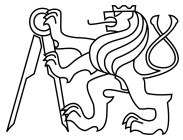
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Systém pro správu multimediálního obsahu založený na ontologiích
Student:	Bc. Jiří Meloun
Vedoucí:	Ing. Martin Balík, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2015/16

Pokyny pro vypracování

1. Analyzujte vlastnosti existujících systémů pro správu multimediálního obsahu (CMS). Nastudujte principy sémantického webu, využití v současných systémech a přínosy pro správu multimediálního obsahu a výměnu dat mezi systémy.

2. Navrhněte a implementujte v jazyce Java webový multimediální CMS založený na ontologiích a frontend pro prezentaci obsahu. Cílem je vytvoření jednoduchého prototypu pro experimentální účely, který bude využívat prvky Sémantického webu. Implementujte prvky pro přizpůsobení prezentace uživateli. K implementaci využijte technologie JSF, Spring, Empire a ASF framework.

3. Výslednou aplikaci otestujte z hlediska funkčnosti, použitelnosti a ověřte adaptivní chování.

Seznam odborné literatury

John Hebeler, Matthew Fisher, Ryan Blace, Andrew Perez-Lopez: Semantic web programming, Wiley Publishing, Inc., Indianapolis, IN, USA, 2009

Evgeny Knutov, Paul De Bra, Mykola Pechenizkiy: AH 12 years later a comprehensive survey of adaptive hypermedia methods and techniques, New Review of Hypermedia and Multimedia, 15(1):5-38, 2009

Sergey Sosnovsky, Darina Dicheva: Ontological technologies for user modelling, International Journal of Metadata, Semantics and Ontologies, 5(1):32-71, 2010

Martin Balík, Ivan Jelínek: Adaptive System Framework: A Way to a Simple Development of Adaptive Hypermedia Systems. ADAPTIVE 2013, 20-25, Valencia, Spain, 2013, IARIA

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof.Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 23. prosince 2014

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Systém pro správu multimediálního obsahu založený na ontologiích

Bc. Jiří Meloun

Vedoucí práce: Ing. Martin Balík, Ph.D.

12. ledna 2016

Poděkování

Tímto bych chtěl poděkovat Ing. Martinu Balíkovi, Ph.D. za trpělivost a konzultace, které se občas protáhly do pozdních hodin. Dále bych chtěl poděkovat své rodině za poskytnutí studijního prostředí během studií.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 12. ledna 2016 .....

Odkaz na tuto práci

Meloun, Jiří. *Systém pro správu multimediálního obsahu založený na ontologiích.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Práce se zabývá návrhem a implementací prototypu systému pro správu multimediálního obsahu (CMS) založeného na ontologiích. Nejprve představuje tradiční způsoby datového modelování s dopady na sdílení informací, dále již poskytuje základní přehled klíčových technologií sémantického webu, jako jsou datový model RDF a ontologie, a na příkladech demonstruje jejich základní principy a použití. Kromě toho také stručně představuje některé charakteristiky ze světa adaptivních systémů. Poté již následuje analýza vlastností vybraných CMS, analýza a návrh samotného systému, mj. zahrnující vymezení doménového modelu a jeho pokrytí pomocí existujících ontologií s vytvořením nových výrazů ve vlastní ontologii. V rámci realizace pak nejprve obecně představuje hlavní použité technologie a jejich základní konfiguraci, a to včetně volby datového úložiště a jeho typu. Dále představuje vybrané části implementace systému a nakonec také uvádí testování implementovaného systému.

Klíčová slova sémantický web, ontologie, SPARQL, RDF, adaptivní systémy, systém pro správu obsahu, multimediální obsah

Abstract

This thesis is focused on design and development of a prototype Ontology-based multimedia content management system (CMS). At first, it introduces traditional methods of data modeling with implications for data sharing, furthermore it provides a basic overview of key Semantic web technologies, such as RDF data model and ontologies, and also illustrates examples of usage of the basic principles. Moreover, it also introduces some characteristics of the world of adaptive systems. Afterwards, the analysis of the characteristics of the selected CMS is introduced, followed by analysis and design of the system itself. Among others, it includes the definition of domain model and its coverage through existing ontologies and newly created expressions in own ontology. Within realisation part, it describes the main used technologies and their basic configuration, including a choice of data repository and its type. Furthermore it presents selected parts of the system implementation and finally, it provides testing of the implemented system.

Keywords Semantic web, ontologies, SPARQL, RDF, adaptive systems, content management system, multimedia content

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Objem nejen multimediálního obsahu se v posledních letech rychle rozrůstá. Stále více interních aplikací se také přesouvá na web, což má za následek ještě rychleji se rozrůstající objem dat na webu. Potencionální hodnota těchto dat je obrovská, avšak (nejen) pro multimediální obsah, kam řadíme především různé kombinace textu, audia, videa nebo obrázků, vzniká velký problém s následným znovupoužitím dat, jejich výměnou nebo získáváním jejich obsahu. Naprosto klíčovou roli hrají metadata, která tyto typy obsahu popisují. V tomto směru, ale i v mnoha dalších, mohou výrazně pomoci technologie sémantického webu.

Cíl práce

Hlavním cílem práce je pokusit se o vytvoření jednoduchého prototypu systému pro správu multimediálního obsahu (CMS) založeného na ontologiích. To znamená, že systém bude operovat s daty uloženými v sémantickém datovém úložišti. Systém bude implementován pomocí technologií Java Server Faces (JSF), Spring, Empire a Adaptive System Framework (ASF) framework. Kromě toho, že systém bude využívat prvky sémantického webu, bude dále zahrnovat i prvky pro přizpůsobení prezentace uživateli.

Motivace autora

Jeden z důvodů zvolení tohoto tématu pro svou závěrečnou práci je ten, že nabízí možnost vyzkoušet si práci se zajímavými technologiemi. Celý projekt zároveň představuje velkou výzvu, kombinují se v něm jak technologie, které jsou ve světě informačních technologií rozšířené, tak technologie, které zatím nemají takovou podporu a mají spíše experimentální charakter. Velkou motivací je také rozšíření svých znalostí v oblasti vývoje webových aplikací na platformě Java. V rámci studií jsem se také již seznámil s tvorbou struktu-

rovaných, sémanticky anotovaných dat. Vždy se však jednalo o jednorázový proces, zatímco v této práci má tato činnost zcela jiný, dynamický charakter.

Struktura práce

Kapitola 1 představuje zejména teoretický úvod do sémantického webu. Popisuje především klíčové pilíře, které formují sémantický web, uvedeny jsou obecné přínosy sémantického webu a na příkladech jsou vysvětleny základní technologické principy. Jelikož prototyp systému má využívat adaptivní prvky, první kapitola také přináší stručný úvod do adaptivních systémů. Samotný závěr kapitoly pak zahrnuje propojení těchto dvou světů.

Kapitola 2 nejprve shrnuje vlastnosti vybraných existujících systémů pro správu multimediálního obsahu spolu s využitím prvků sémantického webu v těchto systémech. Dále již následuje analýza požadavků na vlastní prototyp systému, konceptuální model a jeho pokrytí pomocí existujících ontologií, případně ontologie vlastní. Zbytek kapitoly je věnován návrhu systému.

Kapitola 3 se již zabývá realizací prototypu systému. Úvod kapitoly obecně představí technologie uvedené v zadání této práce, konfiguraci projektu a obecné informace o systému. Zbytek kapitoly se zabývá popisem implementace vybraných částí prototypu systému.

Kapitola 4 je věnována testování implementovaného prototypu systému. Je zde zahrnuto testování funkčnosti implementovaného prototypu a také jeho použitelnosti na základě testování s reálnými uživateli. Jsou zde diskutovány výsledky testování použitelnosti, ale také některé poznatky z testování systému v různých webových prohlížečích. Závěr této kapitoly je věnován ověření adaptivního chování.

Kapitola 1

Úvod do sémantického webu a adaptivních systémů

Úvodní kapitola shrnuje základní pojmy z oblasti sémantického webu a adaptivních systémů. Nejdříve popisuje důležité části, principy a přínosy sémantického webu, poté stručně charakterizuje adaptivní systémy. Cílem této kapitoly je poskytnout základní přehled podstatných částí těchto dvou oblastí, přičemž závěr kapitoly je věnován i propojení těchto dvou světů.

1.1 Sémantický web

Sémantický web, jak již sám název napovídá, se zabývá sémantikou, tedy významem. Nejedná se o nový web, ale o rozšíření toho stávajícího, tedy služby World Wide Web (WWW) [1]. Současný web je převážně tvořen nestrukturovaným obsahem, který je určen především pro lidské zpracování. Nevyužívá žádný formální popis dat, data jsou pouze obalena pomocí značek jazyků HyperText Markup Language (HTML), resp. Extensible HyperText Markup Language (XHTML), které využívají webové prohlížeče pro prezentaci dat uživatelům, na kterých poté je porozumět významu dat. Data jsou také obsažena na různých stránkách, které jsou propojeny pomocí hypertextových odkazů. Jedná se tedy o web propojených dokumentů, který není přizpůsoben pro automatické strojové zpracování, snadnou integraci dat, znovupoužití dat pro různé účely atd. Snahou sémantického webu je naplnit tyto vlastnosti a přeměnit zmíněný web propojených dokumentů na web propojených dat. Zde je vhodné zmínit publikační model Linked Data. Jak uvádí [2], Linked Data je soubor osvědčených postupů pro zveřejňování a propojování strukturovaných dat na webu. Tyto osvědčené postupy představil Tim Berners-Lee, zakladatel konsorcia World Wide Web Consortium (W3C).

Klíčem k úspěchu je především dobře definovaná a jednotná struktura dat. V dnešní době je většina dat ukládána do relačních databází, sémantický web

využívá znalostní databáze. Zatímco v relačních databázích jsou data uložena v tabulkách, resp. řádcích tabulek, v sémantickém webu jsou data reprezentována pomocí tzv. tvrzení, jak bude detailněji vysvětleno dále. Sémantický web je mj. založen na několika osvědčených standardech, přičemž základním stavebním kamenem je protokol Hypertext Transfer Protocol (HTTP) a Uniform Resource Identifier (URI). URI [3] představuje unikátní identifikátor v rámci celého globálního prostoru a jeho používání jakožto identifikátoru pro jakýkoliv objekt nebo koncept je jedním z principů sémantického webu.

1.1.1 Datové modelování a sdílení informací

Ještě než budou detailněji představeny další základní stavební bloky formující sémantický web, je vhodné nejprve uvést tradiční způsoby datového modelování, resp. reprezentace informací a jejich dopady na sdílení dat. Jak uvádí [1], pro sdílení informací mezi systémy je třeba překonat nejprve problém syntaktický a teprve poté ten sémantický. Syntaktický problém spočívá v nalezení společného přístupu k vyměňovaným datům, sémantický v začlenění daných dat do struktury cílového systému.

1.1.1.1 Objektová serializace

Nejjednodušší a nejpřímočařejší způsob je tzv. binární serializace objektů. Na straně zdrojového systému je provedena serializace dat, na straně cílového systému jejich deserializace. Základní syntaktický problém tohoto přístupu je ten, že cílový systém potřebuje předem detailně znát strukturu dat na straně zdrojového systému, aby deserializace byla vůbec možná. Ze sémantického pohledu musí také daná data pochopit, přičemž metadata důležitá pro toto pochopení jsou tvořena názvy vlastností jednotlivých objektů, a najít přesné mapování na vlastní datovou strukturu.

1.1.1.2 Relační model

Relační způsob uložení dat je v dnešní době nejvíce využívaný. Pro sdílení dat mezi systémy oproti předchozí metodě nabízí, alespoň co se týká syntaktického problému, výrazné zlepšení. Relačních databázových systémů existuje celá řada. Díky standardizovaným Application Programming Interface (API)[1] a dotazovacímu jazyku Structured Query Language (SQL) se velikost syntaktického problému sdílení dat do jisté míry zjednodušuje, nicméně stále existuje. Cílový systém si musí být vědom typu zdrojového databázového systému, problémy také mohou způsobit různé verze implementací zmíněných API.

Přetrvává také sémantický problém, je zapotřebí porozumět schématům jednotlivých databází. Metadata zde tvoří definice sloupců tabulek. Je také

[1] např. JDBC

důležité si uvědomit, že v rámci případného dotazování nelze využít primárních klíčů jednotlivých záznamů určitých tabulek pro jejich spojování, jelikož s velkou pravděpodobností nebudou synchronizovány a další mapování by bylo vyžadováno. Z toho všeho je zřejmé, že (nejenom) u systémů, které byly navrženy původně nezávisle, je zapotřebí značného úsilí k zajištění výměny nějakých dat – je to časově i finančně náročné a je nutné výrazné zapojení lidského faktoru.

1.1.1.3 Hierarchický model

Typickým představitelem hierarchické reprezentace dat je Extensible Markup Language (XML). Jedná se o velmi často používaný a standardizovaný formát pro výměnu dat mezi systémy, má značnou podporu programovacích jazyků a různých nástrojů, nejlépe z doposud zmíněných metod adresuje syntaktický problém sdílení dat.

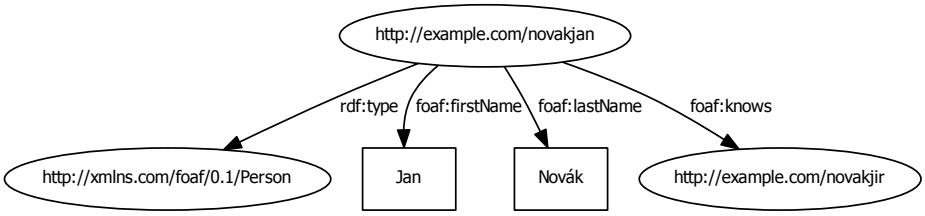
Stále však i zde zůstává sémantický problém. XML umožňuje strukturovat data pomocí vlastních značek s případnými atributy. Tyto značky a atributy však nemají samy o sobě žádný význam. Navíc každý systém může využívat i jinou stromovou strukturu, jiné pojmenování značek atp., což potenciální výměnu dat mezi takovými systémy značně ztěžuje. Ačkoliv je do jisté míry možné k řešení těchto problémů využít Extensible Stylesheet Language Transformations (XSLT) transformace, stále se nejedná o dostatečně flexibilní řešení.

1.1.1.4 Shrnutí

Všechny tři výše uvedené přístupy demonstrovaly pouze některé základní problémy, se kterými se lze setkat. Ze sémantického aspektu zaostává každý z nich a sdílení dat mezi systémy je komplikované, vyžadující značné zapojení člověka do celého procesu. Jak uvádí [1], pro zjednodušení automatizovaného sdílení dat mezi systémy na úrovni sémantiky je zapotřebí propojení metadat a samotných dat. Přesně to nabízí technologie sémantického webu. Následující dvě podkapitoly se zabývají dvěma klíčovými součástmi sémantického webu, a to datovým modelem pro popis dat (kapitola 1.1.2) a ontologiemi (kapitola 1.1.3).

1.1.2 Resource Description Framework

Resource Description Framework (RDF) [4] představuje standardizovaný způsob popisu informací na základě tzv. tvrzení (z angl. statements). Každé tvrzení se skládá vždy ze tří částí, a to subjektu, predikátu a objektu. Subjektem je něco, co popisujeme, prostřednictvím predikátů subjektu přiřazujeme vlastnosti a určujeme vztah mezi subjektem a objektem tvrzení, hodnoty predikátů pak tvoří objekt tvrzení. Díky této konstrukci bývá tvrzení také označováno



Obrázek 1.1: Ukázka RDF grafu referenčního příkladu

jako tzv. RDF trojice (z angl. RDF Triple). Subjekty a predikáty jsou typicky identifikovány pomocí URI, u objektů tvrzení závisí především na tom, zda objekt představuje jiný zdroj nebo se jedná o tzv. literál. Literály reprezentují konkrétní jednoduchá data, jako jsou např. textové řetězce nebo čísla. Z toho je zřejmé, že literály nemohou být subjektem tvrzení, nýbrž pouze jeho objektem.

RDF data formují graf, resp. RDF graf. Tento graf je orientovaný, jeho uzly jsou tvořeny subjekty a objekty tvrzení, hrany reprezentují predikáty. Orientace hrany je vždy od subjektu k objektu tvrzení. V literatuře a různých vizualizačních nástrojích RDF dat se používá konvence taková, že uzly grafu představující zdroje se vždy reprezentují pomocí oválu, naopak literálové uzly jsou reprezentovány pravoúhelníkem.

Pro demonstrativní vyjádření určité informace s užitím RDF mějme entitu reálného světa, fiktivní osobu, o které víme, že její jméno je „Jan Novák“. V sémantickém webu tato osoba představuje zdroj, který identifikujeme pomocí URI, např. `http://example.com/novakjan`. Tento zdroj reprezentuje subjekt. Celé jméno můžeme rozdělit na křestní jméno a příjmení, popis zdroje se tedy bude skládat ze dvou tvrzení, přičemž jeden predikát bude vyjadřovat vlastnost „má křestní jméno“ s hodnotou objektu „Jan“ a druhý „má příjmení“ s hodnotou objektu „Novák“. Objekty těchto tvrzení jsou literály zmíněné dříve. Obrázek 1.1 zachycuje vyjádření těchto informací v RDF grafu, přičemž je navíc doplněn o dvě další tvrzení. Jedno deklaruje jakého typu je daný subjekt (detailněji bude vysvětleno v kapitole 1.1.3) a druhé spojuje zdroj s jiným zdrojem pomocí predikátu, který sémanticky vyjadřuje vztah „zná“. Tento jednoduchý příklad ukazuje, jak flexibilně lze pomocí datového modelu RDF vyjadřovat informace, a to v podstatě o čemkoliv.

1.1.2.1 RDF serializační formáty

Pro výměnu samotných RDF dat mezi systémy existuje několik serializačních formátů, přičemž každý z nich využívá standardizovaný popis dat pomocí tvrzení. Mezi jeden z nejvýznamnějších patří RDF/XML [5] využívající zápis v značkovacím jazyku XML. Základní použití je představeno v ukázce 1.1, jež vychází z referenčního příkladu (viz obrázek 1.1). V příkladu je použita

dobře známá ontologie Friend of a Friend (FOAF) [6], jejíž URI je `http://xmlns.com/foaf/0.1/`.

Ukázka 1.1 Ukázka zápisu RDF dat ve formátu RDF/XML

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3      xmlns:foaf="http://xmlns.com/foaf/0.1/">
4
5    <rdf:Description rdf:about="http://example.com/novakjan">
6      <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
7      <foaf:firstName>Jan</foaf:firstName>
8      <foaf:lastName>Novák</foaf:lastName>
9      <foaf:knows rdf:resource="http://example.com/novakjir"/>
10   </rdf:Description>
11 </rdf:RDF>
```

Popis subjektu začíná elementem `rdf:Description` s atributem `rdf:about`, jehož hodnota udává URI popisovaného zdroje (řádek č. 5). Následující vnitřní elementy reprezentují predikáty a vztahují se k danému subjektu. Příklad demonstruje použití obou typů tvrzení, tj. kdy je objektem tvrzení literál (řádek č. 7 a 8) a kdy jiný zdroj (řádek č. 6 a 9). Příklad také demonstruje uvedení skutečnosti o tom, jakého typu je popisovaný zdroj, k čemuž je použit element `rdf:type` s atributem `rdf:resource`, jehož hodnota pak udává URI konceptu představujícího osobu v ontologii FOAF. Příklad demonstroval základní použití této syntaxe, detailnější informace jsou dostupné v [5]. Na závěr pouze uveďme, že Multipurpose Internet Mail Extensions (MIME) typ pro tuto syntaxi je „application/rdf+xml“.

Mezi další syntaxi patří Terse RDF Triple Language (Turtle) [7], jakožto textová reprezentace RDF grafu. Turtle syntaxe je pro svoji čitelnost a uživatelskou přívětivost velmi oblíbená, umožňuje zápis RDF dat zkrátit a zpřehlednit. Syntaxe Turtle je také používána při tvorbě dotazů v jazyku SPARQL (viz kapitola 1.1.5). Ukázka 1.2 demonstruje základní použití na stejných datech jako v minulém příkladu.

Ukázka 1.2 Ukázka zápisu RDF dat ve formátu Turtle

```
1  @prefix foaf: <http://xmlns.com/foaf/0.1/> .
2
3  # popis zdroje v~syntaxi Turtle
4  <http://example.com/novakjan>
5    a foaf:Person ;
6    foaf:firstName "Jan" ;
7    foaf:lastName "Novák" ;
8    foaf:knows <http://example.com/novakjir> .
```

Zápis začíná uvedením prefixu pro ontologii FOAF. Komentáře lze vytvořit pomocí znaku # (hash), URI musí být uvedena v ostrých závorkách, každé tvrzení je zakončeno tečkou. Pro zkrácení zápisu se využívá několika symbolů. Místo zápisu `rdf:type` lze na místě predikátu použít zkrácenou variantu s použitím znaku „a“ (řádek č. 5). V rámci popisu jednoho zdroje je vhodné subjekt neuvádět vždy znovu, ale za každým objektem použít středník a pokračovat rovnou dalším predikátem. MIME typ pro tento serializační formát je „text/turtle“, pro více informací o této syntaxi lze doporučit [7].

Dalším RDF serializačním formátem je RDF in Attributes (RDFa) [8], kterým lze sémanticky anotovat data přímo v HTML dokumentech. RDFa přidává nové atributy a znovu používá ty staré. Žádný nově vzniklý atribut nemá vliv na vizuální podobu výsledné stránky. Příklad základního použití je opět zachycen v ukázce 1.3.

Ukázka 1.3 Ukázka použití RDFa pro anotaci dat přímo v HTML

```
1  <div vocab="http://xmlns.com/foaf/0.1/" resource="http://example.com/
       novakjan" typeof="Person">
2    <p>Moje jméno je <span property="firstName">Jan</span>
3    <span property="lastName">Novák</span> a znám
4    <a rel="foaf:knows" href="http://example.com/novakjir">Jiřího Nováka</a>
5    </p>
6  </div>
```

Dále stojí za to zmínit syntaxi N-Triples [9], která je vhodná pro výměnu velkých objemů dat. N-Triples je zjednodušená verze syntaxe Turtle, zápis dat nelze žádným způsobem zkrátit, každé tvrzení se typicky píše na samostatný řádek s úplným uvedením subjektu, predikátu a objektu. MIME typ této syntaxe je „application/n-triples“.

Serializaci N-Triples je velmi podobná serializace N-Quads [10], která přidává podporu pro pojmenované grafy. Pro zájemce na úplný závěr této části ještě uveďme syntaxi TriG [11], která rozšiřuje syntaxi Turtle, a JSON-LD [12].

1.1.3 Ontologie

Předchozí kapitola se zaměřovala na datový model RDF, s jehož pomocí lze velmi flexibilně modelovat data. Pouze RDF však nestačí, jelikož RDF je samo o sobě obecný a abstraktní datový model, který data popisuje pomocí tvrzení (subjekt, predikát, objekt). K datům je třeba připojit sémantiku. Předchozí kapitola již částečně představovala zakomponování sémantiky, a to na obrázku 1.1, resp. v rámci demonstrování užití jednotlivých serializačních formátů. K popisu dat jsou nutné jisté vyjadřovací prostředky reprezentující sémantický význam. K jejich definici se využívají slovníky, taxonomie a ontologie [1].

Slovníky, taxonomie i ontologie jsou si velmi podobné a velmi často jsou zaměňovány, všechny tři obsahují definice výrazů. Ve stručnosti lze říci, že slovníky obsahují množinu jednoznačně definovaných pojmů, taxonomie nabízejí hierarchické uspořádání konceptů a ontologie představují definice konceptů a jejich vztahů, resp. definice tříd a vlastností. Ontologie představuje formální znalostní model zaměřující se typicky na určitou doménu. Např. ontologie FOAF použitá v předchozí kapitole zahrnuje zejména prostředky pro popis lidí a jejich vzájemných vztahů.

Pro tvorbu ontologií se využívají jazyky k tomu určené, a to zejména RDF Schema (RDFS) [13] a Web Ontology Language (OWL) [14]. RDFS představuje slovník výrazů, který rozšiřuje základní slovník RDF. Prostřednictvím slovníku RDF a RDFS lze vytvářet základní ontologie. OWL rozšiřuje slovník RDFS a zavádí nové konstrukty, kterými lze definovat přesnější a důkladnější koncepty. Tabulka 1.1 obsahuje jmenné prostory daných slovníků a prefixy, které jsou s nimi konvenčně spojené. Je uveden také standardizovaný slovník Simple Knowledge Organization System (SKOS) [15], který slouží k vyjadřování konceptuálních hierarchií, resp. taxonomií, a který je v ontologiích často využíván [2]. Je vhodné podotknout, že i dané slovníky pro svoji vlastní definici využívají datový model RDF.

Tabulka 1.1: Základní slovníky pro tvorbu ontologií

Prefix	**Jmenný prostor**
rdf	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
rdfs	`http://www.w3.org/2000/01/rdf-schema#`
owl	`http://www.w3.org/2002/07/owl#`
skos	`http://www.w3.org/2004/02/skos/core#`

Ontologie hrají v sémantickém webu klíčovou roli, nejenže umožňují data sémanticky popisovat, ale také lze na jejich základě automaticky odvozovat různé dodatečné a užitečné informace. Obecně je také doporučováno co nejvíce znovu používat již existující ontologie a nezavádět zcela novou ontologii. Důsledkem toho je např. snadnější integrace dat z různých systémů nebo pouhá lepší srozumitelnost pro člověka, který prochází data v nějakém RDF prohlížeči. Tvorbě vlastní ontologie by proto měl předcházet určitý průzkum, zda se již požadované koncepty nenacházejí v nějakých existujících ontologiích. K tomu lze využít různé sémantické vyhledávače. V případě potřeby se poté pouze dodefinují specifické koncepty v ontologii vlastní. Slovníky RDFS a OWL také nabízejí výrazy umožňující mapovaní konceptů napříč různými ontologiemi a jejich vhodné použití opět mj. usnadňuje strojové zpracování.

1.1.3.1 RDF Schema

Základní stavební bloky ontologií tvoří třídy a vlastnosti. Jejich význam je podobný třídám a atributům z objektově orientovaného programování, nicméně

existují malé rozdíly. Popisované subjekty jsou typicky instancemi definovaných tříd. Někdy se též instance třídy označují jako jedinci nebo členové dané třídy. Dva významné prvky jazyka RDFS jsou `rdfs:Class` a `rdfs:Property`. RDFS definuje také vlastnosti pro specifikaci doménové třídy a možných hodnot predikátů. Tyto a některé další základní výrazy ze slovníku RDF, resp. RDFS ukazuje tabulka 1.2, bližší popis a další prvky je možné nalézt např. v [13] nebo přímo v definici jednotlivých slovníků.

Tabulka 1.2: Výběr základních konstruktů ze slovníků RDF a RDF Schema

Výraz	**Použití**
`rdfs:Class`	Třída zdrojů, které jsou RDF třídami.
`rdfs:Property`	Třída pro všechny RDF vlastnosti.
`rdf:type`	Vlastnost určuje skutečnost, že zdroj je instancí dané třídy.
`rdfs:label`	Anotační vlastnost označující koncept.
`rdfs:comment`	Anotační vlastnost pro bližší specifikaci konceptu.
`rdfs:subClassOf`	Vlastnost určující, že třída je podtřídou jiné třídy.
`rdfs:subPropertyOf`	Vlastnost určující, že definovaná vlastnost je dílčí vlastností jiné vlastnosti.
`rdfs:domain`	Vlastnost určující příslušnost definovaného predikátu k určité třídě. Zdroje popsané daným predikátem jsou instancí specifikované třídy.
`rdfs:range`	Vlastnost určující přípustné hodnoty definovaného predikátu.

1.1.3.2 Web Ontology Language

OWL rozšiřuje vyjadřovací schopnosti jazyka RDFS a je doporučován samotným W3C jako hlavní formalizovaný jazyk pro reprezentaci ontologií v sémantickém webu [16]. Při tvorbě jedné ontologie je možné v praxi využívat prvky ze všech zmíněných slovníků v závislosti na požadované přesnosti, která pak ovlivňuje mj. možnosti, potažmo výpočetní složitost odvozování. Tabulka 1.3 představuje pro ukázku některé základní výrazy definované ve slovníku OWL.

1.1.4 Mikroformáty a Microdata

V kapitole 1.1.2.1 byl mj. představen RDF serializační formát RDFa, který lze použít k sémantickému anotování dat přímo v rámci HTML dokumentů. Tato podkapitola stručně uvádí další dvě technologie, které se používají k podobnému účelu. Jedná se o Microdata a Mikroformáty.

První technologií jsou Microdata [17]. Jedná se o mechanismus umožňující anotování obsahu HTML dokumentu takovým způsobem, aby byl strojově čitelný a zpracovatelný. Stejně jako RDFa, Microdata zavádí nové atributy,

Tabulka 1.3: Výběr základních konstruktů definovaných ve slovníku OWL

Výraz	**Použití**
`owl:Ontology`	Třída představující tzv. hlavičku samotné ontologie obsahující její základní popis.
`owl:Thing`	Třída všech instancí, resp. nejvyšší úroveň konceptu.
`owl:ObjectProperty`	Třída pro všechny vlastnosti, jejichž hodnoty tvoří jiné zdroje.
`owl:DatatypeProperty`	Třída pro všechny vlastnosti, jejichž hodnoty tvoří literály.
`owl:equivalentClass`	Vlastnost určující, že dvě třídy definují stejnou věc, resp. mají zcela stejné instance.
`owl:equivalentProperty`	Vlastnost určující, že dvě vlastnosti definují stejný vztah.
`owl:sameAs`	Vlastnost určující, že dva zdroje, resp. instance, představují stejnou věc.

které opět neovlivňují vizuální podobu webové stránky. K anotování se využívají schémata, resp. slovníky, které obsahují definice výrazů. Při anotaci se poté typicky využívá konstruktu „název-hodnota“. Mezi nejvýznamnější slovník patří Schema.org[2], který byl vytvořen ve spolupráci organizací Google, Microsoft a Yahoo. Tento slovník má také několik RDF reprezentací (více viz kapitola 2.5.1).

Oproti tomu Mikroformáty tvoří množina definovaných otevřených datových formátů, které jsou zaměřeny na specifické typy dat. Jednotlivé formáty však nemají velké vyjadřovací schopnosti a jejich rozšiřitelnost není jednoduchá. Jak také uvádí [2], Mikroformáty nelze použít pro sdílení libovolných dat na webu.

1.1.5 Protokol a dotazovací jazyk SPARQL

SPARQL Protocol and RDF Query Language (SPARQL) [18] je protokol a dotazovací jazyk pro RDF data. Je jednou z dalších klíčových součástí sémantického webu. Stejně jako je důležitý jazyk SQL pro relační databáze, je SPARQL důležitý pro sémantická úložiště RDF dat. Datovým zdrojem přitom může být speciální databáze podporující ukládání RDF dat a dotazování se nad nimi, ale např. i pouhý soubor obsahující RDF data. Datová úložiště podporující ukládání RDF dat se nezřídka označují jako RDF úložiště (z angl. RDF Store) nebo úložiště trojic (z angl. Triple Store). Původní verze SPARQL 1.0 [18], jež se stala standardem W3C v roce 2008, umožňuje pouze dotazování se s cílem data získat, ne jejich aktualizaci. Tu umožňuje až novější specifikace

[2] http://schema.org

SPARQL 1.1 [19], standard W3C z roku 2013. Nová verze přináší kromě zmíněné podpory aktualizace dat i další zajímavé vlastnosti. Nutno podotknout, že pro využití nových vlastností této specifikace je nutná podpora ze strany datového úložiště.

Velká výhoda jazyka a protokolu SPARQL spočívá v tom, že umožňuje dotazování se nad daty, která jsou uložena i v různých datových zdrojích. K tomu se využívají tzv. SPARQL endpointy, což jsou zjednodušeně řečeno rozhraní jednotlivých datových úložišť, která umožňují přijmout dotaz a vrátit jeho výsledek. Tato rozhraní jsou typicky identifikována svým URI. Jako příklad lze uvést SPARQL endpoint v sémantickém webu velmi významného datového zdroje DBpedia, který je identifikován URI `http://dbpedia.org/sparql`. Je vhodné uvést, že DBpedia je datový zdroj RDF dat, jenž obsahuje automaticky extrahované informace z celosvětově známé internetové encyklopedie Wikipedia[3] a je jedním z prvních datových zdrojů sémantického webu, potažmo Linked Data [2]. Rozhraní datového zdroje mohou následně využívat jak lidé (např. přímo v rámci webového prohlížeče), tak samotné aplikace, které mohou obohacovat svá data o informace, jež se např. v jejich databázi nenacházejí. Jako zajímavost uveďme, že SPARQL umožňuje dotazovat se nad různými datovými zdroji i v rámci jednoho dotazu. Tato technika se označuje pojmem „Federated Query“ [20] a je součástí novější verze SPARQL 1.1.

Následující podkapitola popisuje pouze některé základy jazyka SPARQL, pro více informací lze doporučit přímo samotnou specifikaci SPARQL 1.0 [18], resp. SPARQL 1.1 [19].

1.1.5.1 Základní syntaxe a typy dotazů

Dotaz v jazyku SPARQL je založen na množině tzv. vzorů trojic (z angl. triple patterns), jež tvoří základní vzor grafu (z angl. basic graph pattern). Na základě množiny základních vzorů grafu je následně vymezen podgraf z dotazovaného grafu. Syntaxe jazyka vychází ze syntaxe Turtle. Dotaz ve většině případů obsahuje proměnné, které se v dotazu typicky uvozují znakem „?“, a lze je v rámci vzorů trojic použít na jakémkoli místě, tj. jako subjekt, predikát nebo objekt tvrzení. Mezi základní typy dotazů jazyka SPARQL patří SELECT, CONSTRUCT, DESCRIBE a ASK. Jednotlivé typy se mj. liší také tím, jaký typ výsledku vracejí:

SELECT Výstupem dotazu je množina n-tic proměnných, které byly uvedeny za klauzulí SELECT. Pouze tyto proměnné jsou zařazeny do výstupu dotazu. Hodnota je do proměnných přiřazena podle množiny základních vzorů grafu, které se definují uvnitř klauzule WHERE.

CONSTRUCT Výstupem tohoto typu dotazu je graf, resp. podgraf, jenž je zkonstruován na základě definované šablony trojic (z angl. triple template), která je naplněna na základě množiny základních vzorů grafu

[3] http://www.wikipedia.org/

uvnitř klauzule WHERE. Pomocí tohoto typu dotazu lze provádět transformaci grafu.

DESCRIBE Výstupem je stejně jako v předchozím případě graf, resp. podgraf. Dotaz se typicky používá v případech, kdy předem přesněji neznáme strukturu dotazovaných dat. Výsledek pak zahrnuje popis vyhovujících zdrojů.

ASK Výstupem tohoto typu dotazu nejsou žádná data, ale pouze logická hodnota, která je určena na základě existence řešení dle definovaného vzoru grafu. Dotaz vrátí hodnotu `true`, pokud je řešení nalezeno.

1.1.5.2 Příklad SPARQL dotazu typu SELECT

Pro základní představu o tom, jak takový SPARQL dotaz vypadá, je dále uveden příklad. Ukázka 1.4 zahrnuje vstupní data zapsaná v syntaxi Turtle. Jedná se o popis dvou smyšlených entit reálného světa.

Ukázka 1.4 Vstupní RDF data zapsaná v syntaxi Turtle

