Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Formuláře pro získávání RDF dat

Plzeň 2017 Filip Jani

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 3. května 2017

Filip Jani

Abstract

This bachelor thesis cointains introduction to OWL and RDF. It deals with design and implementation of Java library used for dynamic generation of web forms and processing data to RDF. As a part of this thesis is also design and implementation of web application which uses created library to demonstrate library functions and options.

Abstrakt

Tato bakalářská práce poskytuje seznámení s oblastí OWL a RDF. Zabývá se návrhem a realizací Java knihovny pro dynamické generování formulářů a jejich obsluhu s uložením dat v RDF. Následně se zabývá návrhem a implementací ukázkové webové aplikace používající navrženou knihovnu pro demonstraci její funkčnosti a možností použití.

Poděkování

Tímto bych chtěl poděkovat Ing. Petru Včelákovi za cenné rady, připomínky a čas, který mi věnoval při vedení této práce.

Obsah

1 Úvod

Tématem této bakalářské práce jsou formuláře pro získávání RDF dat. V první části práce seznámím čtenáře s oblastí RDF a OWL. Popíši nalezená existující řešení pro popis a sběr dat prostřednictvím automaticky generovaných formulářů a rozdíly mezi nimi. V další části analyzuji požadavky kladené na formuláře jako celek a na jejich jednotlivé vstupní prvky. Dále se budu věnovat návrhu a implementaci Java knihovny, která usnadní generování HTML formulářů z ontologií, zpracování odeslaných formulářů a uložení dat do RDF. V další části provedu návrh a implementaci ukázkové webové aplikace, která bude používat vytvořenou knihovnu a demonstrovat tak funkčnost a možnosti použití této knihovny. V poslední části této práce provedu diskuzi ohledně dosažených výsledků, možnosti rozšíření aplikace a porovnání s řešením jiných autorů.

2 RDF a OWL

2.1 Resource Description Framework

Resource Description Framework (dále jen RDF) [1] je standardní framework pro výměnu dat na webu. Poskytuje model pro popis zdrojů. Model umožňuje specifikovat trojice (zdroj, vlastnost, hodnota vlastnosti). Význam těchto trojic lze číst jako „Daný *zdroj* má danou *hodnotu* dané *vlastnosti*“. Jedna trojice tvoří tvrzení. V tvrzení je zdroj nazýván *subjektem*, vlastnost *predikátem* a hodnota *objektem*. Hodnotou vlastnosti může být literál a nebo identifikátor jiného zdroje [2]. V případě literálu lze hodnotě definovat datový typ. O jaký datový typ se jedná je možné vybrat například ze slovníku XML Schema datatypes [3]. Každá trojice [4]:

- Je tvořena subjektem, predikátem a objektem.
- Je kompletní a unikátní fakt.
- Může být spojena s dalšími trojicemi, ale přesto si zachovává svůj vlastní jedinečný význam.

Trojice je možné zapsat i pomocí grafu s orientovanými hranami, podrobněji viz sekce 2.5.

RDF je primárně určen pro situace, kde je informace zpracována strojově, ale je čitelný i lidsky. Používá se například pro výměnu informací mezi aplikacemi aniž by došlo ke ztrátě nebo zkreslení informace. Výhoda tohoto řešení je, že informace pak může být použita i v aplikacích pro které nebyla primárně určena.

2.2 RDF Schema

RDF Schema (dále jen RDFS) [5] rozšiřuje slovník RDF. Umožňuje tak lépe specifikovat vlastnosti jednotlivých RDF objektů.

RDFS oproti RDF umožňuje jednotlivé zdroje rozdělovat do skupin neboli tříd. Zdroje, které patří do třídy jsou označovány jako instance třídy. Základní třídy v RDFS jsou:

- *rdfs:Resource* – Všechny vytvořené třídy jsou vždy podtřídou této třídy. Je to třída, která obsahuje vše.

- *rdfs:Class* – Třída zdrojů, které jsou také třídy.
- *rdfs:Literal* – Třída literálů (literál je používán pro hodnoty typu: řetězec, číslo, datum, atd.)
- *rdfs:Datatype* – Třída datových typů.
- *rdf:Property* – Třída vlastností.

Pomocí RDFS můžeme těmto zdrojům určovat různé vlastnosti, například, omezení hodnot jakých může zdroj nabývat (*rdfs:range*) a nebo můžeme zdroji doplnit anotaci srozumitelnou pro člověka jako je *rdfs:label*.

