relevantních pro řečenou problematiku. Výsledné řešení je demonstrováno na jednoduché aplikaci a v závěru jsou nastíněny možnosti budoucího rozšíření.

Abstract

This bachelor thesis describes the design and implementation of framework for simplification of creating, managing and using of components in Czech PHP framework Nette and its template language Latte. Project is realized as a library with help of the system of extensions in Nette. The library respectively uses markup language XML and internal configuration format of Nette called NEON. Text also contains analysis of that parts of Nette, which are relevant for this topic. Final solution is demonstrated on simple application and in conclusion future possible expansion is discussed.

Klíčová slova

Web, PHP, Nette Framework, komponenty, NEON, Latte, XML, XML schémata, JSON

Keywords

Web, PHP, Nette Framework, components, NEON, Latte, XML, XML schema, JSON

Citace

Michal Kvita: Rámec pro tvorbu komponent v prostředí Nette, bakalářská práce, Brno, FIT VUT v Brně, 2013

Rámec pro tvorbu komponent v prostředí Nette

Prohlášení

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

.....
Michal Kvita
14. května 2013

Poděkování

Rád bych poděkoval vedoucímu mé práce panu Ing. Radku Burgetovi, Ph.D. za hodnotné rady a odborné vedení, dále děkuji Ivanu Hadámkovi za poskytnutí rodinné podpory během tvorby této práce.

Obsah

Kapitola 1

Úvod

S rozvojem internetu a webových technologií se v posledním desetiletí ustálil pojem webová aplikace. Pod tímto názvem je možné si představit interaktivní médium, které uživatel ovládá prostřednictvím webového prohlížeče. Od klasické statické webové stránky se však liší dynamickými vlastnostmi z hlediska klientské části, serverové části či vhodně realizovanou kombinací obou částí. Webové aplikace mají široké možnosti využití a v některých oblastech jsou výhodnější k nasazení než jejich desktopové protějšky. Běžně se tak objevují jako informační systémy, nejrůznější portály, sociální sítě a další nástroje v internetovém světě, kde se jako důležitý aspekt objevuje multimediální interaktivita či rychlé spojení s okolním světem. Vytváření takovýchto aplikací je komplexní problematikou, což klade na jejich tvůrce požadavky z mnoha různých okruhů znalostí. Lze jmenovat například značkovací jazyky webu HTML či XHTML, skriptovací jazyk JavaScript pro manipulaci s objektovým modelem stránky na straně klienta nebo některou ze serverových technologií pro dynamické generování obsahu jako např. CGI skripty, jazyk PHP či technologie ASP.NET.

Český PHP framework Nette je jedním z nástrojů, který programátorům výrazně usnadňuje vývoj webových aplikací na serverové straně. Mezi jeho přední vlastnosti patří i možnost tvorby tzv. komponent. Jedná se o části kódu, které jsou při správném návrhu znovupoužitelné v rámci jedné aplikace či dokonce napříč několika různými projekty. Komponenty mohou efektivním způsobem komunikovat pomocí mechanismu signálů, zařazovat se do stromové hierarchie rodičů a potomků a přes šablonovací jazyky, jako je např. Latte, vykreslovat v různých podobách jako části webové stránky. Tyto jmenované aspekty jsou dále podstatné i pro sdílení takového kódu s ostatními programátory nebo správu uživatelského rozhraní dané webové aplikace.

1.1 Cíl práce

Cílem této práce je vytvořit nástavbu nad Nette Frameworkem v podobě knihovny, která usnadní práci s komponentami. Tento záměr se rozděluje na vícero prvků. Primárně lze jmenovat snazší vytváření komponent skrze generování samotných tříd objektů pomocí deklarativního zápisu ve značkovacím jazyce XML. Na základě toho také přidat efektivnější tvorbu instancí komponent jednotlivých vygenerovaných tříd. Rámec také přidává novou základní třídu pro vykreslitelné komponenty s rozšířenou funkcionalitou a zjednodušuje práci se šablonami přidáním nového filtru a podpůrných maker. Správná funkčnost komponenty vykreslené jako části webové stránky často závisí i na připojení externích souborů, jako např. skriptů v jazyce JavaScript či kaskádových stylů jazyka CSS – práce by měla na-

bídnout jednodušší řešení i této problematiky. Mezi důležité vlastnosti patří taktéž to, aby daná komponenta mohla spolehlivě komunikovat se svým okolím, aniž by ztratila vlastnost znovu-použitelnosti a mohla uchovávat svůj vnitřní stav. Všechny tyto diskutované aspekty budou souhrnně obsaženy v navrhované knihovně, která se realizuje za využití interního systému pro rozšíření v Nette. Název vytvářené knihovny je **Nexi**. Součástí řešení práce bude i demonstrační aplikace a vizualizační nástroj pro vytvářené komponenty.

1.2 Popis jednotlivých kapitol

Kapitola 1 obsahuje úvodní slova a stanovené cíle této práce, kapitola 2 je zaměřena na teoretickou analýzu Nette Frameworku, na popis jeho částí a širší rozbor těch vlastností, které jsou primárně směrodatné pro řešení této práce. Kapitola 3 se zabývá značkovacím jazykem XML, jeho jmennými prostory a validačními schématy. Dále je zde stručně rozebrán i formát JSON. Kapitola 4 obsahuje kompletní návrh vytvářené knihovny Nexi vycházející z uvedených cílů práce a teoretických analýz předchozích kapitol. Kapitola 5 se zabývá samotnou implementací knihovny, objektovým návrhem a využitými technologiemi a postupy. V kapitole 6 je implementovaná knihovna testována na jednoduché demonstrační aplikaci. Závěr v kapitole 7 shrnuje výsledky této práce, jejího využití a navrhuje možná rozšíření do budoucna.

Kapitola 2

Nette Framework

Nette Framework (dále jen Nette) je nástroj napsaný v jazyce PHP[1] pro usnadnění a zefektivnění vývoje webových aplikací. Vznikl v roce 2004 a jeho autorem je český programátor David Grudl. Od roku 2008 je šířen jako open source projekt pod licencemi New BSD a GNU General Public License (GPL) a za aktivní podpory komunity vývojářů se stal jedním z nejpoužívanějších webových frameworků v České republice. Z jeho předních vlastností lze pro účely této práce jmenovat následující:

- Kvalitní objektový návrh vytvářených aplikací vyznačující se architektonickým vzorem Model-View-Controller (MVC)
- Rozšíření vlastností PHP o přidání podpory událostmi řízeného programování
- Rychlá konfigurace aplikací skrze interní formát NEON
- Rozsáhlá podpora programovací techniky Dependency Injection (DI)
- Možnost tvorby komponent, tj. znovupoužitelných částí kódu
- Jednoduchý způsob přidávání nových rozšíření
- Bohatý šablonovací jazyk Latte
- Zabezpečení proti častým útokům na webové aplikace jako např. Cross-site scripting (XSS), Cross-request forgery (CSRF), session hijacking, session fixation a další
- Kvalitní ladící nástroje
- Automatické načítání tříd
- Vysoký výkon na základě využívání vyrovnávací paměti cache

V následujících sekcích je proveden hlubší teoretický rozbor některých z uvedených vlastností. V textu je místy odkazováno na konkrétní názvy tříd a rozhraní a jejich metody. Tyto informace vždy vychází z API Nette Frameworku[2] verze 2.0.10 stabilní uvolněné 6.3. 2013 pro PHP 5.3 nebo PHP 5.4. Ve většině případů je uvedeno plně kvalifikované jméno dané třídy či rozhraní. Zkrácená jména bez jmenných prostorů jsou uváděna tam, kde tato informace výrazně zhoršuje přehlednost zápisu nebo jasně vyplývá ze zbytku textu.

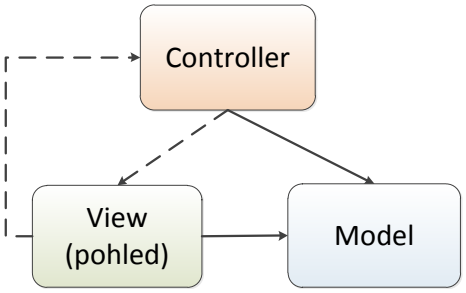
[1] PHP je buďto rekurzivní zkratka PHP Hypertext Preprocessor či původní význam Personal Home Page

[2] API je dostupné na adrese: `http://api.nette.org/2.0/`, případně je i zahrnuto v rámci API knihovny Nexi vyvíjené v této práci

2.1 MVC aplikace a presentery

Aplikace v Nette jsou postavené na známé architektuře Model-View-Controller (dále jen MVC), její schéma je zobrazeno na obrázku 2.1. Obecně tento systém rozčleňuje aplikaci do tří základních vrstev [2]:

- **Model** – Datová a funkční složka aplikace. Model zajišťuje svůj vnitřní stav a ven nabízí pevné rozhraní, přes které jej lze ovládat, tzn. měnit jeho stav či zjišťovat stav aktuální. Při správné implementaci modelová vrstva nikdy neví o existenci pohledů a controllerů, tedy musí vždy fungovat jako samostatná, nezávislá součást návrhu.
- **View** (česky pohled) – Vrstva, která se stará o zobrazení dat (získaných např. z modelu) a uživatelského rozhraní. V rámci webových aplikací je většinou reprezentována nějakým šablonovacím systémem, který se stará o vykreslení webové stránky či její části.
- **Controller** – Nejkomplexnější vrstva, obecně lze říci, že controller je určitou výkonnou jednotkou aplikace, která se stará o provázání funkčnosti všech tří vrstev. Zachytává požadavky uživatele, na jejichž základě volá příslušnou aplikační logiku (model) a dále pak iniciuje překreslení daného pohledu.

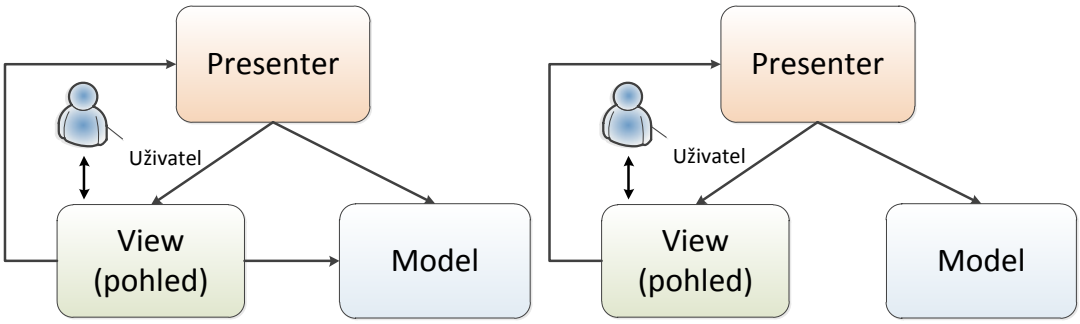


Obrázek 2.1: Základní schéma architektury MVC

Původní návrh MVC byl realizován na přelomu 70. a 80. let minulého století Trygve Reenskaugem [19]. Od jeho uvedení prošel koncept určitou transformací a byly definovány jeho nejrůznější deriváty. Jedním z nich je i Model-View-Presenter (dále jen MVP) [1].

Pohled v MVP oproti MVC zpracovává uživatelský vstup. Např. reaguje na kliknutí myši a volá příslušnou metodu presenteru, deleguje tedy uživatelské akce. Presenter má v sobě aplikační i prezentační logiku. Pracuje s modelem a zajišťuje příslušnou aktualizaci pohledu nebo jej ovlivňuje přímo. Martin Fowler dále rozdělil MVP do dvou kategorií (viz obrázek 2.2) na základě toho, jak velkou zodpovědnost má pohled v celkovém systému [1]:

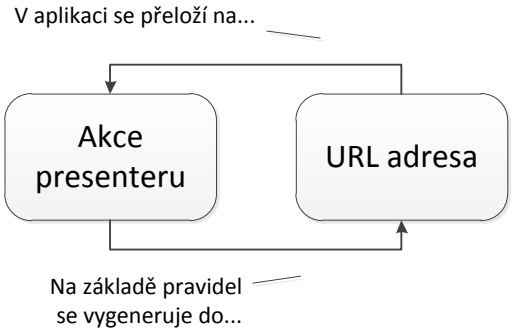
- **Supervising Controller** – Pohled zde má přímou vazbu na model. Tato kategorie se u webových aplikací vyskytuje primárně tam, kde se využívá ve větší míře technologie AJAX a část prezentační logiky se tak přesouvá do jazyka JavaScript na klientské straně. Často se využívá v kombinaci s návrhovým vzorem Observer či technologií data-binding.
- **Passive View** – Pohled zde nemá vazbu na model, tato kategorie odpovídá standardním webovým aplikacím, kde se veškerá prezentační logika odehrává na straně serveru.



Obrázek 2.2: Schéma MVP Supervising Controller (vlevo) a Passive View (vpravo)

Nette částečně těží z derivátu MVP, primárně pak jeho kategorie Passive View. Je však důležité zmínit, že individuálních variant MVC a MVP je mnoho, jejich obecné definice není dobré brát příliš dogmaticky pro každý případ užití, spíše je lze chápat jako určitá nosná doporučení, která se pak přizpůsobí danému návrhu a implementaci.