```
1 @prefix foaf: <http://xmlns.com/foaf/0.1/> .
2 <http://example.com/novakjan> a foaf:Person ;
3   foaf:firstName "Jan" ;
4   foaf:lastName "Novák" ;
5   foaf:knows <http://example.com/novotjir> .
6
7 <http://example.com/novotjir> a foaf:Person ;
8   foaf:firstName "Jiří" ;
9   foaf:lastName "Novotný" .
```

Příklad jednoduchého SPARQL dotazu typu SELECT demonstruje ukázka 1.5. Za klauzulí SELECT jsou uvedeny tři proměnné, které budou zahrnuty do výstupu. V klauzuli WHERE je specifikována množina vzorů trojic, jež tvoří jeden základní vzor grafu. Cílem dotazu je ze vstupních dat získat křestní jména a příjmení všech osob, které někoho znají, přičemž do výsledku chceme také zařadit jejich příjmení.

Ukázka 1.5 Ukázka jednoduchého SPARQL dotazu typu SELECT

```
1 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2 SELECT ?firstNameX ?lastNameX ?lastNameY WHERE {
3   ?x a foaf:Person .
4   ?x foaf:firstName ?firstNameX .
5   ?x foaf:lastName ?lastNameX .
6   ?x foaf:knows ?y .
7   ?y foaf:lastName ?lastNameY .
8 }
```

Tabulka 1.4 pak ukazuje výstup dotazu z ukázky 1.5. Záhlaví jednotlivých sloupců obsahuje názvy proměnných, které byly v rámci dotazu SELECT zahrnuty do výsledku. Jejich hodnoty jsou přiřazené dle vzoru grafu. Výsledkem dotazu je jeden záznam, neboť popis zdroje `http://example.com/novotjir` nevyhovuje zadanému vzoru grafu z důvodu absence vlastnosti `foaf:knows`.

Tabulka 1.4: Výstup jednoduchého SPARQL dotazu

firstNameX	**lastNameX**	**y**
"Jan"	"Novák"	"Novotný"

Na závěr je uvedena jedna ze zajímavých nových vlastností specifikace SPARQL 1.1, která se nazývá „Property Path“. Jedná se o možnou cestu mezi dvěma uzly v grafu. Nejjednodušším případem je délka cesty rovná 1, a to v případě klasického vzoru trojice. S použitím „Property Path“ můžeme vytvářet cesty v grafu libovolné délky a zápis dotazu také zkrátit. Ukázka 1.6 ilustruje jedno z možných použití. Účel dotazu je stejný jako v ukázce 1.5 a dotaz vrátí stejný výsledek, jako v předchozím případě. Použití „Property Path“ se nachází na řádku č. 6.

Ukázka 1.6 Ukázka použití konstruktu Property Path

```
1 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2 SELECT ?firstNameX ?lastNameX ?lastNameY WHERE {
3   ?x a foaf:Person .
4   ?x foaf:firstName ?firstNameX .
5   ?x foaf:lastName ?lastNameX .
6   ?x foaf:knows/foaf:lastName ?lastNameY .
7 }
```

1.1.5.3 Shrnutí

V sémantickém webu má SPARQL, dotazovací jazyk a protokol, jednu z klíčových rolí. Je určen pro dotazování se nad RDF daty. Dotazy vycházejí ze syntaxe Turtle, která umožňuje jeho rychlou tvorbu. Dotazy jsou ve většině případů pro člověka snadno pochopitelné, neboť přirozeným způsobem reflektují samotný obsah dat. Některé konstrukty jazyka se podobají těm z dotazovacího jazyka SQL. Je však důležité si uvědomit, že každý operuje se zcela jiným datovým modelem, SPARQL s grafem a SQL s relačním modelem.

Způsob reprezentace dat jako RDF graf přináší do tvorby dotazu řadu výhod. Nemusíme znát přesnou strukturu dat jako tomu je u jazyka SQL, kde musíme znát mnoho informací o daném schématu, např. názvy tabulek, sloupců, co tvoří primární klíč atp. Zjednodušeně řečeno stačí pouze vědět, jaké typy dat se v datovém zdroji nacházejí. Dokonce však nemusíme o datech vědět ani to, neboť SPARQL nabízí dotaz typu DESCRIBE a stačí v podstatě znát

pouze rozhraní datového zdroje, tedy SPARQL endpoint. Zbytek se můžeme dozvědět postupným dotazováním. Relativně snadné je také dotazování se nad hierarchiemi tříd, kde se lze dotazovat na instance obecnější třídy a všechny její odvozené typy.

1.2 Adaptivní systémy

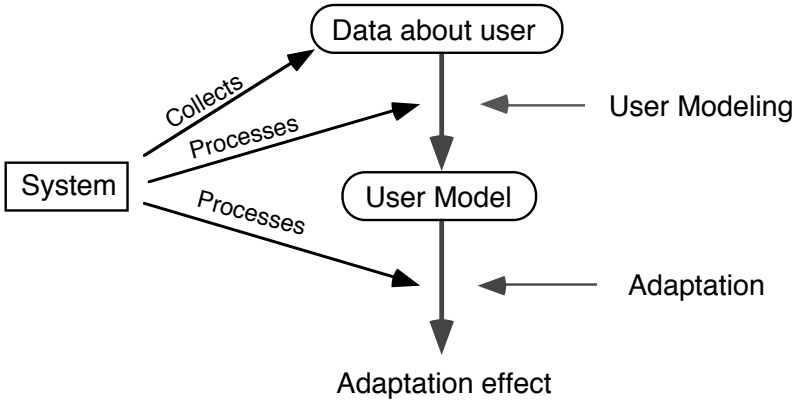
Adaptivní systémy, často se lze setkat také s označením adaptivní hypermediální systémy, jsou stručně řečeno ty systémy, které se nějakým způsobem přizpůsobují konkrétnímu uživateli, a to v závislosti na informacích, které o daném uživateli ať už přímo či nepřímo systém nasbíral. Může se jednat např. o zobrazení personalizovaného obsahu určité stránky nebo navigační podporu. Adaptivní hypermediální systémy jsou dle [21] systémy využívající hypertext, resp. hypermédia, zahrnují tzv. uživatelský model a ten následně využívají pro samotnou adaptaci. Uživatelský model představuje soubor informací o specifickém uživateli, které jsou pro adaptaci potřeba. Tyto informace systém získává typicky buď monitorováním uživatelovy interakce s daným systémem nebo explicitním zadáním dat do systému samotným uživatelem. Většinou jsou oba způsoby kombinovány.

Rozvoj a výzkum oboru výrazně rostl v posledních zhruba 20 letech a jedním z hlavních průkopníků tohoto oboru je Peter Brusilovsky, jenž v roce 1996 představil v práci [21] adaptivní metody a techniky. Od té doby se vyvinulo několik adaptivních systémů, převážně zaměřených na oblast elektronického vzdělávání. Systémy byly často vyvíjeny zcela nezávisle, resp. neexistoval žádný ustálený referenční model pro podporu vývoje těchto systémů. Jedním takovým referenčním modelem se stal AHAM, který byl následně využit v implementaci systému AHA!. Postupem času se různé skupiny snažily rozšířit model AHAM nebo vyvinout nové referenční modely, navzdory tomu se však stále nějaký společný model nedařilo ustanovit [22].

1.2.1 Základy adaptace

Adaptace, jak již bylo zmíněno výše, je založena na vytváření a aktualizování uživatelského modelu, jenž obsahuje nashromážděné informace o konkrétním uživateli. Tento proces znázorňuje obrázek 1.2. Uživatelský model je tvořen ve většině případů na základě doménového modelu daného systému.

Před samotnou adaptací je vhodné si položit a zodpovědět 6 hlavních otázek, které vycházejí z práce [21] Brusilovskyho. První otázka, „Kde?“, se zaobírá tím, v jakých systémech může být adaptace nápomocná. Druhá otázka „Proč?“ se zajímá o to, proč je vlastně adaptace potřeba, v čem může uživatelům pomoci. Třetí, „Čemu?“, a následně čtvrtá, „Co?“, se týká toho, podle jakých rysů uživatele lze adaptaci provádět, resp. co můžeme v systému adaptovat, přičemž první staví především na informacích uložených v uživatelském modelu a druhá koresponduje s doménovým modelem. Další otázka „Kdy?“



Obrázek 1.2: Proces adaptace založený na uživatelském modelu [21]

se zabývá tím, kdy je adaptace prováděna a poslední, šestá otázka „Jak?“, se již zabývá především implementační úrovní a uvažuje konkrétní adaptivní metody a techniky [22].

1.2.2 Adaptivní metody a techniky

V kontextu adaptace se rozlišují dle [22] tři její základní typy, a to adaptace obsahu, adaptace navigace a adaptace prezentace. Je vhodné poznamenat, že mezi jednotlivými třemi typy někdy existuje tenká hranice a některé techniky mohou být zařazeny do více než jedné kategorie.

Adaptace obsahu spočívá buď v zobrazení, resp. skrytí určité informace před uživatelem, nebo v jejím zvýraznění či naopak utlumení. V prvním případě tedy daná informace není vůbec uživateli zpřístupněna, v druhém se systém snaží uživateli napovědět, na jakou část informace by se měl zaměřit. Mezi používané techniky lze zmínit např. přibližování/oddalování, zvětšování/zmenšování velikosti textu.

Adaptace navigace se v zásadě rozlišuje podle toho, zda jsou navigační prvky uživateli skryty, případně jestli je deaktivováno jejich použití, nebo jsou nějakým způsobem upraveny, např. seřazením podle důležitosti či různým barevným označením napovídající uživateli další pohyb.

Do adaptace prezentace se následně řádí část technik, které byly využity už v předchozích dvou typech, a dále také tzv. adaptace rozvržení pro adaptaci různých zařízení nebo preferovaných rozvržení.

1.3 Sémantický web a adaptivní systémy

Závěr kapitoly je stručně věnován propojení sémantického webu a adaptivních systémů. Jak je naznačeno v práci [22], jistá skupina autorů adaptivních systémů postupem času vytvořila potřebu kombinovat různé adaptivní systémy,

resp. informace, které o určitém uživateli systémy nasbíraly, tedy jeho uživatelský model. Za tímto účelem se nabízelo využití technologií sémantického webu (viz kapitola 1.1), které mj. znovupoužitelnost a integraci dat podporují a zjednodušují.

Využití sémantického webu v adaptivních systémech znamená přechod od klasické reprezentace dat k reprezentaci dat založené na ontologiích [16]. Jednou z důležitých částí byla proto snaha o vytvoření společných ontologií pro popis uživatelských modelů, resp. uživatelských profilů. Uživatelský model do této části znamenal jak informace získané monitorováním uživatelovy interakce s daným systémem, tak statické informace, které uživatel do systému sám zadal. Někdy se lze setkat s rozdělením těchto informací do uživatelského modelu a uživatelského profilu, přičemž uživatelský model zahrnuje ty dynamicky zjištěné informace, uživatelský profil ty statické, např. zjištěné při registraci nebo ze stránky s nějakým nastavením.

1.3.1 GOMAWE

Generic Ontology-based Model for Adaptive Web Environments (GOMAWE) [23] je obecný model založený na sémantické reprezentaci dat určený pro adaptivní webové prostředí. Tento model má vícevrstvou architekturu, přičemž velmi důležitá je vrstva datová, která je rozdělená na tři části, a to doménový model, uživatelský model a adaptační model. Uživatelský model přitom koresponduje s rozdělením na uživatelský model a uživatelský profil, jak již bylo zmíněno v předchozí části. Pro podporu vývoje adaptivních systémů byl vytvořen ASF [24], který je na modelu GOMAWE postaven. Framework ASF bude detailněji představen v kapitole 3.1.3 v části realizace.

Kapitola 2

Analýza a návrh

Tato kapitola se zabývá analýzou a návrhem implementovaného systému. Na úvod kapitoly jsou nejprve v rámci analýzy uvedeny vlastnosti některých existujících systémů pro správu (nejen) multimediálního obsahu. Důležitou fází byl také výběr konceptů z již existujících ontologií a případné zavedení nových konceptů ve vlastní ontologii. Zbývající část kapitoly je věnována samotnému návrhu prototypu systému.

2.1 CMS systémy

Jako Content Management System (CMS) jsou obecně označovány systémy, které v různé míře, určené komplexností daného systému, umožňují správu různého typu obsahu. Ke správě obsahu je typicky využíváno k tomu určené uživatelské rozhraní a není tak zapotřebí přímého přístupu uživatele k databázi. Uživatelské rozhraní těchto systémů je typicky rozděleno zvlášť pro administraci systému (backend) a zvlášť pro prezentaci obsahu koncovému uživateli (frontend). Webové systémy pro správu obsahu se také často označují jako redakční systémy, pomocí kterých lze typicky tvořit články, které v sobě mohou dále obsahovat různý multimediální obsah, diskuzní fórum nebo hodnocení. Typickou vlastností je v případě webových systémů také přizpůsobení barevného tématu vzhledu systému. Rozsah systémů je velký, následující analýza stručně popisuje významné webové CMS se zaměřením na správu multimediálního obsahu i využití prvků sémantického webu.

2.1.1 Drupal

Drupal[4] je komplexní open-source CMS systém pro organizaci, správu a publikaci téměř jakéhokoli obsahu s velkou možností přizpůsobení daným potřebám. Počátky systému se datují do roku 1999, od této doby se i díky velké komunitě

[4] http://drupal.org

vývojářů a modulární architektuře velmi výrazně rozrostl. Samotný systém je napsán v programovacím jazyku PHP: Hypertext Preprocessor (PHP), pro uložení dat využívá relační databázi. Nabízí tisíce modulů, díky jejich kombinaci je možné vytvořit téměř jakýkoliv typ systému. Pro správu multimédií je k dispozici mnoho modulů, díky kterým je umožněno jejich nahrávání, anotace, transformace do různých formátů nebo jejich následné zobrazení koncovému uživateli.

Drupal umožňuje využití technologií sémantického webu. Již skoro od samotného počátku systému byla snaha o zakomponování technologie RDF do jeho datové struktury. [25] V současné době Drupal již umožňuje mapování jeho datové struktury na koncepty z ontologií i následný export v podobě RDF. Obsažen je i modul pro podporu datového modelu Schema.org.

2.1.2 WordPress

WordPress[5] je populární open-source CMS systém implementovaný v programovacím jazyku PHP. Je určen pro tvorbu blogů, webových stránek, avšak s využitím značného množství pluginů lze opět vytvořit téměř jakýkoliv typ systému. Pro uložení dat je opět využívána relační databáze. Správa multimédií je umožněna především díky pluginu „rtMedia“[6]. Ten umožňuje např. konverzi multimédií do jiných formátů, shlukování médií do kolekcí nebo nastavování jejich viditelnosti.

Vzhledem k sémantickému webu, systém nabízí pluginy pro podporu datového modelu Schema.org, ale i pluginy[7] pro mapování vlastních objektů na některé významné ontologie (např. FOAF) a následné poskytnutí zkonvertovaných dat v podobě RDF/XML jako webový zdroj dat (z angl. web feed).

2.1.3 Joomla

Joomla[8] je open-source CMS systém, opět implementovaný v jazyku PHP. Pro uložení dat využívá relační databázi. Pro správu multimédií je určena zejména komponenta Media Manager, která umožňuje nahrávání nových multimédií, organizaci adresářů a jejich správu.

Joomla od verze 3.3 zahrnuje vestavěnou podporu pro technologii Microdata a automaticky anotuje pouze některé části obsahu. Zápis Microdata je vložen přímo do zobrazovacích šablon a nelze tak nastavit, pro jaké typy obsahu mají být použity. [26]

[5] https://wordpress.org/
[6] https://wordpress.org/plugins/buddypress-media/
[7] např. plugin https://wordpress.org/plugins/lh-rdf/
[8] http://www.joomla.org/

2.1.4 Shrnutí

Všechny uvedené systémy v sobě zahrnují širokou škálu funkcí. Díky modulům nebo pluginům umožňují snadnou rozšiřitelnost o nové funkcionality a díky tomu je možné tyto systémy využít ke správě téměř jakéhokoliv obsahu. Všechny uvedené systémy pracují v základu s relační databází, nikoliv se sémantickým datovým úložištěm dat, nicméně každý z uvedených systémů již umožňuje v jisté míře anotaci dat s využitím prvků sémantického webu. Jedná se především o způsob anotace dat přímo v zobrazovacích šablonách, k čemuž využívají zejména datový model Schema.org, na který mapují některé prvky své datové struktury a následně využívají technologie Microdata nebo RDFa.

2.2 Požadavky

Následující část zahrnuje funkční a nefunkční požadavky na prototyp systému. Funkční požadavky specifikují, co by měl systém umožňovat, zatímco nefunkční popisují především obecné požadavky na použité technologie a některá implementační specifika. Zde je vhodné uvést, že cílem práce není vytvořit systém, který by se snažil jakkoli konkurovat existujícím CMS systémům, ale využít relativně nové technologie pro implementaci jednoduchého prototypu využívající prvky sémantického webu. Od toho se odvíjejí i požadavky na samotný systém.

2.2.1 Funkční požadavky

Systém má umožňovat správu multimediálního obsahu. Jednu z hlavních činností představuje tvorba/editace článků. Tyto články mají zahrnovat určitý typ média, v základu se jedná o audio, obrázek nebo video. Správa článků a médií je povolena pouze autorizovaným uživatelům (tvůrci obsahu), systém tedy musí umožňovat registraci, resp. přihlášení. Systém dále umožní autorizovanému uživateli (administrátor) správu všech ostatních uživatelů, kde může měnit jejich role v rámci systému nebo je ze systému odstranit, a který dále může spravovat všechny dostupné články.

Systém bude rozlišovat typ článku na základě typu média, který obsahuje, přičemž článek s obrázky musí zahrnovat alespoň jeden obrázek, článek s audiem právě jednu audio nahrávku, resp. článek s videem právě jedno video. Médium, které je již přítomné v některém z článků, systém nepovolí odstranit. Systém bude uchovávat u článků a médií informace o tom, kdo a kdy provedl poslední změnu. Odstranění záznamů z celého systému bude systémem provedeno pouze nastavením indikátoru a přidáním informace o tom, kdo tuto akci inicioval.

Tvorba/editace článků bude probíhat pro každý typ článku vyplněním všech společných údajů. Článek po úspěšném vytvoření muže být ve stavu publikovaném, v tom případě bude zahrnut ve výpisu článků pro běžné uži-

vatele, resp. ve stavu nepublikovaném a do výpisu článků pro běžné uživatele zahrnut nebude. Výběr média při tvorbě/editaci článku je možný ze všech již nahraných médií v systému, přičemž systém vždy nabídne pouze seznam médií dle typu daného článku.

Všichni uživatelé mohou upravovat své osobní údaje, procházet vytvořené články a zobrazovat jejich detail, v rámci kterého mohou zobrazit, resp. přehrát daný typ multimediálního obsahu. Systém má také zahrnovat adaptivní prvky, které se vztahují vždy k určitému uživateli. Z toho vyplývá, že do systému bude povolen vstup až po úspěšné autentizaci i pro běžné uživatele systému (členové). Články lze v rámci zobrazení jejich detailu hodnotit a pokud je přidávání komentářů k článku povoleno, tak i komentovat.

2.2.2 Nefunkční požadavky

Níže je uveden seznam nefunkčních požadavků na systém. Většina uvedených technologií vyplývá přímo ze zadání této práce. Výsledný prototyp bude také nasazen na jeden ze serverů Fakulty elektrotechnické Českého vysokého učení technického pro účely závěrečného testování s uživateli.

- Systém bude přístupný přes webové rozhraní.
- Systém bude víceuživatelský.
- Systém bude uchovávat hesla uživatelů v zašifrované podobě.
- Systém bude operovat s RDF daty. Jako úložiště pro RDF data bude použito datové úložiště Sesame.
- Systém bude implementovaný v programovacím jazyku Java.
- Systém bude postaven na frameworku Spring.
- Systém bude využívat frameworky Adaptive System Framework (ASF) a Java Server Faces (JSF).
- Systém bude využívat implementaci Java Persistence API (JPA) Empire. Jako implementace rozhraní `PersistenceProvider` bude použita implementace `EmpireJPAPersistenceProvider` z frameworku ASF.
- Systém bude generovat pro zdroje HTTP schéma URI, ale URI nemusí být dereferencovatelná.
- Systém bude mít jednoduché uživatelské rozhraní. V základu bude jednotné pro všechny uživatelské role systému a specifická umístění systému musí být zabezpečena.
- Systém nebude provádět konverzi nahrávaných médií do jiných formátů.

- Vlastní texty šablon a informační hlášení systému nebudou zasazena přímo do implementace, ale budou v externím souboru. Systém však bude v základu lokalizován pouze v českém jazyce.

2.3 Případy užití

Případy užití popisují možné činnosti, které uživatelé se samotným systémem mohou provádět. Systém je víceuživatelský a proto se v případech užití vyskytují různí tzv. aktéři. Na základě funkčních požadavků byli identifikováni čtyři aktéři, jež jsou dále označováni jako: *a*) anonymní uživatel, tj. uživatel, který není přihlášen do systému; *b*) člen, tj. uživatel, který vlastní základní uživatelský účet vzniklý po registraci a je přihlášen do systému; *c*) editor, tj. uživatel, který může vše, co člen a navíc tvořit samotný obsah; a konečně *d*) administrátor, tj. uživatel, který může vše co editor a má navíc další možnosti správy systému.

Následující případy užití jsou pouze rámcové a nejsou také uvedeny případy užití jako filtrování, řazení atp. Popis případů užití je pro větší přehlednost rozdělen na tři části. První se zaměřuje na činnosti týkající se správy údajů samotných uživatelů, druhá se zaměřuje na činnosti související s články a poslední na činnosti související s médii.

2.3.1 Případy užití – lidé

Případy užití této části se zabývají takovými činnostmi aktérů se systémem, které přímo ovlivňují jejich stav v rámci systému. Model těchto případů užití ilustruje obrázek F.1.

- **Přihlásit se** – Anonymní uživatel zadá své přihlašovací údaje. Systém ověří zadané údaje a v případě platnosti zadaných údajů je uživatel přihlášen do systému, v opačném případě je uživatel vyzván k opravě chybně zadaných údajů.
- **Registrovat se** – Anonymní uživatel vyplní požadované údaje. Systém ověří zadané údaje a v případě platnosti zadaných údajů systém uživatele zaregistruje a automaticky přihlásí do systému, v opačném případě je uživatel vyzván k opravě chybně zadaných údajů.
- **Odhlásit se** – Systém odhlásí uživatele.
- **Upravit osobní údaje** – Uživatel upraví své osobní údaje. Systém ověří zadané údaje. V případě jejich platnosti změny zaeviduje, v opačném případě je uživatel vyzván k opravě chybně zadaných údajů.
- **Nastavit adaptační kritéria** - Uživatel nastaví adaptační kritéria. Systém ověří zadané údaje. V případě jejich platnosti změny zaeviduje, v opačném případě je uživatel vyzván k opravě chybně zadaných údajů.

- **Nastavit vzhled** – Uživatel může nastavit některé parametry vzhledu pro přizpůsobení některých prvků prezentace. Např. pro změnu barevného tématu systému, uživatel provede změnu barevného tématu vzhledu. Systém změní téma vzhledu pro daného uživatele a změnu zaeviduje.

- **Odstranit uživatele** – Administrátor chce odstranit nějakého uživatele. Systém zobrazí seznam uživatelů. Administrátor zvolí záznam k odstranění. Systém po potvrzení akce odstraní uživatele ze systému.

- **Změnit roli uživatele** – Administrátor chce změnit roli nějakého uživatele. Systém zobrazí seznam uživatelů. Administrátor zvolí uživatele, kterému chce změnit jeho roli. Systém zobrazí daný záznam. Administrátor provede změnu role uživatele. Systém zaeviduje změnu uživatelovy role.

- **Přidat uživatele** – Administrátor vyplní požadované údaje. Systém zadané údaje ověří a v případě platnosti zadaných údajů systém vytvoří nového uživatele, v opačném případě je administrátor vyzván k opravě chybně zadaných údajů.

- **Zobrazit seznam uživatelů** – Systém zobrazí administrátorovi seznam všech dostupných uživatelů v systému.

2.3.2 Případy užití – články

Případy užití této části se zabývají takovými činnostmi aktérů se systémem, které operují s články, ať už se jedná o jejich správu nebo normální zobrazení běžnými uživateli. Model těchto případů užití ilustruje obrázek F.2.

- **Ohodnotit článek** – Uživatel má zobrazený detail článku. Uživatel článek ohodnotí užitím dané stupnice. Systém vypočte nové hodnocení a změnu zaeviduje.

- **Komentovat článek** – Uživatel má zobrazený detail článku, u kterého je povoleno přidávání komentářů. Uživatel vloží text nového komentáře. Systém zadaný údaj ověří a v případě platného údaje přidá nový komentář.

- **Zobrazit článek** – Uživatel chce zobrazit detail článku. Systém zobrazí seznam dostupných a publikovaných článků. Uživatel zvolí článek. Systém zobrazí detail článku.

- **Zobrazit seznam článků pro členy** – Systém zobrazí seznam dostupných a publikovaných článků. Uživatel může seznam zobrazit s uplatněním adaptačních kritérií.

- **Odstranit vlastní článek** – Editor chce odstranit článek, u kterého je autorem. Systém zobrazí seznam článků, u nichž je autorem. Editor zvolí záznam k odstranění. Systém po potvrzení akce odstraní článek ze systému.

- **Zobrazit seznam svých článků** – Systém zobrazí seznam dostupných článků, u nichž je editor autorem.

- **Vytvořit článek** – Editor vloží všechny požadované údaje k článku. Systém po potvrzení akce ověří vložené údaje a v případě platnosti všech údajů vytvoří nový článek, v opačném případě je editor vyzván k opravě chybně zadaných údajů.

- **Upravit článek** – Editor upraví požadované údaje. Systém po potvrzení akce ověří vložené údaje a v případě platnosti všech údajů zaeviduje změny, v opačném případě je editor vyzván k opravě chybně zadaných údajů.

- **Odstranit článek** – Administrátor chce odstranit libovolný článek. Systém zobrazí seznam všech dostupných článků. Administrátor zvolí záznam k odstranění. Systém po potvrzení akce odstraní článek ze systému.

2.3.3 Případy užití – mediální soubory

Případy užití této části se zabývají takovými činnostmi aktérů se systémem, které operují se samotnými médii. Model těchto případů užití ilustruje obrázek F.3.

- **Zobrazit médium** – Uživatel má zobrazený detail článku. Systém uživateli zobrazí obsah média ve formě dle typu daného média.

- **Nahrát médium** – Uživatel vyplní metadata k médiu a vybere médium k nahrání do systému. Systém ověří zadané údaje i podporu daného média. V případě platnosti zadaných údajů je zahájeno nahrávání souboru do systému a zaevidování všech údajů.

- **Upravit metadata média** – Uživatel chce upravit metadata k médiu. Systém zobrazí seznam médií. Uživatel vybere médium, jehož metadata chce změnit. Systém zobrazí metadata daného média. Uživatel provede změnu. Systém ověří zadané údaje a v případě platnosti zadaných údajů změny zaeviduje.

- **Zobrazit seznam médií** – Systém zobrazí seznam všech dostupných médií v systému.

- **Odstranit médium** – Uživatel chce odstranit médium ze systému. Systém zobrazí seznam médií. Uživatel zvolí záznam k odstranění. Systém po potvrzení této akce zkontroluje, zda se médium nevyskytuje v nějakém článku. Pokud se nevyskytuje, médium odstraní.
- **Přidat médium k článku** – Uživatel vytváří nový článek a chce přiřadit nové médium. Systém nabídne seznam médií typu dle typu článku. Uživatel zvolí médium, které chce přidat a potvrdí. Systém dané médium přidá k článku.
- **Změnit médium k článku** – Uživatel vytváří/upravuje článek a chce změnit médium. Systém nabídne seznam médií typu dle typu článku. Uživatel zvolí nové médium a potvrdí. Systém zvolené médium přidá k článku.

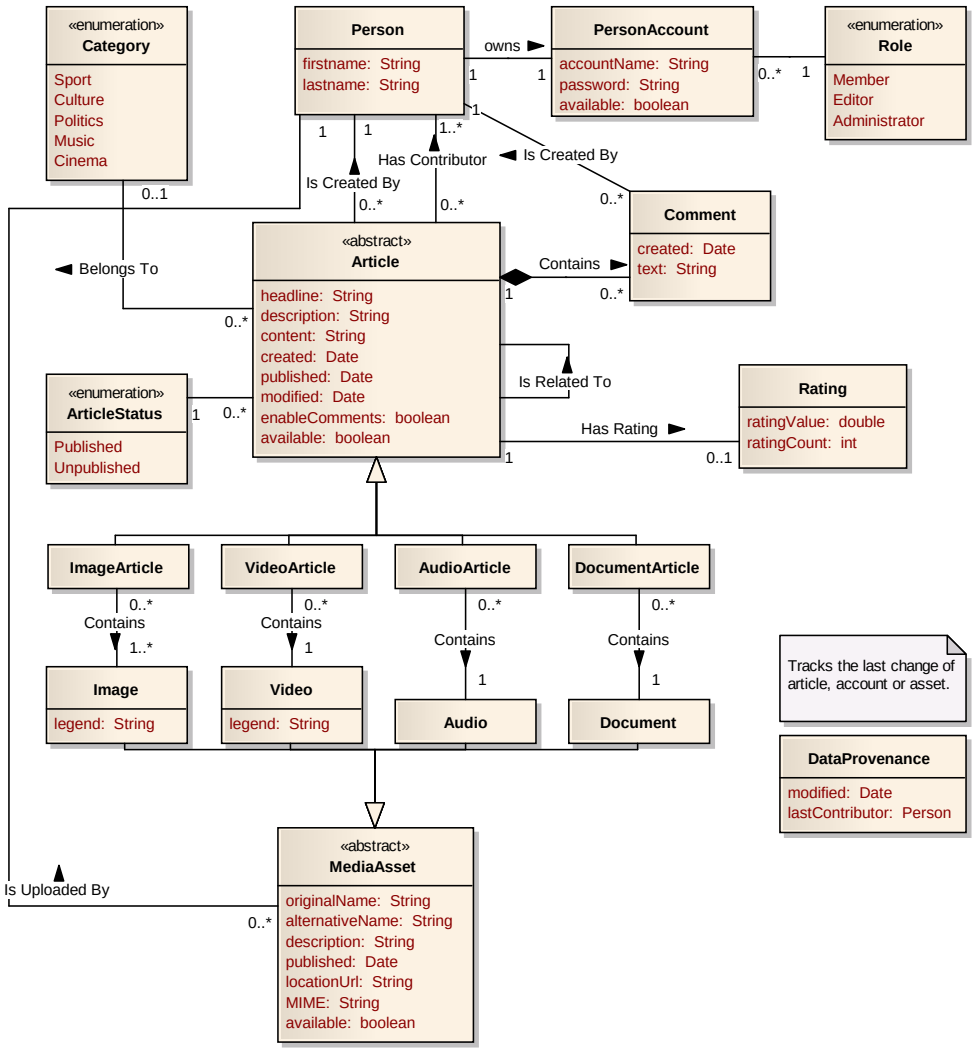
2.4 Doménový model

Doménový model vychází z analýzy funkčních požadavků a je znázorněn na obrázku 2.1. Ačkoliv to nebývá u konceptuálního modelu zvykem, jsou uvedeny i datové typy. Nejedná se o konečnou verzi využívanou samotným systémem, z tohoto modelu dále vychází analýza existujících ontologií za účelem pokrytí jednotlivých entit, atributů a vztahů (viz kapitola 2.5.1), nebo k případnému dodefinování vlastních konceptů (viz kapitola 2.5.2). Je vhodné zde také poznamenat, že některé části z doménového modelu, jako je např. zařazení článku do kategorie nebo typ článku obsahující libovolný dokument, nejsou ve výsledném prototypu obsaženy, přičemž prvnímu uvedenému je také věnována část diskuze v závěru práce.

Následující část se zaměřuje na stručný popis jednotlivých prvků doménového modelu, a to pouze takovým, u kterých nemusí být z daného diagramu ihned zřejmý jejich význam. Entita `MediaAsset` reprezentuje mediální soubor, přičemž atribut `originalName` představuje původní název souboru, resp. atribut `alternativeName` název, pod jakým je uložen na diskovém oddílu serveru. Atribut `legend` u obrázků a videí představuje krátký titulek, který se např. běžně používá u obrázků jako hodnota atributu `title` značkovacího jazyka HTML. Entita `Rating` reprezentuje hodnocení článku s atributy pro celkový počet hodnocení, resp. jeho hodnotu. Entita `DataProvenance` není v diagramu, kvůli jeho vyšší přehlednosti, spojena vazbou s žádnou další entitou, nicméně vztahuje se na entity článek, uživatelský účet, médium a modeluje uchovávání informace o datu poslední změny a jejího původce.

2.5 Návrh ontologie

Tato podkapitola popisuje pokrytí všech jednotlivých tříd, atributů a vztahů doménového modelu pomocí ontologií. Jak bylo zmíněno v kapitole 1.1.3,



Obrázek 2.1: Doménový model

tvorbě vlastní ontologie by měl předcházet určitý průzkum existujících ontologií a až poté k případnému dodefinování specifických konceptů v ontologii vlastní.

Ve světě sémantického webu se vyskytují již velmi známé ontologie, které se typicky zaměřují na určitou doménu. Pokud si však nejsme těchto ontologií vědomi, může být tato část návrhu obtížná. Pro hledání existujících konceptů je proto v praxi vhodné využít různé sémantické vyhledávače, které s touto časově náročnou a přitom velmi důležitou činností člověku pomáhají. Jedná se hlavně o Linked Open Vocabularies (LOV)[9] a sémantický vyhledávač Swoogle[10].

2.5.1 Analýza existujících ontologií

Cílem této analýzy je prozkoumat existující slovníky a najít v nich výrazy, které by bylo možné využít pro pokrytí všech prvků z doménového modelu, tj. každou třídu, atribut a vztah. Následující části popisují nejprve slovníky, které byly identifikovány jako možní adepti a které jsou, alespoň v určité míře, využity v samotném systému, dále také některé možné alternativy, kterými lze určité části modelu pokrýt taktéž nebo je lze využít i v případě budoucího rozvoje. V rámci popisu nejsou popsány všechny použité třídy, resp. vlastnosti z jednotlivých ontologií, kompletní pokrytí je uvedeno v kapitole 2.5.3.

Dublin Core Metadata Initiative

Dublin Core Metadata Initiative (DCMI) je jednou z prvních iniciativ, která vytvořila slovník pro popis základních metadat, která jsou společná pro většinu zdrojů. Definované výrazy jsou rozděleny do několika slovníků, přičemž základní výrazy definuje slovník DCMI Metadata Terms (DCTERMS) [27] s jmenným prostorem `http://purl.org/dc/terms/`.

Slovník definuje výrazy jako jsou např. název (titulek), popis, autor, vydavatel, přispěvatel, datum vytvoření, datum změny a další. Slovník reviduje a rozšiřuje původní slovník Dublin Core Metadata Element Set (DCMES) [28] s jmenným prostorem `http://purl.org/dc/elements/1.1/`, který povoluje pro většinu predikátů jako hodnotu tvrzení použít literál nebo zdroj. Slovník DCTERMS již pro většinu predikátů povoluje jako objekt tvrzení použít pouze jiný zdroj, jelikož definuje `rdfs:range`. Příklady použití a rozdíly mezi těmito dvěma slovníky lze nalézt v [29].

Při pokrytí budeme vycházet ze slovníku DCTERMS. Pro určení autora každého článku, resp. komentáře bude použit predikát `creator`, podobně pro přispěvatele predikát `contributor`, resp. predikát `publisher` pro označení člověka, který do systému nahrál daný soubor. Pro určení data vytvoření každého zdroje bude sloužit vlastnost `created`, pro datum změny `modified` a

[9] http://lov.okfn.org/dataset/lov/
[10] http://swoogle.umbc.edu/

pro datum zveřejnění článku, resp. datum nahrání multimédia do systému použijeme vlastnost `issued`. Pro popis každého zdroje bude použita vlastnost `description`. Pro původní název souboru bude použita vlastnost `title` a vlastnost `alternative` pro název, pod jakým bude uložen na diskovém oddílu serveru.

Friend of a Friend

FOAF [6] již byl v předchozím textu několikrát použit a byl představen jako slovník pro popis lidí a jejich vzájemných vztahů. Slovník ale nabízí i výrazy, pomocí kterých lze také v určité míře popisovat organizace, dokumenty, projekty, skupiny atp.

Z tohoto slovníku bude použita hlavně třída `foaf:Person` pro popis uživatelů systému a odpovídající predikáty, jako je např. jméno, příjmení, email. Pro popis uživatelského účtu se nabízí použití třídy `foaf:OnlineAccount`, lepším řešením je ale použití třídy z následujícího slovníku, která je vymezena přesně daným způsobem a je podtřídou právě zmíněné třídy `foaf:OnlineAccount`.

Semantically-Interlinked Online Communities

Semantically-Interlinked Online Communities (SIOC) [30] s jmenným prostorem ontologie `http://rdfs.org/sioc/ns#` je slovník, který rozšiřuje slovník FOAF a nabízí další výrazy pro popis sociálních vztahů online komunit. Z tohoto slovníku bude použita např. třída `foaf:OnlineAccount` pro reprezentaci uživatelského účtu nebo `sioc:Role` pro reprezentaci uživatelské role.

Schema.org

Schema.org, jak již bylo uvedeno v první kapitole, představuje nejrozšířenější datové schéma pro technologii Microdata, jež bylo vyvinuto ve spolupráci předních webových vyhledávačů. Jedná se o obecný datový model[11], který je odvozen z RDFS a který je rozšiřován a aktualizován velkou komunitou, jež zahrnuje jak celé organizace, tak např. i samostatné vývojáře. Pro datový model je charakteristické to, že se nezaměřuje na jednu specifickou doménu, ale zahrnuje celou řadu vyjadřovacích prostředků pro různé domény, podobně jako např. ontologie zmíněného datového zdroje DBpedia.

Pro Schema.org přitom existuje i několik RDF reprezentací, díky čemuž je možné celý datový model využívat i v aplikacích, které potřebují operovat právě s daty ve formě RDF. Ačkoliv projekt Schema.org nabízí vlastní OWL verzi celého datového modelu, vzniklo několik dalších reprezentací. Příkladem může být OWL verze TopBraid [31], která automaticky zpracovává celý datový model Schema.org. Tato ontologie koresponduje s původním datovým modelem, pro koncepty využívá i stejná URI, nicméně zavádí několik vlastních

[11] http://schema.org/docs/datamodel.html

konvencí, aby celá ontologie více vyhovovala RDF/OWL nástrojům. Např. místo třídy `schema:Thing` z původní ontologie je využívána třída `owl:Thing` nebo místo vlastních datových typů využívá standardní XML Schema Datatypes (XSD). Z těchto důvodů bude prototyp systému využívat právě verzi TopBraid.

Z analýzy datového modelu Schema.org vyplynulo, že pro mnoho konceptů z doménového modelu je možné Schema.org využít. Např. pro základní typ článku se nabízí třída `schema:Article`, pro obecný multimediální soubor třída `schema:MediaObject` i s jejími podtypy, dále lze pokrýt i např. hodnocení či komentování článků. Výhodou datového modelu Schema.org je i to, že díky své popularitě spousta jiných ontologií mapuje své vyjadřovací prostředky na koncepty Schema.org a naopak. Např. třída `schema:Person` je ekvivalentní konceptu pro osobu z ontologie FOAF nebo třída `schema:ImageObject` je ekvivalentní třídě `dcmitype:Image` [27] nebo `foaf:Image` [6].

2.5.1.1 Další možné ontologie pro popis multimédií

Pro pokrytí velké části doménového modelu budou využity výše uvedené ontologie, to ale neznamená, že by nebylo možné některé koncepty pokrýt pomocí jiných ontologií, případně mezi některými koncepty dodefinovat mapování pomocí jazyka RDF, resp. OWL. Níže je stručný popis několika ontologií, které by bylo možné také využít pro popis multimediálních souborů.

Ontology for Media Resources 1.0 [32] je ontologie vyvinuta pod záštitou W3C. Ontologie zahrnuje množinu vyjadřovacích prostředků pro základní anotaci multimédií. Základní třídou je `ma-ont:MediaResource`, kterou je možné reprezentovat obrázek nebo libovolný audiovizuální zdroj.

EBUCore [33] je ontologie, jejíž vydavatelem je sdružení[12] národních vysílacích stanic. Ontologie obsahuje velmi rozsáhlé datové schéma, které poskytuje prostředky pro popis téměř jakéhokoliv audiovizuálního obsahu. Základní třída pro reprezentaci multimédia je `ebucore:MediaResource`.

2.5.2 Dodefinovaní vlastních konceptů

Z předchozí analýzy existujících slovníků vyplývá, že většinu tříd, atributů a vztahů lze pokrýt užitím konceptů z již existujících slovníků. Ve vlastní ontologii proto dodefinujeme pouze několik vlastních konceptů, které se přitom pokusíme i v rámci ontologie namapovat na již existující koncepty užitím vlastností k tomu určených. Připomeňme, že tyto predikáty se nacházejí ve slovníku RDFS, resp. OWL. Je vhodné také poznamenat, že při tvorbě ontologie lze využít speciální nástroje, které jsou k tomu určeny. Mezi jeden z nejznámějších patří Protégé[13]. V ontologii dodefinujeme následující koncepty:

[12] European Broadcasting Union
[13] http://protege.stanford.edu/

- Třídy pro jednotlivé typy článků.
- Predikát pro určení toho, zda je v rámci článku povoleno přidávat komentáře.
- Predikát indikující dostupnost zdroje v rámci systému. Odstranění záznamů není na základě funkčních požadavků vykonáno permanentním způsobem, ale pouze nastavením indikátoru.
- Predikát pro určení toho, kdo provedl poslední změnu daného zdroje.
- Koncepty pro jednotlivé role v rámci systému, tj. administrátor systému, editor a běžný registrovaný uživatel.
- Koncepty pro jednotlivé stavy článků. S tím souvisí i definice predikátu přiřazující instance článků s těmito koncepty.

2.5.2.1 Obecné informace k návrhu

Mezi první náležitost při tvorbě ontologií patří definování URI jmenného prostoru. Z principu se má jednat o takové URI, které bude perzistentní a které, když bude např. dereferencováno ve webovém prohlížeči, bude zobrazen popis ontologie. Jako příklad lze uvést URI jmenného prostoru ontologie FOAF, jejíž URI je `http://xmlns.com/foaf/0.1/`. Pokud na toto URI přistoupíme v rámci webového prohlížeče, bude nám nabídnut popis ontologie v pro člověka srozumitelné podobě. Navíc je vhodné poznamenat, že bude zobrazen webový dokument s jejím popisem, jehož Uniform Resource Locator (URL) se liší od URI identifikující samotnou ontologii.

K tomuto přesměrování se v praxi využívá strategie označovaná 303 URIs. Tato strategie spočívá v tom, že klient zasílá serveru HTTP požadavek na dané URI, přičemž v HTTP hlavičce sděluje, v jaké reprezentaci (zda preferuje např. RDF dokument nebo HTML dokument) chce získat požadovaná data. Server na základě tohoto požadavku odpovídá klientovi, kterému zasílá HTTP odpověď, jež mj. obsahuje URI dokumentu obsahující popis daného zdroje. Klient následně provádí poslední HTTP požadavek na toto nové URI a server zasílá požadovaný dokument.

K zajištění toho, aby bylo URI ontologie stále neměnné a přitom postupem času nedošlo při jeho dereferencování k nenalezení popisu dané ontologie, lze využít službu Online Computer Library Center (OCLC) Persistent Uniform Resource Locators (PURL)[14]. Tato služba umožňuje registraci perzistentního URL a konfiguraci přesměrování na URL obsahující popis daného zdroje. Perzistentní URL je tedy spravováno samotnou službou, kdežto popis se může nacházet kdekoli jinde. Pokud následně nastane situace, kdy je potřeba změnit URL obsahující tento popis, stačí pouze v rámci služby upravit danou adresu. Stojí za to také zmínit, že tuto službu využívá např. i slovník DCMI.

[14] http://purl.org/

2.5.2.2 Vlastní návrh

Pro ontologii bylo na základě výše zmíněného navrženo URI jmenného prostoru `http://purl.org/multimedia-asset-management#`, avšak k registraci u dané služby v rámci této práce nedojde. Pro ontologii byl také navržen preferovaný prefix `ma`, což je v samotné ontologii definováno pomocí patřičného predikátu ze slovníku VANN [34] s jmenným prostorem `http://purl.org/vocab/vann/`, který lze využít pro anotaci slovníků. Tyto informace se zapisují typicky do hlavičky ontologie, pro kterou se používá třída `owl:Ontology` ze slovníku OWL (viz kapitola 1.1.3.2). Přehled využívaných slovníků v samotné ontologii uvádí tabulka 2.1.

Tabulka 2.1: Přehled využívaných slovníků ve vlastní ontologii

Prefix	**Jmenný prostor**
rdf	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
rdfs	`http://www.w3.org/2000/01/rdf-schema#`
owl	`http://www.w3.org/2002/07/owl#`
skos	`http://www.w3.org/2004/02/skos/core#`
vann	`http://purl.org/vocab/vann/`
xsd	`http://www.w3.org/2001/XMLSchema#`
dcterms	`http://purl.org/dc/terms/`
schema	`http://schema.org/`
sioc	`http://rdfs.org/sioc/ns#`
foaf	`http://xmlns.com/foaf/0.1/`
ma	`http://purl.org/multimedia-asset-management#`

Na základě analýzy je třeba specifikovat třídy pro jednotlivé typy článků. V ontologii pro ně proto definujeme nové třídy, v jejichž definici však uvedeme, že jsou podtřídou třídy `schema:Article`. Z toho mj. vyplývá, že každá instance nově definované třídy bude zároveň instancí této obecnější třídy. Ukázka 2.1 demonstruje definici třídy `ma:AudioArticle` zapsanou v syntaxi Turtle, jež představuje článek s audio nahrávkou.

Ukázka 2.1 Definice třídy pro audio článek