2.3 W3C Web Ontology Language

Web Ontology Language (dále jen OWL) [6] je jazyk sémantického webu navržený skupinou W3C a je součástí Semantic Web technology stack. Slouží k reprezentaci bohatých a komplexních znalostí o věcech, skupinách věcí a vztahy mezi nimi. Oproti RDFS dokáže všechny tyto vztahy popisovat více komplexně.

Rozšiřuje RDFS o omezení kardinality například „Osoba může mít maximálně jednu adresu trvalého bydliště.“. Dále také o pravdivostní funkce jako je konjunkce, disjunkce či ekvivalence například „Osoba a Adresa jsou vzájemně disjunktní. Zdroj nemůže být Osoba a Adresa zároveň.“. A také o restrikce například „Na Osobu aplikujeme vlastnost *má rodiče*, ta pak musí být spojena s alespoň jednou instancí třídy Osoba.“[7].

OWL dokumenty nebo také ontologie mohou být umístěny na web a mohou odkazovat na nebo být odkazovány z jiných ontologií.

Poskytuje tři různé podjazyky rozdělené podle způsobu použití koncovým uživatelem:

- *OWL Full* – Používá všechny dostupné konstrukce jazyka OWL bez omezení. Výhodou je vysoká vyjadřovací schopnost. Nevýhodou je pak vysoká složitost zpracování jazyka.
- *OWL DL* – Jedná se o kompromis mezi vyjadřovací silou a složitostí zpracování jazyka.
- *OWL Lite* – Nejjednodušší verze jazyka OWL, to znamená, že je snazší na zpracování, ale také má nejmenší vyjadřovací schopnost.

2.4 Prvky ontologie

Obsah ontologie lze rozdělit na tři základní prvky. Na Individua (*Individuals*), Třídy (*Classes*) a na Vlastnosti (*Properties*).

Individuál je konstanta nebo objekt. Reprezentují reálné objekty, které již v rámci ontologie nebudou dále členěny.

Koncept ontologických tříd je podobný konceptu tříd v objektově orientovaném programování. Objekty reálného světa mohou být rozděleny do skupin objektů s podobnými charakteristikami [8]. Třída seskupuje právě taková individua, která mají něco společného. V ontologiích je možné definovat nadtřídy a podtřídy. Podtřída specializuje (rozšiřuje) svou nadtřídu. Všechny instance podtříd jsou zároveň instancemi své nadtřídy. OWL obsahuje dvě předdefinované třídy, *owl:Thing* a *owl:Nothing*. Třída *owl:Thing* obsahuje všechna vytvořená individua. Všechny vytvořené třídy jsou její podtřídou. Třída *owl:Nothing* je prázdná třída, každá vytvořená třída má jako podtřídu právě *owl:Nothing* [9].

OWL rozlišuje mezi dvěmi hlavními kategoriemi vlastností. Jedná se o vlastnosti objektů, vlastnosti datových typů. Pro podjazyk OWL DL jsou důležité ještě anotační vlastnosti a ontologické vlastnosti [10][11].

- Vlastnosti objektů (*Object Properties*) – Popisují vztahy mezi dvěmi individui.
- Vlastnosti datových typů (*Datatype Properties*) – Popisují vztahy mezi individui a datovou hodnotou. Datová hodnota může být vymezena obecně (například hodnoty ze slovníku XML Schema datatypes) a nebo konkrétně.
- Anotační vlastnosti (*Anotation Properties*) – Anotační vlastnosti se používají k přidání informací ke třídám, individuím a objektovým vlastnostem nebo vlastnostem s datovým typem. OWL obsahuje pět předdefinovaných anotačních vlastností. Jedna z nich je například *rdfs:comment*, která slouží k uložení komentáře.
- Ontologické vlastnosti (*Ontology Properties*) – Vlastnosti, které popisují samotnou ontologii. Ontologickou vlastností je například vlastnost *owl:backwardCompatibleWith*, která udává s jakou předchozí verzí je ontologie zpětně kompatibilní.

2.5 Příklad uložených RDF dat

V následujícím příkladu ukáži data uložená ve formátu RDF/XML. Pro uložení je však možné použít i další formáty jako jsou N-Triples [12], N-Quads [13] a další.