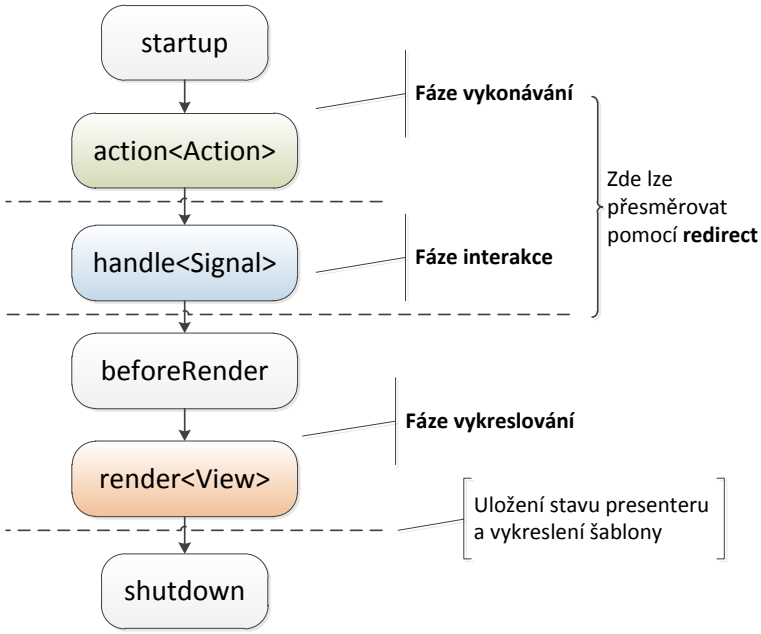
Nyní bude podrobněji rozebráno, jak konkrétně popisované techniky využívá Nette Framework. Modelová vrstva není v Nette plně implementována, framework spíše zajišťuje její odstínění. David Grudl zmiňuje v článku o využití MVC a MVP, že původní idea Nette vycházela z myšlenky, aby odkaz na stránce fungoval pro programátora stejně jako zavolání funkce [19]. Cílem bylo eliminovat nutnost zabývat se manuálně tvarem URL adres a nabídnout tak možnost na základě uvedené akce presenteru (a případných zadaných parametrů) generovat adresu automaticky. Tato adresa poté odpovídá příslušnému vykonání akce na serveru. O překlad akcí presenteru na adresy a zpět se stará systém nazvaný obousměrné routování URL. Tvary URL adres lze pak efektivně měnit nezávisle na struktuře aplikace pouhou změnou daných routovacích pravidel [11]. Schéma této techniky je uvedeno na obrázku 2.3.



Obrázek 2.3: Schéma obousměrného routování v Nette

Presenter v Nette je reprezentován objektem implementujícím jednoduché rozhraní `Nette\Application\IPresenter`. Toto rozhraní obsahuje jedinou metodu `run(Nette\Application\Request $request)`, která má za úkol vracet objekt implementující rozhraní `Nette\Application\IResponse`. V zásadě lze říci, že primárním cílem presenteru v Nette je transformovat objekt HTTP požadavku na příslušnou odpověď. Tou může být např. HTML stránka, obrázek, XML dokument, JSON dokument, soubor na disku či přesměrování [9]. Základní implementací presenteru je objekt třídy

`Nette\Application\UI\Presenter`. Tato třída se skládá z několika důležitých metod, které jsou během toku aplikace vykonány, což je znázorněno na obrázku 2.4.



Obrázek 2.4: Životní cyklus výchozí třídy presenteru v Nette

Nyní budou stručně popsány jednotlivé metody životního cyklu presenteru a jejich význam (v pořadí podle obrázku 2.4) [9]:

- **startup()** – Startovací metoda, kde se mohou např. zkontrolovat uživatelská oprávnění či provézt inicializace některých proměnných.
- **action<Action>()** – Vykonává příslušnou akci presenteru <Action>, avšak ještě nic nevykresluje do šablony. Lze zde provézt dodatečnou přípravu před fází vykreslování, či poslat vlastní odpověď, např. přesměrování pomocí metody `redirect()`. V této metodě je také možné změnit daný <View> pro `render` metody.
- **handle<Signal>()** – Tato metoda vykonává signál <Signal>, signály nebo-li subrequesty jsou blíže popsány v sekci 2.8 pojednávající o komponentách.
- **beforeRender()** – Tato metoda se volá jako první ve fázi vykreslování pro všechny <View>.
- **render<View>()** – Vykreslující metoda pro dané <View>, za normálních okolností daný pohled <View> odpovídá akci <Action>, pokud <View> nebyl změněn v action metodě. Jeden <View> je standardně vykreslován přes jeden soubor šablony stejného názvu.
- **shutdown()** – Metoda se volá při ukončení životního cyklu presenteru.

Tato sekce popisovala základní strukturu aplikace v Nette a vlastnosti presenterů. Presenteru jako komponentě bude ve větší míře věnován prostor v sekci 2.8.

2.2 Šablonovací systém a jazyk Latte

Jazyk PHP byl v původním návrhu zamýšlen jako čistě šablonovací jazyk pro web. V rámci dnešních složitějších webových aplikací je ale jeho nasazení tímto způsobem značně nepraktické, neboť jeho nesprávné použití může být zdrojem mnoha závažných bezpečnostních chyb. Jednou z nejtriviálnějších, ale paradoxně velmi podstatnou je tzv. Cross-Site Scripting (dále jen XSS). Tato chyba může vznikat při neošetření dat vypisovaných v šabloně. Do výsledné webové stránky se tak může dostat nebezpečný JavaScriptový (v této sekci dále jen JS) kód, který se následně spustí v libovolném JS klientovi při každé další návštěvě stránky [28]. Tento kód může např. zcizit přístupové údaje k serverovým sezením uložené v cookies webového prohlížeče. Ty pak útočník použije k realizaci HTTP požadavku, díky kterému může např. následně vystupovat pod účtem cizího uživatele. V případě systémů, kde jsou uchovávána důležitá data, jako např. bankovní systémy či interní informace firem, může mít takový útok nedozírné následky.

S XSS se pojí ještě několik dalších aspektů, které je důležité zmínit. V rámci programování webových aplikací jsou často rozděleny role programátora a kodéra, který se stará o vizuální stránku aplikace. Kodér nemusí mít při své práci se šablonami potřebné znalosti o korektním ošetřování výstupních dat, obzvlášť proto, že v čistém PHP je to značně pracná činnost – je nutné aplikovat různé ošetřovací funkce podle kontextu, kde jsou data vypisována [17]. Např. standardní PHP funkce `htmlSpecialChars()` je použitelná pouze pro ošetřování dat v rámci HTML kódu, v kódu JS je třeba použít jiné ošetření.

Všechny tyto problémy Nette efektivně řeší pomocí vlastního šablonovacího systému a jazyka Latte. Latte obsahuje technologii Context-Aware Escaping [17], která automaticky daný výstup ošetřuje na základě kontextu, kde se data nacházejí. To kodérovi či programátorovi výrazně usnadňuje práci a eliminuje riziko možných bezpečnostních děr. Šablony v Nette navíc neztrácejí na výkonu, neboť veškerý zápis v Latte a jiných šablonovacích jazycích je zkompilován do souboru, který je následně uložen do cache na disk. Šablona je pak znovu kompilována pokaždé, když se změní její zdrojový soubor.

2.2.1 Latte makra a helpery

Makra jsou dynamické části Latte šablony, které řídí výslednou podobu generovaného PHP kódu. Latte makra se zapisují dvojím způsobem [16]:

- Do složených závorek, taková makra mohou být nepárová, např.: `{link Presenter:action}` či párová jako např.: `{if $cond} ... {elseif $cond} ... {else} ... {/if}`
- V rámci (X)HTML tagů pomocí magického prefixu `n`, např.:
  `<a n:href="Presenter:action $parameter">Odkaz</a>`

Makra pokrývají širokou škálu funkcionality, od zápisu podmínek a cyklů, přes podporu rozšiřování a dědičnosti šablon až po generování odkazů a výpis komponent a formulářů [16]. Rozšiřování a dědičnost šablon se realizuje skrze makra `{layout}` a `{block}`. Lze pomocí nich v šablonách jednotlivých pohledů presenterů či v šablonách komponent vytvářet navazující strukturu bez nutnosti opakování kódu. Makrem `{block}` se označí část kódu v dané šabloně. Např. takto:

```
<div id="content">{block content}{/block}</div>
```

V jiné šabloně se pak na cizí šablonu lze odkázat pomocí makra `{layout}` a změnit pouze konkrétní část definovanou daným blokem.

```
{layout '@layout.latte'}
...
{block content}<p>Rozšiřující obsah</p>{/block}
```

Kromě maker jsou další užitečnou pomůckou tzv. helpery. Jedná se o jednoduché funkce, kterými lze dodatečně upravovat výstup. Obsahují např. oříznutí řetězce na určitou délku, změnu malých/velkých písmen či práci s datem a časem [15]. Helpery se zapisují za znakem svislé čárky |, např.: {`$string|upper`}. Latte dále umožňuje programátorům přidávat vlastní nová makra a helpery. To lze učinit jednak pomocí úpravy interní služby `nette.latte` (více o službách v sekci 2.5), případně úpravou objektu šablony v předefinované metodě `templatePrepareFilters($template)` ve třídě vykreslitelných komponent (o komponentách detailně pojednává sekce 2.8).

2.2.2 Vlastní filtry

Kromě Latte lze do šablonovacího systému Nette registrovat i další vlastní filtry, které různými způsoby modifikují finální výstup. Nette nabízí jednoduché rozhraní pro jejich přidávání – opět lze využít metodu `templatePrepareFilters($template)` ve třídě vykreslitelných komponent a od ní zděděných třídách.

2.3 Rozšíření PHP a podpora událostí

Základní předek většiny instancovatelných tříd v rámci Nette je abstraktní třída `Nette\Object`. Tato třída přidává několik velice užitečných, universálních vylepšení či rozšíření [12]. V následujících podsekcích budou některá tato rozšíření blíže představena.

2.3.1 Properties, gettery a settery

V rámci správného objektového návrhu je často žádoucí, aby přístup k privátnímu či chráněnému atributu ve třídě byl spravován skrze příslušné metody, které se někdy nazývají „gettery“ a „settery“. „Getter“ je metoda, která získává obsah atributu (případně jeho modifikaci), „setter“ je naopak metoda, která atributu nastavuje hodnotu, často dohromady s příslušnou validací nové hodnoty. Takový atribut se někdy dohromady s příslušnými přístupovými metodami nazývá „property“ (česky vlastnost).

Potomci třídy `Nette\Object` mohou dále využívat zjednodušenou syntaxi při volání těchto vlastností [12]. Místo zápisu `$object->getAttr()` lze použít jednoduší `$object->attr` a místo zápisu `$object->setAttr($value)` lze napsat `$object->attr = $value`. Jedná se o zjednodušující syntaktickou alternativu, která může zpřehlednit výsledný zdrojový kód.

2.3.2 Reflexe tříd

PHP samotné nabízí možnost zpětné reflexe tříd pomocí třídy `ReflectionClass`. Nette tuto třídu dále rozšiřuje v potomkovi `Nette\Reflection\ClassType` [12]. Objekt této třídy je pak dále zapouzdřen v `Nette\Object`, což umožňuje rychlé a efektivní použití reflexe pro konkrétní objekt. Skrze reflexi lze také číst dokumentační anotace u zdrojového souboru třídy objektu.

2.3.3 Callbacky

Callback je část vykonatelného kódu, který je možné předávat jako argument a pak ve vhodný okamžik vyvolat (např. s příslušnými parametry) [27]. V PHP je callback reprezentovaný polem či správně formátovaným řetězcem. Nette přidává jednoduchou globální funkci `callback()`, která vrací objekt třídy `Nette\Callback` [12]. Ten pak zkontroluje, zda jsou zadané parametry validní a dále nabízí širší funkcionalitu – callback lze například jednoduchým způsobem vyvolat. Příklad zápisu callbacků v Nette:

```
$callback = callback($this, "methodName");
$callback = callback("Class::methodName");
```

2.3.4 Událostmi řízené programování

Třída `Nette\Object` přináší podporu událostmi řízeného programování, neboli anglicky Event-driven programming (dále jen EDP). Toto programovací paradigma se objevuje nativně v některých programovacích jazycích jako např. C#, případně jej lze do jiných jazyků doplnit skrze různé dodatečné knihovny a frameworky. Obecně se toto paradigma vyznačuje dvěma základními prvky [18]:

- **Event** (česky událost) – Jedná se o akci, která může být v kódu vyvolána. Často se tak děje na základě různých externích spouštěčů – např. kliknutí myši či načtení souboru.
- **Handler** – Je určitý vykonavatel, který lze připojit k dané události a po jejím vyvolání se provede s příslušnými parametry vyvolané události.

Toto paradigma se často využívá v desktopových aplikacích u grafického uživatelského rozhraní. Lze takto jednoduše na jednu událost připojit i větší počet vykonavatelů. Např. po zmiňovaném kliknutí myši se vykoná překreslení obrazovky, přehraje se zvukový efekt atd. V rámci webových aplikací přidává `Nette\Object` možnost používat v PHP jednoduchou variantu tohoto paradigmatu [12]. K veřejným atributům tříd (zpravidla začínajícími prefixem `on`) lze pomocí stanoveného zápisu připojovat pole vykonavatelů událostí. To mohou být jednak callbacky popsané v sekci 2.3.3 nebo anonymní funkce. Události lze pak na příslušném místě vyvolat jakoby se jednalo o volání funkce. Příklad zápisu EDP v Nette:

```
// Anonymní funkce je připojena jako vykonavatel k události 'event'
$object->onEvent[] = function($args) {
    // Kód funkce...
};
// Callback je připojen jako vykonavatel k události 'event'
$object->onEvent[] = callback($this, "methodName");
// ...
// Vyvolání události 'event' s parametrem $args
$object->onEvent($args);
```

2.4 Konfigurace a formát NEON

Nette nabízí jednoduchý způsob, jakým lze konfigurovat jeho prostředí. V deklarativním formátu NEON (příklad jeho syntaxe je na obrázku 2.5) se vytvoří konfigurační soubor

či případně několik souborů, které jsou při každé změně znovu zkompilovány do PHP třídy uložené v cache. To je realizováno před samotným spuštěním aplikace. NEON je obecný formát, o jeho sémantice pro konfiguraci Nette aplikací a jednotlivých povolených oddílech pojednává dokumentace [8]. Načítání a kompilaci konfigurace zajišťuje třída `Nette\Config\Configurator`, která dále také realizuje odlišení vývojového a produkčního režimu aplikace. Více o těchto dvou režimech pojednává sekce 2.10.

```
oddil:
    klic: hodnota
    foo: bar
oddil2:
    klic: [1,2,3,4] # Komentář
    klic2:
          ahoj: svete
          tady: bude
          vnitrni:
                - hodnota1
                - hodnota2
```

Obrázek 2.5: Obecný příklad syntaxe konfiguračního formátu NEON

Skrze NEON[3] lze např. konfigurovat následující:

- Nastavení PHP – oddíl `php`
- Nastavení Nette – oddíl `nette`
- Parametry aplikace – oddíl `parameters`
- Služby a továrny (viz sekce 2.5) – oddíly `services` a `factories`
- Rozšíření (viz sekce 2.7) – oddíly podle názvu zaregistrovaného rozšíření

Vygenerovaná třída konfigurace se nazývá systémový kontejner a je potomkem třídy `Nette\DI\Container`. Ta nabízí bohaté rozhraní pro práci se získanou konfigurací. Objekt tohoto typu je pak umístěn jako vlastnost ve třídě presenteru `Nette\Application\UI\Presenter`, která byla podrobně popsána v sekci 2.1. Takto s ním lze v presenterech pohodlně pracovat.

