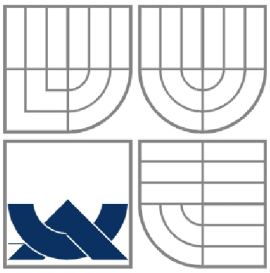
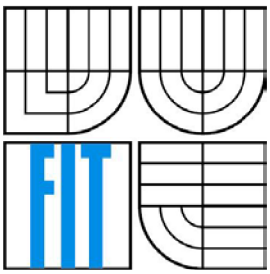


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO SPRÁVU OBSAHU ZALOŽENÝ NA ONTOLOGIÍCH

ONTOLOGY-BASED CONTENT MANAGEMENT SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ONDŘEJ ČEKAN

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK BURGET, Ph.D.

BRNO 2013

Abstrakt

Tato práce se zabývá analýzou, návrhem a implementací systému založeného na sémantických technologiích a ontologických jazycích. Je zde popsán sémantický web a jeho koncept, technologie sémantického webu především RDF a OWL a dále možnost zobrazení na WWW. Další část této práce se zabývá specifikací, analýzou, návrhem a samotnou implementací systému, který zpracuje ontologii a umožní uživateli vytvářet jedince v závislosti na její definici. Vytvořený obsah se pak prezentuje na webu anotovaně. Výsledkem této práce je demonstrační aplikace.

Abstract

This thesis deals with the analysis, design and implementation of a system based on semantic technologies and ontology languages. There is described the semantic web and its concept, technology of semantic web especially RDF and OWL and the ability to view on the Web. Another part of this work treats with the specification, analysis, design actual implementation of the system, which processes the ontology and allows users to create individual depending on the definition of the ontology. The created content is presented on the Web annotated. The result of this work is the demonstration application.

Klíčová slova

Systém pro správu obsahu, sémantický web, sémantické technologie, ontologie, databáze, sémantické úložiště, formulář, šablona, XML, HTML, XHTML, RDF, RDFa, OWL, GRDDL, MVC, PHP, MySQL, ARC2, Nette Framework, PHPOWL, Dibi.

Keywords

Content management system, semantic web, semantic technology, ontology, database, semantic storage, form, template, XML, HTML, XHTML, RDF, RDFa, OWL, GRDDL, MVC, PHP, MySQL, ARC2, Nette Framework, PHPOWL, Dibi.

Citace

Čekan Ondřej: Systém pro správu obsahu založený na ontologiích, diplomová práce, Brno, FIT VUT v Brně, 2013

Systém pro správu obsahu založený na ontologiích

Prohlášení

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

.....
Ondřej Čekan
22.5.2013

Poděkování

Děkuji vedoucímu práce panu Ing. Radku Burgetovi, Ph.D. za cenné rady a odbornou pomoc.

Obsah

Obsah 1
1 Úvod 2
2 Sémantický web 3
2.1 Koncept sémantického webu 4
2.1.1 XML 5
2.1.2 RDF 5
2.1.3 OWL 6
2.2 Možnosti publikování na WWW 8
2.2.1 RDFa 8
2.2.2 GRDDL 8
3 Analýza a návrh systému 9
3.1 Neformální specifikace 9
3.2 Analýza 10
3.2.1 Detaily případů použití 11
3.2.2 Výjimky případu použití 13
3.3 Návrh 14
3.3.1 Návrh databáze 15
3.3.2 Architektura systému 17
3.3.3 Úložiště sémantických dat 17
4 Výběr vhodné platformy 19
4.1 PHP 19
4.1.1 Nette Framework 20
4.1.2 PHPOWL 20
4.1.3 ARC2 21
4.2 MySQL 21
4.2.1 Dibi 22
5 Implementace 23
5.1 Grafický návrh 23
5.2 Převod grafického návrhu do HTML 24
5.3 Implementace systému 25
5.3.1 Adresářová struktura 25
5.3.2 Registrace a přihlašování uživatelů 29
5.3.3 Zpracování ontologie 29
5.3.4 Plnění ontologie 32
5.3.5 Šablonování 34
6 Testování 38
7 Demonstrační aplikace 39
7.1 Instalace aplikace 39
7.2 Popis aplikace 39
8 Závěr 41
Literatura 42
Seznam příloh 44

1 Úvod

V dnešním světě internetu platí pravidlo, kdo nemá vlastní internetovou stránku, jakoby neexistoval. Každý chce být vidět, každý chce přispět něčím svým, a proto každý den přibývá obrovské množství nových stránek, článků a různých informací, které by mohly zajímat široké spektrum návštěvníků.
Na internetu je v současnosti úplně vše, od služeb přes nákup či prodej až po zábavu. Stačí mít přístup k internetu a nekonečné možnosti zábavy jsou dostupné prakticky odkudkoliv. Není potřeba chodit osobně do obchodu, stačí jednoduše vybrat požadovanou věc na internetu a objednat ji z pohodlí domova. Zde se nabízí otázka, zda jsme schopni v této enormní směsi informací nalézt odpovídající odpověď rychle a správně?

Tim Berners-Lee, ředitel World Wide Web Consortium (W3C), přišel s touto otázkou a poukázal na problém, že současný web je změť informací, kde je čím dál složitější nalézt relevantní odpověď. Způsob, jakým jsou doposud data prezentována na webové stránce, je nevhodný ke strojovému zpracování. Značkovací jazyky používané v prostředí webu, tedy HyperText Markup Language (HTML) a jeho dodatečné rozšíření, slouží k formátování textu a vizuálního vzhledu stránky, ale nic už neříkají o významu jednotlivých prvků na stránce. Aktuální princip je určen především pro zobrazení a orientaci člověka, nikoli však ke strojovému zpracování a vyhodnocování. Tuto situaci o neznalosti sémantiky by měl vyřešit takzvaný sémantický web, který Tim Berners-Lee představil. Ten přidává k běžným datům na internetu ještě dodatečná data – metadata, která mají sjednotit jakýkoliv obsah na internetu. Podporuje běžné datové formáty na World Wide Web (WWW), a tak nezáleží na tom, zda je obsahem HTML stránka, obrázek, zvukový či jakýkoliv jiný dokument. Pomocí těchto dodatečných informací není problém strojově zjistit, co je obsahem daného souboru, to znamená, že lze automaticky vyhodnotit například to, co je vyobrazeno na obrázku.

Více informací o sémantickém webu bude probráno v druhé kapitole, ve které je věnována pozornost konceptu sémantického webu a jeho prvkům, kde se převážně zaměřujeme na RDF a OWL. Také v této kapitole popisujeme existující možnosti publikování sémantických dat na webu.

Kapitola třetí se zabývá samotným návrhem systému pro správu obsahu založeného na ontologiích, od neformální specifikace přes analýzu až po návrh systému. Je zde rovněž vysvětlen princip uložení sémantických dat.

V kapitole čtyři jsou vybrány a popsány vhodné technologie pro realizaci navrhovaného systému. Jsou zde rozebrány technologie PHP a MySQL a knihovny pracující na těchto technologiích, které byly použity pro realizaci této webové aplikace.

Pátá kapitola se věnuje samotné implementaci systému. Nejprve je navrhnut grafický návrh systému, který je následně převeden popsaným způsobem do HTML jazyka. Další část vysvětluje principy Nette Frameworku, který tvoří jádro aplikace, systému pro syntaktickou analýzu ontologií, generování formulářů pro vytváření jedinců v závislosti na dané ontologii, ukládání těchto formulářů do sémantického úložiště založeného na MySQL databázi a v neposlední řadě je zde popsáno zobrazení anotovaného obsahu na webu.

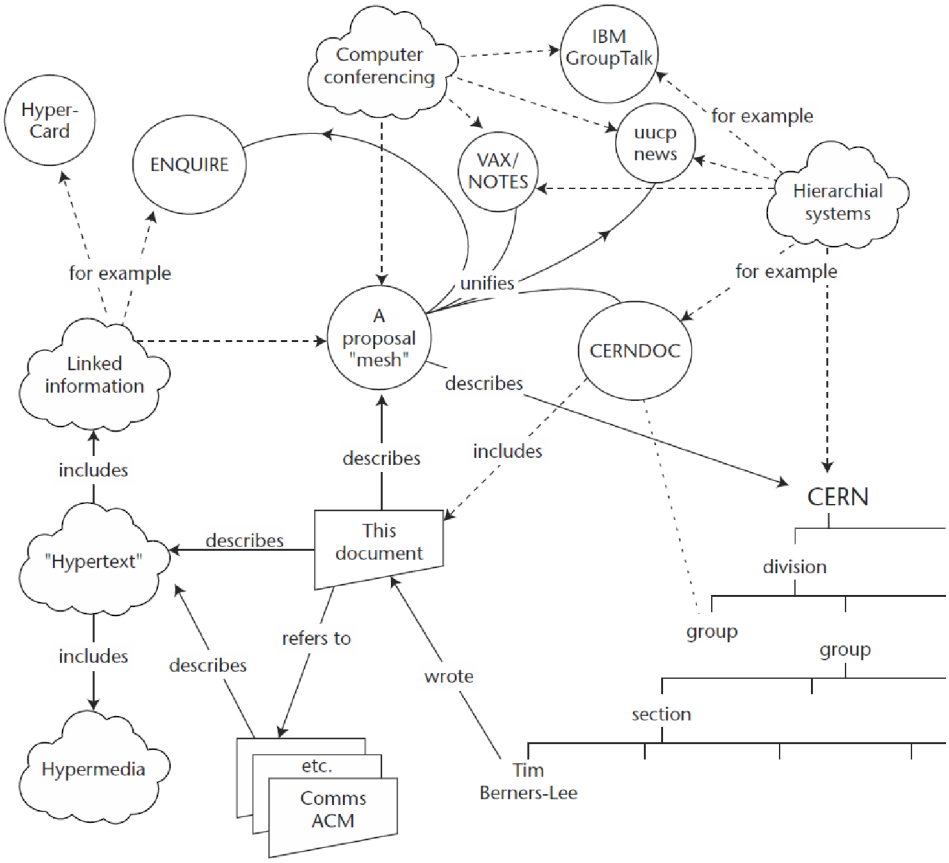
Kapitola šestá stručně popisuje způsoby testování komplexního systému, od testování grafického návrhu, až po zpracování ontologie a zobrazení anotovaného obsahu.

Sedmá kapitola se zabývá popisem a způsobem instalace samotné demonstrační aplikace na webový server.

V závěrečné kapitole jsou zhodnoceny dosažené výsledky s možností dalšího vývoje do budoucna.

2 Sémantický web

První představení sémantického webu se uskutečnilo roku 2001. Tim Berners-Lee jej představil jako budoucnost současného webu. Jeho myšlenka je založena na dvou idejích. První je vytvořit weby, které budou více spolupracovat mezi sebou, a tak pomoci lidem jednoduše sdílet informace a služby. Druhá myšlenka je vytvořit takovou prezentaci dat na internetu, aby byla srozumitelná pro stroje a aby jí bylo možno zpracovávat a vyhodnocovat strojově. Na obrázku 2.1 je původní vize webu tak, jak ji popsal Tim Berness-Lee. [1]

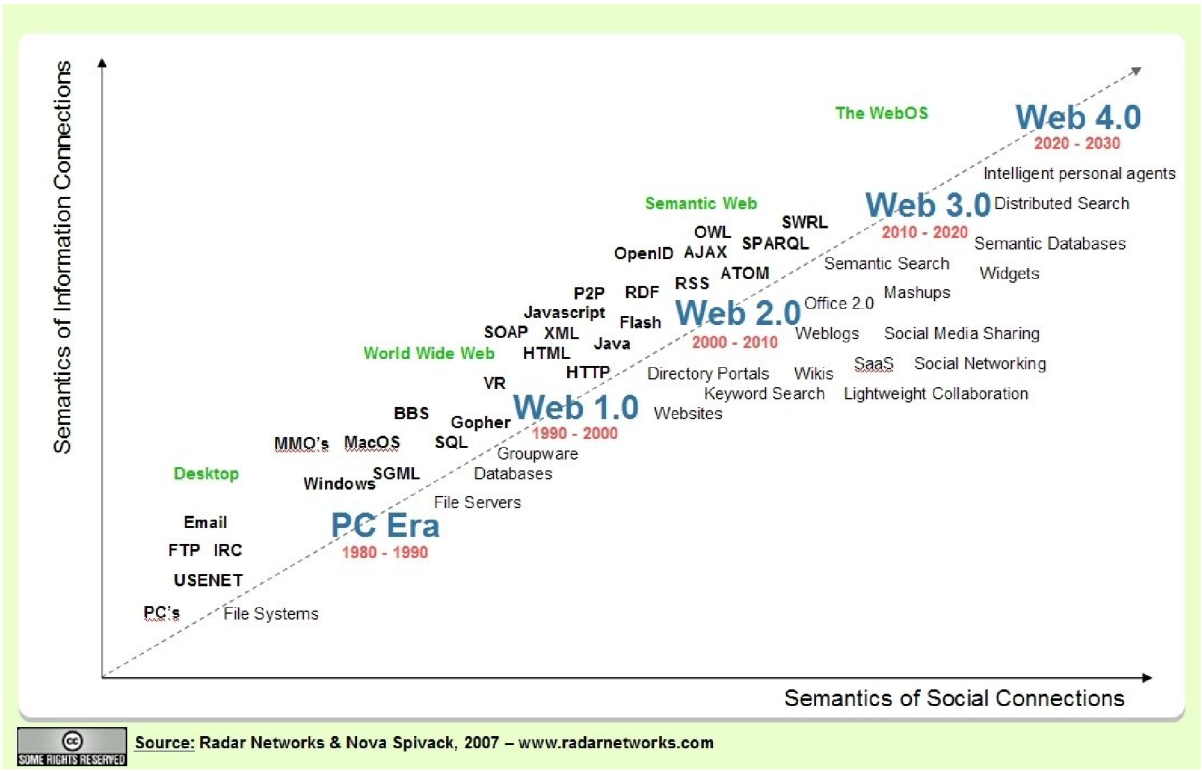


Obrázek 2.1: Původní vize webu popsaného Tim Berness-Lee. [1]

Na tomto diagramu lze vidět princip, který nespočívá pouze v obdržení HTML stránek z webového serveru, ale klíčovou roli v tomto konceptu rovněž hrají vztahy mezi položkami nesoucími informační hodnotu. Tyto vztahy jsou „includes“, „describes“ a „wrote“, tedy dokument „obsahuje“, „popisuje“ a je někým „napsán“. Vztahy v prostředí webu nelze popisovat, proto byla vytvořena technologie, která nabízí tyto možnosti. Jedná se o Resource Description Framework (RDF). Jak již bylo zmíněno v úvodu, hlavním prvkem pro přiřazení těchto vztahů či vazeb je přidání metadat k současné reprezentaci informací na webu. Tím získají data sémantiku. Díky RDF, který zavádí možnost tvrzení, lze zavést strojové zpracování a vyhodnocování.

Standard sémantického webu má na starost organizace W3C. Na následujícím obrázku 2.2 lze vidět, že sémantický web zapadá do období Web 3.0 (léta 2010-2020) a ačkoliv je znám více než 10 let, teď je právě ta doba, kdy by měl sémantický web zažít obrovský rozmach. Měla by vznikat řada sémantických vyhledávačů a sémantických databází. Sémantický web je ale stále ještě ve vývoji,

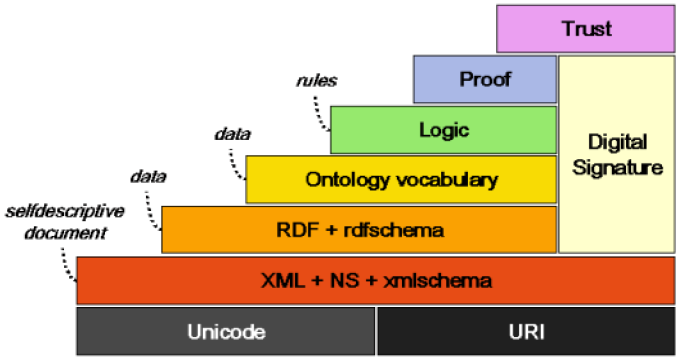
proto některé jeho části popsané v kapitole 2.1 nejsou plně dokončeny, a tak ještě chvíli potrvá, než dospěje do své finální podoby.



Obrázek 2.2: Vývoj webu. [2]

2.1 Koncept sémantického webu

Princip sémantického webu je rozložen do několika vrstev. Každá vrstva představuje určitou technologii používanou v prostředí webu. Rozložení těchto vrstev je znázorněno na obrázku 2.3. Vrstva Unicode zajišťuje použití mezinárodní znakové sady a Uniform resource identifier (URI) slouží jako identifikátor objektů sémantického webu. Extensible Markup Language (XML) vrstva je určena jako standard, který umožňuje převod do jiných jazyků založených na XML standardu. XML slouží ke strukturování dat pomocí vlastních definovaných značek, které mají zvolený význam a jsou sdíleny mezi aplikacemi. RDF a RDF Schema vrstva umožňuje vytvářet tvrzení o objektech identifikovaných pomocí URI. Vrstva ontologie má za úkol odvozování nových tvrzení a vztahů z již existujících. U vrstvy Digital Signature (digitální podpis) probíhá v současnosti standardizace a má na starost detekci změn v dokumentech. Poslední tři vrstvy Logic (logika), Proof (důkazy) a Trust (věrohodnost) jsou zatím stále ještě ve vývoji. Logika umožňuje vytváření speciálních pravidel, která jsou následně ve vyšších vrstvách vyhodnocována a ověřována z hlediska pravdivosti či nepravdivosti. [3]



Obrázek 2.3: Vrstvy sémantického webu. [3]

2.1.1 XML

Extensible Markup Language je standardizovaný značkovací jazyk navržený organizací W3C, určený především pro výměnu informací mezi různými aplikacemi. XML je podmnožina univerzálního značkovacího jazyka SGML. Jedná se o textový dokument, který tvoří značky s konkrétním obsahem. Značky nejsou pevně zadány a záleží tak na uživateli, jaké značky vytvoří a co bude jejich obsahem. XML je strukturovaný formát nezávislý na konkrétní aplikaci. Oproti složitým a komerčním formátům jako DOC či XLS není potřeba k zobrazení XML dokumentu speciálního softwaru, stačí i běžný textový editor. Navíc specifikace XML formátu je volně dostupná, a tudíž není problém do jakékoliv aplikace zahrnout podporu tohoto formátu. Jako implicitní kódování dokumentu se používá Unicode, ale autoři počítali i s použitím jiných kódování, a proto lze pomocí dodatečného parametru jednoduše tato kódování měnit, a tak mít například v různých částech XML dokumentu jiné kódování respektive jazyk. [4]

Jak bylo uvedeno, XML neobsahuje předdefinované značky a uživatel musí značky, které bude používat, vytvořit vlastní. Seznam těchto značek je možné nepovinně definovat v dokumentu typu DTD (Document Type Definition). Definování vlastních značek má výhodu v automatickém vyhodnocení správnosti XML dokumentu, zda obsahuje definované a tím pádem povolené značky.

XML formát se stal úspěšný především díky aplikačně nezávislým dokumentům a datům, standardní syntaxi, která vychází ze značkovacích jazyků, dostupnosti a jednoduchosti.