V příkladu vytvořím instanci osoby (subjekt) s vlastností datového typu určující jméno osoby (predikát) s hodnotou Filip (objekt). Osoba bude mít objektovou vlastnost adresa, která udává kde osoba bydlí. Protože se jedná o objektovou vlastnost je nutné vytvořit instanci adresy. Adresa bude mít dvě vlastnosti datového typu. První bude název města s hodnotou Plzeň, druhá bude poštovní směrovací číslo s hodnotou 30100. Tyto informace uložené ve formátu RDF/XML je možné vidět na výpisu 2.1.

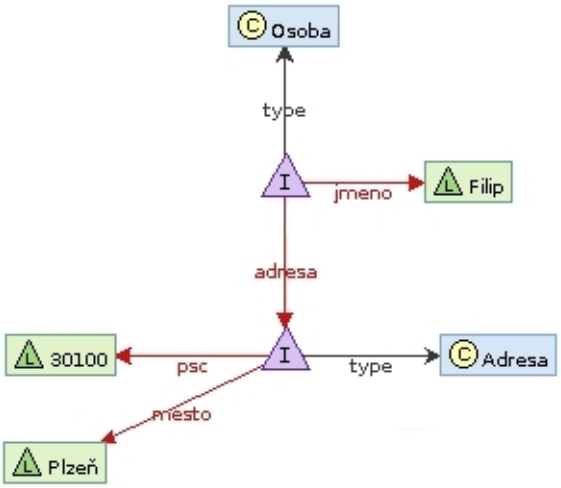
```
1  <rdf:RDF
2      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3      xmlns:ex="http://filek.cz/example.owl#">
4    <ex:Adresa rdf:about="http://filek.cz/adresa/5">
5          <ex:mesto>Plzen</ex:mesto>
6          <ex:psc>30100</ex:psc>
7    </ex:Adresa>
8    <ex:Osoba rdf:about="http://filek.cz/osoba/1">
9          <ex:jmeno>Filip</ex:jmeno>
10         <ex:adresa rdf:resource="http://filek.cz/adresa/5"/>
11   </ex:Osoba>
12 </rdf:RDF>
```

Výpis 2.1: Ukázka uložených dat ve formátu RDF/XML

Ve výpisu 2.1 na řádku číslo 4 vytvoříme instanci třídy *Adresa*, která má ID 5, *Adresa* obsahuje datový typ *město* a jeho hodnota je Plzeň. Také obsahuje datový typ *PSČ* s hodnotou 30100.

Na řádku číslo 8 vytvoříme instanci třídy *Osoba*, která má ID 1 a má *jméno*, které má hodnotu Filip. Na desátém řádku této osobě přiřadíme *adresu* jako odkaz na instanci výše vytvořené adresy s ID 5.

Data se dají zobrazit i jako graf viz obrázek 2.1.



Obrázek 2.1: Data zobrazená jako graf

Na grafu na obrázku 2.1 modré obdélníky značí třídy, fialové trojúhelníky značí instance třídy a zelený obdélník značí literál. Červená hrana značí relace, orientace hrany je směrována od subjektu k objektu.

Z grafu lze vidět, že instance třídy *Osoba* má jako adresu odkaz na instanci třídy *Adresa*.

3 Formuláře

V této kapitole se budu zabývat analýzou požadavků, které jsou kladeny na formuláře a jejich součásti, možností použití formulářů a zobrazováním a editací dat. Formuláře budou psané v HTML 5 [14].

3.1 Požadavky

Na formuláře jsou kladené nejrůznější požadavky ať už se jedná o správné zobrazení vstupního prvku, o zobrazování různých typů hodnot, omezení zadávaných hodnot nebo jejich následnou validaci.

V HTML5 byly představeny nové vstupní prvky usnadňující tvorbu formuláře v tom smyslu, že umožňují zadat pouze hodnotu z určitého rozsahu nebo v požadovaném formátu. Jsou to vstupní prvky typu číslo, email, datum, a další [15]. Bohužel prohlížeče podporují jenom část těchto prvků a to ještě tak nešťastně, že například první dva prohlížeče vstupní prvek dokáží zobrazit správně, ale třetí prohlížeč ne a nebo naopak. Například vstupní prvek datum z HTML 5 funguje na všech nejpoužívanějších prohlížečích kromě prohlížeče Mozilla Firefox [16]. Při vytváření formulářů je proto důležité testovat formuláře na více prohlížečích (alespoň na těch nejvíce používaných jako je Google Chrome, Safari, Mozilla Firefox, Internet Explorer a nebo Opera [17]). To samé platí i pro různá rozlišení monitoru. Pokud se vstupní prvky na nějakém prohlížeči nebo rozlišení nezobrazují správně je třeba implementovat tento prvek jiným způsobem. A to buď pro všechny prohlížeče nebo jen pro prohlížeč ve kterém se nezobrazuje správně