```
1  ma:AudioArticle a owl:Class ;
2    rdfs:subClassOf schema:Article, [
3      a owl:Restriction ;
4      owl:minCardinality "1"^^xsd:nonNegativeInteger ;
5      owl:onProperty schema:audio ] ;
6    rdfs:label "Audio článek"@cs,
7               "Audio article"@en ;
8    rdfs:comment "An audio article, that contains an audio that someone wants to
         play"@en ;
9    rdfs:isDefinedBy ma: .
```

Definice třídy obsahuje také omezení, které vymezuje, že článek tohoto typu musí minimálně jednu audio nahrávku zahrnovat. K tomu je použita třída `owl:Restriction`, pomocí které je definováno omezení mohutnosti na predikát `schema:audio` v kombinaci predikátu `owl:minCardinality`. Horní limit kardinality není uveden kvůli případné znovupoužitelnosti např. jinými systémy, které by mohly v článku zahrnovat více audio nahrávek. Ostatní typy článků jsou definovány podobným způsobem.

V ontologii dále definujeme predikát `ma:enableComments` pro určení toho, zda je v rámci článku povoleno přidávat komentáře. Objektem daného tvrzení je logická hodnota, jedná se tedy o `owl:DatatypeProperty`. Dále pomocí predikátu `rdfs:domain` určíme příslušnost predikátu k třídě `schema:Article` a užitím predikátu `rdfs:range` definujeme rozsah `xsd:boolean`.

Predikát `ma:available`, určující dostupnost zdroje v rámci systému, je definován jako `rdfs:subPropertyOf` vlastnosti `dcterms:available` ze slovníku DCTERMS, která má definovaný rozsah `rdfs:Literal`. Z toho se může zdát, že bychom mohli přímo použít vlastnost `dcterms:available`, ovšem v jejím popisu je mj. uvedeno, že užití této vlastnosti by mělo označovat datum, kdy se daný zdroj stal/stane dostupným. V definici vlastního predikátu, jenž je opět `owl:DatatypeProperty`, je tak vymezen rozsah pouze na `xsd:boolean` a upraven popisek. Příslušnost k třídě není uvedena z důvodu lepší znovupoužitelnosti predikátu. Zde je vhodné připomenout, že pokud má predikát ve své definici zahrnut predikát `rdfs:domain` určující příslušnost k nějaké třídě, znamená to, že zdroj popsaný tímto predikátem je implicitně instancí dané třídy. To však v praxi může být limitující a v rámci lepší znovupoužitelnosti není příslušnost k třídě u tohoto predikátu definována.

K zaznamenání metadat o poslední změně článku, mediálního objektu nebo uživatelského účtu, je navrženo použití třídy `dcterms:ProvenanceStatement` ze slovníku DCTERMS. Instance této třídy bude uchovávat datum provedené úpravy a kdo danou úpravu provedl. Pro druhý uvedený údaj se jako predikát nabízí použít přímo `dcterms:contributor`, ovšem lepším řešením je zavést specifičtější vlastnost. V ontologii definujeme predikát `ma:lastContributor` jako `rdfs:subPropertyOf` dané vlastnosti ze slovníku DCTERMS, přičemž vlastnosti definujeme příslušnost k třídě `dcterms:ProvenanceStatement` a rozsah třídu `foaf:Person` z ontologie FOAF. Objektem tvrzení s tímto predikátem bude reprezentovat jiný zdroj, ne literál, jako tomu bylo v předchozích dvou případech. Definice proto zahrnuje, že se jedná o `owl:ObjectProperty`.

Ontologie mohou obsahovat také instance již existujících tříd, které jsou někdy označovány jako jedinci nebo členové dané třídy. Pro uživatelské role definujeme v ontologii `ma:Member`, `ma:Editor` a `ma:Administrator` jako členy třídy `sioc:Role`, která se nachází v ontologii SIOC [30]. Uživatelský účet `sioc:UserAccount` lze posléze s těmito rolemi propojit pomocí predikátu `sioc:has_function`, jehož rozsahem je právě třída `sioc:Role`.

V ontologii dále definujeme koncepty pro stavy článků. Pro jejich definici využíváme slovník SKOS, který se používá pro tvorbu taxonomií. V ontologii nejdříve vytvoříme schéma `ma:ArticleStatusScheme` pro dané koncepty jako instanci třídy `skos:ConceptScheme`. To demonstruje ukázka 2.2.

Ukázka 2.2 Definice schématu pro koncepty reprezentující stavy článků

```
1  ma:ArticleStatusScheme a skos:ConceptScheme ;
2    rdfs:label "Article status scheme"@en ;
3    rdfs:comment "Concept scheme for description of article status types"@en ;
4    rdfs:isDefinedBy ma: .
```

Tato definice však představuje pouze prázdné schéma. Je nutné definovat samotné koncepty jako instance třídy `skos:Concept`. V jejich definici je třeba také uvést, k jakému schématu konceptů jsou vztaženy, a to pomocí predikátu `skos:inScheme` s hodnotou `ma:ArticleStatusScheme`. Příklad definice takového konceptu ilustruje ukázka 2.3.

Ukázka 2.3 Definice konceptu pro publikovaný stav článku

```
1  ma:Published a skos:Concept ;
2    rdfs:label "Publikovaný"@cs,
3               "Published"@en ;
4    rdfs:comment "Article is published and accessible"@en ;
5    rdfs:isDefinedBy ma: ;
6    skos:inScheme ma:ArticleStatusScheme ;
7    skos:topConceptOf ma:ArticleStatusScheme .
```

Vztah mezi článkem a jeho stavem určuje predikát `ma:articleStatus`, jehož definici ilustruje ukázka 2.4. Jedná se opět o predikát, jehož hodnotu tvoří jiný zdroj, a to nějaký koncept z vytvořeného schématu. V definici predikátu je proto třeba explicitně zajistit, aby jeho rozsahem byl právě koncept ze schématu `ma:ArticleStatusScheme`. K tomu je použita třída `owl:Restriction` aplikovaná na predikát `rdfs:range`. Omezení pouze na koncepty z daného schématu je následně specifikováno dvěma tvrzeními. Jedná se o tvrzení s predikátem `owl:onProperty` na vlastnost `skos:inScheme` a tvrzení s predikátem `owl:hasValue`, jehož hodnotou má být právě definované schéma.

Ukázka 2.4 Definice predikátu pro vazbu mezi článkem a stavem článku

```
1  ma:articleStatus a owl:ObjectProperty ;
2      rdfs:label "Status článku"@cs,
3                 "Classification of article status."@en ;
4      rdfs:comment "Property that determines the status of article."@en ;
5      rdfs:range skos:Concept, [
6        a owl:Restriction ;
7        owl:hasValue ma:ArticleStatusScheme ;
```

```
8         owl:onProperty skos:inScheme ] ;
9       rdfs:isDefinedBy ma: .
```

2.5.3 Vizualizace pokrytí doménového modelu

Předchozí dvě podkapitoly se zaměřovaly na pokrytí doménového modelu pomocí ontologií. K pokrytí většiny tříd, atributů a vazeb jsou znovu použity výrazy z již existujících ontologií, některé jsou dodefinovány v ontologii vlastní. Výsledné pokrytí doménového modelu ilustruje obrázek 2.2, opět jako Unified Modeling Language (UML) diagram tříd.

Výrazy začínající prefixem `ma` jsou deklarovány ve vlastní ontologii. Orientace vazeb mezi jednotlivými třídami není uvedena, tu lze odvodit z dané ontologie podle příslušnosti predikátu k určité třídě (`rdfs:domain`) nebo také z rozsahu daného predikátu (`rdfs:range`). Diagram také nezachycuje vazby `rdfs:subPropertyOf` u vlastních predikátů, které jsou mapovány na jiné predikáty z jiných ontologií.

2.6 Návrh systému

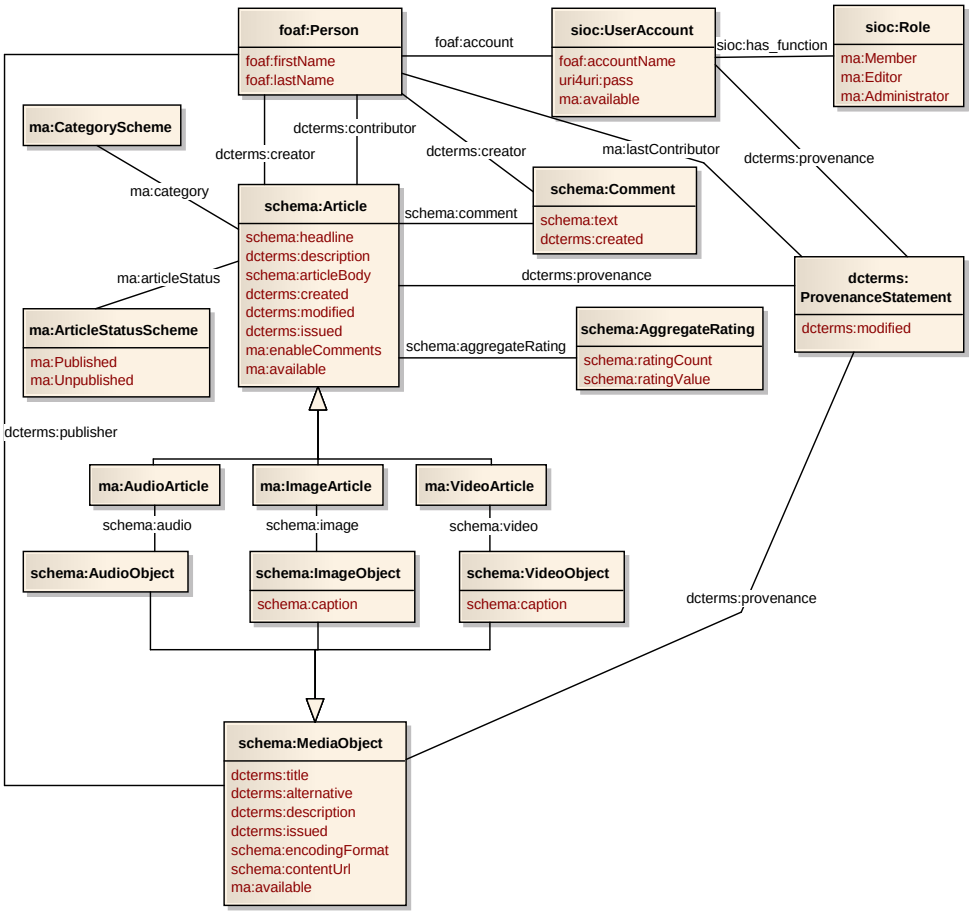
Tato část se již zabývá návrhem implementovaného prototypu. Návrh je rozdělen do několika částí, od návrhu základních kamenů datové vrstvy, přes návrh služeb až po návrh uživatelského rozhraní, který pro zjednodušení spojuje část administrační s tou pro prezentaci uživatelům, přičemž jednotlivé administrační sekce je nutno zabezpečit proti přístupu běžných uživatelů.

2.6.1 Třídy doménového modelu

Třídy doménového modelu systému vycházejí z doménového modelu uvedeného na obrázku 2.1, přičemž některé názvy hlavních entit z uvedeného modelu jsou přejmenovány, aby korespondovaly s názvem třídy v dané ontologii. Tento krok je pouze pro zvýšení přehlednosti, názvy doménových tříd systému nemusejí odpovídat názvům tříd z ontologií. Příkladem změny je entita `MediaAsset`, jež je přejmenována na `MediaObject`. K názvům všech mediálních objektů je pak připojen stejný sufix, např. v případě audio nahrávky je místo názvu třídy `Audio` používán název `AudioObject`.

Výčtové typy zachycené na obrázku 2.1, jako je stav článku nebo role uživatelů v systému, nejsou v entitních třídách navrhovaného systému realizovány přímo, ale jsou umístěny do separátních tříd, které je zabalují. Příčinou tohoto návrhu je omezení v použitém frameworku Empire, které je detailněji představeno v kapitole 3.3.1.1.

Doménový model uvedený na obrázku 2.1 mj. zachycuje hodnocení článků. V diagramu je však pouze zachycena skutečnost taková, že článek může mít



Obrázek 2.2: Pokrytí doménové modelu pomocí ontologií

nějaké hodnocení, jež uchovává informaci o celkovém počtu hodnocení a samotné hodnotě hodnocení. Pouze s tímto návrhem není zajištěno uchovávání informace o tom, jak určitý uživatel hodnotil určitý článek. Absence této informace má pak dopad např. na řešení situací, aby jeden uživatel mohl hodnotit článek pouze jednou nebo mohl své hodnocení později změnit.

Návrh systému hodnocení umožňující tuto funkcionalitu ilustruje obrázek F.4. V klasickém přístupu, tj. v přístupu nevyužívající ontologie, by tento model již mohl korespondovat i se skutečnými třídami systému. Avšak v systému založeném na ontologiích je ještě zapotřebí systém hodnocení pokrýt pomocí ontologií. K tomu je využit již často využívaný slovník Schema.org, který potřebné vyjadřovací prostředky definuje. Konkrétní pokrytí demonstruje obrázek F.5. Na rozdíl od klasického modelu na obrázku F.4 si lze povšimnout několika odlišností. Přibyla navíc nová entita, jejíž instance jsou typu `schema:Review`. Vazba mezi uživatelským hodnocením a článkem není ilustrována přímo s entitou pro článek, ale s obecnou entitou, jejíž instance jsou

typu `owl:Thing`. Ačkoliv v systému lze prozatím hodnotit pouze články, hodnocení je možné díky použitým predikátům a třídám z ontologie Schema.org snadno rozšířit i pro jiné entity, např. pro mediální objekty. Vhodné je také poznamenat, že díky použitému predikátu `schema:itemReviewed` je subjektem tvrzení entita `Review` a objektem entita, která je hodnocena.

Na základě výše uvedeného systému hodnocení a jeho následné realizace, byla v rámci doménového modelu dodatečně vytvořena nová entitní třída `Thing`, jejíž instance reprezentuje libovolný zdroj, tj. instanci třídy `owl:Thing` v sémantickém webu. Tato entita přitom neobsahuje žádné atributy kromě jednoznačného identifikátoru a její použití je vhodné v takových případech, kdy je zapotřebí pouze v datovém úložišti vytvořit tvrzení, jehož objektem je právě zdroj reprezentovaný touto třídou.

2.6.2 Uživatelský model

Uživatelský model se týká adaptivní části systému. Připomeňme, že do uživatelského modelu jsou ukládány ty informace, které o daném uživateli systém zjistil přímo či nepřímo. Může se jednat o informace získané monitorováním uživatelovy interakce, explicitně vložené uživatelem, ale také informace získané jiným systémem, které byly do systému integrovány. Na základě těchto informací pak dochází k přizpůsobení prezentovaných dat konkrétnímu uživateli.

Navržený a implementovaný prototyp systému k adaptaci využívá pouze ty informace, které jsou zadány do systému samotným uživatelem, typicky se jedná o informace získané ze stránky s nastavením. Framework ASF [24] tato data zařazuje do uživatelského profilu, přičemž každý uživatel má z principu vlastní uživatelský profil. V uživatelském profilu jsou tyto informace ukládány ve formě klíč-hodnota, přičemž klíč tvoří konstanta, která charakterizuje danou vlastnost uloženou v uživatelském profilu. Tento celek se také označuje jako atribut. Do uživatelského profilu jsou ukládány následující atributy (zvýrazněná je hodnota klíče):

- `websiteDefaultTheme` – Barevné téma vzhledu systému. Uživatel je schopen nastavit barevné téma vzhledu systému. Jako hodnota se používá název daného tématu vzhledu. Uložení barevného tématu vzhledu do uživatelského profilu je pro samotný systém vhodné řešení, pro potencionální využití této informace jinými systémy je však zapotřebí, aby porozuměly tomu, co daný název tématiky představuje. Tento problém lze řešit např. definicí seznamu barevných témat jako schéma konceptů ve vlastní ontologii, to však není zahrnuto do této práce.

- `websiteDefaultNumberOfRecordsInList` – Informace o maximálním počtu zobrazovaných záznamů na jedné stránce u stránkování. Většina dnešních webových aplikací zahrnuje nějakou část, která zobrazuje uživatelům nějaký výčet dat v podobě stránkování, uživatelům přitom může

být umožněno nastavení počtu záznamů na jedné stránce. V našem systému hodnota tohoto atributu ovlivňuje globální nastavení pro každé místo se stránkováním. Pokud tento atribut není nastaven, systém použije výchozí počet záznamů na stránku.

- `websiteDefaultPagingNavigatorPosition` – Pozice stránkovacího navigačního panelu v případě aktivního stránkování. Pozice panelu může být vzhledem k vypisovaným záznamům umístěna nahoře, vespod, resp. nahoře i vespod.

- `articleRatingValueLimitToDisplay` – Dolní limit hodnocení článku. Aby byl článek zařazen do výsledku, musí mít vyšší nebo stejné hodnocení než tato hranice.

- `articleRatingCountLimitToDisplay` – S předchozí informací souvisí další informace, a to spodní hranice počtu hodnocení článku, od které bude adaptace na základě předchozího atributu aktivní. Tato informace je nutná, protože v případě, kdy by článek neměl žádné hodnocení, automaticky by se do výsledků článek nezařazoval.

- `prefferedArticleType` – Uložení preferovaného typu článku. Uživatelům se budou zobrazovat jen články tohoto typu.

- `prefferArticleAllowingComment` – Atribut slouží pro vypisování pouze článků, u kterých je povoleno přidávání nových komentářů. Skutečnost, že článek již nějaké komentáře zahrnuje, ale přidávání nových komentářů je později zakázáno, nemá pro aplikaci tohoto atributu na výpis článků vliv.

Stojí za to poznamenat, že pro výše uvedené atributy není zapotřebí navrhovat datové typy, ať už pro jednotlivé klíče atributů nebo jejich hodnoty, neboť to je dáno použitým frameworkem ASF. Ačkoliv framework ASF implementuje metody pro uložení např. logických nebo numerických hodnot, vždy jsou ukládány ve formě textových řetězců.

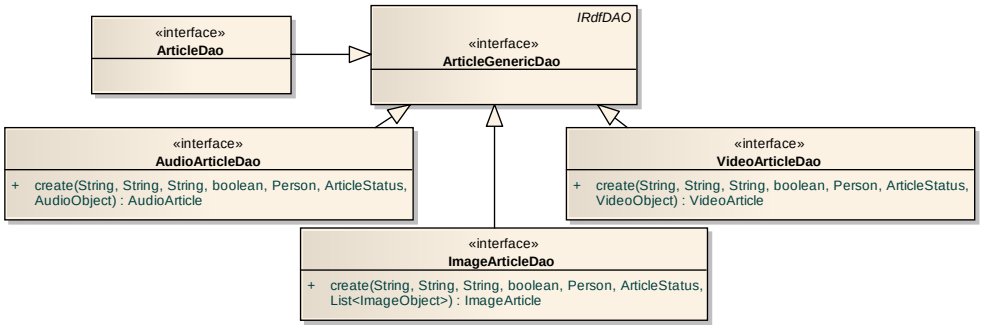
Strukturu uložení atributů ve formě trojic také určuje framework ASF, resp. jím využívaná ontologie GOMAWE. Uživatelský profil je sice v systému reprezentován jako jeden objekt, který poskytuje rozhraní pro přístup k jednotlivých atributům, resp. k jejich uložení, nicméně každý atribut je v RDF datech reprezentován jako samostatný zdroj, který je přímo asociován s určitým uživatelem.

2.6.3 DAO objekty

Přístup k datům v datovém úložišti je realizován pomocí Data Access Object (DAO) objektů využívající návrhový vzor DAO. Tento vzor je možné implementovat různými způsoby, navržené řešení využívá rozhraní s generickým

typem. Toto rozhraní definuje základní CRUD[15] operace pro generický typ, načež je implementováno generickou abstraktní třídou. Z této třídy potom dědí DAO objekty pro jednotlivé entitní typy.

Systém je navržen takovým způsobem, že vytváření každé nové instance doménové entity je realizováno daným DAO objektem. Obrázek 2.3 ilustruje navrženou hierarchii pro jednotlivé typy článků, generické rozhraní `IRdfDAO` je součástí frameworku ASF.



Obrázek 2.3: Návrh hierarchie DAO pro všechny typy článků

Návrh dává možnost snadného rozšíření v případě nových typů článků. Podobná hierarchie je vytvořena i pro jednotlivé typy multimediálních objektů. Popisem implementace DAO se stručně zabývá také část realizace.

2.6.4 DTO objekty

Systém využívá tzv. Data Transfer Object (DTO) objektů. Tyto objekty zapouzdřují data entitních tříd, také je do jisté míry zjednodušují. Objekty zahrnují typicky stejné atributy jako entitní doménové třídy a slouží k poskytnutí samotných dat zobrazovacím šablonách nebo naopak k jejich sběru. Důvodem jejich zavedení je mj. odstranění přímé závislosti webové vrstvy a zobrazovacích šablon na doménových entitních třídách. Tyto objekty již také nevyužívají dědičnost, přičemž v případě jednotlivých článků se využívá jeden velký DTO objekt, v případě multimediálních objektů má každý typ vlastní DTO objekt.

2.6.5 SPARQL dotazy pro seznamy

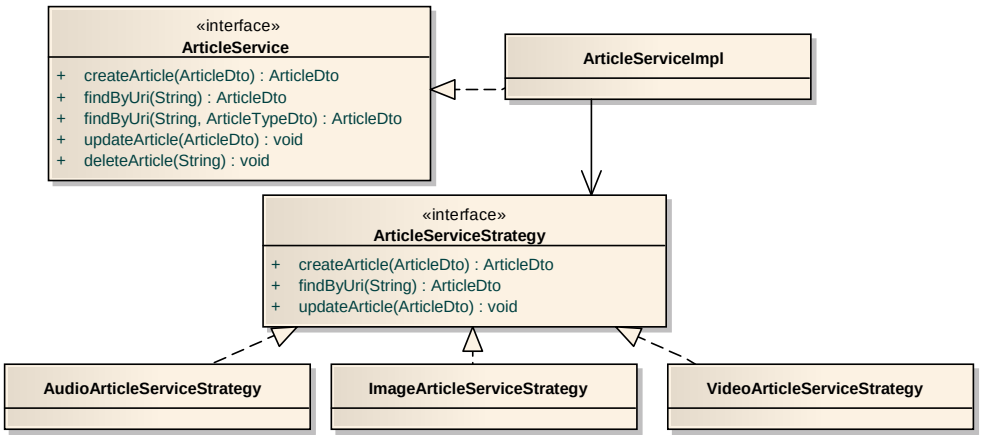
Systém zahrnuje několik sekcí, které zahrnují dlouhé výpisy záznamů v tabulkové podobě, např. výpis multimediálních objektů, článků atp. Pro jednotlivé sekce jsou navrženy vlastní třídy, které se dotazují pouze na ta data, která jsou pro danou sekci potřeba. K tomu se využívá speciálních tříd z frameworku ASF a více se tomuto věnuje část realizace 3.3.5.

[15] Operace pro vytváření, čtení, editaci a mazání.

2.6.6 Služby

Servisní vrstva zahrnuje hlavní aplikační logiku a slouží jako prostředník mezi datovou a webovou vrstvou. Jednotlivé služby pro přístup k datům využívají definovaná rozhraní DAO objektů, přičemž data poté poskytují skrz vlastní rozhraní webové vrstvě. Uvnitř služeb se také provádí mapování mezi DTO objekty a doménovými entitními třídami, které reflektují použité ontologie. Z hlediska návrhu systému je vrstva se službami vhodným místem, kde lze do jisté míry zjednodušovat vazby, které jsou definované v ontologiích, a to např. z důvodu předání dat zobrazovacím šablonám.

Jedna z hlavních služeb systému je určena pro operace s články. Základní návrh této služby znázorňuje obrázek 2.3. Služba implementující rozhraní `ArticleService` na základě typu článku, který je určen výčtovým typem v daném DTO objektu, nastaví strategii, podle které se vykoná operace specifická pro daný typ článku. Příkladem může být tvorba nového článku, kde každý typ článku využívá rozhraní svého DAO objektu, jak bylo naznačeno na obrázku 2.3.



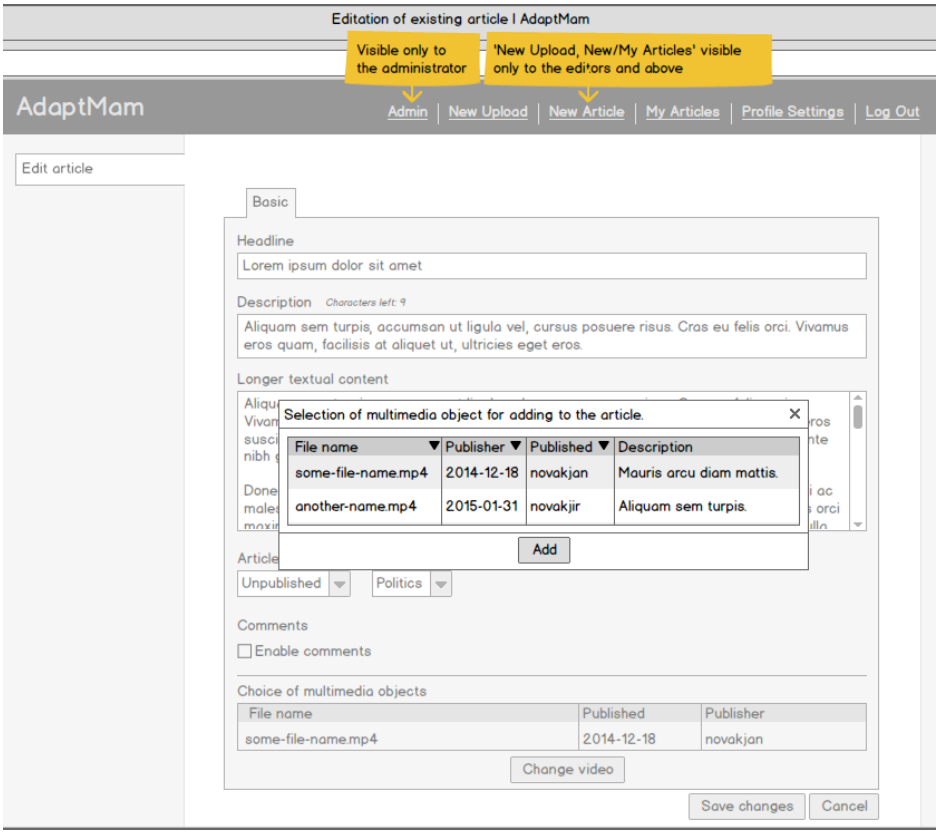
Obrázek 2.4: Návrh služby pro entity reprezentující články

2.6.7 Návrh uživatelského rozhraní

Prvotní návrh uživatelského rozhraní probíhal pomocí lo-fi prototypů, a to téměř pro každou obrazovku implementovaného systému. K návrhu byl použit nástroj Balsamiq Mockups[16], který např. umožňuje propojení jednotlivých navržených obrazovek pomocí odkazů a lze tak již v rámci této fáze návrhu průchodem obrazovek simulovat základní použitelnost uživatelského rozhraní.

V návrhu uživatelského rozhraní bylo zohledňováno umístění jednotlivých sekcí systému vzhledem k uživatelským rolím. Všechny administrační sekce,

[16] https://balsamiq.com/



Obrázek 2.5: Ukázka návrhu uživatelského rozhraní – editace článku

tj. sekce pro uživatele s rolí editora nebo administrátora, byly během implementace zabezpečeny proti přístupu běžných uživatelů. Při návrhu byl pak kladen důraz na jednoduché ovládání systému, kde všechny hlavní funkce jsou dostupné z hlavního navigačního panelu v záhlaví systému, případné specifické části jsou dostupné z postranního kontextového menu. Ukázku návrhu obrazovky jedné z administračních obrazovek ilustruje obrázek 2.5.

2.7 Model nasazení

Základní popis umístění jednotlivých komponent na jednotlivé fyzické uzly a způsob jejich komunikace je znázorněn obecnějším modelem nasazení na obrázku F.6. Celý systém tvoří distribuovatelný webový archiv, který je nasazen na aplikačním serveru. Na aplikační server je třeba také nasadit Sesame Server, jenž poskytuje přístup do datových repositářů Sesame.

Kapitola 3

Realizace

Tato kapitola popisuje určité části realizace. V samotném úvodu jsou nejprve stručně popsány použité hlavní technologie a frameworky, jako je framework Spring, JSF, Empire a v neposlední řadě také framework Adaptive System Framework (ASF). Dále následuje stručný popis nastavení celého projektu, včetně nastavení datového úložiště pro RDF data. Zbývající část kapitoly se věnuje specifickým částem implementační části a také řešením některých problémů.

3.1 Použité technologie a hlavní frameworky

Systém je napsán v programovacím jazyku Java, který lze určitě zařadit mezi jeden z nejpopulárnějších programovacích jazyků v dnešní době. Volba programovacího jazyka vyplynula přímo ze zadání, resp. z v něm uvedených technologií. Samotný systém je postaven na platformě Java Platform Enterprise Edition (Java EE).

Jelikož je využívána celá řada různých větších či menších frameworků, resp. knihoven, bylo by velmi problematické spravovat je v rámci projektu manuálně. Proto je využíván nástroj Apache Maven[17], který slouží k usnadnění sestavování aplikací a také k již zmíněné správě knihoven tzv. třetích stran. Klíčovou součást nástroje Maven představuje soubor „pom.xml“, typicky umístěným v kořenovém adresáři projektu. Tento soubor představuje Project Object Model (POM) popisující projekt jako objekt. Do tohoto souboru zapisujeme mj. základní informace o projektu a závislosti na knihovnách třetích stran. Uvedené knihovny jsou následně typicky hledány v tzv. centrálním repositáři, a to v případech, kdy se nenacházejí v repositáři lokálním, do kterého jsou případně z centrálního repositáře staženy. Může však nastat situace, kdy se knihovna v centrálním repositáři nenachází, např. není-li žádoucí vystavovat vytvořenou knihovnu veřejně. I v tomto případě lze stále uvést

[17] http://maven.apache.org/

danou závislost do souboru „pom.xml“, je ale nutné doinstalovat danou knihovnu do lokálního repositáře manuálně[18]. Ukázka 3.1 demonstruje uvedení jedné závislosti na knihovně v souboru „pom.xml“.

Ukázka 3.1 Ukázka vložení knihovní závislosti do souboru pom.xml

