

MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY



Budování metadatového editoru nad GWT

DIPLOMOVÁ PRÁCE

Jiří Kremser

Brno, 2011

Prohlášení

Prohlašuji, že tato diplomová práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Vedoucí práce: RNDr. Tomáš Pitner, Ph.D.

Poděkování

Na tomto místě bych rád poděkoval panu docentu RNDr. Tomáši Pitnerovi, Ph.D. za jeho odborné rady a vstřícný přístup, dále Ing. Petru Žabičkovi za poskytnutí možnosti zpracovat toto zajímavé téma a uvést jej do praxe, Mgr. Pavle Švástové děkuji za cenné informace z knihovnického prostředí a v neposlední řadě své přítelkyni Bc. Viole Kučerové za její trpělivost, kterou se mnou měla při psaní této práce.

Shrnutí

Cílem práce je prozkoumat technologie a návrhové vzory spojené s platformou GWT (*Google Web Toolkit*) zejména MVP (*Model View Presenter*), *EventBus* a *Dependency Injection*. Nastínit problematiku rozličných metadatových formátů a popsat open source repozitář *Fedora Commons*.

Student navrhne a implementuje webovou aplikaci metadatového editoru pro digitální objekty uložené v repozitáři *Fedora Commons*. Editor bude spolupracovat s knihovnickým systémem Kramerius 4. Komunikace s repozitářem bude probíhat prostřednictvím RESTful nebo SOAP webových služeb. Hlavním využitím editoru bude editace strukturálních metadat, *Dublin Core* metadat a metadat ve formátu MODS. Systém by měl v budoucnu umožnit vkládání nových zdigitalizovaných objektů do repozitáře a tvorbu jejich metadat.

Klíčová slova

Metadata, GWT, MODS, Dublin Core, MVP, Model View Presenter, Event Bus, Fedora Commons, Fedora, Repozitář, Kramerius, Web 2.0, RIA, Rich Internet Application, webová aplikace, HTML5, RDF, sémantický web, Java, knihovna, digitalizace, editor, AJAX, RPC, REST, Security, OpenID

Obsah

1	Úvod	1
2	Metadata	2
2.1	<i>Dublin Core</i>	2
2.2	<i>MARC21</i>	3
2.3	<i>MODS</i>	5
2.4	<i>METS</i>	5
2.5	<i>RDF</i>	6
2.6	<i>OAI-ORE</i>	7
2.7	<i>Interoperabilita</i>	7
2.7.1	Z39.50	7
2.7.2	OAI-PMH	8
2.8	Sémanticky web	9
3	Repozitář Fedora	11
3.1	<i>Digitální objekt</i>	11
3.2	<i>Formát FOXML</i>	13
3.3	<i>API</i>	14
4	Kramerius4	15
4.1	<i>Projekt NDK</i>	15
5	Bohaté webové aplikace	17
5.1	<i>AJAX</i>	18
5.2	<i>Reverzní AJAX</i>	20
5.3	<i>Nevýhody AJAXu</i>	21
5.4	<i>Současné RIA rámce</i>	23
6	HTML5	25
6.1	<i>CSS3</i>	25
6.2	<i>Struktura dokumentu</i>	26
6.3	<i>Microdata</i>	27
6.4	<i>SVG a podpora elementu Canvas</i>	28
6.5	<i>Audio a video</i>	28
6.6	<i>App Cache, Web Storage a databáze</i>	29
6.7	<i>Geolokace</i>	29
6.8	<i>Web Workers</i>	30
7	Google Web Toolkit	31
7.1	<i>Historie a budoucnost</i>	32
7.2	<i>Členění GWT aplikace</i>	32
7.3	<i>GWT-RPC</i>	33
7.4	<i>GWT-Platform</i>	35
7.5	<i>Smart GWT</i>	36
8	Návrhové vzory vhodné nejen pro GWT	37
8.1	<i>MVP</i>	37

8.2	<i>Event Bus</i>	39
8.3	<i>Dependency Injection</i>	39
8.4	<i>Command pattern</i>	41
9	Analýza	42
9.1	<i>Zhodnocení stávajícího editoru</i>	42
9.2	<i>Dohoda s firmou Incad</i>	44
10	Návrh	45
10.1	<i>Případy užití</i>	45
10.2	<i>Databáze</i>	47
10.3	<i>Architektura</i>	48
11	Popis metadatového editoru	50
11.1	<i>Instalace a konfigurace</i>	50
11.2	<i>Nasazení</i>	52
11.3	<i>Bezpečnost</i>	53
11.4	<i>Statistiky</i>	55
12	Závěr	56
	Literatura	60
A	Uživatelské prostředí	61

Kapitola 1

Úvod

S přeměnou starého modelu knihovních institucí do dnešní podoby digitálních knihoven roste potřeba spravovat a editovat metadata připojená ke zdigitalizovaným dílům. Metadata jsou nezbytnou součástí samotného díla, protože slouží k indexaci a popisu digitálního objektu. V drtivé většině případů jsou metadata generována automaticky v procesu digitalizace nebo stahována odjinud. Nastávají ale případy, kdy je potřeba metadata buď přidat manuálně, nebo upravit ty automaticky získaná. V takovém případě nezbývá než použít editor metadat.

Tématem mojí diplomové práce je vytvořit webovou aplikaci metadatového editoru pro nově vznikající systém pro zpřístupnění – Kramerius 4. Kramerius 4 zpřístupňuje digitální dokumenty z velmi propracovaného open source repozitáře Fedora. Je tak architektonicky odděleno uchovávání digitálních objektů od jejich zpřístupnění. Metadatový editor bude také pracovat rovnou nad daty z tohoto repozitáře a systém Kramerius 4 upozorní až v momentě provedení změny v metadatech, kvůli následné reindexaci.

Ve své práci nejprve stručně popíší použité technologie a standardy z knihovnického světa. Mezi ně patří metadatové formáty, protokoly pro výměnu metadat, repozitář Fedora, technologie s ním spjaté a systém Kramerius. Jelikož problematika kolem metadat a digitálních knihoven je úzce spjata s myšlenkou sémantického webu, venuji pozornost také jemu.

Prostřední část práce bude věnovaná tvorbě bohatých webových aplikací a technologiím souvisejícím s Webem 2.0, zejména pak použitému apliačnímu rámci *Google Web Toolkit* a návrhovým vzorům, které jsou s ním spojeny. Popíší také nový standard HTML5 a některé novinky, jenž přináší.

Poslední čtyři kapitoly budou obsahovat konkrétní informace o budování metadatového editoru včetně fáze analýzy a návrhu. Tato část práce může sloužit také jako manuál vyvinuté aplikaci. V příloze najde čtenář snímky obrazovky s výslednou webovou aplikací. V kontextu dlouhodobého uchovávání dat je potřebné a výhodné držet se důsledně standardů a využívat open source softwarová řešení. Aplikace je proto šířena pod licencí GNU GPL verze 2 a využívá hojně výhod aplikačních rámčů.

Kapitola 2

Metadata

Metadata, neboli data o datech, jsou strukturovaná data popisující rozličné kvality popisovaných dat. Mezi tyto kvality patří například význam, původ, název či struktura. Jsou velmi často využívána v knihovnickém prostředí pro usnadnění katalogizace; mnoho standardů pro metadata vzešlo právě z tohoto prostředí. S nástupem informačních technologií ale nabývají na významu i v jiných oblastech, protože fakt, že jsou strukturovaná, umožňuje strojům lépe pracovat s významem dat. Může se jednat o indexaci dat podle vybraných atributů, ale i složitější operace typu logického odvozování v sémantických sítích prostřednictvím ontologií.

Obecným problémem metadat je skutečnost, že je musí někdo k původním datům dodat. Existují ale metody strojového učení a dolování v textech, které tento postup mohou do jisté míry automatizovat. Jsou známy docela spolehlivé techniky pro určování klíčových slov v textech jen za pomocí statistických metod [27]. Podobně na techniku „extrakce informací“ lze nahlížet jako na proces doplnění metadat. Pod technikou extrakce informací se skrývá dolování v semi-strukturovaných¹ a nestrukturovaných datech. Problémem těchto kvantitativních metod je určitě závislost na jazyce spojená s chybami v datech způsobených nedokonalou přesností OCR systémů. Chyby v datech sice lze identifikovat a označit za šum, ale například v historických rukopisech je přesnost OCR žalostná. Automatizovat tvorbu metadat v knihovnickém prostředí je navíc náročné také z toho důvodu, že na knihách se bibliografické údaje nevyskytují vždy na stejných místech. Například datum vydání může být na začátku knihy, na konci, může chybět nebo může být napsáno římskými číslicemi. Nejspolehlivější je v tomto ohledu stále člověk. Protože je to ale práce velmi monotonné, tak se mnohdy neprovádí.

V této kapitole budou popsány nejběžnější metadatové formáty používané v knihovnickém prostředí, yly pro výměnu metadat a stručně také sémantický web.

2.1 Dublin Core

Dublin Core je množina metadatových elementů sloužících k popisu libovolného zdroje. Standard² vznikl roku 1995 v Dublinu (Ohio, USA), odtud také název. Původním záměrem bylo vytvořit velmi obecný rámec pro popis elektronických zdrojů, aby se tak usnadnilo

1. Často (X)HTML, to pak místo termínu „extrakce informací“ používáme spíše termín „dolování z webu“.
2. *Dublin Core* je ISO standard a je definován v ISO 15836:2009 a NISO Z39.85-2007 a také v RFC 5013-2007.

jejich vyhledávání. Díky své obecnosti dokáže popsat celou škálu zdrojů, mezi které patří: knihy, videa, nahrávky, mapy, hudební partitury a mnohé další. Nejčastější instance *Dublin Core* metadat je ve formě samotného XML dokumentu, může se ale objevit i jako podčást nějakého kontejnerového formátu či ve formě RDF.

Dublin Core se vyskytuje ve dvou variantách: **jednoduchý** (někdy označovaný jako nekvalifikovaný) a **kvalifikovaný**. Jednoduchý *Dublin Core* obsahuje patnáct elementů, z nichž každý je nepovinný, opakovatelný a nezáleží na pořadí jeho výskytu v zápisu metadat. Elementy jsou vypsány v následující tabulce.

Title	Creator	Subject	Description	Publisher
Contributor	Date	Type	Format	Identifier
Source	Language	Relation	Coverage	Rights

Tabulka 2.1: Základní *Dublin Core* elementy.

Kvalifikovaná varianta obsahuje všech základních patnáct elementů a přidává ještě další tři: **Audience**, **Provenance** a **RightsHolder**. Mimo tyto tři elementy umožňuje kvalifikovat jak elementy, tak i jejich hodnoty. Kvalifikovaná hodnota se může vyskytnout například v elementu **Date** v podobě řetězce 2010-12-22:IS08601. Kvalifikátory pro elementy upřesňují význam daného elementu; opět na příkladu elementu **Date** lze upřesnit o jaké datum se jedná – **Date.Created**, **Date.Modified**, **Date.Issued** a podobně. V případě, že aplikace nerozumí kvalifikovanému DC, pracuje s elementy, jako by byly z jednoduché sady.

„Na základě výzvy pracovní skupiny *Dublin Core in Multiple Languages* pracuje Knihovnicko-informační centrum Masarykovy university v Brně ve spolupráci se specialisty v oblasti knihoven na vytvoření české verze metadatového standardu *Dublin Core* pro popis a podporu vyhledávání elektronických informačních zdrojů v českém prostředí.“ —[14]

2.2 MARC21

MARC (*Machine-Readable Cataloging*) je podstatně starší standard než *Dublin Core*. Byl vyvinut v šedesátých letech národní knihovnou USA – *Library of Congress (LoC)* a jeho navržení odpovídá potřebám tehdejší doby [30]. Je dobré si uvědomit, že jazyk SQL navrhli Chamberlin a Boyce roku 1974 a o dva roky později pak Peter Chen navrhl ERD diagram. MARC přišel do doby bez relačních databází, kde úspora každého bitu byla podstatná. To je ale ve sporu s požadavkem na čitelnost metadat lidmi. Navzdory jeho nečitelnosti a zastaralosti je MARC, respektive MARC21, dnes v knihovnách nejrozšířenější a je využíván rozšířeným knihovnickým systémem Aleph. MARC21 je novější varianta MARCu, která ale nic nemění na způsobu zápisu metadat, ten je sekvenční. Pro představu je zde uveden začátek MARC21 záznamu z Májeho.

```

LEADER 01297nam a22004212a 4500
001     000039700
003     CZ BrMZK
005     19951201143423.0
008     951201s1964 xr f cze d
015     $a cnb000480694
040     $a BOA002 $b cze
100 1   $a Mácha, Karel Hynek, $d 1810-1836 $7 jk01072915
245 10  $a Máj $c Karel Hynek Mácha
...

```

První sloupec udává identifikátor pole, prostřední je indikátor a znak následující bezprostředně za znakem „\$“ je identifikátor podpole. Kombinací těchto tří entit lze určit význam daného textu. Například v poli „100“ za řetězcem „\$a“ následuje vždy jméno autora, za znakem \$d rok narození/úmrtí a podobně. MARC21 poskytuje pro knihovnické účely velmi vyčerpávající popis zdrojů, bohužel jeho tvorba není bez zaškolení snadná a jeho zpracování počítáčem také není nejjednodušší. Existuje ale formát MARC21XML, který využívá výhod formátu XML. Elementy MARC21XML jsou *record*, *controlfield*, *datafield* a *subfield*. Ty odpovídají jednotlivým výše popsaným částem MARC záznamu. Výhoda MARC21XML spočívá v tom, že LoC zpracovala XSLT šablony pro konverzi z MARC21XML do DC nebo MODS. Pro úplnost uvádíme předchozí záznam zkonzertovaný do MARC21XML.

```

<collection xmlns="http://www.loc.gov/MARC21/slim">
  <record>
    <leader>01297nam a22004212a 4500</leader>
    <controlfield tag="001">000039700</controlfield>
    <controlfield tag="003">CZ BrMZK</controlfield>
    <controlfield tag="005">19951201143423.0</controlfield>
    <controlfield tag="008">951201s1964 xr f cze d</controlfield>
    <datafield tag="015" ind1=" " ind2=" ">
      <subfield code="a">cnb000480694</subfield>
    </datafield>
    <datafield tag="040" ind1=" " ind2=" ">
      <subfield code="a">BOA002</subfield>
      <subfield code="b">cze</subfield>
    </datafield>
    <datafield tag="100" ind1="1" ind2=" ">
      <subfield code="a">Mácha, Karel Hynek,</subfield>
      <subfield code="d">810-1836</subfield>
      <subfield code="7">jk01072915 </subfield>
    </datafield>
    <datafield tag="245" ind1="1" ind2="0">
      <subfield code="a">Máj</subfield>
      <subfield code="c">Karel Hynek Mácha</subfield>
    </datafield>
  ...

```

2.3 MODS

Standard DC je velmi obecný, dobře navržený a jednoduchý, ale nepostačuje na podrobný popis. Naopak MARC21 je zase příliš složitý, rozsáhlý a úzce zaměřený. Kompromisem mezi nimi je standard MODS (*Metadata Object Description Schema*) navržený roku 2002 taky americkou LoC. Poslední verze 3.4 vyšla v červnu 2010. MODS je založen na formátu XML a obsahuje dvacet základních elementů zobrazených v tabulce [2.2]. Struktura ale není „plochá“ jako v případě DC, ale elementy mohou obsahovat další podelementy. Každý element je, stejně jako u DC, nepovinný a opakovatelný. Některé elementy umožňují vložit element z jiného jmenného prostoru a rozšířit tak metadatový záznam o externí XML dokument. U mnoha elementů se vyskytuje atribut **authority**, který určuje externí autoritu, to znamená, že hodnota elementu bude splňovat určité požadavky. Může se jednat o kontrolovaný slovník, tedy výčet možných hodnot (např. *iso639-2b* pro určení jazyka), nebo požadavek na formát dat například v případě časových údajů.

<i>titleInfo</i>	<i>name</i>	<i>typeOfResource</i>	<i>genre</i>	<i>originInfo</i>
<i>language</i>	<i>physicalDescription</i>	<i>abstract</i>	<i>tableOfContents</i>	<i>targetAudience</i>
<i>note</i>	<i>subject</i>	<i>classification</i>	<i>relatedItem</i>	<i>identifier</i>
<i>location</i>	<i>accessCondition</i>	<i>part</i>	<i>extension</i>	<i>recordInfo</i>

Tabulka 2.2: Základní MODS elementy.

Zajímavý je element *relatedItem*, protože jeho obsahem může být další dokument ve formátu MODS. Z toho vyplývá, že toto rekurzivní zanořování může pokračovat do nekonečna. To se ovšem samozřejmě nedoporučuje. V metadatovém editoru dovoluji nastavit hloubku zanoření tohoto elementu z praktických důvodů kvůli výkonu aplikace.

2.4 METS

Pro úplnost stručně popíší i tento standard, i když s ním editor nepracuje. Standard METS (*Metadata Encoding and Transmission Standard*) slouží jako kontejner pro ukládání metadat MARC, MODS, DC, NISOIMG, TEI a dalších. Kromě popisných metadat může obsahovat také administrativní a strukturální metadata. Administrativní metadata můžou popisovat například formát dokumentu, rozlišení v případě obrazových dat, údaje o jeho digitalizaci, autorská práva a další. Strukturální metadata, jak už název napovídá, popisují strukturu digitálního objektu. V případě knih a novin se totiž většinou vytvářejí metadata o každé stránce, která je potom potřeba nějak komponovat do větších celků. METS umožňuje v elementu *structMap* nadefinovat související digitální objekty, většinou se jedná o jeho podčásti, například právě stránky, články nebo kapitoly. V elementu *structMap* se uvádí jen identifikátory podčástí.

2.5 RDF

RDF (*Resource Description Framework*) je soubor specifikací konsorcia W3C pro konceptuální modelování objektů a relací mezi nimi. Základní jednotkou pro popis je v RDF trojice (subjekt x, relace R, objekt y) mající význam „x je v relaci R s objektem y“. Aby si každý popisující nevytvářel své vlastní relace, jsou k dispozici kontrolované slovníky s relacemi ve formě XML jmenných prostorů³. Subjekty a objekty bývají označovány identifikátorem URI. Na RDF lze také nahlížet z pohledu logiky jako na množinu pravdivých tvrzení, faktů, tvaru $R(x, y)$, kde R je predikát. Jelikož RDF dovoluje tvořit tvrzení o tvrzení prostřednictvím tzv. reifikace, odpovídají RDF triplety dokonce predikátům vyššího řádu. RDF lze chápat také jako pojmenovaný orientovaný graf, kde vrcholy grafu, ze kterých vede hrana, odpovídají subjektům, vrcholy, do kterých vede hrana, objektům a pojmenované hrany odpovídají relacím.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:kuchyne="http://muj-slovnik.cz/kuchyne">
  <rdf:Description rdf:about="http://lide.cz/Ema">
    <kuchyne:mele>maso</kuchyne:mele>
  </rdf:Description>
</rdf:RDF>
```

Příklad popisuje skutečnost, že „Ema mele maso.“. Hodnota atributu `rdf:about` elementu `rdf:Description` je subjekt, element `kuchyne:mele` je predikát a jeho obsah je objekt.

RDF se pro svoji matematickou podstatu často používá k reprezentaci ontologií. Ontologie, respektive ontologické datové modely, slouží zejména v oblasti umělé inteligence k reprezentaci doménové znalosti. Dle mého názoru jsou ontologie vhodné spíše pro specifickou doménu, kde se dají využít ve znalostních a expertních systémech. Zajímavým projektem je databáze Freebase [20], která se snaží být „vysší ontologií“ (*upper ontology*), tzn. neomezovat se jen na jedinou doménu⁴. Freebase staví na principech Webu 2.0 a zapojuje komunitu do tvorby, ale také do využívání jejich ontologie. Delegace výstavby otevřených ontologií na komunitu spadá pod iniciativu *LinkedData* (linkeddata.org).

Existuje celá řada specifikací a jazyků využívajících RDF jako například: OAI-ORE, SPARQ (SPARQL Protocol and RDF Query Language), OWL (Web Ontology Language), RIL (RDF Inference Language), 3Store a další. Ty jsou ale nad rámec této práce.

3. Pozorný čtenář jistě chápe, že to problém neřeší, protože si může každý tvořit svůj slovník.

4. O tom, že projekt Freebase je perspektivní, svědčí i fakt, že jej od společnosti Metaweb odkoupila společnost Google 16. 7. 2010.

2.6 OAI-ORE

Motivací pro vznik standardu OAI-ORE (*Open Archives Initiative, Object Exchange and Reuse*) byla potřeba jednoznačně identifikovat množinu nebo kolekci zdrojů jako jeden složený zdroj. Jako typický případ užití standardu OAI-ORE bývá často uváděno označení souvisejících výstupů jednoho vědeckého experimentu jako jeden samostatný zdroj. Různé přílohy, protokoly, statistiky měření a podobně tak budou svázány se samostatnou textovou částí, která může být dostupná samozřejmě ve více formátech. Podobně různé verze téhož obrázku, lišící se pouze svým rozlišením, například na webu *Flicker*, mohou tvořit jeden složený zdroj. Takovýto zdroj může být součástí jiného složeného zdroje a může být strojově zpracováván lépe než jeho jednotlivé části bez vazeb na celek. Záměrem zde bylo využít tento koncept v sémantickém webu a také v automatizaci vytváření tak zvaných „*splash page*“. Reprezentativním příkladem takového stránky může být stránka⁵ na arXiv.org s názvem vědeckého článku, s abstraktem, s odkazy na stažení článku v různých formátech a podobně, kde se všechny údaje generují automaticky na základě metadat agregovaného zdroje.

OAI-ORE agregovaný zdroj zpravidla popisuje i samotnou agregaci, tedy svou strukturu, své části a své reprezentace. Poté obsahuje alespoň jednu mapu zdroje, která také popisuje samotnou aggregaci. Agregace může být popisována různými mapami zdrojů, což slouží k vytváření pomyslných pohledů. Existuje pěkná analogie se souhvězdími. Souhvězdí je pouze člověkem pojmenovaný tvar vzdálených jasných hvězd, ale nic nevypovídá o vlastnostech tohoto uskupení, jako například vzdálosti mezi hvězdami, gravitační síle apod. Stejně tak mapa zdroje slouží vždy pro nějaký účel, ale opravdové relace mezi objekty zachycuje aggregace. Jak aggregace, tak mapa zdroje musí být serializovány do formátů RDF nebo ATOM.

2.7 Interoperabilita

„*Interoperability: The compatibility of two or more systems such that they can exchange information and data and can use the exchanged information and data without any special manipulation.*“ —Arlene G Taylor, 2004

2.7.1 Z39.50

Protokol Z39.50 [42] je standardizován podle ANSI/NISO Z39.50 a ISO 23950 a je udržován americkou *Library of Congress*. Jedná se o stavový protokol na bázi klient-server, který je nezávislý na operačním systému a databázi. Protokol podporuje celou řadu akcí, mezi které patří třeba *browse*, *search*, *sort* a *retrieval*. Pro vyhledávání se mohou používat protokoly ISO 8777, SQL, CCL a jiné. Vyhledávací dotaz využívá specifickou množinu atributů, která musí být explicitně zmíněna. Například množina atributů *bib-1* [12] slouží pro bibliografické knihovnické systémy.

5. Např. <http://arxiv.org/abs/hep-th/0507171>.

2.7. INTEROPERABILITA

ZING (Z39.50 International: Next Generation) je název pro úsilí zmodernizovat protokol Z39.50 pro webové prostředí. K tomuto úsilí patří specifikace SRW (*Search/Retrieve Web Service*) a SRU (*Search/Retrieve URL Service*). SRW využívá SOAP webové služby, zatímco SRU funguje pouze přes HTTP GET, kde dotaz se kóduje do URL; v podstatě se tak jedná o REST webovou službu, která je „jen pro čtení“. Oba protokoly využívají velmi obecný jazyk CCL (*Contextual Query Language*, dříve *Common Query Language*). Na velmi podobném principu jako SRU funguje dnes velmi rozšířený protokol OAI-PMH.

2.7.2 OAI-PMH

OAI-PMH (*Open Archives Initiative, Protocol for Metadata Harvesting*) je protokol navržený roku 1999 s cílem umožnit sdílení metadat mezi různými repozitáři a umožnit tak federativní vyhledávání napříč větším množstvím repozitářů. To sice umožňuje i protokol Z39.50 ve své základní variantě, ale nevýhodou je, že v případě Z39.50 degraduje rychlosť vyhledávání ve více repozitářích na rychlosť odezvy nejpomalejšího repozitáře. To je však obecně problém všech meta-vyhledávačů. Oproti tomu princip federativního hledání staví na průběžném sklízení metadat a následném hledání v lokální bázi.

OAI-PMH identifikuje tyto dvě strany: poskytovatelé dat a poskytovatelé služeb. Poskytovatelé dat nabízejí metadatové zdroje přes protokol OAI-PMH a umožňují tak poskytovatelům služeb tyto zdroje v pravidelných intervalech sklízet a budovat nad nimi nadstavbové služby. Protokol neklade omezení na formát metadat, ale nařizuje, aby pro každý zdroj byla dostupná verze v nekvalifikovaném *Dublin Core* a aby jako nosný formát bylo použito XML. Metadatové záznamy lze logicky seskupovat do množin, přičemž kardinalita tohoto seskupení je M:N. V Moravské zemské knihovně jsou například definovány množiny s metadaty o starých mapách, starých fotografiích, rukopisech a podobně.

Technicky je protokol řešen na bázi protokolu HTTP, kdy se na URL se službou pošle požadavek GET nebo POST a služba vrátí XML s odpovědí. XML s odpověďí může obsahovat například výpis podporovaných formátů, výpis identifikátorů metadatových záznamů, informace o repozitáři, názvy množin metadat, nebo samotná metadata. V případě sklizně velkého množství metadat funguje následující scénář. Sklizející pošle požadavek, například GET, na