3.1.1 Typy vstupů HTML

V této podsekci se budu věnovat takovým vstupním prvkům, které bude možné vygenerovat z ontologií, které používají jako `rdfs:range` typ ze slovníku XML Schema datatypes [3]. Pro lepší čitelnost a přehlednost budu pro jmenný prostor `www.w3.org/2001/XMLSchema#` používat prefix `xsd`.

Textový vstup

Jeden ze základních prvků HTML formulářů je vstup pro text. Vstupní prvek typu text umožňuje zadání jakéhokoliv řetězce. V HTML je definován kódem `<input type="text">`.

Bude používán pro prvky, které mají pro `rdfs:range` uvedeno `xsd:string`. V prohlížeči pak vypadá jako na obrázku 3.1.

Obrázek 3.1: Vstupní prvek typu text

Tlačítko odeslat

Abychom mohli formulář odeslat, potřebujeme nějaké tlačítko. K tomu slouží prvek typu submit. HTML kód tohoto prvku je `<input type="submit">`.

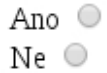
V prohlížeči se zobrazí jako na obrázku 3.2.

Obrázek 3.2: Odesílací tlačítko

Přepínač

Vstupní prvek přepínač dokáže zobrazit několik přepínačů s tím, že je možné vybrat jenom jednu možnost. V aplikaci bude používán pro prvky, které mají `rdfs:range` typu `xsd:boolean`. Aplikace bude generovat dva přepínače, s hodnotou `true` a `false`. HTML kód pro jeden přepínač je definován následovně `<input type="radio">`.

Ukázku můžeme vidět na obrázku 3.3.



Obrázek 3.3: Přepínač

Číslo

Vstupní prvek číslo umožňuje zadávání pouze číselných hodnot a to jak celých čísel tak i reálných. Jeho HTML kód je `<input type="number">` a v aplikaci bude používán pro prvky mající `rdfs:range` jeden z následujících typů:

- `xsd:integer`

- `xsd:double`
- `xsd:float`
- `xsd:nonNegativeInteger`
- `xsd:nonPositiveInteger`
- `xsd:negativeInteger`
- `xsd:positiveInteger`

Vstupnímu prvku je možné nastavit atribut `min` a `max`, čímž můžeme omezit hodnoty pro poslední čtyři výše zmíněné položky.

Například pro `nonNegativeInteger` (nezáporné celé číslo) nastavíme hodnotu atributu `min` na 0, atribut `max` nemusíme vyplňovat. Do vstupního prvku číslo však lze zadat i desetinné číslo a aplikace požaduje celé číslo. Řešením je použít ověření pomocí JavaScriptu a ověření odeslaných hodnot na serveru.

Vstupní prvek pak vypadá jako na obrázku 3.4.

Obrázek 3.4: Vstupní prvek typu číslo

Datum a Datum s časem

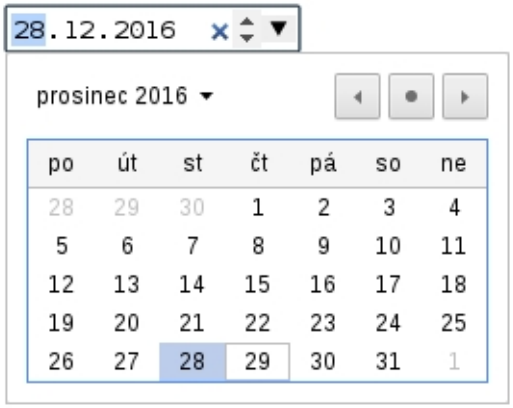
Prvek datum slouží k výběru data. V aplikaci bude používán pro prvky mající `rdfs:range` typu `xsd:date`. HTML kód je definován jako `<input type="date">` a jeho ukázku můžeme vidět na obrázku 3.5.