```
1 <dependencies>
2   <dependency>
3     <groupId>cz.cvut.fel.asf</groupId>
4     <artifactId>asf-core</artifactId>
5     <version>${asf.version}</version>
6   </dependency>
7 </dependencies>
```

Během samotné implementace systému bylo využíváno jako Integrated Development Environment (IDE) open-source prostředí NetBeans[19], které práci s nástrojem Maven podporuje. Výhoda použití Maven však spočívá také v tom, že celý tzv. projekt není závislý přímo na vývojovém prostředí, a tudíž ho lze snadno integrovat do jiných vývojových prostředí, které nástroj Maven podporují.

Distribuovatelný webový archiv zahrnující celý systém je třeba také nasadit na aplikační server. V rámci projektu používáme open-source aplikační server GlassFish. V zásadě by mělo být většinou jedno, jaký aplikační server je použit, nicméně kvůli nahrávání multimediálních souborů na diskový oddíl serveru, resp. konfiguraci následného servírování statického obsahu, to neplatí zcela. Konfigurace se totiž u různých aplikačních serverů provádí různě, přičemž v rámci tohoto projektu je řešena pouze pro aplikační server GlassFish, který je také instalován na serveru určeného pro testování.

3.1.1 Spring

Spring Framework [35] je open-source framework pro vývoj (nejen) webových aplikací v programovacím jazyku Java. Je modulární, zahrnuje zhruba 20 dílčích modulů, přičemž v projektu lze využít pouze takové, které jsou potřeba.

Základní vlastností frameworku je odstínění tvorby závislostí, resp. provázání programových objektů z aplikace na framework, k čemuž je využíván návrhový vzor Inversion of Control (IoC), resp. Dependency Injection (DI). Objektové závislosti lze ve frameworku spravovat různými způsoby, ať už pomocí definicí v konfiguračních XML souborech nebo užitím Java anotací. Velmi důležitý je tzv. aplikační kontext, ve kterém jsou jednoduše řečeno udržovány všechny objekty spravované frameworkem Spring.

Pro zabezpečení systému, tj. autorizaci a autentizaci, byl vzhledem k použití frameworku Spring přirozeným způsobem vybrán framework Spring Secu-

[18] http://maven.apache.org/guides/mini/guide-3rd-party-jars-local.html
[19] https://netbeans.org/

rity [36]. Jednou z výhod takto zabezpečených aplikací je dle [36] jejich snadná přenositelnost napříč různými cílovými prostředími.

V začátcích vývoje byla využívána také technologie Spring Web Flow [37]. Tato technologie je určena pro implementaci navigačních toků (z angl. flows), resp. přechodů mezi jednotlivými stránkami. Od této technologie ale bylo z několika důvodů upuštěno, zejména kvůli problému souvisejícím s přechody mezi stránkami užitím běžných HTML odkazů (ne pomocí tlačítek) a kvůli problému s tzv. bookmarkingem URL adres.

3.1.2 Empire

Empire [38] je implementací JPA. Na rozdíl od např. Hibernate[20], což je jedna z implementací JPA pro objektově relační mapování, je Empire implementací pro sémantická datová úložiště, resp. RDF data. Jedním z jeho cílů je zjednodušit vývojářům práci s RDF daty uvnitř aplikací, které s těmito daty operují, potažmo je odstínit od detailní znalosti RDF. K dotazování se nad RDF daty je podporován jazyk SPARQL.

Základní použití frameworku Empire dobře popisuje [39] a je také představeno v kapitole 3.3. Stojí za to také zmínit, že framework Empire interně využívá Sesame API, což je také jeden z hlavních důvodů, proč je jako datové úložiště zvoleno právě Sesame.

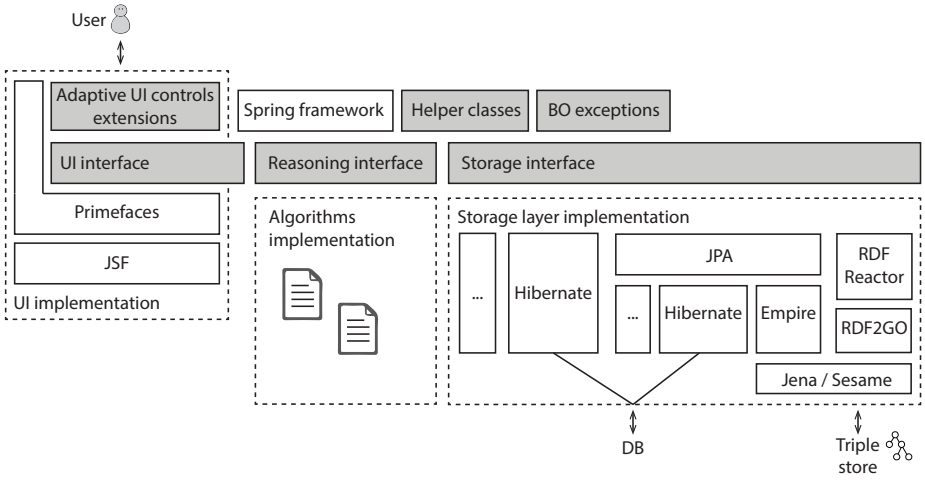
3.1.3 Adaptive System Framework

Framework Adaptive System Framework (ASF) je akademický projekt, jenž má sloužit zejména k usnadnění tvorby adaptivních webových aplikací. Zahrnuje však i řešení pro různé problémy, které je třeba řešit i v neadaptivních webových aplikacích a lze ho tak využít i pro tvorbu běžných aplikací. ASF je postaven na teoretickém modelu GOMAWE (viz kapitola 1.3.1) a jeho obecnou architekturu ilustruje obrázek 3.1.

ASF framework byl navrhován tak, aby byl obecně použitelný. Proto je rozdělen do několika komponent s vlastními odpovědnostmi, poskytuje řadu rozhraní na úrovni různých vrstev potencionálního systému, přičemž jejich implementace by měla být postavena na rozšířených a známých technologiích [24]. Např. vrstva uživatelského rozhraní je založena na technologii JSF a využívá knihovny PrimeFaces.

Framework ASF, který byl pro tuto práci poskytnut, je rozdělen v zásadě na tři hlavní moduly, a to ASF.Core, ASF.Persistence a ASF.Web.Primefaces. Modul ASF.Core je základní modul frameworku, jenž zahrnuje řadu rozhraní a generických tříd pro adaptivní algoritmy, DAO objekty nebo dotazy podporující stránkování, resp. řazení, ale také zde nalezneme různé pomocné utility. Dále se zde nachází třídy pro reprezentaci uživatelského modelu, resp. uživatelského profilu a třída pro jejich načtení, která je implementována dle

[20] http://hibernate.org/



Obrázek 3.1: Obecná architektura frameworku ASF [24]

návrhového vzoru Singleton. Další moduly pak využívají některé prvky tohoto modulu.

Modul ASF.Persistence představuje rozhraní pro uložení dat a obsahuje několik různých implementací v závislosti na použité technologii uložení dat. Např. ASF.Persistence.Hibernate pro framework Hibernate, zejména pak pro tuto práci důležitější ASF.Persistence.Empire pro framework Empire. Součástí tohoto modulu je mj. třída `EmpireJPAPersistenceProvider` implementující rozhraní `PersistenceProvider`. Tato třída vznikla v rámci práce [40], v rámci které také došlo k přizpůsobení frameworku Empire takovým způsobem, aby mohl být použit s frameworkem ASF. Součástí tohoto modulu jsou také třídy pro reprezentaci uživatelského modelu, resp. uživatelského profilu ve formě RDF, přičemž pro anotaci využívají ontologii GOMAWE.

Poslední modul, ASF.Web.Primefaces, obsahuje řadu utilit, které lze využít při tvorbě webové vrstvy aplikací, resp. jejich uživatelského rozhraní. Obsahuje např. logiku pro zachytávání chyb vzniklých v aplikaci, případně dokonce k jejich odeslání na email. Pokud se jedná o specifickou chybu frameworku ASF, a to `BusinessCheckException` definovanou v modulu ASF.Core, dochází k automatickému zobrazení chybového hlášení přímo klientovi. K tomu využívá technologii JSF, resp. PrimeFaces.

3.1.4 Java Server Faces

Java Server Faces (JSF) je standardizovaná Java technologie, resp. komponentově orientovaný framework pro tvorbu webových aplikací. Usnadňuje tvorbu uživatelského rozhraní, potažmo jeho integraci s aplikační logikou. Stručně řečeno, uživatelské rozhraní je vytvářeno pomocí speciálních značek z různých knihoven, přičemž data jsou poskytována z Java tříd a naopak. Pod technolo-

gii JSF lze zařadit také dvě systémem využívané knihovny třetích stran, a to PrimeFaces a OmniFaces.

PrimeFaces[21] je velmi oblíbená open-source knihovna pro JSF, jež zahrnuje početnou množinu hotových User Interface (UI) komponent. Knihovna je založena na standardu JSF 2.0 Asynchronous JavaScript and XML (AJAX) API [41], pro korektní funkčnost komponent je tak nutné mít zapnutý JavaScript. Knihovně PrimeFaces konkuruje knihovna RichFaces[22], vzhledem k využití knihovny PrimeFaces jedním z modulů ASF frameworku byla zvolena knihovna PrimeFaces.

OmniFaces[23] je také knihovna pro JSF, od výše zmíněných se však nezaměřuje na tvorbu bohatých UI komponent, ale na řešení častých problémů spojených s drobnými nedostatky v implementaci JSF API.

3.2 Konfigurace projektu a obecné informace

Pro realizaci celého projektu byla zapotřebí poměrně rozsáhlá konfigurace a řada různých nastavení, ať už se jedná o uvedení všech knihoven třetích stran a jejich správných verzí do zmíněného souboru „pom.xml“ nebo o konfiguraci všech technologií pro zajištění jejich správné kooperace. Tato podkapitola uvádí pouze stručný popis základních nastavení, některá další jsou případně uvedena v rámci konkrétní části realizace.

Mezi jeden z hlavních konfiguračních souborů každé standardní webové aplikace patří tzv. deployment deskriptor „web.xml“, nacházející se typicky v adresáři „WEB-INF“. Tento konfigurační soubor obsahuje zejména nastavení servletů, listenerů, filtrů a kontextových parametrů. Jeden z nejdůležitějších kontextových parametrů je zde `contextConfigLocation` s hodnotou, která udává umístění hlavního konfiguračního souboru Spring Frameworku, a to „application-context.xml“ nacházejícím se v adresáři „WEB-INF“. Spolu s ním je třeba nastavit také listener `ContextLoaderListener` z frameworku Spring. Dále pouze uveďme, že se zde nachází např. konfigurace filtru pro Spring Security nebo upload souborů.

Konfigurační soubor „application-context.xml“ zahrnuje všechna hlavní nastavení frameworku Spring, resp. celého projektu. V zájmu zvýšení přehlednosti jsou v tomto souboru importovány další konfigurační soubory pro jednotlivé technologie, jak uvádí ukázka 3.2. Kromě těchto vložení se zde mj. také nachází dvě důležitá nastavení, a to skenování tříd pro jejich automatické zavedení do aplikačního kontextu (řádek č. 11) a aktivování anotací uvnitř tříd, které jsou v aplikačním kontextu již zavedeny (řádek č. 13). Dle [35] se, v rámci dané verze frameworku Spring, již druhé nastavení uvádět nemusí, neboť je implicitně zajištěno prvním uvedeným.

[21] http://primefaces.org
[22] http://richfaces.org/
[23] http://omnifaces.org/

Ukázka 3.2 Import jednotlivých konfiguračních souborů v aplikačním kontextu

```
1  ...
2  <!-- Import configuration files (such as datasource, security etc.) -->
3  <import resource="datasource-config.xml" />
4  <import resource="mail-config.xml" />
5  <import resource="asf-config.xml" />
6  <import resource="security-config.xml" />
7  <import resource="locale-config.xml" />
8  <import resource="dozer-config.xml" />
9
10 <!-- Enable scanning packages to find and register beans -->
11 <context:component-scan base-package="cz.cvut.fit.adaptmam" />
12 <!-- Activate annotations in beans already registered in app. context -->
13 <context:annotation-config />
```

Aby však automatické skenování tříd mělo účinek, je zapotřebí hlavičky požadovaných tříd anotovat pomocí specifických anotací frameworku Spring, např. anotací `@Service` v rámci servisní vrstvy. Aktivace anotací uvnitř tříd má pak význam např. pro anotaci `@Autowired`, díky které dochází k automatickému provázání objektu, již nacházejícím se v aplikačním kontextu, s takto anotovanou proměnnou, a to na základě jejího typu.

Nastavení frameworku Spring a JPA, resp. spojení s datovým úložištěm se konfiguruje v souboru „datasource-config.xml“, resp. v „persistence.xml“. Druhý jmenovaný se nachází standardně v adresáři „META-INF“ a obsahuje definici jedné[24] perzistentní jednotky (z angl. persistence unit). Ta obsahuje mj. parametry pro spojení s datovým úložištěm a také zde definujeme, že se jako implementace rozhraní `PersistenceProvider` bude používat zmiňovaná třída `EmpireJPAPersistenceProvider`.

Integrace frameworku Spring a frameworku JSF se provádí hlavně v souboru „faces-config.xml“, kde se zavádí třída `SpringBeanFacesELResolver` z frameworku Spring, jakožto implementace rozhraní `ELResolver`. Tím docílíme toho, že můžeme v prezentačních šablonách využívat služeb tříd zavedených v aplikačním kontextu. Spolu s touto konfigurací ještě registrujeme listener `RequestContextListener` v souboru „web.xml“.

3.2.1 Zabezpečení systému

Zabezpečení systému je realizováno pomocí zmíněného frameworku Spring Security [36], jenž je konfigurován v souboru „security-config.xml“. Je zde konfigurována autorizace a autentizace, ale také šifrování[25] komunikace mezi klientem a serverem pomocí protokolu Hypertext Transfer Protocol Secure (HTTPS).

[24] Pro jednotkové testy je definován vlastní aplikační kontext i perzistentní jednotka ve zdrojích pro testy.

[25] Přidáno až na základě návrhu na zlepšení od jednoho z účastníků závěrečného testování.

Autorizací je myšleno zabezpečení jednotlivých sekcí v systému, a to na základě uživatelské role osoby operující se systémem. V tomto souboru se nachází zabezpečení v kontextu URL adres, v samotných šablonách jsou pak zabezpečeny případné menší fragmenty obsahu, a to např. jeho skrytí neautorizovaným uživatelům.

Autentizace spočívá v ověření uživatelovy identity. Uživatel je typicky vyzván k zadání přihlašovacích údajů, které systém ověří a přidělí případné role, načež je vpuštěn do systému. Realizovat autentizaci lze v Spring Security různými způsoby. V rámci realizace v konfiguračním souboru zavádíme autentizační manager `ProviderManager` z frameworku Spring Security, jemuž předáváme instanci autentizačního poskytovatele `DaoAuthenticationProvider`, také z frameworku Spring Security. Tomu dále nastavujeme vlastní implementaci rozhraní `UserDetailsService` a typ hašovacího algoritmu pro zabezpečení ukládaných hesel. Systém využívá algoritmus bcrypt [42], jehož implementace je součástí frameworku Spring Security. Implementace procesu autentizace je pak realizována ve vlastní servisní třídě, která implementuje vlastní rozhraní `PersonAuthenticationProviderService`.

3.2.2 Nastavení aplikačního serveru pro multimédia

Uživatelé systému, resp. uživatelé s rolí editora a administrátora, mohou nahrávat multimediální soubory přímo v rámci systému. Tyto soubory se ukládají na diskový oddíl serveru, přičemž do datového úložiště se ukládají pouze metadata k danému souboru.

Multimédia jsou nahrávána do standardního adresáře „docroot“ aplikačního serveru GlassFish, a to v doméně, ve které je nasazen samotný systém. Multimédia jsou v rámci tohoto adresáře navíc ukládána do dynamicky vytvářených podadresářů (pokud neexistují), a to dle typu daného média. Je vhodné poznamenat, že ukládání souborů do adresáře „docroot“ má své výhody a nevýhody.

Výhoda spočívá např. v tom, že můžeme snadno pracovat s relativními cestami a při standardním nastavení serveru GlassFish není v případě přenosu systému (např. z Windows na Linux) vyžadována žádná další úprava konfigurace. Nevýhoda je mj. ta, že jsou soubory ukládány přímo do umístění aplikačního serveru. V konfiguračním souboru „glassfish-web.xml“ je proto provedeno nastavení tzv. alternativních „docroot“ adresářů, díky kterým lze servírovat[26] staticky uložené zdroje i mimo umístění aplikačního serveru. Ukázka nastavení je demonstrována v [43], kde je dobře vysvětlen i celý princip.

3.2.3 Datové úložiště a volba repositáře

Jako datové úložiště pro RDF data je použito open-source datové úložiště Sesame, přičemž komunikace mezi systémem a datovým úložištěm probíhá

[26] Zpracování HTTP požadavku na dané URL.

přes HTTP protokol, resp. OpenRDF Sesame Server. Spojení je nastaveno v souboru „persistence.xml“, kde je uveden, kromě URL k Sesame Server a dalších parametrů, i název repositáře, do kterého jsou data ukládána.

Sesame nabízí dle dokumentace[27] několik typů repositářů, v rámci této práce byly použity dva z nich, a to „Native Store“ a „In Memory Store“. V případě první varianty jsou data ukládána na diskový oddíl, v případě druhé se pracuje s daty v operační paměti. Více informací lze nalézt v příloze B, avšak zde je důležité poznamenat, že zmíněné typy repositářů je možné vytvořit s různými podtypy, přičemž volba typu má pak dopad nejen na výkon, psaní SPARQL dotazů, ale také na správnou funkčnost použitého frameworku Empire. Pro demonstraci uveďme následující porovnání repositářů typu „Native Java Store“, resp. „Native Java Store RDF Schema“.

- „Native Java Store“ zahrnuje pouze taková tvrzení, která jsou v RDF datech přímo deklarována. Tento repositář není sám o sobě schopen obohatit data o žádná další tvrzení. Vložíme-li do repositáře např. data z ukázky 1.1 spolu s ontologií FOAF, repositář není schopen data obohatit o tvrzení, že v dané ukázce popisovaný subjekt je i instancí třídy `foaf:Agent`, jíž je třída `foaf:Person` podtypem.

- „Native Java Store RDF Schema“ předchozí obohacení dat provádí a v popisu daného subjektu lze tento fakt nalézt. Podobným způsobem jsou data obohacována i v rámci predikátů. To znamená, že po vložení RDF trojice obsahující predikát `X`, který je dílčí vlastností predikátu `Y`, dojde k obohacení dat o další tvrzení. Toto nové tvrzení má jako predikát právě `Y`, subjekt i objekt je samozřejmě stejný.

Během realizace bylo testováním zjištěno, že použití druhého uvedeného typu repositáře má negativní vliv na práci s použitým frameworkem Empire. Při vyhledání subjektu z datového úložiště jsou trojice mapovány na atributy entitní třídy (podle anotací, bude uvedeno dále v části 3.3.1). Pokud se v RDF datech o určitém subjektu vyskytuje více tvrzení se stejným predikátem, ale různým objektem, dojde při mapování na daný atribut entitní třídy k chybě. Tuto situaci lze řešit v mnohých případech použitím kolekce, ale to nám nedává vůbec žádnou kontrolu nad tím, s jakými daty dále v systému pracujeme. Navíc mohou nastat další problémy v případech, kdy se v rámci jedné entitní třídy používají zároveň dva (nebo více) atributů, jenž je jeden anotován predikátem, který je dílčí vlastností predikátu uvedeného v anotaci druhého atributu.

Kvůli výše uvedenému, implementovaný prototyp systému pracuje s daty uloženými v repositáři typu „Native Java Store“. To má určitý vliv na psaní SPARQL dotazů. Např. pokud bychom se chtěli dotázat na všechny zdroje, které jsou instancí třídy `foaf:Agent`, nemůžeme v rámci SPARQL dotazu

[27] http://rdf4j.org/sesame/2.7/docs/users.docbook?view

typu SELECT jako vzor trojice psát přímo `?s a foaf:Agent`. Musíme se dotázat na instance třídy `foaf:Agent` a zároveň její odvozené typy, k čemuž je možné využít vlastnosti „Property Path“ specifikace SPARQL 1.1.

3.2.4 Lokalizace systému

Systém je lokalizován pouze v českém jazyce. Pro uložení samotných textů se používají soubory s příponou „properties“, ve kterých se vždy využívá zápisu klíč-hodnota. Samotná lokalizace je rozdělena do několika souborů. Hlavní lokalizační konfigurace, jež zahrnuje texty pro informační zprávy ze systému a pro texty šablon, je importována do definice aplikačního kontextu z „locale-config.xml“. Dále jsou lokalizována výchozí chybová hlášení (původně většinou v anglickém jazyce) použitých frameworků a knihoven, zejména JSF a PrimeFaces.

3.2.5 Základní logické členění balíčků

Základní logické členění architektury systému, resp. jednotlivých balíčků ilustruje obrázek F.7. Struktura následujícího textu je pak rozdělena do tří vrstev, přičemž jsou popisovány pouze některé zajímavé části a největší důraz je kladen na vrstvu datovou.

3.3 Datová vrstva

Tato podkapitola se zabývá datovou vrstvou samotné aplikace, zejména pak doménovými entitami a DAO objekty, které představují rozhraní pro přístup k datovému úložišti. Také jsou zde uvedeny dotazy pro stránky se seznamy záznamů. Tyto dotazy nejsou závislé na doménových entitních třídách, přičemž jsou implementovány s pomocí frameworku ASF, ve kterém bylo mj. ke zprovoznění pomocné funkcionality nutné provést několik změn, některé jsou přímo uvedeny dále v tomto textu. Upravené zdrojové kódy využitých ASF modulů jsou dostupné na přiloženém CD.

3.3.1 Doménové entitní třídy

Entitní třídy doménového modelu korespondují s modelovanými daty. V tradičním přístupu, využívajícím relační databáze, reprezentuje jedna entitní třída typicky jednu tabulku a její instance poté řádek v dané tabulce. V našem případě reprezentuje instance doménové třídy nějaký zdroj, resp. subjekt tvrzení, každý jednotlivý atribut pak predikát a hodnota atributu tvoří objekt tvrzení.

Všechny entity doménového modelu dědí z třídy `AbstractPersistable`, jež je součástí frameworku ASF. Třída obsahuje především atribut `RdfKey`, který reprezentuje URI a slouží jako identifikátor instancí doménových tříd.

Neméně důležité jsou však jednotlivé anotace. Ukázka 3.3 demonstruje základní použití anotací v třídě modelující osobu z doménového modelu. Řádek č. 1 obsahuje standardní anotování entitní třídy, tato anotace je povinná. Řádek č. 2 definuje do pole (vždy klíč-hodnota) prefix následovaný URI jmenného prostoru, díky tomu následně nemusíme u dalších anotací uvádět celé URI. Řádek č. 3 mj. určuje, jakým typem bude v RDF datech reprezentována instance dané třídy při ukládání dat do datového úložiště. Tato anotace je také povinná. Jednotlivé atributy se anotují pomocí `@RdfProperty`. Při ukládání dat tato anotace určuje, jaký predikát má být v rámci daného tvrzení použit, naopak při získávání dat se podle této anotace určí, do jakého atributu se má hodnota objektu tvrzení přiřadit. Je vhodné připomenout, že pokud se v RDF datech v popisu jednoho subjektu nachází více než jedno tvrzení se stejným predikátem a v entitní třídě je uveden pouze samostatný atribut anotovaný daným predikátem, dojde při získávání dat z úložiště k chybě, neboť nelze rozhodnout, jaká hodnota se má do atributu uložit. Tuto situaci je pak nutné řešit použitím kolekce.

Ukázka 3.3 Ukázka použití anotací v entitní třídě Person

```
1  @Entity
2  @Namespaces({Foaf.PREFIX, Foaf.NAMESPACE})
3  @RdfsClass(Foaf.Person)
4  public class Person extends RdfUser {
5      @RdfProperty(Foaf.firstName)
6      private String firstname;
7      @RdfProperty(Foaf.lastName)
8      private String lastname;
9      @OneToOne(fetch = FetchType.EAGER)
10     @RdfProperty(Foaf.account)
11     private PersonAccount account;
12       ...
13 }
```

Testováním bylo zjištěno, že v případě volání vlastního SPARQL dotazu, který má být proveden jako první, tj. před jakýmkoliv standardním dotazem, který by operoval s danou entitní třídou, dochází k chybě dotazu způsobené nedefinovaným prefixem. Pro zajištění správné funkčnosti frameworku Empire je proto nutné všechny anotované doménové entitní třídy zaregistrovat. To lze zajistit pomocí třídy `RdfGenerator` a užitím její statické metody `init` s uvedením seznamu všech entitních tříd. Ukázka 3.4 demonstruje registraci dvou doménových entit ve vlastní třídě `EmpireEntityInitializator`.

Ukázka 3.4 Inicializace doménových entit pro korektní fungování Empire

```
1  public final class EmpireEntityInitializator {
2      static {
```

```
3            RdfGenerator.init(Arrays.asList(new Class<?>[]{
4                Person.class,
5                PersonAccount.class,
6                 ...
7            }));
8        }
9    }
```

Vytvořenou třídu `EmpireEntityInitializator` pak registrujeme do aplikačního kontextu přímo v hlavním konfiguračním souboru frameworku Spring s explicitním uvedením `lazy-init="false"`.

Ukázka 3.5 Zavedení třídy EmpireEntityInitializator do aplikačního kontextu