```
http://repozitar?verb=ListRecords&metadataPrefix=mods&from=2010-01-01
```

Obdrží XML s prvními n záznamy ve formátu MODS, změněnými od uvedeného data, kde n je nastavitelná proměnná. Se záznamy je v odpovědi také tzv. *resumptionToken* s určitou platností a hodnotou. Pro obdržení XML s následujícími n záznamy pak musí sklízející vydat další požadavek na

```
http://repozitar?verb=ListRecords&resumptionToken=secret
```

a situace se opakuje, dokud existují další záznamy, nedojde k chybě, nebo nevyprší platnost tokenu. Platnost je také nastavitelná ve vlastnostech systému OAI-PMH poskytovatele dat.

2.8 Sémanticky web

„I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web – the content, links, and transactions between people and computers. A ‘Semantic Web’, which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The ‘intelligent agents’ people have touted for ages will finally materialize.“

—Tim Berners-Lee, 1999

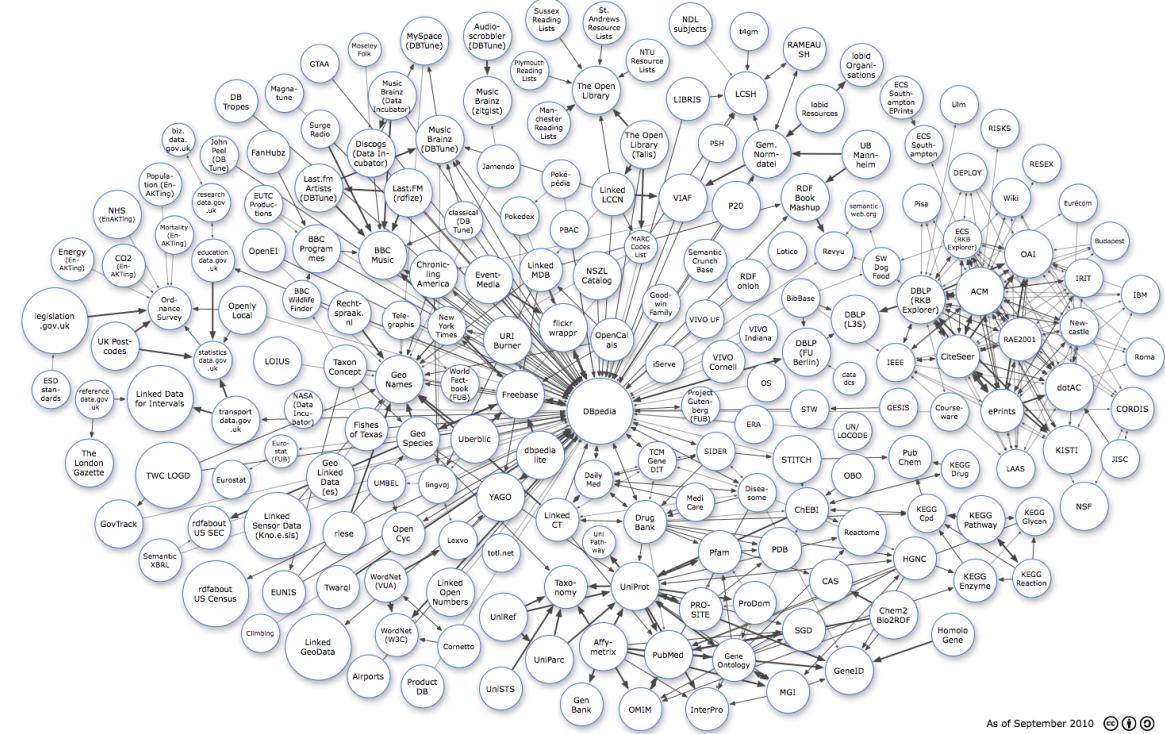
Tim Berners-Lee je člověk, který v roce 1989 v CERNu přišel s myšlenkou webu a následně implementoval jeho prapočátky. Myšlenky sémantického webu, jichž je také původcem, jsou úzce spjaty s umělou inteligencí a s problémem porozumění sémantice informace strojem. Z uvedené citace bych zdůraznil slovo „emerge“, které je typické pro modelování „bottom-up“. Sémantický web vlastně využívá modelování odspodu. Jsou známy cíle, tzn. kooperující autonomní softwaroví agenti jednající v prospěch uživatele, ale bez potřebné infrastruktury a dat srozumitelných strojům je to pořád pouhé sci-fi. Zastavám názor, že pokud není systém nebo agent schopen autoreference, není schopen ani selského rozumu [22]. Autoreferenční tvrzení se v RDF dají modelovat, což podle Gödelových vět o neúplnosti znamená, že existují nedokazatelná pravdivá tvrzení. Tento fakt ale v principu ničemu nevadí, protože nepotřebujeme neomylné softwarové agenty, ale spíše flexibilní a robustní agenty využívající selského rozumu.

Sémantický web může sloužit ale mnohem pragmatičtěji už dnes, a to například ve formě zacílené reklamy podle preferencí uživatele. Nebo spojením s dalšími novými technologiemi, jako například s QR kódy a s chytrými telefony, může nabízet informace o zboží, včetně recenzí nebo užitečných informací spojených s polohou uživatele typu: „Daný produkt je v obchodě o 200 metrech vedle levnější“ a podobně. Na podobné úkony stačí agenti zacílení na specifický úkol, kde se o inteligenci moc hovořit nedá. Jednou z největších výzev sémantického webu je zlepšení vyhledávání informací v dnešním heterogenním prostředí Internetu. Ke všem těmto aplikacím je ale nejprve potřeba, aby Internet byl jedna velká ontologie, nadneseně řečeno. Tim Berners-Lee založil iniciativu *Linked Data* a navrhl čtyři základní pravidla [29].

1. Používejte URI jako jména pro identifikaci věcí.
2. Používejte HTTP URI, aby lidé a agenti mohli vyhledat věci podle jejich URI – dereferencovat.
3. Poskytněte užitečné informace o dané věci prostřednictvím RDF/XML, když je věc vyhledána.
4. Poskytněte rovněž odkazy na související věci s danou věcí, když je věc vyhledána.

2.8. SÉMANTICKÝ WEB

Iniciativa *Linked Data* je především o zapojení komunity do tvorby ontologií prostřednictvím RDF, ale také o propojování už existujících dat napříč ontologiemi. K *Linked Data* se hlásí například *DBpedia* (v dubnu 2010 obsahovala 3,4 miliónu věcí), což jsou znalosti z encyklopédie Wikipedia převedené do RDF trojic, ale také už zmiňovaný projekt Googlu – Freebase a další. Velký krok vpřed spatruji v HTML5 a *Microdata* [31], které popíší dále v této práci v sekci 6.3. Ačkoli standard *Microdata* není tak expresivní jako RDF, je natolik jednoduchý a prakticky využitelný, že jej weboví návrháři společně s mikroformáty využívají.



Obrázek 2.1: The Linking Open Data cloud diagram [28].

Kapitola 3

Repozitář Fedora

Fedora (*Flexible and Extensible Digital Object and Repository Architecture*) nemá nic společného se známou distribucí Linuxu společnosti Red Hat. Jak zkratka napovídá, jedná se o propracovaný repozitář digitálních objektů. Repozitář Fedora vznikl jako výzkumný projekt na Cornell University roku 1997; založili jej Sandy Payette a Carl Lagoze. Druhý jmenovaný je mimo jiné také jedním z autorů standardů OAI-PMH a OAI-ORE. Ve své první verzi využívala Fedora pro komunikaci standard CORBA. O dva roky později byl repozitář na University of Virginia přepracován na technologii Java Servlet s relační databází a testován na zátěž deseti miliónů digitálních objektů. Roku 2001 dostaly obě univerzity grant z Mellonovy nadace a Fedora se tak pomalu začala přetvářet do dnešní podoby. Současná verze Fedory je verze 3.4.1, je open source a je postavena plně na webových službách a XML. Za zmínku určitě stojí i rok 2007, kdy byl spuštěn projekt Fedora Commons, což je komunitně založený projekt na principech Webu 2.0. Z důvodu zjednoznačnění významu termínu „Fedora“ se často používá označení „Fedora Commons“ i pro repozitář.

Fedora byla už od počátku koncipována jako univerzální repozitář pro heterogenní digitální zdroje. To znamená, že před samotným nasazením do konkrétní instituce je potřeba provést poměrně složité nastavení a instalaci. Repozitář Fedora po instalaci nenabízí žádné uživatelsky přívětivé rozhraní, po instalaci se sice nainstaluje i webová aplikace pro hledání a zobrazování digitálních objektů, ta je ale určena spíše vývojářům a určitě nenabízí takové komfortní rozhraní jako například repozitář *DSpace*. Na druhou stranu Fedora nabízí platformu, na které může aplikace sloužící pro zpřístupnění zdrojů snadno vzniknout. To je i případ aplikace Kramerius (viz kapitola 4) nebo open source projektu *Fez* [19].

3.1 Digitální objekt

Základní jednotkou pro ukládání digitálních záznamů je digitální objekt. Je to identifikátorem jednoznačně určitelná entita, která může obsahovat datové proudy. Jednotlivé datové proudy představují reprezentace digitálního objektu. V případě obrazových záznamů se jedná často o náhled a různé kvality obrázku, v případě naskenovaných stránek může existovat datový proud pro OCR a podobně. Kromě volitelných datových proudů existují i proudy rezervované Fedorou. Jsou to proudy AUDIT, DC, RELS-EXT a RELS-INT. Proud AUDIT slouží k zaznamenávání informací o změnách digitálního objektu, DC je proud s metadaty v *Dublin Core* a RELS-EXT, resp. RELS-INT, slouží k označení souvisejících digitálních objektů pomocí RDF. Situace je znázorněna na obrázku 3.1.

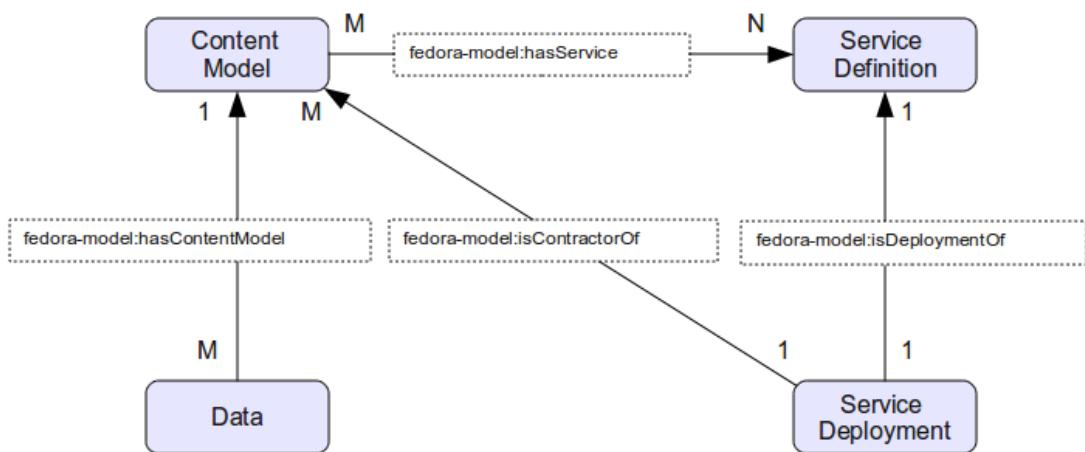
3.1. DIGITÁLNÍ OBJEKT



Obrázek 3.1: Obsah digitálního objektu (přepracováno z [18]).

Fedora podporuje několik typů digitálních objektů. Jsou jimi: datové objekty (*Data*), objekty modelu (*Content Model*), objekty s definicí služeb (*Service Definition*) a objekt s definicí nasazení služby (*Service Deployment*). Na iniciální vložení všech nedatových¹ digitálních objektů lze nahlížet ještě jako na instalaci samotného repozitáře. Objekty modelu tvoří společně s definicemi služeb doménový model repozitáře. Objekty s definicemi služeb definují rozhraní chování instančních objektů modelu. Relace mezi třídami objektů daných typů je zobrazena na obrázku 3.2.

1. Z čtyř výše jmenovaných všechny kromě typu *Data*.



Obrázek 3.2: Typy digitálních objektů repozitáře Fedora (přepracováno z [17]).

3.2 Formát FOXML

Pro uchovávání digitálních objektů v repozitáři slouží formát FOXML 1.1, což je v podstatě XML validní proti schématu `foxml1-1.xsd`². Pro vkládání (v terminologii repozitářů tzv. *ingest*) nebo exportní operace lze využít i formáty FOXML 1.0, METS 1.0, METS 1.1, ATOM 1.1, nebo ATOM Zip 1.1. Při vložení digitálního objektu v jiném než výchozím formátu je provedena konverze do výchozího, a ten je následně uchováván. Struktura digitálního objektu (obrázek 3.1) je poměrně intuitivně serializována do formátu FOXML následovně.