2.5 Objekty jako služby a továrny

V konfiguraci Nette popsané v sekci 2.4 lze vytvářet objekty, kterým se říká služby a továrny. Ty budou nyní definovány:

- **Služby** – Jsou objekty vytvářené dle návrhového vzoru Singleton [3], tedy unikátní instance tříd s konkrétním nastavením, které lze opakovaně získávat a využívat.

[3] Více ukázek formátu NEON lze nalézt na stránce `http://ne-on.org/`

- **Továrny** – Na rozdíl od služeb továrny vytváří vždy novou instanci dané třídy (volně vychází z návrhového vzoru Factory Method [3]), dále mohou také obsahovat parametry.

Výhodou zápisu služeb a továren v rámci konfigurace v NEON je to, že je lze jednoduchým způsobem předávat jako parametry dalším službám. Příklad této techniky v NEON:

```
service1:
   class: Some\Class(%parameter%)
   setup:
      - setValue(value)
service2: Some\Another\Class(@service1)
#Služba service1 (odkazovaná pomocí znaku @)
byla předána jako parameter službě service2
```

Důležitým pojmem, který je dále nutné zmínit, je tzv. auto-wiring [8], což je schopnost Nette předat službě závislosti automaticky na základě detekce požadovaných parametrů – podle požadovaného typu u daného parametru či podle anotací `@return`.

Kromě NEON konfigurace lze služby a továrny vytvářet i skrze nová rozšíření popsaná v sekci 2.7.

2.6 Dependency Injection

Dependency Injection (dále jen DI) je pojem, který při správném použití může tvořit určité filozofické jádro dobrého návrhu aplikace. Jedná se o jednoduchý princip, kdy samostatná součást kódu (v objektovém návrhu v zásadě třída) získává své závislosti (atributy třídy, se kterými bude pracovat) zvnějšku, tedy z kódu, který volá danou třídu. V rámci objektového programování lze dále ještě rozlišit dva způsoby použití DI [3]:

- **Constructor injection** – Závislosti jsou předávány jako parametry v konstruktoru třídy.
- **Setter injection** – Závislosti jsou předávány v nastavovací metodě třídy (tzv. „setter“, viz podsekce 2.3.1).

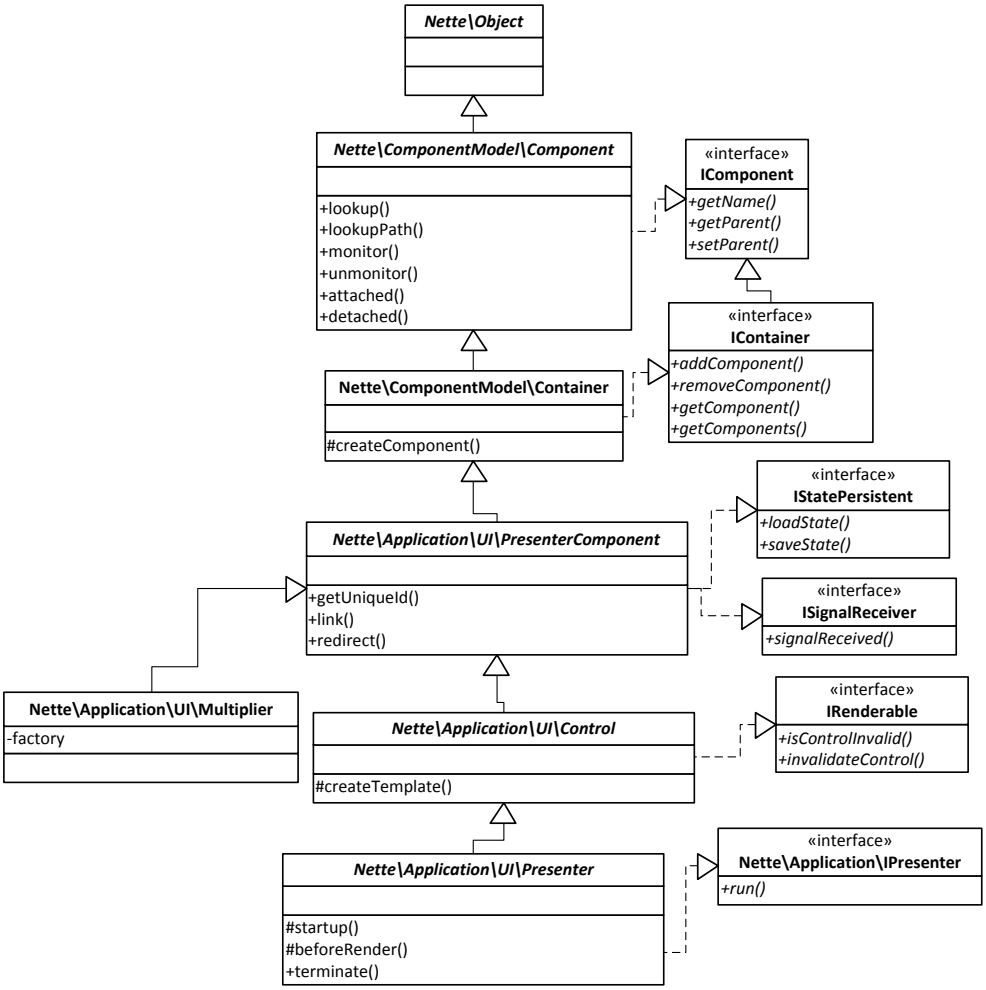
Nette staví na principech DI [7] a dále usnadňuje možnosti jeho využití skrze pomůcky jako je třída systémového kontejneru a jeho služby a továrny popsané v sekci 2.5. S tímto přichází ještě další pojem zvaný Inversion of Control (neboli česky převrácené řízení), který deklaruje, že třída se již nestará o to, jak potřebné závislosti získá – to je řízeno mimo třídu.

2.7 Tvorba rozšíření

Nette nabízí jednoduchý způsob, jak do frameworku přidávat nová rozšíření. Každé rozšíření je reprezentováno jedinou třídou, která dědí od `Nette\Config\CompilerExtension` [13]. Tato třída poskytuje metody, ve kterých je implementováno načtení a zpracování dodatečné konfigurace. Každé rozšíření je pak v konfiguračním souboru NEON reprezentováno jedním novým oddílem, který má stejný název jako poskytnutý název zaregistrovaného rozšíření. V rámci Nette samotného jsou některé jeho části taktéž realizovány formou rozšíření, např. interní nastavení Nette, nastavení PHP či deklarace konstant. Nová rozšíření většinou slouží k přidávání širší funkcionality v podobě nových služeb a továren (viz sekce 2.5).

2.8 Komponenty a jejich vlastnosti

Tato sekce tvoří teoretické jádro této práce. Komponentu v Nette představuje objekt, který je při správném návrhu znovupoužitelný v rámci aplikací. Díky tomu je možné komponenty sdílet i s jinými vývojáři či je opakovaně využívat ve více projektech [10]. Ve většině případů bývají komponenty vykreslitelné jako prvek webové stránky, např. formulář, tabulka, menu nebo anketa. Takové komponenty jsou reprezentovány objekty potomků třídy `Nette\Application\UI\Control`. Tato třída v sobě obsahuje továrnu na šablonu, skrze kterou se komponenta vykresluje. Model komponent v Nette dále sahá do větší hloubky, obsahující vlastnosti jako je možnost kolekce vytvářených komponent do stromu rodičů a potomků. Hierarchie jednotlivých tříd a rozhraní implementujících celkový systém komponent je znázorněna v diagramu na obrázku 2.6.



Obrázek 2.6: Diagram tříd a rozhraní komponentového modelu v Nette

Na vrcholu je rozhraní `IComponent`, které obsahuje metody pro získání jména komponenty a správu jejího rodiče. Toto rozhraní je standardně implementováno ve třídě `Component`, kde je položen základ komponentového stromu rodičů a potomků. Klíčová je pro to metoda `lookup($type)`. Ta vyhledává v hierarchii směrem nahoru objekt daného typu. Například zavolání metody `$component->lookup("Nette\Application\UI\Presenter")`

vrátí daný presenter, pokud je k němu (i třeba přes několik úrovní) komponenta připojena. Metoda `lookupPath($type)` poté vrací cestu, což je řetězec vzniklý spojením unikátních názvů všech komponent mezi aktuální a hledanou komponentou [10]. Důležité jsou i metody `attached($obj)` a `detached($obj)`, které jsou automaticky volány při připojení/odpojení komponenty k rodiči daného typu. Aby tak bylo učiněno, je nutné nejprve daný typ monitorovat pomocí `monitor($type)`.

Od rozhraní `IComponent` dědí rozhraní `IContainer`, které obsahuje metody pro přidávání, odebírání, získání a iteraci nad komponentami. Od tohoto rozhraní dědí všechny rodičovské komponenty. Celý strom komponent je pak tvořen větvemi v podobě objektů `IContainer` a listů `IComponent` [10]. Jedna větev pak samozřejmě nesmí obsahovat dva listy se stejným unikátním názvem. Příklad tohoto zanořování komponent je na obrázku 2.7. Standardní implementací `IContainer` je třída `Container`, která skrze metodu `createComponent($name)` deleguje řízení na tovární metody `createComponent<Name>()` v poděděných třídách. Tímto se realizuje tzv. „lazy“ vytváření potomků, tedy komponenta-potomek je vytvořena a přidána do stromu komponent, až když je jí potřeba, např. při jejím vykreslení v šabloně. Příklad jednoduché vytvářecí metody:

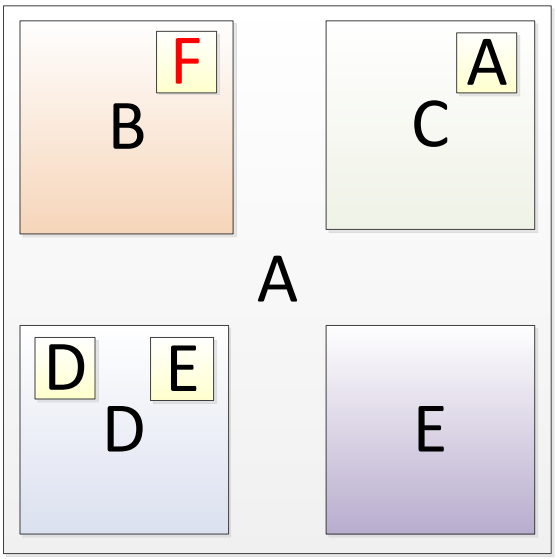
```
// Vytváří se komponenta s názvem 'test' třídy TestComponent
protected function createComponentTest()
{
   return new TestComponent;
}
```

Drobný problém, který může vyvstat při použití těchto továrních metod je, pokud je žádoucí volat dvě komponenty napříč ve svých továrnách, když tyto komponenty ještě nejsou vytvořeny. Např. v tovární metodě komponenty `test` je volána komponenta `test2` a v tovární metodě komponenty `test2` je zase volána komponenta `test`. Toto může nastat třeba tehdy, když programátor chce propojit obě komponenty navzájem pomocí mechanismu událostí a vykonavatelů popsaném v podsekci 2.3.4. Jelikož obě komponenty ještě nejsou vytvořeny, jsou pak střídavě donekonečna volány jejich tovární metody a dojde k zacyklení. Jedním z možných řešení této situace v Nette je komponenty takto propojit např. již v metodě akce presenteru (viz sekce 2.1), tím je však porušena jejich „lazy“ vlastnost.

Třída `PresenterComponent` představuje komponentu připojenou k presenteru jako nejvyššímu rodiči. Tyto komponenty si presenter uchovává po dobu svého životního cyklu [10]. Unikátní identifikátor komponenty získaný na základě její pozice ve stromu lze získat metodou `getUniqueId()`. Např. červeně zvýrazněná komponenta `F` na obrázku 2.7 by měla identifikátor `A-B-F`. Třída komponent presenteru dále implementuje dvě rozhraní. `ISignalReceiver`, což zajišťuje možnost přijímání signálů (viz podsekce 2.8.1) a `IStatePersistent`, které slouží ke schopnosti ukládat a načítat stav komponenty. To Nette standardně dělá skrze tzv. persistentní parametry popsané v podsekci 2.8.2.

Třída `Multiplier` představuje nástroj pro dynamickou tvorbu potomků [4]. Do této třídy je uložena továrna na komponentu v podobě anonymní funkce či callbacku (viz podsekce 2.3.3). Při zavolání tovární metody `createComponent($name)` u této třídy je pak řízení delegováno na vyvolání příslušné uložené továrny s předaným identifikátorem nového potomka. Takto lze tuto třídu využít jako rodiče pro neomezené množství dynamicky vytvářených potomků.

Další třída v hierarchii na popisovaném obrázku je již samotná `Control` pro vykreslitelné komponenty. Ta ještě implementuje rozhraní `IRenderable`, což umožňuje snazší práci



Obrázek 2.7: Zanoření komponent do stromu – rodičové a potomci

s technologií AJAX, jak popisuje podsekce 2.8.4. Nejspodnější složkou diagramu je pak již známá třída `Presenter` popsaná v sekci 2.1.

2.8.1 Signály

Signál (neboli také subrequest) je komunikace s Nette aplikací čistě pod prahem jednoho pohledu, tedy komunikace, které se dějí, aniž by se daný pohled změnil. Pohled je možné změnit pouze v rámci akce presenteru, o čemž bylo pojednáno v sekci 2.1. Signál způsobí znovu-načtení stránky, stejně jako klasický požadavek (výjimku tvoří AJAXové subrequesty, viz podsekce 2.8.4).

V komponentách třídy `Nette\Application\UI\PresenterComponent` je signál zachycen v metodách tvaru `handle<Signal>()` s vloženými parametry a předán příslušné komponentě, které je určen [10]. Jelikož třída presenteru je taktéž komponentou, jak demonstroval obrázek 2.6, může být příjemcem signálu i ona.