```
1  <!-- Empire entities initialization on startup -->
2  <bean class="cz.cvut.fit.adaptmam.model.common.empire.
       EmpireEntityInitializator" lazy-init="false" />
```

Je vhodné také poznamenat, že doménové entitní třídy nevyužívají při deklaraci vnitřních doménových objektů JPA kaskádové typy, ale pro všechny CRUD operace pro každou entitu se vždy využívá rozhraní příslušného DAO objektu. Nepoužití této funkcionality má několik důvodů. Pokud bychom např. použili kaskádový typ pro automatické ukládání vnořených objektů, framework Empire automaticky pro takovýto objekt vygeneruje Uniform Resource Name (URN), nikoliv URI s HTTP schématem.

3.3.1.1 Výčtové typy a Empire

Některé koncepty z doménového modelu jsou modelovány jako výčtové typy, v jazyku Java tzv. `enum`. Jedná se např. o možné uživatelské role v systému, které jsou definovány přímo v ontologii jako instance třídy `sioc:Role` z ontologie SIOC.

Entitní třída `PersonAccount` modelující uživatelský účet pak zahrnuje atribut, jenž je anotován predikátem `sioc:has_function`. Tento atribut přitom nemůže být přímo výčtového typu, neboť framework Empire s nimi nedokáže přímo pracovat. Ačkoliv ukládání dat je tímto způsobem možné, při získávání dat z datového úložiště dojde k chybě. Framework Empire se pokouší zavolat výchozí konstruktor na výčtovém typu, ty jsou však ve výčtových typech privátní.

Řešením této situace je použití obalující třídy, která má daný výčtový typ jako atribut. Třída je přitom deklarována jako instance `sioc:Role` a je použita jako datový typ atributu anotovaného predikátem `sioc:has_function` v entitní třídě `PersonAccount`. Zde se také uplatňuje použití DTO objektů, které tyto vazby zjednodušují a v rámci DTO třídy `PersonAccountDto` je již daná role modelována přímo výčtovým typem.

3.3.2 Uživatelský profil

Uživatelský profil zahrnuje informace zadané přímo uživatelem, jež jsou reprezentovány ve formě klíč-hodnota. Je implementován třídou `UserProfile`, jež rozšiřuje třídu `RdfUserProfile` z frameworku ASF. Zahrnuje atributy deklarované klíčovým slovem `final`, jejich hodnota tvoří název klíče. Ukázka 3.6 ilustruje implementaci uživatelského profilu na příkladu barevného motivu systému.

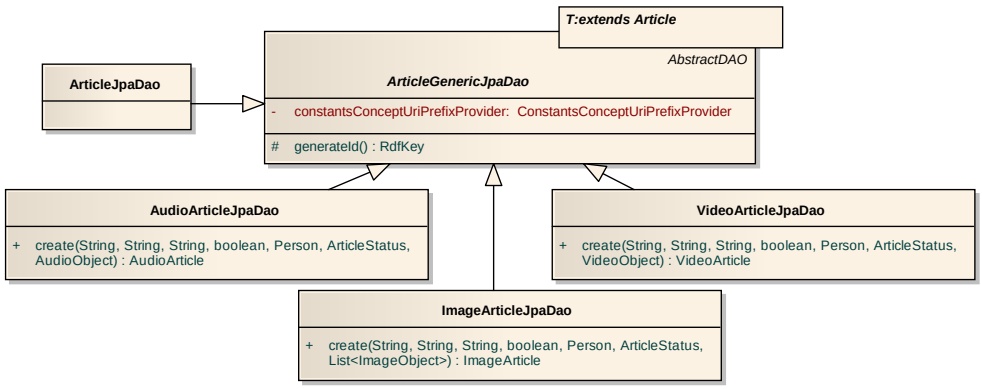
Ukázka 3.6 Demonstrace implementace uživatelského profilu

```
1  public class UserProfile extends RdfUserProfile {
2      ...
3      private final String websiteDefaultTheme = "websiteDefaultTheme";
4      public UserProfile(RdfUser user) {
5          super(user);
6      }
7      public String getWebsiteDefaultTheme() {
8          return getAttributeValue(websiteDefaultTheme);
9      }
10     public void setWebsiteDefaultTheme(String value) {
11         putAttributeValue(websiteDefaultTheme, value);
12     }
13     ...
14 }
```

V konstruktoru na řádku č. 4 je použita třída `RdfUser` z frameworku ASF. Tuto třídu proto musí rozšiřovat vlastní entitní třída `Person`, přičemž je vhodné poznamenat, že obě jsou anotované jako instance třídy `foaf:Person`. V těle metod jsou potom volány metody pro uložení, resp. načtení daného atributu, k čemuž využívá framework ASF vlastní DAO objekty. Přístup k uživatelskému profilu je řešen v servisní vrstvě, používá se k tomu třída `AdaptationManager` z frameworku ASF.

3.3.3 DAO objekty

Základní logické členění ilustruje obrázek F.8. DAO objekty představují rozhraní pro přístup k datovému úložišti a dodržují návrhový vzor DAO. Všechny implementující třídy jsou označeny anotací `@Repository` a rozšiřují abstraktní generickou třídu `AbstractDAO` z frameworku ASF, jež implementuje rozhraní `IRdfDAO`. Obrázek 3.2 ilustruje implementaci DAO na hierarchii článků. Jedná se o implementaci struktury, která byla ilustrována na obrázku 2.3 v části návrhu.



Obrázek 3.2: Implementace DAO na příkladu článků

3.3.3.1 Podpora pro kontrolu aplikační logiky na DAO vrstvě

Framework ASF poskytuje podporu pro řízené odstraňování doménových entit přímo v rámci DAO objektů. Odstranění doménové entity pomocí metody poskytované DAO objektem je možné pouze v případě, kdy vlastní DAO objekt implementuje generické rozhraní `ICanDelete`, resp. `ICanComplexDelete`. První rozhraní definuje jednu metodu `checkCanDelete`, jež vrací objekt typu `BusinessCheckResult`. Na základě výsledku, určeném naší aplikační logikou, je buď vyhozena výjimka typu `BusinessCheckException` nebo je provedeno odstranění objektu. Druhé rozhraní pak rozšiřuje první uvedené, přičemž definuje metody pro operace, které se mají provést buď před nebo po odstranění daného doménového objektu.

Jelikož v rámci prototypu systému nedochází téměř nikdy k přímému odstranění objektu z datového úložiště, nýbrž pouze k nastavení indikátoru, nejsou v současné době uvedená rozhraní DAO objekty využívána, přičemž nastavení příznaku je řešeno ve vrstvě služeb. V případě budoucí potřeby lze však kontrolní logiku pro odstraňování doménových objektů snadno implementovat.

Systém však využívá kontrolní logiky při vytváření některých doménových entit, pro které jejich DAO objekty implementují vlastní generické rozhraní `ICanCreate` s metodou `checkCanCreate`. Její implementace následně zahrnuje kontrolní logiku pro případ vytvoření nové instance daného doménového objektu. Může být např. kontrolováno, zda některé objekty zahrnuté v nově vytvářeném objektu skutečně v datovém úložišti existují, případně zda nebyly odstraněny, přičemž k získání takového objektu z datového úložiště se v těchto případech využívá rozhraní `IStorageDAO` z frameworku ASF.

Vlastní kontrolní logika pro vytváření, potažmo potencionální použití řízené logiky pro odstranění existujících entit, jsou dalšími důvody, proč se v doménových entitách nevyužívá propagace pomocí kaskád, ale CRUD operace jsou realizovány přímo pomocí metod DAO objektů.

3.3.3.2 Generování URI pro koncepty

Při ukládání nových dat do datového úložiště je zapotřebí vygenerovat pro daný zdroj jeho identifikátor, a to s HTTP schéma URI. K tomu je možné využít přímo metodu `generateId()` z třídy `AbstractDAO`. Takto vygenerované URI má tvar složený z URI daného konceptu z ontologie, za který je připojeno Universally Unique Identifier (UUID). To nemusí být vždy žádoucí, např. pokud bychom chtěli v budoucnu mít URI dereferencovatelná. Proto všechny DAO objekty překrývají zmíněnou metodu. V rámci generování URI se pak vždy využívá vlastní společný prefix pro dány koncept, za který je pro zjednodušení opět připojeno UUID. URI prefixy pro koncepty je možné konfigurovat v souboru „constants-concept-uri-prefix.properties“.

3.3.4 DTO objekty

Všechny DTO objekty rozšiřují abstraktní `BaseDto`, které implementuje rozhraní `Serializable`. Použitím DTO objektů dochází k odstranění přímé závislosti webové vrstvy na entitních třídách používaných frameworkem Empire. Také zjednodušují hierarchii doménových entit, např. se používá jeden DTO objekt pro všechny typy článků. Odstraňují také uvedený problém obalujících tříd pro výčtové typy. Navíc díky jejich použití nevzniká problém se serializací objektů entitních tříd používaných frameworkem Empire. Tento problém byl zjištěn v počátcích vývoje, kdy se pro navigaci v systému využívala technologie Spring Web Flow.

DTO objekty jsou využívány především k přenosu dat mezi servisní a webovou vrstvou. Jedno z typických použití je takové, že jsou jejich atributy v zobrazovacích šablonách mapovány na formulářové prvky, odkud jsou naplněny daty. Z tzv. backing bean, což jsou jednoduše řečeno třídy webové vrstvy obsluhující události vzniklé v uživatelském rozhraní, se volají jednotlivé metody služeb. Zde jsou data mapována na entitní doménové třídy a zpracovány do datového úložiště. Atributy DTO objektů, které jsou vázány na formulářové prvky v zobrazovacích šablonách, využívají také anotace ze standardu JSR 303 [44].

Je vhodné uvést, že pro mapování dat mezi těmito objekty a doménovými entitami bylo nutné navrhnout způsob realizace. Manuální mapování by bylo problematické, proto byl proveden krátký průzkum možných řešení a pro zajištění automatického mapování byl zvolen framework Dozer[28], který nabízí možnost konfigurace pomocí XML a lze ho snadno použít s frameworkem Spring.

[28] http://dozer.sourceforge.net/

3.3.5 SPARQL dotazy pro data nezávislá na entitních třídách

V systému nejsou všechny dotazy, načítající data z datového úložiště, realizovány pouze pomocí DAO objektů. Stránky zobrazující záznamy dat, typicky v tabulkové podobě nebo jako seznam, zahrnují pouze část informací, ať už z jedné nebo více entitních tříd. K tomuto účelu jsou v ASF frameworku určeny speciální třídy, které implementují také logiku pro stránkování nebo řazení.

Pro implementaci vlastních SPARQL dotazů je určena abstraktní generická třída `QueryBySPARQL` z modulu ASF.Persistence.Empire. Tato třída je určena pro tvorbu libovolných dotazů, které vracejí data nezávislá na doménových entitních třídách. Jako generický typ se této třídě předává objekt, který zapouzdřuje získaná data. Tento objekt musí rozšiřovat abstraktní třídu `SparqlQueryRecord`.

Vlastní SPARQL dotazy poté definujeme ve vlastní třídě, která rozšiřuje třídu `QueryBySPARQL` s daným typem. V třídě implementujeme všechny abstraktní metody, přičemž jednu z metod tvoří metoda pro počet záznamů. Tato metoda je dále zpracována uvnitř metody `getRowCount()` v uvedené třídě `QueryBySPARQL`. Pro zajištění správné funkčnosti bylo třeba tuto metodu v ASF frameworku upravit, ukázka 3.7 ilustruje starou (řádek č. 2), resp. novou implementaci.

Ukázka 3.7 Změna metody getRowCount() v třídě QueryBySPARQL

```
1  public long getRowCount() {
2    // return (Long)getQueryForCount().getSingleResult();
3      BindingSet bindingSet = (BindingSet) getQueryForCount().getSingleResult();
4      return Long.valueOf(bindingSet.getValue("count").stringValue());
5  }
```

Výstup SELECT dotazu pro počet záznamů je nyní vázán na proměnnou `count` a dotazy v třídách rozšiřující třídu `QueryBySPARQL` musí mít do této proměnné vázán výsledek dotazu, implementaci naznačuje ukázka 3.8.

Ukázka 3.8 Implementace dotazu SELECT COUNT

```
1  protected String getQueryStringForCount() {
2      return String.format("SELECT (COUNT(DISTINCT ?article) AS ?count)
           WHERE {"
3          + "%s"
4          + "}", getWhereExpression());
5  }
```

3.4 Servisní vrstva

Servisní vrstva zahrnuje převážnou část aplikační logiky systému. Poskytuje služby jednotlivým backing bean a využívá rozhraní DAO objektů pro přístup k datům. Implementující třídy jsou anotovány pomocí anotace `@Service` a metody, které využívají přístup k datovému úložišti, jsou anotovány jako `@Transactional`.

Na této vrstvě dochází k mapování DTO objektů na doménové entitní třídy a naopak. Všechny metody servisní vrstvy jsou také nakonfigurovány pro odesílání vzniklých chyb za běhu systému na emailovou adresu, definovanou v konfiguračním souboru „mail-config.xml“.

Základní strukturu servisní vrstvy zachycuje obrázek F.9. Cílem této podkapitoly není detailněji popisovat jednotlivé služby, nicméně blíže uvádí zmíněné reportování vzniklých chyb a dále také službu, která operuje s uživatelským profilem uživatelů.

3.4.1 Notifikace o chybách vzniklých za běhu systému

Jak již bylo zmíněno v úvodu této kapitoly, všechny metody servisní vrstvy jsou nakonfigurovány k tomu, aby se chyby vzniklé za běhu systému, resp. chyby typu `RuntimeException` odesílaly na definovanou emailovou adresu. Motivací pro přidání této funkcionality je rychlá informovanost o chybách vzniklých již v době, kdy je systém využíván reálnými uživateli a během implementace nebo testování se tyto potencionální chyby nepodařilo odhalit.

Je vhodné podotknout, že ke zprovoznění veškeré funkcionality nebylo zapotřebí implementovat žádný vlastní programový kód, využívá se techniky Aspect Oriented Programming (AOP). K zachytávání, resp. logování chyb se využívá implementace, jež je zahrnuta ve frameworku ASF, pro odesílání zpráv na definovaný email se pak využívá implementace z frameworku Spring.

Nezbytná však byla správná konfigurace, která se nachází v souboru „mail-config.xml“. Mezi jedno z nejdůležitějších nastavení patří určení toho, na jaká místa v systému, resp. na jakou vrstvu se funkcionalita reportování chyb aplikuje. Z hlediska architektury systému by byla pro reportování co možná největšího počtu potencionálních chyb nejvhodnější aplikace na vrstvu webovou, nicméně aplikována je na servisní vrstvu, přičemž jsou reportovány takové chyby, které vznikly buď přímo v některé servisní metodě nebo do ní byly propagovány.

Přesun na vrstvu služeb je z důvodu relativně problémové konfigurace na backing bean, které nejsou v aplikačním kontextu, ale jsou pouze označeny anotací `@Configurable` (o důvodech použití této anotace více pojednává kapitola 3.5). Ukázka 3.9 zahrnuje část konfigurace ze souboru „mail-config.xml“, a to konfiguraci všech míst (tzv. pointcut), které se mají obohatit o danou funkcionalitu (tzv. advice). Výraz na řádku č. 3 zajišťuje aplikaci funkcionality na libovolnou metodu v servisní vrstvě [45].

Ukázka 3.9 Definice pointcut pro aplikaci funkcionality pomocí AOP

```
1 <aop:config proxy-target-class="false">
2     <aop:advisor advice-ref="runtimeExceptionInterceptor"
3                  pointcut="execution(* cz.cvut.fit.adaptmam.service.*.*(..))" />
4 </aop:config>
```

Samotná implementace tedy využívá techniky AOP, pomocí kterého je každá metoda servisní vrstvy obohacena o danou funkcionalitu. Stručně řečeno, třída `RuntimeExceptionInterceptor` z frameworku ASF implementuje rozhraní `MethodInterceptor`, resp. metodu `invoke`, kde je volána původní metoda, tj. určitá metoda servisní vrstvy, a v případě chyby typu `RuntimeException` se chyba zpracuje a reportuje.

3.4.2 Služba pro práci s uživatelským profilem

Pro práci s uživatelským profilem slouží rozhraní `UserProfileService`, jehož implementace využívá třídu `AdaptationManager` z frameworku ASF. Této třídě je nutné nastavit vlastní implementaci uživatelského profilu a příslušného DAO objektu z frameworku ASF, což demonstruje ukázka 3.10.

Ukázka 3.10 Konfigurace AdaptationManager

```
1 <bean id="adaptationManager" class="cz.cvut.fel.asf.adapt.gomawe.
      AdaptationManager">
2     <property name="userProfileImplementationClass" value="cz.cvut.fit.
          adaptmam.model.usermodel.UserProfile" />
3     <property name="userProfileAttributeDAO">
4         <bean class="cz.cvut.fel.asf.adapt.gomawe.storage.empire.
              RdfUserProfileAttributeDAOImpl" />
5   </property>
6 </bean>
```

Po tomto nastavení lze třídu `AdaptationManager` využít k získání objektu uživatelského profilu, který se vždy vztahuje k určitému uživateli a se kterým lze dále pracovat. Načtení uživatelského profilu demonstruje ukázka 3.11.

Ukázka 3.11 Přístup k uživatelskému profilu

```
1 UserProfile userProfile = AdaptationManager.getCurrent().getUserProfile(person);
```

V rámci dané servisní třídy také dochází k mapování všech atributů z uživatelského profilu do `UserProfileDto` a naopak, případně pak k jeho přenosu do webové vrstvy systému. Pro efektivnější práci s uživatelským profilem služba poskytuje i rozhraní pro individuální nastavení jednotlivých atributů.

3.5 Webová vrstva

Z hlediska implementace lze obsah webové vrstvy rozdělit do dvou rovin. Jednu rovinu tvoří především JSF stránky, šablony, jejich styly a další komponenty určené pro samotnou webovou prezentaci, druhou rovinu tvoří programový kód pro její podporu.

Základní členění první roviny je znázorněno v příloze G. Pro tvorbu jednotlivých stránek je využíván framework JSF. Každá z obsahových stránek vždy využívá šablonu, která definuje hlavní rozložení obsahových prvků na stránce. Byly vytvořeny dvě šablony, jedna pro sekce, jež jsou dostupné neautentizovaným uživatelům, jedna pro všechny sekce určené autentizovaným uživatelům. Každá šablona pak využívá vlastní Cascading Style Sheets (CSS) styl. Pro podporu tvorby nejvíce používaných formulářových prvků byly vytvořeny vlastní kompozitní komponenty, jejichž užitím lze velmi zjednodušit zápis takovýchto prvků. Jejich použití je podobné tomu, jako je použití standardních komponent z již existujících knihoven, tj. v JSF stránce je deklarován jmenný prostor pro danou kompozitní komponentu a v samotném zápisu kódu lze následně přes příslušný prefix přistoupit k dané komponentě, resp. k jejímu rozhraní. Samotnou implementaci kompozitních komponent lze nalézt na přiloženém CD, pro více informací k jejich tvorbě lze velmi doporučit kapitolu věnovanou kompozitním komponentám v [46].

Základní logické členění druhé roviny je znázorněno v příloze F na obrázku F.10. Jedním z hlavních pilířů jsou backing bean. Jedná se o třídy, které přímo obsluhují události vzniklé v uživatelském rozhraní. Na základě těchto událostí se typicky volají metody jednotlivých služeb, kde dochází k jejich zpracování a uložení/získání dat do/z datového úložiště. Dále jsou zde např. znovu použitelné samostatné komponenty, resp. validátory a konvertory, které jsou využívány v zobrazovacích šablonách.

Následujících několik podkapitol zahrnuje pouze několik vybraných částí, které jsou implementovány v rámci webové vrstvy. Kompletní implementaci je možné nalézt na přiloženém CD.

3.5.1 Architektura backing bean a jejich vlastnosti

Architekturu backing bean ilustruje obrázek F.11. Dříve než bude vysvětlen její popis a důvod jejího návrhu, je žádoucí stručně uvést tzv. rozsah působnosti (z angl. scope) backing bean. V žádném případě není cílem plně vysvětlit všechny možné rozsahy ani technologické dopady, pro více informací lze využít literaturu [35], resp. [46].

3.5.1.1 Rozsah působnosti backing bean

Rozsah působnosti definuje především délku existence každé instance backing bean. Je také nutné rozlišovat rozsah působnosti v rámci frameworku Spring a

v rámci frameworku JSF, obě technologie je totiž možné používat zcela samostatně, přičemž každá poskytuje vlastní technologii pro řešení DI a udržování vytvořených instancí – framework Spring má vlastní kontejner, resp. aplikační kontext, pro framework JSF lze využít technologii Contexts and Dependency Injection (CDI) z platformy Java EE [46]. Pro obě technologie platí, že rozsah působnosti je možné nastavit příslušnou anotací[29].

Tabulka 3.1 představuje rozsahy působnosti, které se nejčastěji používají pro nastavení délky existence instance backing bean. Tabulka také uvádí použití anotací jak ve frameworku Spring, tak ve frameworku JSF.

Tabulka 3.1: Rozsahy působnosti v rámci backing bean

Rozsah	**Spring anotace**	**JSF anotace**	**Popis**
request	`@Scope("request")`	`@RequestScoped`	Existence instance po dobu obsloužení jednoho HTTP požadavku.
session	`@Scope("session")`	`@SessionScoped`	Existence instance po dobu jedné uživatelské relace.
view	neexistuje	`@ViewScoped`	Existence instance po dobu interakce na stránce v jedné záložce prohlížeče.

3.5.1.2 Práce ve více záložkách

To, jaký rozsah působnosti v deklaraci backing bean použijeme, má dopad i na samotnou interakci uživatelů se systémem. Pokud bychom pro „backing-bean“ využívali pouze anotace z frameworku Spring, dle tabulky 3.1 se jedná o rozsah `@Scope("request")`, zejména však rozsah `@Scope("session")`, nebylo by možné umožnit bezproblémovou práci uživatele ve více záložkách v rámci jednoho okna webového prohlížeče, aniž by interakce uživatele na jedné stránce neovlivňovala stav (např. nastavení filtrů) stejné stránky otevřené na jiné záložce.

Proto backing bean, jež jsou využívány stránkami, pro které je vhodné práci na více záložkách umožnit, mají nastaven rozsah působnosti pomocí anotace `@ViewScoped` z frameworku JSF. Anotace podobné vlastnosti ve frameworku Spring chybí a vlastní implementace daného rozsahu pro framework Spring přesahuje rámec této práce. Pro všechny backing bean, jejichž instance existují po dobu obsloužení jednoho HTTP požadavku (např. stránka pro registraci nového uživatele) nebo po dobu jedné uživatelské relace (např. stránka

[29] Framework JSF anotace umožňuje od verze 2.0

s nastavením), jsou použity anotace z frameworku Spring a jsou tedy plně řízeny frameworkem Spring.

3.5.1.3 Řešení problému s automatickým provazováním objektů

Třída anotovaná `@ViewScoped` (spolu s anotací `@ManagedBean`) je JSF bean, nikoliv Spring bean. Není proto udržována v aplikačním kontextu frameworku Spring a vzniká (nejen) problém s automatickým provazováním objektů pomocí anotace `@Autowired`. To lze napravit tím, že třídy daného typu anotujeme pomocí anotace `@Configurable` z frameworku Spring a zároveň přidáme závislosti pro AOP, nastavíme AspectJ plugin v souboru „pom.xml“ a v definici aplikačního kontextu uvedeme `<context:spring-configured />`.

Řešení tedy využívá techniky AOP, přičemž svazování aspektů do aplikace je díky `<context:spring-configured />` prováděno v době kompilace aplikace, jedná se o tzv. statické AOP. Pokud bychom v definici aplikačního uvedli `<context:load-time-weaver />`, proces svazování aspektů do aplikace by probíhal za běhu aplikace a jednalo by se o dynamické AOP, které by vyžadovalo další konfiguraci. Každý typ svazování aspektů má své výhody a nevýhody, statická verze nabízí vyšší výkon, ale naopak nelze upravovat chování aspektů za běhu aplikace.

Pouze výše uvedené nestačí k zajištění funkčnosti `@Autowired` anotace v případech deserializace, a to nejen u tříd, které mají nastaven rozsah anotací `@ViewScoped`, ale i u tříd s rozsahem působnosti `@Scope("session")`. Po deserializaci těchto bean totiž vznikají chyby typu `NullPointerException`, a to u závislostí, jež jsou deklarovány jako `transient`[30], což jsou typicky závislosti na služby. K vyřešení tohoto problému lze využít dynamické AOP, avšak z důvodu relativně komplikované konfigurace dynamického AOP je použito statické AOP. Pro vyřešení tohoto problému se pak využívá upravená implementace standardní Java Development Kit (JDK) metody `readObject`, která se automaticky volá při deserializaci.

3.5.1.4 Popis architektury backing bean

Na základě výše uvedených vlastností vznikla architektura na obrázku F.11. Všechny backing bean s rozsahem definovaným anotací `@Scope("request")` jsou přímým potomkem abstraktní třídy `AbstractBean`, jež je potomkem abstraktní třídy `AbstractController` z frameworku ASF a která také implementuje rozhraní `Serializable`. Všechny ostatní backing bean jsou navíc potomkem abstraktní třídy `AbstractSessionViewScopedBean`, která zahrnuje upravenou implementaci JDK metody `readObject`, ve které je iniciováno provázání závislostí, jež jsou deklarovány s anotací `@Autowired`.

Všechny backing bean, které mají rozsah působnosti definován anotací `@Scope("request")` nebo `@Scope("session")`, využívají anotaci `@Component`

[30] Takové závislosti nejsou serializovány.

z frameworku Spring, naopak `@ViewScoped` backing bean využívají anotaci `@ManagedBean` z frameworku JSF. Pro obě varianty platí, že ke každé backing bean lze díky integraci frameworků Spring a JSF přistupovat pomocí jazyka Unified Expression Language (EL) z JSF stránek.

3.5.2 Stránkované výpisy seznamů

Zobrazování velkého počtu záznamů je v uživatelském rozhraní realizováno stránkováním, přičemž se využívá odloženého načítání (z angl. Lazy loading). K tomu se využívá vždy komponenta z knihovny PrimeFaces, která odložené načítání podporuje. Tyto komponenty obsahují mj. parametr `value` pro určení zdroje dat a parametr `lazy`. V dané backing bean se poté jako datový model využívá generické třídy `QueryLazyDataModel` z modulu ASF.Web.Primefaces, jakožto implementace rozhraní `LazyDataModel` z knihovny PrimeFaces. Generický typ je nahrazen vždy třídou, která zapouzdřuje data pro datový model. Celé stránkování pak využívá zmíněných SPARQL dotazů uvedených v části 3.3.5.

3.5.3 Zachycení výjimek a zobrazení chyby uživatelům

Všechny backing bean dědí z třídy `AbstractController`, která se nachází ve frameworku ASF. Tato třída zahrnuje mj. logiku pro zachytávání výjimek typu `BusinessCheckException`. Každá takto zachycená výjimka je následně automaticky zobrazena uživateli v podobě chybové zprávy, k čemuž se využívá JSF frameworku, resp. knihovny PrimeFaces.

Aby k zachycení výjimky mohlo vůbec dojít, musí být obsluhující kód akce zahrnut do k tomu určené metody, a to do metody `tryExecute`, která přijímá jako parametr implementaci rozhraní `Runnable`. Ukázka 3.12 demonstruje metodu, která obsluhuje událost pro odstranění multimediálního objektu ze systému.

Ukázka 3.12 Obalení obsluhujícího kódu pro zachycení výjimek

```
1  public void deleteMediaObject(final String mediaObjectUri) {
2      if (tryExecute(new Runnable() {
3          public void run() {
4              mediaObjectService.deleteMediaObject(mediaObjectUri);
5      }
6      })) {
7          FacesMessageFactory.createInfoMessage(null, resourceBundle.get("media.
               deleted"), null);
8      }
9  }
```

Na řádku č. 4 je zavolána služba, která zkontroluje, zda je odstranění objektu možné. Pokud se nachází multimediální objekt v některém článku, služba vy-

generuje danou výjimku, která je pak automaticky zpracována a následně také zobrazena uživateli pomocí komponenty PrimeFaces. Pokud nebyla vyhozena žádná výjimka, systém zobrazí informační hlášení.

Konstrukce pro zachytávání výjimek typu `BusinessCheckException` je použita ve většině metod, které obsluhují především akce vyvolané uživateli. Všechny výjimky daného typu také nejsou kvůli této funkcionalitě zpracovávány na jiných vrstvách, ale jsou propagovány až do webové vrstvy. Ukázka 3.12 demonstrovala nejjednodušší možné použití celého konceptu.

3.5.4 Změna barevné tématiky

K změně barevné tématiky systému se využívá komponenty `themeSwitcher`, jež se nachází v knihovně PrimeFaces. Aby tato komponenta zafungovala, je nutné v souboru „web.xml“ uvést kontextový parametr `primefaces.THEME` s hodnotou, jež udává název dané tématiky. K tomu se využívá údaj uložený v uživatelském profilu, který je v rámci této vrstvy spravován komponentou `UserProfileBean`. Je vhodné uvést, že hodnota uvedená v kontextovém parametru je získávána každý HTTP požadavek, resp. přechod v uživatelském rozhraní, proto je žádoucí, aby tato hodnota pokaždé nenačítala hodnotu z databáze. Uživatelský profil je však z podstaty věci načten pouze jednou a udržován v rámci tzv. sezení, tj. jedné uživatelské relace.

3.5.5 Nahrávání a zobrazení multimédií

Pro nahrávání médií je použita komponenta `fileUpload` z knihovny PrimeFaces. Je použita s nastavením `mode="simple"`, které označuje použití nativního rozhraní daného webového prohlížeče. Tato realizace je z toho důvodu, že v rámci nahrávání nových souborů je obsahem formuláře i zadávání metadat a všechna data jsou zpracována v rámci jedné uživatelské akce.

Prezentace multimédií je pak realizována podle jeho typu. Pro obrázky se používá komponenta `lightBox` z knihovny PrimeFaces, pro přehrání audio a video nahrávek se využívá značek `audio`, resp. `video` z HTML5. Protože systém neprovádí žádnou konverzi vstupních souborů, je v rámci kontroly validován jejich typ, nejprve na základě přípony souboru a následně také na základě typu MIME, přičemž podtyp MIME součástí kontroly již není.

Kapitola 4

Testování

Tato kapitola se zabývá testováním systému. Během vývoje byly určité části implementace podrobeny tzv. unit testování, jinak označovanému jako jednotkové testování. Jednotkovému testování jsou podrobeny především DAO objekty a servisní vrstva, ale z části také mapování mezi doménovými třídami a DTO objekty, ačkoliv je toto mapování prováděno automaticky pomocí již dříve zmíněného frameworku Dozer. Systém byl také testován samotnými uživateli, čemuž se věnuje kapitola 4.2.

4.1 Unit testování

V začátku vývoje bylo testování zaměřeno především na DAO objekty. Pro otestování komunikace s datovým úložištěm a pro ověření správného mapování samotných RDF dat a doménových entit, o které se stará framework Empire, je používán specifický typ RDF repositáře v rámci datového úložiště Sesame (viz kapitola 3.2.3). Způsob tohoto testování se liší od testování servisní vrstvy, proto je tato podkapitola rozdělena na testování DAO objektů a testování servisní vrstvy.

4.1.1 Testování DAO objektů

Pro testování DAO objektů se využívá RDF repositář typu „In Memory Store“, který ukládá data do operační paměti. Navíc se využívá takové nastavení, aby v případě vypnutí, resp. restartování serveru nebyla všechna data z operační paměti uložena na disk a repositář byl tak při opětovné inicializaci prázdný. Pro nastavení spojení s daným testovacím repositářem používáme vlastní konfigurační soubor „persistence.xml“ určený pouze pro testování, a který je umístěn ve standardním adresáři „META-INF“, ovšem umístěným mezi zdroji pro testy.

Kromě testování vlastních DAO objektů je také testováno ukládání/získávání dat do/z uživatelského profilu z frameworku ASF, jenž pro tyto operace

využívá vlastní DAO objekty. Z frameworku ASF byla také testována část implementace rozhraní `IStorageDAO`, která je zahrnuta ve frameworku ASF v modulu ASF.Persistence.Empire. Testováním implementace tohoto rozhraní se zjistilo, že metoda `IStorageDAO#findById(RdfKey id)` nepracuje správně a data, o kterých bylo známo, že v datovém úložišti existují, se pomocí této metody nepodařilo vyhledat. Ukázka 4.1 představuje část testu implementace zmíněného rozhraní. V ukázce si lze také všimnout, že používáme anotaci `@RunWith(SpringJUnit4ClassRunner.class)`, díky které, jednoduše řečeno, můžeme používat vlastnosti frameworku Spring v unit testech. Anotací `@ContextConfiguration(...)` pak definujeme umístění aplikačního kontextu pro běh testů.

Ukázka 4.1 Ukázka z testování implementace rozhraní IStorageDAO z ASF

```
1  @Transactional
2  @RunWith(SpringJUnit4ClassRunner.class)
3  @ContextConfiguration(locations = "classpath:test-application-context.xml")
4  public class StorageDaoTest {
5      @Autowired
6      private PersonDao personDao;
7      @Autowired
8      private IStorageDAO storageDao;
9      private Person person;
10
11     @Before
12     public void setUp() {
13         person = personDao.create("Jan", "Novák");
14     }
15
16     @Test
17     public void testFindByIdMethod() {
18         assertNotNull("Person was not properly created or not found!",
               storageDao.findById(person.getId()));
19     }
20       ...
21 }
```

Pro zjištění problému je třeba nahlédnout do implementace dané metody ve frameworku ASF, kterou představuje ukázka 4.2. Klíčový problém se nachází na řádku č. 3, kde je použit specifický konstrukt „Property Path“ (více viz kapitola 1.1.5 o jazyku SPARQL) pro nalezení všech zdrojů, které jsou instancí třídy `owl:Thing` nebo odvozeným typem. Připomeňme, že v OWL světě je typicky každá definovaná třída v ontologiích implicitně podtřídou právě `owl:Thing`, z čehož dále plyne, že jakákoliv instance libovolné třídy je také instancí třídy `owl:Thing`.

Z výše uvedeného se může zdát, že metoda by měla fungovat správně, neboť cílem dotazu je najít zdroj s daným URI, který je v OWL světě sémantického

webu implicitně instancí třídy `owl:Thing`. Důvod, proč se zdroj nepodařilo nalézt, spočívá v tom, že k odvození dané vlastnosti je zapotřebí odvozování na úrovni OWL, přičemž Sesame nabízí standardně odvozování pouze na úrovni RDFS. Jinými slovy, metoda by nám vrátila požadovaný výsledek v případě explicitního uvedení skutečnosti o tom, že třída, jejíž je hledaný zdroj instancí, je podtřídou třídy `owl:Thing`. Pro úplnost dodejme, že metoda by také zafungovala v případě, kdy je v popisu daného zdroje explicitně uvedeno, že je instancí třídy `owl:Thing`.

Ukázka 4.2 Původní implementace metody IStorageDAO#findById(RdfKey id)

```
1  public IRdfPersistable findById(RdfKey id) {
2      Query query = getEntityManager().createQuery("where {
3          ?result rdf:type/rdfs:subClassOf* " + Owl.Thing + " . "
4          + " filter regex(str(?result),??idString) . }");
5      query.setParameter("idString", id.toString());
6      query.setHint(RdfQuery.HINT_ENTITY_CLASS, AbstractPersistable.class);
7      List<AbstractPersistable> resultList = query.getResultList();
8      if (resultList.size() > 1) {
9          throw new RuntimeException("Multiple entities found for query");
10     }
11     return resultList.size() == 1 ? (AbstractPersistable) resultList.get(0) : null;
12 }
```

Na základě výše uvedeného byla navržena nová implementace dané metody frameworku ASF, která nevyžaduje odvozování na úrovni OWL ani explicitní uvedení skutečnosti o tom, že je hledaný zdroj instancí třídy `owl:Thing`. Návrh nové implementace reprezentuje ukázka 4.3. Implementace metody byla také optimalizována, aby se k nalezení zdroje s daným URI nepoužíval regulární výraz (viz řádek č. 4 ukázky 4.2).

Ukázka 4.3 Nová implementace metody IStorageDAO#findById(RdfKey id)