```

<digitalObject PID="uniqueID">
  <objectProperties>
    <property/>
    ...
  </objectProperties>
  <datastream ID="datastream_1" control_group="X">
    <!-- content -->
  </datastream>
  ...
  <datastream ID="datastream_N" control_group="E">
    <datastreamVersion ID="datastream_N.0">
      <contentLocation REF="http://link/to/data" TYPE="URL"/>
    </datastreamVersion>
  </datastream>
</digitalObject>
  
```

2. <http://www.fedora.info/definitions/1/0/foxml1-1.xsd>.

Datový proud obsahuje atribut `control_group`, který určuje, zda datový proud data obsahuje³ (hodnota X), zda obsahuje pouze interní identifikátor a data jsou uložena v adresářové struktuře Fedory (hodnota M), nebo zda proud obsahuje URL odkaz na data (hodnoty R a E). Rovněž je možné datové proudy verzovat.

3.3 API

Aplikační rozhraní pro práci s repozitářem Fedora využívá webové služby a je poměrně kvalitní. Dělí se na API-A, API-M a REST API.

- **API-A** je rozhraní přes SOAP webové služby a „A“ ze zkratky znamená Access. Jedná se tedy o množinu metod, které nemodifikují digitální objekt. Metody jsou vypsány v následující tabulce.

<code>describeRepository</code>	<code>findObjects</code>	<code>resumeFindObjects</code>
<code>getObjectHistory</code>	<code>getObjectProfile</code>	<code>getDatostreamDissemination</code>
<code>listDatstreams</code>	<code>getDissemination</code>	<code>listMethods</code>

Tabulka 3.1: Metody API-A.

- **API-M** je také rozhraní přes SOAP webové služby, M ze zkratky znamená Modify. API-M obsahuje dohromady dvacet metod, a to modifikujících i nemodifikujících. Metody jsou vypsány v následující tabulce.

<code>addDatostream</code>	<code>compareDatostreamChecksum</code>	<code>getDatostream</code>
<code>getDatostreamHistory</code>	<code>getDatstreams</code>	<code>modifyDatostreamByRef</code>
<code>modifyDatostreamByValue</code>	<code>setDatastreamState</code>	<code>setDatastreamVersionable</code>
<code>purgeDatostream</code>	<code>addRelationship</code>	<code>getRelationships</code>
<code>purgeRelationship</code>	<code>modifyObject</code>	<code>purgeObject</code>
<code>export</code>	<code>getNextPID</code>	<code>getObjectXML</code>
<code>ingest</code>	<code>validate</code>	

Tabulka 3.2: Metody API-M.

- **REST API** Fedory zahrnuje, až na metody `describeRepository` a `getDatstreams`, všech 27 metod z API-A a API-M.

Ke všem uvedeným rozhraním existují soubory WSDL, respektive WADL pro REST, a je tak možné nechat si klientskou část služby vygenerovat automaticky z těchto souborů. Kromě těchto základních rozhraní existuje ještě rozhraní pro hledání ve vazbách RDF (*Resource Index Search*) a rozhraní pro hledání podle metadat v *Dublin Core*.

3. V případě binárních dat se používá kódování Base64.

Kapitola 4

Kramerius4

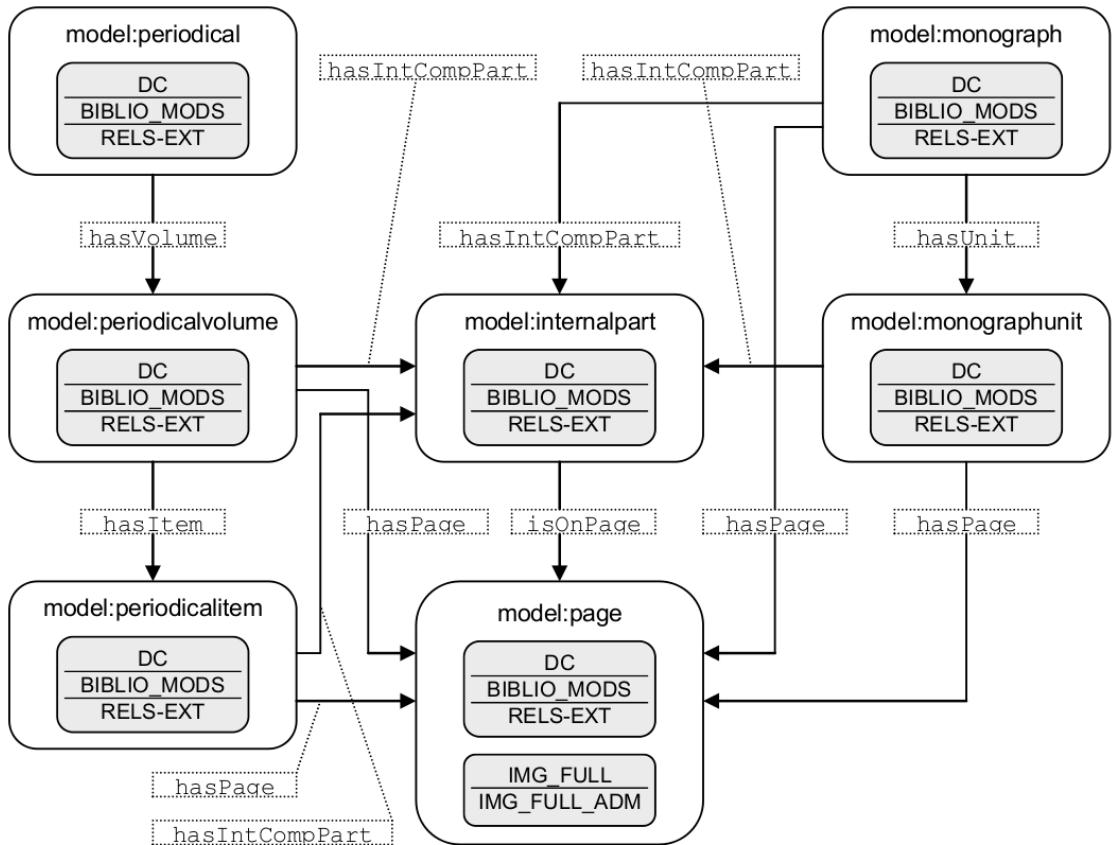
Kramerius verze 4 je nově vznikající knihovnický informační systém určený pro zpřístupnění digitalizovaných záznamů. Jeho vývoj je financován v rámci VISK 7 (Veřejné informační služby knihoven). Cílem programu VISK 7 je mimo jiné zveřejnění starších periodik a monografií veřejnosti. U starších a vzácných titulů hrozí riziko poškození v případě zapůjčení. Proto je mnohem výhodnější zpřístupnit tento obsah, jenž mnohdy spadá pod národní dědictví, elektronicky. Ostatně motivací pro vznik systému byly povodně roku 2002. Rovněž je potřeba digitalizovat a zpřístupnit tituly tištěné na kyselém papíru, jeli-kož životnost používaného papíru je řádově 100–200 let.

Systém Kramerius byl původně vyvíjen firmou Qbism¹. Projekt byl pozastaven a na začátku roku 2010 jej převzala firma Incad². V současnosti instituce³ stále používají nej-častěji verzi 3.3.0; verze 4 už je ale téměř dokončená⁴. Kramerius 3.x využívá k ukládání digitálních objektů adresářovou strukturu na disku a relační databázi Postgres. Pro ucho-vávání metadat slouží XML soubory, které jsou validní proti DTD pro periodika a mono-grafie. Periodika a monografie jsou prvky doménového modelu i v nové verzi. Kramerius ve verzi 4 je postaven na repozitáři Fedora. Veškerá data a metadata tak získává pomocí API, popsaného v předešlé kapitole, přímo z repozitáře. Kompletní doménový model za-znamenaný na úrovni objektů modelů v repozitáři Fedora je znázorněn na obrázku 4.1. Na obrázku jsou patrné i datové proudy digitálních objektů.

4.1 Projekt NDK

V současnosti probíhá velký projekt NDK (Národní digitální knihovna), jehož rozpočet je téměř 300 miliónů korun. Projekt je z 85 % hrazen ze strukturálních fondů EU prostřednic-tvím programu IOP (Integrační operační program). Cílem projektu je právě masivní digi-talizace pomocí robotických skenerů. Vybudovány budou dvě digitalizační centra v Praze a v Brně. Součástí projektu je také systém pro dlouhodobé uchovávání dat (*LTP system*) provozovaný na dvou lokalitách. Dále také systém pro zpřístupnění těchto digitalizovaných

-
1. Firma Qbism vytvořila Kramerius 3.x a začala vytvářet verzi 4, tu ale nedokončila.
 2. Převzali také hotový návrh a modul pro import dat z Krameria 3 do repozitáře Fedora.
 3. Mezi instituce využívající Kramerius 3 patří například: Knihovna AV ČR, Moravská zemská knihovna, Moravskoslezská vědecká knihovna v Ostravě, Vědecká knihovna v Olomouci, Státní technická knihovna, Jihočeská vědecká knihovna v Českých Budějovicích, Národní filmový archiv, Městská knihovna v Praze, Severočeská vědecká knihovna v Ústí n. L a další.
 4. Ukázka na <http://krameriusdemo.mzk.cz>.



Obrázek 4.1: Diagram relací mezi třídami objektů daných modelů. Se souhlasem převzato z [38].

dokumentů a také dokumentů sklízených z českého webu v rámci projektu WebArchiv. Do roku 2014 se plánuje zdigitalizovat a zpřístupnit 26 milionů stran a 100 milionů webových stránek prostřednictvím projektu WebArchiv (webarchiv.cz). V současnosti je systém Kramerius 4 považován za systém, který bude sloužit k zpřístupnění digitalizovaných dat, situace se ale může změnit v okamžiku, kdy bude znám systémový integrátor a jeho řešení. Veškeré informace o projektu jsou dostupné z [32].

Kapitola 5

Bohaté webové aplikace

S růstem penetrace Internetu ve všech zemích světa se původně statický Internet postupně změnil na aplikační platformu. Ačkoli desktopové (nativní) aplikace, ať už ve formě tlustých klientů, kancelářských balíků či her, jsou stále používané a populární, je, dle mého názoru, patrný trend přesunu směrem k aplikacím na webu. S moderními technologiemi se stírají rozdíly mezi desktopovou aplikací a aplikací v prohlížeči. Typický uživatel dnes tráví většinu svého času na počítači v internetovém prohlížeči a v podstatě jej používá jako virtuální operační systém. Podle předpovědi společnosti Gartner dojde v roce 2013 k okamžiku, kdy pro přístup k Internetu bude více lidí používat mobilní telefon než PC [21]. To s sebou nese obrovské možnosti pro webové aplikace citlivé na kontext, ať už geografický či jakýkoli jiný. Komunikace, sdílení a kooperace se staly pilíři Webu 2.0 a s ním souvisejících aplikací, jako jsou sociální sítě, Wikipedie, blogy a další. Web je zde pouze pár desetiletí a za tu dobu se dokázal transformovat do podoby média, v němž vznikají nejrůznější komunity. V těchto komunitách se poté uplatňují wikinomické principy a jiné emergentní vlastnosti komplexních sociálních systémů.

Web společnost dramaticky změnil a stále mění. Lidé přirozeně vždy navazovali kontakty a následně i sdíleli myšlenky s lidmi na základě jejich geografické lokace, tzn. se svými sousedy, spolupracovníky. Virtuální prostředí webu ale umožňuje lidem spojit se nad nějakým tématem nezávisle na lokaci. Bylo by zajímavé pozorovat, jak by například Gottfried Wilhelm Leibniz¹ komunikoval v dnešních podmínkách. Zda-li by, nadneseně řečeno, vybádal víc, nebo by vedl s I. Newtonem na různých vědeckých fórech ohnivé diskuse na téma nekonečně malých veličin. Těžko spekulovat. Nicméně faktem je, že vrcholy pomyslného grafu interakcí ve webovém prostředí podléhají mocninnému rozdělení (*Power law*) a můžeme v něm proto pozorovat, ale také využívat, fenomén *The Long Tail* či *Small world*. Tyto implikace z teorie grafů a statistiky umožňují analýzu především sociálních sítí, ale také nacházejí využití v oblasti SEO a monetizace bohatých webových aplikací.

Mohlo by se zdát, že oproti desktopovým aplikacím budou ty webové pomalejší, to ale plně závisí na tom, jak je aplikace navržena, co se vykonává na straně klienta, co už na straně serveru a také jak a kde je aplikace nasazena. V posledních letech se s rozvojem virtualizace objevují poměrně levné *cloudové hostingy*. Aplikace je v *cloudu* replikována napříč větším množstvím virtuálních strojů; tak je dosaženo vysoké dostupnosti aplikace

1. Udržoval rozsáhlou korespondenci s významnými filozofy tehdejší doby, která čítala na desetitisíce dopisů.

a uživatel platí jen za spotřebovaný strojový čas. Nevýhodou hostování aplikace v *cloudu* může být „*vendor lock-in*“ [5], protože perzistentní vrstva aplikace se typicky liší *cloud* od *cloudu*, a pomineme-li rozdíly mezi .NET a Java platformou, tak aplikace nasazená v *Google App Engine* nepoběží v *Microsoft Azure* nebo *Amazon EC2*. Naštěstí vznikají knihovny typu *jClouds*, které přidávají další abstraktní vrstvu a umožňují tak do jisté míry sjednotit vývoj pro různé typy PaaS dodavatelů. Google začal nabízet komerční variantu svého *App Engine*, která zvládá jazyk SQL. Koncept vertikálního dělení databáze do více instancí výborně škáluje, ale už není tak úplně jednoduché na tuto architekturu navázat stávající aplikace využívající hojně relačních databází, jazyk SQL či ORM rámce. Návrh systému by měl ale vždy odpovídat tomu, jestli systém bude nasazen v cloudu, či nikoli. Kvůli replikaci by aplikace měly být navrženy tak, aby jejich rozhraní bylo co nejvíce bezestavové. V případě migrace jedné instance na jinou při zotavení z chyby je totiž stav aplikace typicky ztracen.

Termín bohaté webové aplikace (*Rich Internet Applications*) byl poprvé použit v roce 2002 firmou *Macromedia*, ačkoli koncept byl už znám dříve. Jedná se o webovou aplikaci, která přináší bohaté uživatelské rozhraní s okamžitou odezvou z desktopových aplikací do prohlížeče, a podle společnosti Gartner se za tímto pojmem skrývají jen takové aplikace, kdy je do prohlížeče potřeba doinstalovat nějaké běhové prostředí nebo softwarový rámcem. Jmenovitě Flash, Silverlight, Java. S využitím AJAX nebo nastupující HTML5 se ale běžně jako RIA označuje i aplikace, kde není potřeba nic kromě samotného prohlížeče s integrovaným interpreterem JavaScriptu². Rychlosť interpreteru JavaScriptu v dnešních prohlížečích se mnohonásobně zvýšila oproti situaci před pár lety zejména díky konkurenčnímu boji mezi výrobci největších internetových prohlížečů, v posledních pár letech zejména Google Chrome.

5.1 AJAX

Zkratka AJAX (*Asynchronous JavaScript and XML*) podobně jako HTML5 není jedna osamocená technologie, ale spíše skupina souvisejících technologií. Samotný termín AJAX představil Jesse James Garrett v roce 2005. Ačkoli patent pro tento typ uživatelského rozhraní byl podán už roku 2003, používala se tato technologie už mnohem dříve, například v aplikaci *Outlook Web Access* roku 2000. Konsorcium W3C vydalo roku 2006 pracovní návrh (*working Draft*) rozhraní XMLHttpRequest [7]. V současnosti je už standard nově³ ve stavu kandidát na doporučení (*Candidate Recommendation*).

XMLHttpRequest, dále XHR, je rozhraní pro komunikaci prohlížeče, respektive skriptovacího jazyka v prohlížeči, s aplikací na serveru. Vzdálené volání může být jak blokující, a tudíž synchronní, tak neblokující, tedy asynchronní. Důležité je zmínit, že vzdálená volání pomocí XHR nevyžadují znovunačtení celé webové stránky, ale zpravidla pouze aktualizují některé elementy DOM (*Document Object Model*) stránky. Uživatelská zkušenost tak

2. Respektive EcmaScriptu a jeho nejčastějšího dialetu JavaScript. Budu ve své práci tyto termíny používat zaměnitelně.

3. Vesel v platnost 3. srpna 2010 (<http://www.w3.org/News/2010.html#entry-8871>).

připomíná okamžitou odezvu z desktopových aplikací. XHR je v současnosti podporován všemi hlavními prohlížeči a jeho vytvoření se provede příkazem

```
var xhr = new XMLHttpRequest();
```

V prohlížečích Microsoft Internet Explorer 6 a nižších se z historických a jiných důvodů vytváří objekt jinak, ale to by měl právě řešit vznikající standard W3C. Rozhraní poskytuje následující metody a atributy

- **open(method, url, async, user, password)** – Inicializuje spojení a předchází tak každému požadavku. Metoda má čtyři parametry, přičemž jen první dva jsou povinné. Parametr **method** udává HTTP metodu, protože XHR komunikuje protokolem HTTP. Nepovinný parametr **async** je typu boolean, a je-li roven false, bude volání blokující.
- **setRequestHeader(header, value)** – Nastaví hlavičku typu **header** na hodnotu **value**.
- **send(data)** – Odešle požadavek na serverovou část.
- **abort()** – Ukončí probíhající požadavek.
- **readyState** – Atribut určen jen pro čtení, určující stav probíhajícího požadavku. Možné hodnoty tohoto atributu zároveň popisují typický životní cyklus XHR požadavku. Hodnoty jsou UNSENT, OPENED, HEADERS_RECEIVED, LOADING a DONE. Jejich význam je intuitivní.
- **onReadyStateChange** – Atribut, kterému se nastavuje funkce, která se má vykonat když instance XHR změní svůj stav **readyState**. Jedná se tedy o typickou techniku zpětného volání pro asynchronní volání vzdálených metod.
- **status** – Atribut nesoucí HTTP kód, který informuje o úspěšnosti požadavku (200, 404 a jiné).
- **statusText** – Textová podoba předchozího atributu.
- **getResponseHeader(header)** – Vrací HTTP hlavičku typu **header**. Volá se až po stavu HEADERS_RECEIVED.
- **getAllResponseHeaders()** – Vrací všechny⁴ nastavené HTTP hlavičky. Volá se také až po stavu HEADERS_RECEIVED.
- **responseText** – Atribut obsahující odpověď serveru v textové podobě. Dá se použít i jako kontejner pro data ve formátu JSON a následně z nich pomocí ECMAScriptové funkce **eval()** vytvořit objekty.
- **responseXML** – Obsahuje odpověď serveru ve formátu XML, která může být dále zpracovávaná metodami DOM.

4. S výjimkou HTTP hlaviček jejichž typ je **Set-Cookie** a **Set-Cookie2**.

5.2 Reverzní AJAX

Web nebyl navržen pro webové aplikace, proto se hodně jednoduchých věcí dělá na webu složitě. Jedním takovým příkladem je technika nazývaná reverzní AJAX. Je to metoda, která dovoluje poslat data asynchronně ze serveru na klienta; důležité je zmínit, že volání iniciuje server bez vědomí klienta. Příkladem může být vyskakovací okno v případě nové zprávy v emailu. Pro tento případ užití existuje několik možností:

1. **Polling** je nejjednodušší technika, která neustále posílá požadavky na server a zjišťuje stav odpovědi. Zřejmě se zde zbytečně plýtvá strojovým časem klienta i serveru a zahlcuje se spojení.
2. **Piggyback** (v překladu „nošení na zádech“) znamená, že server jakmile dokončí zpracovávání požadavku, tak si odpověď připraví, ale nic nikam neposílá. Až v momentě, kdy klient pošle *jakýkoli* požadavek, je původní odpověď na klient zaslána společně s novou odpovědí. V případě aplikace, kde klient komunikuje relativně často nebo v aplikacích, kde reakční doba není příliš kritická, se jedná o techniku, jenž plýtvá nejméně zdroji. Nevýhodou může být dlouhé zpoždění.
3. **Streaming** využívá HTTP perzistentního spojení, tedy spojení, kdy se pravidelně zasílají *HTTP Keepalive* zprávy. Podle HTTP 1.1 jsou všechna spojení považována za perzistentní, pokud se neurčí jinak. Skutečnost je ale taková, že například *timeout* na serveru Apache 2.2 je pouhých 5 sekund. Rozdíl oproti *pollingu* je ten, že při *pollingu* se posílá mnoho HTTP požadavků a mnoho prázdných odpovědí, kdežto tady se pošle jen jeden, který se udržuje naživu, a až v případě, že server zpracuje požadavek, pošle odpověď. Nevýhodou tohoto přístupu je, že HTTP 1.1 povoluje maximálně dvě souběžná připojení prohlížeče s web serverem. V praxi se toto omezení obchází přes vytvoření nového *hostname* pro tentýž server a komunikuje se přes Streaming s tímto „jiným“ serverem.
4. **Dlouhý polling** – Klasické použití XHR probíhá tak, že prohlížeč vytvoří XHR objekt, nastaví *callback* funkci přes *onreadystatechange* a pokračuje ve vykonávání skriptu, v případě, že se jedná o asynchronní volání. Specifikace XHR nedefinuje chování v případě, že logika na straně serveru vykonává například nekonečný cyklus. Timeout musí být nastaven na straně klienta v JavaScriptu. Scénář klasického použití XHR lze převést na techniku *pollingu*, to znamená, že se bude zkoušet XHR volání s nastavenou hodnotou pro *timeout* na *x* sekund, kde *x* může být například náhodné číslo od tří do patnácti periodicky stále dokola, dokud server neuzná za vhodné poslat klientovi nějakou notifikaci, například o nové poště. Oproti klasickému *pollingu* je zde zajištěno, že i v intervalu mezi dvěma požadavky dá server vědět v případě, že je důvod.

Techniky *Streaming* a *dlouhý polling* bývají často souhrnně označovány jako technologie *Comet*, ale také *Ajax Push* či *Reverse Ajax*.

5.3 Nevýhody AJAXu

Za jednu z největších nevýhod bohatých webových aplikací v současnosti považuji fakt, že obsah těchto aplikací je obtížné indexovat, a proto se ani běžně neindexuje. V praxi to znamená, že je velmi složité docílit toho, aby bylo možné stránku nalézt prostřednictvím nějakého fulltextového vyhledávače. Mezi bohatou internetovou aplikací a webem (website) existuje jen úzká hranice. Například služba Twitter bude někde mezi těmito dvěma koncepcemi. Jeden z rozdílů může být právě potřeba indexace obsahu, která u aplikací není až tak kritická, protože nejsou obsahově zaměřené, ale spíše zaměřené na určité chování.

Další nevýhoda je spojena s provázaností obsahu obrazovky v daný časový okamžík na URL. V interaktivní dynamické aplikaci napsané prostřednictvím AJAXu se totiž nemění URL po odeslání a přijetí asynchronního volání. To přináší dojem okamžité odezvy, ale jednotlivé stavy aplikace se pak hůrce adresují, a tudíž i hůrce procházejí pomocí tlačítka „zpět“ v prohlížeči. Částečně lze toto chování vylepšit za pomocí „kotvy“ (anchor) v URL. Část URL za znakem „#“ je totiž kvůli specifikaci protokolu HTTP ignorována serverovou částí aplikace. V klientské části je ale tato část URL zjistitelná pomocí `document.location.hash`.

Další poměrně závažnou nevýhodou bohatých webových aplikací využívajících AJAX je otázka bezpečnosti. Přesun funkcionality na stranu klienta s sebou nese riziko útoku na aplikační logiku serveru. Kód na straně klienta je totiž veřejný všem, a i přestože existují nástroje pro obfuscaci kódu, existují také metody reverzního inženýrství, jak kód opět přečíst. Zná-li útočník kód na straně klienta, může se pokoušet například o následující útoky, které jsou částečně převzaty z [9, 10].

1. Předpokládejme webovou aplikaci, která po přihlášení uživatele kontaktuje server prostřednictvím XHR a zjistí, například v databázi, má-li uživatel administrátorská práva. V případě, že ano, vrátí se kladná odpověď na stranu klienta a v GUI aplikace se například zobrazí položka „administrace“. To je regulérní požadavek na aplikaci. Problém ale může nastat v případě, použije-li útočník například JavaScriptový debugger Firebug a zapne si breakpoint do callback metody XHR a nastaví odpověď na kladnou hodnotu i v případě, že je záporná. Je-li z JavaScriptového kódu patrné, co vlastně provádí, lze pomocí debuggeru pozměnit logiku za běhu aplikace. Útočník tak může snadno například zakomentovat veškeré validace na straně klienta. V případě, že chce útočník pozměnit logiku funkce, nad kterou nemá kontrolu, například v případě využívání nějaké JavaScriptové knihovny, lze funkci překrýt za pomocí *function hijacking*. Tyto techniky bych souhrnně nazval *pozměnění logiky na straně klienta*.

2. Aplikace často využívají líného natahování klientského kódu ze serveru, útočník tak kód debugovacím nástrojem nevidí, protože dynamicky přibývá. Řešením pro útočníka je vypsat si všechny dostupné funkce v prostředí a z názvů hádat logiku, nebo funkce překrýt; v případě JavaScriptu přereferencovat.

```
var ret = "";
for(var i in window) {
    if(typeof(window[i]) == "function") {
        ret += i + "\n";
    }
}
alert(ret);
```

3. Ze znalosti kódu na klientovi může útočník identifikovat slabá místa pro vedení útoku odepření služeb. V případě, že granularita aplikačního rozhraní serveru je příliš vysoká, což v případě aplikací využívajících AJAX bývá, je možné, že některý typ volání alokuje nějaký zdroj a až jiný typ volání jej dealokuje. Myšlenka v návrhu mohla být například taková, že ta dvě volání budou vždy následovat chvílkou po sobě, a když druhé volání nenastane, zdroj se stejně dealokuje po uplynutí nějakého času automaticky sám. Rozpozná-li ale útočník volání, které pouze zdroj alokuje, může opakovaně alokovat, dokud server nezahltí.

Možností je jistě více, chtěl jsem jen poukázat na fakt, že bezpečnost a transakce bohatých webových aplikací musí být zajištěna výhradně na straně serveru, protože kód na straně klienta může být podvržený. Také je dobré mít na paměti, že bezpečnostní rizika mohou plynout ze špatného návrhu systému a použití sebelepšího aplikačního rámce je potom k ničemu. Nicméně aplikační rámce určitě bezpečnostní rizika snižují.

Rozhraní objektu XHR není sice příliš složité, ale proto, že skriptovací jazyk na straně klienta, typicky ECMAScript, je slabě typovaný a jeho zpracování dědičnosti není tak intuitivní jako v jazyce Java, je velmi složité psát rozsáhlější aplikace pouze za pomocí tohoto jazyka. Náročná je také práce s DOM a chování polí také není někdy zřejmé, což lze demonstrovat na následujícím jednoduchém příkladu.