Vygenerování odkazu na signál pro danou komponentu lze realizovat pomocí metody `PresenterComponent::link($destination)`. Tuto metodu většinou není nutné volat přímo, v šablonovacím jazyce Latte popsaném v sekci 2.2 jsou pro toto určena příslušná makra `{link}` a `n:href`. Formát zadaného signálu je `{signalReceiver}-{signal}`. `{signalReceiver}` je jednoznačný identifikátor komponenty ve stromu směrem od komponenty, kde je signál volán. Část `{signalReceiver}` je nepovinná a v případě jejího neuvedení posílá komponenta signál sama sobě, což je nejčastější případ. Část URL (příslušné GET parametry) reprezentující signál může vypadat např. takto:

```
<Base URL address>?component-value=10&do=component-signal
```

Po vykonání signálu se většinou využívá metody `PresenterComponent::redirect($destination)` k přesměrování na URL základní akce bez signálu.

2.8.2 Persistentní parametry

Určité atributy třídy komponenty je možné přenášet napříč více požadavky jako GET parametry. K tomu slouží magická anotace `@persistent`, kterou lze označit příslušné veřejné atributy třídy. Parametry jsou takto přenášeny až do chvíle, kdy uživatel stránku s touto komponentou opustí [10]. Parametr se také nepřenáší, pokud má svou výchozí hodnotu.

2.8.3 Vykreslení komponenty

Vykreslení komponenty se v šabloně standardně realizuje pomocí Latte makra `{control}` [16]. Tomuto makru se předává jednoznačný identifikátor komponenty v rámci stromu směrem od aktuální komponenty, kde je vykreslování prováděno. Do makra lze dále předat značku, určitý specifikující řetězec pro vykreslení oddělený dvojtečkou. V rámci této práce bude nazván vykreslovací mód. Dále lze také předat specifikující parametry. Ty budou v této práci nazvány jako vykreslovací parametry. Příklad volání vykreslení komponenty skrze popsané makro:

```
{* Vykreslení komponenty "compId" *}
{control compId}
{* Vykreslení komponenty "compId2" s vykreslovacím módem "small"
a parametrem "foo" s hodnotou "bar" *}
{control compId2:small foo => bar}
```

Zápis výše pak zkompilovaný do PHP vypadá takto:

```
$control->getComponent("compId")->render();
$control->getComponent("compId2")->renderSmall(array("foo" => "bar"));
```

Důležité je zmínit, že třída `Nette\Application\UI\Control` neimplementuje příslušné `render` metody, nezajišťuje správu daného vykreslovacího módu a parametrů ani samotné vykreslení do šablony (např. nastavením souboru se šablonou). Třída zajišťuje pouze vytvoření objektu šablony a předání základních proměnných. Ostatní uvedené musí zajistit programátor sám ve svých poděděných třídách komponent.

2.8.4 AJAX v komponentách

AJAX (Asynchronous JavaScript and XML) je moderní webová technologie, díky které se může webová stránka překreslit na základě nových dat získaných ze serveru, aniž by došlo k jejímu celkovému znovu-načtení. To může zajistit rychlejší a pohodlnější práci s aplikací. V Nette lze AJAX efektivně využívat pomocí tzv. snippetů [5]. V šabloně jsou části stránky označeny pomocí Latte makra `{snippet}` a při provedení signálu lze oznámit jejich invalidaci, tj. požadavek na překreslení těchto částí. To se děje pomocí metody `Nette\Application\UI\Control::invalidateControl($snippet)`. Tyto části stránky jsou pak v příslušné struktuře poslány jako tzv. HTML „výstřižky“, které na klientské straně zpracuje obslužný JavaScriptový kód. Navzdory svému názvu není v Nette u AJAXu využíváno XML jako interní formát pro odeslání odpovědi. Místo toho se používá více odlehčený formát JSON (podrobněji popsaný v sekci 3.4).

2.9 Session

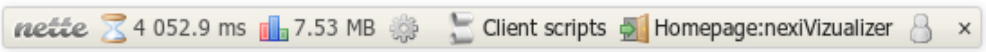
V rámci webových aplikací se často objevuje nutnost uchovávání stavu aplikace na serveru. Je potřeba ukládat např. přihlášení uživatele či obsah nákupního košíku. Čisté PHP realizuje přístup k těmto proměnným skrze tzv. session neboli sezení, ke kterému se přistupuje přes globální proměnnou `$_SESSION`. Její použití je jednoduché (používá se stejně jako klasické pole v PHP), nicméně při složitějších aplikacích může docházet ke zmatkům při kolizi názvů různých proměnných. Nette tuto situaci zjednodušuje pomocí objektové obálky třídy `Nette\Http\Session` [14]. Ta umožňuje rozdělovat prostor sezení na jednotlivé sekce v rámci jmenných prostorů a dále např. zjednodušuje nastavování expirace (doby platnosti) těchto jednotlivých sekcí.

2.10 Ladící nástroje

Poslední část analýzy Nette Frameworku je věnována jeho ladícím nástrojům. Ladění je esenciální součást vývoje v kterémkoliv jazyce či prostředí. Při programování v PHP je relativně snadné udělat mnoho těžko odhalitelných chyb, proto je důležité mít k dispozici vhodné nástroje, které rychlé nalezení chyb usnadní. Aplikace v Nette se může spustit ve dvou základních režimech – vývojovém a produkčním. Zvolení režimu je možné provézt buďto manuálně nebo jej může framework detekovat automaticky na základě IP adresy serveru [6]. Aplikace má pak odlišné chování pro oba režimy, ve vývojovém je např. možné používat příslušné ladící nástroje, v produkčním se místo toho chyby logují do souborů a uživateli je zobrazena jednoduchá stránka s chybou. Třídy s ladícími pomůckami se v Nette nacházejí ve jmenném prostoru `Nette\Diagnostics`. Je zde k dispozici efektivnější výpis proměnných do HTML kódu či ladícího okénka, logování chyb do souborů a mnoho dalších nástrojů.

2.10.1 Ladící okno

Ladící okno je plovoucí lišta implementovaná v jazyce JavaScript (viz obrázek 2.8) a zobrazovaná ve vývojovém režimu. Jedná se o velice užitečný nástroj, který zobrazuje nejrůznější informace o běhu aplikace či pomáhá v její navigaci a správě. Ladící okno a jeho jednotlivá podokna lze libovolně posouvat po stránce s tím, že jejich pozice se ukládají do cookies webového prohlížeče. Pozice se tady neztratí ani po znovu-načtení stránky. Každý z jednotlivých panelů či podoken je reprezentován třídou implementující rozhraní `Nette\Diagnostics\IBarPanel`. Z toho plyne, že lze do ladícího okna jednoduše přidávat nová vlastní podokna/panely.



Obrázek 2.8: Ladící okno pro vývojový režim

2.10.2 Modrá chybová obrazovka

Vizualizace chyb a výjimek se vývojovém režimu děje skrze tzv. modrou chybovou obrazovku[4] (ukázka na obrázku 2.9). Jedná se o rozsáhlou informační stránku o dané chybě, která skrze jednotlivé sekce informuje o všech aspektech dané chybové situace. Je zde přehledně vidět název dané chyby (či výjimky), soubor a konkrétní řádek, kde k chybě došlo, call-stack, výpis HTTP požadavku, výpis nastavení serveru atd. Stejně jako do ladícího okna, i do modré chybové obrazovky lze přidávat dodatečné vlastní sekce.

Obrázek 2.9: Příklad modré chybové obrazovky

[4]Termín „modrá obrazovka“ je slangové označení užívané pro chybové obrazovky v rámci softwarových aplikací. Vznikl podle známých chybových oznámení v operačních systémech Microsoft Windows. Jelikož oficiální dokumentace Nette tento termín používá, byl takto zachován i v rámci této práce.

Kapitola 3

XML a JSON

XML bylo definováno konsorciem W3C jako formát pro přenos dokumentů a dat. XML je zkratka pro eXtensible Markup Language, tj. česky rozšiřitelný značkovací jazyk [22]. Návrh XML vychází ze staršího, obecnějšího formátu SGML (Standard Generalized Markup Language), ze kterého vzešel i formát HTML (Hyper-Text Markup Language) pro popis webových stránek.

Primárním aspektem XML, který je důležité uvést, je jeho universálnost. Zatímco zmiňovaný formát HTML má jasně definovaný účel a sémantiku, využití XML závisí čistě na svobodné režii vývojáře. Takto jej lze v softwarové branži nasadit pro široké množství činností. O jeho oblibě svědčí i vysoký počet formátů z něj vycházejících – v rámci technologií užívaných na webu lze jmenovat např. syndikační formáty rodiny RSS a Atom nebo formát SVG pro popis vektorové grafiky.

Syntaxe XML je definována z mnoha nejrůznějších pravidel. Základem je struktura elementů a jejich možných atributů [26]. Její příklad je demonstrován na obrázku 3.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <element1 atribut="hodnota">Text</element1>
  <element2 />
  <element3 atribut2="hodnota2">Jiný text</element3>
  <element4>
     Text
     <element5 atribut="hodnota"
    atribut2="hodnota2">Text</element5>
  </element4>
  <element6 atribut3="Jiná hodnota" />
</root>
```

Obrázek 3.1: Příklad syntaxe formátu XML

Povinný je vždy kořenový element (na obrázku je to `root`), který se skládá z dalších elementů, které mohou obsahovat zase další elementy, holý text či kombinaci obou zmíněných. Jednotlivé elementy mohou mít i atributy s příslušnými hodnotami (na obrázku

např. `element1`). Elementy mohou být i tzv. uzavřené (tj. že nemají žádný obsah, ale mohou mít atributy). Příkladem takovýchto elementů jsou na obrázku `element2` a `element6`. Jednotlivé elementy musí být dále korektně zanořeny, stejným způsobem jako např. závorky v zápise matematických výrazů. Např. následující zápis v XML je chybný:

```
<element1><element2>Obsah elementu</element1></element2>
```

Jména elementů v XML mohou obsahovat písmena, číslice a některé další tisknutelné znaky, nesmí však obsahovat mezery. Jméno dále také nesmí začínat číslicí nebo kterýmkoliv znakem interpunkce. V rámci dobrých návyků je dále dobré se v názvu vyhýbat znakům pomlčky, tečky a dvojtečky. Obzvlášť dvojtečka je důležitá, neboť má v XML speciální význam pro prefixy užívané u jmenných prostorů [24].

3.1 XML prefixy a jmenné prostory

Při používání XML často vzniká konflikt v rámci názvů jednotlivých elementů. Např. když se kombinuje vícero XML dokumentů z různých aplikací mohou se objevit dva elementy se stejným názvem, ale jiným sémantickým významem [25]. Pro vyřešení této situace je zavedena možnost užití prefixů v názvech elementů. Prefix a název se odděluje pomocí znaku dvojtečky. Výsledný zápis takovýchto elementů pak může vypadat například takto:

```
<a:element atribut="hodnota">Obsah elementu</a:element>
<a:element2 />
```

Aby mohl být prefix korektně použit jako jmenný prostor, je nutné jej definovat skrze atribut `xmlns` v rámci kořenového elementu, kde je prefix využíván [25].

3.2 XML schémata

Jak již bylo uvedeno na začátku této kapitoly, formát XML je universální, vývojář si tedy musí sám navrhnout sémantický význam jednotlivých elementů a atributů v rámci užívaných XML dokumentů. Ke kontrole validity dokumentů na příslušnou navrženou strukturu lze využít již existující technologie jako jsou DTD (Document Type Definition) nebo XML schémata [22]. XML schéma je zvláštním XML dokumentem obsahujícím elementy ve jmenném prostoru `xs`, který popisuje prvky jiného XML dokumentu. Schéma nabízí rozsáhlé možnosti definice povolených elementů, atributů, jejich struktury a pořadí. Dále také umožňuje zavádět restrikce a pravidla pro názvy elementů, obsah atributů a skrze dotazovací jazyk XPath kontrolovat náročnější vazby elementů [22]. Schémata tak zavádějí jistou formální definici datových formátů založených na XML. Výhoda takové formalizované definice je, že je jednoznačná a znemožňuje různé interpretace na rozdíl od toho, kdyby definice byla zapsána v přirozeném jazyce [21].

3.3 XML v PHP

V jazyce PHP lze s XML dokumenty pracovat více způsoby. Jednou z nejvíce flexibilních možností je využití technologie DOM (Document Object Model). Skrze tuto technologii je příslušný dokument načten do paměti a jeho stromová struktura je uložena v objektech reprezentujících jednotlivé části dokumentu. Pomocí těchto objektů pak lze jednoduše

zjišťovat obsah částí dokumentu, případně je modifikovat [21]. DOM nabízí široké API i pro využívání dalších XML technologií, jako např. validaci pomocí zadaných XML schémat popsaných v sekci 3.2.

3.4 Formát JSON

JSON (JavaScript Object Notation) je odlehčený formát užívaný pro výměnu dat. Je jednoduše čitelný a zapisovatelný pro vývojáře a snadno zpracovatelný a generovatelný v rámci stroje. Je založen na podmnožině jazyka JavaScript, avšak jinak se jedná o formát zcela nezávislý na jazyce, což vysoce zvyšuje jeho užitečnost a použitelnost. Zakládá se na dvou universálních a široce podporovaných strukturách [20]:

- Kolekce párů klíč:hodnota. Tento koncept bývá v různých jazycích realizován v mnoha podobách. Např. jako objekt, záznam, struktura či asociativní pole.
- Tříděný seznam hodnot. Ten se v jazycích obvykle objevuje jako pole, vektor či posloupnost.

Příklad formátu JSON je na obrázku 3.2:

```
{
    "klic": {
          "klic2": "hodnota",
          "klic3": 25,
          "klic4": [1,2,3,4,"abc"]
    },
    "klic2": true,
    "klic3": {
          "klic4" : [],
          "klic5": "Text"
    }
}
```

Obrázek 3.2: Příklad syntaxe formátu JSON

Velká výhoda JSONu narozdíl od XML je menší velikost přenášených dat, jelikož v XML často podstatnou část dokumentu tvoří samotná meta-struktura elementů a atributů [23]. JSON je tak v rámci moderních webových aplikací využíván pro mnoho účelů. Primárně lze zmínit jeho nasazení pro zasílání odpovědí ze serveru v rámci technologie AJAX. Velké množství JavaScriptových knihoven (např. těch, které vychází ze známé knihovny jQuery) také používá rozsáhlé JSON struktury jako vstupní parametry pro své objekty. Častý postup v rámci webových aplikací je pak provézt na serveru (např. v jazyce PHP) serializaci dané interní datové struktury do formátu JSON, která se pak při výpisu dat v šabloně vloží do příslušného JavaScriptového kódu.

Kapitola 4

Návrh knihovny Nexi

Návrh knihovny Nexi (dále jen Nexi) vychází z cílů práce definovaných v sekci 1.1 a využívá informační základ poskytovaný ve dvou předchozích teoretických kapitolách. Knihovna je realizována za využití systému rozšíření Nette Frameworku, který byl popsán v sekci 2.7. To umožňuje její rychlou a efektivní integraci do projektů, kde se s ní bude pracovat.

Určitým nedostatkem při práci s komponentami v Nette je nutnost častého psaní opakujícího se kódu. Tento jev se objevuje např. u signálů popsaných v podsekci 2.8.1. Ty komponenta využívá ke komunikaci pod prahem jednoho pohledu. V kódu třídy komponenty se objevují ve formě jednotlivých obslužných metod signálů. Ty bývají většinou velice podobné, např. obsahují vyvolání události (viz podsekce 2.3.4), na kterou lze později navázat vykonávací metody, zavolání přesměrování či požadavek na zneplatnění části stránky u AJAXových požadavků (viz podsekce 2.8.4).

V komponentách se také někdy opakují vlastnosti tříd (viz podsekce 2.3.1) složené z privátních atributů a přístupových metod. Ty bývají někdy velmi triviální, neboť pouze získávají daný atribut či provádějí jednoduchou validaci na správný typ nové hodnoty.

Další překážkou, kterou musí programátor často řešit je vykreslování komponent. V podsekci 2.8.3 bylo diskutováno vykreslování komponenty pomocí makra a nutnost zachytávat předané vykreslovací módy a parametry manuální implementací vykreslovacích metod v poděděných třídách komponent. V těchto metodách je také nutné nastavovat šablonu komponenty formou souboru, ve kterém se šablona nachází.

V šablonách komponent je také často nutné řešit náročnější vykreslování parametrů, což může někdy způsobit příliš složitě zapisované šablony pouze za použití nativních maker v Latte. Vykreslovaná část komponenty také může záviset na externím zdroji vkládaném do HTML stránky. Toto se pak i u větších projektů někdy řeší neefektivním vkládáním všech skriptů a stylů do hlavičky stránky pro všechny vykreslované případy.

Motivace tvorby této práce je primárně zjednodušit uvedené aspekty a zefektivnit tak vývoj aplikací pod Nette snazším použitím komponent a jejich šablon.

Na základě uvedených skutečností lze nyní shrnout klíčové vlastnosti, které bude nová knihovna Nexi obsahovat:

- Vytvoření nové základní třídy pro vykreslitelné komponenty
- Přidání nových Latte maker pro snazší práci v šablonách komponent
- Zavedení nového XML filtru jako alternativní zápis pro vykreslení komponenty
- Podpora správy načítání klientských skriptů a stylů

- Možnost generování jednoduchých tříd komponent ze souborů XML
- Konfigurace Nexi ve formátu NEON a rychlejší vytváření instancí komponent

V následujících sekcích jsou tyto jednotlivé části do detailů popsány.

4.1 Nová základní třída vykreslitelných komponent

Teorie komponentového modelu v Nette byla do hloubky popsána v sekci 2.8. Vykreslování komponent v šablonovacím jazyce Latte pak probíhá pomocí makra `{control}`, které bylo blíže představeno v podsekci 2.8.3. V této podsekci bylo také zmíněno, že základní třída pro vykreslitelné komponenty v Nette neobstarává vykreslení samotné ani zachycení a zpracování vykreslovacího módu a parametrů předávaných při zavolání diskutovaného makra. Ty slouží k jednoduché možnosti vykreslování dané komponenty do více různých podob. Obstarání těchto tří aspektů tvoří filozofický základ nové třídy pro komponenty v navrhované knihovně. Třída tak obsahuje několik nových vlastností (viz podsekce 2.3.1) pro vykreslovací mód, vykreslovací parametry, soubor se šablonou a adresář s danou šablonou. Mód a parametry lze tímto způsobem nastavit do výchozích hodnot při vytváření instance komponenty ještě před fází vykreslení. Při samotném vykreslení např. pomocí makra `{control}` pak lze změnit výchozí vykreslovací mód či doplnit vykreslovací parametry hodnotami předanými v makru. Výsledné vykreslovací parametry a mód se pak předávají jako proměnné do šablony. Mód je řetězec obsahující alfa-numerické znaky a parametry jsou uloženy ve speciálním objektu, který implementuje nové rozhraní pro práci s parametry a jejich další manipulaci. To zahrnuje jejich filtraci, validaci či serializaci do formátu JSON (viz sekce 3.4) pro vkládání do JavaScriptových kódů.

Nová třída dále podporuje tzv. nastavování klientských závislostí komponenty (této části se více věnuje sekce 4.4). Jednotlivé instance komponent mohou deklarovat svou závislost na externích klientských skriptech či kaskádových stylech odkazovaných v HTML stránce. Tím se zvýší modularita a jednodušší použitelnost vykreslitelné komponenty.

Třída umožní i udržení vnitřního stavu komponenty na serveru. V podsekci 2.8.2 byla popsána výchozí možnost ukládání a načítání stavu komponent v Nette přes tzv. persistentní parametry. Základní komponentová třída v Nexi přidává nové rozhraní pro možnosti práce se stavem komponenty. Dané rozhraní pak lze implementovat různými způsoby – např. za využití Nette session a jeho jmenných prostorů (viz sekce 2.9).

Poslední důležitou vlastností nové třídy je rozsáhlejší podpora pro propojení komponent skrze mechanismus událostí. Paradigma událostmi řízeného programování v Nette bylo popsáno v podsekci 2.3.4. Implementace tohoto paradigmatu je zahrnuto v základní třídě pro instancovatelné objekty v Nette, lze jej tedy bez problému užívat i ve třídách komponent. V sekci 2.8, kde se probíraly jednotlivé třídy realizující komponenty v Nette, byla zmíněna i třída pro rodičovské komponenty a nastíněn problém, který vyvstává, jsou-li dvě prozatím neexistující komponenty potomků propojeny pomocí událostí ve svých továrních metodách. Knihovna nabídne řešení tohoto problému skrze ukládání požadovaných propojení a jejich realizaci, až když jsou obě komponenty bezpečně vytvořeny. Tak nikdy nedojde k cyklickému volání továrních metod.

Nexi přidává i novou třídu pro presentery. Ta nabízí pouze jednoduchou funkcionalitu v podobě jednoduššího přístupu k některým novým službám knihovny.

Obě nové třídy dále rozšiřují možnosti vytváření potomků komponent, o čemž je pojednáno v sekci 4.6.

4.2 Nová Latte makra

Latte a jeho součásti bylo podrobně představeno v sekci 2.2. Při doplnění nové třídy komponent s podporou vykreslovacích módů a parametrů (viz sekce 4.1) je užitečné, aby práci s nimi v šabloně usnadnila nová makra. Knihovna zahrnuje nové makro pro odchycení daného vykreslovacího módu v šabloně a dále také makra sloužící jako nástavba nad metodami rozhraní pro přístup k vykreslovacím parametrům. Takto lze jednoduše komponentu vykreslovat v mnoha různých podobách. Další obsažené makro umožňuje vykreslovat předané údaje (např. vykreslovací parametry či jejich část) do podoby řádkového zápisu kaskádových stylů CSS.

Kromě maker speciálně pro novou třídu komponent jsou obsažena i obecná makra, která lze po zaregistrování knihovny využívat v jakékoliv Latte šabloně. Zde jsou zahrnuta makra pro práci s vykreslením načtených klientských skriptů a stylů (více v sekci 4.4) a dále také makro pro generování atributu `id` v rámci HTML kódu. Tento generovaný identifikátor pro prvky HTML vychází z jednoznačného identifikátoru ve stromu presenteru dané komponenty, kde je makro užito. Pokud je pak každá komponenta vykreslena do stránky pouze jednou, je tímto zajištěno, že se žádný identifikátor v HTML nebude opakovat.

4.3 Vykreslování komponent pomocí XML elementů

Knihovna přidává nový filtr do šablonovacího systému (viz podsekce 2.2.2), který podporuje vykreslení komponenty skrze zápis pomocí XML elementu s příslušným prefixem (viz sekce 3.1). Toto slouží jako alternativa k zápisu pomocí makra `{control}`. Aby bylo možno komponenty takto vykreslovat, je nutné příslušný element zaregistrovat v příslušné části konfigurace ve formátu NEON (více v sekci 4.6).

4.4 Podpora načítání klientských skriptů a stylů

Jak již bylo zmíněno, komponenty vykreslené jako části webové stránky mohou často záviset na externím souboru vloženém do HTML stránky. Tím může být kaskádový styl CSS obsahující design komponenty či nějaká knihovna v jazyce JavaScript, jejíž funkcionalita se ve vykresleném kódu komponenty dále využívá.

Tyto externí soubory bývají často připojeny k HTML stránce na jiném místě než v části, kterou vykresluje komponenta, např. v hlavičce stránky či až na konci těla. Druhotným aspektem, který je také často nutno řešit, je fakt, že některé takto připojené kódy na sobě mohou záviset. To bývá většinou případ již zmíněných JavaScriptových knihoven. Např. knihovna `jQueryUI` závisí na knihovně `jQuery` a příslušném kaskádovém stylu, které tak musí být do stránky připojeny jako první.

V rámci popisu Latte byla zmíněna možnost rozšiřování a dědičnosti šablon (viz podsekce 2.2.1). Použití této funkcionality na uvedenou situaci je teoreticky možné, nicméně vyvstává několik dalších problémů. Za prvé dědičnost šablon nelze příliš efektivně aplikovat napříč šablonami pohledů presenterů a šablonami jednotlivých komponent. Za druhé neřeší se zde zmíněná závislost jednotlivých klientských skriptů a stylů.

Knihovna Nexi řeší tento problém pomocí deklarace klientských skriptů a stylů v rámci své konfigurace v NEON. Do konfigurace jsou zadány i informace o tom, jakým způsobem na sobě příslušné skripty závisí. Příslušná instance komponenty pak může v rámci nové

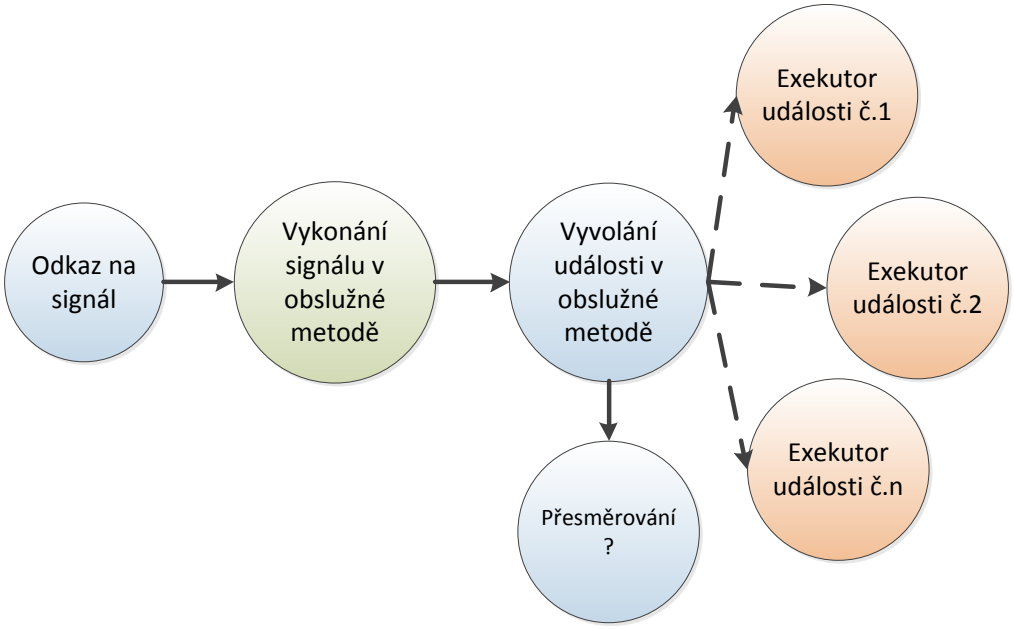
třídy v Nexi uvést svou závislost na daném klientském skriptu či stylu (viz sekce 4.1), čímž se načtou i veškeré závislosti.

Knihovna dále realizuje příslušné vykreslení načtených skriptů a stylů v šabloně. K tomu jsou v Nexi připravena nová speciální Latte makra nabízející širokou škálu možností pro dané vykreslení. Pro informaci o tom, kolik a jaké klientské skripty byly načteny, přidává knihovna panel do ladícího okna popsaného v podsekci 2.10.1.

4.5 Generování tříd komponent z formátu XML

Jak již bylo nastíněno v úvodu této kapitoly, jednotlivé třídy vykreslitelných komponent se často skládají z opakujících se částí kódů. V některých případech se toto dá příslušně řešit objektovou dědičností, někdy však může být nápomocné jednoduché třídy generovat, což právě zajišťuje tato funkcionalita Nexi. Jako formát souborů, ze kterých se budou třídy generovat, byl zvolen značkovací jazyk XML popsaný v kapitole 3. K validaci těchto XML dokumentů, zda splňují navrženou sémantickou strukturu pak byla vybrána XML schémata představená v sekci 3.2.