2.1.2 RDF

RDF (Resource Description Framework) je framework pro obecný popis webových zdrojů. Jedná se o technologický základ sémantického webu navržený skupinou W3C. Slouží k popisu, použití a výměně metadat. Základem tohoto frameworku je tzv. RDF trojice, která je nositelem tvrzení. Každé tvrzení tvoří tedy tři položky a to *Subjekt-Predikát-Objekt*. Predikát je nositelem dané vlastnosti subjektu a hodnota této vlastnost je definovaná v objektu. Jako příklad takovéto trojice můžeme uvést tvrzení – venku jemně mrholí. Potom subjekt je venku, predikát neboli vlastnost je sloveso mrholí a objektem je slovo jemně. Pokud se přesuneme do prostředí webových technologií, tak RDF popisuje informace o webových zdrojích, což je titulek stránky, autor, datum úpravy, popis obsahu a další. Každý zdroj je identifikován pomocí své URI. Zdroj může vystupovat jak na straně subjektu, tak na straně objektu. [5]

Základní syntaxe RDF vychází ze značkovacího jazyka XML, kde jednotlivé části tvrzení jsou řazeny za sebe ve formě prvků a atributů. Jedná se o tzv. serializaci. Obálkovým prvkem RDF dokumentu je RDF prvek. Obsahuje atributy, které definují jmenné prostory. Samotná tvrzení jsou tvořeny prvkem *Description*, jehož atribut *about* definuje zdroj, tedy subjekt. Podprvky v tomto prvku

představují jednotlivé predikáty, jejichž hodnoty definují obsah objektů. Příklad RDF dokumentu je znázorněn na obrázku 2.4. Tento způsob definice není ale jediný možný, jelikož RDF specifikace nabízí i jiné možnosti, jak vytvářet tvrzení. [5]

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

  <rdf:Description rdf:about="http://www.vutbr.cz">
    <rdf:title>Vysoké učení technické v Brně</rdf:title>
    <rdf:author>CVIS VUT v Brně</rdf:author>
  </rdf:Description>

</rdf:RDF>
```

Obrázek 2.4: Příklad RDF souboru.

2.1.3 OWL

OWL (Web Ontology Language) se používá stejně jako RDF tam, kde je potřeba informace zpracovávat automatizovaně aplikací. Nejedná se o interpretaci dat pro člověka. Je potřeba především, aby těmto informacím porozuměl stroj a dokázal obsah na webu zpracovat a vyhodnotit. Oproti nižším vrstvám sémantického webu (RDF, XML) nabízí rozšířenější možnosti pro vyjádření sémantiky. Ontologie reprezentuje význam jednotlivých pojmů a jejich vzájemné vztahy. OWL vychází z DAML OIL webového ontologického jazyka. OWL spravuje organizace W3C, pracovní skupina pod názvem OWL Working Group.

Samotný OWL jazyk je tvořen třemi převážně odlišnými podjazyky OWL Lite, OWL DL a OWL Full, jejichž vyjadřovací síla se postupně v každém podjazyku zvyšuje. Záleží na zvolení skupinou uživatelů a programátorů, který podjazyk je použit. V této podkapitole bylo čerpáno z [6].

OWL Lite

OWL Lite je jazyk, který poskytuje nejnižší vyjadřovací sílu ze všech příbuzných OWL jazyků. Formální složitost OWL Lite je nižší oproti dále přestaveným podjazykům OWL jazyka. Je určen především pro uživatele, kteří si vystačí pouze s hierarchií tříd a jednoduchými omezeními. Podporuje omezení kardinality, u které lze zvolit pouze mohutnost velikosti 0 anebo 1.

OWL DL

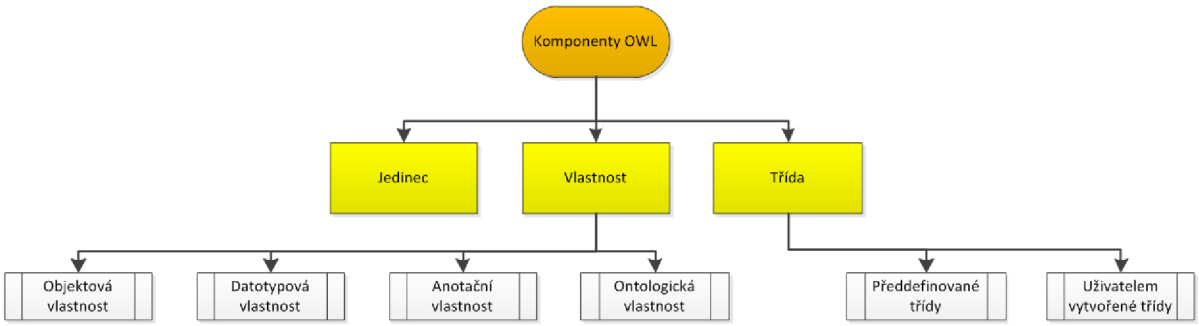
OWL DL je určen pro ty uživatele, kteří chtějí mít zachovánu co možná největší vyjadřovací sílu při zachování výpočetní úplnosti a rozhodnutelnosti. To znamená, že veškeré závěry lze jednoznačně vyhodnotit a to v konečném čase. OWL DL využívá veškeré prostředky, které samotný OWL jazyk nabízí, ale platí zde omezení, při kterých může být daná konstrukce použita.

OWL Full

OWL Full nabízí maximální vyjadřovací sílu. Jelikož OWL vychází z RDF, tak tuto verzi využijí především uživatelé, kteří chtějí syntaktickou svobodu RDF formátu. Není zde ale garantována výpočetní úplnost a rozhodnutelnost. OWL Full také umožňuje rozšířit ontologii o předdefinované slovníky. Jelikož OWL je velmi rozsáhlý jazyk není pravděpodobné, že by nějaká aplikace zahrnovala veškeré jeho výpočetní možnosti.

2.1.3.1 Komponenty jazyka OWL

OWL jazyk se skládá ze tří základních komponent, jimiž jsou jedinci, vlastnosti a třídy. Přehled komponent OWL jazyka znázorňuje následující obrázek číslo 2.5. Jednotlivé části budou popsány v následujících odstavcích. V této podkapitole bylo čerpáno z [7] a [8].



Obrázek 2.5: Komponenty jazyka OWL.

Jedinci

Jedinci neboli individua reprezentují objekty v doméně, kterou sledujeme. Jedná se o konkrétní výskyt, instance třídy. Daný jedinec může patřit do jedné nebo více tříd současně. Jako příklad můžeme uvést několik jedinců: Itálie, Albert, Trabant a jiné.

OWL nepoužívá unikátní jména UNA (Unique Name Assumption). Tím pádem mohou existovat alespoň dvě odlišná jména, která mohou odkazovat na jednoho stejného jedince. V OWL musí být proto explicitně uvedeno, zda daná individua jsou stejná anebo jsou každá jiná. Abychom tuto shodu či neshodu vyjádřili, existuje pro ni konstrukce `owl:sameAs` a `owl:differentFrom`.

Vlastnosti

Vlastnosti reprezentují vztahy mezi jedinci. Jedná se o binární relaci nad množinou jedinců, propojuje tedy dva jedince mezi sebou. Jako příklad uvedeme vlastnost „vlastní auto“ nad jedinci „Albert“ a „Trabant“, tedy „Albert vlastní auto Trabant“.

Rozlišujeme čtyři typy vlastností:

- objektová – vlastnost spojuje jedince s jedincem,
- datotypová – vlastnost spojuje jedince s hodnotou určitého datového typu,
- anotační – vlastnost přidává další informace (metadata) k jedincům, třídám, ontologii,
- ontologická – vztahy mezi ontologiemi.

Třídy

Třída (Class) v OWL představuje obecně chápaný význam pojmu třída, tedy množina jedinců, které mají stejné vlastnosti. Třídy jsou hierarchicky členěny do nadtříd a podtříd – taxonomie. Třídy lze rozdělit dvojím způsobem. První rozdělení je na třídy předdefinované a uživatelem vytvořené. Druhý způsob je rozdělení na anonymní a pojmenované třídy. Předdefinovanou třídou je například třída `owl:Thing`. Tato třída je nejobecnější a spadají do ní všechny ostatní třídy, i ty, které vytvoříme. Z hlediska pojmenování tříd OWL nenařizuje žádnou konvenci. Třídy budeme pojmenovávat svým názvem. V prostředí webu se bude jednat o URI. Takováto třída je tedy pojmenovaná. Zavedením axiomů subsumpce, ekvivalence a disjunkce odpovídající logickému výrazu nad pojmenovanými třídami získáme třídu anonymní. Anonymní třídu lze získat i z jiných anonymních tříd.

2.2 Možnosti publikování na WWW

Publikování sémanticky anotovaného obsahu na WWW představuje nadstavbu či rozšíření standardního HTML či XHTML (Extensible HyperText Markup Language) jazyka doplněním o existující nebo speciální značky a atributy. [9]

Jednu z možností, jak vkládat strojově čitelné informace do struktury internetových stránek, představují tzv. mikroformáty. Jedná se o uložení sémantických informací přímo do existujících atributů k existující značce. Pro tyto účely se používají atributy `class` a `rel`, popřípadě `rev`. [10]

V další části textu si představíme dvě složitější řešení, jak tuto sémantickou informaci vložit do struktury internetové stránky. Jednou z možností je použití RDFa, druhou pak GRDDL.

2.2.1 RDFa

RDFa (RDF in attributes) navrhla organizace W3C a představuje způsob, jak zobrazit RDF dokument pomocí XHTML jazyka. RDFa je podobný mikroformátům, obsahuje standardní syntaxi jako mikroformáty, ale oproti nim používá jiné značky a atributy respektive slovník termů. Nevýhodou tohoto řešení je nevalidita takto vytvořeného XHTML dokumentu, jelikož nestandardní prvky v dokumentu nejsou považovány za validní. [11] Příklad RDFa je ukázán na obrázku 2.6.