```
var odstavce = document.getElementsByTagName("p");
for (var i = 0; i < odstavce.length; i++) {
    odstavce[i].parentNode.removeChild(odstavce[i]);
}
```

Intuitivně by kód měl nejprve vybrat všechny HTML elementy p, tedy odstavce, a poté je smazat. Ve skutečnosti ale smaže pouze liché odstavce. Pole `odstavce` se po vykonání příkazu uvnitř cyklu aktualizuje opětovným zavoláním příkazu `getElementsByTagName`, tentokrát ale tato funkce vrátí o odstavec méně, protože jeden byl odstraněn. Hodnota čítače `i` se ale zvýší a prvek v poli na pozici 1, což je ale ve skutečnosti druhý odstavec stránky, už nebude nikdy smazán. Podobně pro další sudé odstavce na stránce. Na druhou stranu, podpora pro jazyk je v každém grafickém internetovém prohlížeči implicitně povolena, jazyk je interpretován velmi rychle a v poslední době se objevují implementace

ECMAScriptu na straně serveru jako například *Node.js*. Pro odstínění vývojáře od nízkoúrovňové práce s XHR v ECMAScriptu vzniklo mnoho softwarových rámců a knihoven.

5.4 Současné RIA rámce

Existuje celá řada rámců a sad nástrojů pro práci s technologií AJAX. V podstatě pro každý programovací jazyk, který se používá pro tvorbu webových aplikací, existuje hned několik variant. Na jedné straně jsou rámce a technologie, které vyžadují nainstalovat nějaké běhové prostředí či rozšíření do prohlížeče, a na straně druhé technologie využívající čisté HTML a XHR. Popíšu zde některé rámce z druhé kategorie, a to pro jazyky Java a JavaScript. GWT zde chybí, protože si zaslouží vlastní kapitolu.

- **DWR** (*Direct Web Remoting*) je knihovna pro jazyk Java, která umožňuje volat Javovský kód na straně serveru z JavaScriptu. Na straně serveru běží servlet, který zpracovává požadavky a posílá odpovědi zpět do prohlížeče. DWR vygeneruje třídy (*stubs*) v JavaScriptu na základě rozhraní aplikační logiky na serveru. Klient tak volá JavaScriptové funkce jakoby lokálně. Ve skutečnosti se ale provede vzdálené volání na servlet, ten provede zpracování (*unmarshalling*) dat z formátu JSON a deleguje volání na příslušnou třídu a metodu, ta poté vykoná svou činnost, a data jsou po převodu (*marshalling*) do formátu JSON poslána klientovi. Volání z klienta jsou asynchronní a používá se technika *callback* funkce. DWR umožňuje využívat také reverzní AJAX, a to metody *polling*, *Comet* a *Piggyback*. Tato knihovna je do jisté míry podobná GWT. Hlavní rozdíl je ten, že GWT umožňuje vytvářet prezentativní vrstvu, tzn. kromě samotného RPC mechanizmu obsahuje také propracovaný systém grafických komponent a jejich rozvrhování.

Dostupnost: open source s licencí *Apache Software License*, verze 2.

- **SmartClient** je velmi propracovaný komponentově orientovaný JavaScriptový rámec vytvořený firmou *Isomorphic Software*. SmartClient je určen pro integraci s JavaScriptem na straně klienta a Java na straně serveru, existuje ale i GWT wrapper s názvem SmartGWT, kterému bude věnována pozornost v sekci 7.3. Mimo komponent je zde výborná podpora pro definici datových zdrojů (*DataSources*) a jejich následné propojení s widgety (*Data binding*). AJAX je zde tedy podporován na vyšší úrovni abstrakce, protože *Data binding* automaticky sám natahuje popřípadě ukládá data asynchronně na pozadí.

Datové zdroje je mimo jiné možné definovat deklarativně v XML a možností je celá řada. Z definice datového zdroje umí komerční varianty tohoto rámce generovat databázové schéma, je zde také podpora pro *Hibernate*. Datový zdroj může být, kromě databáze, typu XML souboru, JSON, RESTful/SOAP webové služby, nebo lze napsat vlastní implementaci. Všechny widgety podporují také *drag-and-drop*. Největší nevýhodu spatřuji v tom, že podpora ani dokumentace pro základní bezplatnou variantu není na příliš vysoké úrovni. Více detailů v kapitole 7.

Dostupnost: základní varianta je open source s licencí *The Lesser GNU Public License, version 3*, rozšířené verze jsou komerční. Edice *Pro* stojí 745 dolarů na jednoho vývojáře, edice *Power* 1950 dolarů na vývojáře a pro edici *Enterprise* raději cenu uvedenou nemají. Nabízejí množstevní slevy.

- **Dojo** – Sada nástrojů Dojo obsahuje spoustu předpřipravených widgetů, tedy předprogramovaných grafických komponent, které lze snadno zakomponovat do webové aplikace. Ty se souhrnně nazývají *Dijit*. Programovací jazyk pro Dojo je JavaScript; některé widgety lze využít přes tzv. nativní rozhraní i v GWT. Widgety jsou vytvořeny kombinací HTML, CSS (vzhled) a JavaScriptového kódu (chování) a patří mezi ně například kalendář, bohatý textový editor, nabídky, grafy, vyskakovací okna a spousta dalších. Je zde také podpora pro AJAX v podobě XHR wrapperu. Dojo je navržen modulárně, proto při využití některých vlastností není potřeba zahrnovat do aplikace celou knihovnu, ale jen její použitou podmnožinu a jádro. Tento princip je ostatně společný většině těchto rámců.

Dostupnost: open source s licencemi *BSD License* a *the Academic Free License*.

- **jQuery** – Pro svou jednoduchost v současnosti nejrozšířenější [25] JavaScriptová knihovna. Využívá se zejména společně s jazykem PHP, ale není to pravidlem. Umožňuje například vybírat, měnit a vytvářet objekty DOM,⁵ animace, práci s událostmi a samozřejmě velmi snadnou a intuitivní podporu pro asynchronní volání serveru tedy AJAX. Zde je ukázka volání z JavaScriptu.

```
$.post("backend.php",
    { message: "Hello AJAX!", person: "Jiri Kremser" },
    function(data){
        alert("Server response: " + data);
    }
);
```

Dostupnost: open source s licencemi *GNU General Public License*, verze 2 a *MIT License*.

5. K vybírání slouží CSS selektory, nebo XPath.

Kapitola 6

HTML5

Rozdíly v možnostech desktopových a webových aplikací se s nástupem HTML5 stírají, proto v této kapitole popíší právě HTML5. Formálně je HTML5 specifikací konsorcia W3C, která by měla navazovat na své předchůdce HTML 4.01 a XHTML 1.1. V intuitivní rovině, a běžně se tak i používá, se jedná o skupinu technologií, které rozšiřují současné prostředky k vytváření webu o nové syntaktické konstrukty a nové kaskádové styly CSS3. HTML5 je dalším krokem v procesu přesunu desktopových aplikací na web; většina nových vlastností totiž umožňuje využívat lépe zdroje počítače, na němž běží prohlížeč. Například grafickou kartu, vícejádrové procesory, webkameru, zvuk, video, je zde podpora pro *drag-and-drop* a další. Fakt, že HTML5 je opravdu platformou pro tvorbu bohatých webových aplikací, potvrzuje i skutečnost, že specifikace se nejdříve jmenovala *Web Applications 1.0*. Původní specifikace byla navržena roku 2004 skupinou WHATWG (Web Hypertext Application Technology Working Group). Konsorcium W3C specifikaci přijalo za svou až roku 2010, to proto, že W3C prosazovalo spíše specifikaci XHTML 2.0, na které ale roku 2009 přestalo pracovat.

Ukazuje se tak, že akademiky navržené XHTML, které vychází z XML a staví na validnosti oproti XML Schema, bylo převálcováno účelnějším a jednodušším HTML5 a tlaky z průmyslové sféry. Očekává se, že specifikace HTML5 dosáhne stavu „kandidát na doporučení“ roku 2012. Všechny vyspělé prohlížeče a Microsoft Internet Explorer už ale začínají jednotlivé prvky HTML5 zpracovávat. Nástroje pro AJAX budou muset pravděpodobně začít v budoucnu podporovat tuto novou specifikaci, aby zůstaly konkurenceschopné.

6.1 CSS3

Nová verze kaskádových stylů je ve vývoji už celých deset let a poněvadž je poměrně rozsáhlá, tak je rozdělena do jednotlivých modulů řešících jen podčást problematiky. Jednotlivé moduly a jejich stav implementace lze nalézt na [13]. Zde uvedu jen ty nejzajímavější. Mezi užitečné drobnosti v CSS3 patří například kulaté rohy bez nutnosti použít obrázky, stín u textu, výplně s přechody barev, barvy s alpha kanálem. CSS3 obsahuje také vylepšené selektory, tzv. *Media Queries*. Ty slouží ke zjištění parametrů klienta, jako je například rozlišení, a na jejich základě nabízí vhodný kaskádový styl. Další zajímavou vlastností nových kaskádových stylů je možnost animovat element na stránce. K dispozici jsou afinní transformace nebo pouhá aplikace stylu, vše ale s nastavitelným chováním v čase.

6.2. STRUKTURA DOKUMENTU

Následující příklad vytvoří styl `magic-box`, který animuje element tak, že do nekonečna v třísekundových intervalech zvětšuje písmo z 10 na 60 pixelů a mění barvu pozadí postupně z červené na zelenou a poté na žlutou.

```
@-webkit-keyframes name_of_animation {
    0% {
        background: red;
        font-size: 10px;
    }
    50% {
        background: green;
    }
    100% {
        background: yellow;
        font-size: 60px;
    }
}

.magic-box {
    -webkit-animation-name: name_of_animation;
    -webkit-animation-timing-function: ease-in-out;
    -webkit-animation-duration: 3s;
    -webkit-animation-iteration-count: infinite;
}
```

V příkladu lze zaznamenat vysoký výskyt termínu „`webkit`“. Jedná se o *vendor prefix*; ten znamená varování pro vývojáře, že implementace ještě není konečná a může se změnit. „`webkit`“ je *vendor prefix* prohlížečů postavených na jádře `Webkit`. Používání CSS3 animací by mělo usnadnit vývoj a nahradit JavaScript a Flash, jenž se k témtu účelům používají v HTML4.

6.2 Struktura dokumentu

Nové značky na označení částí dokumentů přináší do dokumentu více sémantiky. V HTML4 se kvůli rozložení elementů na stránce a propojenosti na kaskádové styly využívá často nic neříkající element `div`. HTML5 by měla obohatit syntaxi jazyka o značky jako

- `header` – Hlavička stránky.
- `nav` – Navigační část stránky, nejčastěji menu.
- `aside` – Boční panel stránky, který se neváže k obsahu hlavní stránky. Typicky reklama.
- `article` – Ucelená textová část, například článek nebo komentář.
- `section` – Určuje, že obsah stránky k sobě logicky patří. Snad se nestane v budoucnu všudepřítomným elementem jako je dnes element `div`.

- **footer** – Patička stránky, může obsahovat informace o autorovi, autorských právech nebo odkazy na související dokumenty.
- **figure** – Například video nebo obrázek s popisem a názvem.

Takto rozdělená stránka může být lépe vyhodnocována roboty. Roboti dnešních vyhledávačů dnes penalizují weby s duplicitním obsahem, k tomu ale často dochází i tak, že se na jednotlivých stránkách webu opakuje navigační menu, hlavička, patička. Explicitním označením těchto částí lze tomuto neoprávněnému penalizování zabránit.

6.3 Microdata

Microdata [31] společně s RDFa (*RDF in Attributes*) a mikroformáty jsou standardy pro popis sémantiky v HTML stránkách. Jednoduchou metodou tak lze částem stránky v HTML kódu přiřazovat jasný význam díky předem definovaným slovníkům. Zatímco mikroformáty pro vkládání sémantické informace zneužívají atribut **class**, pro *Microdata* jsou v HTML5 vyhrazeny speciální atributy **itemscope**, **itemtype**, **itemprop** a **itemref** a element **meta** s atributem **content**.

Velkou výhodu mikroformátů a *Microdata* spatřuji v jednoduchosti jejich používání a praktických výhodách z toho plynoucích. Prohlížeče zatím sice ještě nemají zabudovanou podporu pro jejich interpretaci, ale existují rozšíření, která například dokáží informaci o místě otevřít na mapě nebo v případě popsané osoby ji přidat do kontaktů. Google například interpretuje *Microdata*, mikroformáty a RDFa ve svých snippetech¹. Právě takovými praktickými metodami lze webové návrháře přimět k budování sémantického webu. Problém s klasickými metodami, které využívají RDF, je jejich příliš složitá tvorba. Tento problém *Microdata* odstraňuje, navíc existují algoritmy k převádění na RDF, takže pak lze využít nástroje pro práci s RDF.

```
<div itemprop="rating" itemscope itemtype="data-vocabulary.org/Rating">
    Rating: <span itemprop="value">8.5</span>
    <meta itemprop="best" content="10" />
</div>
```

Příklad 6.3.1: Příklad zápisu sémantické informace „hodnocení 8,5 z 10“.

Příklad ukazuje použití *Microdata* v HTML5. Element **meta** nebude v prohlížeči zobrazen. Slovníků pro *Microdata* je celá řada a lze využívat i známé mikroformáty jako: *hCard*, *hCalendar*, *hAtom*, *hReview*. V současnosti dvě největší skupiny slovníků pro *Microdata* tvoří slovníky od Googlu (prefix www.data-vocabulary.org) a slovníky od skupiny WHATWG (prefix microformats.org). Výhoda *Microdata* oproti mikroformátům spočívá také v Javascriptovém API pro práci s popsanými objekty uvnitř webové stránky.

1. Útržek stránky, který vyhledávač zobrazí na stránce s výsledky hledání.

6.4 SVG a podpora elementu Canvas

SVG (Scalable Vector Graphics) je grafický vektorový formát, který lze v HTML5 vkládat přímo do DOM. V HTML4, podobně jako v případě videa, se tyto obrázky vkládají pomocí elementů, jež slouží jako kontejner. Používají se elementy *embed*, *object* a *iframe*. Nevýhodou tohoto přístupu je skutečnost, že nelze přistupovat k jednotlivým částem SVG stromu obrázku prostřednictvím DOM a následně jej dynamicky měnit. To v HTML5 lze.

Element *Canvas* představuje, jak už název napovídá, plátno, do nějž lze ECMAScriptem dynamicky kreslit přes rozhraní tzv. kontextu. Kromě rozhraní pro 2D grafiku už některé prohlížeče podporují i rozhraní pro práci s 3D grafikou. Příkladem může být WebGL, což je implementace *OpenGL ES*. Výrobci prohlížečů slibují do budoucna nativní podporu grafických karet. Je tedy otázkou jestli herní průmysl v budoucnu přesedlá na aplikační platformu Internetu. Pro demonstrační účely HTML5 byla například naportována hra *Quake II Open Source*, která využívá právě WebGL, WebSockets, HTML5 audio a další HTML5 vlastnosti. Element *Canvas* poskytuje vývojářům bohaté aplikační rozhraní, které je ale relativně nízkoúrovňové.

6.5 Audio a video

Když se navrhovaly předchozí verze značkovacího jazyka HTML, nikdo netušil, jak bude Internet vypadat v roce 2011. Podpora pro přehrávání videa tak nebyla integrována do jazyka. HTML4 řeší podporu videa podobně pragmaticky jako podporu SVG, tedy pomocí obalovacích elementů *object* a přehrávače nejčastěji *Adobe Flash Player*. HTML5 podporuje video přímo zahrnutím značky *video* do syntaxe jazyka. Vkládat videa je tak stejně snadné jako vkládat obrázky.

```
<video src="url" controls autobuffer>
    Váš prohlížeč nepodporuje HTML5 element video.
</video>
```

Element obsahuje dva nepovinné atributy *controls* a *autobuffer*. První atribut určuje, zda má prohlížeč k videu přidat ovládací tlačítka. Vzhled těchto tlačítek je ponechán na prohlížeči a lze jej upravit kaskádovými stylami. Druhý parametr zapne automatické načítání videa před jeho spuštěním. Vnitřek elementu *video* obsahuje tzv. *fallback*, jinými slovy obsah, jenž je zobrazen v případě, že prohlížeč element ještě nepodporuje. Chce-li mít vývojář jistotu, je doporučováno video nejprve převést do formátů² OGG (Theora, Vorbis), MPEG 4 (H.264, AAC), WebM (VP8), a poté nabídnout prohlížeči všechny alternativy. Ten vybere podle toho, jaký kodek podporuje, a video spustí, popřípadě nabídne *Flash Player*, v případě že nepodporuje žádný formát nebo HTML5 značku *video*.

Element *audio* je analogií k elementu *video*. Používání je velmi podobné, také je doporučeno předzpracovat audiozáznam nejprve do více formátů. Oba elementy zpřístupňují rozhraní v ECMAScriptu, jímž lze záznamy spouštět, pozastavovat, ovládat hlasitost apod.

2. V závorkách jsou uvedeny používané kodeky pro video a audio proud.

6.6 App Cache, Web Storage a databáze

Všechny tři metody z titulku se snaží řešit ukládání dat na straně klienta, tedy v prohlížeči. *App Cache* je metoda, která řeší zpřístupnění webových aplikací i v režimu offline, podobně jako *Google Gears*. Vývoj *Google Gears* byl zastaven, protože podpora pro offline zpřístupnění byla začleněna přímo do HTML5 právě v podobě *App Cache*. Metoda je jednoduchá, požaduje vytvořit speciální soubor, označovaný jako manifest, s cestami k souborům, které se mají kešovat. Ty potom v režimu offline budou k dispozici v lokální paměti prohlížeče. Aplikace ale často vyžaduje ke své práci generovat data nová. K tomuto účelu nabízí HTML5 dvě metody – *Web Storage* a zabudovanou databázi.

Web Storage je relativně jednoduchá databáze založená na principu klíč-hodnota. ECMAScriptový objekt nejvyšší úrovně, tedy objekt `window`, obsahuje dva nové atributy: `window.localStorage` a `window.sessionStorage`. První typ uchovává záznamy do doby, než jsou z databáze skriptem odstraněny, zatímco druhý typ persistence uchovává jen po dobu sezení. *Local Storage* navíc umožňuje sdílet přístup k datům napříč více otevřenými prohlížeči. Rozdíl oproti *cookies* je ten, že hodnoty se neposílají v HTTP hlavičkách na server s každým požadavkem jako v případě *cookies*.

V případě, že nestačí *Web Storage*, je v HTML5 k dispozici komplexnější řešení v podobě integrované databáze. V současnosti existují dvě různá řešení integrované databáze v prohlížeči. *Web Database* je podporována prohlížeči Chrome, Safari a Opera; a *IndexedDB* je podporována prohlížeči Firefox a Internet Explorer. *Web Database* je vlastně rozhraní k databázi *SQLite*, takže podporuje jazyk SQL a transakce; zatímco *IndexedDB* je databáze objektová postavená na principu B-stromů a záznamy jsou zpřístupňovány v ECMAScriptu přes podobné rozhraní jako mají kolekce v Javě. Výhodou tohoto přístupu může být fakt, že tak nevzniká „špagetový kód“, kde se mixuje ECMAScript a SQL jako v případě *Web Database*.

6.7 Geolokace

API pro geolokaci poskytuje přístup k uživatelské poloze přes ECMAScript. Ve specifikaci není uvedeno, jak se tato poloha získává, to je čistě v režii zařízení, kde je web provozován. V praxi se tak může poloha zjistit z GPS zařízení v chytrém mobilním telefonu, IP adresy nebo ze spojení s BTS (*base transceiver station*) v GSM (*Global System for Mobile Communications*) sítích. Určení polohy musí nejprve uživatel povolit v prohlížeči.

```
function showMap(position) {
    // např. ukaž mapu zacentrovanou na
    // (position.coords.latitude, position.coords.longitude).
}