Prvek Datum s časem obsahuje navíc časovou hodnotu. Jeho HTML kód je `<input type="datetime-local">`. Bude používán pro prvky mající `rdfs:range` typu `xsd:dateTime`. Jeho ukázka je na obrázku 3.6.

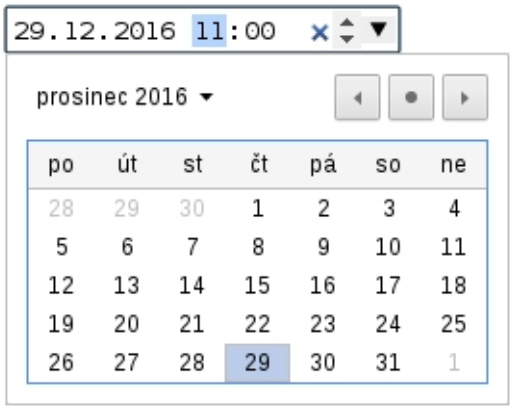
Tyto dva prvky nejsou podporované prohlížeči Mozilla Firefox.

Rozevírací seznam

Rozevírací seznam umožňuje výběr hodnot ze seznamu. V aplikaci bude používán pro prvky, které mají jako `rdfs:range` odkaz na instanci třídy. Ve formuláři potom budeme chtít vybrat hodnoty pouze z existujících instancí. HTML kód rozevíracího seznamu je definován párovým HTML tagem



Obrázek 3.5: Vstupní prvek typu datum



Obrázek 3.6: Vstupní prvek typu datum s časem

`<select>`. Jeho jednotlivé položky pak tagem `<option>`, který je také párový.

Ukázka seznamu je na obrázku 3.7

Obrázek 3.7: Rozevírací seznam

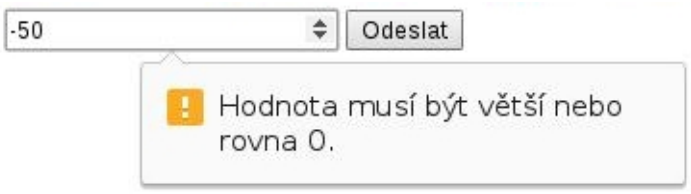
3.1.2 Ověření zadaných hodnot

Ověřování zadaných hodnot můžeme řešit jak na straně klienta tak i na serveru.

Ověřování na straně klienta

U HTML 5 prvků při odeslání formuláře HTML 5 zajišťuje ověření, že hodnota byla zadána správně, pokud nebyla, tak nepovolí odeslání formuláře a zobrazí chybovou hlášku.

Například pro vstupní prvek číslo s atributem `min="0"` nelze zadat záporné číslo. Pokud je zadáno záporné číslo a uživatel formulář odešle, zobrazí se mu chybová hláška (viz obrázek 3.8).



Obrázek 3.8: Chybová hláška při nesprávné hodnotě

Problém nastává tehdy, pokud nemá uživatel povolen JavaScript v prohlížeči, nebo prohlížeč nepodporuje daný vstupní prvek, pak se formulář odešle i s nesprávnou hodnotou. Proto je nutné všechny hodnoty ověřovat i na serveru.

Ověřování na straně serveru

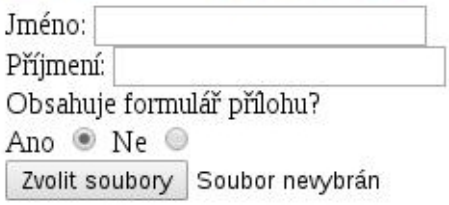
Všechny hodnoty je nutné ověřit na straně serveru i přesto, že se uživateli podařilo formulář odeslat. Mohlo by se stát, že data, která uživatel vyplnil mají jiný datový typ, než který je povolený a výstupní data by byla nevalidní. Například místo celého čísla bylo zadáno číslo reálné a podobně.

3.2 Podmíněné formuláře

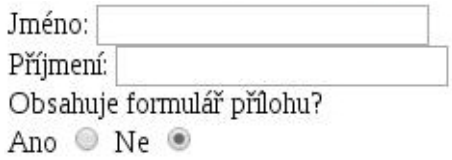
Podmíněné formuláře jsou takové formuláře, které dynamicky mění svůj obsah v závislosti na nějaké akci uživatele nebo podmínkové proměnné.