Daný XML dokument v sobě popisuje jednotlivé třídy komponent a jejich součásti. Jeden dokument může takto popisovat neomezené množství komponent i jejich strukturu z hlediska umístění ve jmenných prostorech PHP. XML schémata příslušně kontrolují unikátnost uvedených názvů tříd v rámci jednoho prostoru. V dokumentu lze také nastavit dědičnost mezi jednotlivými třídami i třídami mimo generovací proces a implementaci zadaných rozhraní.



Obrázek 4.1: Demonstrace propojení signálu a události v rámci generovaného kódu třídy komponenty

Jednotlivé elementární části kódu třídy komponenty, které lze pak ze zápisu v XML generovat jsou:

- Vlastnosti či tzv. properties, které se skládají z atributů a případných přístupových metod
- Obslužné metody signálů spolu s parametry a možným voláním přesměrování
- Události
- Jednoduché veřejné metody, které lze použít jako vykonavatele událostí (v rámci knihovny nazývané také exekutory či exekuční metody), taktéž s příslušnými parametry.
- Propojení obslužných metod signálů s vyvoláním události. To je velice užitečné pro zajištění efektivní komunikace komponenty se svým okolím, aniž by ztratila vlastnost znovu-použitelnosti. V dané obslužné metodě se vyvolává událost, na kterou pak může programátor připojit neomezené množství vykonavatelů. Může se jednat např. o exekuční metody jiných komponent. Demonstrace této techniky je na obrázku 4.1.
- Volání invalidace, tj. zneplatnění a požadavek o překreslení částí stránky při použití AJAXu (viz podsekce 2.8.4). Lze ji generovat do volání v exekučních metodách nebo v obslužných metodách signálů.
- Konstanty třídy

Generování zahrnuje i kontrolu, zda byl daný zdrojový XML soubor změněn od posledního zpracování. V případě, že nebyl, generování se znovu neprovádí. Třídy také obsahují speciální dokumentační anotaci, která označuje, že byla třída vygenerována. Pokud ji programátor odstraní, nedochází již k opětovnému přepsání třídy.

Smysl této anotace lze popsat na následujícím příkladě. Programátor si ze zdrojového XML souboru vygeneruje dvě třídy komponent, např. pojmenované jako `ComponentA` a `ComponentB`. Třída `ComponentA` je abstraktní a dědí od nové základní třídy vykreslitelných komponent v Nexi (viz sekce 4.1). Dále obsahuje bohatou generovanou funkcionalitu v podobě příslušných obslužných metod signálů a propojení s voláním událostí, exekučních metod, vlastností atd. Třída `ComponentB` není abstraktní a dědí od třídy `ComponentA`. Po vygenerování obou tříd programátor zruší danou dokumentační anotaci u třídy `ComponentB` a tato třída se již nebude dále přepisovat. Poté může pohodlně manuálně editovat třídu `ComponentB` a úpravou XML dokumentů generováním doplňovat základní aspekty v rodičovské třídě `ComponentA`.

4.6 Konfigurace Nexi a komponenty jako továrny v NEON

Celá knihovna lze příslušně konfigurovat v rámci formátu NEON popsaném v sekci 2.4. Jak bylo v této sekci uvedeno, zaregistrované rozšíření je v konfiguraci reprezentováno oddílem o stejném názvu. Knihovna je zaregistrovaná pod svým názvem `nexi`. Pod tímto konfiguračním oddílem jsou pak v rámci rozšíření definovány další pododdíly, každý pro konfiguraci příslušného aspektu knihovny. Je zde část pro nastavení klientských skriptů a stylů, část pro generování tříd, pro nastavení XML filtru a pro dynamické vytváření potomků komponent. Poslední jmenované bude ještě nyní popsáno.

Vytváření komponent stylem „lazy“ v Nette bylo popsáno v sekci 2.8. Zde byla zmíněna třída pro rodičovské komponenty, která při požadavku na neexistujícího potomka deleguje jeho vytvoření na příslušnou tovární metodu. Poděděné třídy komponent pak příslušné

tovární metody implementují. Takto je potomek vytvořen, až když je ho skutečně potřeba, např. při vykreslení nebo při realizaci propojení skrze události.

Nexi zavádí možnost definovat v konfiguraci komponenty podobným způsobem jako továrny systémového kontejneru popsané v sekci 2.5. Vytvářecí metoda je pak předefinována v nových základních třídách pro komponenty a presentery (popsané v sekci 4.1) tak, že při dotazu na nevytvořeného potomka a v případě neexistence tovární metody předá řízení příslušné továrně v systémovém kontejneru. To může být výhodné z hlediska toho, že není nutné v jednotlivých třídách komponent implementovat příslušné tovární metody, stačí je pouze zapsat do konfigurace v NEON. Zápis takovýchto komponent umožňuje dále využívat i třídu multiplikátoru, která byla taktéž zmíněna v sekci 2.8. Ta příslušným způsobem zapouzdří vytvořenou továrnu, že je pak možné na jejím základě vytvářet neomezené množství potomků s předanými identifikátory. Nexi umožňuje i v konkrétních rodičovských třídách implementovat určitou editační metodu továrny, která komponentu vytvořenou přes systémový kontejner ještě příslušným způsobem modifikuje. Takto lze různě kombinovat přístup pro vytváření potomků skrze zápis v konfiguraci a v rámci samotného kódu.

Kapitola 5

Implementace knihovny Nexi

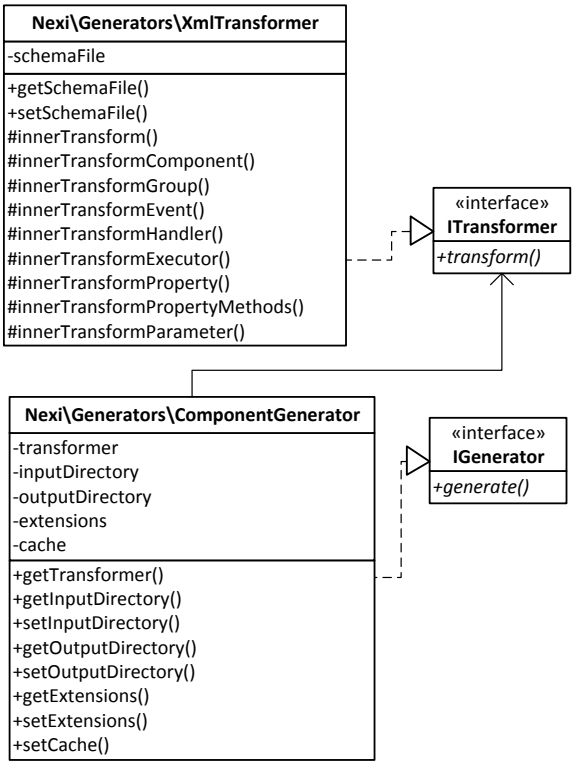
Na základě návrhu knihovny bude nyní popsána její konkrétní implementace a základy použití. Stejně jako v kapitole 2 i zde jsou v textu místy uváděna plně kvalifikovaná jména (výjimečně pouze zkrácená jména, kde informace o jmenném prostoru není potřeba) tříd, rozhraní a jejich metod v rámci API nové knihovny Nexi či Nette Frameworku. Tam je vždy vycházeno z verze použité pro tvorbu této práce – Nette 2.0.10 stable uvolněná 6.3. 2013 pro PHP 5.3 nebo PHP 5.4. Verze PHP 5.2 je již více zastaralá a není knihovnou podporována. Primárním důvodem pro to je absence jmenných prostorů v této verzi. Zdrojové soubory knihovny byly všechny vyvíjeny v IDE prostředí NetBeans verze 7.3 pro PHP.

Knihovna je strukturovaná stejným způsobem jako samotný Nette Framework. Kořenový adresář je `Nexi` podle názvu knihovny, hierarchie podadresářů pak přímo odráží hierarchii jmenných prostorů jednotlivých tříd. Každá třída je umístěna v jednom souboru stejného názvu. Neplatí to pouze pro třídy výjimek, ty jsou umístěny vždy dohromady pro jeden jmenný prostor v příslušném souboru `exceptions.php`. Základní třída instancovatelných objektů je zde `Nexi\Object`. Jedná se pouze o prázdnou, formální třídu, která dědí od `Nette\Object` popsanou v sekci 2.3. Knihovna dále kráčí ve stopách Nette i ve smyslu dodržování dobrých návyků u svého objektového návrhu, primárně skrze využití programovací techniky Dependency Injection popsané v sekci 2.6.

Jak bylo dále popsáno v sekci 2.7, rozšíření pro Nette jsou reprezentována potomky třídy `Nette\Config\CompilerExtension`, která obsahuje metody pro zkontrolování a zpracování příslušné NEON konfigurace do parametrů a služeb systémového kontejneru (viz sekce 2.5). Pro knihovnu Nexi tyto metody implementuje poděděná třída `Nexi\Config\Extension`. Rozšíření lze pak do aplikace jednoduše zaregistrovat pomocí statické metody `Nexi\Config\Extension::register(Nette\Config\Configurator $configurator)`. To se zpravidla děje v zaváděcím souboru aplikace, kde se předá vytvářená třída konfigurátoru. Metoda `register` rozšíření registruje pod názvem `nexi`, čili stejným jako jméno samotné knihovny s malým písmenem na začátku. Jak již bylo řečeno, stejný název jako zaregistrované rozšíření má pak také vždy jeho hlavní oddíl konfigurace v NEON. Pod hlavním oddílem následuje několik zavedených pododdílů druhé úrovně. Konkrétně jsou to `client`, `generator`, `xml` a `components`. Blíže budou představeny v sekci 5.6. Sekce, které nyní následují, se zaměřují na implementaci konkrétních integrálních součástí uvedených v návrhu, v kapitole 4.

5.1 Implementace generátoru tříd komponent

V sekci 4.5 byl popsán návrh funkcionality pro generování jednoduchých tříd komponent z formátu XML. V Nexi toto zajišťují dvě základní rozhraní. Prvním je `Nexi\Generators\ITransformer`, které skrze svou metodu `transform($inputFile)` zadává obecnou transformaci z deklarativního zápisu v souboru do interní reprezentace PHP třídy. Druhým je `Nexi\Generators\IGenerator`, které pak obsahuje generování tříd samotných. Výchozí implementací rozhraní transformátoru je třída `Nexi\Generators\XmlTransformer`, která transformuje XML dokumenty do interních reprezentací jednoduchých tříd Nette komponent. Třída `Nexi\Generators\ComponentGenerator` pak implementuje generátor a na základě výsledků získaných z transformátoru generuje nové třídy komponent na disk. Schéma uvedených tříd a rozhraní je na obrázku 5.1.



Obrázek 5.1: Diagram tříd a rozhraní zajišťujících generování

Třída generátoru obsahuje dále několik vlastností pro svá nastavení. Zmínit lze cestu ke vstupnímu adresáři se zdrojovými XML soubory `inputDirectory`, cestu k výstupnímu zapisovatelnému adresáři pro generované třídy `outputDirectory` či povolené přípony pro zdrojové XML soubory `extensions`. Třída ke své práci může využívat vyrovnávací paměť `cache`, kam si ukládá časy posledního zpracování jednotlivých zdrojových souborů a následně je porovnává s časy poslední modifikace. Takto je zajištěno, že se generuje pouze ze souborů, které byly od posledního zpracování změněny. Vygenerované soubory komponent obsahují v dokumentaci třídy anotaci `@generated`. Při smazání této anotace se daný soubor již nebude přepisovat při příštím procesu generování. Možné použití tohoto principu bylo nastíněno v závěru sekce 4.5.

Třída transformátoru má jako svou vlastnost cestu k souboru s validačním schématem

XML (viz sekce 3.2). Při nenastavení této vlastnosti je použito výchozí, doporučené validační schéma, které je součástí knihovny Nexi. Nachází se v souboru `components.xsd` v adresáři pro jmenný prostor `Nexi\Generators`. Schéma určuje povolenou strukturu zdrojových XML souborů. Popis tohoto schématu, povolených elementů, atributů a jejich významu lze nalézt v příslušné referenční uživatelské příručce na přiloženém CD nosiči. O jeho obsahu pojednává příloha A.

Na obrázku 5.2 je ukázka zdrojového XML souboru podle sémantiky definované výchozím schématem. Uvedený soubor je převzat z demonstrační aplikace, viz sekce 6.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<components author="John Doe">

   <component name="BaseExampleComponent" modifier="abstract"> Example component for sandbox
    demonstration
      <handler name="click" invoke="true" asArray="true">Handler of click event
         <parameter name="borderColor" type="string">New border color</parameter>
         <parameter name="backgroundColor" type="string">New background color</parameter>
         <invalidate />
      </handler>
      <executor name="executeValue">
         <parameter name="input" type="array">Input of values</parameter>
         <invalidate />
      </executor>
      <property name="colors" access="protected" type="array"
    value="[black,white,yellow,blue,green,red,purple,pink,cyan,brown]"> Accepted colors
      </property>
   </component>

   <component name="ExampleComponent" extends="\BaseExampleComponent">
   </component>

   <component name="SliderComponent"> Example Component for Jquery UI slider/bar
   </component>

   <group namespace="Inner">
      <component name="VizualizerComponent">
      </component>
   </group>

</components>
```