// Požadavek na zjištění polohy
navigator.geolocation.getCurrentPosition(showMap);
```

6.8 Web Workers

JavaScript je z podstaty jednovláknový skriptovací jazyk, proto v případě složitějších výpočtů přestává uživatelské rozhraní, které JavaScript využívá, reagovat na podněty uživatele. Tomu se snaží zabránit *Web Workers* tak, že je možné pustit JavaScriptový kód v separátním vlákně. Tento kód musí být v samostatném souboru a zpravidla provádí nějakou náročnou operaci. V kódu pro *Web Worker* se nesmí vyskytovat operace pro práci s DOM, může se ale komunikovat se serverem prostřednictvím XHR, přistupovat k objektu `navigator` a následně zjistit polohu zařízení nebo vrátit výsledek svého počítání metodou `postMessage()`

Kapitola 7

Google Web Toolkit

GWT je komponentově zaměřený open source aplikační rámc od společnosti Google. V sekci 5.3 jsem zmínil některé nevýhody jazyka JavaScript plynoucí z jeho navržení. Jedná se například o slabou typovou kontrolu a špatný kód. Platforma Java EE nabízí celou řadu nástrojů pro tvorbu rozsáhlých projektů, a toho GWT využívá. S trochou nadsázky lze říct, že s GWT lze napsat webovou aplikaci postavenou na technologii AJAX a nenapsat ani jeden řádek zdrojového kódu v JavaScriptu. Tento rámc totiž funguje tak, že veškerá aplikační logika pro *front-end* se programuje v jazyku Java a GWT komplátor vytvoří optimalizovaný JavaScriptový kód z kódu napsaného v Javě. Vývojář tak může používat IDE s doplňováním kódu, zachytávání chyb při komplikaci, automatické testy, nástroje pro sestavování aplikace, nasazování aplikace do servlet kontejnerů a další vlastnosti platformy Java. Zdrojový kód psaný v Javě je, dle mého názoru, mnohem lépe čitelný a udržovatelný nežli zdrojový kód napsaný v JavaScriptu. V případě GWT, kde si uživatel může vytvářet nové komponenty hrají čitelnost a z ní vyplývající udržovatelnost kódu zásadní roli. Znovupoužitelnost komponent a modulární systém tohoto rámce směrují vývojáře k vývoji volně svázaných komponent. GWT komplátor podporuje podmnožinu jazyka Java, jmenovitě balíky:

- `java.lang`
- `java.lang.annotation`
- `java.util`
- `java.io`
- `java.sql`

Protože jazyk Java byl uvolněn jako open source, je možné přidat některé třídy například z balíku `javax` ke zdrojovým kódům aplikace tak, že v konfiguračním souboru modulu se nastaví elementem `super-source` cesta k nim. Pak je lze využívat na straně klienta také. Doporučuje se to ale jen v krajních případech, protože vyjmout síť na sobě závislých tříd z JRE není snadné. Omezení na výše uvedené balíky platí jen v případě kódu použitého na straně klienta, tedy toho, který se bude konvertovat do JavaScriptu.

7.1 Historie a budoucnost

GWT bylo uvolněno roku 2006 pod licencí *Apache Software License* ve verzi 2. Až do verze 1.5 byla na straně klienta podporována pouze Java 1.4. S verzí 1.5, uvolněnou roku 2008, přibyly podpora pro anotace, generika, *for-each* cyklus a další. Od verze 2.0 lze tzv. hostovaný mód spouštět přímo v internetovém prohlížeči s nainstalovaným zásuvným modulem. Hostovaný mód slouží ve fázi vývoje k ladění aplikace; klientská část aplikace není přeložena do JavaScriptu, ale bytekód Javy je interpretován pomocí JVM a je tak umožněno debugování. Do verze 2.0 byl k tomuto účelu používán speciální prohlížeč založený na jádřech Internet Explorer nebo Mozilla/Gecko. Současná verze 2.1.1 byla uvolněna 17. 12. 2010 a mezi hlavní novinky patří podpora pro návrhový vzor *Model View Presenter*.

Společnou úlohou aplikačních rámců je zrychlit a zlevnit vývoj počítačových systémů. Google a společnost VMware, respektive její divize SpringSource, společnými silami spojili rámce Spring Roo a GWT. Spring Roo je nástroj pro rapidní vývoj aplikací (RAD) s využitím generátorů kódu a aspektově orientovaného přístupu s rámcem AspectJ. Od GWT verze 2.1 tak lze s využitím Spring Roo vygenerovat poměrně velkou část aplikace. Z konference Google Developer Day 2010 jsem měl dojem, že tímto směrem chce Google s GWT opravdu dál pokračovat. Zda je využití Spring Roo+GWT užitečné i v pozdějších fázích vývoje, či se jedná pouze o usnadnění začátků vývoje vygenerováním kostry projektu, ukáže až čas. Už nyní ale můžu potvrdit, že vygenerovat projekt, zásuvným modulem Spring Roo pro IDE Eclipse a následně nasadit aplikaci do cloudu *App Engine*, také pomocí modulu pro Eclipse, je opravdu velmi rychlé a snadné.

Je velmi pravděpodobné, že až se situace kolem standardu HTML5 ustálí, hodlá Google integrovat HTML5 do GWT. S HTML5 a JavaScriptem sice lze vytvářet bohaté webové aplikace mnohem jednodušejí než nyní s HTML4, pořád to ale bude ten stejný JavaScript se všemi jeho nevýhodami, a proto je dle mého názoru vyšší úroveň abstrakce a silně typovaný jazyk krok správným směrem.

7.2 Členění GWT aplikace

Aplikace psané s GWT používají členění projektu do hierarchie adresářů, a to takové, že v adresáři **client** se nachází prezentační vrstva aplikace, která se překládá do JavaScriptu a v adresáři **server** pak logika, která se používá v serverové části aplikace. V serverové části aplikace se tak nachází například servlety, které jsou z klienta asynchronně volány, ale v podstatě se může jednat o libovolnou *business* logiku v libovolném aplikačním rámci. Doporučuje se ještě využívat adresář **shared** a umístit tam třídy, které budou sdíleny jak serverem tak klientem. Může se jednat o objekty DTO (*Data transfer object*) nebo třídy s konstantami. V konfiguračním souboru modulu je pak nutno označit adresář **shared** jako adresář s klientskou částí aplikace, tedy že se má překládat do JavaScriptu. Platí, že serverové třídy mohou přistupovat k třídám klientským, ale ne naopak. Zavedením adresáře pro sdílené třídy tak explicitně uvedeme, na které klientské třídy se odkazuje ze serverových tříd, a závislosti tak lze lépe sledovat a odhalovat například chyby v návrhu už

v raných fázích vývoje. Popsaným adresářům může samozřejmě hierarchicky předcházet libovolná balíková struktura, ale také mohou dále obsahovat libovolné balíky.

Základní kompilační a konfigurační jednotkou je v GWT modul. Ten musí mít konfigurační soubor s příponou `.gwt.xml`. Taktéž by se podle konvence mělo shodovat jméno konfiguračního souboru se jménem třídy, která tvoří vstupní bod modulu¹. Modul může například obsahovat jména ostatních modulů, na kterých je závislý. Některé základní moduly jsou uvedeny v tabulce [7.1]. Jejich zdrojové kódy určené pro klientskou část budou také komplikovány do JavaScriptu. Kompilátor obsahuje mechanizmus, jenž zjistí, který kód je dosažitelný ze vstupního bodu aplikace, a v případě nedosažitelného kódu tento nekomplikuje. To je výhodné jednak proto, že aplikace tak nezabírá tolik místa a je rychleji stažena při používání do prohlížeče, a také proto, že stačí zahrnout celý modul pro práci, řekněme s XML, a začít používat jen podčást funkcionality. Později, když vývojář začne používat další část funkcionality pro práci s XML soubory, není potřeba nic měnit. Jednoduchý GWT modul je znázorněn na příkladu [7.2].

Modul	Logické jméno	Definice modulu	Obsah
User	<code>com.google.gwt.user.User</code>	<code>User.gwt.xml</code>	Jádro GWT
HTTP	<code>com.google.gwt.http.HTTP</code>	<code>HTTP.gwt.xml</code>	HTTP komunikace
JSON	<code>com.google.gwt.json.JSON</code>	<code>JSON.gwt.xml</code>	Tvorba a parsování JSON
JUnit	<code>com.google.gwt.junit.JUnit</code>	<code>JUnit.gwt.xml</code>	Integrace rámce JUnit
XML	<code>com.google.gwt.xml.XML</code>	<code>XML.gwt.xml</code>	Tvorba a parsování XML

Tabulka 7.1: Standardní GWT moduly, které je možno zahrnout do aplikace [23].

7.3 GWT-RPC

Pro asynchronní volání aplikační logiky serveru se v tomto rámci používá nejčastěji metoda GWT-RPC. Ta využívá Java serializaci při posílání objektů, a na straně serveru tak musí nutně existovat část aplikace napsaná v Javě, která objekty deserializuje, provede patřičné volání a odpověď opět serializuje a pošle zpátky klientovi. GWT ale umožňuje, kromě metody GWT-RPC, asynchronní volání z klienta i prostřednictvím dat serializovaných do JSON nebo XML. V takovém případě sice neexistuje tak těsná vazba mezi klientskou a serverovou částí aplikace², ale na druhou stranu serializace a deserializace XML je poměrně náročná na čas v porovnání se serializací a deserializací objektů v Javě. Bezprostřední odezva bohatých webových aplikací je přitom jednou z priorit. Je tedy na návrháři, aby rozhodl, zda upřednostní čistý návrh, nebo rychlosť odezvy aplikace.

1. Tato třída musí implementovat rozhraní `com.google.gwt.core.client.EntryPoint`.

2. Při použití platformově nezávislého XML nebo JSON může být serverová část aplikace napsaná v jiném programovacím jazyku.

```

<module rename-to='meditor'>
  <!-- Pure GWT -->
  <inherits name='com.google.gwt.user.User' />
  <inherits name="com.google.gwt.inject.Inject" />
  <inherits name="com.google.gwt.resources.Resources" />

  <!-- Smart GWT -->
  <inherits name='com.smartgwt.SmartGwt' />
  <inherits name='com.smartgwt.SmartGwtNoScript' />
  <inherits name='com.smartgwt.tools.SmartGwtTools' />

  <!-- GWT Platform -->
  <inherits name='com.gwtplatform.dispatch.Dispatch' />
  <inherits name='com.gwtplatform.mvp.Mvp' />

  <!-- The app entry point class. -->
  <entry-point class='cz.fi.muni.xkremser.editor.client.MEditor' />

  <!-- Paths for translatable code. -->
  <source path='client' />
  <source path='shared' />

  <!-- Internationalization support. -->
  <extend-property name="locale" values="en"/>
  <extend-property name="locale" values="cs"/>
  <set-property-fallback name="locale" value="en"/>
</module>

```

Příklad 7.2.1: Příklad definice GWT modulu.

Pro vytvoření asynchronního volání pomocí GWT-RPC je potřeba nejprve vytvořit třídy a rozhraní, které danou logiku vykonávají. Konkrétně je nutné:

1. Vytvořit rozhraní (serverové), které rozšiřuje rozhraní `RemoteService` a obsahuje množinu metod, které budou asynchronně volány.
2. Vytvořit implementaci tohoto rozhraní. Implementační třída musí navíc dědit z třídy `RemoteServiceServlet`.
3. Vytvořit asynchronní rozhraní, přes které se budou volání z klienta vykonávat. Asynchronní rozhraní musí obsahovat stejné metody jako rozhraní serverové, až na to, že všechny musí vracet `void` a signatury jednotlivých metod musí být obohaceny o atribut `AsyncCallback<T> callback`. Typ `T` je typ návratové hodnoty dané metody, který implementuje rozhraní `Serializable`³, nebo `Serializable`⁴.

3. Z balíku `com.google.gwt.user.client.rpc`

4. Z balíku `java.io`. Tato možnost existuje od verze 1.4.10.

4. Nastavit implementační třídu jako Java servlet v popisovači nasazení (*Deployment Descriptor*).
5. Zavolat zvolenou metodu.

Při volání je třeba předat metodě jako parametr *callback*, aby se v případě úspěšného návratu z funkce mohla v novém vlákně provést metoda `onFailure`, nebo `onSuccess`. První uvedená se provede v případě vyhození výjimky. Volání může vypadat následovně.

```
MyServiceAsync myService = (MyServiceAsync) GWT.create(MyService.class);
myService.sendEmail(message, new AsyncCallback<String>() {

    public void onFailure(Throwable caught) {
        Window.alert("RPC to sendEmail() failed.");
    }

    public void onSuccess(String result) {
        label.setText(result);
    }
});
```

7.4 GWT-Platform

Pro GWT vzniká celá řada nových vylepšení, komponent, ale i plnohodnotných aplikačních rámců. Jedním takovým rámcem je GWT-Platform [24], často zkracován na GWTP. GWTP využívá celou řadu návrhových vzorů a dobrých programátorských praktik. Vznikl odštěpením z projektů *gwt-dispatch* a *gwt-presenter*, jejichž vlastnosti také obsáhl. Leitmotivem tohoto rámce jsou návrhové vzory popsané v následující kapitole, tedy: *Model View Presenter, Event Bus, Dependency Injection* a *Command pattern*.

Zajímavý modul GWTP je *Crawler*, jehož úkolem je udělat aplikaci indexovatelnou vyhledávači. Princip je popsán v [8] a ve stručnosti funguje následovně. Google se zavázal, že když jeho *Googlebot* narazí na URL tvaru

`www.example.com/ajax.html#!key=value`

nebude indexovat HTML kód získaný požadavkem GET na toto URL, ale na URL tvaru

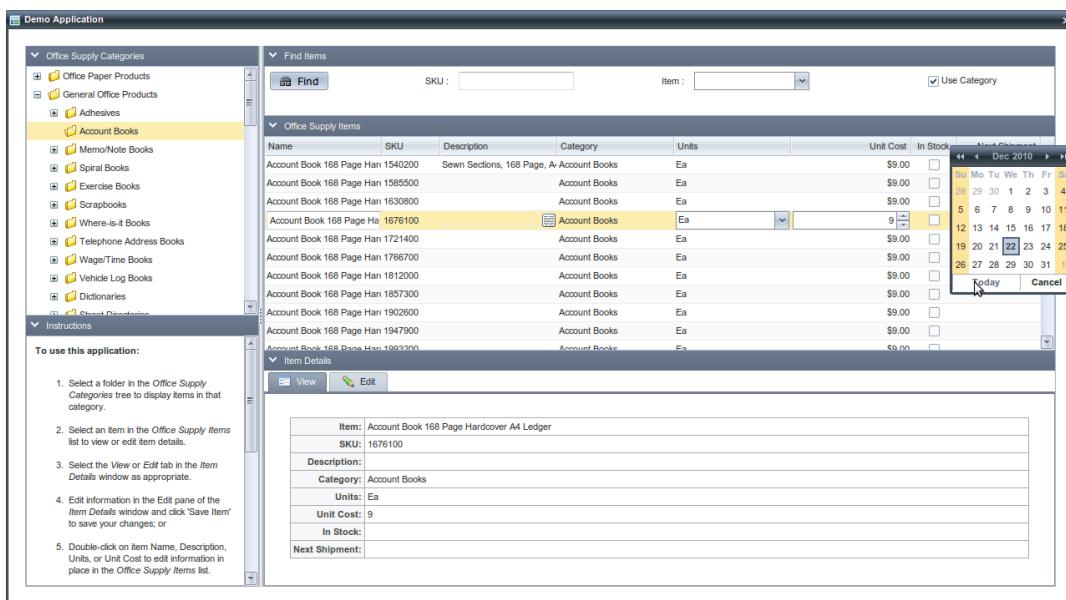
`www.example.com/ajax.html?_escaped_fragment_=key=value.`

Hodnota atributu `_escaped_fragment_` je tedy část za znakem „#“ z původního URL bez znaku „!“. Webová aplikace podle přítomnosti tohoto atributu rozpozná, že se jedná požadavek pro *crawler*, a je v její kompetenci vrátit takový obsah, jaký chce, aby byl indexován. K témtu účelům existuje knihovna *HtmlUnit*, která mimo jiné zvládá vytvářet HTML obraz. HTML obraz je serializace DOM po načtení stránky a interpretaci skriptů.

7.5 Smart GWT

Tento rámcový je nadstavbou (wrapper) nad aplikačním rámcem Smartclient, je vyvíjen stejnou firmou a má také stejné licenční podmínky. Základní varianta je zdarma a nadstandardní varianty jsou zpoplatněné. V GWT je možnost⁵ definovat metodu jako `native`, a poté v ní napsat do komentáře kód v JavaScriptu. Tato metoda je při komplikaci rovnou vložena do výsledného JavaScriptového kódu. Tímto způsobem je napsán aplikační komponentově orientovaný rámec Smart GWT.

Komponenty Smart GWT jsou velmi propracované⁶ a celý rámec se jeví jako jednoduchý. Proto jsem si jej zvolil pro prezentační vrstvu aplikace metadatového editoru. Bohužel dokumentace není na příliš dobré úrovni a API komponent není konzistentní. Například metoda pro vrstvení widgetů na sebe se liší widget od widgetu. Většinou se nazývá `addMember()`, ale někdy také `setTab()` či `setPane()`. Nevýhodou je také velikost aplikace, která se musí stáhnout do prohlížeče. V komerční verzi ale existuje nástroj pro před-natáhnutí (*prefetching*) aplikace, například při přihlašování. Přes tyto a jiné nevýhody je uživatelská zkušenosť se Smart GWT na dobré úrovni. Na obrázku 7.1 je zobrazena ukázková aplikace.



Obrázek 7.1: Jeden ze skinů ve Smart GWT.

5. Tato možnost se nazývá jako JSNI (*JavaScript Native Interface*) a je do jisté míry podobná Javaovskému JNI (*Java Native Interface*).
6. <http://www.smartclient.com/smartgwt/showcase>.

Kapitola 8

Návrhové vzory vhodné nejen pro GWT

V této kapitole popíší některé návrhové vzory, které se vyplatí užívat při vývoji s GWT. Vzory MVP, *EvenBus* a *Command* jsou součástí knihovny GWTP popsané v předešlé kapitole. Vzor *Dependency Injection* je v GWT dostupný prostřednictvím knihovny GIN (klientská část aplikace) a na straně serveru buď přes aplikační rámec *Spring*, nebo *Google Guice*.

Návrhové vzory je výhodné začleňovat do aplikace hned z několika důvodů. Jedním z nich je identifikace a pojmenování určitého programátorského konceptu, což potom usnadňuje komunikaci problému v týmu. Dalším důvodem je fakt, že návrhové vzory jsou časem prověřené techniky, jak strukturovat aplikaci¹, a při správném použití tak nabízí určitou garanci kvality a méně příležitostí navrhnut aplikaci nevhodně.

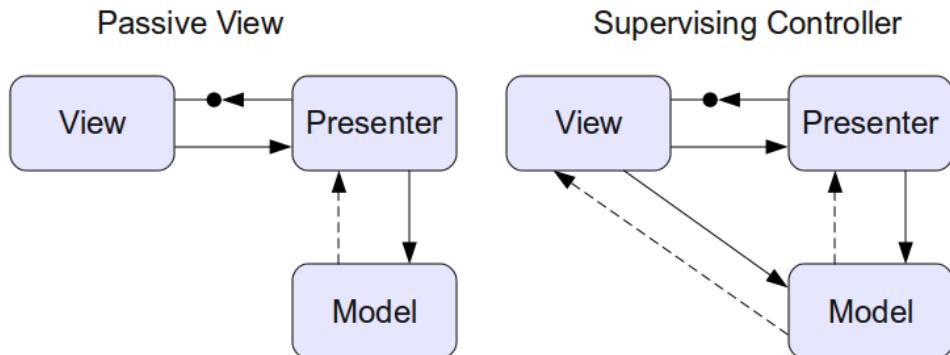
8.1 MVP

Model View Presenter je architektonický vzor [39] podobný známějšímu *Model View Controller* (MVC). Dle mého názoru je nevýhodou vzoru MVC, ale i MVP, že jsou definovány poměrně vágně a obecně. Základní myšlenkou je rozdělit aplikaci na části *Model*, *View* a *Controller* tak, aby změna jedné části měla co nejmenší dopad na změnu ostatních částí. Jinými slovy minimalizovat závislosti mezi částmi aplikace a umožnit tak jejich souběžný vývoj, jejich nahraditelnost a lepší udržovatelnost. *Model* drží data a *business logiku* aplikace, *View* zobrazuje uživatelské rozhraní a *Controller* má na starosti tok událostí v aplikaci a obecně aplikační logiku.

V praxi se ale pochopení tohoto vzoru často liší a v případě, že v daném programovacím jazyku není MVC inherentně podporováno, jako například v *Ruby on Rails*, je velmi obtížné implementovat tento vzor správně, a naopak je velmi snadné jej porušovat. Odbytý návrh častokrát inklinuje k řešení, v němž jsou závislosti jednotlivých komponent prostoupeny celou aplikací. Dalším častým antivzorem používaným v MVC je stav, kdy v aplikaci existuje jedna třída, která je dostupná například přes vzor *Service Locator* nebo *Dependency Injection* a která vykonává většinu logiky (antivzor *God Object*). Při posuzování oddělenosti komponent je také důležité mít na paměti, že závislosti za běhu aplikace (*runtime dependencies*) nejsou tak svazující jako závislosti v čase komplikace (*build time dependencies*).

1. Platí zejména pro strukturální a architektonické návrhové vzory.

Vzor MVP vznikl v raných devadesátých letech minulého století, tedy přibližně o dvacet let později než MVC. Microsoft začal podporovat MVP pro platformu .NET v roce 2006. Hlavním rozdílem je přesunutí některých kompetencí z části *Controller* (v MVP část *Presenter*) do *View*. *View* tak zodpovídá za obsluhu událostí jako stisky klávesnice, přejetí myši a podobně. Tuto obsluhu událostí zastává v architektuře MVC *Controller*. *View* komponenta může odpovídat například obrazovce aplikace a musí implementovat nějaké rozhraní; *Controller* pak drží referenci na *View* přes toto rozhraní. Fakt, že se programuje k rozhraní a ne k implementaci, umožňuje nahraditelnost implementace *View* například pomocí *Dependency Injection*. V roce 2006 navrhl Martin Fowler rozdělit tento stále ještě příliš obecný vzor na dva konkrétnější: *Supervising Controller* [37] a *Passive View* [34]. Většina ale tyto vzory označuje stále pouze jako MVP, nebo ještě hůře MVC. Obrázek 8.1 ilustruje interakce mezi jednotlivými částmi aplikace obou zmíněných přístupů.



Obrázek 8.1: MVP deriváty.

Passive View navrhuje tvorit část *View* co nejjednodušší, která sice obsluhuje podněty uživatele, ale logiku spjatou s danou akcí deleguje na část *Presenter*. Tato metoda umožňuje především snadno testovat aplikace s grafickým rozhraním. Protože většina logiky je v části *Presenter*, stačí otestovat tuto část s použitím *mock* implementací pro *View*. To v kontextu webových aplikací znamená především zrychlení procesu testování Pro plnohodnotné testy včetně uživatelského rozhraní pak lze použít například *Selenium*. Aplikační rámcem GWTP podporuje MVP ve formě právě *Passive View*.

Supervising Controller je vhodné využít v případě, že grafické komponenty podporují *data binding*. *View* tak může provádět základní CRUD operace přímo na modelu. Propojení částí *View* a *Model* je typicky definováno deklarativně, a proto *View* neobsahuje příliš složitou logiku. Úlohou části *Presenter* je zpracovávat komplexnější logiku prezentace vrstvy a případně reagovat na podněty uživatele. *Supervising Controller* je koncepcně blíže původní MVC architektuře, ale *View* není tak složité, protože je zde také kladen důraz na testovatelnost. Tento model je použit v aplikačním rámci Smart GWT.

8.2 Event Bus

Hlavním cílem MVP je rozdělit aplikaci na volně propojené (*loosely coupled*) komponenty. Jedním z dalších způsobů, jak docítit co nejmenší závislosti mezi komunikujícími komponentami, je mechanizmus sběrnice událostí. Ta funguje na principu synchronního způsobu komunikace *Publish/Subscribe*. Komponenta A tak nevolá přímo metody komponenty B, ale komponenta B se musí nejprve zaregistrovat k naslouchání na sběrnici událostí na určitý typ události. Potom, když komponenta A chce komunikovat, vyšle na sběrnici událost, a všechny komponenty, které na daný typ události čekají, jsou upozorněny. Sběrnice tak funguje jako komunikační kanál a komponenty o sobě v podstatě nemusí vědět, odtud plyne volné propojení. Tento způsob komunikace je vhodné používat v případech kdy daná událost může zajímat více částí aplikace. Naopak v případě, že obě komunikující strany jsou dopředu známy a nemění se, je výhodnější použít klasické volání metod, protože je mnohem rychlejší.

Příklad registrace naslouchající komponenty B typu `ObjectOpenedEvent`:

```
addRegisteredHandler(ObjectOpenedEvent.getType(), new ObjectOpenedHandler() {
    @Override
    public void onObjectOpened(ObjectOpenedEvent event) {
        if (event.isStatusOK()) {
            onAddObject(event.getItem(), event.getRelated());
        }
    }
});
```

Příklad vyvolání události komponentou A:

```
ObjectOpenedEvent.fire(this, status, digitalObject, related);
```

Oba příklady využívají rámec GWTP, stejně by kód vypadal i v GWT. GWTP navíc umožňuje nechat si vygenerovat třídy `ObjectOpenedEvent`, `ObjectOpenedHandler` na základě anotací v kódu.

8.3 Dependency Injection

*Dependency Injection*²(DI) je jedna z technik obecnějšího principu IoC (*Inversion of control*), která má za úkol snížit závislosti mezi komponentami tím, že se vyhodnotí až za běhu aplikace poskytovatelem závislostí (*dependency provider/container*). Nejsou tedy zaneseny přímo do zdrojového kódu zainteresovaných tříd, ale do konfiguračních souborů například v XML. Termín zavedl Martin Fowler a technika se stala populární zejména díky aplikaci rámci *Spring*. Dalšími známými rámci, které podporují DI, jsou *PicoContainer*, *Google Guice* a v podstatě i EJB 2.x a 3.x.

V GWT existuje knihovna pro DI s názvem Gin. Protože část aplikace na straně serveru není v GWT ničím omezena, lze použít libovolné řešení z výše popsaných. V klientské

2. Existuje překlad „vkládání závislostí“, ale příliš se nepoužívá.

8.3. DEPENDENCY INJECTION

části aplikace je vývojář odkázán právě na Gin, což je de facto *Google Guice* pro GWT. Z diagramů v části MVP je patrné, že *Presenter* nedrží referenci na konkrétní implementaci *View*, ale pouze na její rozhraní. V GWTP není potřeba tuto referenci nějak explicitně vytvářet, protože to je v kompetenci právě poskytovatele závislostí. Vyhodnocení, která implementace *View* bude vložena, závisí na konfiguraci. Z toho plyne, že lze vytvořit různé konfigurace například pro testování aplikace a produkční nasazení, navíc fakt, že závislosti jsou rozhodovány až za běhu aplikace, znamená, že není ani potřeba aplikaci kompilovat³. Je zde vidět obecný koncept mnoha aplikačních rámců, že části aplikace, zejména opakující se rutinní činnosti, jsou prováděny automaticky aplikačním rámcem na základě deklarativně definovaných konfigurací.