```
<p xmlns:dc="http://purl.org/dc/elements/1.1/"
about="http://www.example.com/books/wikinomics">
     In his latest book
     <cite property="dc:title">Wikinomics</cite>,
     <span property="dc:creator">Don Tapscott</span>
     explains deep changes in technology,
     demographics and business.
     The book is due to be published in
     <span property="dc:date" content="2006-10-01">October
     2006</span>.
</p>
```

Obrázek 2.6: Příklad RDFa [9]

2.2.2 GRDDL

GRDDL (Gleaning Resource Descriptions from Dialects of Languages) je mechanismus, který pomocí algoritmu reprezentovaného v XSLT (Extensible Stylesheet Language Transformations) dokáže převést XML či XHTML dokument do formátu RDF. U tohoto způsobu je zajištěna validita, jelikož dokument zahrnuje atribut, ve kterém je definován speciální prostor jmen. [12]

3 Analýza a návrh systému

V této kapitole jsou popsány jednotlivé kroky návrhu systému pro správu obsahu od neformální specifikace požadavků na systém přes analýzu až po samotný návrh. Cílem je definovat případy použití systému, jeho strukturu, návrhový vzor a veškeré potřebné prvky, podle kterých pak může být systém implementován.

3.1 Neformální specifikace

Cílem zadání je vytvořit webovou aplikaci, která zpracuje ontologii a umožní publikování anotovaných informací o konceptech této ontologie na WWW. Požadavkem je univerzální aplikace, které se předá definice ontologie se všemi adekvátními vazbami a uživatel má pak možnost pomocí webového formuláře vytvářet konkrétní objekty dané ontologie s odpovídajícími vlastnostmi a vazbami.

Systém umožní uživateli registrovat se do systému vyplnění základních osobních informací a následně se přihlásit pomocí uživatelského jména a hesla. Po přihlášení bude mít uživatel možnost zobrazit seznam dostupných ontologií, které jsou uložené na webovém serveru. Jednu z dostupných ontologií smí zvolit jako výchozí ontologii, jejíž anotovaný obsah se bude zobrazovat na hlavní stránce aplikace. Nové ontologie bude uživatel přidávat do systému importem zdrojového souboru z lokálního disku uživatele.

Po zvolení konkrétní ontologie získá uživatel možnost operovat s touto ontologií. To zahrnuje možnost vytvářet nové jedince dané ontologie, upravovat nebo mazat tyto jedince, přidávat, upravovat nebo odebírat vlastnosti či vztahy k těmto jedincům a to v závislosti na dané ontologii, s kterou je aktuálně pracováno. Takto vytvořená a naplněná ontologie by měla zůstat na serveru uložena, aby se uživatel po opětovné návštěvě, případně přihlášení, dostal do stavu ontologie jako před opuštěním webové aplikace.

U každé ontologie je vyžadován import šablon, které uživatel dodá speciálně pro danou ontologii. Tyto šablony se použijí pro zobrazení anotovaných dat, což představují uživatelem vytvořená individua.

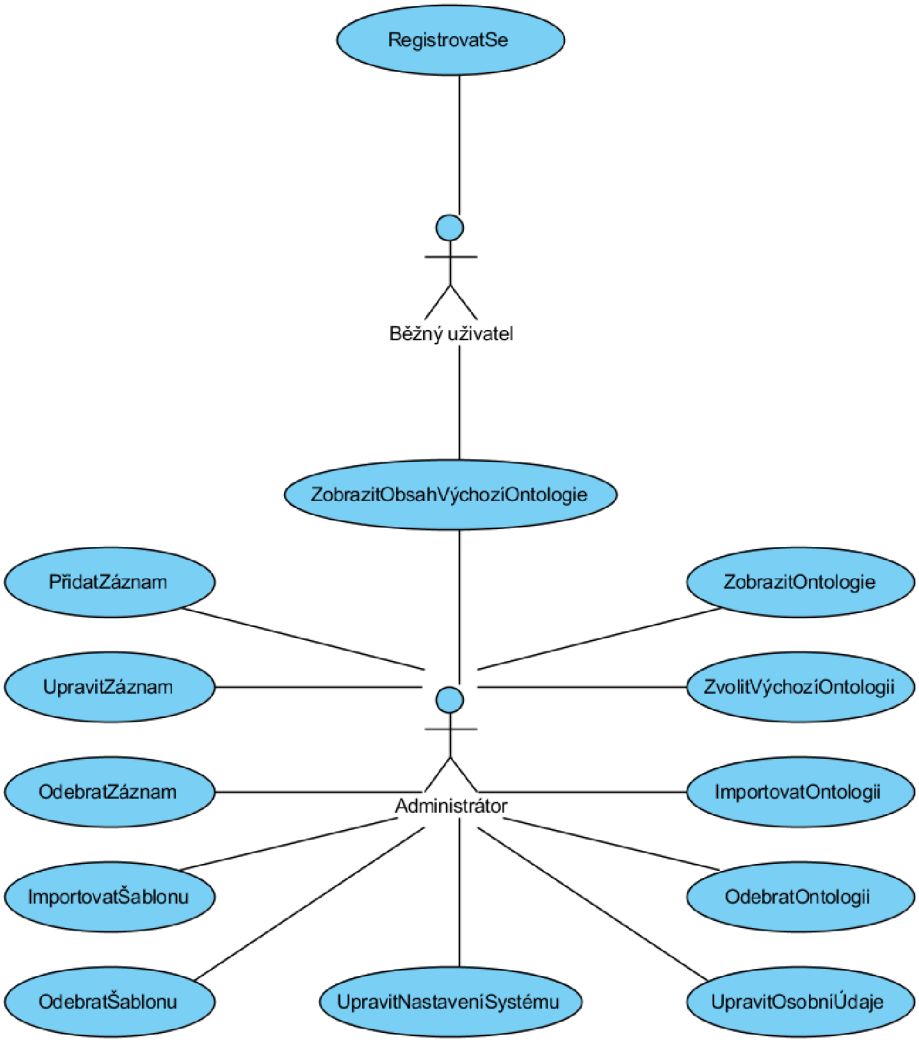
Systém by měl podporovat alespoň dva typy uživatelů. Jeden by měl být označen jako běžný uživatel. Běžný uživatel je uživatel, který není přihlášen do systému, a tudíž nemůže pracovat s ontologiemi. Smí zobrazit pouze anotovaná data výchozí ontologie, kterou druhý typ uživatele zvolil v administraci. Druhý typ uživatele označený jako administrátor rozšiřuje možnosti běžného uživatele a stane se z něj přihlášením do systému. Administrátor by měl oproti běžnému uživateli mít možnost měnit své osobní informace a spravovat ontologie způsobem popsaným výše.

3.2 Analýza

Z neformální specifikace byly odvozeny následující požadavky na systém. Byl vytvořen diagram případů použití, který zahrnuje činnosti, které mohou jednotliví aktéři vykonávat. Systém bude rozlišovat mezi dvěma typy rolí:

1) **Běžný uživatel** – aktér, představující nepřihlášeného uživatele. Je mu umožněno zobrazení výchozí ontologie. Aktér se smí rovněž registrovat do systému. Spravovat ontologie je tomuto aktérovi zakázáno.
2) **Administrátor** – představuje aktéra, který rozšiřuje činnosti běžného uživatele a tímto aktérem se stane z běžného uživatele přihlášením do systému. Tomuto aktérovi je umožněna správa ontologií, což představuje zobrazení dostupných ontologií, import a mazání ontologie, zvolení výchozí ontologie, možnost nahrání a odebrání šablon ke zvolené ontologii a vytváření, úprava a mazání záznamů individuí zvolené ontologie. Administrátor nemá možnost registrace nového uživatele, jelikož je přihlášený, ale díky tomu má možnost upravit své osobní údaje a rovněž základní nastavení serveru.

Výsledný diagram případu použití je zobrazen na obrázku 3.1.



Obrázek 3.1: Diagram případů použití.

3.2.1 Detaily případů použití

Případ použití:	RegistrovatSe	**ID:**	1
Stručný popis:	Zaregistrování nového uživatele do systému.		
Primární aktéři:	Běžný uživatel		
Předpoklady:	1. Aktér není přihlášen do systému a v systému je povolena registrace.		
Následné podmínky:	1. Systém zaregistroval nového uživatele.		
Hlavní tok:	1. Případ použití se spustí, když aktér vybere odkaz „Registrace“. 2. Aktér vyplní alespoň všechna povinná pole formuláře. 3. Pokud aktér nevyplnil některá povinná pole správně, zobrazí se chybová hláška a je vyžádána náprava, jinak pokračuj krokem 4. 4. Po volbě „Zaregistrovat“ systém zaregistruje nového uživatele.		
Výjimky:	Storno; Selhání operace; Selhání systému		
Frekvence:	Zřídka		

Případ použití:	ZobrazitObsahVýchozíOntologie	**ID:**	2
Stručný popis:	Zobrazí anotovaný obsah výchozí ontologie.		
Primární aktéři:	Běžný uživatel, Administrátor		
Předpoklady:	1. Je zvolena výchozí ontologie, pro kterou jsou definovány šablony.		
Následné podmínky:	1. Systém zobrazil anotovaný obsah výchozí ontologie.		
Hlavní tok:	1. Případ použití se spustí, když aktér vybere stránku (šablonu) z menu ontologie na hlavní stránce. 2. Aktérovi je zobrazen obsah výchozí ontologie v závislosti na šabloně.		
Výjimky:	Selhání operace; Selhání systému		
Frekvence:	Často		

Případ použití:	ZobrazitOntologie	**ID:**	3
Stručný popis:	Zobrazí dostupné ontologie v systému.		
Primární aktéři:	Administrátor		
Předpoklady:	1. Aktér je přihlášen do systému.		
Následné podmínky:	1. Systém zobrazí dostupné ontologie v systému.		
Hlavní tok:	1. Případ použití se spustí, když aktér vybere stránku „Ontologie“. 2. Aktérovi je zobrazen seznam dostupných ontologií v systému.		
Výjimky:	Selhání operace; Selhání systému		
Frekvence:	Často		

Případ použití:	ZvolitVýchozíOntologii	**ID:**	4
Stručný popis:	Zvolí výchozí ontologii v systému.		
Primární aktéři:	Administrátor		
Předpoklady:	1. Aktér je přihlášen do systému. 2. Aktér je na stránce „Ontologie“.		
Následné podmínky:	1. Systém nastaví výchozí ontologii v systému.		
Hlavní tok:	1. Případ použití se spustí, když aktér zvolí odkaz „Použít“ u jedné z dostupných ontologií. 2. Systém nastaví zvolenou ontologii jako výchozí.		
Výjimky:	Selhání operace; Selhání systému		
Frekvence:	Zřídka		

Případ použití:	ImportovatOntologii	**ID:**	5
Stručný popis:	Vloží novou ontologii do systému.		
Primární aktéři:	Administrátor		
Předpoklady:	1. Aktér je přihlášen do systému. 2. Aktér je na stránce „Ontologie“.		
Následné podmínky:	1. Systém přidá novou ontologii získanou z disku počítače aktéra.		
Hlavní tok:	1. Případ použití se spustí, když aktér zvolí odkaz „Nahrát“ ontologii. 2. Systém ověří zadané údaje, v případě nesprávnosti je zobrazena aktérovi chybová zpráva a je vyzván k nápravě, jinak pokračuj krokem 3. 3. Systém přidá novou ontologii.		
Výjimky:	Selhání operace; Selhání systému		
Frekvence:	Zřídka		

Případ použití:	OdebratOntologii	**ID:**	6
Stručný popis:	Odebere zvolenou ontologii ze systému.		
Primární aktéři:	Administrátor		
Předpoklady:	1. Aktér je přihlášen do systému. 2. Aktér je na stránce „Ontologie“.		
Následné podmínky:	1. Systém odebere zvolenou ontologii.		
Hlavní tok:	1. Případ použití se spustí, když aktér zvolí odkaz „Odstranit“ u jedné z dostupných ontologií. 2. Systém odebere zvolenou ontologii.		
Výjimky:	Storno; Selhání operace; Selhání systému		
Frekvence:	Zřídka		

Případ použití:	UpravitOsobníÚdaje	**ID:**	7
Stručný popis:	Upraví osobní informace aktéra v systému.		
Primární aktéři:	Administrátor		
Předpoklady:	1. Aktér je přihlášen do systému.		
Následné podmínky:	1. Systém upraví osobní informace aktéra.		
Hlavní tok:	1. Případ použití se spustí, když aktér vybere odkaz „Můj profil“. 2. Aktér vyplní alespoň všechna povinná pole formuláře. 3. Pokud aktér nevyplnil některá povinná pole správně, zobrazí se chybová hláška a je vyžádána náprava, jinak pokračuj krokem 4. 4. Po volbě „Upravit údaje“ systém upraví osobní informace uživatele.		
Výjimky:	Storno; Selhání operace; Selhání systému		
Frekvence:	Výjimečně		

Případ použití:	UpravitNastaveníSystému	**ID:**	8
Stručný popis:	Upraví základní nastavení systému.		
Primární aktéři:	Administrátor		
Předpoklady:	1. Aktér je přihlášen do systému.		
Následné podmínky:	1. Systém upraví nastavení.		
Hlavní tok:	1. Případ použití se spustí, když aktér vybere odkaz „Nastavení systému“. 2. Po volbě „Upravit systém“ systém upraví nastavení.		
Výjimky:	Storno; Selhání operace; Selhání systému		
Frekvence:	Výjimečně		

Případ použití:	PřidatZáznam/UpravitZáznam	**ID:**	9/10
Stručný popis:	Vloží/upraví jedince dané ontologie.		
Primární aktéři:	Administrátor		
Předpoklady:	1. Aktér je přihlášen do systému. 2. Aktér vybral třídu ontologie.		
Následné podmínky:	1. Systém přidá/upraví jedince třídy dané ontologie.		
Hlavní tok:	1. Případ použití se spustí, když aktér zvolí odkaz „Přidat záznam“/„Upravit záznam“. 2. Systém ověří zadaný identifikátor, v případě nesprávnosti je zobrazena aktérovi chybová zpráva a je vyzván k úpravě, jinak pokračuj krokem 3. 3. Systém přidá/upraví jedince.		
Výjimky:	Storno; Selhání operace; Selhání systému		
Frekvence:	Často		

Případ použití:	OdebratZáznam	**ID:**	11
Stručný popis:	Odebere zvoleného jedince z ontologie.		
Primární aktéři:	Administrátor		
Předpoklady:	1. Aktér je přihlášen do systému. 2. Aktér vybral třídu ontologie.		
Následné podmínky:	1. Systém odebere zvoleného jedince.		
Hlavní tok:	1. Případ použití se spustí, když aktér zvolí odkaz „Smazat“ záznam. 2. Systém odebere zvoleného jedince z ontologie.		
Výjimky:	Storno; Selhání operace; Selhání systému		
Frekvence:	Zřídka		

3.2.2 Výjimky případu použití

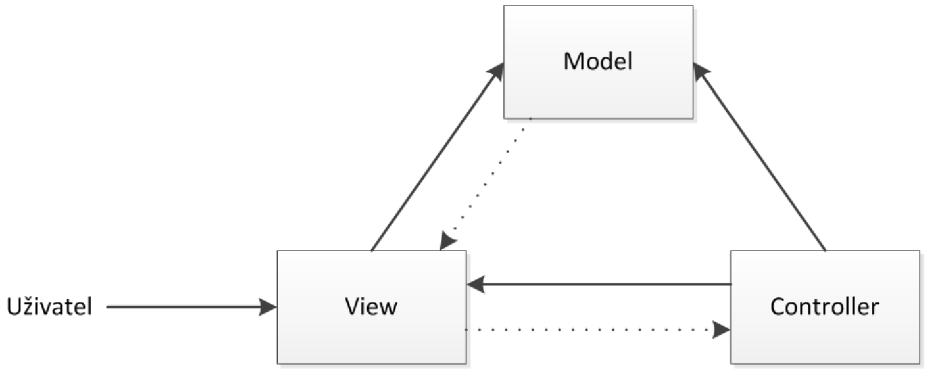
Název výjimky:	Storno	**ID:**	E.1
Stručný popis:	Přerušení operace.		
Primární aktéři:	Dle daného případu užití.		
Předpoklady:	1. Aktér provedl storno operaci.		
Následné podmínky:	1. V systému nebyly provedeny žádné změny.		
Tok:	1. Alternativní tok může aktér spustit kdykoliv před posledním krokem hlavního toku.		
Frekvence:	Zřídka		

Název výjimky:	Selhání operace/systému	**ID:**	E.2/E.3
Stručný popis:	Systém nedokáže pokračovat ve své činnosti.		
Primární aktéři:	Dle daného případu užití nebo systém.		
Předpoklady:	1. Nekorektní chování systému. 2. Systém nedokáže pokračovat ve své činnosti.		
Následné podmínky:	1. V systému nebyly provedeny žádné změny nebo je systém ukončen.		
Tok:	1. Alternativní tok může vyvolat systém během provádění hlavního toku. 2. Systém informuje uživatele o selhání.		
Frekvence:	Výjimečně		

3.3 Návrh

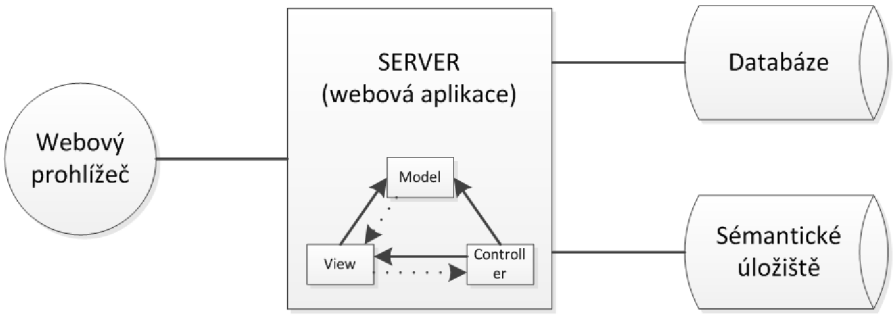
Jelikož se jedná o webovou aplikaci, komunikace s uživatelem bude probíhat pomocí webového prohlížeče a protokolu HTTP (Hypertext Transfer Protocol), který je nejvíce používaným v prostředí internetu a webových stránek. Jedná se o bezstavový protokol, fungující na principu dotaz-odpověď.

Softwarová architektura celkového systému bude vycházet z námi zvoleného návrhového vzoru MVC (Model-View-Controller), který je vhodným a často používaným vzorem ve webovém prostředí. Tento návrhový vzor rozděluje daný systém na tři oddělené, ale spolupracující části. Tyto části tvoří datový model aplikace, uživatelské rozhraní a řídící logika. Datový model ve vzoru MVC je zodpovědný za modelování domény a přidává aplikační logiku. Uživatelské rozhraní odpovídá za prezentaci modelu tak, aby byl srozumitelný převážně pro člověka. Ve webovém prostředí bývá model nejčastěji převáděn do značkovacího jazyka HTML. Řídící logika zpracovává a reaguje na události, především tedy na uživatelské podměty. [13] Obecné schéma návrhového vzoru MVC je zobrazeno na obrázku 3.2.



Obrázek 3.2: Návrhový vzor MVC.

Na obrázku 3.3 je pak ukázáno obecné schéma systému. Je zde možno vidět, že budou použita dvě úložiště. Relační databáze bude použita pro uložení dat týkajících se systému jako takového, tedy údaje o uživatelích, nastavení a zabezpečení funkčnosti systému. Struktura relační databáze bude popsána v další podkapitole. Sémantické úložiště bude sloužit pro uložení vytvořeného obsahu ontologie. Uložení bude realizováno pomocí sémantických technologií, které budou popsány rovněž v následující podkapitole.



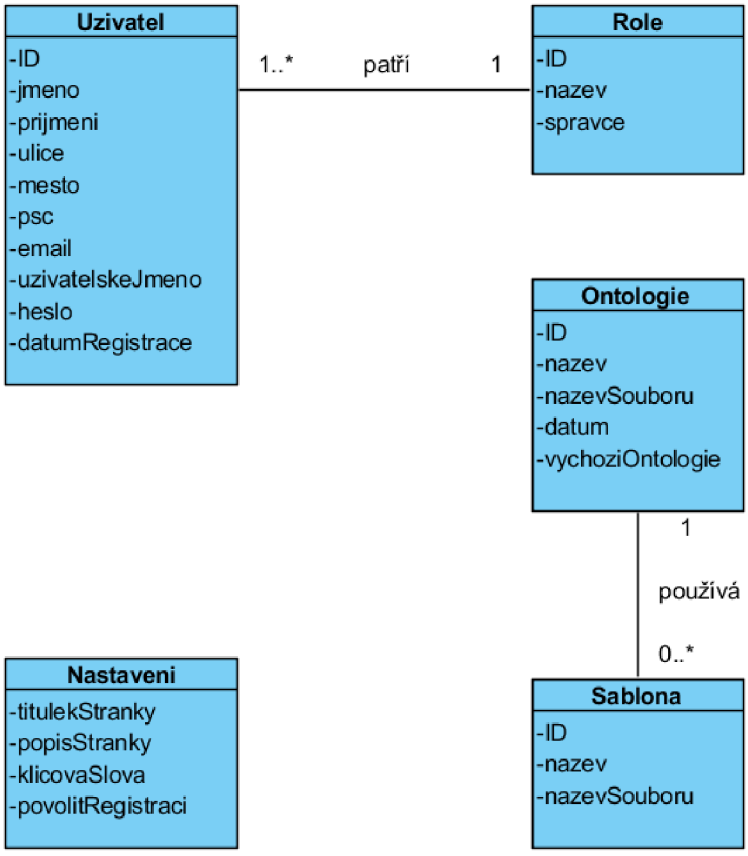
Obrázek 3.3: Obecné schéma systému.

3.3.1 Návrh databáze

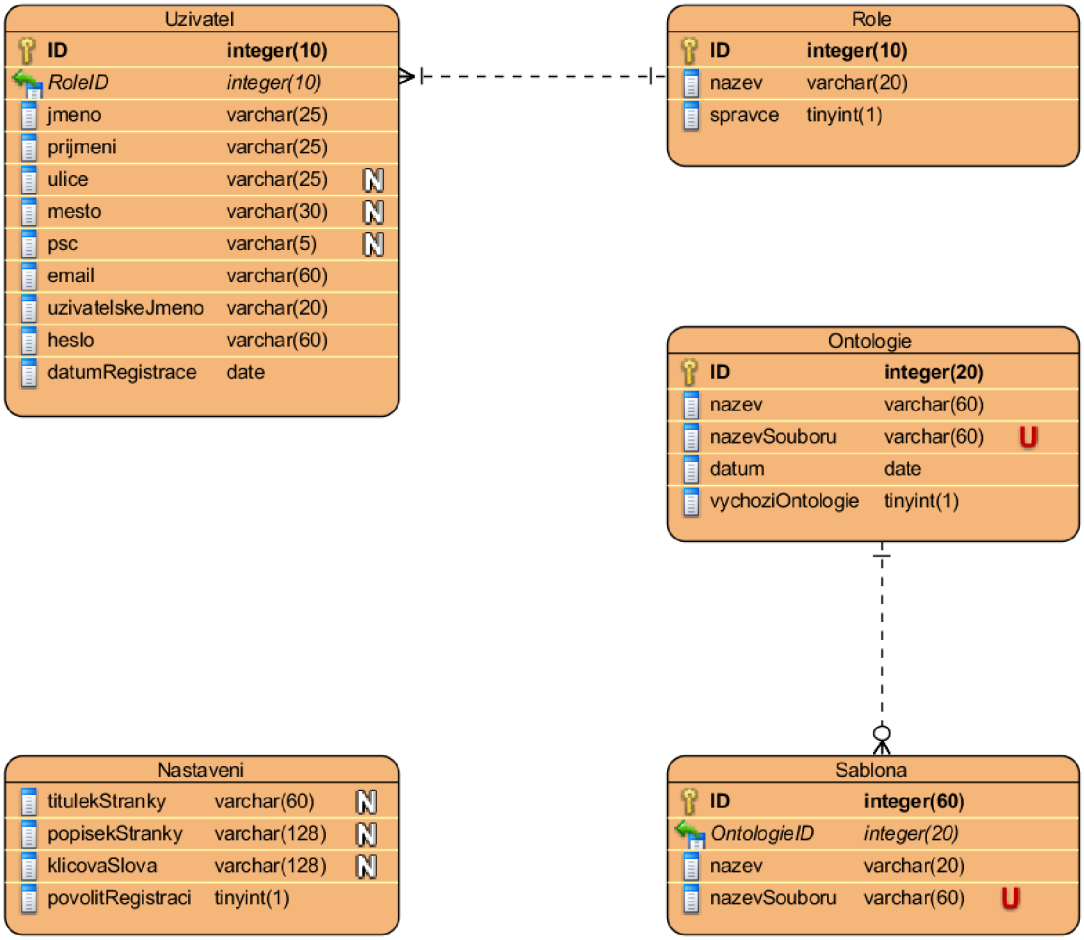
V této podkapitole bude navrženo a ukázáno schéma databáze. Při návrhu databáze byly zahrnuty požadavky na systém, které vyplynuly z neformální specifikace. Z neformální specifikace plyne, že je nutné uchovávat informace o uživatelích, případně jejich rolích, informace o nastavení základních parametrů systému a rovněž také evidovat dostupné vložené ontologie a jejich šablony. Dále byly při návrhu databáze brány ohledy na potřeby funkčnosti celkového systému.

Z těchto všech aspektů bylo odvozeno 5 tříd, které jsou potřeba pro zajištění požadované funkčnosti navrhovaného systému. Tyto třídy zobrazuje konceptuální diagram tříd na obrázku 3.4. Diagram zachycuje data v „klidu“, jedná se o tzv. statickou strukturu. Dále na obrázku 3.5 je navrženo schéma databáze. Význam jednotlivých tříd je následující:

- **Uzivatel** – zahrnuje informace o uživatelích systému.
- **Role** – role, ke které je přiřazen každý uživatel.
- **Nastaveni** – základní parametry pro nastavení systému.
- **Ontologie** – informace o dostupných ontologiích v systému.
- **Sablona** – šablony pro ontologie.



Obrázek 3.4: Konceptuální diagram tříd.



Obrázek 3.5: Návrh schématu databáze.

Pro upřesnění uvedeme význam jednotlivých sloupců u tabulek Role, Ontologie a Sablona, kde by nemuselo být jasné, k čemu je určen daný sloupec.

Tabulka Role obsahuje identifikátor role „ID“, název role „nazev“ a sloupec „spravce“, který nese bitový příznak 0 nebo 1. V případě, že u dané role jsou vyžadovány administrátorské pravomoci, je tento příznak nastaven do hodnoty 1, v opačném případě je nastavena hodnota 0.

Tabulka Ontologie zahrnuje primární klíč „ID“, název ontologie „nazev“, název souboru „nazevSouboru“, který určuje název uložené ontologie na serveru. Sloupec „datum“ nese informaci o datu, kdy byla importována ontologie do systému. Sloupec vychoziOntologie obsahuje bitový příznak, který je nastaven do hodnoty 1 u jediné ontologie, která má být jako výchozí. U ostatních ontologií je tento sloupec v hodnotě 0.

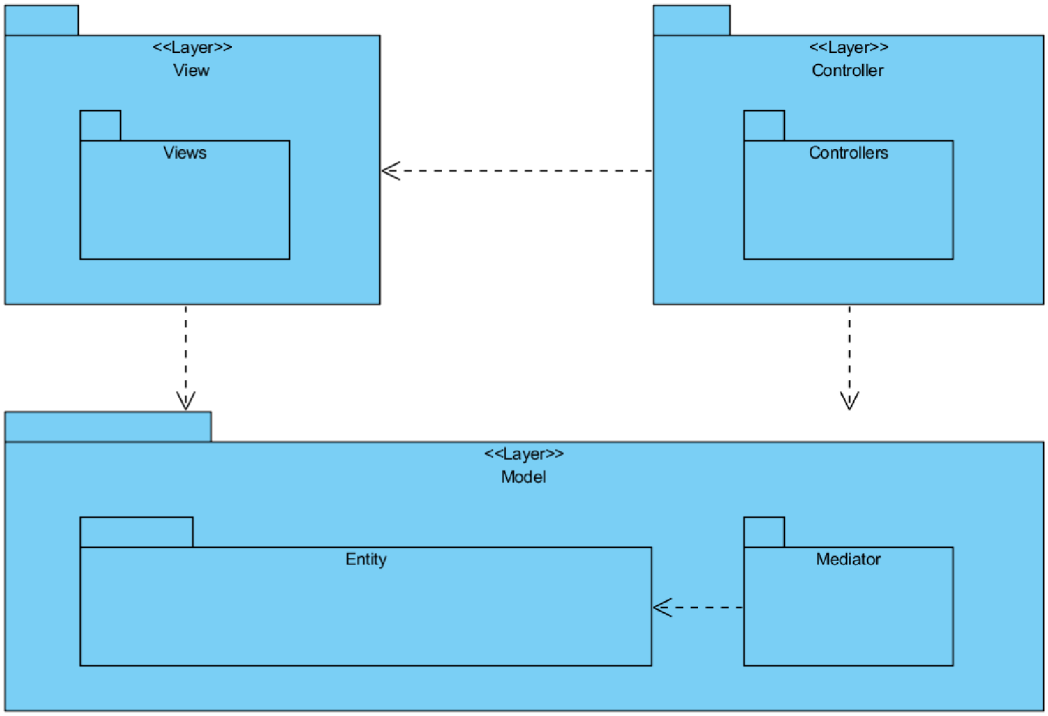
Tabulka Sablona obsahuje opět identifikátor „ID“. Součástí tabulky je také cizí klíč „OntologieID“, který odkazuje na jednu konkrétní ontologii, ke které šablona patří. Dále obsahuje název šablony „nazev“ a název souboru „nazevSouboru“, pod jakým je daná šablona uložena na webovém serveru.

3.3.2 Architektura systému

Jak již bylo uvedeno, softwarová architektura navrhovaného systému bude vycházet z návrhového vzoru MVC. [13] Systém byl rozdělen do tří částí tak, aby odpovídal návrhovému vzoru. Jednotlivé vrstvy si nyní stručně popíšeme:

- Vrstva **Controller** – zpracovává události, především vytvořené uživatelem. Může posílat příkazy vrstvě View, která změní prezentaci modelu, nebo obdobně může zaslat příkazy vrstvě Model, která upraví stav systému.
- Vrstva **Model –** přidává datům aplikační logiku v závislosti na požadavcích vrstvy Controller.
- Vrstva **View** – získává z modelu potřebná data, aby vytvořil výstupní reprezentaci, především pro porozumění člověku.

Architektura systému je ukázána na obrázku 3.6, kde jsou zobrazeny pouze základní vrstvy systému.



Obrázek 3.6: Architektura systému vzoru MVC.

3.3.3 Úložiště sémantických dat

S rozvojem sémantických technologií přišla otázka, jak sémantická data uchovávat. Specifické uložení dat je typické pro každou novou technologii, standard či jazyk. To, jak budou data uložena, ovlivňuje parametry jejich pozdějšího zpracování. Lze využít současných úložišť jako například relačních databází, ale tento způsob je komplikovaný. V této kapitole bylo čerpáno z [14].

3.3.3.1 RDF úložiště

Pro sémantická RDF data byla vytvořena nová speciální úložiště, tzv. RDF úložiště nebo také „tripple store“, která se snaží zajistit co nejlepší kvalitu služeb pro uchování sémantických dat. K dispozici jsou dva hlavní způsoby uložení a to buď databázový, nebo nativní.

Databázová úložiště

Pro ukládání se využívají relační databáze. Relační databáze ale poskytují databázové fixní schéma pro popis struktury. Oproti tomu RDF data mají schéma dynamické, což patří mezi jejich výhodu, ale rovněž to představuje problém s ukládáním. Proto existují tři způsoby mapování, jak lze RDF data mapovat na schéma relační databáze:

- nezohledňující schéma,
- zohledňující schéma,
- kombinovaná.

Nativní úložiště

Nativní úložiště přestavují vlastní řešení pro ukládání sémantických dat. Snaží se odstranit problémy při ukládání do relačních databází. Nativní úložiště je obvykle řešeno ukládáním vlastního formátu do souboru či přímo do operační paměti.

3.3.3.2 Dotazovací jazyky

S přístupem RDF úložiště bylo rovněž potřeba vytvořit nový dotazovací jazyk. Využití SQL (Structured Query Language) jazyka, který se používá v relačních databázích, není vhodné z důvodu efektivity úložiště. Proto vznikly nové dotazovací jazyky, které jsou inspirovány jazykem SQL nebo jsou založeny na pravidlech (podobné jazyku Prolog). Jazyky inspirovány SQL jsou například RQL, SeRQL, RDQL. Jako jazyky založené na pravidlech uvedeme TRIPLE, N3.

Problém u těchto jazyků je ten, že nejsou standardizovány. Tento problém je od roku 2004 v řešení a vzniká jazyk SPARQL, který zavádí standardizaci. SPARQL je složen ze dvou částí: dotazovacího jazyka a protokolu. Dotazovací jazyk dovoluje klást dotazy na uložená data a protokol definuje způsob dotazu a odpovědi.

4 Výběr vhodné platformy

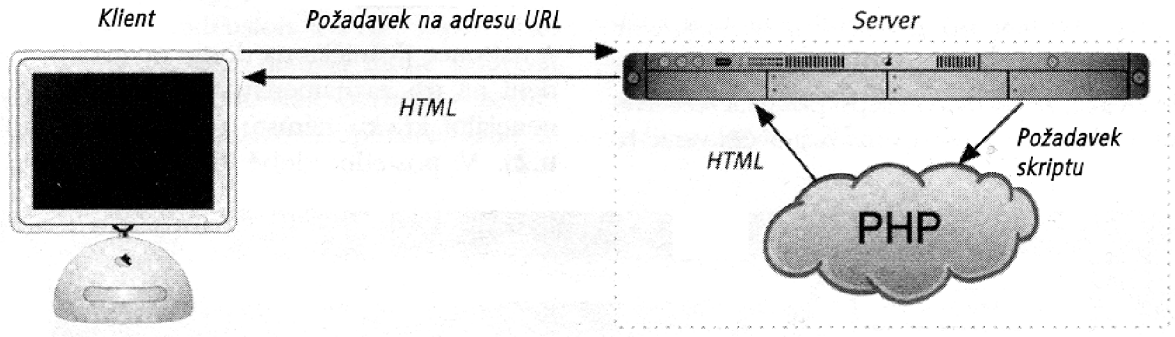
Jelikož se jedná o webovou aplikaci, je možno vybírat mezi několika technologiemi používanými ve webovém prostředí. Základním stavebním prvkem každé webové stránky je značkovací jazyk HTML případně XHTML. Tento jazyk představuje způsob, jak označit některé části dat na webu a vizuálně je interpretovat uživateli. S HTML souvisí i technologie CSS (Cascading Style Sheets) a JavaScript, které rozšiřují standardní možnosti HTML jazyka. Díky CSS je možné oddělit vzhled od obsahu dokumentu. JavaScript umožňuje provádění skriptů v klientském prohlížeči a tím je možno plynule provádět činnosti bez potřeby opětovného načtení dané stránky.

Tyto technologie jsou určeny pro tvorbu klientské části a jsou použity prakticky na každé webové stránce, tudíž i v případě naší aplikace budou použity. V rámci serverové části byly po dohodě s vedoucím zvoleny technologie PHP a MySQL. Tyto technologie jsou zdarma dostupné a v současné době patří mezi nejpoužívanější technologie pro vytváření webových prezentací. Rovněž k těmto technologiím a jejich jazykům existuje rozsáhlá a propracovaná dokumentace.

4.1 PHP

PHP [15] neboli PHP: Hypertext Preprocessor je mnohoúčelový skriptovací jazyk používaný především pro tvorbu dynamických webových stránek, ale nejen pro ně. V PHP lze vytvářet rovněž konzolové nebo desktopové aplikace. Je šířen pod open-source licencí, což je jeden z důvodů, proč se stal tak oblíbeným. Druhým důvodem je jeho jednoduchost použití. Kombinuje vlastnosti z několika programovacích jazyků, a tak dává uživateli větší volnost v syntaxi, která je podobná jazyku C. PHP je často používán ve spojení s databázovým serverem MySQL a webovým serverem Apache, které zajišťují základní software pro chod webového serveru. Často se rovněž používá označení LAMP, což znamená Linux, Apache, MySQL a PHP.

Na obrázku 4.1 je znázorněn princip získání PHP dokumentu ze serveru. Komunikace probíhá přes protokol HTTP, tedy dotaz-odpověď. Klient zašle dotaz s požadavkem na získání konkrétního PHP dokumentu daným URL adresou. Server tento dotaz obdrží a než pošle odpověď zpět klientovi, provede kód napsaný v tomto PHP dokumentu. Výsledek po provedení dokumentu pošle uživateli. V případě požadavku na statický soubor, což může reprezentovat HTML dokument, obrázek, textový dokument a podobně, je vynechána část zpracování PHP a ihned je tento požadovaný dokument zaslán zpět uživateli bez jakékoliv dodatečné činnosti.



Obrázek 4.1: Požadavek na PHP server. [15]

Aby webový server poznal, jaká část kódu je skript v jazyce PHP a jaká není, je nutné veškerý PHP kód umístit mezi značky:

```
<?php … ?>
```

popřípadě jen:

```
<? … ?>.
```

Kód mezi těmito značkami bude zpracován interpretem jazyka PHP před odesláním zpět klientovi.

4.1.1 Nette Framework

Nette Framework [16] dále jen Nette je rozsáhlá knihovna založená na jazyce PHP. Nette je šířen pod open-source licencí, je založen na architektuře MVP (Model-View-Presenter), obdoba MVC, a pro vývojáře nabízí řadu tříd a funkcí, které řeší často používané operace při vytváření webových aplikací. Jedná se například o vytvoření a zpracování formulářů, přihlašování uživatelů, práci s databází a jiné. Nette využívá vlastní šablonovací systém, vytváří cache soubory pro rychlejší načítání stránek, umožňuje používat dodatečná rozšíření. Vývojáři rovněž nabízí ladící nástroje, které usnadní a pomohou zavčas odhalit případnou chybu v programování. Mezi další výhody patří dokonalé zabezpečení, které eliminuje zneužití bezpečnostních děr, podporuje dobré návyky a objektový návrh aplikace, obsahuje rozsáhlou dokumentaci s řadou návodů, podporuje nejmodernější webové technologie a patří mezi nejrychlejší dostupné frameworky.

Na Nette je postavena řada známých webových stránek například Mladá fronta, Česko-Slovenská filmová databáze, Deník.cz. Díky zmiňovaným výhodám bude i naše aplikace postavena na jádře Nette Framework. Více informací o Nette lze nalézt na oficiálních stránkách http://www.nette.org/.

4.1.2 PHPOWL

PHPOWL [17] je aplikační knihovna napsaná v jazyce PHP, která umožňuje číst RDF/OWL soubory, a zároveň také provádí syntaktickou analýzu nad čteným obsahem. Knihovna byla vytvořena 16.2.2012 a její poslední verze 0.9.2 je z data 25.5.2012, z čehož je patrné, že knihovna nebyla přibližně rok aktualizována. Knihovna není příliš rozsáhlá a zahrnuje analýzu pouze některých značek a atributů (viz tabulka 4.1) a jejich základní konstrukce v RDF/OWL dokumentu. Tato knihovna je šířena pod licencí Creative Commons Attribution Non-Commercial License V2.0, která zahrnuje možnosti bezplatného použití a modifikace pro nekomerční účely. Tato knihovna je použita v práci viz kapitola 5, ale jelikož obsahuje zpracování pouze minimálního množství značek a atributů, musela být značně rozšířena tak, aby pokryla všechny důležité značky a atributy jazyka RDF/OWL. Seznam rozšíření je uveden v příloze 1.

Značka	Atributy
owl:Ontology	-
owl:Class	rdf:about
owl:ObjectProperty	rdf:about
rdfs:subClassOf	rdf:resource
rdfs:subPropertyOf	rdf:resource, rdf:about
rdf:RDF	xml:base
rdfs:comment	-
+ instance tříd a vlastností	

Tabulka 4.1: Přehled podporovaných značek a atributů knihovny PHPOWL.

Z tabulky 4.1 vyplývá, že knihovna umí z RDF/OWL souboru syntakticky zjistit, jaké má daná ontologie třídy, jaká třída má jaké podtřídy popřípadě nadtřídy, jaké jsou v ontologii objektové vlastnosti, jaká je základní adresa zpracovávané ontologie a v poslední řadě dokáže přečíst konkrétní instance tříd a vlastností a vytvořit trojice *Subjekt-Predikát-Objekt*. Tato syntaktická analýza je naprosto nedostatečná pro naši práci, protože vůbec nevypovídá o tom, jaká třída má jaké vlastnosti. Knihovna podporuje pouze objektové vlastnosti, tedy vztah *třída-vlastnost-třída*, ale nepodporuje například důležitou datotypovou vlastnost, která reprezentuje vztah *třída-vlastnost-hodnota*. Dále je patrné, že ačkoliv se knihovna vydává za RDF/OWL knihovnu, tak podporuje převážně OWL jazyk popřípadě RDF Schema, ale čisté RDF již nikoli, proto jsme se tímto problémem také zabývali v kapitole 5.

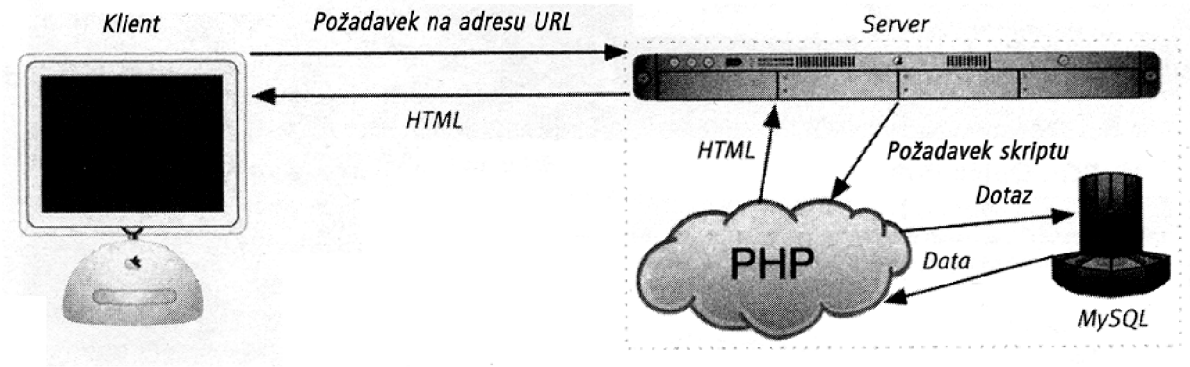
4.1.3 ARC2

ARC2 [18] je knihovna pro PHP, která má široké použití a zaměřuje se na sémantický web a především na práci s RDF soubory. Knihovna je šířena pod licencí W3C Software License nebo GPL. Je tedy volně dostupná a jednoduše se používá. Obsahuje speciální třídy a metody pro syntaktickou analýzu RDF/XML, N-Triples, Turtle a dalších formátů. Dále umožňuje serializaci dat do těchto formátů, nabízí RDF úložiště, které je řešeno pomocí MySQL databáze, a s tím je spojená podpora SPARQL dotazovacího jazyka nad tímto úložištěm. Jako užitečná se jeví také možnost rozšíření pomocí zásuvných modulů, podpora proxy adresy a přesměrování.

4.2 MySQL

MySQL [15] představuje multiplatformní databázový systém, který využívá dotazovací jazyk SQL (Structured Query Language). Databáze vznikla v roce 1995 a v současné době se jedná o populární a velmi používanou databázi pro webové aplikace ve spojení s PHP. MySQL je relační databáze využívající fixní schéma pro uložení dat. Je k dispozici v bezplatné nebo komerční licenci. Svou oblibu si získala především díky jednoduchému nasazení, volnou šiřitelností a velkému výkonu.

Pro správu MySQL databáze existuje řada aplikací, které usnadňují práci s datovým úložištěm a v přehledné podobě zobrazí jeho obsah. Často používanou aplikací je tzv. Adminer, který je například součástí Nette. Další možností je využití aplikace phpMyAdmin.



Obrázek 4.2: Požadavek na PHP server s dotazem do MySQL databáze. [15]

Na obrázku 4.2 je znázorněn průběh zaslání požadavku od klienta na server a zpět. V případě, že je potřeba provést dotaz do databáze, je při provádění PHP kódu skript pozastaven a je proveden dotaz do databázového serveru. Skript dále pokračuje až po získání odpovědi z databáze.

4.2.1 Dibi

Dibi [19] je knihovna pro PHP verze 5, která má na starosti nejen práci s MySQL databází, ale podporuje rovněž databáze PostgreSQL, SQLite, MS SQL, Oracle a další. Vznikla z toho důvodu, že klasické PHP funkce poskytují pouze rozhraní pro přístup k databázi. Dibi je v tomto dál a řeší otázky SQL injekce, zjednodušuje práci, umožňuje klást sofistikovanější dotazy a také myslí na potřeby přenositelnosti – automatická podpora konvencí, formátování speciálních typů, jednotný přístup k databázi a podobně. Knihovna Dibi dosáhla své konečné verze a sám autor tvrdí, že již nelze vymýšlet další rozšíření knihovny. Tato knihovna má rovněž kvalitní dokumentaci s řadou příkladů. Více informací o knihovně Dibi lze nalézt na oficiálních stránkách http://www.dibiphp.com/.

5 Implementace

V této kapitole je rozebrána celková implementace navrženého systému. Systém je implementován za pomocí dříve zmiňovaných technologií.

Jako první je rozebrán grafický návrh systému a jeho překódování do HTML jazyka. Následuje popis struktury systému, jeho jednotlivých částí, propojení částí do jednoho celku a především popis ontologického řešení.

Navržený systém dostal označení OWK, což je zkratka pro anglický název Web Ontology Kit. OWK by se dalo přeložit jako sada nástrojů pro práci s ontologiemi na webu.

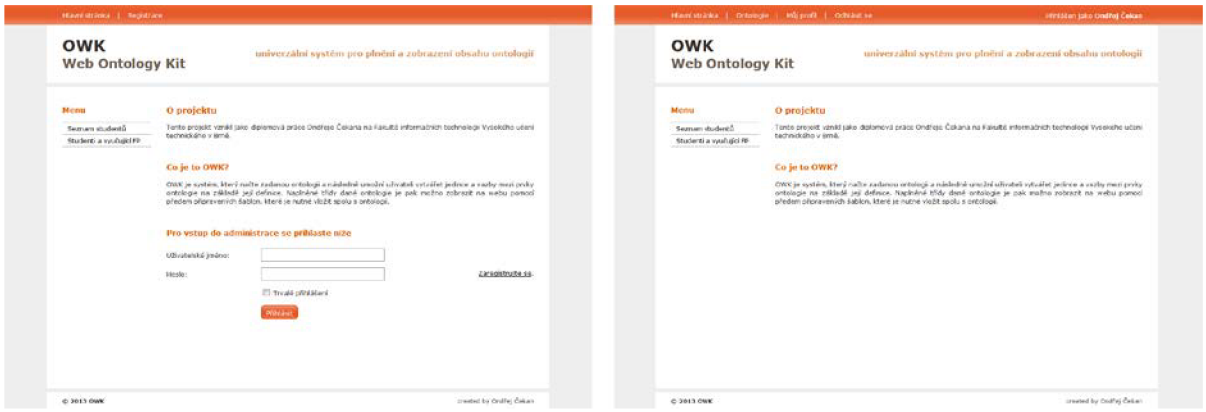
5.1 Grafický návrh

Navržený grafický podklad (viz obrázek 5.1) se snaží působit na uživatele moderním dojmem, přehledností a jednoduchostí. Byl zvolen světlý design, který je rozdělen na tři části – hlavička, obsah, patička, které odděluje šedý pruh splývající s pozadím. Díky tomu návrh působí čistým dojmem.

Hlavička je tvořena nápisem OWK a dodatečným textem, který popisuje samotný účel stránek. To zajistí, že uživateli bude ihned zřejmé, co je podstatou našich stránek, pokud je navštíví. Součástí hlavičky je i oranžový pruh přes celou šíři stránek, který představuje hlavní menu stránek. Hlavička se až na výjimky v zobrazení pro přihlášeného a nepřihlášeného uživatele nemění.

Obsahová část stránek je tvořena bílým pozadím a černým textem. Nadpisy jsou zvýrazněny oranžovou barvou a tučným písmem, aby uživatele na první pohled zaujaly a jednoduše se zorientoval v obsahu. Obsah je specifický pro každou stránku, tudíž je proměnný. Na obrázku hlavní stránky je v obsahové části vyobrazeno další menu, jedná se o menu ontologie. Toto menu bude vysvětleno v další části textu práce.

Celou webovou stránku zakončuje patička. Patička má neměnný obsah a je pevně umístěna u dolního okraje webového prohlížeče respektive stránky.



Obrázek 5.1: Grafický návrh hlavní stránky aplikace pro nepřihlášeného (vlevo) a přihlášeného (vpravo) uživatele

Grafický podklad byl navržen v programu Adobe Photoshop CS5.1. V tomto programu byl grafický návrh rovněž nařezán a vyexportován jako samostatné obrázky. Tyto obrázky jsou dále použity při překódování do HTML jazyka.

5.2 Převod grafického návrhu do HTML

Aby se grafický návrh mohl stát plnohodnotnou webovou aplikací, je potřeba jej převést do HTML jazyka. Ačkoliv pro převod obrázku do HTML existují již nástroje, my tento převod provedeme ručně vlastní tvorbou HTML a CSS kódu. Důvod je prostý, nástroje přidávají do kódu redundanci, vytváří složité konstrukce, většina návrhu je tvořena obrázkem. Oproti tomu vlastní tvorba a především zkušený kodér vytváří čistý a přehledný kód, jasně definované a pojmenované třídy, optimalizuje stránky pro SEO (Search Engine Optimization), obrázky používá minimálně a spíše se zaměřuje na využití možností, které nabízí kaskádové styly.

Proces samotného kódování začíná přípravou základní konstrukce značek, kterou představují značky:

```
<!DOCTYPE html>
<html lang="cs">
        <head>
        …
        </head>
        <body>
        …
        </body>
</html>
```

Značka `!DOCTYPE` definuje prostor jmen, který se má použít. Tato konkrétní definice určuje, že se jedná o HTML verze 5. Následuje značka `html`, která zastřešuje všechen HTML kód a její obsah je tvořen dvěma základními značkami `head` a `body`. Tyto značky se smí v HTML kódu vyskytnout pouze jednou. Značka `head` představuje hlavičku HTML kódů. Obsahuje `meta` informace o stránce jako například autor, kódování, popis stránky a další. V této části jsou rovněž definovány odkazy na externí CSS a JavaScript soubory a mimo jiné také titulek stránky `title`. Druhá značka `body` definuje obsah dokumentu.

Ve značce `body` je umístěna značka `div` s identifikátorem `#wrap`, která pomocí CSS vlastnosti `margin: 0 auto;` vystřeďuje obsah stránky do středu uživatelského prohlížeče. Stejně jako byl rozdělen grafický návrh na tři části – hlavička, obsah, patička, tak byl také HTML kód rozdělen na tři části pomocí blokových značek `div` s identifikátory `#top`, `#main`, `#bottom`. Z důvodu SEO optimalizace je pořadí upraveno tak, aby obsah stránky `#main` byl v co možná nejvrchnější části HTML kódu, proto zbylé dva identifikátory jsou umístěny v kódu níže a tento deficit je ve výsledném zobrazení vykompenzován nastavením absolutní pozice k těmto identifikátorům. Jednotlivé značky jsou dále zanořovány a dle potřeby jsou k nim přiřazovány identifikátory a třídy, ke kterým jsou poté v CSS souboru definovány vlastnosti.

Jak bylo uvedeno v předchozí podkapitole, máme k dispozici potřebné části grafického návrhu ve formě obrázků. Tyto obrázky jsou v CSS souboru propojeny či přiřazeny konkrétním značkám pomocí vlastnosti `background: url('cesta k obrázku');`, která zajistí jejich zobrazení v prohlížeči.

Hlavní menu stránky, popřípadě i menu ontologie, je tvořeno nečíslovaným seznamem `ul` a jeho položkami `li`, které jsou nastavené pomocí CSS tak, aby se zobrazovaly v jedné linii. Tento způsob tvorby menu se po řadě našich zkušeností jeví jako nejpraktičtější. Pomocí této struktury lze vytvořit libovolné zobrazení tohoto menu jen za pomoci CSS vlastností. Pro zajímavost uvedeme, že lze například vytvořit i rozbalovací podmenu v tomto menu.

Pro psaní HTML a CSS kódu byl použit editor NetBeans IDE, který bude stručně popsán v následující podkapitole.

5.3 Implementace systému

Tato podkapitola popisuje strukturu aplikace a postupně představuje význam a funkci důležitých zdrojových souborů.

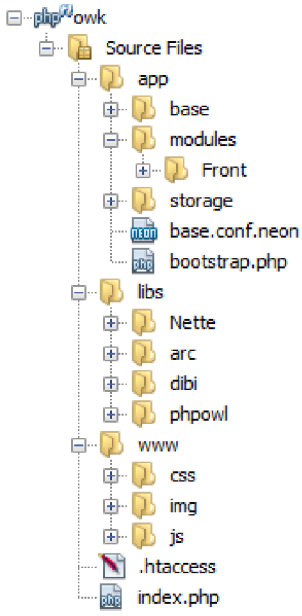
Implementace systému probíhala za pomoci nástroje NetBeans IDE verze 7.2.1, který je zdarma dostupný a umí zvýrazňovat syntaxi pro všechny potřebné jazyky. Do NetBeans IDE byl doinstalován modul pro podporu knihovny Nette Framework, který přidává do nástroje syntaktické zvýrazňování knihovních funkcí, podporu šablonování a jiné. Jako hlavní prohlížeč pro prohlížení vytvářené webové aplikace byl zvolen prohlížeč Mozilla Firefox, který patří mezi nejpoužívanější prohlížeče, je volně dostupný a stáhnout jej lze na adrese http://www.mozilla.com/.

NetBeans IDE

NetBeans IDE [20] je volně dostupný nástroj pro programátory, který umožňuje psát, ladit, překládat a distribuovat vytvářené aplikace. Jedná se o jeden ze dvou produktů vytvářeným pod projektem NetBeans, na jehož vývoji se podílí široká komunita uživatelů. Projekt NetBeans byl založen v roce 2000 společností Sun Microsystems, která je také hlavním sponzorem. NetBeans IDE je napsán v jazyce Java a podporuje kterýkoliv programovací jazyk. Rovněž umožňuje rozšířit svou funkcionalitu pomocí dodatečných modulů, které lze jednoduše doinstalovat.

5.3.1 Adresářová struktura

Aby bylo možno začít pracovat s knihovnou Nette Framework, je potřeba definovat adresářovou strukturu a definovat základní konfigurační pravidla, které Nette vyžaduje pro svůj chod. Na obrázku 5.2 je částečně ukázána adresářová struktura aplikace, která je založena na modulárním objektově orientovaném přístupu a kterou si nyní popíšeme.



Obrázek 5.2: Adresářová struktura aplikace – částečná.

V hlavním adresáři aplikace se nachází tři složky. Složka `app` zahrnuje zdrojové kódy týkající se námi vytvářené aplikace. Tento adresář nás bude zajímat nejvíce. Adresář `libs` slouží pro vkládání knihoven. Obsahuje adresář `Nette`, což je samotný Nette Framework. Další knihovnu představuje složka `arc`, která obsahuje zdrojové soubory pro RDF úložiště ARC2. Knihovnu Dibi, určenou pro práci s databází MySQL, reprezentuje adresář `dibi`. Posledním adresářem v této složce je adresář `phpowl`, který obsahuje zdrojové soubory knihovny PHPOWL pro čtení a syntaktickou analýzu RDF/OWL souborů. Každá knihovna má svůj vlastní zaváděcí soubor, který je zahrnut v souboru `index.php` v hlavním adresáři. Třetí složkou v hlavním adresáři je složka www, která obsahuje podklady z převodu grafického návrhu do HTML jazyka. Neobsahuje samotný HTML kód, ale pouze CSS, JavaScript a grafické soubory. HTML kód je použit v adresáři pro šablony, který je zanořen ve složce `app`. O šablonách si povíme až později v další části textu.

5.3.1.1 Vstupní bod aplikace

Jako u každé webové aplikace je vstupním bodem, který se automaticky načte při návštěvě webové adresy, soubor `index.(htm|html|php|asp|...)`. V našem případě, se jedná o PHP aplikaci, tudíž vstupní soubor je `index.php`. Ten nejprve načte všechny dostupné knihovny pomocí funkce `require` a následně nakonfiguruje přístup k databázi pro sémantické úložiště ARC2. Přístup k sémantickému úložišti je pak možný prostřednictvím proměnné `$store`. Jako poslední operaci v tomto souboru je potřeba načíst zaváděcí soubor `/app/bootstrap.php`, který slouží pro konfiguraci a spuštění knihovny Nette.

5.3.1.2 Soubor .htaccess

Jedná se o speciální soubor, který je na serveru skrytý a který umožňuje měnit nastavení některých vlastností serveru majitelem webové aplikace bez potřeby zažádání o změnu správce serveru. V našem případě je v tomto souboru nastaveno směrování, které přesměruje zadanou URL adresu na adresu s www na začátku v případě, že byla zadána URL adresa bez www. Dále je zde nastaven překlad tzv. pěkných URL adres, což je označováno pod pojmem `rewrite_module`, který musí být povolen v nastavení Apache serveru.

5.3.1.3 Zaváděcí soubor knihovny Nette

Zaváděcí soubor `bootstrap.php` knihovny Nette je umístěn v adresáři `/app`. Nejprve jsou nakonfigurována úložiště, která Nette vyžaduje pro svůj chod. Je vytvořena nová instance třídy `Nette\Config\Configurator`, u které je volána metoda `enableDebugger()`, pro povolení ladících informací a vizualizaci chyb, které jsou i následně ukládány do adresáře `/app/storage/log`. Metoda `setTempDirectory()`, nastavuje u konfigurátoru úložiště dočasných souborů, které jsou ukládány do adresáře `/app/storage/temp`. Pomocí metody `createRobotLoader()` dojde k automatickému načítání PHP tříd v adresářích `/app` a `/libs`. Poslední konfigurací, která se nastavuje, je přidání hlavního konfiguračního souboru `/app/base.config.neon` pomocí metody `addConfig()`. Tento konfigurační soubor nastavuje již vlastní aplikaci, kdežto zaváděcí soubor nastavuje knihovnu Nette. Následuje vytvoření systémového kontejneru metodou `createContainer()`, který obsahuje všechny služby a parametry nutné pro běh aplikace a samotné spuštění aplikace zavoláním metody `application->run()` nad systémovým kontejnerem.

5.3.1.4 Hlavní konfigurační soubor

Nette používá konfigurační soubory obvykle typu `.neon` a jak bylo uvedeno, v zaváděcím souboru je definován hlavní konfigurační soubor `/app/base.config.neon`. V konfiguračních souborech je striktně vyžadováno zarovnání jednotlivých položek. Konfigurační soubor lze rozdělit na tři části.

První část obsahuje konfiguraci pro produkční verzi, tedy verzi, která je již nasazena v ostrém provozu na internetu. Tato konfigurace musí být umístěna v sekci `production < common:`.

Druhou možností je definovat konfiguraci pro vývojovou verzi, která se použije, pokud aplikace běží na lokálním počítači, tj. IP (Internet Protocol) adresa je 127.0.0.1. Konfigurace pro vývojovou verzi má rovněž vlastní sekci pod označením `development < common:`.

Poslední částí je společná konfigurace pro produkční a vývojovou verzi, která je v naší aplikaci použita, a jedná se o sekci `common:`. Ta obsahuje sekci pro zadávání parametrů `parameters:` a následně sekci `database:`, kde jsou zadány údaje pro přístup Nette k MySQL databázi. Je zde zadán host, uživatelské jméno, heslo a databáze. Pro zobrazení MySQL dotazů, které jsou v danou chvíli vykonávány, je v této sekci povolen tzv. profiler zadáním hodnoty `profiler: true`. Posledním příkazem pro databázi je `lazy: true`, který vytváří spojení s databází až ve chvíli, kdy je skutečně potřeba. Sekce `php:` je další sekcí `common:` sekce a je v ní definována domovská časová zóna `date.timezone: Europe/Prague` pro PHP. Sekce `nette:` mění chování Nette Frameworku a je zde v sekci `session:` nastavena doba expirace sezení na 30 dnů a adresář pro ukládání sezení na serveru. V sekci `common:` se nachází ještě dvě sekce a to sekce `services:` a `includes:`. Sekce služeb `services:` umožňuje registrovat služby, pro které následně sama knihovna Nette vygeneruje PHP kód. Takto definujeme službu `DibiConnection`, které předáme parametry pro připojení do databáze a Nette již samo vytvoří konkrétní instanci tohoto připojení. Sekce `includes:` umožňuje vkládat další konfigurační soubory. V naší aplikaci je hlavní konfigurační soubor globální konfigurací a další specifické konfigurace konkrétních částí aplikace vkládáme pomocí dalších konfiguračních souborů definovaných v této sekci. Proto je zde definován soubor `/app/modules/Front/front.conf.neon`, který konfiguruje samotnou webovou aplikaci.

5.3.1.5 Základní třídy – Base

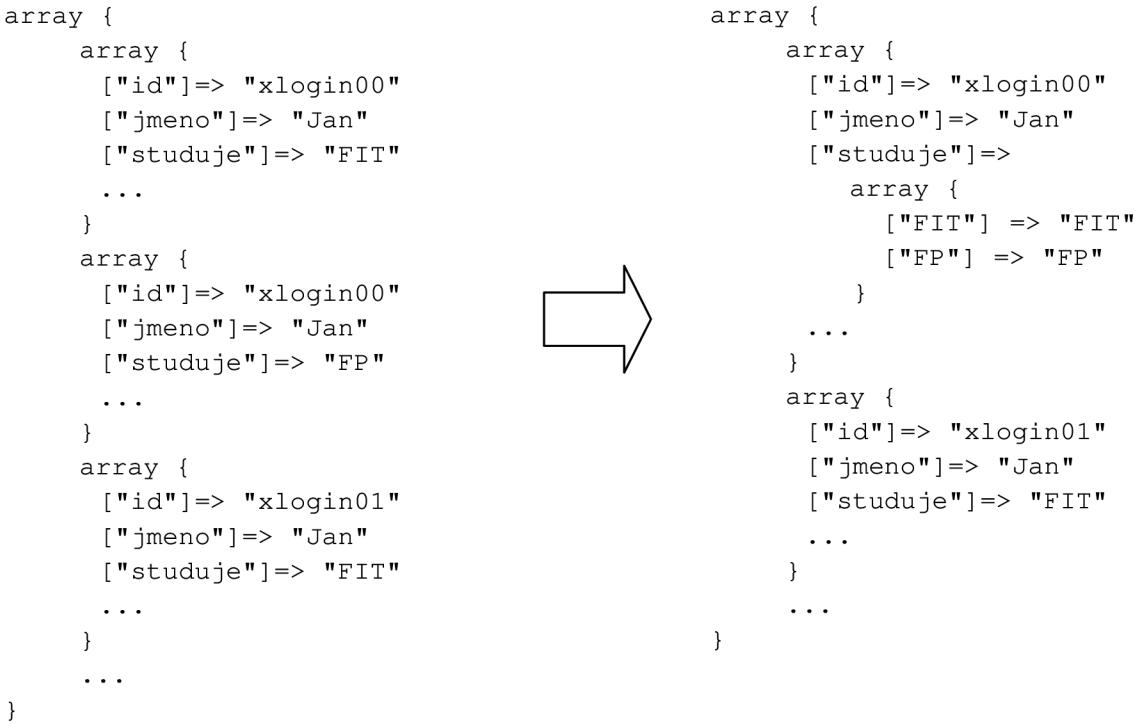
Protože Nette a PHP podporují objektově orientované programování, byly při vývoji systému definovány tři základní třídy, které jsou dle potřeby v dalších třídách děděny. Takto lze přenášet společné proměnné a metody do dalších zanořených tříd. Tyto třídy se nachází v adresáři `/app/base` a jedná se o zdrojové soubory `BaseForm.php`, `BaseModel.php` a `BasePresenter.php`.

`BaseForm.php` dědí ze třídy `Nette\Application\UI\Form` a rozšiřuje tak možnosti standardní třídy pro tvorbu a zpracování formulářů. Definuje třídu `BaseForm`, jejíž konstruktor volá metodu `novalidate()`, která vypíná vyhodnocování formuláře před jeho odesláním pomocí JavaScriptu a vyhodnocování probíhá až po odeslání na serveru. V konstruktoru jsou rovněž redefinovány výchozí chybové hlášky, které se vypíší v případě chybně vyplněného pole. Tyto hlášky jsou uloženy v poli `Nette\Forms\Rules::$defaultMessages`. Tato třída kromě konstruktoru obsahuje ještě tři metody. Metoda `addInput()` je obdobná jako standardní Nette metoda `addText()`, která kromě vložení běžného formulářového pole `<input>` obsahuje navíc pravidlo, že dané formulářové pole musí být vyplněné. Metoda `addSend()` vkládá formulářové

tlačítko `<button>` stejně jako standardní Nette metoda `addSubmit()` s tím rozdílem, že přidává k tlačítku třídu `btn` pro jednodušší stylování pomocí CSS.

`BaseModel.php` dědí ze třídy `\Nette\Object`, což je předkem všech tříd v Nette a je dobré z něj dědit i v případě, že žádné děděné vlastnosti třída nepotřebuje. Tato třída má na starost zajištění přístupu k databázi. Tím pádem díky dědičnosti bude mít každý model v naší aplikaci přístup k databázi také, jelikož tento model bude jejich předek. V konstruktoru pomocí definované služby `DibiConnection` získáme připojení k databázi a uložíme si jej do privátní proměnné `$db`. V případě potřeby databáze model zavolá svou metodu `db()`, která vrací přístupový bod k databázi v uložené proměnné `$db`.

`BasePresenter.php` obsahuje třídu `BasePresenter`, která je základem pro všechny ostatní presentery v naší aplikaci. `BasePresenter` dědí ze standardní Nette třídy `\Nette\Application\UI\Presenter`. V této třídě jsou obsaženy metody, které mohou být volány z každého presenteru respektive každé stránky naší aplikace. Metoda `handleLogout()` slouží pro odhlášení přihlášeného uživatele. Po úspěšném odhlášení je vypsána zpráva metodou `flashMessage()` a uživatel je přesměrován metodou `redirect()` na úvodní stránku aplikace. Metoda `unique_array()` se používá u získaných výsledků ze sémantického úložiště a slouží k seskupení několika výsledků dotazu do jednoho se stejným ID. Princip spočívá v procházení a porovnání jednotlivých výsledků, které jsou uložené v poli. Výsledky jsou seřazeny podle ID, tudíž pokud zjistíme, že dva po sobě jdoucí ID jsou shodné, provedeme porovnání všech hodnot respektive sloupců výsledků. Pokud se hodnoty neshodují, znamená to, že daný sloupec obsahuje více hodnot pro jeden konkrétní ID, a proto tyto hodnoty vložíme do pole a uložíme do sloupce prvního záznamu s daným ID. Po projití všech položek máme v prvním záznamu s daným ID přidány všechny hodnoty z druhého záznamu, proto druhý záznam můžeme odstranit. Takto projdeme všechny výsledky a získáme pole, u kterého je každý identifikátor jiný, tedy získáme správný výsledek. Popsané chování znázorňuje obrázek 5.3.



Obrázek 5.3: Příklad činnosti metody `unique_array()`.

5.3.2 Registrace a přihlašování uživatelů

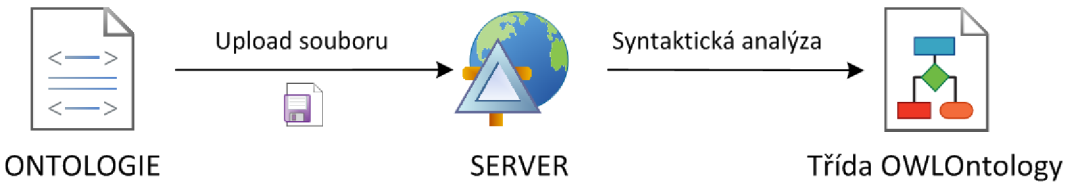
O registrování uživatelů se starají tři soubory a jejich třídy umístěné ve zvláštní komponentě `/app/modules/Front/Components/registration`. Třída `RegControl` má na starost vytvoření registračního formuláře a jeho zpracování po odeslání uživatelem. Formulář reprezentuje instanci třídy `BaseForm`, která pomocí metody `addInput()` nebo `addText()` vkládá textové pole. Pomocí metody `addRule()` nad takto vytvořeným polem lze definovat dodatečná omezení na uživatelský vstup, např. minimální délku. Po odeslání formuláře a jeho automatické validaci ze strany knihovny Nette je jeho obsah předán metodě `formProcess()`, která ověří, zda zadané uživatelské jméno již neexistuje v systému. Pokud jméno neexistuje, je uživatel zaregistrován vložením záznamu do databáze. S databází `Uživatel` operuje model `RegModel`. S drobnými modifikacemi ve funkčnosti se stejný formulář používá rovněž pro úpravu osobních informací.

Přihlašování je taktéž umístěno ve zvláštní komponentě `login`. Třída `LoginControl` podobně jako třída `RegControl` vytváří a zpracovává přihlašovací formulář. O samotné přihlášení uživatele se stará třída `Authenticator` a její metoda `authenticate()`, která po ověření zadaných údajů s databází přihlásí uživatele vytvořením nové identity `Nette\Security\Identity`.

5.3.3 Zpracování ontologie

Zpracování ontologie rozdělujeme na dvě části. První část představuje operace spojené se samotným importem ontologie na server. Druhá část je vlastní zpracování uložené ontologie a její syntaktická analýza, která je realizována pomocí knihovny PHPOWL a která jak již bylo uvedeno musela být značně rozšířena.

Veškeré operace spojené se syntaktickou analýzou ontologie se nachází v jednotlivých souborech knihovny PHPOWL. O vypsání výsledku uživateli se stará presenter `OntologyPresenter.php` umístěný ve složce `/app/modules/Front/Presenters`. Zpracování ontologie znázorňuje obrázek 5.4.



Obrázek 5.4: Princip zpracování ontologie – import ontologie na server a provedení syntaktické analýzy.

5.3.3.1 Import ontologie

Pro importování ontologie je vytvořen speciální formulář, který se nachází pod seznamem dostupných ontologií na serveru. Pro import je nutné zadat název ontologie, pod kterým se bude daná ontologie zobrazovat v seznamu ontologií, a vybrat soubor s ontologií z lokálního disku uživatelského počítače, který musí být typu `.xml`, `.rdf` nebo `.owl`. Ostatní typy nejsou akceptovány a import tudíž nelze provést. Stisknutím tlačítka nahrát dojde k přenosu souboru na server následujícím způsobem. Dojde k přejmenování názvu souboru. Nový název je složen z identifikátoru, který bude dané ontologii přidělen v databázi, a původní koncovky, která může být zmiňovaného typu. Tímto způsobem lze jednoduše zajistit jedinečné názvy ontologických souborů. Zároveň se do databázové tabulky pro

ontologie vloží záznam, který obsahuje identifikátor shodný s názvem souboru bez jeho koncovky, název ontologie, nový název souboru, datum vložení ontologie a příznak, zda se jedná o výchozí ontologii – standardně se vkládá 0. Ontologický soubor je poté uložen v adresáři vytvořeném speciálně pro ontologie, který má umístění `/app/modules/Front/Ontologies`.

5.3.3.2 Syntaktická analýza

Základem knihovny PHPOWL respektive syntaktického analyzátoru jsou třídy `OWLOntology` a `OWLReader`. `OWLOntology` reprezentuje samotnou ontologii. Obsahuje proměnné pro uchování tříd, nadtříd, podtříd, objektových a datotypových vlastností, vazeb, individuí, prostoru jmen a dalších. Tato třída obsahuje také metody, které přidávají hodnoty do těchto definovaných proměnných, a rovněž lze hodnoty z těchto proměnných získat. Po zpracování celé ontologie máme v této třídě uloženy veškeré informace o ontologii.

Třída `OWLReader` představuje třídu pro samotné zpracování ontologie. Její metoda `loadFromFile()` postupně zpracovává zadanou ontologii a výsledky zpracování ukládá do třídy `OWLOntology`. Zpracování řídí XML analyzátor, který je součástí PHP, a pro svou činnost vyžaduje zadání hlavního objektu a dvou metod, které mají na starost zpracování počáteční a koncové značky. V knihovně PHPOWL hlavní objekt představuje instanci třídy `OWLTag`. V případě, že XML analyzátor narazí ve zdrojovém souboru na jakoukoliv počáteční značku, předá ji spolu s atributy hlavnímu objektu a jeho metodě `openTag`, která tuto počáteční značku obslouží. Koncové značky obsluhuje metoda `closeTag`.

`OWLTag` se dá považovat za jádro celého syntaktického analyzátoru. Obsahuje vzorové definice značek a atributů, které umí zpracovat. Jestliže z XML analyzátoru přijde počáteční značka, systém se podívá do pole `$tagHandlerClasses`, ve kterém jsou uloženy známé značky a k nim odpovídající názvy tříd, které danou značku dokáží zpracovat. Například pokud přijde značka `owl:Class`, systém si za `owl` dosadí plnohodnotnou URI tohoto prostoru jmen, výsledek vypadá takto `http://www.w3.org/2002/07/owl#:Class`, a zjistí, že danou značku obsluhuje třída `OWLClassTag`. Pokud neexistuje pro danou značku obsluhující třída, tak danou značku zpracuje třída `OWLTag`. Po zpracování značky třídy, vlastnosti nebo individua je vytvořena nová instance, která je následně uložena do patřičné proměnné ve třídě `OWLOntology`. Pro třídu je to instance `OWLClass`, pro objektovou vlastnost je to `OWLObjectProperty`, pro datotypovou vlastnost `OWLDatatypeProperty` a pro individuum je vytvořena instance třídy `OWLIndividual`.

Jak bylo uvedeno knihovna PHPOWL umožňovala zjistit a ukládat, jaké třídy, vlastnosti a individua se v dané ontologii nacházejí. Tento způsob je nedostačující, jelikož pro plnění ontologie respektive generování formulářů, o kterých se zmíníme v další podkapitole, jsou potřeba vztahy mezi třídami a vlastnostmi. Proto musela být knihovna nejen o tuto činnost rozšířena.

Obecně vztahy definují značky `rdfs:domain` a `rdfs:range`, které jsou zanořeny ve značkách `owl:ObjectProperty`, `owl:DatatypeProperty` nebo `rdf:Property`. Danou vlastnost má objekt umístěný v `rdfs:domain` s objektem v `rdfs:range`. Zpracování těchto značek je řešeno ve třídě `OWLTag`, jelikož je potřeba uchovávat jisté sezení v rámci jedné třídy a umístění těchto značek do zvláštních tříd by bylo problematické pro zpracování. Pro uchování vazeb je definováno speciální pole `_classProperties` umístěné ve třídě `OWLOntology`. Pokud přijmeme značku `rdfs:domain`, tak víme, že otec této značky nese informaci o vlastnosti. K rodičovské značce lze jednoduše přistoupit z aktuální značky pomocí `$this->currentSuperTag`. Pokud jsme v minulosti ještě tuto vlastnost nedefinovali, vložíme ji do

připraveného speciálního pole jako index a doménu vložíme jako jednu z hodnot. Jelikož danou vlastnost může mít více tříd, hodnotou dané vlastnosti je pole domén. Hodnotu domény je potřeba uchovat pro zpracování dalších značek, jelikož se dokážeme pohybovat pouze ve směru syn-rodič, nikoliv syn-syn, a tudíž by nebylo možné uložit hodnoty dalších zpracovávaných značek k dané doméně. Pokud zpracováváme značku `rdfs:range`, hodnotu atributu `rdf:resource` uložíme pod index `from` k dané vlastnosti a doméně v případě, že se jedná o objektovou vlastnost. Pokud se jedná o datotypovou vlastnost, uložíme hodnotu pod index `type`, který určuje datový typ vlastnosti. Další značky určující například omezení pro danou vlastnost jsou uloženy taktéž k dané vlastnosti a doméně. Obrázek 5.5 ukazuje obsah pole `_classProperties`, které obsahuje objektovou vlastnost `studuje` mezi třídami `Student` a `Fakulta` a datotypovou vlastnost `jmeno` typu `string` u třídy `Osoba`.

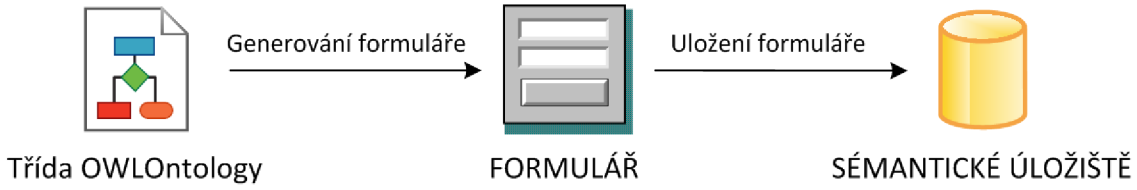
```
["http://stud.fit.vutbr.cz/~xcekan00/DP/school.owl#studuje"]=>
  array(1) {
    ["http://stud.fit.vutbr.cz/~xcekan00/DP/school.owl#Student"]=>
    array(2) {
      ["mincardinality"]=>
      string(1) "1"
      ["from"]=>
      string(56)
"http://stud.fit.vutbr.cz/~xcekan00/DP/school.owl#Fakulta"
    }
  }
["http://stud.fit.vutbr.cz/~xcekan00/DP/school.owl#jmeno"]=>
  array(1) {
    ["http://stud.fit.vutbr.cz/~xcekan00/DP/school.owl#Osoba"]=>
    array(1) {
      ["type"]=>
      string(39) "http://www.w3.org/2001/XMLSchema#string"
    }
  }
```

Obrázek 5.5: Ukázka obsahu pole `_classProperties` s objektovou a datotypovou vlastností.

Po zpracování celé ontologie není výsledný obsah pole `_classProperties` konečný. Není v něm totiž zohledněna dědičnost tříd, a tedy i vlastností. Mějme například třídu `Student`, která je podtřídou třídy `Osoba`. Třída `Osoba` má vlastnost `jmeno`. Vlastnost `jmeno` takto definované ontologie by v našem schématu byla definována pouze pro třídu `Osoba`, i když by měla být definována rovněž pro třídu `Student`. Toto chování je nesprávné, proto je potřeba provést po zpracování ontologie korelaci, která k vlastnostem přidá další třídy, které na základě dědičnosti mají také tuto vlastnost. Korelaci provádí metoda `inferClassProperties()` obsažená v třídě `OWLReader`. Princip korelace je následující. V poli `_classProperties` jsou uloženy vlastnosti a třídy, které mají danou vlastnost. Postupným procházením podtříd těchto tříd zjistíme, jaké další třídy mají mít stejnou vlastnost. Tyto třídy přidáme do původního pole. Rekurentním voláním takto projdeme celý strom podtříd. Po provedení této metody máme v poli kompletní seznam všech tříd, které na základě dědičnosti mají rovněž stejnou vlastnost.

5.3.4 Plnění ontologie

Následujícím krokem po zpracování ontologie je plnění obsahu v závislosti na její struktuře, tedy vytváření objektů tříd, vlastností a vazeb. Tato činnost zahrnuje generování formulářů z uložené reprezentace ontologie ve třídě `OWLOntology` v našem systému. Vygenerované formuláře vložené do struktury webové stránky pak mohou být naplněny uživatelskými daty a následně po odeslání jsou jednotlivé prvky formuláře uloženy do sémantického úložiště k pozdějšímu zpracování, viz kapitola 5.3.5 Šablonování. Činnosti, které je potřeba vykonat, aby bylo možno naplnit ontologii, jsou znázorněny na obrázku 5.6. Generování a ukládání formulářů probíhá opět v presenteru `OntologyPresenter.php`.



Obrázek 5.6: Princip plnění ontologie – generování a uložení formuláře.

5.3.4.1 Generování formulářů

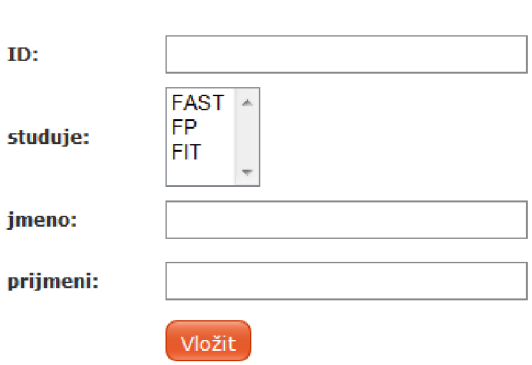
Formuláře se generují pro každou třídu v ontologii. Uživatel vybere jednu z možných tříd a do struktury webové stránky je vygenerován formulář následujícím způsobem.

Pro generování formulářů se používá třída `BaseForm`, kterou jsme popisovali v kapitole 5.3.1.5. Jelikož známe jednu konkrétní třídu, pro kterou chceme generovat formulář, a známe vlastnosti, které daná třída má, tak začneme procházet pole `_classProperties`. Jsou procházeny všechny vlastnosti a u každé zjišťujeme, zda její hodnota obsahuje třídu, kterou hledáme. V případě, že ano, pokusíme se v detailu vlastnosti třídy vyhledat index `type`. Pokud tento index existuje, jedná se o datotypovou vlastnost, v opačném případě bude obsažen index `from`, který značí, že se jedná o objektovou vlastnost. Datotypové vlastnosti definujeme ve formuláři jako textové pole `<input>` a vkládáme je pomocí metody `addText()`. Datotypové vlastnosti nevyžadují vkládání žádného obsahu, tedy alespoň u přidávání nového objektu, jelikož obsah vkládá sám uživatel. U objektových vlastností je situace poněkud odlišnější. Objektové vlastnosti váží námi vytvářenou třídu s nějakou již existující instancí zpravidla jiné třídy. Z indexu `from` zjistíme, s jakou odkazovanou třídou má být námi tvořená třída ve vztahu. Nyní se provede dotaz `SELECT` do sémantického úložiště pomocí jazyka SPARQL pro získání individuí odkazované třídy. V klauzuli `WHERE` pomocí znaku „`a`“ určíme, že chceme všechny identifikátory individuí dané třídy, které se naváží do pole `?id`. Musí se rovněž zohlednit princip dědičnosti a zahrnout do dotazu i individua všech podtříd odkazované třídy. Na obrázku 5.7 je ukázán příklad SPARQL dotazu, který vrátí identifikátory všech uložených individuí třídy `Fakulta`.