```
1  public IRdfPersistable findById(RdfKey id) {
2      Query query = getEntityManager().createQuery("WHERE {"
3          + "?result ?p ?o . "
4          + "FILTER(?result=??paramId)}")
5          .setParameter("paramId", id.value())
6          .setHint(RdfQuery.HINT_ENTITY_CLASS, AbstractPersistable.class);
7      List<AbstractPersistable> resultList = query.getResultList();
8      return resultList.isEmpty() ? null : (AbstractPersistable) resultList.get(0);
9  }
```

4.1.2 Testování servisní vrstvy

Jednotkové testování servisní vrstvy se provádí odlišným způsobem než testování DAO objektů. V servisní vrstvě dochází mj. typicky k volání metod jed-

notlivých DAO objektů, které vyžadují přístup k datovému úložišti. V rámci unit testování servisních metod však není žádoucí využívat reálnou implementaci těchto metod. Místo toho je vhodné tyto objekty od skutečné implementace odstínit a nahradit je tzv. mock objektem, na kterém pouze definujeme požadovaný výstup a chování pro daný test. Tento mock objekt se pak v samotném průběhu testu chová jako reálný objekt.

Vytváření mock objektu a nasimulování jeho chování manuálním způsobem pro každý test by bylo velmi pracné a časově náročné. Proto existují již hotové knihovny, které vývojářům s tímto procesem pomáhají. Pro tento způsob testování byla zvolena knihovna Mockito[31]. Mezi další možné alternativy pro programovací jazyk Java lze zmínit knihovny EasyMock[32] nebo její rozšíření PowerMock[33]. Ukázka 4.4 demonstruje základní použití knihovny Mockito v rámci testování metody servisní třídy pro přidání hodnocení k článku.

Ukázka 4.4 Ukázka testování servisní metody s použitím knihovny Mockito