Například formulář, který má přepínač jestli obsahuje přílohu. Pokud uživatel vybere, že ano, zobrazí se mu ve formuláři nový vstupní prvek na výběr souboru. Pokud uživatel vybere, že ne, obsah zůstane stejný.

Tato funkčnost bývá zajištěna pomocí JavaScriptu. Ukázka na obrázku 3.9 a obrázku 3.10.



Obrázek 3.9: Uživatel vybral, že formulář obsahuje přílohu



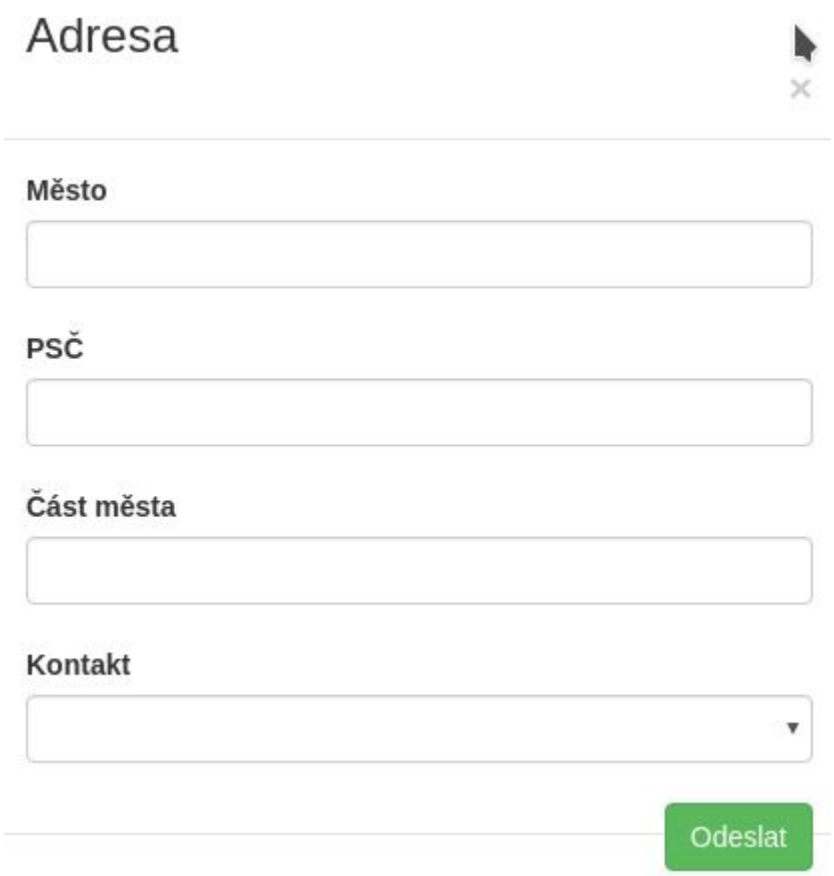
Obrázek 3.10: Formulář přílohu neobsahuje

3.3 Podformuláře

Podformulář lze využít například v situaci, kdy má uživatel hodnotu vybrat ze seznamu hodnot. Pokud se však hodnota v seznamu nenachází, aplikace nabídne možnost přidání nové položky do seznamu. Vygeneruje nový formulář, kde uživatel vyplní hodnoty a odešle ho. Tato nová hodnota se pak přidá do seznamu hodnot. Tato funkčnost bývá zajištěna pomocí JavaScriptu.

Pro ukázku na obrázku 3.11 vybere uživatel možnost přidat novou položku. Po vybrání této možnosti se mu zobrazí nový formulář viz obrázek 3.12, po odeslání tohoto formuláře se přidá položka do seznamu.

Obrázek 3.11: Uživatel vybral možnost „Přidat nový záznam“



Obrázek 3.12: Formulář vygenerovaný pro přidání nové položky „Adresa“ do seznamu

3.4 Existující řešení

Pro popis a sběr dat prostřednictvím automaticky generovaných formulářů jsem nalezl 5 existujících řešení pro webové formuláře a jedno řešení pro formuláře pro mobilní telefony s OS Android, každému z nich se budu podrobněji věnovat níže.

Z nalezených řešení podporuje ontologie pro generování formulářů pouze jedno z nich. Formuláře jsou místo toho generovány pomocí nástroje poskytnutého aplikací, nebo jsou popsány v různých formátech. To samé platí o exportu dat. Z nalezených řešení pouze jedno z nich dokáže ukládat data do formátu RDF.