```
SELECT ?id
        FROM <C:/wamp/www/owk/app/modules/Front/Ontologies/1.owl>
        WHERE {
          ?id a <http://stud.fit.vutbr.cz/~xcekan00/DP/school.owl#Fakulta>.
        }
```

Obrázek 5.7: Příklad SPARQL dotazu.

Objektové vlastnosti definujeme ve formuláři jako více výběrovou nabídku `<select>` a vkládáme je do struktury webové stránky pomocí metody `addMultiSelect()`. Tato metoda vyžaduje předání všech hodnot v poli. Výsledky ze sémantického úložiště je tedy potřeba ještě upravit, aby byly v poli pouze indexy a jejich hodnoty, které se pak zobrazí v nabídce. Nakonec se do výsledného formuláře vloží textové pole `ID` pomocí metody `addInput()` a tlačítko na odeslání metodou `addSend()` a formulář je vypsán uživateli na webovou stránku. Textové pole `ID` představuje jedinečný identifikátor objektu v celé ontologii. Hodnotu `ID` zadává rovněž uživatel a tato hodnota se používá pro výběr objektu, například ve více výběrové nabídce.




```
$form = new BaseForm;
$form->addInput('id','ID:');
$form->addMultiSelect('studuje',
  'studuje:',array('FAST'=>'FAST',
                   'FP'=>'FP',
                   'FIT'=>'FIT')
);
$form->addText('jmeno','jmeno:');
$form->addText('prijmeni','prijmeni:');
$form->addSend('send','Vložit');
```

Obrázek 5.8: Příklad vygenerovaného formuláře (vlevo) a odpovídající zdrojový kód (vpravo).

Na obrázku 5.8 je ukázán příklad vygenerovaného formuláře pro třídu `Student`, která obsahuje dvě datotypové vlastnosti `jmeno` a `prijmeni` a jednu objektovou vlastnost `studuje`, kde lze vybírat objekty třídy `Fakulta`. V pravé části obrázku je pak ukázán odpovídající kód, který tento formulář generuje.

5.3.4.2 Uložení formuláře

Vyplněný formulář je po jeho odeslání uložen do sémantického úložiště. Jedinou vyžadovanou hodnotou, kterou musí uživatel ve formuláři vždy zadat, je identifikátor `ID`. O zpracování formuláře se stará metoda `formProcess()`. Ze získaných formulářových dat je vytvořen SPARQL dotaz. V první části je potřeba ověřit, že zadaný identifikátor se již v databázi nevyskytuje. Vytvoříme proto SPARQL dotaz `SELECT`, obdobný dotazu na obrázku 5.7, kde v klauzuli `WHERE` umístíme dotaz `<$id> a ?id`, kde proměnná `$id` obsahuje identifikátor, který hledáme v databázi. Tento celý dotaz nám vrátí v poli `?id` třídu, do které patří hledaný identifikátor, pokud v databázi existuje, v opačném případě získáme prázdné pole. Po tomto ověření již nic nebrání vytvoření samotného dotazu `INSERT INTO` pro vložení záznamů do databáze.

SPARQL dotazy představují trojice. V rámci SPARQL dotazu je trojice definována jako `<$id> <$vlastnost> $hodnota`. Proměnná `$id` představuje identifikátor vytvářeného individua. Vlastnost `$vlastnost` představuje identifikátor vlastnosti, kterou vkládáme do databáze. Identifikátory vlastností získáme z polí formuláře z atributů `name`. Proměnná `$hodnota` nese informaci o hodnotě, kterou vkládáme. Hodnotou může být identifikátor v případě objektové vlastnosti, nebo jakýkoliv jiný řetězec v případě datotypové vlastnosti. Tímto způsobem vkládáme vlastnosti a jejich hodnoty individuím. Do sémantického úložiště potřebujeme uložit ještě informaci o třídě daného individua. Tuto informaci vložíme tak, že jako vlastnost zadáme „a“ a za hodnotu dosadíme identifikátor třídy. Na obrázku 5.9 je ukázán SPARQL dotaz `INSERT` pro formulář ukázaný a popsaný na obrázku 5.8.

```
INSERT INTO <C:/wamp/www/owk/app/modules/Front/Ontologies/1.owl>
{
  <http://stud.fit.vutbr.cz/~xcekan00/DP/school.owl#xlogin00> a
     <http://stud.fit.vutbr.cz/~xcekan00/DP/school.owl#Student>.
  <http://stud.fit.vutbr.cz/~xcekan00/DP/school.owl#xlogin00>
     <http://stud.fit.vutbr.cz/~xcekan00/DP/school.owl#studuje> 'FIT'.
  <http://stud.fit.vutbr.cz/~xcekan00/DP/school.owl#xlogin00>
     <http://stud.fit.vutbr.cz/~xcekan00/DP/school.owl#studuje> 'FP'.
  <http://stud.fit.vutbr.cz/~xcekan00/DP/school.owl#xlogin00>
     <http://stud.fit.vutbr.cz/~xcekan00/DP/school.owl#jmeno> 'Jan'.
  <http://stud.fit.vutbr.cz/~xcekan00/DP/school.owl#xlogin00>
     <http://stud.fit.vutbr.cz/~xcekan00/DP/school.owl#prijmeni> 'Novák'.
}
```

Obrázek 5.9: Příklad SPARQL dotazu `INSERT`.

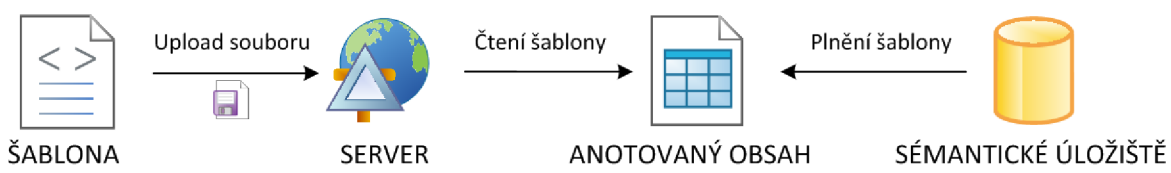
V případě úpravy uloženého individua je použit stejný formulář, který je inicializován hodnotami ze sémantického úložiště pomocí metody formuláře `setDefaults()`. Knihovna ARC2 nepodporuje dotazy typu `UPDATE` případně `MODIFY` pro úpravu záznamů v sémantickém úložišti [21], proto jsou při uložení změněného formuláře nejprve vymazány veškeré záznamy týkající se upravovaného individua SPARQL dotazem `DELETE` a následně jsou nové záznamy vloženy dotazem `INSERT`.

5.3.5 Šablonování

Šablonování je proces vytváření částí HTML kódu, takzvaných šablon, které obsahují speciální značky nebo fráze, které se nahrazují jiným obsahem. Dalo by se říci, že se jedná o mustr, který je použit jednou či vícekrát na stránce, a jeho části jsou nahrazovány konkrétním obsahem například z databáze.

Nette Framework využívá vlastní šablonovací systém, který nese označení Latte [22]. Využívá tzv. makra, která umožňují psát části kódu, syntakticky podobnému jazyku PHP, přímo do šablony, který je při průchodu knihovnou převeden na plnohodnotný PHP kód a následně je zpracován. Makra jsou umisťována ve složených závorkách `{ }`. Každá stránka v naší aplikaci je tvořena vlastní šablonou umístěnou v adresáři `/app/modules/Front/Templates`. Komponenty, například komponenta pro registraci uživatelů, mají šablonu umístěnou v adresáři komponenty ve složce `/app/modules/Front/Components`. Výchozí šablonou celého systému je šablona `@layout.latte`, která tvoří základní strukturu stránek. Do této šablony je dle jednotlivých stránek aplikace vkládán jejich obsah, tj. obsah jiných šablon. Tento obsah je v šabloně nahrazen namísto makra `{include #content}`.

Po naplnění ontologie máme tedy v sémantickém úložišti uložena individua a vlastnosti konkrétních tříd. Tento uložený obsah je potřeba anotovaně prezentovat. K tomu, abychom věděli, co chce uživatel prezentovat, je potřeba k ontologii dodat šablony. Jak ukazuje obrázek 5.10, šablony se uloží na aplikační server, kde proběhne naplnění šablon daty ze sémantického úložiště, viz dále.



Obrázek 5.10: Princip šablonování – import šablony na server, čtení a plnění šablony.

5.3.5.1 Definice šablony

Pro každou ontologii, kterou uživatel vkládá do systému, je potřeba rovněž definovat šablony. Definice šablony označuje třídu popřípadě třídy a jejich vlastnosti, které chceme vypsat. Pro definici těchto údajů se využívají zmiňovaná makra. Definování šablony vyžaduje znalost ontologie. Šablony musí být typu `.latte`. Nette standardně tzv. escapuje proměnné a výrazy, proto je nutné před proměnnými nesoucími anotované vlastnosti uvádět znak vykřičníku `!`.

Jakou třídu chceme v šabloně vypsat nám určuje index pole `$class`. Pokud chceme například získat hodnoty pro třídu `Student`, v kódu šablony se musí objevit odkaz na vícerozměrné pole `$class` s indexem `Student`, tedy `$class['Student']`. `$class['Student']` je rovněž pole, na jehož indexech se nachází jednotliví studenti, kteří kdy byli do systému vloženi. Indexy každého studenta představují vlastnosti dané třídy. Pokud máme třídu `Student` s datotypovou vlastností `jmeno`, tak jméno prvního studenta bychom vypsali pomocí `{!$class['Student'][0]['jmeno']}`. Vypsání jmen všech studentů by pak bylo možno pomocí makra `foreach` následujícím způsobem:

```
{foreach $class['Student'] as $item}
    <div {$item['_owkinfo_']}>
        {!$item['jmeno']}<br />
    </div>
{/foreach}
```

Tímto způsobem lze vypsat datotypové vlastnosti objektů dané třídy. Z uvedeného příkladu vysvětlíme ještě jeden důležitý index `_owkinfo_`, který má každý objekt. Tento index definuje dodatečné atributy, které seskupují vlastnosti daného objektu do jednoho celku, a slouží pro anotaci obsahu.

Objektové vlastnosti oproti datotypovým mají částečně odlišnou strukturu v šabloně. Jelikož může být uloženo více objektů pro jednu objektovou vlastnost, tak objektové vlastnosti nejsou uloženy v řetězci přímo na daném indexu vlastnosti jako je tomu u datotypových vlastností, ale jsou uloženy v poli. Pokud bychom měli u třídy `Student` objektovou vlastnost `studuje`, tak veškeré identifikátory objektů této vlastnosti u prvního studenta bychom vypsali následujícím způsobem:

```
{foreach $class['Student'][0]['studuje'] as $val}
    {!$val}<br />
{/foreach}
```

Jelikož pro identifikaci třídy používáme indexy pole `$class`, tak je zřejmé, že jedna šablona není omezena pouze na jednu konkrétní třídu, ale je univerzální a lze v ní vypsat libovolný počet libovolných tříd dané ontologie.

Díky takto navržené struktuře a makrům, která nám Nette nabízí, můžeme jednoduše omezit vypsání záznamů, které splňují nebo nesplňují definovanou podmínku. Podmínky se v šablonách definují pomocí maker `{if}{/if}`. Pokud bychom chtěli například vypsat jména všech studentů, kteří studují na FIT v ontologii z předešlých příkladů, zápis by vypadal takto:

```
{foreach $class['Student'] as $item}
    {if array_key_exists("FIT",$item['studuje'])}
        {!$item['jmeno']}<br />
    {/if}
{/foreach}
```

Poslední možností definice šablony ontologie je možnost odkázat se na jinou šablonu. Možnost odkazu je užitečná především v případech, kdy vypisujeme všechny objekty dané třídy a jejich všechny vlastnosti by nebylo možné přehledně zobrazit. Takto vypíšeme pouze klíčové vlastnosti, co nás zajímají, a zbylé definujeme ve speciální šabloně, která zobrazí detaily jednoho vybraného objektu. Název šablony, na kterou se budeme z jiné šablony odkazovat, musí začínat řetězcem `detail_`. Způsob definice detailní šablony je shodný s předešlým výkladem, pouze odpadá pole s jednotlivými záznamy, a k samotným vlastnostem se přistupuje přímo z pole `$class['Student']`, tedy například `$class['Student']['jmeno']`.

V šabloně, která bude obsahovat odkaz na detailní šablonu, je potřeba provést definici odkazu. Abychom věděli, na kterou šablonu se máme odkázat, je potřeba tuto informaci v šabloně specifikovat. To se provádí v komentáři umístěném zpravidla v horní části šablony. Komentář se v šabloně zapisuje ve složených závorkách s hvězdičkou `{* komentář *}` a uvnitř musí být obsažen řetězec začínající `link: název_šablony_bez_koncovky = třída;`. Pokud bychom tedy definovali odkaz na šablonu `detail_studenta.latte` takovýmto způsobem `{* link: detail_studenta = Student; *}`, tak odkaz na konkrétní objekt pak získáme v poli v indexu `{$class['Student'][0]['link_detail_studenta']}`.

Poslední možností šablony je vypsání jmenného prostoru dané ontologie pomocí definované proměnné `$ns`.

5.3.5.2 Import šablony

Pro importování šablony je vytvořen speciální formulář, který se nachází pod seznamem dostupných tříd v detailním zobrazení vybrané ontologie. Pro import je nutné zadat název stránky, pod kterým se bude daná šablona zobrazovat v menu ontologie respektive šablon na úvodní stránce aplikace, a vybrat soubor se šablonou z lokálního disku uživatelského počítače, který musí být typu `.latte`. Ostatní typy nejsou akceptovány a import tudíž nelze provést. Stisknutím tlačítka nahrát dojde k přenosu souboru na server následujícím způsobem. Dojde k přejmenování názvu souboru. Nový název je složen z identifikátoru ontologie, pod kterou je daná šablona importována, a původního názvu šablony odděleného podtržítkem. Tímto způsobem lze zajistit jedinečné názvy souborů se šablonami napříč ontologiemi. Zároveň se do databázové tabulky pro šablony vloží záznam, který obsahuje identifikátor šablony, název stránky, identifikátor ontologie a původní název souboru. Soubor se šablonou je poté uložen do adresáře vytvořeného speciálně pro šablony, který má umístění `/app/modules/Front/Ontologies/Templates`.

5.3.5.3 Plnění šablony

Před plněním šablony je nutno provést její analýzu. Musíme nejprve určit, o jaký typ šablony se jedná. Může to být běžná šablona, nebo se může jednat o zmiňovanou detailní šablonu. ID šablony zjistíme z parametru `id` URL adresy stránky. Pomocí identifikátoru vyhledáme záznam v databázi a z názvu souboru zjistíme, o jaký typ šablony se jedná. Co musíme dále ze šablony zjistit, jsou třídy, jejichž objekty je potřeba získat ze sémantického úložiště. Tím, že zjišťujeme, které třídy jsou v šabloně obsaženy, zajišťujeme to, že nebudou zpracovány všechny třídy v ontologii, ale pouze ty, které jsou potřeba. Pokud se jedná o detailní šablonu, v parametru `detail` URL adresy stránky je uložen identifikátor konkrétního objektu v sémantickém úložišti, který máme získat. Pokud se nejedná o detailní šablonu, podíváme se, zda existují definice odkazů na detaily šablon, a tuto informaci si uložíme do pole. Tímto jsme získali všechny potřebné informace ze šablony a můžeme přejít k samotnému zjištění individuí.

Pro každou třídu, kterou jsme identifikovali v šabloně, zjistíme, zda je třída platná, a tedy zda existuje v dané ontologii. Metodou `getPropertiesOfClass()` zjistíme všechny vlastnosti ontologie pro danou třídu. Ze zjištěných vlastností vytvoříme trojice popisované v kapitole 5.3.4.2 a pomocí SPARQL dotazu `SELECT` vybereme veškeré uložené objekty a jejich vlastnosti ze sémantického úložiště. Dalším krokem je provedení metody `unique_array()` nad získaným výsledkem z databáze, který seskupí získané záznamy dle principu popsaného v kapitole 5.3.1.5. Jelikož objektové vlastnosti máme v šabloně definované v poli, je potřeba provést další korelaci získaných výsledků. V případě, že uložený objekt obsahoval více hodnot pro danou objektovou vlastnost, je výsledek objektové vlastnosti po volání metody `unique_array()` správně zařazen v poli. Pokud ale objekt obsahoval jen jednu hodnotu pro danou objektovou vlastnost, výsledek objektové vlastnosti není v poli, proto je potřeba získaný výsledek volání metody `unique_array()` předat metodě `objectPropertyToArray()`, která již zajistí požadovaný formát výstupu. Dalším krokem je anotace obsahu. O anotaci více v následující podkapitole 5.3.5.4. V případě odkazu v šabloně je pro každé individuum vložen index `link_název_šablony` s odkazem na detailní šablonu. Výsledný výstupní formát je předán do šablony pomocí příkazu `$this->template->class[$className]`, kde `$className` určuje jednu z identifikovaných tříd.

5.3.5.4 Anotace obsahu

Anotace obsahu je prováděna dle RDFa dokumentace organizace W3C [11]. Anotace přidává dodatečné atributy ke značkám zapouzdřujícím určitou vlastnost získanou ze sémantického úložiště. Každý objekt má ve svém poli obsahujícím především vlastnosti rovněž index `_owkinfo_`. Tento index obsahuje definici atributů `about` a `typeof`. Atribut `about` nese identifikátor daného objektu a zapouzdřuje tak vlastnosti objektu do jednoho celku. Atribut `typeof` definuje typ objektu, tedy nese informaci o jeho třídě.

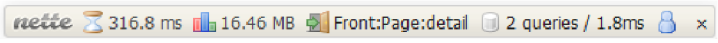
Anotace se provádí před zasláním definovaného pole do šablony. Je procházen každý objekt, kterému je přidán index `_owkinfo_` s popsanými atributy. Rovněž je procházena každá vlastnost objektu a je anotována následujícím způsobem. Hodnota vlastnosti je obalena značkou řádkového prvku `<span>`, kterému je přidán RDFa atribut `property` určující vlastnost objektu a obsahující URI dané vlastnosti. Takto je anotován obsah, je tedy znám objekt svým identifikátorem, svou třídou a svými vlastnosti.

6 Testování

Testování a ladění probíhalo již během vývoje webové aplikace. Některé syntaktické chyby byly odhaleny již v editoru NetBeans, jiné odhalil samotný interpret jazyk PHP při zpracování kódu a zbylé syntaktické a některé sémantické chyby dokázal odhalit Nette Framework. Nette umožňuje zapnout tzv. laděnku, která dokáže vizualizovat chyby. Ukáže přesné místo v kódu, kde chyba vznikla, jaké byly v tu danou chvíli obsahy proměnných, parametrů apod. Laděnka je ukázána na obrázku 6.1, kde odchytila chybu nedefinovaného indexu pole.

Obrázek 6.1: Laděnka knihovny Nette.

Pro ladění systému je také vhodným pomocníkem tzv. debugger bar, který je rovněž součástí Nette. Zobrazuje se v pravém dolním rohu stránky a zobrazuje informace o době provádění skriptu, kolik skript spotřeboval paměti, nebo například počet dotazů do databáze a to včetně konkrétních odeslaných a získaných hodnot. Debugger bar je ukázán na obrázku 6.2.



Obrázek 6.2: Debugger bar knihovny Nette.

Pro testování vizuálního vzhledu aplikace jsme použili čtyři nejpoužívanější prohlížeče. Stránky byly testovány v prohlížeči Mozilla Firefox verze 20.0.0.1, Internet Explorer verze 7, 8 a 9, Google Chrome verze 26.0.1410.64 a Opera verze 12.12. V těchto prohlížečích se stránky zobrazily korektně dle definovaného grafického návrhu.

Pro zjištění obsahu polí byla použita PHP funkce `var_dump()`, která dokáže vizualizovat obsah zvoleného pole včetně jeho indexů. Testování knihovny PHPOWL jsme prováděli na námi vytvořených ontologiích pomocí implementovaného debuggeru knihovny, který zobrazuje postup zpracování ontologie včetně metod, které jsou volány, tříd, které danou značku obsluhují, apod.

V případě chybně zadaného uživatelského vstupu je tato činnost rozpoznána respektive odchycena pomocí konstrukce `try { } catch { }` a uživateli je zobrazena chybová zpráva pomocí metody presenteru `error()`.

Díky testování jsme odhalili chyby systému již během vývoje aplikace a tyto nedostatky mohly být proto v raném stádiu odchyceny a nepronikly tak do dalších částí implementace.

7 Demonstrační aplikace

V této části je popsán způsob instalace a vlastní popis demonstrační aplikace. Pro chod webové aplikace je vyžadován interpret jazyka PHP minimálně verze 5.3 a databázový server MySQL minimálně verze 5. Pro samotný běh Nette Frameworku je potřeba splnit požadavky v nastavení PHP a Apache serveru, které lze nalézt na adrese: http://doc.nette.org/cs/requirements.

7.1 Instalace aplikace

Na přiloženém CD jsou k dispozici zdrojové soubory aplikace. Pro instalaci je potřeba celý adresář `/owk` nacházející se v hlavní složce CD přesunout na webový server. Rovněž se v hlavním adresáři CD nachází soubory `owk_database.sql` a `owk_storage_database.sql`, které slouží pro vytvoření a naplnění tabulek MySQL databáze. Tabulky lze vytvořit v jedné databázi, ale v rámci přehlednosti doporučujeme použít dvě databáze, pokud je to možné. Po provedení této operace je nutné nakonfigurovat připojení naší aplikace a knihovny ARC2 do databáze. Připojení naší aplikace se nastavuje v konfiguračním souboru `/owk/app/base.conf.neon`. Připojení knihovny ARC2 do databáze se nastavuje v souboru `/owk/index.php`. Po těchto krocích je vše připraveno pro vyzkoušení aplikace zadáním adresy `http://adresa_serveru/owk/` do webového prohlížeče.

7.2 Popis aplikace

Samotná webová aplikace se skládá z několika stránek. Po navštívení webové prezentace je na úvodní stránce možnost přihlášení uživatele. Horní menu obsahuje dva odkazy, odkaz na hlavní stránku a možnost registrace. V levém menu označovaném jako menu ontologie je možno procházet šablony, respektive anotovaná data načtená z obsahu naplněné výchozí ontologie. Kliknutím na jednu z možných šablon se zobrazí anotovaný obsah v závislosti na definici šablony. Ukázková šablona zobrazuje obsah pomocí tabulky.