Obrázek 5.2: Příklad zdrojového XML souboru pro transformaci

Při zpracovávání XML souboru se špatnou syntaxí či špatnou sémantickou strukturou oproti zvolenému schématu jsou vyhozeny výjimky dědící od abstraktní třídy `Nexi\Generators\XmlException`. Výjimky této třídy jsou následně zpracovávány ve třídě `Nexi\Diagnostics\BlueScreen`, která přidává novou sekci do „modré“ chybové informační stránky Nette popsané v podsekci 2.10.2. Tato přidaná sekce podrobně zobrazuje chyby ve vložených XML souborech.

Samotnou transformaci zadaných XML dokumentů ve třídě transformátoru pak zajišťuje několik chráněných metod, kdy je vždy jedna metoda vyhrazena pro transformaci konkrétní generovatelné části kódu. Jejich soupis byl uveden v návrhu v sekci 4.5. Základní transformační metoda je `innerTransform(DOMDocument $xml)`, další lze vidět na obrázku 5.1. Jednotlivé chráněné metody lze pak snadno předefinovat v případných poděděných třídách XML transformátoru. Pro načtení, zpracování a validaci XML souborů se interně využívá technologie DOM pro PHP popsaná v sekci 3.3.

5.2 Implementace načítání klientských skriptů

Načítání klientských skriptů a správu jejich závislostí navrženou v sekci 4.4 zajišťují v Nexi čtyři klíčová rozhraní a jejich výchozí implementace. Rozhraní `Nexi\ClientManagement\IClientItem` reprezentuje klientský prvek a metody pro zjišťování jeho typu, cesty a atributů. Toto rozhraní implementuje třída `ClientItem`. Rozhraní `Nexi\ClientManagement\IClientHolder` je kolektor pro jednotlivé prvky. Obsahuje metody pro vkládání nových dat pod určitým klíčem či anonymně bez klíče, zpětné získání na základě klíče, iteraci či holé vytvoření nového prvku typu `IClientItem` na základě parametrů. Třída `ClientHolder` implementuje toto rozhraní a přidává vlastnosti pro výchozí cesty k prvkům přes webový server a fyzickou adresářovou strukturu na disku. Rozhraní `Nexi\ClientManagement\IClientRenderer` obsahuje metody pro vykreslování klientských prvků. Jeho výchozí implementace `Nexi\ClientManagement\ClientRenderer` vykresluje prvky podle jejich typu do podoby příslušných HTML elementů. Např. tímto klasickým způsobem:

```
<link type="text/css" rel="stylesheet"
href="/webserver/Projects/BP/Nexi/sandbox/www/css/style.css" />
<script type="text/javascript"
src="/webserver/Projects/BP/Nexi/sandbox/www/js/jqueryNetteNexi.js"></script>
```

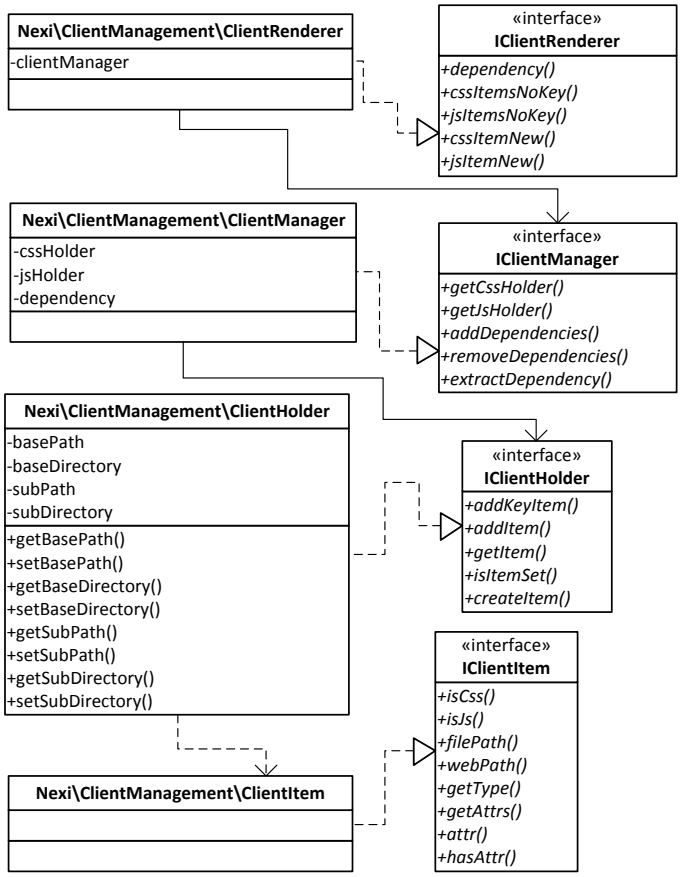
Třída pro vykreslování potřebuje ke své činnosti objekt posledního popisovaného rozhraní `Nexi\ClientManagement\IClientManager`. Toto rozhraní je určitý správce, který zajišťuje provázání klientských skriptů a stylů se stromem jejich závislostí. Rozhraní v sobě obsahuje metody na načtení, odebrání a iteraci klientských prvků z uložených závislostí. Dále také dvě metody pro přímý přístup ke dvěma interním objektům typu `IHolder`, z nichž jeden udržuje klientské prvky typu CSS (kaskádové styly) a druhý typu JS (JavaScript kódy). Rozhraní je implementováno ve třídě `Nexi\ClientManagement\ClientManager`, která ke své vnitřní činnosti dále využívá třídu `Nexi\Collections\TreeCollection` pro ukládání již zmiňovaných závislostí do stromové hierarchie. Schéma popsaných tříd a rozhraní ze jmenného prostoru pro klientskou správu je znázorněno na obrázku 5.3.

Pro lepší přehlednost všech načtených klientských skriptů, stylů a jejich atributů je ve vývojovém režimu přidáno nové podokno do ladícího okna Nette popsaného v podsekci 2.10.1. Ukázka nového podokna je na obrázku 5.4.

5.3 Implementace nových základních tříd

Jak bylo popsáno v sekci návrhu 4.1, knihovna Nexi obsahuje nové základní třídy pro vykreslitelné komponenty a presentery.

Hlavní důležitost má implementovaná třída komponent `Nexi\Application\Control`, která dědí od základní třídy pro vykreslitelné komponenty v Nette `Nette\Application\UI\Control`. Nová třída zajišťuje nastavení a zpracování vykreslovacího módu, parametrů a souboru a adresáře se šablonou. V případě nenastavení souboru šablony či jejího adresáře je automaticky zvolena šablona ve stejném adresáři či se stejným názvem jako soubor s třídou komponenty. Výchozí vykreslovací mód lze nastavit skrze vlastnost `renderMode` a zabránit jeho možnému přepsání při předávání údajů u vykreslování přes vlastnost `renderModeLock`. Výchozí vykreslovací parametry jsou přístupné pomocí vlastnosti `renderParameters`. Parametry jsou nejprve interně ukládány v poli,



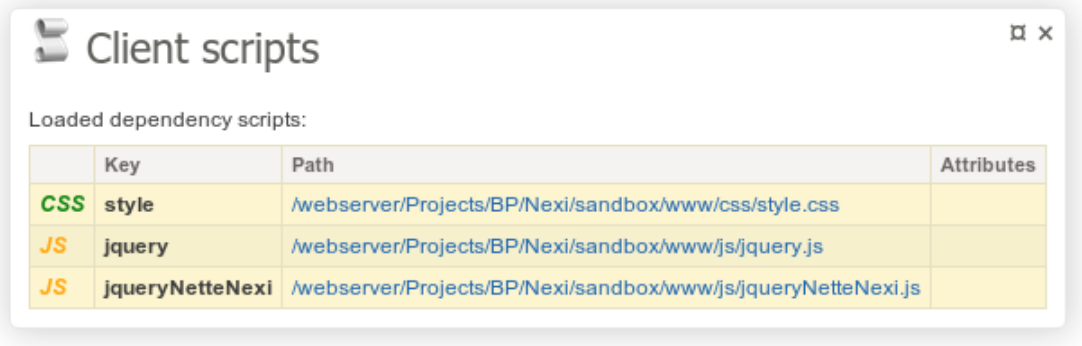
Obrázek 5.3: Diagram tříd a rozhraní zajišťujících klientskou správu

tedy v podobě klíče a hodnoty. Klíčem pak může být jakákoliv skalární hodnota (většinou řetězec). Při vykreslování a předávání nových vykreslovacích parametrů jsou výchozí a předávané hodnoty spojeny a případný konflikt stejných klíčů je řešený pomocí vlastnosti `renderParametersJoin`. Ta udává, zda má prioritu daný výchozí vykreslovací parametr či předávaný. Parametry jsou následně umístěny do objektu implementujícího rozhraní `Nexi\Collections\IParametersHolder`. Výchozí implementací tohoto rozhraní použitou v nové třídě komponent je třída `Nexi\Collections\ArrayCollection`.

Zachycení předávaného vykreslovacího módu a parametrů se děje pomocí předefinovaní magické metody v PHP `__call($name, $arguments)`. Ta zpracovává neexistující volané metody s předponou `render` a dále volá chráněnou finální metodu `performRendering($renderMode, array $renderParameters)`, která realizuje nastavení zvoleného souboru se šablonou a předávání výsledného vykreslovacího módu a parametrů do šablony. Ty jsou pak v šabloně k dispozici jako proměnné `$mode` pro mód a `$parameters` nebo `$rp` pro parametry.

Klíčová je dále chráněná metoda `editRendering($renderMode, $renderParameters, $template)`, kterou může programátor předefinovat v poděděných třídách a zde pak příslušné, diskutované prvky ještě manuálně modifikovat (např. na základě aktuálního stavu komponenty).

Třída dále obsahuje chráněné tovární metody



Obrázek 5.4: Podokno s informacemi o načtených klientských skriptech

`createClientManager(Nette\Application\UI\Presenter $presenter)` pro vytváření objektu klientského manažeru popsaného v sekci 5.2 a `createState(Nette\Application\UI\Presenter $presenter)` pro vytváření objektu stavu komponenty. Ten je v Nexi realizován novým rozhraním `Nexi\State\IState`. Výchozí implementací užitou i ve třídě komponent je `Nexi\State\SessionState`, která, jak již z názvu vyplývá, užívá k ukládání stavu session, konkrétně pak objekty pracující se session v Nette (viz sekce 2.9).

Nastavení klientských závislostí komponenty lze učinit např. pomocí metody `client($args)`, které jsou dány klíče klientských skriptů a stylů, na kterých komponenta závisí. Tyto vložené hodnoty jsou poté vevnitř předány metodám klientského manažeru, který následně načte potřebné závislosti.

Navrhovaný mechanismus pro propojování komponent, který řeší problém cyklického volání továrních metod komponent je řešen skrze metodu `connectWith()`. Takto lze uložit propojení komponenty s jinou komponentou (případně i se sebou). Propojení se pak realizují, až při připojení komponenty ke stromu presenteru v metodě `attached($obj)` (viz sekce 2.8), kdy je již zdrojová komponenta bezpečně vytvořena a nebude se volat její tovární metoda. Pro samotné připojení vykonavatelů k události využívá Nexi alternativní mechanismus, než ten standardní v Nette uvedený v podsekci 2.3.4. Mechanismus je implementovaný skrze statickou metodu `Nexi\Connector::connect()`.

Základní třída pro presentery v Nexi je `Nexi\Application\Presenter`, která dědí od `Nette\Application\UI\Presenter`. Tato třída má relativně jednoduchou funkcionalitu – poskytuje přístup k novým službám rozšíření (viz sekce 5.6).

Obě nové třídy registrují v metodě `templatePrepareFilters()` nový XML filtr popsaný v sekci 5.5, třída komponent navíc registruje nová Latte makra pro komponenty popsaná v sekci 5.4.

Obě dvě třídy dále předefinovávají metodu `createComponent($name)` pro vytváření potomků způsobem, jaký byl popsán v návrhu v sekci 4.6. Vytváření komponenty je zde delegováno na příslušnou továrnu komponenty v systémovém kontejneru. Jako možné editační metody zmíněné v návrhu lze v poděděných třídách vytvářet metody ve tvaru `editComponent<Name>($comp, $name)`, které vrací editovanou instanci komponenty potomka.

5.4 Implementace nových Latte maker

Nexi přidává dvě kategorie nových maker do Latte. První kategorií jsou makra přístupná pouze pro šablony komponent dědících od třídy `Nexi\Application\Control`. Zde patří např. makra `{renderMode}` a `{renderModeRegexp}`, která pracují s vykreslovacím módem. Jejich použití může být následující:

```
{renderMode default,small}
   Tato část šablony bude vykreslena pouze
   ve vykreslovacích módech default a small
{/renderMode}
{renderModeRegexp test[0-9]+}
   Tato část šablony bude vykreslena pouze
   ve vykreslovacích módech splňujících zadaný regulární výraz
{/renderModeRegexp}
```

Další makra pro komponenty realizují nástavbu nad metodami rozhraní `Nexi\Collections\IParametersHolder` pro práci s parametry. Patří zde např. makra `{get}`, `{add}`, `{select}` a další. Makro `{css}` dále realizuje vykreslení zvolených parametrů do podoby řádkového zápisu kaskádového stylu.

Druhou kategorií maker jsou makra obecná. Ty přidává třída `Nexi\Templating\Macros`. Tato makra jsou přístupná v kterékoliv Latte šabloně po zaregistrování rozšíření knihovny Nexi. V těchto makrech je obsažena nástavba nad rozhraním `Nexi\ClientManagement\IClientRenderer` pro vykreslování klientských prvků. Např. makro `{dependency}` vykreslí všechny načtené klientské prvky ze stromu závislostí (viz sekce 5.2). Dalším zde přidaným makrem je makro `{id}`, které usnadňuje generování HTML `id` atributů v šablonách komponent a akcí presenterů. Vygenerovaný identifikátor pak kromě vloženého názvu obsahuje i identifikátor konkrétní komponenty ze stromu presenteru. Toto makro lze použít i pomocí n-prefixového zápisu (viz podsekce 2.2.1), použití může být následující:

```
<div n:id="paragraph">...</div>
```

V komponentě s unikátním identifikátorem `compId` pak daný atribut `id` ve výsledné HTML stránce vypadá např. takto:

```
<div id="paragraph-compId-cmp">...</div>
```

Kompletní, podrobný seznam všech nových Nexi maker, jejich použití a parametrů lze nalézt v příslušné referenční uživatelské příručce – viz příloha A.

5.5 Implementace XML filtru

XML filtr umožňující zápis vykreslení komponent pomocí XML elementu zajišťuje třída `Nexi\Templating\XmlFilter`. Tato třída implementuje rozhraní `Nexi\Templating\IFilter` obsahující jedinou metodu `process($input)`. Při aplikaci filtru je tato metoda vykonána a XML elementy s příslušným zaregistrovaným prefixem (jako výchozí prefix je zvoleno `nx`) jsou nahrazeny za volání vykreslovacího makra `{control}`. Filtr je tedy vždy použit ještě před aplikací filtru Latte. Vykreslovací parametry lze zadávat pomocí atributů elementu.

Jednotlivé elementy jdou ve třídě `XmlFilter` i tzv. předpřipravit uložením nastavení jednotlivých elementů. Lze zde uložit identifikátor komponenty, kterou bude daný element vykreslovat, její vykreslovací mód a parametry.

5.6 Popis nových služeb a konfigurace Nexi

Knihovna Nexi přidává několik nových služeb a továren systémového kontejneru využívaných napříč funkcionalitou popisovanou v předchozích sekcích. Jsou implementovány z již uvedených tříd. Tyto služby/továrny lze využívat i manuálně, ke snazšímu získávání některých z nich jsou ve třídě `Nexi\Application\Presenter` implementovány přístupové metody.

Jako příklad služeb lze uvézt např. `nexi->clientManager` vytvořená z popsané třídy klientského manažeru či služba `nexi->generator` ze třídy generátoru komponent. Jako příklad továrny je zde `nexi->state` pro tvorbu stavu komponenty. Všechny služby a továrny z rozšíření jsou pak umístěny do zvláštního přístupového pod-kontejneru s názvem `nexi`.

Konfigurace knihovny v NEON se skládá z několika hlavních pododdílů. Pododdíl `client` slouží ke konfiguraci nastavení klientských skriptů a jejich závislostí (viz sekce 5.2). Pododdíl `generator` k nastavení generátoru komponent (viz sekce 5.1). V pododdílu `xml` se nastavují vlastnosti XML filtru (viz sekce 5.5) a v pododdílu `components` lze vytvářet komponenty jako továrny. Podrobné informace o možnostech nastavení knihovny v NEON jsou uvedeny v příslušné referenční příručce – viz příloha A.