3.4.1 Open Data Kit

Open Data Kit [18] je jedno z řešení pro sběr dat. Avšak ke sběru dat nepoužívá webové formuláře, ale mobilní telefony s operačním systémem Android.

Pro generování formulářů není možné použít ontologie. Formuláře jsou generovány buď ručně pomocí poskytnuté aplikace a nebo pomocí formuláře popsaného v Excel souboru.

Data jsou exportována buď do formátu KML nebo CSV, není možnost exportovat data do RDF formátu.

3.4.2 XForms

XForms [19] je XML aplikace sloužící k popisu formulářů ve formě XML. XForms jsou rozděleny podle architektury MVC (Model - View - Controller) nebo česky Model - Pohled - Kontroler. V Modelu je definovaný samotný formulář, jednotlivé vstupy a jejich definiční obory. V Pohledu se definuje jak se bude formulář zobrazovat koncovému uživateli. A nakonec Kontroler slouží k řízení operací mezi Modelem a Pohledem, manipulaci s daty a k ukládání dat.

XForms podporují podmíněné generování formulářů, validaci na straně klienta, validaci na straně serveru a také export dat do formátu XML.

Jediný problém u XForms je, že není nativně podporován žádným prohlížečem a je tedy nutné použít doplňky do prohlížeče nebo různé frameworky, které pomocí JavaScriptu zobrazí stránku uživateli v použitelné podobě.

3.4.3 ActiveRaUL

Pro generování formulářů pomocí ontologií byl k dispozici nástroj ActiveRaUL, který dokázal exportovat data do RDF. V průběhu psaní bakalářské práce však přestaly webové stránky aplikace fungovat [20]. Jediné co po aplikaci zůstalo je odborný článek [21].

Z článku se lze dočíst, že webová aplikace dokázala generovat formuláře z ontologií poskytnutých uživatelem. Tyto formuláře obsahovaly prvky různých typů (text, přepínač, rozevírací seznam, atd...). Data z formulářů se ukládala do RDF formátu. Aplikace měla stejnou funkčnost jako je téma této práce.

3.4.4 Google Forms

Google Forms [22] sice nativně nepodporují automatické generování formulářů, ale je možné formuláře generovat pomocí Google Apps Script [23].

Nevýhoda tohoto řešení je, že Google Apps Script nepodporuje generování pomocí ontologie, neobsahuje žádné metody pro práci s nimi a zatím ani neexistuje žádná knihovna, která by tuto práci umožňovala.

Práce s ontologiemi by tedy byla možná pouze pomocí funkce implementované přímo v Google Apps Script a to *XML Service*, která slouží ke získávání dat z XML souborů. Práce s ontologiemi by byla velice nepraktická.

Google Forms exportují data do formátu CSV, XLSX a nebo ODS. Převod do RDF by se tedy musel dělat pomocí externí aplikace.

3.4.5 Angular Schema Form

Angular Schema Form [24] je JavaScriptový framework, který umožňuje dynamicky generovat formuláře z JSON Schema pomocí AngularJS. Frameworku se předá JSON objekt, ve kterém se definují jednotlivé vstupní prvky formuláře. Vstupním prvkům lze nadefinovat různé atributy jako typ, ověřování hodnoty pomocí regulárního výrazu nebo text chybové hlášky v případě špatně zadané hodnoty. Framework pak z daného JSON objektu vygeneruje HTML formulář.

Formulář je možné vyplnit a odeslat běžným způsobem jako obyčejný HTML formulář a nebo způsobem, který doporučují vývojáři. Doporučený způsob je zpracovat formulář pomocí funkce z frameworku, která převede výstupní hodnoty formuláře do JSON objektu. Výhodou tohoto způsobu je také možnost validace hodnot formuláře.

Pro použití tohoto řešení by bylo nutné naprogramovat metodu, která generuje JSON objekt pro vybrané prvky z ontologie. To samé platí i pro převod získaných dat do formátu RDF.

3.4.6 xsd-forms

Knihovna xsd-forms [25] slouží pro generování webových formulářů popsaných v XML formátu, přesněji XML Schema (XSD). Podobně jako u Angular Schema Form knihovna generuje JavaScriptový kód, který dokáže validovat odeslané formuláře.