```
bind(Log.class).toProvider(LogProvider.class).in(Singleton.class);
bind(Configuration.class).to(ConfigurationImpl.class).asEagerSingleton();
bind(IPaddressChecker.class).to(RequestIPaddressChecker.class);

// action handlers
bindHandler(PutUserRoleAction.class, PutUserRoleHandler.class);
bindHandler(RemoveUserRoleAction.class, RemoveUserRoleHandler.class);
...

// DAO
bind(InputItemDAO.class).to(InputItemDAOImpl.class).asEagerSingleton();
bind(UserDAO.class).to(UserDAOImpl.class).asEagerSingleton();
...

// Fedora
bind(FedoraAccess.class).to(FedoraAccessImpl.class).in(Scopes.SINGLETON);
bind(FedoraAccess.class).annotatedWith(Names.named("securedFedoraAccess"))
    .to(SecuredFedoraAccessImpl.class).in(Scopes.SINGLETON);
bind(NamespaceContext.class).to(FedoraNamespaceContext.class)
    .in(Scopes.SINGLETON);

// static injection
requestStaticInjection(FedoraUtils.class);
requestStaticInjection(AuthenticationServlet.class);
```

Příklad 8.3.1: Příklad konfigurace závislostí v *Google Guice* pro serverovou část aplikace.

Příklad 8.3.1 slouží jako ukázka konfigurace; jednotlivé řádky tvoří anglické věty, takže je kód dobře čitelný. Zajímavá je konfigurace pro Fedoru, tam je řečeno, že v případě použití anotace `@Named("securedFedoraAccess")` se použije uvedená implementace. V *Google Guice* se pro určení, že se na dané místo v kódu do daného rozhraní má vložit instance, používá anotace `@Inject`⁴.

3. Neplatí v případě *Google Guice*, kde je konfigurace tvořena kódem v Javě.

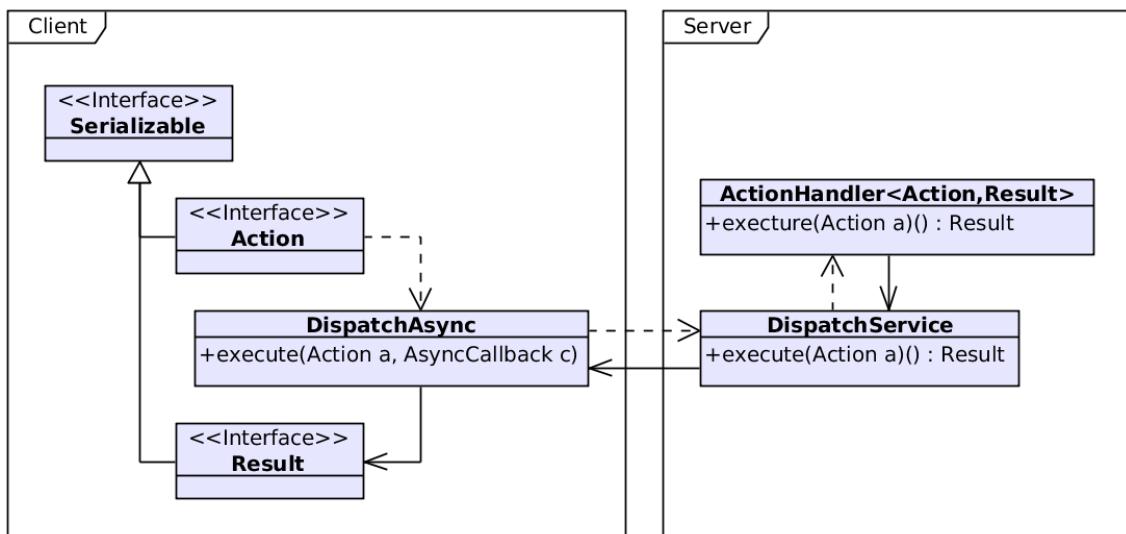
4. Nejčastěji se vyskytuje nad konstruktorem, možné je ale využít i vkládání přes anotovanou metodu.

8.4 Command pattern

Command pattern patří mezi behaviorální návrhový vzor popsaný původně skupinou *Gang of Four* v [3]. Umožňuje zapouzdřit operaci jako objekt. To s sebou oproti klasickému volání metody nese možnost tuto operaci provést později, provést více operací dávkově nebo seřadit operace tak, aby jejich vykonávání po sobě bylo co nejfektivnější. V případě, že operace vrací výsledek, je možné tímto návrhovým vzorem snadno umožnit kešování dvojic „požadavek a výsledek“. Dalším využitím tohoto vzoru může být implementace operací „zpět“ a „znovu“.

Protože v GWT už existovala třída *Command*, rozhodli se tvůrci nevnášet do pojmenování zmatek a nazvali třídu pro zapouzdření operace jako *Action*, výsledek volání jako *Result*, vykonavatele vlastní práce, který se původně nazývá *Receiver*, označují jako *ActionHandler* a konečně část původně označovaná jako *Invoker* starající se o plánování akcí odpovídá třídě *DispatchService*. Použitím nestandardního *DispatchService* lze docílit kešování operací. Pro využití operací „zpět“ a „znovu“ je potřeba k modifikujícím operacím vždy definovat jejich inverzní operace v metodě *undo(A action, R result, ExecutionContext context)* ve třídě dědící z *ActionHandler*. Parametr *ExecutionContext context* se používá i v jiných situacích a slouží k propojení drobných atomických operací do jedná velké transakce. Při vyhození výjimky tak lze vrátit předchozí operace podobně jako v databázovém prostředí.

Na obrázku 8.2 jsou přerušovanými čarami zobrazeny komunikace sloužící ke spuštění akce, která se provádí v potomkově třídy *ActionHandler*, a plnými čarami pak tok výsledku zpátky na stranu klienta. Ten získá výsledek z objektu *AsyncCallback*.



Obrázek 8.2: *Command pattern* v GWT.

Kapitola 9

Analýza

V této části práce bude popsána motivace pro vznik nového metadatového editoru a také bude probrána možnost spojení aplikací editoru a Krameria 4. Po první analýze požadavků jsem musel upustit od záměru nasadit aplikaci metadatového editoru do *cloudu*, protože by systém měl v dalších fázích zvládat vytváření nových metadat k naskenovaným stránkám a seskupovat je do větších celků, jako jsou články, ročníky, kapitoly (viz datový model na obrázku 4.1). Proces vytváření nových digitálních objektů s sebou nese potřebu mít naskenované stránky dostupné v adresářové struktuře. Obvykle se jedná o připojenou diskovou jednotku přes protokol NFS, to ale v aplikaci nasazené v *cloudu* nelze jednoduše uskutečnit. První verze nového metadatového editoru tak bude zvládat editovat existující digitální objekty z repozitáře Fedora a ukládat je po editaci zpět.

9.1 Zhodnocení stávajícího editoru

Pro systém Kramerius verze 3.x je k dispozici metadatový editor [15], který vznikl speciálně pro potřeby Moravské zemské knihovny upravením editoru [16] z projektu „Česká digitální matematická knihovna“ (DML-CZ). Tento editor vytvořil z velké části Martin Šárfy, systém na propojení článkových referencí pak implementoval Jiří Janků. Editor je komplexním nástrojem pro digitalizaci tištěných zdrojů, editaci jejich metadat a publikování do publikačního systému¹. Systém například obsahuje podporu pro OCR, automatickou detekci článků v matematických časopisech, generování PDF, autoritní databázi a další užitečné vlastnosti.

Lukáš Talich implementoval ve své diplomové práci konverzní nástroj, který umožňuje vkládat do repozitáře Fedora digitální objekty (DO) ve formátu FOXML, které byly vytvořeny v metadatovém editoru, dále v textu označovaný jako „konvertor“. Naopak DO z Fedory umožňuje konvertor nainportovat do metadatového editoru a editovat. Konvertor byl implementován ještě v době, kdy systém Kramerius 4 nebyl vůbec dostupný², nicméně návrh doménového modelu se od té doby nijak dramaticky nezměnil. Konvertor sice funguje, ale nebyl nasazen do ostrého provozu pravděpodobně kvůli komplikacím uvedeným níže. Aplikace konvertoru jsou v podstatě dva programy ve formě skriptů s definovaným rozhraním.

1. Ve své původní verzi pro repozitář DSpace, jenž je používán v projektu DML-CZ, a ve verzi pro MZK pro publikování do systému Kramerius 3.x (diplomová práce Tomáše Hofmana).

2. Projekt se však dramaticky zpozdil. Nejen kvůli tomu jej převzala firma Incad.

9.1. ZHODNOCENÍ STÁVÁJÍCÍHO EDITORU

Při zvažování, zda zaintegrovat konvertor do systémů Kramerius 4 a metadatového editoru z projektu DML-CZ, vystalo několik komplikací, kvůli kterým jsem po konzultaci s Petrem Žabičkou z MZK usoudil, že bude výhodnější vydat se cestou nového editoru metadat. Mezi tyto komplikace patří:

1. Nesourodé technologie použité v systému metadatového editoru

Jádro editoru je založeno na aplikačním rámci *Ramaze* a jazyku *Ruby*, postupně, jak se aplikace zvětšovala, přibývaly komponenty v jiných jazycích. Po důkladné analýze jsem zjistil, že kromě *Ruby* využívá editor *Bash* skripty, části aplikace napsané v jazyce *C*, část pro OCR je v jazyce *C++*. Konvertor je v jazyce *Java*, skripty v jazyce *Perl*, pro zobrazování náhledů stránek bylo zvoleno *PHP* a pro podporu *AJAXu* zase *JQuery*. Instalace takového množství komponent je pro knihovníka nadlidským úkolem, proto je editor distribuován jako virtuální obraz s distribucí *Debian* pro *VMware*. V případě, že instituce má licenci pro *VMware*, je instalace poměrně jednoduchá.

2. Nesourodá architektura systému metadatového editoru a úložiště *Fedora*

Editor (DML-CZ) využívá pro reprezentaci metadat kromě souborů XML také svou vlastní adresářovou strukturu. Autor konvertoru o ní ve své práci napsal: „Největší nevýhodou této struktury je fakt, že ji není možné reprodukovat pouze ze znalosti obsahu digitálních objektů. Sama adresářová struktura totiž nese netriviální informace (např. identifikátory nebo časové údaje), čímž vzniká nekompatibilita se systémem Kramerius.“

Další komplikací plynoucí z nesourodé architektury a odlišné reprezentace metadat obou systémů je přístup k reprezentaci strukturálních metadat. Editor (DML-CZ) si ukládá pořadí stránek v textovém souboru *pagemap.txt* a strukturální metadata jsou implicitně zakódovaná do adresářové struktury, zatímco *Fedora* využívá koncept pojmenovaného grafu v podobě *RDF*. Strukturální metadata ukládá explicitně k digitálním objektům do proudu *RELS-EXT*.

3. Nemožnost verzování DO

Fedora podporuje udržovat více verzi datových proudu v jednom DO. Při změně datového proudu DO se vytvoří jeho nová verze. Metadatový editor (DML-CZ) však podporuje jen jednu aktuální platnou verzi metadat. Přístup konvertoru je takový, že při importu z *Fedory* do editoru se z dat v uložených ve *FOXML* vytvoří adresářová struktura pro daný digitální objekt za použití posledních verzí datových proudu. V případě exportu z editoru zpátky do repozitáře je původní DO smazán, tzn. i veškerá jeho historie změn, a je nahrán DO nový, který obsahuje jen poslední verzi a je sestaven z metadat uchovávaných aplikací editoru.

4. Jeden datový proud

Pro export do repozitáře je povoleno využít v souborech FOXML jen jeden datový proud (viz strana 42 v [38]). V praxi ale DO obsahuje průměrně kolem sedmi datových proudů.

5. Možná inkonzistence

Editor využívá vlastní systém identifikátorů pro digitální objekty, přičemž Fedora využívá standard UUID. V aplikaci konvertoru tak bylo zavedeno mapování identifikátorů mezi systémy. Fakt, že v jeden moment může být DO přítomen na dvou místech, navíc s jiným identifikátorem, vede většinou k inkonzistencím.

I přes tyto výtky je aplikace konvertoru funkční a autor si zaslouží moje uznání, protože vyvíjet aplikaci pro tehdy ještě neexistující software muselo být jistě obtížné.

9.2 Dohoda s firmou Incad

Nejzásadnějším rozhodnutím bylo, zda webovou aplikaci nového metadatového editoru zaintegrovat rovnou do systému Kramerius 4, nebo ji pojmut jako samostatnou webovou aplikaci. Kramerius 4 je vyvíjen na platformě Java EE, využívá Google Guice, na některé části je použito GWT, ale převládá technologie servletů. Kramerius 4 vzniká jako open source pod licencí *GNU General Public License v3*. Pro aplikaci nového metadatového editoru byly použity stejné technologie. To je výhodné z hlediska udržovatelnosti obou systémů.

Firma Incad sídlí v Praze a po několika poradách jsme se dohodli, že aplikace metadatového editoru bude aplikací vně systém Kramerius 4, protože na něm není v podstatě vůbec závislá. Nový metadatový editor by měl umět editovat metadata digitálních objektů uložených v repozitáři Fedora, proto jej lze použít i v případech, kdy instituce využívá repozitář Fedora, ale nevyužívá systém Kramerius 4. Komunikace mezi systémy Kramerius 4 a metadatovým editorem probíhá jen ve dvou případech, a to v případě, kdy je digitální objekt upraven a publikován do repozitáře, tehdy je potřeba spustit reindexaci³, aby bylo možné objekt vyhledat pomocí systému Kramerius 4. Druhým případem komunikace je otevření editace digitálního objektu přímo z Krameria 4. Jedná se tedy o velmi volnou vazbu a systémy jsou na sobě v podstatě nezávislé, což se o závilsti metadatového editoru na repozitáři Fedora říci nedá.

3. Systém Kramerius 4 využívá pro indexaci Apache Solr a knihovnu Apache Lucene.

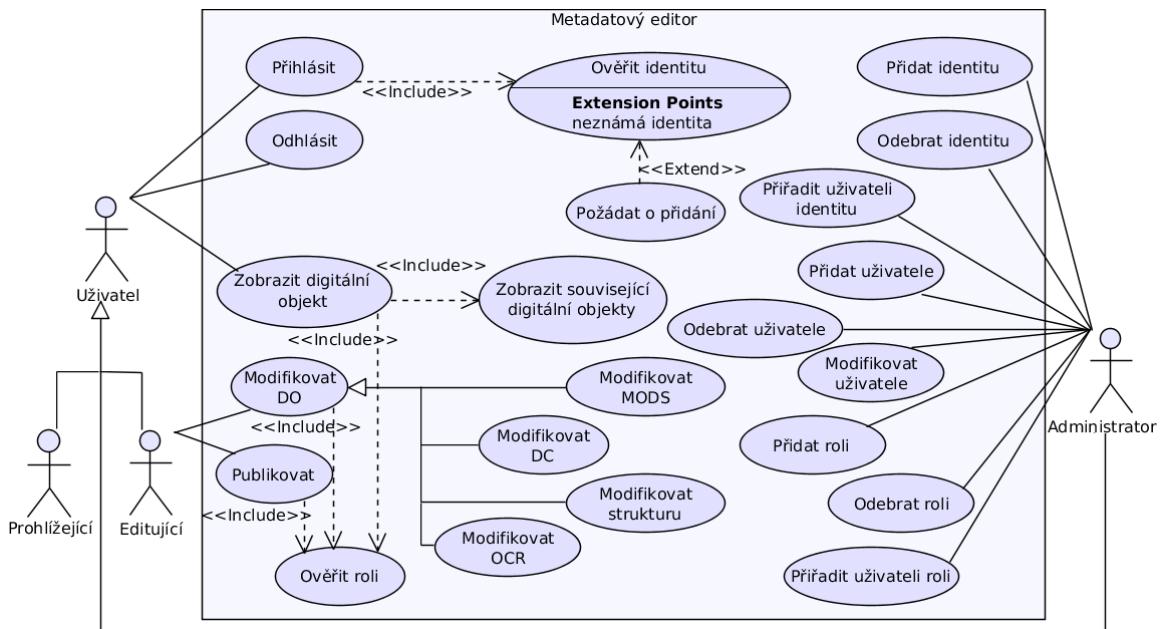
Kapitola 10

Návrh

V této kapitole se budu zabývat návrhem systému. Systém bude popsán převážně pomocí UML diagramů a ERD diagramu v případě schématu databáze.

10.1 Případy užití

Po konzultaci funkčních požadavků s lidmi, kteří editor budou používat, jsem vytvořil diagram s případy užití 10.1. Protože rolí v systému je poměrně hodně, byly pro tento účel sloučeny podobné role jako *structure editor*, *OCR editor*, *can publish* do jediné role „Editující“ a nemodifikující role do role „Prohlížející“.



Obrázek 10.1: Diagram užití metadatového editoru.

10.1. PŘÍPADY UŽITÍ

Diagram užití popisují následující uživatelské příběhy (*user stories* [40]).

1. Uživatel spustí webovou aplikaci buď odkazem z Krameria, nebo zadáním URL do prohlížeče. Není-li přihlášen, nebo v případě, že vypršela platnost sezení, je vyzván k autentizaci přes *OpenID*. V případě, že přišel z aplikace Kramerius, je pro něj otevřen digitální objekt, který chce editovat. V druhém případě je otevřena uvítací stránka, kde má možnost ověřit dostupnost systémů Fedora a Kramerius a také vložit UUID digitálního objektu, jenž chce editovat. Digitální objekt, který chce editovat, může také vybrat z levé nabídky naposledy otevřených objektů. V této nabídce jsou zobrazeny buď digitální objekty naposledy otevřené všemi uživateli, nebo jen přihlášeným. Mezi možnostmi lze volit.
2. Uživateli je umožněno procházet ve struktuře digitálního objektu tak, že pro přechod na předka¹ ve struktuře digitálního objektu stačí kliknout na DO v levé liště v části „referenced by“. Pro přechod na potomka je nutno nejprve vybrat digitální objekt v části se zobrazenou strukturou², a poté přes kontextové menu zvolit *Edit*. V případě stránky ji lze dvojklikem zobrazit, aplikace však neposkytuje pro prohlížení takový komfort jako Kramerius.
3. V případě, že je již jeden DO otevřen, při otevření dalšího jsou zobrazeny oba dva. V případě otevření třetího je první zavřen a jsou zobrazeny jen poslední dva. To umožňuje metodou *drag-and-drop* přesunovat například stránky nebo kapitoly mezi různými DO. Otevřené DO lze zavírat tlačítkem s krížkem vpravo nahoře.
4. Editace struktury probíhá, jak už bylo zmíněno, metodou *drag-and-drop*. Lze tak snadno měnit pořadí stránek, článků atd. Části tvořící strukturu (budou je označovat jako položky) lze vybírat za pomoci kláves CTRL a SHIFT s efektem známým z většiny OS. V kontextovém menu pro editaci struktury jsou navíc volby usnadňující manipulaci s položkami. Lze tak vybrat všechny položky, zrušit nebo invertovat výběr, kopírovat vybrané do schránky, vložit položky ze schránky do struktury a smazat vybrané.
5. Editace metadat v *Dublin Core* je intuitivní. Protože všech patnáct elementů je opakovatelných, lze stiskem tlačítka se znakem plus přidat nový formulářový prvek a naopak stiskem tlačítka se znakem mínsus poslední prvek odebrat.
6. Editace MODS metadat funguje na stejném principu, ale protože struktura MODS je mnohem složitější, je stránka s editací rozdělena ještě na dvacet dalších tabů, které odpovídají dvaceti elementům na nejvyšší úrovni standardu MODS. Vzhledem k tomu, že knihovní pracovníci znají spíše formát MARC21, je po přesunu kurzoru myši nad příslušný formulářový prvek nabídnuta návodčina včetně ekvivalentu v MARC21.

1. Z metadat pro stránku se tak lze například dostat k metadatům kapitoly, monografie apod.
2. Liší se v závislosti na typu DO, například stránka už žádné potomky nemá.

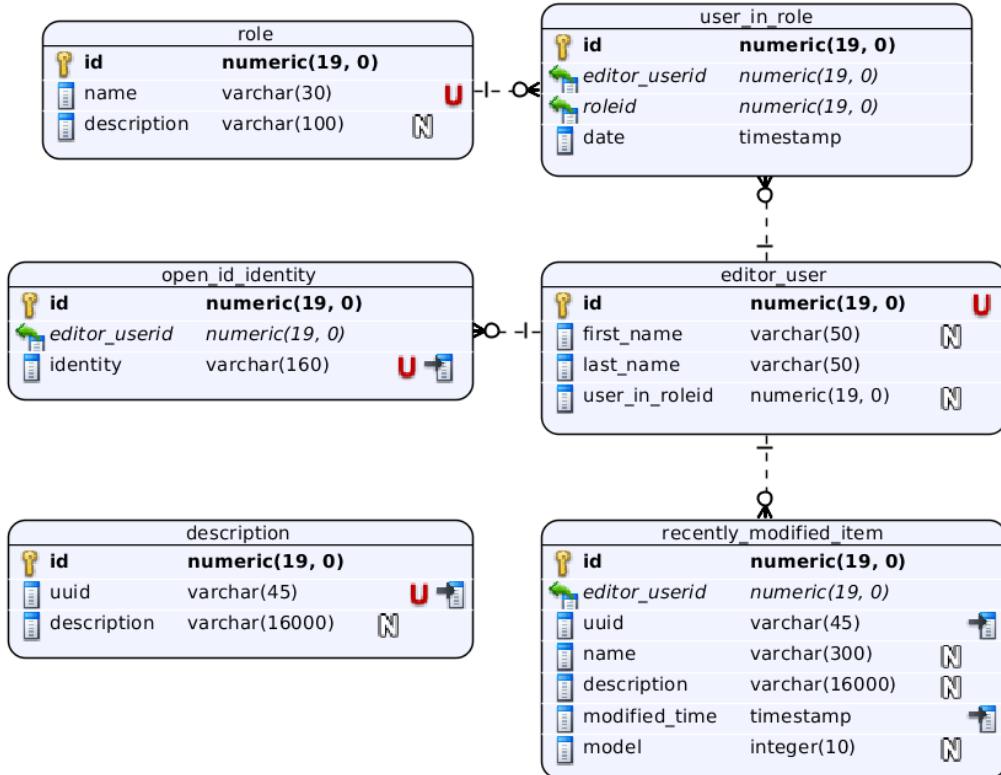
7. Uživatel si může psát k editovaným DO poznámky, které poté nebudou součástí metadat publikovaných do repozitáře. Tato možnost je dostupná přes tab *Description*. Vpravo lze vybrat, zda bude poznámka viditelná všem editujícím, nebo jen aktuálnímu přihlášenému uživateli. Protože nástroj pro editaci poznámek obsahuje možnosti pro obarvování textu, je možné v případě sdílené poznámky využít toto místo i například pro komunikaci, podobně jako v případě *Google Documents*. Osobní poznámka se navíc zobrazuje při najetí kurzoru myši nad DO v nabídce posledně otevřených.
8. Uživatel si může zobrazit soubor ve formátu FOXML, který reprezentuje DO tak, jak jej aplikace získala z repozitáře. Soubor je dostupný přes tab s názvem FOXML a je určený jen pro čtení. Tato reprezentace metadat nereflektuje změny prováděné v ostatních tabech, což by mohlo být zajímavé vylepšení do budoucna.
9. V menu vpravo nahoře lze DO znova načíst z repozitáře a zahodit tak předchozí změny, publikovat do repozitáře, v případě, že je uživatel oprávněn akci provést. Dále se v této nabídce vyskytují položky, které budou dostupné až v dalších verzích vývoje. Jedná se o ukládání DO v průběhu editace, uzamknutí jednotlivých tabů nebo celého dokumentu s úmyslem zamezit konkurenční přístup, a odstanění celého DO.
10. Má-li přihlášený uživatel právo editovat uživatele, je mu zobrazena vpravo nahoře volba *User Management*. Po kliknutí je zobrazena stránka s editací uživatelů. Je tak možno přidávat nové uživatele, přiřazovat uživatele do rolí, nebo přiřazovat uživatelům *OpenID* identity.
11. Přihlásí-li se uživatel do systému s neznámou *OpenID* identitou, je přesměrován na stránku, kde je mu řečeno, že má kontaktovat správce pro přidání, a událost je zalogovaná.
12. Uživatel se může kdykoli odhlásit a ukončit tak práci s aplikací.

10.2 Databáze

Protože systém převážně pracuje s daty přímo z repozitáře, není databázové schéma komplikované. Obsahuje šest tabulek znázorněných na diagramu 10.2. Tabulky `role`, `user_in_role`, `open_id_identity` a `editor_user` jsou využívány autentizačními a autorizačními mechanizmy aplikace. Tabulka `recently_modified_item` uchovává naposledy modifikované DO společně s osobní poznámkou k DO. Poznámka vzniká tak, jak je napsáno v bodě [7] v předchozí sekci. Poslední tabulkou je tabulka `description`; ta slouží k ukládání sdílené poznámky také popsané v předchozí sekci. Pro vytváření hodnot sloupce `id` má každá tabulka navíc svou vlastní sekvenci (SQL).

10.3. ARCHITEKTURA

Jako databázový systém byl zvolen open source *PostgreSQL* (dále jen *Postgres*). Tato volba byla učiněna částečně proto, že systém Kramerius 4 využívá také *Postgres*, a také proto, že se jedná o pravděpodobně nejlepší open source variantu.



Obrázek 10.2: Entity-relationship diagram.

10.3 Architektura

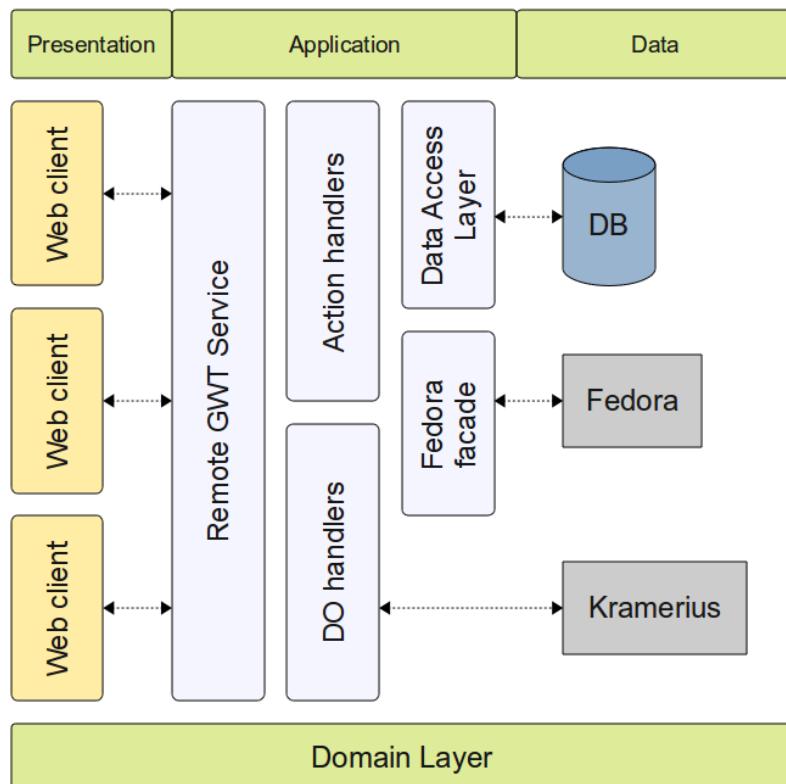
Metadatový editor staví na vícevrstvé architektuře, jak je ukázáno na obrázku 10.3. Toto členění aplikace do nezávislých celků umožňuje například provozovat databázi na jiném zařízení a je také vhodné pro provozování aplikace v *clusteru*.

Pro komunikaci s databází byla vytvořena vrstva, v níž pro každou třídu z doménového modelu, jenž je potřeba persistovat, je vytvořeno DAO³ rozhraní a jeho implementace. Pomocí návrhového vzoru *Dependency Injection* jsou tato rozhraní inicializována nastavenými implementacemi za běhu aplikace. Při změně architektury datové vrstvy, např. na cloudové řešení jako *BigTable*, je potřeba dodat nové DAO implementační třídy. Zpočátku jsem využíval výhod ORM rámce *Hibernate*, ale doba komplikace byla delší asi o 40 vteřin a schéma databáze je opravdu jednoduché, přešel jsem na nízkoúrovňové JDBC.

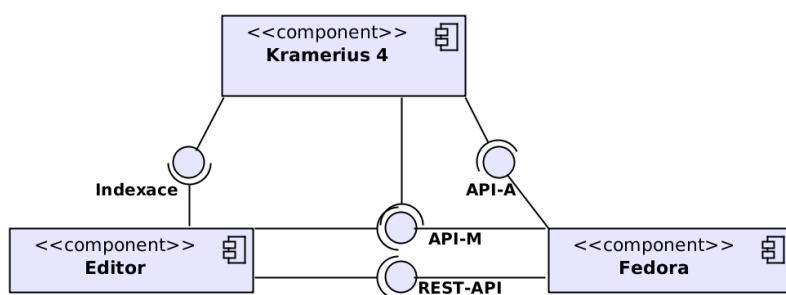
3. Data access object je Java EE návrhový vzor [5] sloužící k vytvoření abstrakce pro práci s databází.

10.3. ARCHITEKTURA

Pro komunikaci s repozitářem Fedora slouží fasáda s názvem *FedoraAccess*. Ta je částečně převzata ze zdrojových kódů pro Krameria 4 a částečně implementována. Ke komunikaci se používají SOAP webové služby v případě metod převzatých z Krameria a RESTful webové služby v případě nově dopsaných metod. Interakce systémů Editor, Kramerius 4 a Fedora je znázorněna na diagramu komponent 10.4. Je z něj též vidět, že Editor také volá indexaci v systému Kramerius 4.



Obrázek 10.3: Architektura systému.



Obrázek 10.4: Diagram komponent znázorňující interakci systémů.

Kapitola 11

Popis metadatového editoru

V této kapitole popíši, jak editor nainstalovat a nakonfigurovat, dále zde budou podány informace o nasazení systému na veřejné IP adresu. Popíši také, jak je aplikace zabezpečená a jak využívá autentizaci přes *OpenID*; nakonec zmíním statistiky projektu, především řádky kódu. Nebudu zde popisovat strukturu projektu, balíky tříd, ani diagramy tříd, protože projekt je poměrně rozsáhlý. Veškeré zdrojové kódy včetně samotné aplikace v archivu *war* a schéma databáze jsou dostupné z <http://code.google.com/p/meta-editor/>.

11.1 Instalace a konfigurace

Instalace metadatového editoru je jednoduchá. Protože systém využívá databázi *Postgres*, je třeba ji nejprve nainstalovat. Poté vytvořit uživatele a databázi například následujícími příkazy.