Na stránce s registrací je možno zaregistrovat nového uživatele zadáním osobních informací a zvolením uživatelského jména a hesla. Po zaregistrování je možno se ihned přihlásit na hlavní stránce. V systému je vytvořen výchozí uživatelský účet s přihlašovacími údaji `admin:admin`.

Po přihlášení se v horním menu objeví nové odkazy na další stránky. Jedná se o odkazy na ontologie, úpravu osobních informací a nastavení serveru. Rovněž je zde možnost odhlášení ze systému.

Zvolením odkazu `Ontologie` přejde uživatel na stránku s dostupnými ontologiemi v systému a rovněž je mu umožněno nahrání nové ontologie. S ontologiemi je možno manipulovat, buď zvolit jednu jako výchozí, zobrazit podrobnosti, nebo smazat ontologii. Kliknutím na odkaz `Podrobnosti` u některé z dostupných ontologií, dojde k přechodu na stránku s detailními informacemi o zvolené ontologii. Na této stránce je vypsán seznam tříd, které v ontologii existují a pro které můžeme tedy vytvářet individua. Ve spodní části stránky je zobrazen seznam šablon dané ontologie s možností přidání nové nebo odebrání existující šablony z ontologie. Zvolením jedné z dostupných tříd, kliknutím na odkaz `Zobrazit třídu`, je načtena další stránka. Na ní jsou zobrazena individua zvolené třídy, která již někdy byla přidána do systému, a jsou uložena v sémantickém úložišti. Na této stránce jsou k dispozici odkazy, pomocí kterých lze buď přidat nový

záznam, nebo upravit popřípadě smazat záznam existující. V případě přidání nebo úpravy záznamu je zobrazen formulář s vlastnostmi dané třídy, který lze vyplnit.

Zvolením odkazu `Můj profil` v hlavním menu aplikace je uživateli zobrazen formulář s uloženými osobními informacemi, které může změnit.

Poslední stránka `Nastavení systému` dostupná pro přihlášeného uživatele slouží pro nastavení webové stránky. Uživatel zde může měnit základní metainformace stránky jako titulek, klíčová slova a popisek stránky, nebo je zde možnost povolení či zakázání registrování uživatelů do tohoto systému.

Náhledy aplikace je možno nalézt v příloze 2 tohoto dokumentu.

8 Závěr

Tato práce se zabývala návrhem a implementací systému pro správu obsahu založeného na ontologiích. Nejprve byl vysvětlen sémantický web, důvod jeho vzniku a jeho koncept, ve kterém jsme se převážně zaměřili na základ sémantického webu, který představuje RDF a OWL. Vysvětlili jsme RDFa, který umožňuje zobrazit sémantická data v prostředí internetových stránek.

V další části jsme se věnovali návrhu systému. Byla definována neformální specifikace samotného systému, od které se pak odvíjely další kroky návrhu. Z neformální specifikace byl systém analyzován a byl vytvořen diagram případů použití. Z analýzy vyplynul návrh systému, kde byli identifikováni jednotliví aktéři a jejich pravomoci. Byla definovaná architektura systému dle návrhového vzoru MVC a byl vytvořen návrh schématu relační databáze. Poslední část návrhu se zaměřuje na sémantické úložiště.

Z vytvořeného návrhu byl systém implementován. Nejprve byly vybrány vhodné webové technologie pro realizaci systému a následně započala implementace. Prvními kroky bylo vytvoření grafického návrhu aplikace, který byl následně převeden do HTML jazyka. Následujícími kroky bylo vytvoření systému založeného na knihovně Nette Framework. V této části byla popsána samotná knihovna a následně principy syntaktického zpracování zadané ontologie. Z načtené ontologie byl vygenerován webový formulář obsahující objektové a datotypové vlastnosti vybrané třídy. Uživatelem naplněný formulář byl uložen do sémantického úložiště ARC2, které je založeno na MySQL databázi. Hodnoty uložené v sémantickém úložišti byly dále použity pro zobrazení anotovaného obsahu vložením do předem připravených šablon, které uživatel dodává společně s definovanou ontologií.

V poslední části práce byly popsány metody testování vytvořené aplikace, vysvětlen způsob instalace aplikace na webový server a celkově popsán výsledný systém.

Implementace tohoto systému přinesla nesmírnou výhodu a to především v anotovaném obsahu a také v univerzálnosti použití. Systém může být použit prakticky na správu čehokoliv, v závislosti na právě používané ontologii. Není problém vést záznamy o studentech, předmětech, vyučujících a jejich vzájemných vztazích nebo například spravovat druhy pečiva včetně jejich surovin, vést záznam o tom, jaká surovina je kde použita a podobně, stačí naimportovat příslušnou ontologii. Uživatel má rovněž možnost díky šablonovacímu systému definovat vlastní rozložení a rozmístění jednotlivých prvků ontologie.

Dalším krokem, kterým by se mohl ubírat vývoj této aplikace, je rozšiřování podpory dalších značek a atributů ontologického jazyka RDF a OWL. Tyto standardy jsou značně rozsáhlé a demonstrační aplikace implementuje podmnožinu těchto standardů. Rozhraní systému je pro budoucí rozšíření navrženo a počítá s tím. Systém je vytvořen v českém jazyce, za zvážení stojí jeho zveřejnění a případně multijazyková verze, která by jistě přivedla širokou řadu příznivců ontologických jazyků a anotovaných dat.

Byl navrhnut a vytvořen systém, který načte ontologii, zpracuje ji, umožní uživateli vytvářet jedince daných tříd ontologie a následně tyto jedince prezentovat na webu pomocí anotovaných informací. Tyto anotovaná data pak mohou být dále zpracovávána strojově a následně použita k dalším účelům.

Literatura

[1] DACONTA, Michael C, Leo Joseph OBRST a Kevin T SMITH. *The Semantic Web: a guide to the future of XML, Web services, and knowledge management*. Indianapolis, Ind.: Wiley Pub., c2003, 281 p. ISBN 04-714-3257-1.

[2] NEEDLEMAN, Rafe. Web 2.0 Expo: What to watch for. *Webware – CNET* [online]. 2008 [cit. 2012-10-07]. Dostupné z: http://news.cnet.com/8301-17939_109-9923360-2.html

[3] World Wide Web Consortium (W3C). *W3C Semantic Web Activity* [online]. 2001 [cit. 2012-10-07]. Dostupné z: http://www.w3.org/2001/12/semweb-fin/w3csw

[4] XML. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2012 [cit. 2012-11-04]. Dostupné z: http://cs.wikipedia.org/wiki/Xml

[5] Hanyáš, P.: *Sémantický web – tutoriál a demonstrační příklady*, bakalářská práce, Brno, FIT VUT v Brně, 2007, 41 s.

[6] World Wide Web Consortium (W3C). *OWL Web Ontology Language Overview* [online]. 2004 [cit. 2012-11-05]. Dostupné z: http://www.w3.org/TR/owl-features/

[7] LACY, Lee W. *OWL: representing information using the web ontology language*. Victoria, B.C: Trafford, 2005. ISBN 14-120-3448-5.

[8] Znalostní technologie I. *KAPITOLA 4.: PŘEDSTAVENÍ JAZYKA OWL - KOMPONENTY JAZYKA OWL* [online]. 2006 [cit. 2012-11-18]. Dostupné z: http://lide.uhk.cz/fim/ucitel/fshusam2/lekarnicky/zt1/zt1_kap04_02.html

[9] BURGET, Radek. *Ontologie a sémantický web* [online]. 2010 [cit. 2013-01-11]. Dostupné z: https://www.fit.vutbr.cz/study/courses/PIS/private/cviceni/semweb2010.pdf

[10] Mikroformát. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2013 [cit. 2013-01-11]. Dostupné z: http://cs.wikipedia.org/wiki/Mikroform%C3%A1t

[11] World Wide Web Consortium (W3C). *RDFa Core 1.1: Syntax and processing rules for embedding RDF through attributes* [online]. 2012 [cit. 2013-01-11]. Dostupné z: http://www.w3.org/TR/rdfa-syntax/

[12] World Wide Web Consortium (W3C). *Gleaning Resource Descriptions from Dialects of Languages (GRDDL)* [online]. 2008 [cit. 2013-01-11]. Dostupné z: http://www.w3.org/2004/01/rdxh/spec

[13] PLOTZ, Marc. Principles Of MVC for PHP Developers. *HTML Goodies: The Ultimate HTML Resource* [online]. 2012 [cit. 2013-01-12]. Dostupné z: http://www.htmlgoodies.com/beyond/php/article.php/3912211/Principles-Of-MVC-for-PHP-Developers.htm

[14] Čížek, P.: *Úložiště znalostí*, diplomová práce, Brno, Fakulta informatiky, Masarykova univerzita, 2007, 50 s.

[15] ULLMAN, Larry. *PHP a MySQL: názorný průvodce tvorbou dynamických WWW stránek.* Vyd. 1. Brno: Computer Press, 2004, 534 s. ISBN 80-251-0063-4.

[16] Nette Framework. *Rychlý a pohodlný vývoj webových aplikací v PHP* [online]. 2008 [cit. 2013-05-08]. Dostupné z: http://nette.org/cs/#toc-features

[17] SourceForge - Download, Develop and Publish Free Open Source Software. *PHPOWL* [online]. 2012 [cit. 2013-05-08]. Dostupné z: http://sourceforge.net/projects/phpowl/

[18] GitHub · Build software better, together. *Home · semsol/arc2 Wiki* [online]. 2011 [cit. 2013-05-08]. Dostupné z: https://github.com/semsol/arc2/wiki

[19] Nette Framework. *Nette\Database vs dibi* [online]. 2012 [cit. 2013-05-08]. Dostupné z: http://pla.nette.org/cs/nette-database-vs-dibi

[20] NetBeans. *Vítejte u NetBeans* [online]. 2011 [cit. 2013-05-08]. Dostupné z: https://netbeans.org/index_cs.html

[21] GitHub · Build software better, together. *Using ARC's RDF Store* [online]. 2011 [cit. 2013-05-12]. Dostupné z: https://github.com/semsol/arc2/wiki

Seznam příloh

Příloha 1: Seznam podporovaných značek a atributů rozšířené knihovny PHPOWL.
Příloha 2: Náhledy obrazovek demonstrační aplikace.
Příloha 3: CD se zdrojovými kódy.

Příloha 1

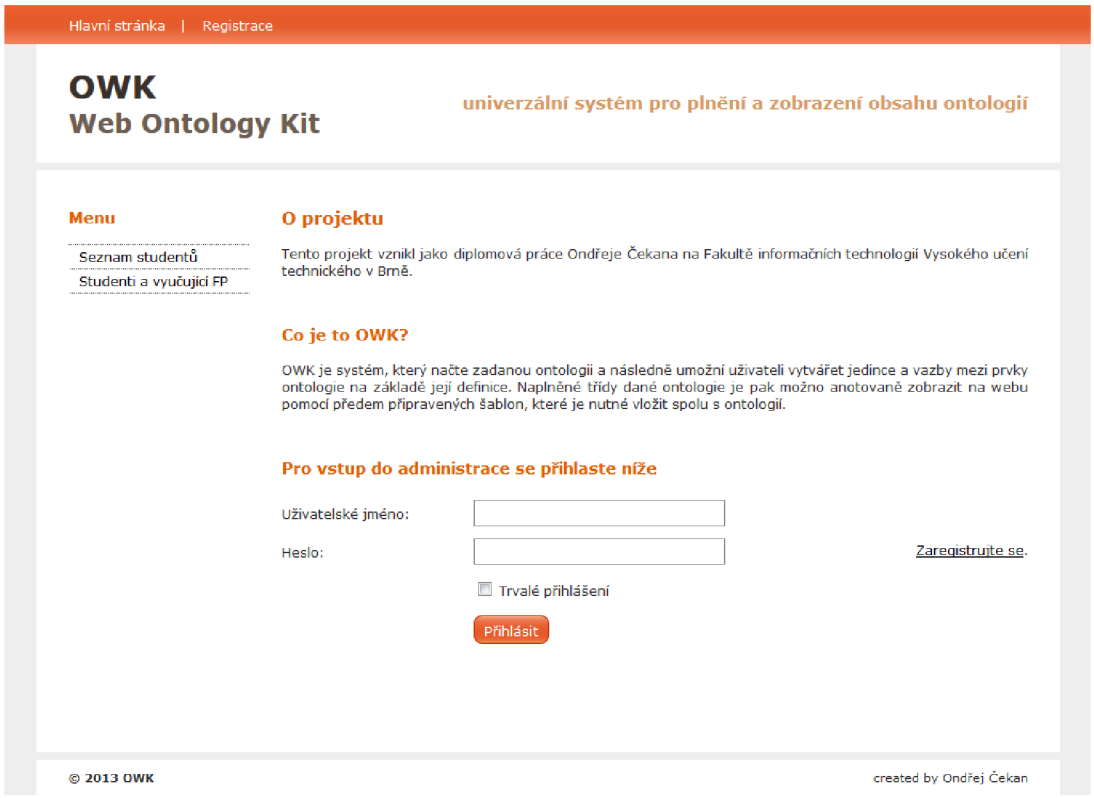
Seznam podporovaných značek a atributů rozšířené knihovny PHPOWL

Značka	Atributy
owl:Ontology	-
owl:Class	rdf:about, rdf:ID, rdfs:label
owl:ObjectProperty	rdf:about, rdf:ID
owl:DatatypeProperty	rdf:about, rdf:ID
rdfs:subClassOf	rdf:resource
rdfs:subPropertyOf	rdf:resource, rdf:about
rdf:RDF	xml:base
rdfs:comment	-
owl:Restriction	-
owl:onProperty	rdf:resource
owl:minCardinality	rdf:datatype
owl:maxCardinality	rdf:datatype
owl:cardinality	rdf:datatype
owl:valuesFrom	rdf:resource
owl:allValuesFrom	rdf:resource
owl:someValuesFrom	rdf:resource
rdfs:domain	rdf:resource
rdfs:label	-
rdfs:range	rdf:resource
rdfs:Class	rdf:about, rdfs:label
rdf:Property	rdf:about, rdfs:label
+ instance tříd a vlastností	

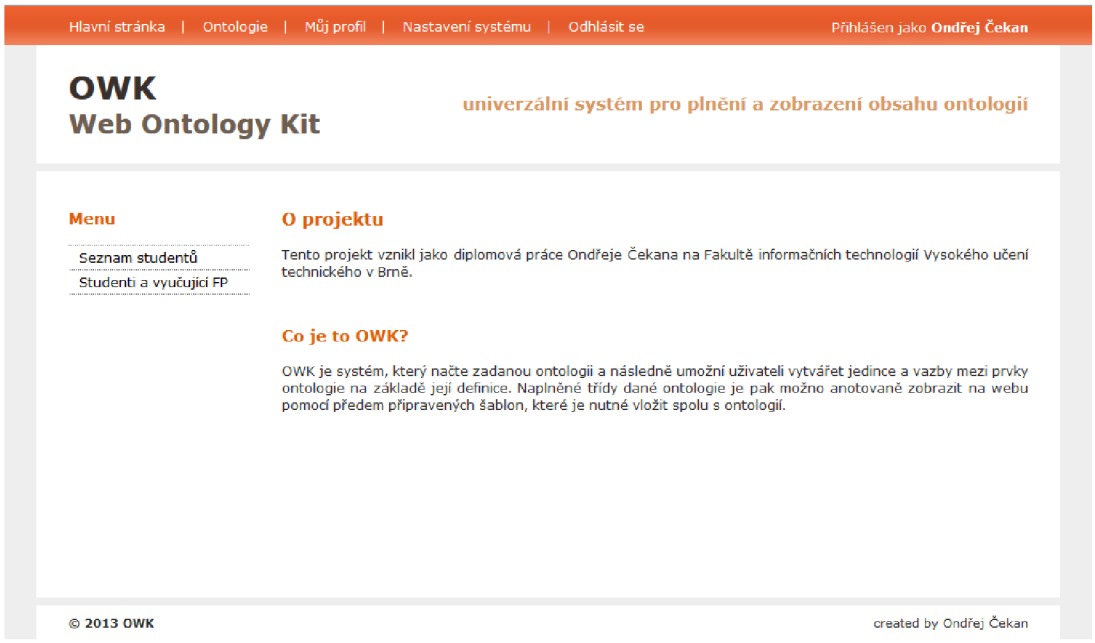
Příloha 2

Náhledy obrazovek demonstrační aplikace

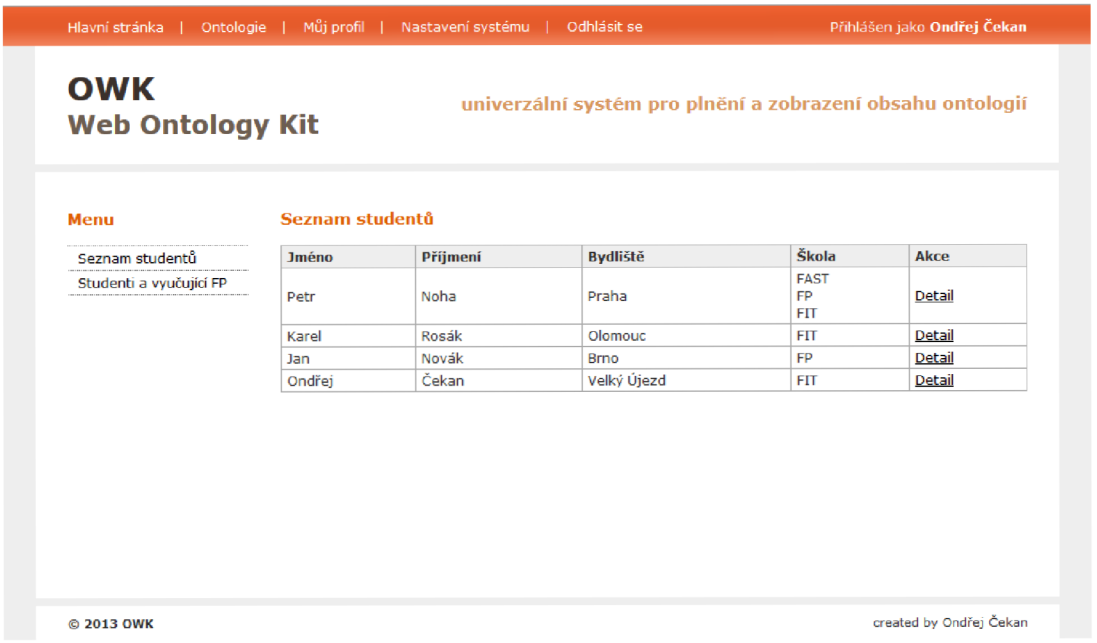
`Hlavní stránka` nepřihlášeného uživatele



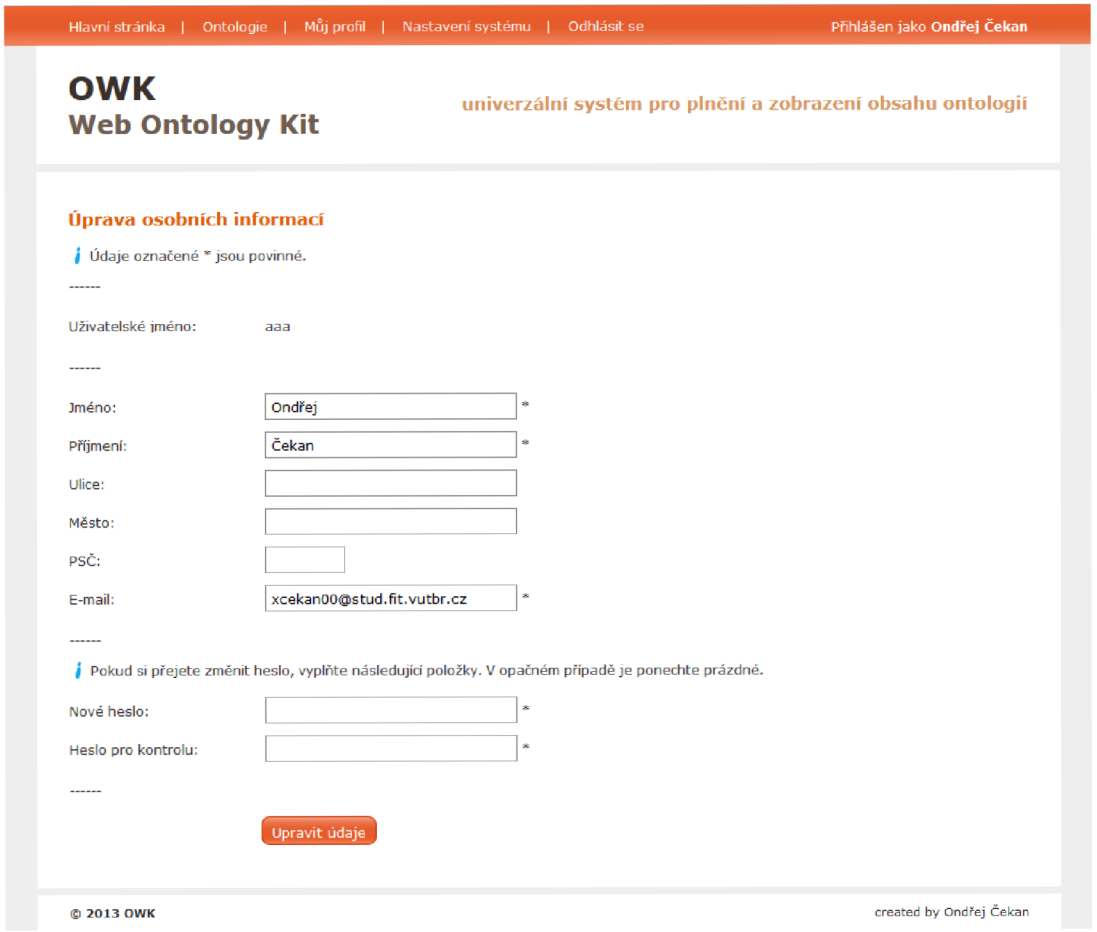
`Hlavní stránka` přihlášeného uživatele



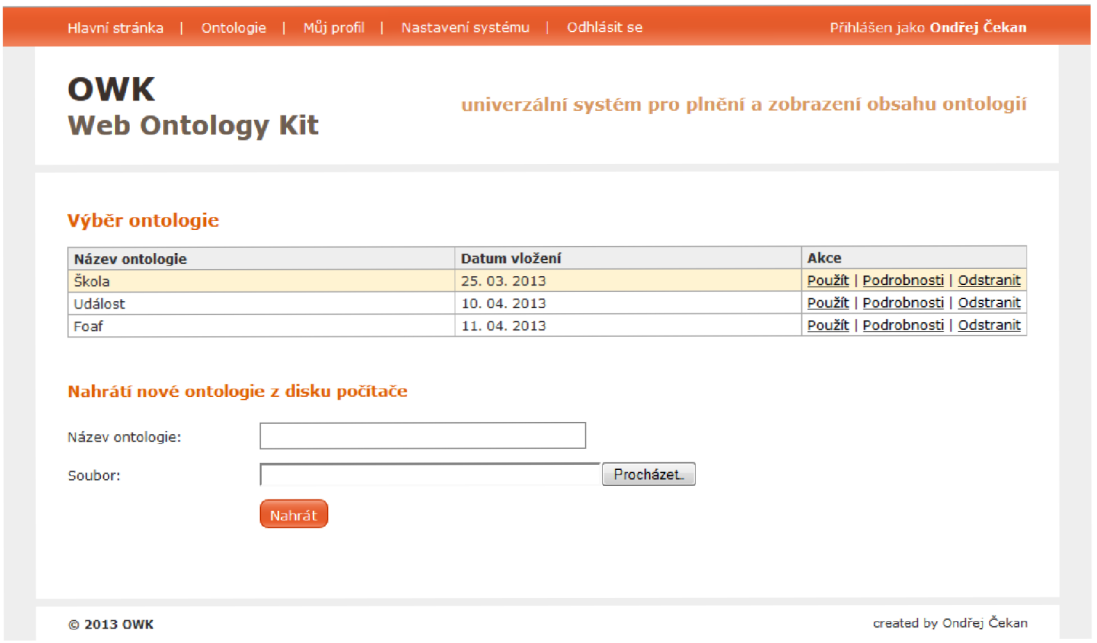
Zobrazení šablony s anotovaným obsahem



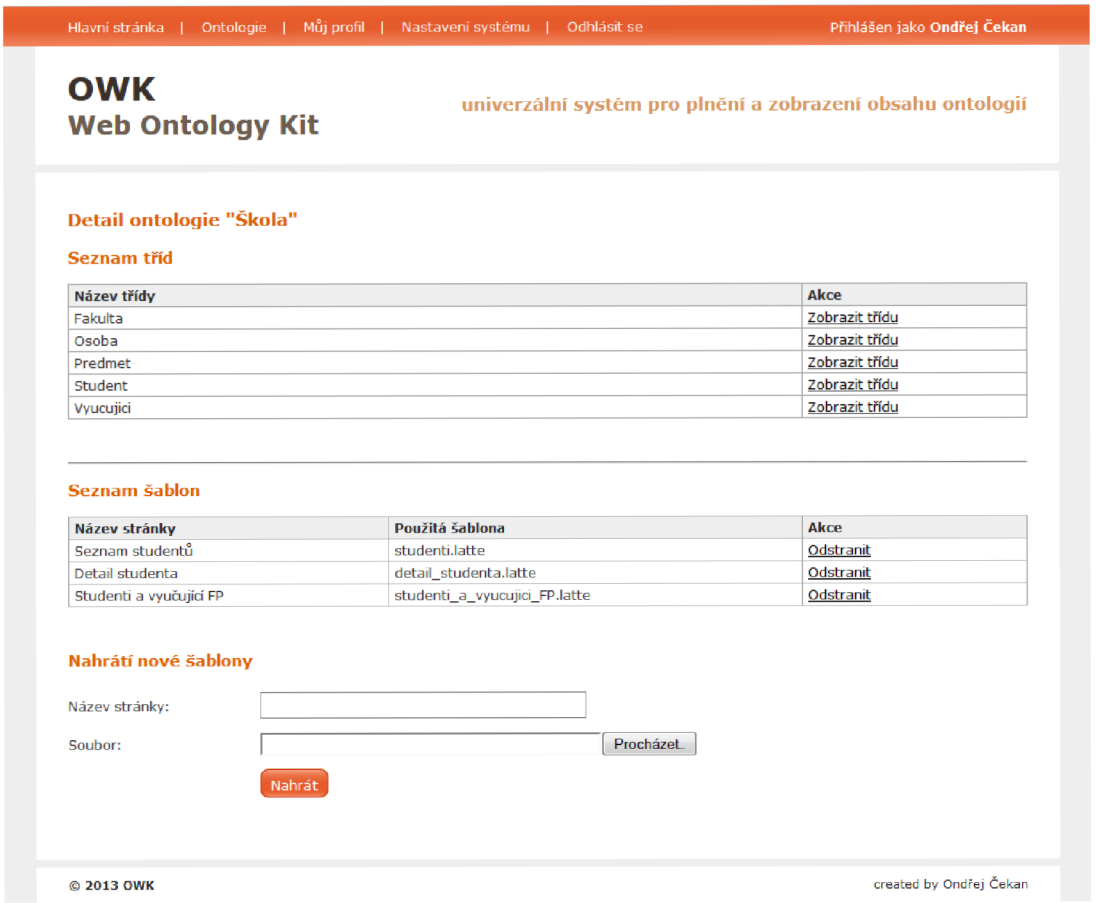
Stránka `Můj profil`



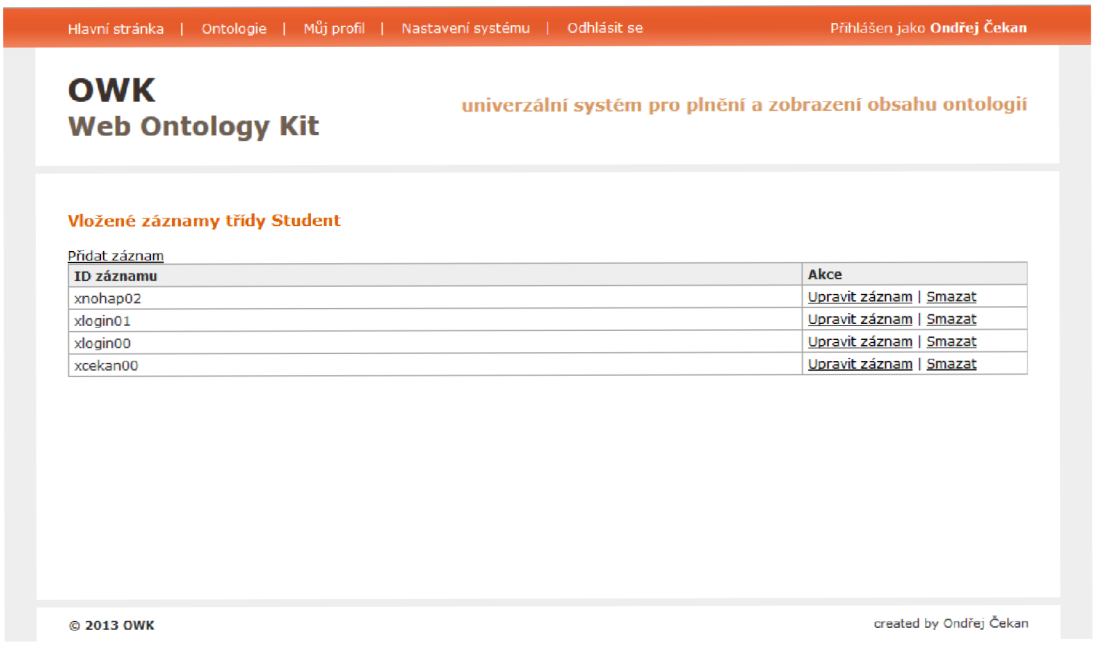
Stránka `Ontologie`



Stránka `Detail ontologie`



Stránka `Vložené záznamy` vybrané třídy



Stránka `Vložení nového záznamu`