Kapitola 6

Testování knihovny

Implementovaná knihovna byla otestována v rámci demonstrační aplikace. Ta je zahrnuta do kostry základního projektu v Nexi, který rozšiřuje podobnou kostru (tzv. „sandbox“), která je součástí Nette Frameworku (viz příloha A). Pro vývoj knihovny byl využit webový server Apache verze 2.2, demonstrační aplikace byla pak testována na webových prohlížečích Firefox a Google Chrome v systému Kubuntu Linux.

Následující sekce 6.1 popisuje strukturu demonstrační aplikace. Pro vizualizaci vytvářených komponent lze využít implementovaný vizualizační nástroj, který je taktéž součástí aplikace. Ukázka nástroje je na obrázku 6.1.

6.1 Demonstrační aplikace

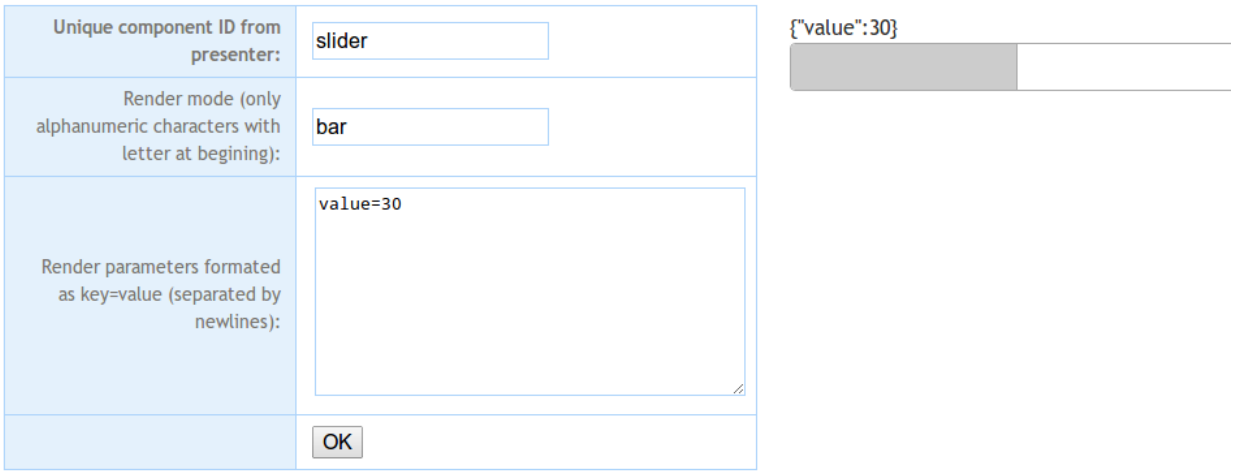
Demonstrační aplikace rozšiřuje základní projekt Nette o několik nových akcí presenteru `Homepage`:

- **nexiDefault** – V této akci je prezentována demonstrace jednoduchého propojení komponent třídy `ExampleComponent`. Třída dědí od abstraktní třídy `BaseExampleComponent`. Obě třídy jsou generovány skrze mechanismus generování tříd komponent a vhodně upraveny. V šabloně jsou vykresleny dvě komponenty zmiňované třídy a tyto instance jsou definovány pomocí zápisu jako továren v NEON. Propojení pomocí událostí je zde realizováno pouze pomocí metody `Connector::connect()` v akci presenteru. Další ukázkou zde je použití makra `{id}` pro pomoc u vykreslení dvou JavaScriptových prvků a jejich propojení na úrovni klienta.
- **nexiDemo** – V této akci je implementována demonstrační aplikace knihovny. V prvé části se jedná o komponentu třídy `Demo\DirectoryComponent`, která dědí od abstraktní třídy `Demo\BaseDirectoryComponent`. Tato komponenta realizuje výpis souborů v daném adresáři a dále pomocí signálu `changeDirectory` i navigaci mezi jednotlivými adresáři v jejich struktuře na disku. Požadovaný nový adresář je zkontrolován a uložen do stavu komponenty. Komponenta obsahuje dále signál `changeFile` volající příslušnou vygenerovanou událost stejného jména. Skrze ni je k této komponentě připojena druhá komponenta třídy `Demo\PictureComponent` dědící od abstraktní třídy `Demo\BasePictureComponent`. Ta přijme ve své exekuční metodě události cestu k souboru a jedná-li se o obrázek s příponou `.jpg`, `.gif` nebo `.png` uložené v prostoru webového serveru, tak jej zobrazí pomocí HTML. Všechny třídy komponent jsou opět

generovány a dědící třídy příslušným způsobem upraveny. Komponenty jsou vykresleny za použití XML filtru a příslušného zaregistrovaného elementu.

- **nexiVizualizer** – Tato akce obsahuje vizualizační nástroj pro komponenty. Vizualizátor samotný je také realizován jako generovaná a upravená komponenta `Inner\VizualizerComponent` a dále se v šabloně skládá ze dvou částí. Za prvé je zde formulář (vykreslovaný vlevo) pro zadávání údajů o požadované komponentě, kterou chce vývojář vykreslit. Jedná se o již známý identifikátor komponenty ze stromu presenteru, vykreslovací mód a vykreslovací parametry. Druhou částí je pak samotná vykreslená komponenta na základě vložených údajů. S komponentou lze libovolně posouvat po stránce či ji zvětšovat a zmenšovat.

XML vzory komponent jsou uloženy v souborech `base.xml` a `demo.xml` ve složce `app/designs`. Cílový adresář pro vygenerované třídy je pak `app/components`.



Obrázek 6.1: Vizualizační nástroj pro komponenty

Kapitola 7

Závěr

Cílem této práce bylo vytvořit pomocnou nástavbu nad Nette Frameworkem v podobě nové knihovny, která usnadní použití a správu komponent a jejich šablon. Části Nette Frameworku, které mají se stanovenou problematikou co do činění byly podrobně analyzovány, vyzdvihly se jejich klíčové vlastnosti, stejně tak i aspekty, které framework přímo neřeší a nechává v rukou samotných programátorů. Dále byl také proveden jednoduchý rozbor formátu XML a jemu příbuzných technologií.

Získané teoretické poznatky byly zužitkovány v rámci návrhu, ve kterém byl stanovený cíl dále rozčleněn do několika vzájemně se doplňujících částí. Výsledkem je komplexní knihovna nazvaná Nexi, která usnadňuje tvorbu jednoduchých tříd komponent pomocí mechanismu generování ze souborů XML, zjednodušuje práci se šablonami přidáním nových maker a podporou tzv. vykreslovacích módů a parametrů. Jako další zjednodušení práce nabízí správu klientských stylů a skriptů, řešení problému jejich závislostí a snadné spojení tohoto konceptu s vytvářenými komponentami. Komponenty lze dále také více efektivněji propojovat pomocí mechanismu událostí v Nette, vykreslovat v šablonách skrze XML elementy a definovat jejich vytváření v konfiguračních souborech NEON. Knihovna obecně ctí filozofii Nette ve smyslu modularity jednotlivých částí a snadné integrace do projektů.

Možná budoucí rozšíření knihovny by se mohla zabývat rozsáhlejšími možnostmi při generování tříd z XML či ještě komfortnější implementací systému pro práci s klientskými styly a skripty. Knihovna by se dále mohla zaměřit i na zjednodušení práce s formuláři v Nette, obzvláště pak zajistit jejich snadnější propojení s klientským kódem v šabloně. Širší podpora by se měla realizovat i pro použití technologie AJAX. Obecně lze říci, že zaměření knihovny by mělo dále spět k zajištění pohodlnější návaznosti mezi klientskou a serverovou částí komponent.

Literatura

[1] Bernard, B. *Prezentační vzory z rodiny MVC* [online]. 11.5. 2009 [cit. 29. dubna 2013]. Dostupné na: <http://www.zdrojak.cz/clanky/prezentacni-vzory-zrodiny-mvc>.

[2] Bernard, B. *Úvod do architektury MVC* [online]. 7.5.2009 [cit. 29. dubna 2013]. Dostupné na: <http://www.zdrojak.cz/clanky/uvod-do-architektury-mvc>.

[3] Böhmer, M. *Návrhové vzory v PHP*. [b.m.]: Computer Press, 2012. ISBN 978-80-251-3338-5.

[4] Dobeš, V. *Multiplier* [online]. [cit. 30. dubna 2013]. Dostupné na: <http://pla.nette.org/cs/multiplier>.

[5] Dokumentace Nette Frameworku. *AJAX & snippety* [online]. [cit. 30. dubna 2013]. Dostupné na: <http://doc.nette.org/cs/ajax>.

[6] Dokumentace Nette Frameworku. *Debugování a zpracování chyb* [online]. [cit. 30. dubna 2013]. Dostupné na: <http://doc.nette.org/cs/debugging>.

[7] Dokumentace Nette Frameworku. *Dependency Injection* [online]. [cit. 30. dubna 2013]. Dostupné na: <http://doc.nette.org/cs/dependency-injection>.

[8] Dokumentace Nette Frameworku. *Konfigurace* [online]. [cit. 30. dubna 2013]. Dostupné na: <http://doc.nette.org/cs/configuring>.

[9] Dokumentace Nette Frameworku. *MVC aplikace & presentery* [online]. [cit. 30. dubna 2013]. Dostupné na: <http://doc.nette.org/cs/presenters>.

[10] Dokumentace Nette Frameworku. *Píšeme komponenty* [online]. [cit. 30. dubna 2013]. Dostupné na: <http://doc.nette.org/cs/components>.

[11] Dokumentace Nette Frameworku. *Routování URL* [online]. [cit. 30. dubna 2013]. Dostupné na: <http://doc.nette.org/cs/routing>.

[12] Dokumentace Nette Frameworku. *Rozšíření jazyka PHP* [online]. [cit. 30. dubna 2013]. Dostupné na: <http://doc.nette.org/cs/php-language-enhancements>.

[13] Dokumentace Nette Frameworku. *Rozšíření pro DI kontejner* [online]. [cit. 30. dubna 2013]. Dostupné na: <doc.nette.org/cs/di-extensions>.

[14] Dokumentace Nette Frameworku. *Sessions* [online]. [cit. 30. dubna 2013]. Dostupné na: <http://doc.nette.org/cs/sessions>.

[15] Dokumentace Nette Frameworku. *Výchozí helpery šablon* [online]. [cit. 30. dubna 2013]. Dostupné na: <http://doc.nette.org/cs/default-helpers>.

[16] Dokumentace Nette Frameworku. *Výchozí Latte makra* [online]. [cit. 30. dubna 2013]. Dostupné na: <http://doc.nette.org/cs/default-macros>.

[17] Dokumentace Nette Frameworku. *Šablony* [online]. [cit. 30. dubna 2013]. Dostupné na: <http://doc.nette.org/cs/templating>.

[18] Ferg, S. *Event-Driven Programming: Introduction, Tutorial, History* [online]. 8.2. 2006 [cit. 30. dubna 2013]. Dostupné na: <http://eventdrivenpgm.sourceforge.net>.

[19] Grudl, D. *Nette Framework: MVC & MVP* [online]. 24.3.2009 [cit. 29. dubna 2013]. Dostupné na: <http://www.zdrojak.cz/clanky/nette-framework-mvc-mvp>.

[20] JSON. *Úvod do JSON* [online]. [cit. 2. května 2013]. Dostupné na: <http://www.json.org/json-cz.html>.

[21] Kosek, J. *PHP a XML*. [b.m.]: Grada Publishing, 2009. ISBN 978-80-247-1116-4.

[22] Mlýnková, I., Nečaský, M., Pokorný, J. et al. *XML Technologie Principy a aplikace v praxi*. [b.m.]: Grada Publishing, 2008. ISBN 978-80-247-2725-7.

[23] Večeřa, Z. *JSON: první kroky aneb používáme PHP a jQuery* [online]. 22.3.2010 [cit. 2. května 2013]. Dostupné na: <http://programujte.com/clanek/2008073101-json-prvni-kroky-aneb-pouzivame-php-a-jquery>.

[24] W3Schools. *XML Elements* [online]. [cit. 1. května 2013]. Dostupné na: <http://www.w3schools.com/xml/xml_elements.asp>.

[25] W3Schools. *XML Namespaces* [online]. [cit. 1. května 2013]. Dostupné na: <http://www.w3schools.com/xml/xml_namespaces.asp>.

[26] W3Schools. *XML Syntax Rules* [online]. [cit. 1. května 2013]. Dostupné na: <http://www.w3schools.com/xml/xml_syntax.asp>.

[27] Wikipedia, the free encyclopedia. *Callback (computer programming)* [online]. 17.3. 2013 [cit. 30. dubna 2013]. Dostupné na: <http://en.wikipedia.org/wiki/Callback_(computer_programming)>.

[28] Závodský, P. *XSS stále na scéně* [online]. 6.4. 2011 [cit. 30. dubna 2013]. Dostupné na: <http://www.root.cz/clanky/xss-stale-na-scene>.

Příloha A

Obsah CD

`/readme.txt` – obecné informace ohledně práce, požadavky a návod k použití

`/doc/tex` – zdrojové soubory LaTeX k této práci, sestavení do PDF lze provézt zadáním `make` v tomto adresáři

`/doc/tex/fig` – obrázky z této práce, ve formátech PDF či PNG

`/doc/pdf/project-doc.pdf` – tato technická zpráva ve formátu PDF

`/doc/pdf/ref/XMLdesign.pdf` – uživatelská příručka popisu výchozích XML zdrojových souborů formálně definovaných v souboru `/Nexi/lib/Generators/components.xsd` pro generování tříd komponent v knihovně Nexi

`/doc/pdf/ref/NEONconf.pdf` – uživatelská příručka popisu konfigurace knihovny Nexi ve formátu NEON

`/doc/pdf/ref/LatteMacros.pdf` – uživatelská příručka nových Latte maker v knihovně Nexi

`/Nexi/lib` – zdrojové soubory knihovny Nexi

`/Nexi/API-reference` – API knihovny Nexi dohromady s API Nette Frameworku verze 2.0.10 stabilní ze dne 6.3. 2013 pro PHP 5.3 nebo 5.4, přístupné skrze soubor `index.html`

`/Nexi/sandbox` – rozšířená základní struktura aplikace v Nette pro knihovnu Nexi, obsahuje demonstrační aplikaci a vizualizační nástroj na komponenty

`/Nexi/Requirements-Checker/checker.php` – nástroj Nette Frameworku pro kontrolu, zda webový server a jeho nastavení splňuje minimální požadavky