```
CREATE USER meditor WITH PASSWORD 'changeme';
CREATE DATABASE meditor WITH OWNER meditor;
ALTER SCHEMA public OWNER TO meditor;
```

Webová aplikace splňuje specifikaci *Java Servlet 2.5*, proto k její nasazení postačí libovolný aplikační server splňující tuto specifikaci. Aplikace předpokládá konfigurační soubor *configuration.properties* umístěný v *\$HOME/.meditor*. Pokud konfigurační soubor není přítomen, je vytvořen společně s adresářem. Povinné hodnoty, které nelze odvordinut, jako například URL Krameria 4 a Fedory, je však třeba doplnit. Všechny nastavitelné atributy zachycuje tabulka 11.1.

11.1. INSTALACE A KONFIGURACE

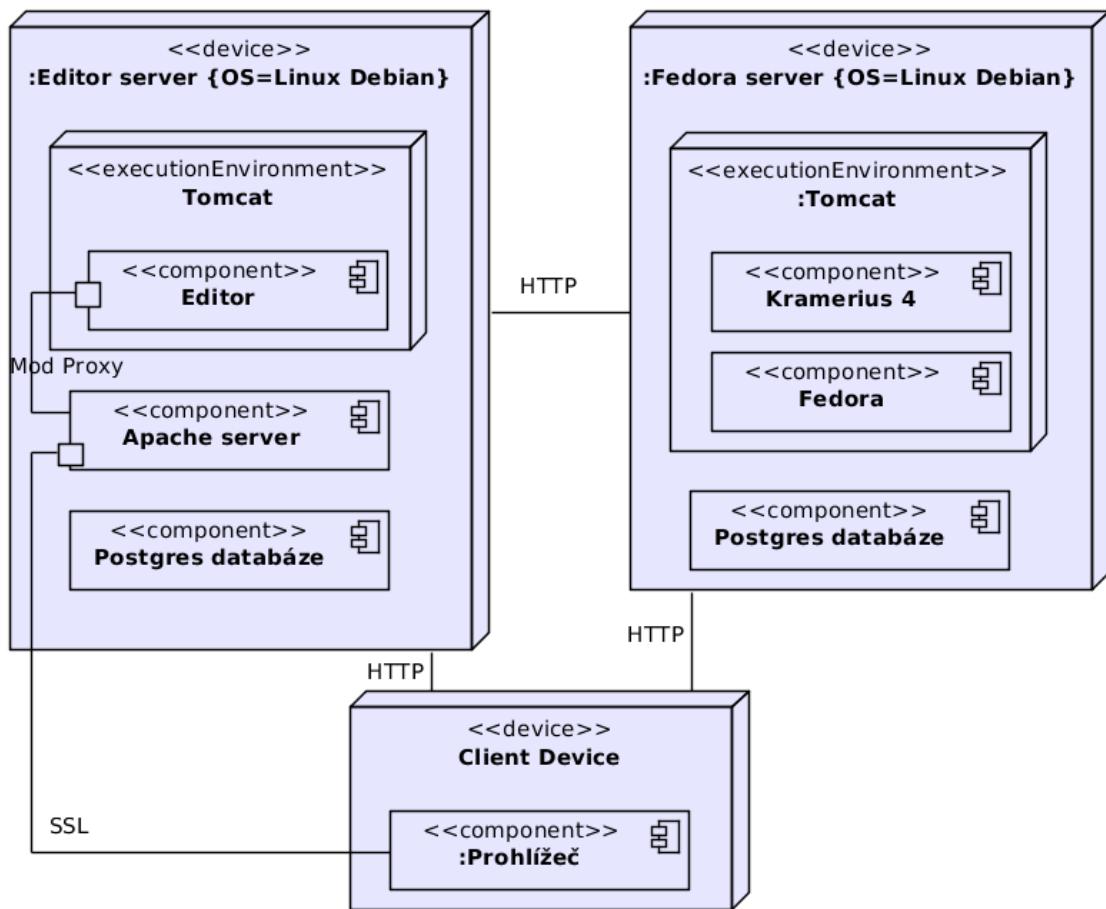
Atribut	Výchozí hodnota	Význam
krameriusHost	nutno vyplnit	URL na systém Kramerius 4
krameriusLogin	nutno vyplnit	přihlašovací jméno pro Kramerius 4
krameriusPassword	nutno vyplnit	heslo pro Kramerius 4
fedoraHost	nutno vyplnit	URL na repozitář Fedora
fedoraLogin	nutno vyplnit	přihlašovací jméno pro Fedoru
fedoraPassword	nutno vyplnit	heslo jméno pro Fedoru
dbHost	localhost	IP nebo hostname databáze
dbPort	5432	port databáze
dbLogin	meditor	přihlašovací jméno pro DB
dbPassword	nutno vyplnit	heslo pro databázi
dbName	meditor	jméno databáze
recentlyModifiedNumber	10	počet objektů, jenž budou zobrazeny vlevo v liště
z39.50Profile	mzk	při použití tohoto atributu není třeba vyplňovat následující 3. Možnosti jsou mzk, muni, nkp_skc a nkp_nkc.
z39.50Host	aleph.mzk.cz	adresa systému poskytující službu Z39.50
z39.50Port	9991	port systému poskytující službu Z39.50
z39.50Base	MZK01-UTF	název báze v službě Z39.50
barcodeLength	10	délka čárového kódu (podle něj hledá klient Z39.50)
accessUserPatterns	*	regulární výrazy určující IP adresy z nichž je možno k aplikaci přistupovat
accessAdminPatterns	127.*, localhost	totéž pro přístup administrátora
openIdApiKey	775a3d3ec29d...	kód získaný při registraci služby Janrain
openIdApiUrl	https://rpxnow.com	URL na Janrain API
gui.showInputQueue	yes	zobrazí vstupní frontu naske-novaných stránek
inputQueue	\$HOME/.meditor/input	cesta ke vstupní frontě
documentTypes	periodical, monograph	zobrazí se ve vstupní frontě vlevo v liště

Tabulka 11.1: Přehled atributů v konfiguračním souboru pro metadatový editor.

Atributy s prefixem „z39.50“ a poslední tři atributy nejsou v současné verzi využity. Ačkoli Z39.50 klinet je v aplikaci zabudovaný a funkční, není potřeba jej využívat, protože systém zatím neumí vytvářet nová metadata, ale pouze upravuje existující.

11.2 Nasazení

Webová aplikace je v současné verzi nasazena na virtuálním serveru v Moravské zemské knihovně a dostupná přes URL <http://editor.mzk.cz>. Vývojová verze aplikace Kramerius 4 je nasazena na serveru <http://krameriusdemo.mzk.cz>¹. Na stejně adrese je nasazena i instance repozitáře Fedora, tj. <http://krameriusdemo.mzk.cz:8080/fedora>. Databáze pro editor je nasazena na stejném stroji jako editor. Celá situace je zaznamenána diagramem nasazení 11.1.

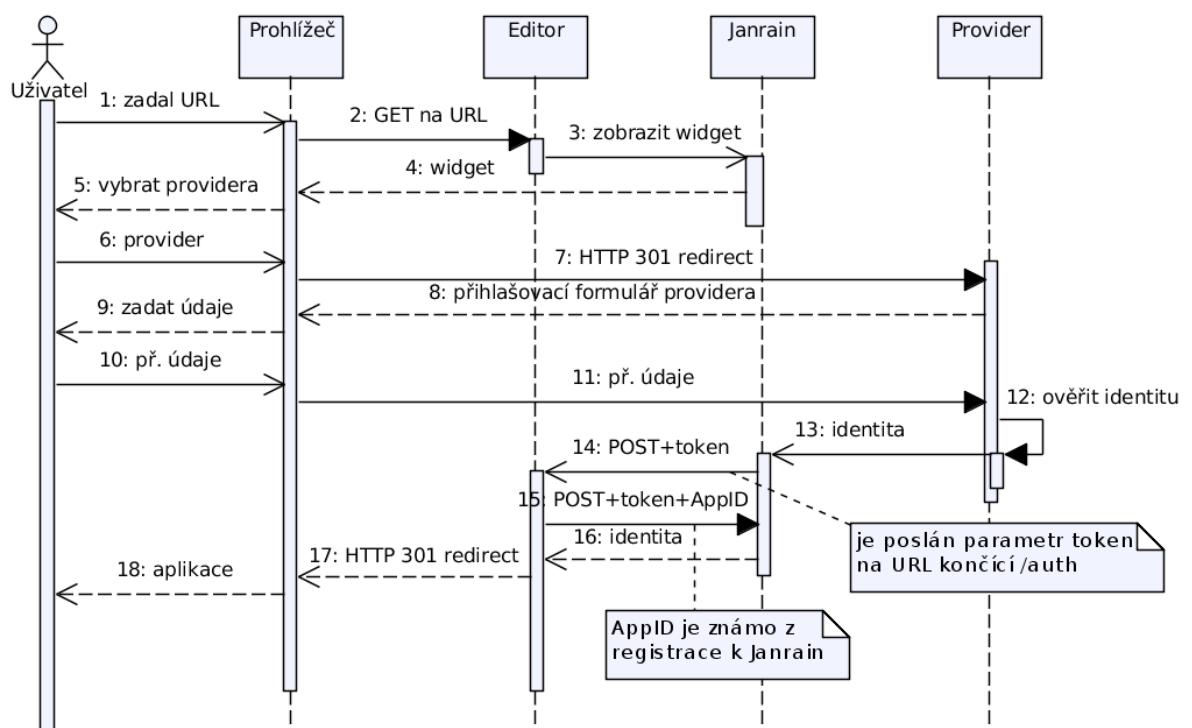


Obrázek 11.1: Diagram nasazení celého systému.

1. V budoucnu se pravděpodobně změní na <http://kramerius.mzk.cz>, kde je teď nasazena verze 3.

11.3 Bezpečnost

Autentizace uživatelů je řešena standardem *OpenID*. Konkrétně je využit *widget* společnosti *Janrain* [26]. Ten provede autentizaci u vybraného poskytovatele identity (také označován jako správce identit) a v kladném případě pak přesměruje uživatele zpátky na aplikaci editoru (v terminologii *OpenID* tzv. *Relying Party*). Editor si ověří, jestli má vrácený *OpenID* identifikátor v databázi, a v případě, že ano, uloží si jej do session a uživatel je přesměrován na požadovanou stránku v editoru, nyní již přes protokol SSL. Jeden uživatel může mít přiřazených více *OpenID* identifikátorů². Pro názornost je situace detailně rozpracována na diagramu 11.2.



Obrázek 11.2: Sekvenční diagram znázorňující proces autentizace.

Pro přesměrování nepřihlášeného uživatele na stránku s přihlášením používá aplikace třídu *AuthenticationFilter*, tzn. potomka *javax.servlet.Filter*. Filter rozhodne o tom, zda je uživatel přihlášený podle přítomnosti identifikátoru v session. V aplikaci existuje také servlet *AuthenticationServlet*, který zpracuje komunikace 14, 15, 16 a 17 ze sekvenčního diagramu.

2. Například může platit, že se chce přihlašovat přes Google, jindy přes *LinkedIn* a jindy přes Facebook.

11.3. BEZPEČNOST

Webová aplikace metadatového editoru je spuštěna v servlet kontejneru *Apache Tomcat*, kvůli lepší bezpečnosti je též využit *Apache HTTP Server*, který přes modul `mod_proxy` směruje požadavky z portu 443 na SSL konektor kontejneru Tomcat (port 8443). Certifikát používaný v *Apache HTTP Server* na stroji `editor.mzk.cz` je podepsán asociací *TERENA*, jež je standardně důvěryhodná pro všechny nejpoužívanější prohlížeče; uživatel tak nemusí potvrzovat důvěryhodnost certifikátu manuálně.

Standard *OpenID* zajišťuje pouze autentizaci, autorizace je řešena na úrovni systému. Editor obsahuje několik rolí, které s sebou nesou práva, která uživatel s příslušnou rolí má. Jeden uživatel může mít roli více. V současné verzi systému je rozlišována pouze role `admin`, která má práva editovat uživatele a přidávat jim role a identity. Ostatní role budou dopracovány v pozdějším vývoji.

Jedna z výhod plynoucích z použití *OpenID*, *OAuth* a podobných protokolů je skutečnost, že uživatel se nemusí registrovat do každého systému zvlášť³, ale používá třetí stranu k provedení autentizace. Na druhou stranu používání těchto mechanizmů snižuje anonymitu uživatele na Internetu a osobní údaje o něm tak lze mnohem lépe agregovat z více heterogenních zdrojů. Je otázkou, zda tyto služby přináší více užitku nežli rizik zneužití. Jasně ale je, že pro technologie Webu 2.0, jako jsou například *mashup* aplikace, neexistuje zatím žádná principiálně lepší varianta.

Díky výhodám spjatým s protokolem *OpenID* nemusí editor ukládat uživatelská hesla a řešit jejich kódování, hašování se solem, posílání po nezabezpečeném kanálu apod. Autentizační mechanizmus bude v budoucnu pravděpodobně nahrazen technologií *Shibboleth* nebo propojením *OpenID* na LDAP systém v dané instituci.

3. Studie provedená společností *BitDefender* ukázala, že ze vzorku osobních údajů na sociálních sítích používá 75 % uživatelů stejné přihlašovací údaje k dané službě a k emailu získanému při registraci do dané služby [35].

11.4 Statistiky

V projektu jsem používal výhod generátorů kódu. Celkem 8010 řádků bez komentářů a prázdných řádků bylo vygenerováno. Jedná se o třídy tvořící klienta pro webovou službu k rozhraní Fedora API-A a API-M (4066 řádků), třídy vygenerované nástrojem *xjc* ze XML Schéma pro standard MODS (3047 řádků). Jedná se o třídy sloužící k přístupu k XML pomocí rámce JAXB (*Java Architecture for XML Binding*). Dále byly generovány třídy rámcem GWTP na základě anotací v kódu, zejména třídy reprezentující akce (viz sekce 8.4) a události (viz sekce 8.2). Podrobnější statistiky jsou zobrazeny v tabulkách [11.2] a [11.3]. K výpočtu řádků kódu byl použit nástroj *cloc*⁴. Zdrojové kódy jsou dostupné pod licencí GNU GPL verze 2 na adrese <http://code.google.com/p/meta-editor>.

Jazyk	Souborů	Prázdné řádky	Komentáře	Řádky kódu	Řádky cel.
Java	263	4206	8972	20578	34019
CSS	1	36	2	228	267
XSD	2	0	22	152	176
XML	3	35	48	151	237
HTML	4	41	34	135	214
Bash	2	2	1	7	12
Suma	275	4320	9079	21251	34650

Tabulka 11.2: Statistika napsaných řádků kódu.

Jazyk	Souborů	Prázdné řádky	Komentáře	Řádky kódu	Řádky cel.
Java	161	2468	12993	8010	23632

Tabulka 11.3: Statistika vygenerovaných řádků kódu.

4. <http://cloc.sourceforge.net>

Kapitola 12

Závěr

V teoretické části této diplomové práce jsem nejprve popsal nejpoužívanější standardy pro ukládání metadat a jejich výměnu. V dalších kapitolách jsem popsal architekturu a pozadí vývoje systémů, se kterými webová aplikace metadatového editoru spolupracuje. Kapitoly 5 a 6 můžou sloužit jako shrnutí a učební pomůcka v oblasti bohatých webových aplikací. Může se zdát, že kapitola o HTML5 není příliš tématicky spjata se zbytkem práce. Dle mého názoru tomu tak ale není, protože s rozšířením tohoto standardu budou muset všechny rámce sloužící k tvorbě RIA aplikací přehodnotit své principy a nakonec se adaptovat na tento standard.

Velkou pozornost věnuji v práci aplikačnímu rámci *Google Web Toolkit*, protože je využit jako hlavní technologie pro prezentační vrstvu metadatového editoru. Základní principy tohoto rámce jsou popsány velmi stručně, o to větší pozornost je věnována pokročilejším tématům jako jsou návrhové vzory spojené s GWT a pomocné rámce. Díky tomuto rámci lze vyvíjet rozsáhlé aplikace a využít všech výhod spjatých s platformou Java EE.

Další části práce popisují postupně analýzu, návrh a popis metadatového editoru. V části, která se zabývá analýzou, je zhodnocen stávající editor pro systém Kramerius 3 a následně obhájeno rozhodnutí vytvořit editor jako samostatnou webovou aplikaci. Část návrhu je doprovázena četnými UML diagramy a je v ní popsána uživatelskými příběhy i funkcionality výsledného editoru. V poslední části popisují instalaci, konfiguraci a nasazení editoru.

Vyvinutá aplikace splňuje požadavky, jež na ni byly kladené, to znamená, že umožňuje editovat metadata reprezentovaná digitálními objekty serializovanými do formátu FO-XML. Využívá moderních technologií a působí tak dojemem desktopové aplikace. Odezva aplikace je poměrně rychlá, až na případy, kdy se provádí mnoho požadavků do systému Fedora. Jedná se o případ, kdy má kniha mnoho stránek, a je třeba náhled pro každou stránku. V takovém případě se aplikace může jevit jako pomalejší. Tento nedostatek by šlo odstranit v budoucích fázích vývoje například využitím projektu *Memcached* nebo *Ehcache*.

Nejvíce práce představovalo zabudovat editaci pro standard MODS, protože standard je velmi rozsáhlý a flexibilní. Formát je rekursivní, takže může obsahovat v jednom ze svých elementů například celou svou kopii. Kdybych mohl zpětně změnit některá rozhodnutí, nepoužil bych rámec Smart GWT, pro nepříliš dobrou dokumentaci a neintuitivní API. Naopak rámec GWTP mi usnadnil značně vývoj. Mám zkušenosť s velkým projektem v GWT, kde jsme nevyužívali vzor MVP, ani žádný rámec postavený nad platformou

GWT, a vývoj a údržba systému byla o poznání komplikovanější. Bohužel je dnes budoucnost rámce GWTP nejistá, jelikož Google nabízí vlastní implementaci návrhového vzoru MVP přímo v GWT od verze 2.1. I když dle mého názoru ne tak propracovanou.