```
1  @RunWith(MockitoJUnitRunner.class)
2  public class RatingServiceImplTest {
3      @Mock
4      private RatingDao ratingDao;
5      @Spy
6      private final Mapper mapper = new DozerBeanMapper();
7      @InjectMocks
8      private final RatingService ratingService = new RatingServiceImpl();
9
10     @Test
11     public void testRateArticle() {
12         Rating rating = new Rating(3, "5");
13         when(ratingDao.findArticleRating(anyString())).thenReturn(rating);
14         when(ratingDao.update(rating)).thenReturn(rating);
15
16         RatingDto ratingDto = ratingService.rateArticle("anyUri", 1);
17
18         verify(ratingDao).findArticleRating("anyUri");
19         verify(ratingDao).update(rating);
20         verifyNoMoreInteractions(ratingDao);
21         assertEquals("Rating value is wrong!", 4, ratingDto.getRatingValue(), 0);
22         assertEquals("Rating count is wrong!", 4, ratingDto.getRatingCount());
23     }
24       ...
25 }
```

Z ukázky si lze všimnout, že oproti testování DAO objektu v ukázce 4.1 používáme pro běh testu anotaci `@RunWith(MockitoJUnitRunner.class)`, která

[31] http://mockito.org/
[32] http://easymock.org/
[33] https://code.google.com/p/powermock/

automaticky zajistí správné vytvoření testované servisní třídy s potřebnými závislostmi. Pokud bychom byli nuceni použít z nějakého důvodu zmíněnou anotaci z frameworku Spring, musíme pro správné fungování knihovny Mockito v testovací třídě explicitně zařídit injektování závislostí.

4.2 Uživatelské testování

Pro uživatelské testování byl systém nasazen na server Fakulty elektrotechnické Českého vysokého učení technického. Charakteristiky serveru uvádí tabulka 4.1. Jako aplikační server byl použit server GlassFish verze 3.1.2, na který byl nasazen jak samotný systém, tak OpenRDF Sesame Server pro přístup k datovému úložišti Sesame (viz model nasazení na obrázku F.6).

Tabulka 4.1: Vlastnosti testovacího serveru a prostředí

Hardware	
Paměť RAM	6 GB
Procesor	Intel(R) Xeon(R) CPU E5310,@ 1.60GHz
HDD	50 GB
Software	
Operační systém	Linux 2.6.24-gentoo-r8-02 (amd64)
Java Runtime	SMI Java HotSpot(TM) 64-Bit Server VM (1.6.0_17)
Aplikační server	GlassFish Server 3.1.2.2
Datové úložiště	OpenRDF Sesame 2.7.3 (Native Java Store)

Testování se účastnilo celkem pět uživatelů (ze sedmi oslovených) z akademické obce. Výběr participantů nebyl zaměřen na pohlaví ani věk, nicméně všichni participanti byli muži ve věku 20 až 30 let a s používáním výpočetní techniky měli jisté, lze říci nadprůměrné, zkušenosti. Navíc se již v minulosti sami setkali s testováním uživatelského rozhraní pomocí heuristické evaluace, jež je jednou z metod testování použitelnosti a která se tomuto testování s uživateli přibližuje.

4.2.1 Scénáře

Pro účely testování byla uživatelům po registraci automaticky přidělena role editora, aby měli přístup k většině funkcionalit prototypu systému. Uživatelé dostali scénáře, ve kterých měli splnit požadované úkoly. Po jejich splnění uživatel vyplnil krátký dotazník. Nato proběhla také krátká diskuze, která pomohla některé výsledky upřesnit. Scénáře pro testovaní s uživateli jsou zahrnuty v příloze D.

4.2.2 Výsledky testování s uživateli

Po absolvování všech scénářů účastníci vyplnili dotazník, který zahrnoval několik otázek zabývající se subjektivními dojmy z prototypu systému, problémy či návrhy na jeho zlepšení. Všechny otázky dotazníku se nachází v příloze E, odpovědi jsou zahrnuty na přiloženém CD a jsou také diskutovány dále. Tabulka 4.2 uvádí odpovědi účastníků, a to na část otázek, kde nebylo typem odpovědi textové pole. Číslice v tabulce vycházejí z rozsahu 1–5, přičemž představují běžné hodnocení jako ve škole.

Tabulka 4.2: Odpovědi na část otázek z dotazníku po testování

P1	**P2**	**P3**	**P4**	**P5**
Měli jste před tímto testováním nějaké povědomí o sémantickém webu?				
Ne	Ano	Ano	Ano	Ano
Zdálo se Vám ovládání systému přehledné?				
1	1	2	2	2
Jak se Vám líbilo základní grafické zpracování (barevná tématika)?				
2	1	2	1	1
Zkoušeli jste změnit výchozí barevné téma?				
Ano	Ano	Ano	Ano	Ano
Přišel Vám systém hodnocení a komentování článků přehledný?				
1	2	3	2	1
Jak byste ohodnotili celkový dojem ze systému?				
2	1	2	2	3
Zdá se Vám systém nahrávání souborů a nových článků zvlášť jako vhodný?				
Ne	Ano	Ne	Ne	Ano
Zaregistrovali jste požadované změny v rámci filtrování článků s využitím adaptačních kritérií?				
Ano	Ano	Ano	Ano	Ne

V následujících podkapitolách jsou detailněji probrány některé z otázek, zejména pak ty, u nichž bylo typem odpovědi textové pole a nejsou tak zařazeny v tabulce 4.2.

Měli jste před tímto testováním nějaké povědomí o sémantickém webu?

Tato otázka nesouvisela přímo s vlastním testováním, cílem otázky bylo pouze prozkoumat, zda účastníci měli povědomí o sémantickém webu před tímto testováním. Čtyři účastnící odpověděli na tuto otázku kladně, pouze jeden neměl

žádné povědomí o sémantickém webu. Po přezkoumání výsledků dotazníku proběhla s participanty ještě diskuze a až na jednoho účastníka z těch, kteří odpověděli na tuto otázku kladně, neměli zbylí tři o sémantickém webu hlubší představu – např. o významu datového modelu RDF nebo ontologiích.

Bylo provedení nějaké úkonu z uvedených scénářů nejasné? Pokud ano, uveďte číslo scénáře a co Vám dělalo problémy.

Všichni účastníci neměli s vykonáním úkolů ve scénářích větší problémy, pouze jednomu participantovi nebylo jasné ověření funkčnosti adaptačních kritérií, kterému chybělo bližší vysvětlení jejich dopadu. V rámci nastavení adaptačních kritérií byla proto přidána dodatečná nápověda a zlepšena byla také informační hlášení zobrazovaná po najetí kurzoru myši přes dané formulářové prvky.

Napište stručné zhodnocení. Můžete sdělit, co se Vám líbilo, navrhnout nějaké změny, které by podle Vás systém vylepšily atp.

V rámci poslední otázky mohli participanti více rozvinout své dojmy, čehož také všichni s různou mírou využili. Tato otázka byla z pohledu informační hodnoty odpovědí nejbohatší, participanti vlastními slovy shrnuli své poznatky. Následující seznam shrnuje pouze návrhy na zlepšení, resp. nová řešení některých situací.

- Registrace byla možná se slabým heslem[34]. Síla hesla nebyla vůbec předmětem validace, což bylo následně změněno a slabé heslo již validací úspěšně neprojde. V rámci změny hesla v nastavení profilu nebylo také ověřováno heslo staré, tato ochrana je již také součástí systému. Do registračního formuláře byla také přidána položka pro email.

- Formulář pro nahrání nového souboru do systému za určitých podmínek kombinuje anglický jazyk s českým. Jedná se např. o popisek tlačítka pro výběr souboru z diskového oddílu, které může být pojmenováno např. „Choose file“, „Browse“ atp. Pro nahrání souboru na server je použit standardní prvek jazyka HTML, pojmenování daného prvku je pak určeno implicitně každým webovým prohlížečem a nelze ho, bez použití nějaké knihovny, jednoduchým způsobem přepsat.

- V případě kliknutí na nadpis nepublikovaného článku (v seznamu článků na stránce „Moje články“) je zobrazen detail článku pro běžné uživatele, ovšem s chybovým hlášením, že se uživatel pokouší zobrazit nepublikovaný článek. Odkaz byl v případě daného stavu článku v tomto seznamu deaktivován, ačkoliv toto nebylo nevalidní chování systému, nové řešení je více použitelné, neboť daný přechod na detail byl veskrze zbytečný.

[34]Systém zobrazuje informační hlášení o síle hesla s použitím knihovny PrimeFaces.

- Dvěma participantům chvíli trvalo najít přehled nahraných souborů, který se nacházel v kontextovém menu na stránce, která byla přístupná z hlavního navigačního menu pod volbou „Nahrát soubor“. Tato položka byla přejmenována na „Správa médií“.
- Tvorba, resp. editace článku nenabízela v rámci přiřazení média k článku žádnou možnost vyhledávání. Přidáno bylo vyhledávání na základě názvu souboru.
- Do nastavení adaptačních kritérií byla přidána dodatečná nápověda a mírně upravena informační hlášení zobrazovaná při přechodu kurzoru myši přes dané formulářové prvky.

Kromě výše uvedených úprav bylo provedeno několik dalších. Např. komunikace mezi klientem a serverem je nyní zabezpečena pomocí protokolu HTTPS. Formulářové tlačítko, jež slouží jako potvrzení formuláře pro nahrání nového souboru na server, je nyní po jeho stisku a úspěšné validaci deaktivováno. Po úspěšném přihlášení do systému je cílová stránka určena na základě uživatelské role, např. administrátor systému je nyní přesměrován přímo do administrátorské sekce (více viz příloha C).

Je vhodné také zmínit problém, který souvisel s vyhledáváním. Jeden z respondentů si všiml, že vyhledávání na základě textového řetězce je závislé na velikosti písmen s diakritikou – např. slovo „článek“ a „Článek“ nebylo v rámci vyhledávání identické, i když samotná implementace vyhledávání velikost písmen ignorovala. Tento problém byl opraven, oprava spočívala v nastavení příznaku funkce `REGEX` jazyka SPARQL.

4.2.3 Shrnutí testování s uživateli

Téměř všechny uvedené návrhy byly zařazeny do současné podoby prototypu systému a zlepšily tak jeho uživatelskou použitelnost. Kromě některých zmíněných náležitostí systém dle účastníků testování fungoval očekávaným způsobem a působil přehledně.

4.3 Testování v různých prohlížečích a rozlišeních

Předmětem testování byla celková použitelnost systému v různých prohlížečích a rozlišeních, tj. zda jsou správně zobrazovány a umístěny prezentační prvky, případně jejich chování při změně velikosti okna prohlížeče atd. To, zda lze přehrát audio nebo video v daném prohlížeči, je předem dáno tím, zda použitý prohlížeč podporuje dané HTML5 značky. Testováno také nebylo chování v mobilních zařízeních, pro něž nebyl systém vůbec optimalizován.

Systém byl testován v nejpoužívanějších prohlížečích dle statistiky[35] zveřejněné na stránce W3Schools, a to v prohlížeči Mozilla Firefox (verze 43.x) a

[35] http://www.w3schools.com/browsers/browsers_stats.asp

Chrome (verze 47.x). Samostatnou kapitolou pak bylo testování v prohlížeči Internet Explorer (IE), v němž byl systém testován nejdříve v nejnovější verzi (verze 11.x) a poté byly pomocí vestavěné funkce emulovány verze nižší.

Stručný závěr testování je ten, že systém je, při použití v daných verzích prohlížečů Firefox, Chrome a IE 9 a vyšší, použitelný bez větších problémů. Vzhledem k rozlišení obrazovky, systém je v optimálním zobrazení možné využívat pro rozlišení 1366×768 a vyšší. Taková rozlišení, dle další statistiky[36], využívá naprostá většina uživatelů. Pro menší rozlišení pak bylo prostřednictvím Media Queries z CSS optimalizováno např. zobrazení hlavního navigačního panelu.

Další nalezený problém nesouvisí s rozlišením obrazovky ani s použitým typem prohlížeče, ale s rozlišením přehrávaných videonahrávek. Jelikož systém neumožňuje úpravu rozlišení obrázků ani videonahrávek, je nutné videonahrávky ve větším rozlišení, než je šířka přehrávače umístěného v obsahové části šablony, spouštět v režimu celé obrazovky, jinak dojde k ořezání zobrazovací plochy přehrávače.

Během testování byla zjištěna také aplikační chyba, jež se vyskytovala pouze v prohlížeči IE. Chyba spočívala v nekorektním získání původního názvu souboru. Místo uložení informace pouze o samotném názvu, byl do datového úložiště ukládán název souboru, jenž byl tvořen absolutní cestou k souboru na klientském zařízení. Problém byl vyřešen pomocí knihovny Commons IO[37], která implementuje získání názvu souboru korektně. Předchozí implementace využívala knihovny Primefaces.

4.4 Ověření adaptivního chování

Tato část se stručně zabývá ověřením adaptivního chování. Implementovaný prototyp systému umožňuje adaptaci pouze na základě informací, které jsou do systému zadávány samotnými uživateli. Tyto informace lze nastavit v sekci s nastavením profilu. Pro ověření adaptivního chování nebyla z charakteru adaptace vytvořena žádná sofistikovaná metodika za účelem jejího ověření, ale ověření probíhalo pouze empirickým způsobem.

V sekci s nastavením lze mj. změnit barevné téma vzhledu systému, pozici navigačního panelu v případě stránkování, resp. maximální počet záznamů na jedné stránce. Ověření spočívalo pouze z pozorování, např. zda se barevné téma vzhledu systému opravdu mění. V rámci tohoto ověření tedy bylo na stránce s nastavením změněno barevné téma vzhledu systému, bylo také provedeno odhlášení ze systému a opětovné přihlášení, po kterém je daná informace o vzhledu systému správně načtena z uživatelského profilu.

Do uživatelského profilu je možné uložit adaptační kritéria, která zahrnují preferovaný typ článku, nastavení limitů týkající se hodnocení článků nebo

[36] http://www.w3schools.com/browsers/browsers_display.asp
[37] https://commons.apache.org/proper/commons-io/

výpis článků umožňující přidávání nových komentářů. Všechny tyto informace slouží k adaptaci výpisu článků pro běžné uživatele systému. Ověření probíhalo opět na základě pozorování, chování bylo také částečně ověřeno účastníky testování s uživateli.

Adaptace na základě hodnocení článku se skládá z nastavení dvou na sobě závislých položek, a to z limitu počtu hodnocení článku, resp. z limitu hodnoty samotného hodnocení. Limit počtu hodnocení je zaveden z toho důvodu, aby nebyly automaticky skrývány články, které ještě nebyly hodnoceny, přičemž v rámci nastavení kritérií je navíc systémem určena minimální spodní mez limitu počtu hodnocení na hodnotu 3 (tato hodnota nebyla určena nijak empiricky). Pro ověření bylo v systému vytvořeno několik článků, které byly ohodnoceny. Následně bylo aktivováním adaptace testováno, zda se ve výpisu článků *a*) zobrazují články, které mají nižší hodnocení než nastavený limit hodnocení a zároveň menší počet hodnocení než nastavený limit počtu hodnocení; *b*) zobrazují články, které mají vyšší nebo stejné hodnocení jako nastavený limit hodnocení a zároveň vyšší nebo stejný počet hodnocení jako nastavený limit počtu hodnocení; *c*) nezobrazují články, které mají nižší hodnocení než nastavený limit hodnocení a zároveň vyšší nebo stejný počet hodnocení než nastavený limit počtu hodnocení. Všechny tyto body byly splněny.

Závěr

Kapitola 1 nejprve uvedla problémy tradičních způsobů výměny dat mezi systémy a následně popisovala klíčové body sémantického webu, jako je datový model RDF, ontologie a SPARQL protokol a dotazovací jazyk pro RDF data. RDF je jeden ze základních pilířů sémantického webu, který umožňuje data popisovat standardizovaným způsobem ve formě RDF trojic. Ontologie pak umožňují vložit datům sémantický význam. Mezi jednu z největších výhod lze zařadit skutečnost, že takto popsané informace v sobě nesou jak samotná data, tak informaci o jejich významu, což má mj. za následek výrazně jednoduší výměnu dat mezi systémy.

Začátek kapitoly 2 byl věnován analýze vlastností nejpoužívanějších webových CMS systémů, které nabízejí širokou podporu pro správu (nejen) multimediálního obsahu. U každého systému bylo také shrnuto využití prvků sémantického webu. Bylo zjištěno, že každý z analyzovaných systémů, na rozdíl od implementovaného prototypu, využívá ve výchozím stavu pro uložení dat relační databáze. Některé nabízejí mapování jejich vlastního datového schématu na sémantické datové slovníky, zejména pak datový model Schema.org a pomocí technologie Microdata nebo RDFa anotují obsah v zobrazovacích šablonách.

Dalším bodem již byl návrh a implementace prototypu systému. Zbytek kapitoly 2 se tak věnoval nejprve základní analýze prototypu, byl vytvořen doménový model, jehož převážnou část se následně povedlo pokrýt užitím konceptů z již existujících ontologií. Pro specifickou část tohoto modelu pak bylo vytvořeno několik konceptů ve vlastní ontologii. Další část kapitoly se již zabývala návrhem samotného systému. Byly představeny základní prvky systému, včetně těch, na základě kterých se provádí přizpůsobení prezentace uživateli. Závěr kapitoly byl mj. věnován návrhu uživatelského rozhraní. Už při jeho návrhu byl zohledněn fakt, že prototyp spojuje vstup do administračních částí se vstupem pro běžné uživatele. Administrační sekce přitom nejsou pro běžné uživatele viditelné, v implementaci musely být proti neoprávněnému přístupu zabezpečeny.

Následně kapitola 3 se celá zabývala implementací prototypu. Pro realizaci prototypu měly být využity technologie JSF, Spring, Empire a ASF framework. Implementovaný prototyp integruje všechny uvedené technologie a plně operuje nad daty uloženými v sémantické grafové databázi Sesame, kde jsou uložená ve formě RDF. K jejich dotazování se využívá dotazovací jazyk SPARQL. S realizací prototypu byla spjata řada problémů. Jedním z prvních bylo např. samotné rozhodnutí o volbě typu datového repositáře s ohledem na odvozování dat, na funkčnost frameworku Empire a tvorbu SPARQL dotazů během implementace. Implementovaný prototyp umožňuje správu několika typů obsahu. Podporuje práci s multimediálními soubory typu audio, video a obrázek. Všechny tyto typy jsou definované v ontologiích, přičemž produkovaná data jsou instancemi těchto typů a jsou uložena spolu s významem v sémantickém datovém úložišti. To mj. zjednodušuje jednak jejich potenciální výměnu, jednak jejich rozšiřitelnost a opětovnou použitelnost.

Nakonec byla otestována základní funkčnost a použitelnost implementovaného prototypu. Na základě podnětů byly do systému zakomponovány drobné změny a opravy. Ověření adaptivního chování bylo provedeno empirickým způsobem. Prototyp systému nezahrnuje příliš adaptivních prvků, k adaptaci dochází pouze na základě informací, které byly do systému vloženy samotnými uživateli.

Návrhy na další rozvoj

Implementovaný prototyp je možné dále rozšiřovat, a to nejen díky jeho vícevrstvé architektuře a silně komentovanému programovému kódu, ale i mnohým poznatkům uvedeným v textu této práce.

Vylepšení zpracování multimédií

V současném systému se neprovádí téměř žádná automatická extrakce technických metadat z nahrávaných multimediálních souborů. Reprezentace těchto metadat ve formě RDF by vylepšila jejich hodnotu a zvyšovala by možnosti dalšího znovupoužití. K sémantickému popisu nových informací lze využít, kromě již systémem využívaných slovníků, i např. slovníky [32] nebo [33], jenž byly zmíněny v kapitole 2.5.

Kategorizace článků a multimédií

Jedním z dalších možných rozšíření systému je přidání zcela nového modulu pro správu kategorií, případně žánrů, a to např. ve formě vestavěného editoru. Pomocí tohoto editoru by bylo možné dynamicky tvořit hierarchie kategorií, do kterých by následně bylo možné zařazovat jednotlivé články nebo multimediální soubory. To by mj. přispělo jednak k dalšímu sémantickému anotování obsahu, zároveň by tím mohla vzniknout nová možnost adaptace na základě

průchodu uživatele systémem. Důležité by bylo zohlednit reprezentaci kategorií v ontologii, a to i vzhledem k použitému frameworku Empire.

Vylepšení adaptace

Rozvoj může nastat kromě výše uvedeného i co se týká adaptace, ta je nyní velmi omezená, zahrnuje adaptaci pouze na základě informací, které byly do systému vloženy samotnými uživateli. Možností pro rozšíření adaptace je několik. Lze dále rozšiřovat ty ukládané do uživatelského profilu, ale především se zaměřit na práci s uživatelským modelem a monitorovat uživatelovu interakci se systémem. Např. jaké články si zobrazuje, jaké žánry multimediálních souborů si přehrává atd. Na základě toho by bylo následně možné upřednostňovat takový obsah, o který má uživatel největší zájem.

Literatura

[1] Hebeler, J.; Fisher, M.; Blace, R.; aj.: *Semantic Web Programming*. Indianapolis, Indiana: Wiley Publishing, Inc., 2009.

[2] Heath, T.; Bizer, C.: *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web: Theory and Technology, Morgan & Claypool, první vydání, 2011.

[3] Berners-Lee, T.; Fielding, R.; Masinter, L.: Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, The Internet Society, Leden 2005, [vid. 2015-05-02]. Dostupné z: `https://www.ietf.org/rfc/rfc3986.txt`

[4] Schreiber, G.; Raimond, Y.: RDF 1.1 Primer. W3C note, W3C, Červen 2014, http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/.

[5] Gandon, F.; Schreiber, G.: RDF 1.1 XML Syntax. W3C recommendation, W3C, Únor 2014, http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/.

[6] Brickley, D.; Miller, L.: FOAF Vocabulary Specification 0.99. [online], Leden 2014, [vid. 2015-05-02]. Dostupné z: `http://xmlns.com/foaf/spec/`

[7] Carothers, G.; Prud'hommeaux, E.: RDF 1.1 Turtle. W3C recommendation, W3C, Únor 2014, http://www.w3.org/TR/2014/REC-turtle-20140225/.

[8] Sporny, M.; Herman, I.; Adida, B.; aj.: RDFa 1.1 Primer - Second Edition. W3C note, W3C, Srpen 2013, http://www.w3.org/TR/2013/NOTE-rdfa-primer-20130822/.

[9] Seaborne, A.; Carothers, G.: RDF 1.1 N-Triples. W3C recommendation, W3C, Únor 2014, http://www.w3.org/TR/2014/REC-n-triples-20140225/.

[10] Carothers, G.: RDF 1.1 N-Quads. W3C recommendation, W3C, Únor 2014, http://www.w3.org/TR/2014/REC-n-quads-20140225/.

[11] Seaborne, A.; Carothers, G.: RDF 1.1 TriG. W3C recommendation, W3C, Únor 2014, http://www.w3.org/TR/2014/REC-trig-20140225/.

[12] Lanthaler, M.; Sporny, M.; Kellogg, G.: JSON-LD 1.0. W3C recommendation, W3C, Leden 2014, http://www.w3.org/TR/2014/REC-json-ld-20140116/.

[13] Guha, R.; Brickley, D.: RDF Schema 1.1. W3C recommendation, W3C, Únor 2014, http://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

[14] Schreiber, G.; Dean, M.: OWL Web Ontology Language Reference. W3C recommendation, W3C, Únor 2004, http://www.w3.org/TR/2004/REC-owl-ref-20040210/.

[15] Miles, A.; Bechhofer, S.: SKOS Simple Knowledge Organization System Reference. W3C recommendation, W3C, Srpen 2009, http://www.w3.org/TR/2009/REC-skos-reference-20090818/.

[16] Sosnovsky, S.; Dicheva, D.: Ontological technologies for user modelling. *International Journal of Metadata, Semantics and Ontologies*, ročník 5, č. 1, 2010: s. 32–71.

[17] Hickson, I.: HTML Microdata. W3C note, W3C, Říjen 2013, http://www.w3.org/TR/2013/NOTE-microdata-20131029/.

[18] Prud'hommeaux, E.; Seaborne, A.: SPARQL Query Language for RDF. W3C recommendation, W3C, Leden 2008, http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

[19] Harris, S.; Seaborne, A.: SPARQL 1.1 Query Language. W3C recommendation, W3C, Březen 2013, http://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

[20] Prud'hommeaux, E.; Aranda, C. B.: SPARQL 1.1 Federated Query. W3C recommendation, W3C, Březen 2013, http://www.w3.org/TR/2013/REC-sparql11-federated-query-20130321/.

[21] Brusilovsky, P.: Methods and techniques of adaptive hypermedia. *User Modeling and User Adapted Interaction*, ročník 6, č. 2-3, 1996: s. 87–129.

[22] Knutov, E.; De Bra, P.; Pechenizkiy, M.: AH 12 years later: a comprehensive survey of adaptive hypermedia methods and techniques. *New Review of Hypermedia and Multimedia*, ročník 15, č. 1, 2009: s. 5–38.

[23] Balík, M.; Jelínek, I.: Generic Ontology-based Model for Adaptive Web Environments, A Revised Formal Description Explained within the Context of its Implementation. In *16th International Conference on Computational Science and Engineering*, Sydney, Australia, 2013, s. 495–500.

[24] Balík, M.; Jelínek, I.: Adaptive System Framework: A Way to a Simple Development of Adaptive Hypermedia Systems. In *ADAPTIVE 2013 : The Fifth International Conference on Adaptive and Self-Adaptive Systems and Applications*, Valencia, Spain, 2013, s. 20–25.

[25] Corlosquet, S.: Drupal 7 and RDF. [online], Říjen 2010, [vid. 2015-05-02]. Dostupné z: `http://www.slideshare.net/scorlosquet/drupal-7-and-rdf`

[26] Kloostra, S.: Practical use of Microdata in Joomla! 3 and 2.5. [online], Červenec 2014, [vid. 2015-05-02]. Dostupné z: `http://magazine.joomla.org/issues/issue-july-2014/item/2161-practical-use-of-microdata-in-joomla-3-and-2-5`

[27] Dublin Core Metadata Initiative: DCMI Metadata Terms. [online], Červen 2012, [vid. 2015-05-02]. Dostupné z: `http://dublincore.org/documents/dcmi-terms/`

[28] Dublin Core Metadata Initiative: Dublin Core Metadata Element Set, Version 1.1. [online], Červen 2012, [vid. 2015-05-02]. Dostupné z: `http://dublincore.org/documents/dces/`

[29] Rühle, S.; Baker, T.; Johnston, P.: User Guide/Publishing Metadata. [online], 2015, [vid. 2015-05-02]. Dostupné z: `http://wiki.dublincore.org/index.php/User_Guide/Publishing_Metadata`

[30] Bojars, U.; Breslin, J. G.: SIOC Core Ontology Specification. [online], Březen 2010, [vid. 2015-05-02]. Dostupné z: `http://rdfs.org/sioc/spec/`

[31] Knublauch, H.: OWL version of schema.org. [online], Listopad 2014, [vid. 2015-05-02]. Dostupné z: `http://topbraid.org/schema/`

[32] Champin, P.-A.; Bürger, T.; Michel, T.; aj.: Ontology for Media Resources 1.0. W3C recommendation, W3C, Únor 2012, http://www.w3.org/TR/2012/REC-mediaont-10-20120209/.

[33] Evain, J. P.: EBUCore - the Dublin Core for media. [online], Leden 2013, [vid. 2015-05-02]. Dostupné z: `https://www.ebu.ch/metadata/ontologies/ebucore/index.html`

[34] Davis, I.: VANN: A vocabulary for annotating vocabulary descriptions. [online], 2005, [vid. 2015-05-02]. Dostupné z: http://vocab.org/vann/.html

[35] Johnson, R.; Hoeller, J.; Donald, K.; aj.: *Spring Framework Reference Documentation*. 2014, 4.0.7.RELEASE. Dostupné z: https://docs.spring.io/spring-framework/docs/4.0.7.RELEASE/spring-framework-reference/pdf/spring-framework-reference.pdf

[36] Alex, B.; Taylor, L.: *Spring Security Reference Documentation*. 3.0.8.RELEASE. Dostupné z: http://docs.spring.io/spring-security/site/docs/3.0.x/reference/springsecurity.pdf

[37] Donald, K.; Vervaet, E.; Grelle, J.; aj.: *Spring Web Flow Reference Guide*. Version 2.4.0. Dostupné z: http://docs.spring.io/autorepo/docs/webflow/2.4.0.RELEASE/reference/pdf/spring-webflow-reference.pdf

[38] Grove, M.: Empire: RDF & SPARQL Meet Java & JPA. In *Semantic Technology Conference*, San Francisco, USA, 2010.

[39] Grove, M.: Empire: RDF & SPARQL Meet JPA. [online], Duben 2010, [vid. 2015-03-20]. Dostupné z: http://www.dataversity.net/empire-rdf-sparql-meet-jpa/