Ve vývoji aplikace budu pokračovat i po odevzdání této práce, aby bylo možné vytvářet nová metadata z naskenovaných stránek. Editor by měl také umožnit zamykání části DO a také ukládat mezivýsledek editace a následně v editaci pokračovat. Editor bude využit Moravskou zemskou knihovnou a pravděpodobně i Národní knihovnou České republiky v Praze. Chtěl bych se dále hlouběji zabývat myšlenkou automatického zpracování metadat, extrakce informace a podobnými tématy v rámci doktorského studia, ať už jako vylepšení metadatového editoru, nebo formou samostatných projektů.

Spojení bohatých webových aplikací, které většinou postrádají sémantické informace o svém obsahu, se světem sémantického webu má před sebou velkou budoucnost. V práci jsem se záměrně vyhýbal označení Web 3.0, protože zatím nepocitují žádný kvalitativní pokrok oproti tzv. Webu 2.0, nicméně s nástupem kontextuálně zaměřených aplikací, mobilního Internetu a všudypřítomného počítání (*Ubiquitous computing*) bude tento pokrok znát.

Literatura

- [1] Bartošek, M.: *Digitální knihovny – teorie a praxe*, 2005, Ústav výpočetní techniky, Masarykova univerzita v Brně.
- [2] Hanson, R. a Tacy, A.: *GWT in Action: Easy Ajax with the Google Web Toolkit*, 2007, Manning Publications Co..
- [3] Gang of Four: *Design Patterns: Elements of Reusable Object-Oriented Software*, 1994, Addison-Wesley. 8.4
- [4] Freeman, E. a Robson, E. a Bates, B.: *Head First Design Patterns*, 2004, O'Reilly Media.
- [5] Crawford, W. a Kaplan, J.: *J2EE Design Patterns*, September 2003, O'Reilly Media. 5, 3
- [6] Segaran, T. a Evans, C. a Taylor, J.: *Programming the Semantic Web*, July 2009, O'Reilly Media.
- [7] Kesteren, A.: *XMLHttpRequest*, , 2009, Dostupný z URL <<http://www.w3.org/TR/XMLHttpRequest>> (leden, 2011) . 5.1
- [8] Google: *Making AJAX Applications Crawlable*, 2010, Dostupný z URL <<http://code.google.com/intl/cs-CZ/web/ajaxcrawling>> (leden, 2011) . 7.4
- [9] Hoffman, B. a Sullivan, B.: *Ajax Security*, December 2007, Addison Wesley. 5.3
- [10] Hoffman, B.: *Advanced Ajax Security*, October 2010, Dostupný z URL <http://ptgmedia.pearsoncmg.com/imprint_downloads/voicesthatmatter/gwt2007/presentations/AdvancedAjaxHacking_Hoffman.ppt> (leden, 2011) . 5.3
- [11] McLaughlin, B.: *Head Rush Ajax*, O'Reilly Media, March 2006.
- [12] Network Development & MARC Standards Office: *Bib-1 Attribute Set*, October 2007, Dostupný z URL <<http://www.loc.gov/z3950/agency/defns/bib1.html>> (leden, 2011) . 2.7.1
- [13] WEBFLUX: *CSS3 Module Status*, 2010, Dostupný z URL <<http://www.css3.info/modules>> (leden, 2011) . 6.1
- [14] ÚVT MU: *Dublin Core - Czech homepage*, 2006, Dostupný z URL <http://www.ics.muni.cz/dublin_core/index.html> (leden, 2011) . 2.1
- [15] Šárfy, M.: *Metadatový editor pro digitální knihovny*, 2009, Ústav výpočetní techniky, MU. 9.1

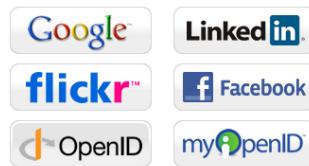
-
- [16] Bartošek, M. a Kovář, P. a Šárky, M.: *DML-CZ Metadata Editor*, 2008, Institute of Computer Science, Masaryk University. 9.1
 - [17] Davis, D.: *Content Model Architecture*, July, 2008, Dostupný z URL <<https://wiki.duraspace.org/display/FCR30/Content+Model+Architecture>> (leden, 2011) . 3.2
 - [18] Fedora Commons contributors: *The Fedora Digital Object Model*, August, 2007, Dostupný z URL <<http://www.fedora-commons.org/documentation/3.0b1/userdocs/digitalobjects/objectModel.html>> (leden, 2011) . 3.1
 - [19] Australian Partnership for Sustainable Repositories: *Fez*, 2009, Dostupný z URL <<http://sourceforge.net/projects/fez/>> (leden, 2011) . 3
 - [20] Wikipedia contributors: *Freebase (database)*, 2008, Dostupný z URL <[http://en.wikipedia.org/wiki/Freebase_\(database\)](http://en.wikipedia.org/wiki/Freebase_(database))> (leden, 2011) . 2.5
 - [21] Gartner, Inc: *Gartner Highlights Key Predictions for IT Organizations and Users in 2010 and Beyond*, 2010, Dostupný z URL <<http://www.gartner.com/it/page.jsp?id=1278413>> (leden, 2011) . 5
 - [22] Hofstadter, D.: *Gödel, Escher, Bach: An Eternal Golden Braid*, 1979, Basic Books. 2.8
 - [23] Google: *Organize Projects - Google Web Toolkit*, 2010, Dostupný z URL <<http://code.google.com/intl/cs-CZ/webtoolkit/doc/latest/DevGuideOrganizingProjects.html>> (leden, 2011) . 7.1
 - [24] Beaudoin, P.: *gwt-platform*, 2010, Dostupný z URL <<http://code.google.com/p/gwt-platform>> (leden, 2011) . 7.4
 - [25] Brewer, G.: *JQuery Usage Statistics*, January 2011, Dostupný z URL <<http://trends.builtwith.com/javascript/JQuery>> (leden, 2011) . 5.4
 - [26] Wikipedia contributors: *JanRain*, 2008, Dostupný z URL <<http://en.wikipedia.org/wiki/JanRain>> (leden, 2011) . 11.3
 - [27] Bondi, M. a Scott, M.: *Keyness in Texts*, 2010, Benjamins. 2
 - [28] Cyganiak, R.: *The Linking Open Data cloud diagram*, 2010, Dostupný z URL <<http://richard.cyganiak.de/2007/10/lod/>> (leden, 2011) . 2.1
 - [29] Wikipedia contributors: *Linked Data*, 2007, Dostupný z URL <http://en.wikipedia.org/wiki/Linked_Data> (leden, 2011) . 2.8
 - [30] Thomale, J.: *Interpreting MARC: Where's the Bibliographic Data?*, Septemner 2010, The Code4Lib Journal, No. 11. 2.2

-
- [31] W3C: *HTML Microdata*, November 2010, Dostupný z URL <<http://dev.w3.org/html5/md>> (leden, 2011) . 2.8, 6.3
 - [32] Melichar, M.: *Národní digitální knihovna*, leden, 2011, Dostupný z URL <<http://www.ndk.cz/narodni-dk>> (leden, 2011) . 4.1
 - [33] Kremser, J.: *OAI-ORE*, 2009, Dostupný z URL <http://dspace.muni.cz/bitstream/ics_muni_cz/923/1/oai-ore.pdf> (leden, 2011) .
 - [34] Fowler, M.: *Passive View*, July, 2006, Dostupný z URL <<http://martinfowler.com/eaaDev/PassiveScreen.html>> (leden, 2011) . 8.1
 - [35] Datcu, S.: *The Limits of Privacy. Is This Your Password?*, August 2010, Dostupný z URL <<http://www.malwarecity.com/blog/the-limits-of-privacy-is-this-your-password-865.html>> (leden, 2011) . 3
 - [36] Mahemoff, M.: *HTTP Streaming - Ajax Patterns*, October 2010, Dostupný z URL <http://ajaxpatterns.org/HTTP_Streaming> (leden, 2011) .
 - [37] Fowler, M.: *Supervising Controller*, June, 2006, Dostupný z URL <<http://martinfowler.com/eaaDev/SupervisingPresenter.html>> (leden, 2011) . 8.1
 - [38] Talich, L.: *Formáty repozitářů digitálních knihoven*, 2010, [diplomová práce] FI MUNI Brno. 4.1, 4
 - [39] Fowler, M.: *GUI Architectures*, July, 2006, Dostupný z URL <<http://martinfowler.com/eaaDev/uiArches.html>> (leden, 2011) . 8.1
 - [40] Ambler, S.: *Introduction to User Stories*, 2009, Dostupný z URL <<http://www.agilemodeling.com/artifacts/userStory.htm>> (leden, 2011) . 10.1
 - [41] Alesso, H. a Smith, C.: *Thinking on the Web: Berners-Lee, Gödel and Turing*, December 2008, .
 - [42] Network Development & MARC Standards Office: *Z39.50 Maintenance Agency Site Index*, 2004, Dostupný z URL <<http://www.loc.gov/z3950/agency/contents.html>> (leden, 2011) . 2.7.1

Příloha A

Uživatelské prostředí

Sign in using your account with

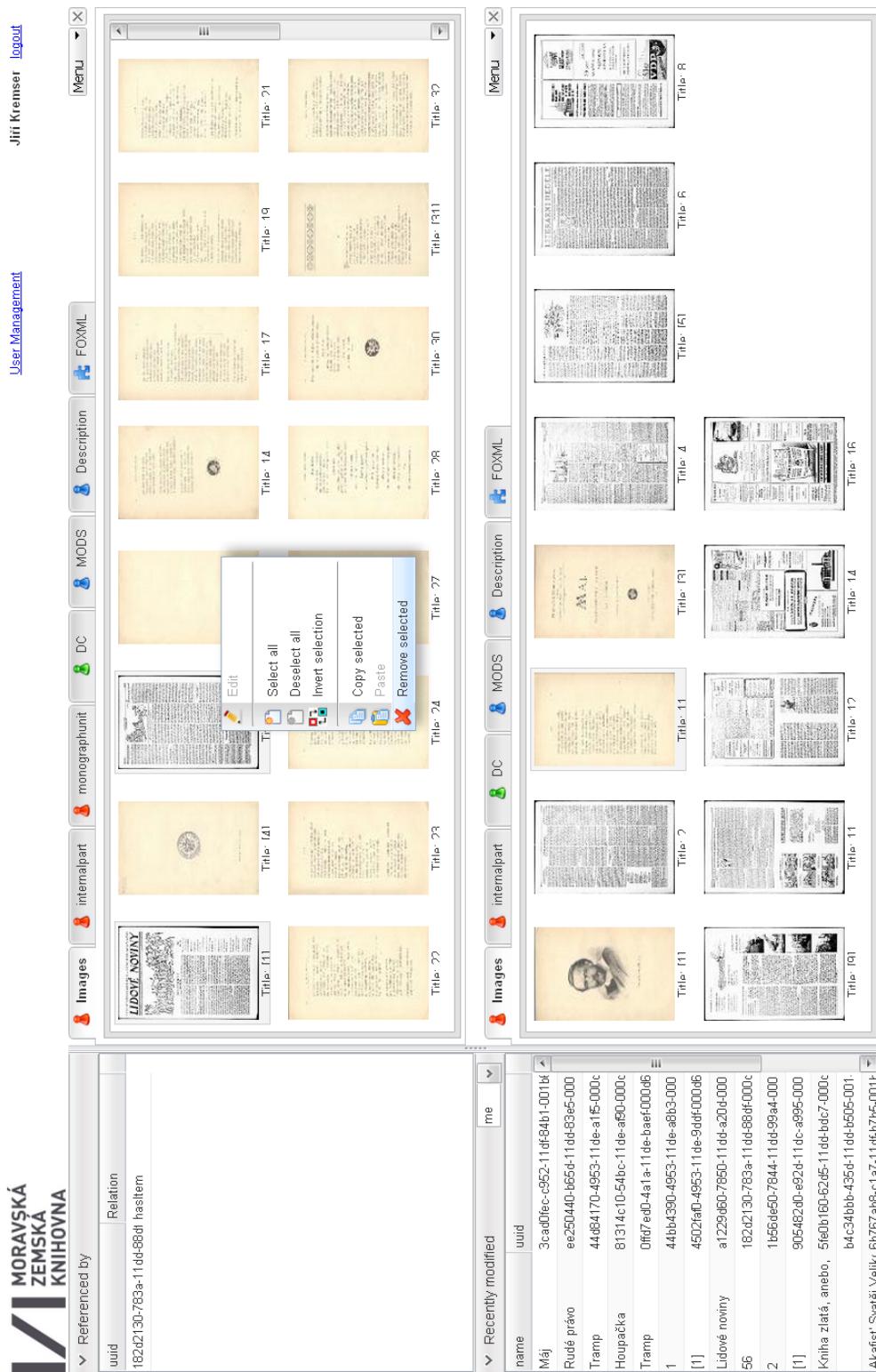


Obrázek A.1: Přihlašovací stránka s výběrem OpenID poskytovatelů.

The screenshot shows the "User Management" interface. On the left, a sidebar lists "Referenced by" items such as "Tramp", "Máj", "Lidové noviny", "Rudé právo", "Houpačka", "[1]", "[56]", "[2]", "Kníha zlatá, aneb, Nový zwěstovatel v... -none-", "Akafist* Svatý Vojtěch Varvare", "[21]", "Komár", "Drobňáštky", "Našinec", and "[4]". The main area has three tabs: "Users", "Roles", and "OpenID identities". The "Users" tab displays a table with columns "First Name" and "Last Name", listing Franta (Běžný Uživatel), Jíří (Kremser), Václav (Rošecák), and Pepa (Zdepa). The "Roles" tab displays a table with columns "Name" and "Description", listing "view_dc" (Can view DC data stream) and "view_users" (Can view users, roles, identities). The "OpenID identities" tab displays a table with a single entry: "identity" with the URL "https://www.google.com/profiles/1112270318244814". Buttons for "Add user", "Remove selected", "Add role", "Remove selected", and "Add identity" are visible at the bottom of their respective sections.

Obrázek A.2: Editace uživatelů.

A. UŽIVATELSKÉ PROSTŘEDÍ



Obrázek A.3: Aplikace s dvěma otevřenými digitálními objekty.

A. UŽIVATELSKÉ PROSTŘEDÍ

User Management

Jiří Kremlík logout

MORAVSKÁ ZEMSKÁ KNIHOVNA

Images Internal part mongraphunit DC MODS Description FOXML

Title Info Name Type Origin Language Physical desc. Abstract Table of Con. Audience

Name

Attributes

Type : personal

Authority :

Script :

xlink :

Lang :

xml:lang :

Transliteration :

ID :

Name Parts

Name Part : Mácha

Name Part : Karel Hynek

Type : family

Type : given

Includes each part of the name that is parsed. Parsing is used to indicate a date associated with the name, to parse the parts of a corporate name (MARC 21 fields \$x10 subfields \$a and \$b).

	[1]	[2]	[3]	[4]
name	uuid	uid	uid	uid
Tramp	44384170-4953-11de-a1f5-000c	ee250440-be5d-11dd-8e5e-0000	3ca0f6fc-c952-11df-8461-0010	3ca0f6fc-c952-11df-8461-0010
Rudé právo	81314c10-54bc-11de-a90-000c	0ff07-ed0-4a1a-11db-baf60006	81314c10-54bc-11de-a90-000c	81314c10-54bc-11de-a90-000c
Máj	44bfbd30-4953-11de-abb3-0000	44bfbd30-4953-11de-abb3-0000	4502fa0-9953-11de-9dd60006	4502fa0-9953-11de-9dd60006
Hlupáčka	1b56ae50-7844-11dd-99ad-0000	1b56ae50-7844-11dd-99ad-0000	a1229d60-7850-11dd-a20d-0000	a1229d60-7850-11dd-a20d-0000
Tramp	90546240-e92d-11dc-a995-0000	b4c34bbb-435d-11dd-bc05-0001	182d2130-783a-11dd-88df-000c	182d2130-783a-11dd-88df-000c
1	Kniha zlatá, aneb, Akafist Svatý Vojtěch, 6b767ab8-c1a7-11de-a7b5-0011	b4c34bbb-435d-11dd-bc05-0001	b4c34bbb-435d-11dd-bc05-0001	b4c34bbb-435d-11dd-bc05-0001
Lidové noviny	5fe0b160-52d5-11dd-bd07-0000	6d77d00-1ab3e-11dd-88cc-0000	6d77d00-1ab3e-11dd-88cc-0000	6d77d00-1ab3e-11dd-88cc-0000
56	1b56ae50-7844-11dd-99ad-0000	8b946f50-49b7-11de-b903-0000	c4cd04d0-4904-11de-96f6-0000	8b946f50-49b7-11de-b903-0000
2	90546240-e92d-11dc-a995-0000	c4cd04d0-4904-11de-96f6-0000	0eaab730-9063-11dd-97de-0000	c4cd04d0-4904-11de-96f6-0000
[1]	Kniha zlatá, aneb, Akafist Svatý Vojtěch, 6b767ab8-c1a7-11de-a7b5-0011	b4c34bbb-435d-11dd-bc05-0001	851fb04c-ce7111df8869d-001b6	b4c34bbb-435d-11dd-bc05-0001
21	851fb04c-ce7111df8869d-001b6	eeec95a0-0e5d-11dd-83e3-0000	eeec95a0-0e5d-11dd-83e3-0000	eeec95a0-0e5d-11dd-83e3-0000
Komár	[4]	[4]	[4]	[4]
Komár	[4]	[4]	[4]	[4]
Dobrníský	[4]	[4]	[4]	[4]
Nášnec	[4]	[4]	[4]	[4]

Display Forms Affiliations Roles

Type : code

Role : cne Authority :

Obrázek A.4: Editace metadat v MODS.

A. UŽIVATELSKÉ PROSTŘEDÍ



Obrázek A.5: Otevřená stránka básně Máj.