[40] Fiala, J.: *Víceúčelový adaptivní vúkový systém založený na ontologiích*. Diplomová práce, České vysoké učení technické v Praze, Leden 2014.

[41] Civici, C.: *PrimeFaces User Guide 5.1*. PrimeTek Informatics, první vydání, 2014.

[42] Hale, C.: How To Safely Store A Password. [online], Leden 2010, [vid. 2015-05-02]. Dostupné z: http://codahale.com/how-to-safely-store-a-password/

[43] Luehe, J.: Alternate Docroots and Local Resource Paths. [online], Únor 2008, [vid. 2015-05-02]. Dostupné z: https://blogs.oracle.com/jluehe/entry/alternate_docroots_and_local_resource

[44] Bernard, E.; Peterson, S.: JSR 303: Bean Validation. [online], Září 2009, [vid. 2015-05-02]. Dostupné z: http://beanvalidation.org/1.0/spec/

[45] Gupta, L.: Writing Spring AOP AspectJ Pointcut Expressions With Examples. [online], Únor 2015, [vid. 2015-11-25]. Dostupné z: http://howtodoinjava.com/2015/02/03/writing-spring-aop-aspectj-pointcut-expressions-with-examples/

[46] Jendrock, E.; Cervera-Navarro, R.; Evans, I.; aj.: *The Java EE 6 Tutorial*. Oracle and/or its affiliates, Leden 2013. Dostupné z: `http://docs.oracle.com/javaee/6/tutorial/doc/javaeetutorial6.pdf`

Příloha A

Seznam použitých zkratek

AJAX Asynchronous JavaScript and XML.

AOP Aspect Oriented Programming.

API Application Programming Interface.

ASF Adaptive System Framework.

CDI Contexts and Dependency Injection.

CMS Content Management System.

CSS Cascading Style Sheets.

DAO Data Access Object.

DCMES Dublin Core Metadata Element Set.

DCMI Dublin Core Metadata Initiative.

DCTERMS DCMI Metadata Terms.

DI Dependency Injection.

DTO Data Transfer Object.

EL Unified Expression Language.

FOAF Friend of a Friend.

GOMAWE Generic Ontology-based Model for Adaptive Web Environments.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

IDE Integrated Development Environment.

IE Internet Explorer.

IoC Inversion of Control.

Java EE Java Platform Enterprise Edition.

JDK Java Development Kit.

JPA Java Persistence API.

JSF Java Server Faces.

LOV Linked Open Vocabularies.

MIME Multipurpose Internet Mail Extensions.

OCLC Online Computer Library Center.

OWL Web Ontology Language.

PHP PHP: Hypertext Preprocessor.

POM Project Object Model.

PURL Persistent Uniform Resource Locators.

RDF Resource Description Framework.

RDFa RDF in Attributes.

RDFS RDF Schema.

SIOC Semantically-Interlinked Online Communities.

SKOS Simple Knowledge Organization System.

SPARQL SPARQL Protocol and RDF Query Language.

SQL Structured Query Language.

Turtle Terse RDF Triple Language.

UI User Interface.

UML Unified Modeling Language.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

URN Uniform Resource Name.

UUID Universally Unique Identifier.

W3C World Wide Web Consortium.

WWW World Wide Web.

XHTML Extensible HyperText Markup Language.

XML Extensible Markup Language.

XSD XML Schema Datatypes.

XSLT Extensible Stylesheet Language Transformations.

Příloha B

Instalační příručka

Tato část popisuje základní prerekvizity a návod pro zprovoznění systému na vlastním zařízení. Jedná se o doporučený postup pro úspěšné zprovoznění systému. Všechny uvedené verze v části B.1 jsou odzkoušené a neměl by vzniknout žádný problém. V případě nahrazení aplikačního serveru za jiný je nutná další konfigurace a tím se tato instalační příručka nezabývá. Instalační příručka se také nezabývá postupem pro správnou instalaci potřebných nástrojů uvedených v části B.1.

B.1 Prerekvizity

- JDK verze 6.
- Aplikační server GlassFish verze 3.1.2. Slouží pro nahrání samotného systému a datového úložiště.
- OpenRDF Sesame Server verze 2.7.3 pro přístup k RDF datovému úložišti Sesame. Vyšší verze 2.7.x by měly být také kompatibilní, je třeba si však dát pozor na to, aby na daném zařízení nebylo více instalací Sesame Server, což v daném případě může vést ke vzniku různých problémů.
- OpenRDF Workbench verze 2.7.3, resp. v závislosti na použité verzi Sesame Server. OpenRDF Workbench je webová aplikace umožňující interakci s databází Sesame. Lze tak procházet samotná data skrz webové rozhraní a např. je také modifikovat. Aplikace bude dále použita pro vytvoření datových RDF repositářů a pro nahrání počátečních dat.
- Apache Maven verze 3. Slouží k sestavení systému ze zdrojových kódů, resp. k získání všech knihovních závislostí. Od verze 3.3 je již vyžadována JDK verze 7, proto je doporučena verze 3.2.5 v případě JDK verze 6.

Základní instalace OpenRDF Sesame Server a webové aplikace OpenRDF Workbench spočívá v nasazení na aplikační server, v našem případě Glass-

Fish. Funkčnost obou částí lze poté ověřit na adrese `http://localhost:8080/openrdf-sesame/overview.view`, resp. `http://localhost:8080/openrdf-workbench`.

B.2 Postup instalace

V případě, že jsou všechny prerekvizity uvedené v sekci B.1 splněny, postupujte podle následujících instrukcí. Ještě předtím je dobré vědět, že v současném nastavení projektu jsou během sestavení systému automaticky spouštěny všechny testy. Část testů se týká i datové vrstvy, resp. DAO objektů, a je proto zapotřebí přístup k datovému úložišti a mít vytvořený testovací RDF repositář. Pro tyto testy se používá specifický typ datového repositáře, tedy ne repositář pro data využívaná samotným systémem. Tvorbou tohoto repositáře se zabývá krok č. 2 v části B.2.1. Pokud nechcete testy spouštět vůbec, můžete změnit hodnotu `skipTests` v nastavení `maven-surefire-plugin` v souboru „pom.xml“ a krok č. 2 přeskočit. Je také možné upravit konfigurační soubor „persistence.xml“ umístěný v adresáři „META-INF“ mezi zdroji pro testy, aby byly testy spouštěny v operační paměti a přitom nebylo nutné vytvářet testovací repositář – stačí pouze odstranit všechny vlastnosti kromě `factory`.

B.2.1 Vytvoření RDF repositářů

1. V aplikaci OpenRDF Workbench vytvořte nový RDF repositář. Jako typ repositáře zvolte „Native Java Store“. Tento typ RDF repositáře ukládá, resp. získává data přímo z disku. Hodnota `ID` musí korespondovat se jménem repositáře uvedeným v souboru „persistence.xml“ nacházejícím se v adresáři se zdrojovými kódy, a to přesně v adresáři „src/main/resources/META-INF/“. V tomto souboru také zkontrolujte a případně upravte hodnotu URL k Sesame Server.

2. Pro testování je určen specifický typ repositáře. V aplikaci OpenRDF Workbench vytvořte nový repositář typu „In Memory Store“. Tento typ repositáře operuje s daty přímo z operační paměti. Hodnota `ID` musí být stejná jako hodnota v souboru „persistence.xml“, tentokrát však umístěným v adresáři „src/test/resources/META-INF/“. V procesu vytváření tohoto repositáře je důležité nastavení vlastnosti `persist` na hodnotu `false`. To způsobí, že data nebudou v případě vypnutí/restartování serveru uložena z operační paměti na disk a při opětovném spuštění bude repositář opět prázdný.

3. Do RDF repositáře vytvořeného v kroku č. 1 nahrajte počáteční data. Jedná se o soubor „ontologies-and-administrator-data.rdf“ z adresáře „data/preinstall/initial-data/“, který zahrnuje ontologie a data pro administrátora systému. Počáteční přihlašovací údaje administrátora systému jsou pak „admin“ a „password“.

B.2.2 Sestavení a nasazení systému

1. Na přiloženém CD v adresáři „data/preinstall/dependency/“ spusťte skript „install-dependency“. Pomocí nástroje Apache Maven dojde k nainstalování všech knihovních závislostí z daného adresáře do lokálního repositáře. Jedná se o knihovny, které se nenacházejí v centrálním repositáři.

2. V kořenovém adresáři se zdrojovými kódy, tj. tam, kde se nachází soubor „pom.xml“, spusťte příkaz `mvn package`. Pomocí tohoto příkazu dojde k vytvoření jednoho distribuovatelného souboru celého systému („adapt-mam.war“), mj. dojde také ke stažení všech potřebných knihoven třetích stran do lokálního repositáře, proto tato akce může trvat trochu déle v závislosti na rychlosti připojení. Pokud nechcete spouštět testy, přičemž jste neupravili zmíněnou hodnotu v souboru „pom.xml“, použijte místo `mvn package` příkaz `mvn package -Dmaven.test.skip=true`.

3. V předchozím kroku vytvořený webový archiv nasaďte na aplikační server GlassFish. Pro úspěšné nasazení je třeba, aby v dané doméně serveru GlassFish existoval adresář „docroot“, který by však měl existovat standardně. O vytvoření dalších podadresářů se již systém postará sám. Po úspěšném nasazení by již měla být aplikace přístupná na adrese `http://localhost:8080/adapt-mam/`.

B.2.2.1 Další volitelné kroky

- V souboru „constants-concept-uri-prefix.properties“, který se nachází v adresáři „src/main/resources“, lze konfigurovat URI prefixy pro jednotlivé koncepty.

- V souboru „constants-file-storage.properties“, nacházejícím se ve stejném adresáři, lze konfigurovat cesty pro nahrávání multimediálních souborů. Tyto hodnoty musí korespondovat s hodnotou vlastnosti `dir` v souboru „glassfish-web.xml“ v adresáři „src/main/webapp/WEB-INF/“.

- V případě nahrazení serveru GlassFish za jiný typ, např. server Tomcat, je kromě konfigurace serveru pro nahrávání a servírování multimediálních souborů, třeba změnit scope u knihovny hibernate-validator v souboru „pom.xml“, a to z důvodu použité JSR 303 validace. V případě serveru GlassFish je u této závislosti třeba nastavit hodnotu atributu scope na `provided`, neboť je v použité verzi serveru GlassFish (3.1.2.2) již obsažena. Pokud scope není takto nastaven, velmi pravděpodobně mohou nastat problémy i s prostým přihlášením do systému, a to např. po znovu nasazení systému na server nebo při deserializaci ze session. K vyřešení problému je poté nutné nasadit systém na čistý server, tj. provést undeploy systému, případně další vyčištění zbytkových souborů

po systému přímo v doméně GlassFish a až poté systém znovu nasadit. Pokud je použit Tomcat server, je třeba naopak nastavit atribut scope na hodnotu `compile`, což je také výchozí nastavení, pokud není atribut scope uveden.

Příloha C

Uživatelská příručka

Tato část popisuje některé základní obrazovky z implementovaného prototypu systému, případně jejich součásti. Je rozdělena do čtyř podkapitol, a to podle jednotlivých uživatelských rolí.

C.1 Společné obrazovky

V této podkapitole jsou představeny především základní obrazovky, které jsou společné všem uživatelům. Jedná se o stránky s přihlášením a registrací do systému.

Vstupní obrazovka s přihlášením

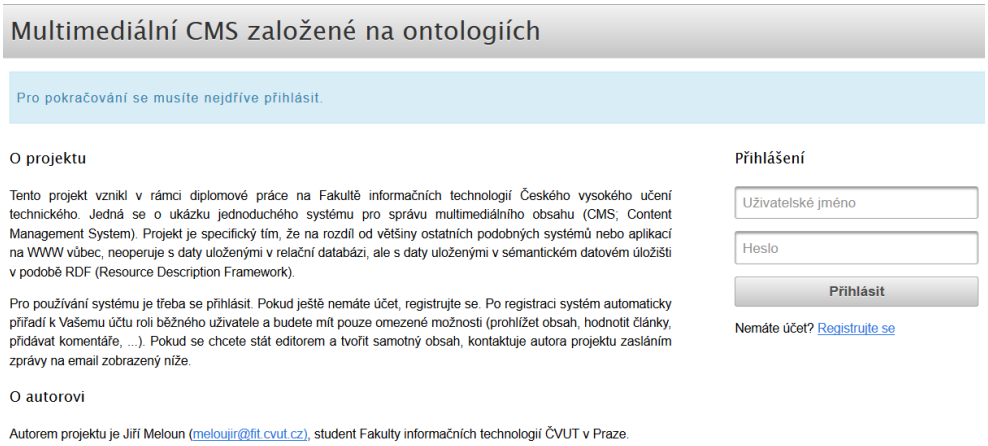
Úvodní obrazovku s přihlašovacím formulářem ilustruje obrázek C.1. Uživatel se může buď přihlásit zadáním uživatelského jména a hesla, resp. přejít do sekce pro novou registraci uživatele do systému.

Registrační obrazovka

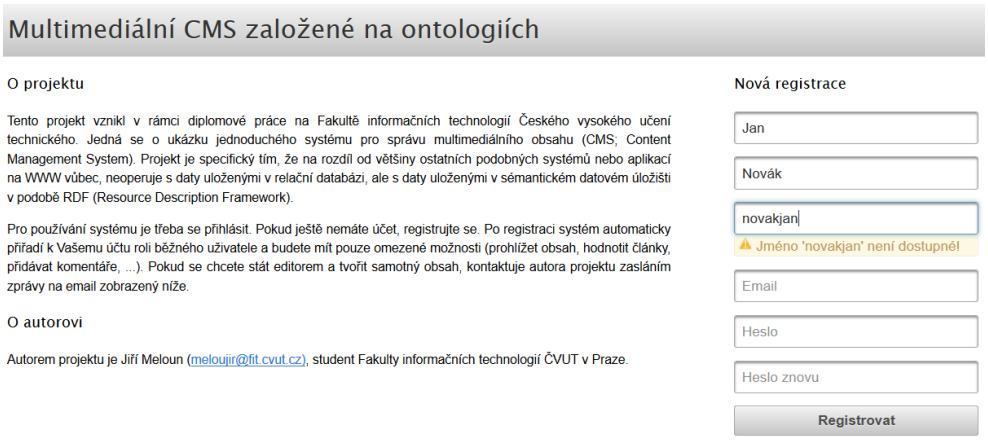
Stránka s registrací je dostupná z přihlašovací stránky. Po úspěšném vyplnění všech údajů je uživatel automaticky přihlášen do systému. Obrazovku s registračním formulářem ilustruje obrázek C.2.

Navigace v záhlaví stránky

Po úspěšném přihlášení, resp. registraci do systému je v záhlaví stránky obsaženo hlavní navigační menu, jehož podoba je dána podle toho, jakou roli přihlášený uživatel v systému má. Obrázek C.3 zachycuje všechny tři možné varianty – část C.3a variantu pro běžného uživatele, část C.3b variantu pro uživatele s rolí editora a část C.3c variantu pro administrátora systému.

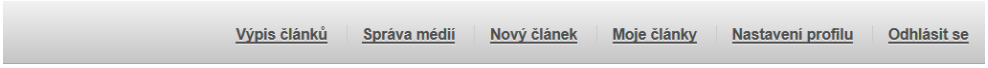


Obrázek C.1: Úvodní obrazovka s přihlašovacím formulářem

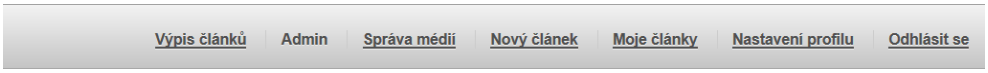


Obrázek C.2: Obrazovka s registračním formulářem

(a) Varianta pro uživatele s rolí člen



(b) Varianta pro uživatele s rolí editor



(c) Varianta pro uživatele s rolí administrátor

Obrázek C.3: Hlavní navigační menu v záhlaví každé stránky

Obrázek C.4: Obrazovka s nastavením osobních údajů

C.2 Obrazovky uživatele s rolí Člen

Tato část popisuje sekce, které jsou dostupné až po úspěšném přihlášení uživatele do systému a které jsou dostupné pro uživatele s rolí člen a výše.

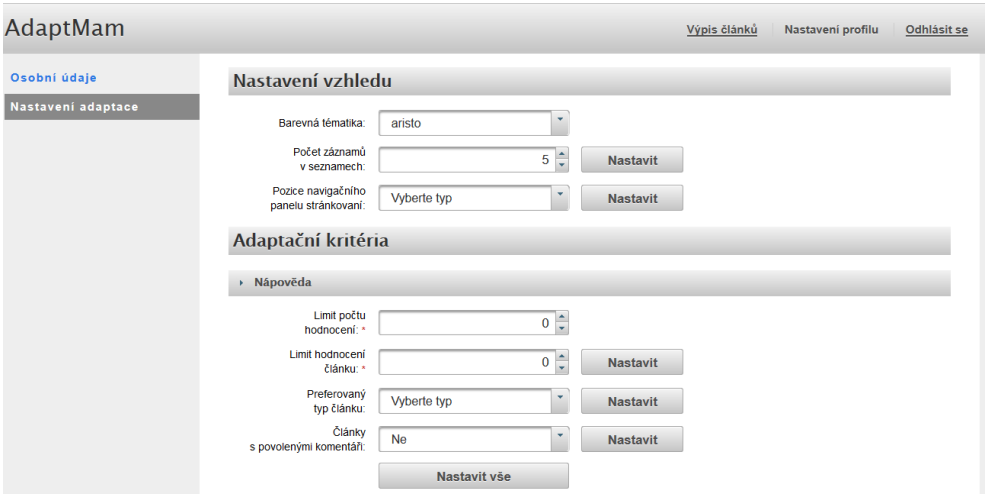
Nastavení profilu

Nastavení profilu je dostupné z hlavního navigačního menu obsaženého v záhlaví stránky (viz obrázek C.3) pod položkou „Nastavení profilu“. Samotné nastavení profilu je pak rozděleno do dvou částí, a to nastavení osobních údajů (obrázek C.4) a nastavení adaptace (obrázek C.5). Obě tyto části jsou přístupné z kontextového menu nacházejícím se v levém postranním panelu. V rámci nastavení adaptace lze mj. změnit barevné téma vzhledu systému, přičemž změna je provedena ihned po výběru z dostupného seznamu.

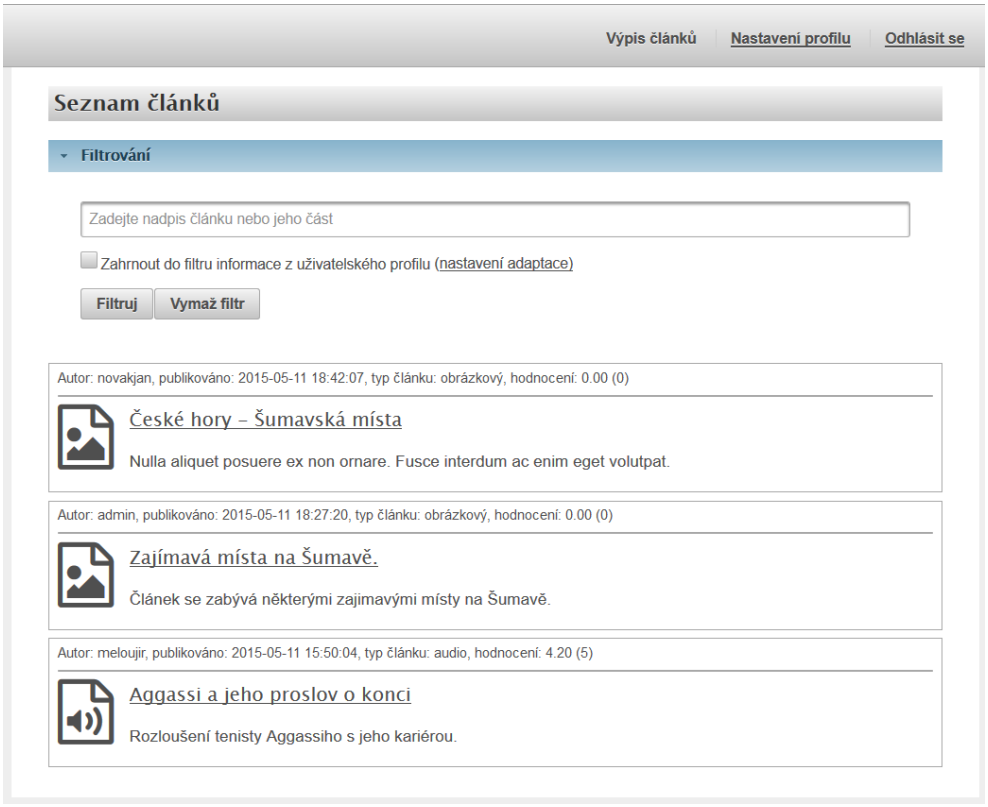
Výpis seznamu článků a detail článku

Obrázek C.6 zachycuje výpis seznamu článků pro běžné uživatele. Všechny články v tomto seznamu jsou dostupné, resp. publikované. Články jsou řazeny sestupně dle data publikace článku. V případě, že je počet článků větší než hodnota uložená v uživatelském profilu nebo výchozí hodnota, je zobrazeno stránkování. Po kliknutí na nadpis článku lze pak přejít na detail daného článku. V rámci filtrování výpisu lze zapnout adaptaci na základě nastavených adaptačních kritérií. Na této stránce se také uživatel nachází ihned po přihlášení, pokud je jeho role v systému člen.

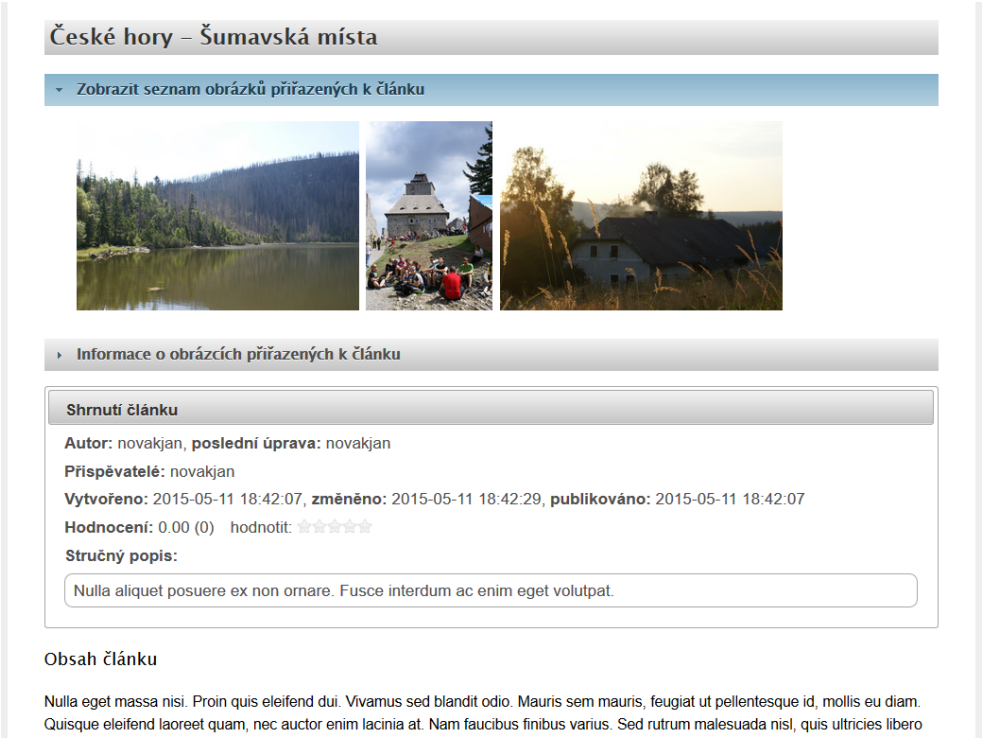
Detail článku zachycuje obrázek C.7. Jedná se o typ obrázkového článku, přičemž přiřazené obrázky k článku jsou zobrazeny v horní části. Po kliknutí na jakýkoliv obrázek je otevřena prezentace obrázků v podobě modálního pře-



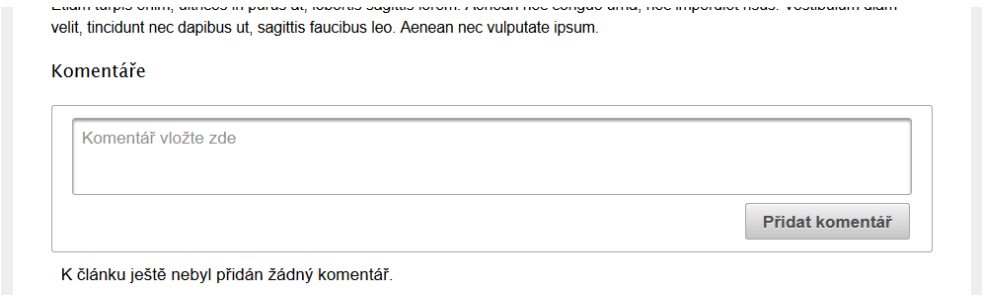
Obrázek C.5: Obrazovka s nastavením adaptace



Obrázek C.6: Obrazovka s výpisem seznamu dostupných článků



Obrázek C.7: Obrazovka běžného detailu článku s obrázky

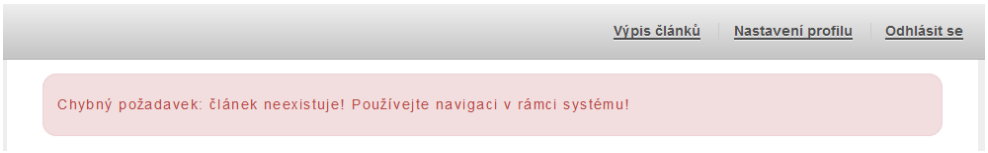


Obrázek C.8: Přidání komentářů k článku

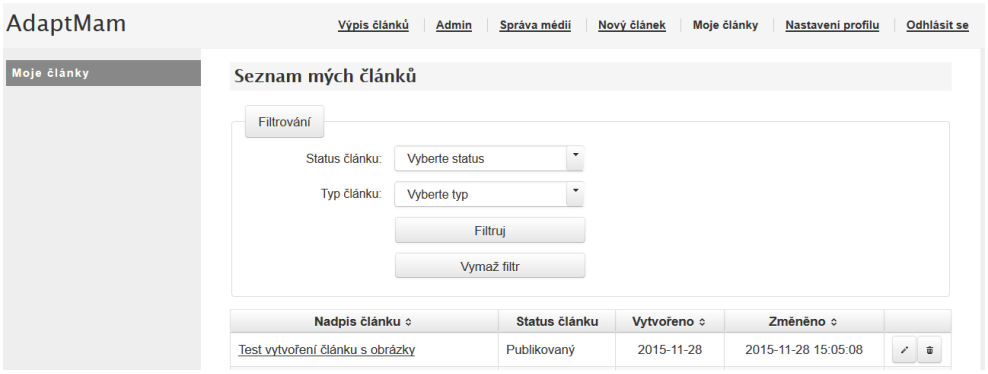
krytí ostatního obsahu. Pro všechny ostatní typy článku je struktura detailu podobná, liší se typem komponenty, která slouží k zobrazení, resp. přehrání daného multimediálního souboru. Pokud je uživatel v roli editora a výše, je v horní části detailu zobrazeno také tlačítko pro editaci článku.

Obrázek C.8 se týká také zobrazení detailu článku, a to přidání komentářů k článku, které je dostupné ve spodní části detailu.

Obrázek C.9 demonstruje zobrazení chybového hlášení, konkrétně o neexistujícím článku. Další chybová hlášení podobného charakteru, jako je např. pokus o zobrazení nepublikovaného článku, jsou zobrazena obdobným způso-



Obrázek C.9: Obrazovka s chybovým hlášením o neexistujícím článku



Obrázek C.10: Obrazovka s výpisem vlastních článků daného editora

bem.

C.3 Obrazovky uživatele s rolí Editor

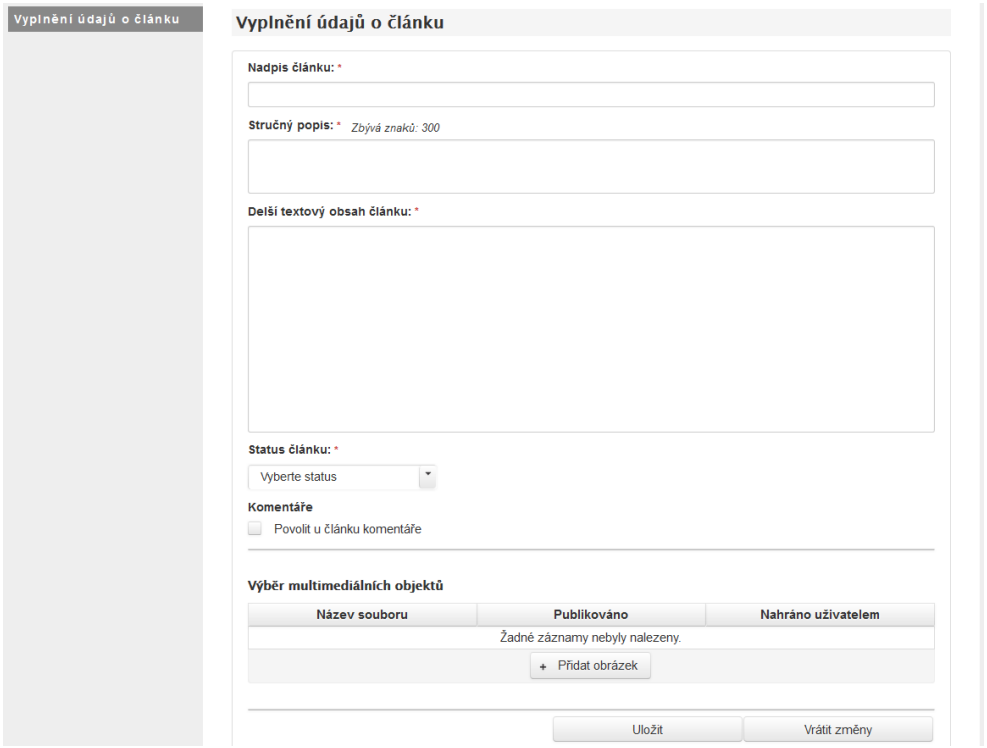
Tato část popisuje sekce, které jsou dostupné pro uživatele s rolí editor a výše. Zaměřuje se především na vytváření článků a správu multimediálních souborů. V rámci specifických částí pro určitý typ mediálního souboru se zaměřuje pouze na práci s obrázky. Obrázek C.10 ilustruje obrazovku s přehledem článků, u kterých je právě přihlášený uživatel autorem. Jednotlivé záznamy jsou vypsány v tabulkové podobě, přičemž k dispozici je filtrování, řazení dle různých kritérií a akční tlačítka pro editaci článku, resp. jeho odstranění. Na této stránce se uživatel nachází po přihlášení, pokud je jeho role v systému editor.

Tvorba nového článku

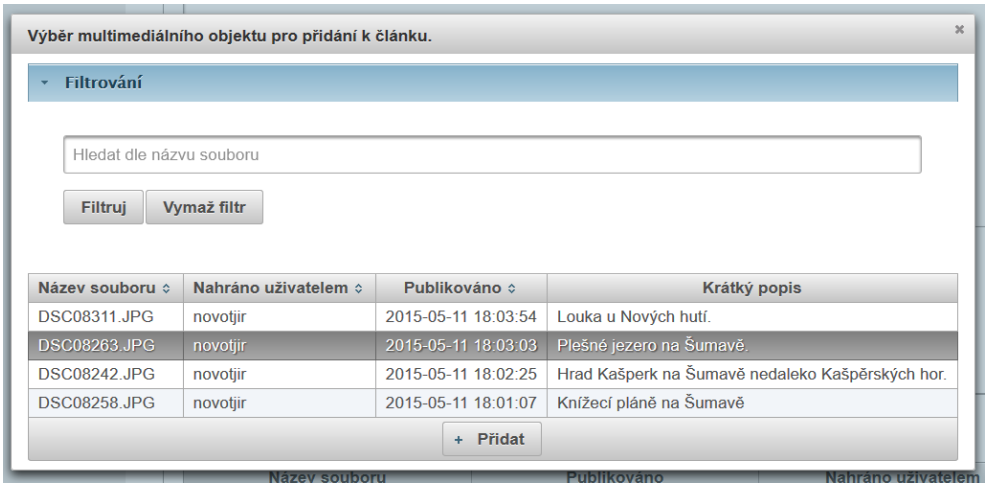
Tvorba nového článku je dostupná z hlavního menu pod položkou „Nový článek“. Samotné tvorbě předchází volba typu článku, což ilustruje obrázek C.11.

Obrázek C.12 představuje editační formulář pro daný typ článku. Stejný formulář se využívá jak pro tvorbu nového článku, tak pro editaci již existujícího. Pro dokončení tvorby, resp. editace slouží tlačítko nacházející se v dolní části formuláře, po jehož aktivaci je ještě zobrazeno potvrzovací dialogové okno. Veškeré neuložené změny lze vrátit do původního stavu pomocí tlačítka dostupného vedle tlačítka pro uložení, této akci opět předchází potvrzení.

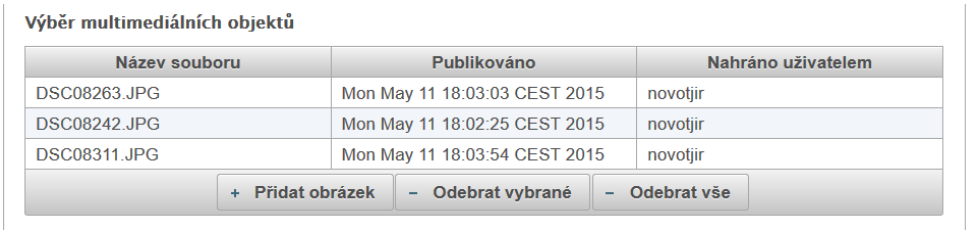
Obrázek C.11: Obrazovka s výběrem typu článku k vytvoření nového článku



Obrázek C.12: Formulář pro vytvoření nového článku s obrázky



Obrázek C.13: Dialogové okno s výpisem obrázků pro přiřazení k článku



Obrázek C.14: Výpis seznamu obrázků přiřazených k článku

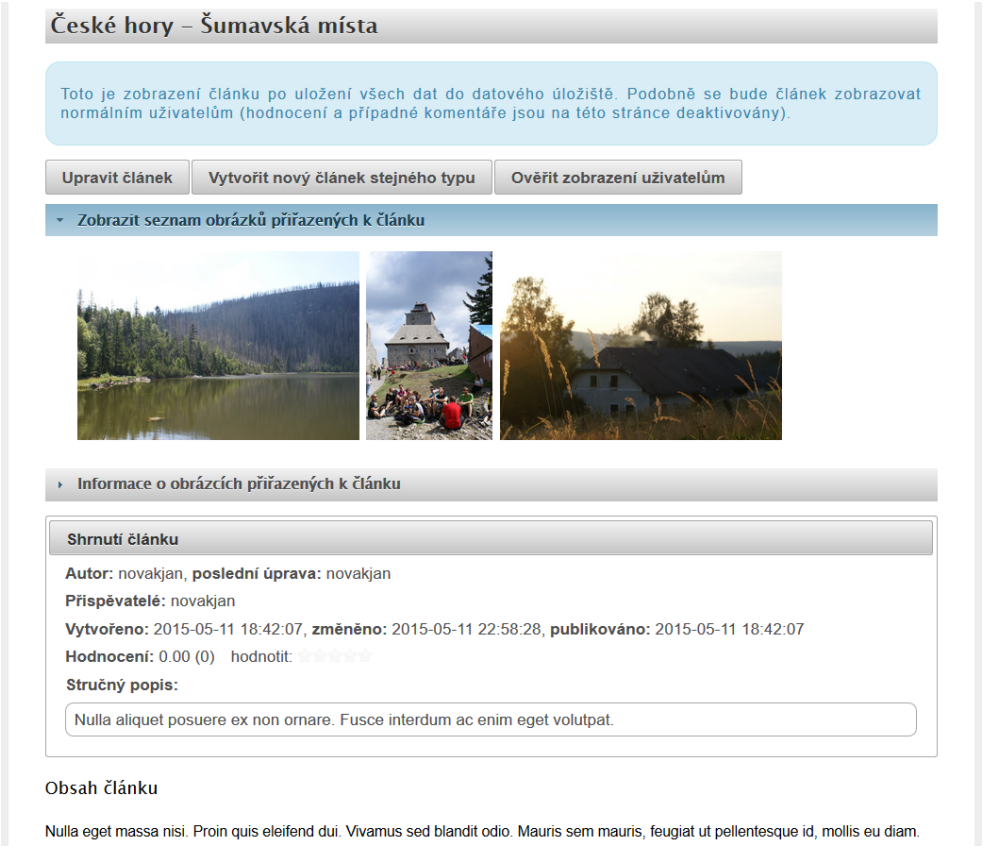
Výběr obrázků pro přiřazení k článku zachycuje obrázek C.13. Jedná se o dialogové okno se seznamem multimediálních souborů daného typu v tabulkové podobě. Je k dispozici filtrování dle názvu, řazení dle různých kritérií a pokud je počet záznamů vyšší než hodnota uložená v uživatelském profilu nebo hodnota výchozí, je použito stránkování.

Po přiřazení obrázku k článku lze libovolně přidávat další, odstranit vybrané nebo všechny přiřazené, jak ilustruje obrázek C.14. V případě ostatních typů článku je styl práce s danými médii podobný, avšak po přiřazení nějakého média k článku je již možná pouze jeho záměna za jiný.

Po úspěšném potvrzení tvorby či editace článku je zobrazen detail článku podobný tomu, jak se bude článek zobrazovat ostatním uživatelům, což demonstruje obrázek C.15. V rámci tohoto detailu jsou mj. k dispozici rychlá akční tlačítka, ať už pro editaci právě vytvořeného článku nebo vytvoření nového článku stejného typu jako právě vytvořený.

Správa multimediálních souborů

Nahrání multimediálního souboru do systému je dostupné z horního navigačního menu pod položkou „Správa médií“. Na zobrazené stránce, viz obrázek

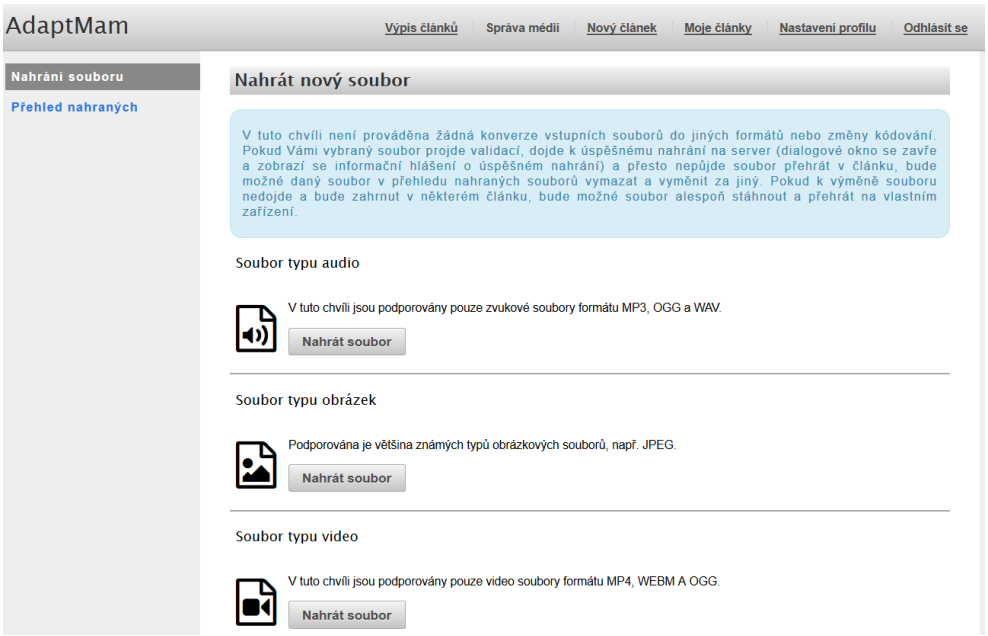


Obrázek C.15: Obrazovka detailu článku s obrázky ihned po jeho vytvoření

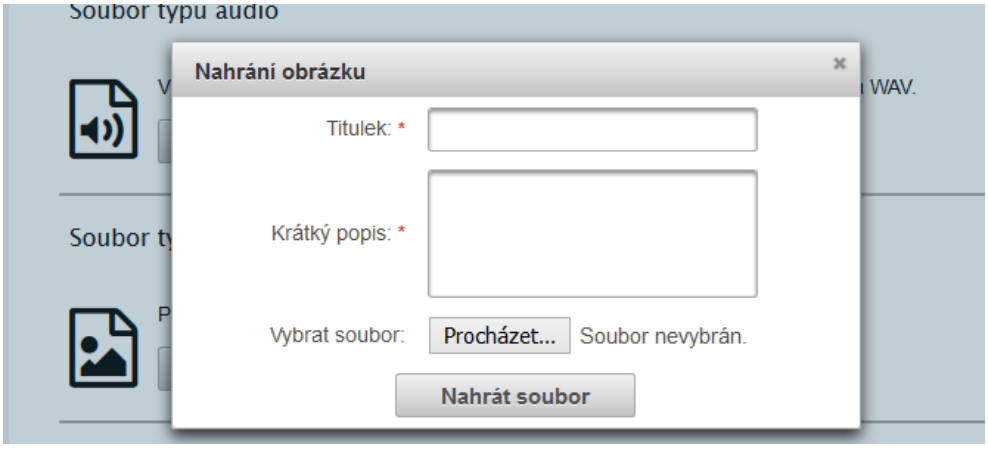
C.16, se nachází volby pro nahrání audia, videa či obrázku.

Po aktivaci příslušného tlačítka je zobrazeno dialogové okno, které zahrnuje formulářová pole (dle typu média) pro anotaci nahrávaného média a standardní formulářový prvek pro výběr souboru z diskového oddílu. To vše demonstruje obrázek C.17. Po úspěšném potvrzení formuláře je tlačítko pro nahrání souboru ihned deaktivováno a jakmile je soubor nahrán na server, je dialogové okno skryto.

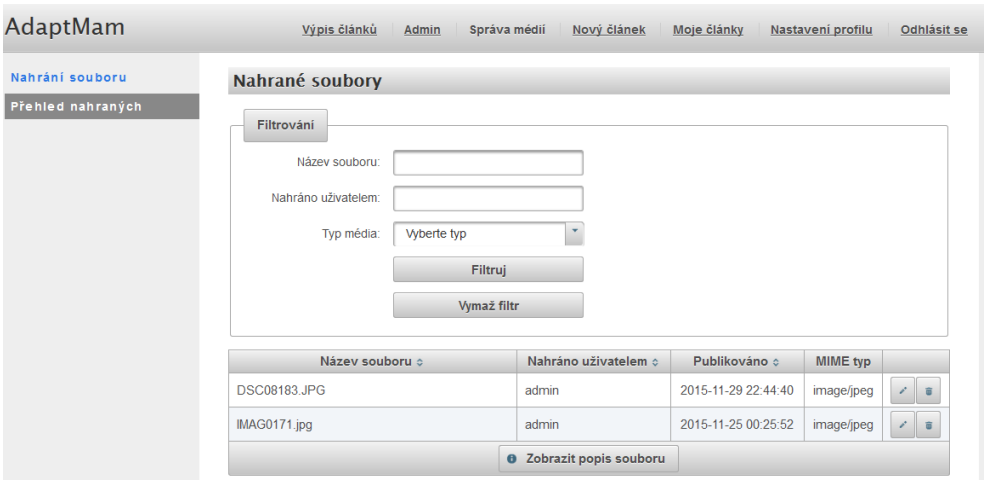
Z kontextového menu lze přejít na přehled nahraných multimediálních souborů do systému, viz obrázek C.18. V tabulkové podobě je zobrazen seznam všech dostupných nahraných multimédií, k dispozici je filtrování, řazení dle různých kritérií a akce pro úpravu, resp. odstranění daného média. V případě první uvedené akce je zobrazeno dialogové okno, jež obsahuje formulářová pole dle typu daného média pro jeho úpravu. Pokud počet záznamů převyšuje výchozí hodnotu nebo hodnotu uloženou v uživatelském profilu, je použito stránkování.



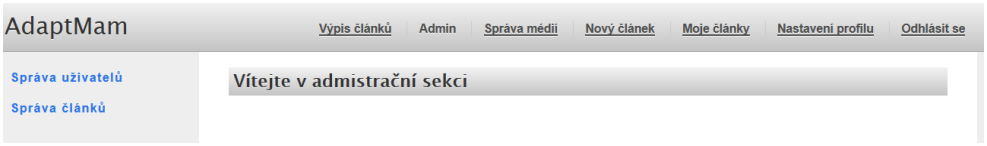
Obrázek C.16: Obrazovka s výběrem typu média k nahrání



Obrázek C.17: Dialogové okno pro nahrání nového obrázku do systému



Obrázek C.18: Obrazovka se základním přehledem nahraných médií



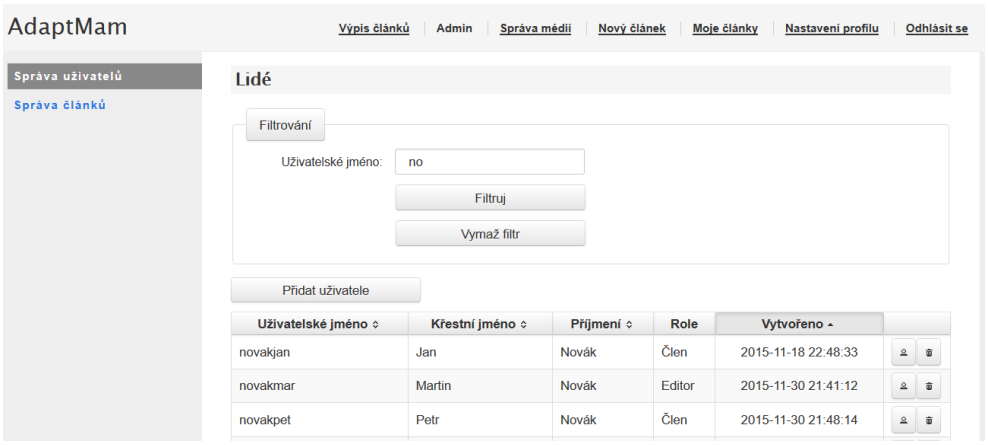
Obrázek C.19: Úvodní obrazovka v sekci pro administrátora systému

C.4 Obrazovky uživatele s rolí Administrátor

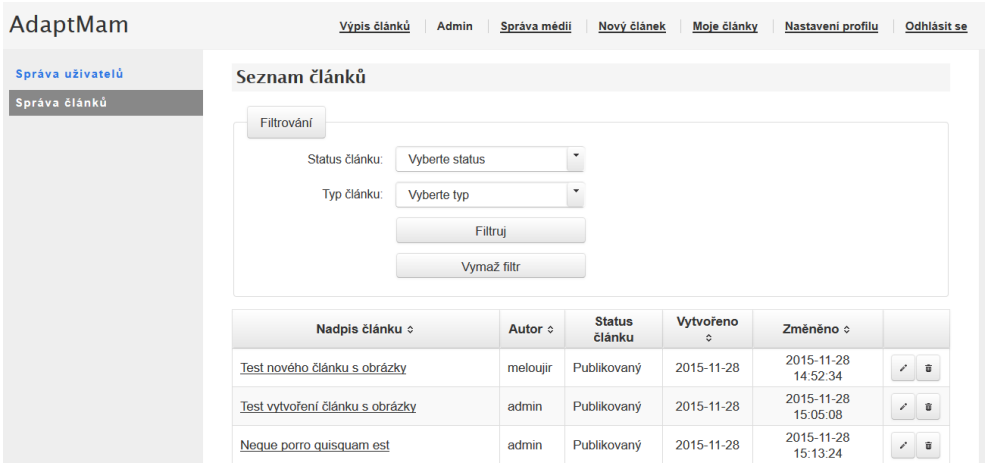
Tato část popisuje sekci, která je dostupná pouze pro administrátora systému. Obrázek C.19 ilustruje uvítací stránku této sekce. Po úspěšném přihlášení do systému se administrátor nachází právě na této uvítací stránce. Z kontextového menu lze pak přejít na administraci uživatelů, resp. článků.

Sekce s administrací uživatelů

Stránku s administrací uživatelů ilustruje obrázek C.20. V rámci této části jsou vypsány všichni dostupní uživatelé v rámci systému, a to v podobě tabulkového výpisu, který lze filtrovat dle uživatelského jména, resp. řadit dle různých kritérií. V pravém sloupci tabulky se na každém řádku tabulky nachází akce, tj. změna uživatelovy role a odstranění uživatele ze systému. V rámci první zmíněné akce je otevřeno dialogové okno, kde administrátor ze seznamu vybere požadovanou roli a akci potvrdí. Pro odstranění uživatele ze systému je nejprve zobrazen potvrzovací dialog. Pokud je počet záznamů vyšší než výchozí hodnota nebo hodnota uložená v uživatelském profilu, je použito stránkování.



Obrázek C.20: Základní obrazovka v rámci administrace uživatelů



Obrázek C.21: Základní obrazovka v rámci administrace článků

Sekce s administrací článků

Stránku s administrací uživatelů ilustruje obrázek C.21, charakter administrace článků je podobný jako v případě administrace uživatelů systému, opět nabízí použití filtrování, řazení dle různých kritérií a akce pro editaci článku, resp. jeho odstranění. Pokud počet záznamů opět převyšuje hodnotu uloženou v uživatelském profilu nebo hodnotu výchozí, je použito stránkování.

Příloha D

Scénáře pro testování s uživateli

Aplikace byla nasazena na server ČVUT FEL `https://ewait.felk.cvut.cz:11181/adapt-mam`.

1. Přihlaste se do systému pod svým účtem. Pokud účet nemáte, zaregistrujte se. Do systému nahrajte nový soubor podporovaného typu. Poté zkontrolujte nahrání nového souboru v seznamu nahraných souborů. Nakonec se odhlaste ze systému.

2. Přihlaste se do systému pod svým účtem. Vytvořte nový článek, který ale ještě nechcete publikovat. Ověřte, že se článek nebude zobrazovat běžným uživatelům. Nakonec se odhlaste ze systému.

3. Přihlaste se do systému pod svým účtem. Zobrazte si pouze svoje články. Najděte nepublikovaný článek a změňte status článku na publikovaný, ověřte jeho zobrazení uživatelům. Ověřte, že nejde odstranit daný soubor v článku z celého systému. Nakonec se odhlaste ze systému.

4. Přihlaste se do systému pod svým účtem. Zobrazte si některý publikovaný článek a ohodnoťte ho. Vložte k němu nějaký komentář. Nakonec se odhlaste ze systému.

5. Přihlaste se do systému pod svým účtem. Nastavte adaptační kritéria a ověřte jejich funkčnost ve výpisu článků. Nakonec se odhlaste ze systému.

6. Přihlaste se do systému jako administrátor. Změňte uživatelskou roli svého účtu na roli běžného uživatele (role člen) a poté se odhlaste. Přihlaste se do systému pod svým účtem a ověřte, že máte přístup pouze k prohlížení obsahu a nastavení profilu. Nakonec se odhlaste ze systému.

Příloha E

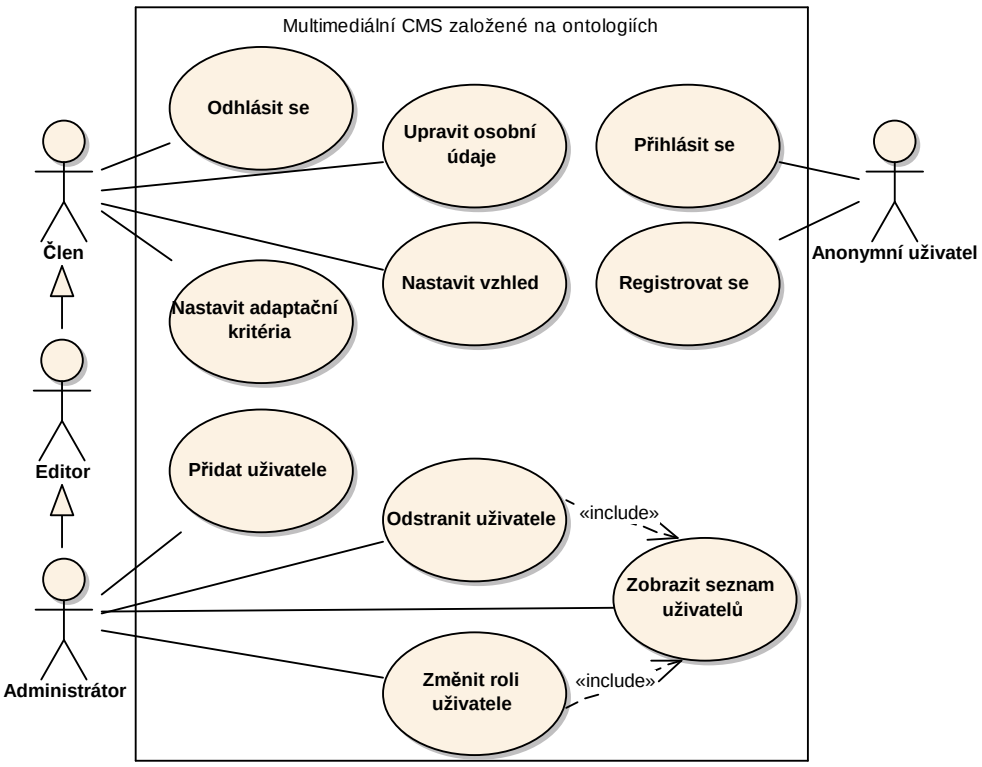
Dotazník po testování s uživateli

Následující seznam obsahuje otázky, které byly součástí dotazníku po projití všech scénářů účastníky testování. V hranatých závorkách je u dané otázky uveden typ odpovědi. Odpověď typu [1-5] znamená hodnocení jako ve škole.

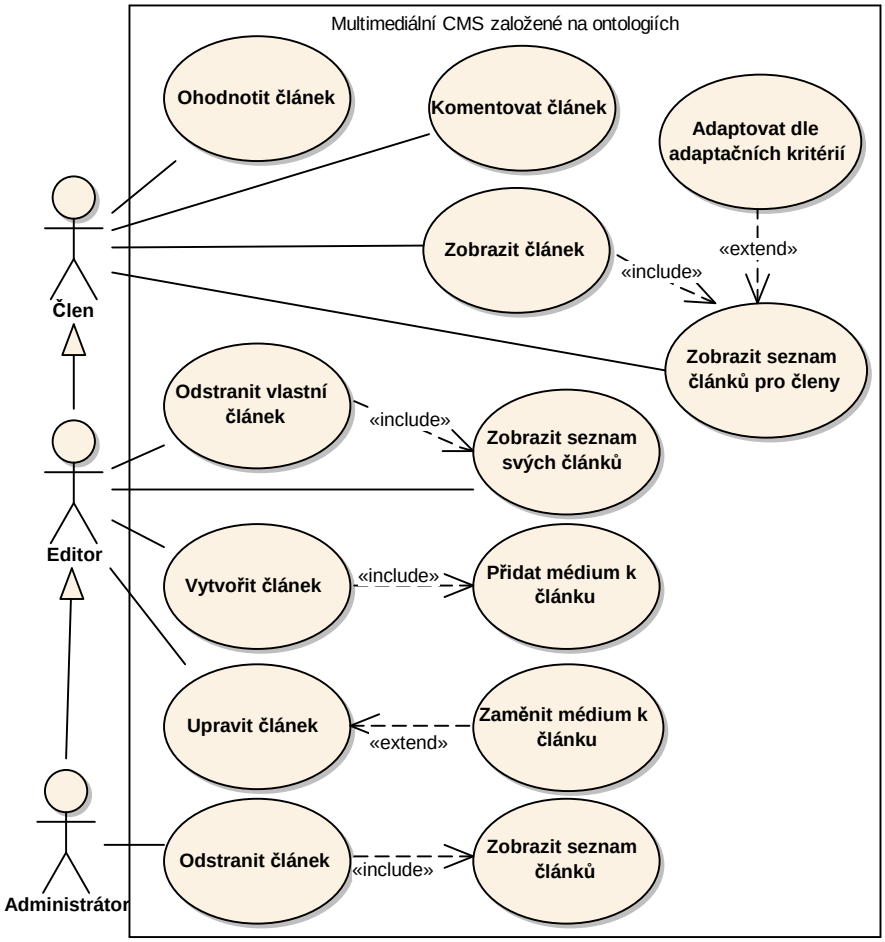
1. Vaše uživatelské jméno bylo: [textové pole]
2. Měli jste před tímto testováním nějaké povědomí o sémantickém webu? [Ano x Ne]
3. Zdálo se Vám ovládání systému přehledné? [1–5]
4. Jak se Vám líbilo základní grafické zpracování (barevná tématika)? [1–5]
5. Zkoušeli jste změnit výchozí barevné téma? [Ano x Ne]
6. Přišel Vám systém hodnocení a komentování článků přehledný? [1–5]
7. Jak byste ohodnotili celkový dojem ze systému? [1–5]
8. Zdá se Vám systém nahrávání souborů a nových článků zvlášť jako vhodný? [Ano x Ne]
9. Zaregistrovali jste požadované změny v rámci filtrování článků s využitím adaptačních kritérií? [Ano x Ne]
10. Bylo provedení nějaké úkonu z uvedených scénářů nejasné? Pokud ano, uveďte číslo scénáře a co Vám dělalo problémy. [textové pole]
11. Napište stručné zhodnocení. Můžete sdělit, co se Vám líbilo, navrhnout nějaké změny, které by podle Vás systém vylepšily atp. [textové pole]

Příloha F

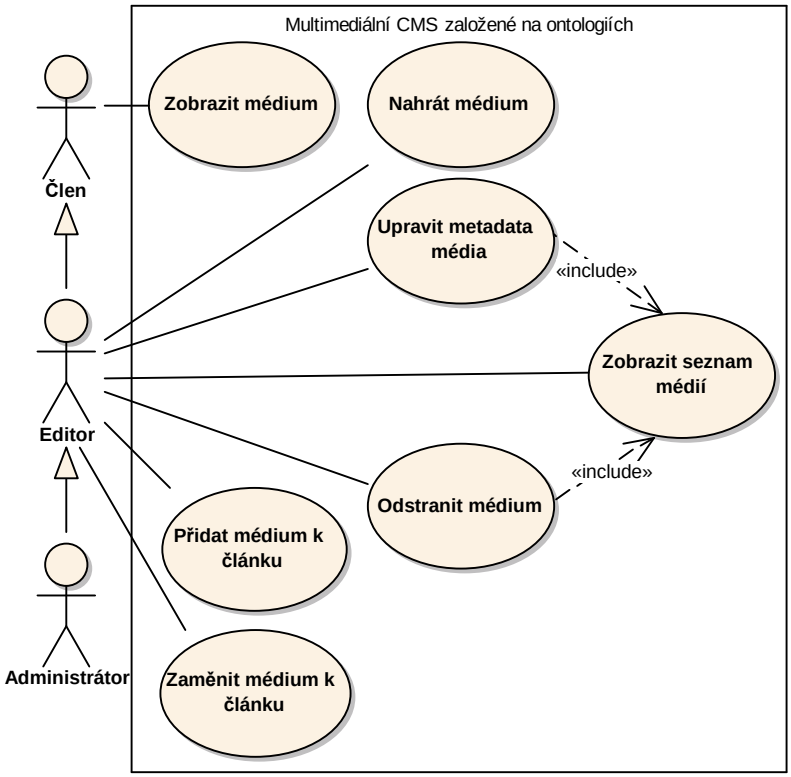
UML modely



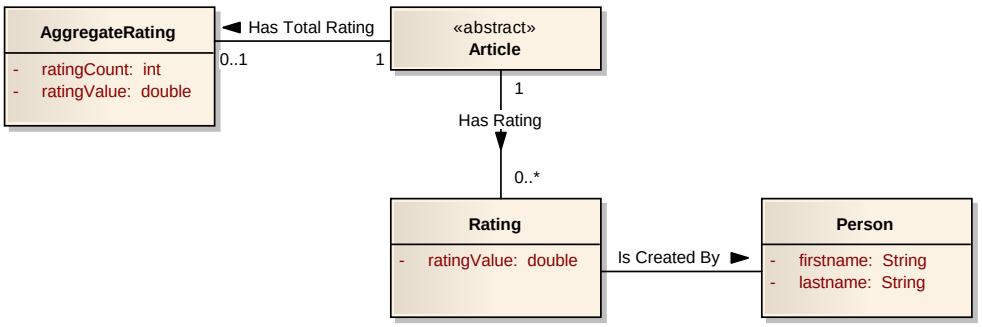
Obrázek F.1: Rámcový model případů užití – lidé



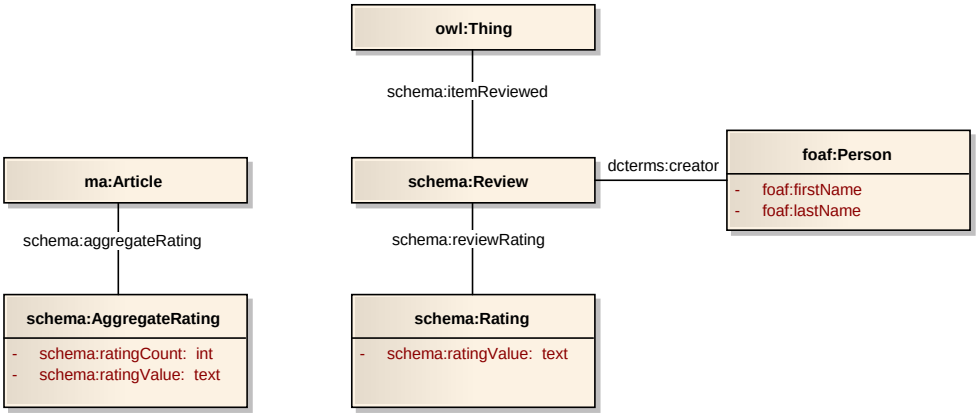
Obrázek F.2: Rámcový model případů užití – články



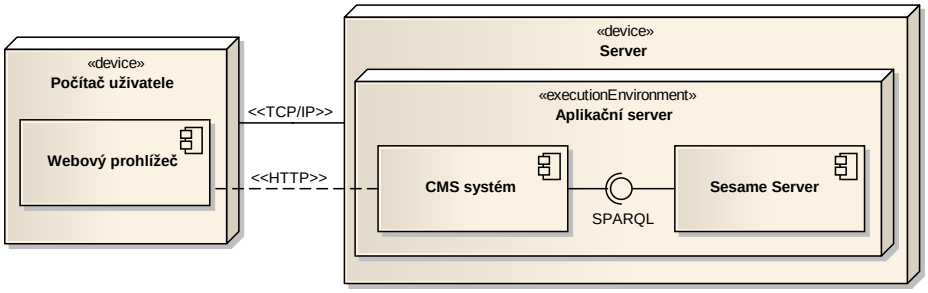
Obrázek F.3: Rámcový model případů užití – média



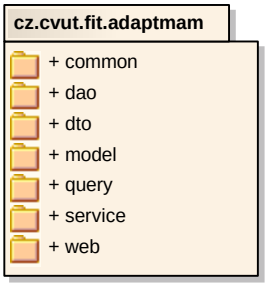
Obrázek F.4: Doménový model zachycující uživatelské hodnocení článků



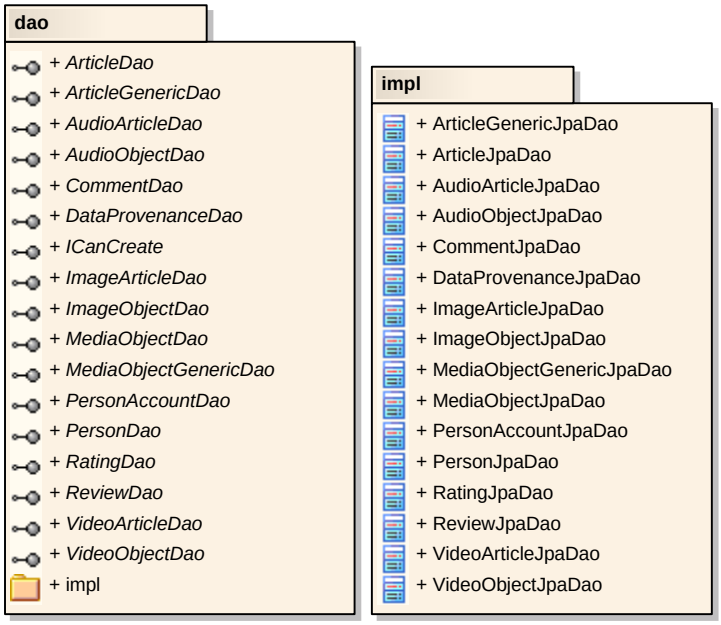
Obrázek F.5: Ontologický model zachycující uživatelské hodnocení článků



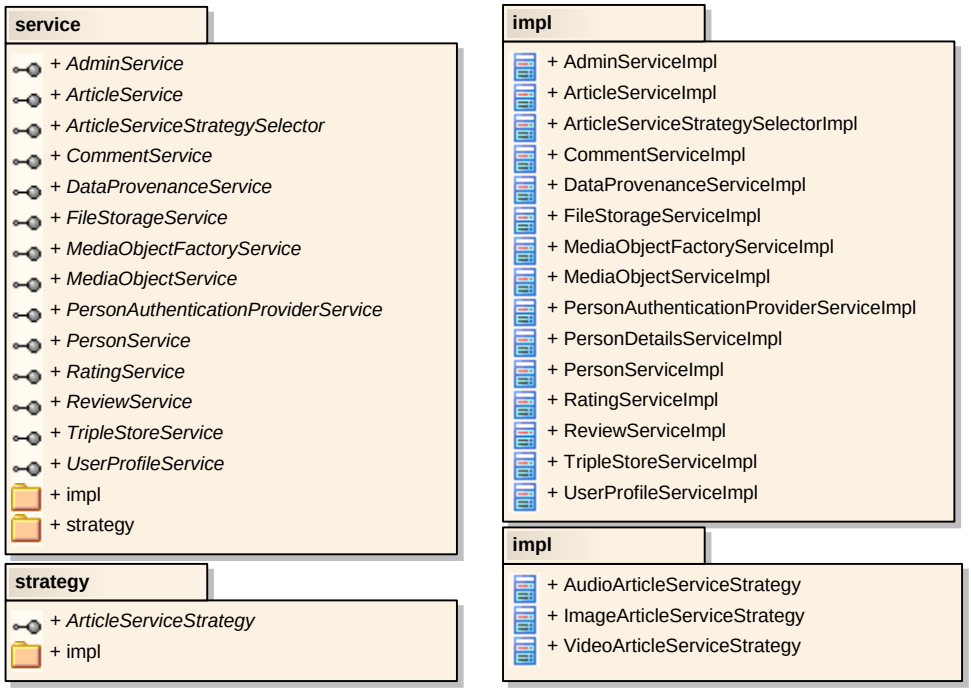
Obrázek F.6: Model nasazení



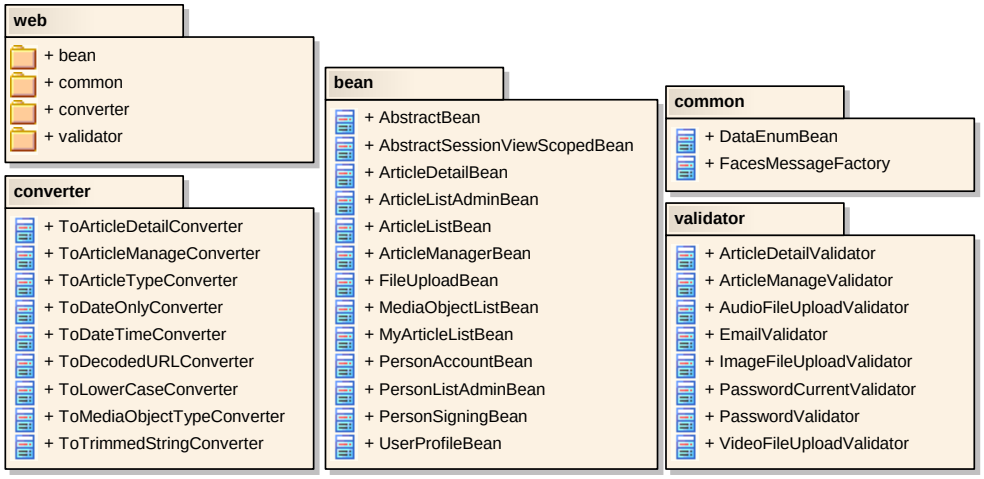
Obrázek F.7: Základní logické členění balíčků prototypu systému



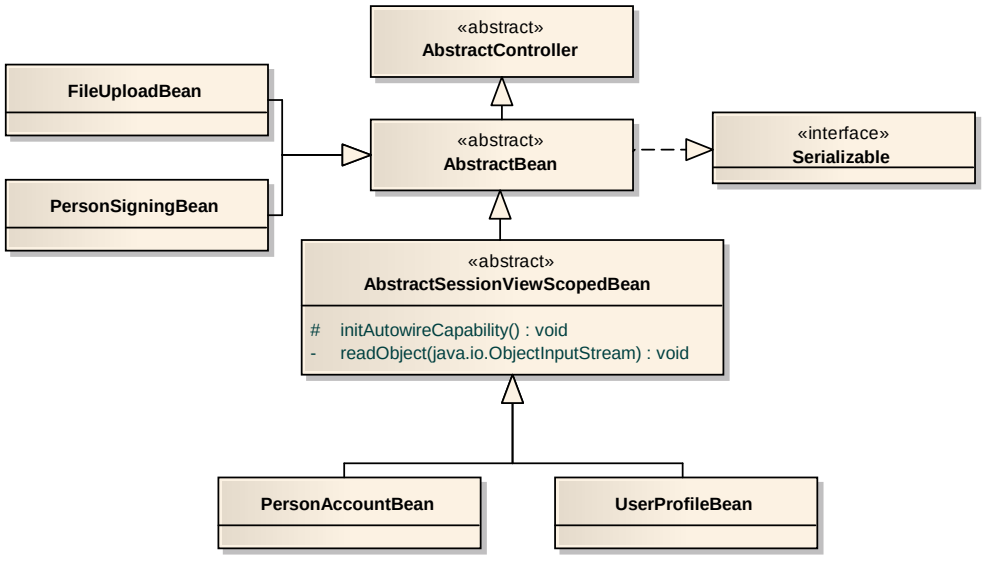
Obrázek F.8: Logické členění DAO prototypu systému



Obrázek F.9: Logické členění servisní vrstvy prototypu systému



Obrázek F.10: Logické členění balíčků webové vrstvy prototypu systému



Obrázek F.11: Architektura backing bean v rámci webové vrstvy

Příloha G

Komponenty pro prezentaci

Níže je uveden stručný popis struktury prvků použité pro prezentaci dat v rámci webové části prototypu systému. Jedná se pouze o přehled hlavních částí, které jsou určeny pro prezentaci. Všechny uvedené části jsou umístěny v adresáři „webapp“. Pro přehlednost nejsou uvedeny všechny soubory, které se v tomto adresáři nacházejí, jako jsou např. konfigurační soubory.

- `resources`
  - `components` ..... Kompozitní komponenty využívané v šablonách
    - `forms` ..... Komponenty pro formuláře
      - `complex` ..... Složitější formulářové prvky
      - `simple` ..... Základní formulářové prvky
  - `css` ..... CSS styly
    - `main` ..... Styly pro hlavní šablonu
    - `welcome` .... Styly pro vedlejší šablonu (pro nepřihlášené uživatele)
  - `images` ..... Statické obrázky pro design
  - `js` ..... Javascriptové skripty
  - `locale` ..... Soubory s lokalizačními texty
- `WEB-INF`
  - `context` ..... Kontextová menu
  - `snippets` ..... Znovu využitelné UI fragmenty
  - `templates` ..... Šablony pro jednotlivé stránky
    - `common` ..... Společné prvky pro všechny šablony
    - `main` ..... Hlavní šablona
    - `welcome` ..... Vedlejší šablona (pro nepřihlášené uživatele)
- ..... Jednotlivé sekce

Příloha **H**

Obsah přiloženého CD

- `data`
  - `asf` ..... framework ASF s upravenými zdrojovými kódy
  - `diagrams` ..... diagramy UML jako EA projekt
  - `mockups` ..... prvotní návrhy uživatelského rozhraní
  - `preinstall` ..... instalace závislostí a počáteční data
  - `testing` ..... výsledky testování
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text`
  - `DP_Meloun_Jiri_2016.pdf` ..... text práce ve formátu PDF