Knihovna nepodporuje generování formulářů z ontologií a ani ukládání dat do formátu RDF. Data jsou ukládána ve formátu XML. Bylo by nutné vytvořit metodu pro převod vstupních prvků ontologie do formátu XSD a výstupních dat do formátu RDF.

4 Návrh Java knihovny

Pro práci s ontologiemi v jazyce Java jsem našel dva frameworky. První z nich je The OWL API [26]. Druhý je framework Apache Jena [27].

Tyto frameworky jsou však velice obsáhlé, primárně slouží pro práci s ontologiemi a výstupními RDF daty, ale nedokáží generovat formuláře z ontologií. Cílem tedy bylo navrhnout Java knihovnu používající jeden z těchto frameworků, která umožní snažší práci s ontologiemi, respektive snažší generování HTML formulářů z ontologií.

4.1 Požadavky na funkčnost knihovny

Nová knihovna by měla fungovat co nejjednodušeji, a to jak pro generování formulářů, editování výstupních dat tak i jejich ukládání.

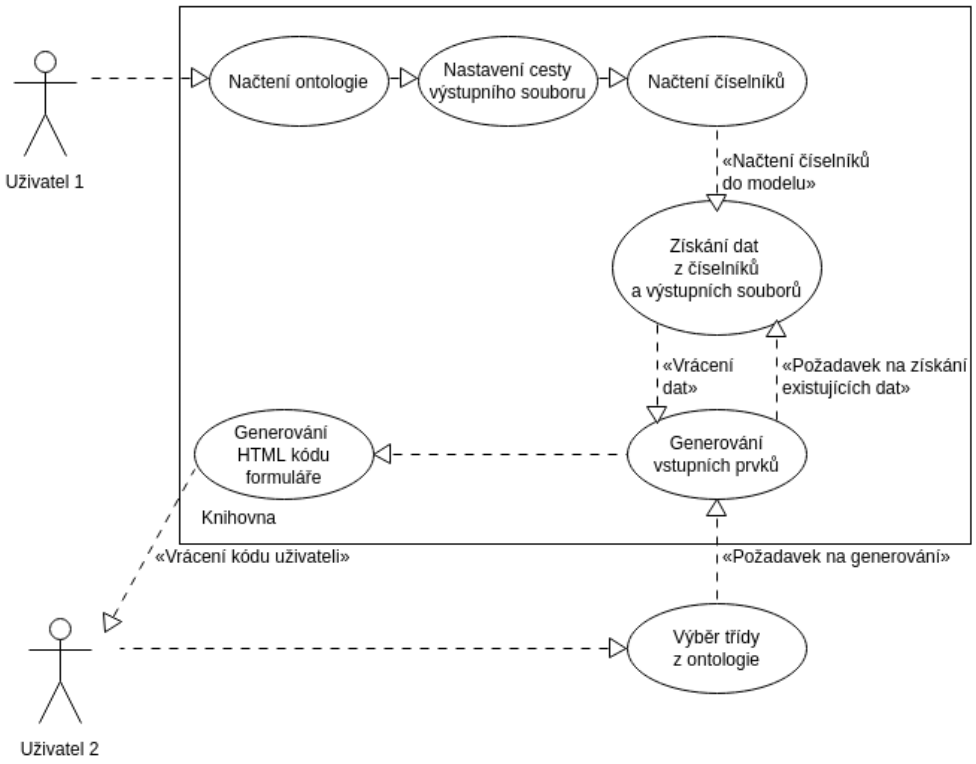
Pokud by uživatel použil pouze Apache Jena, proto aby získal HTML kód formuláře pro danou ontologii a její třídu, musel by si naprogramovat metody, které data získají. Data jsou pak v různých formátech (instance objektů, např.: OntClass [28], Statement [29], apod. . .) a je třeba naprogramovat převod dat do výstupního HTML formátu. Celá funkčnost je zajištěna stovkami řádek kódu a právě proto přichází na řadu nová knihovna, která vygeneruje celý formulář za pomocí pár řádek kódu.

4.1.1 Generování formulářů

Na generování formulářů jsou kladené následující požadavky:

1. Generování formulářů s prvky různých typů (text, číslo, . . .).
2. Generování formulářů s prvky pro které je možné vybrat hodnoty z existujících dat.
3. Možnost upravení šablony výsledného HTML kódu formuláře i jeho jednotlivých vstupních prvků.

Pokud bude uživatel chtít vygenerovat formulář, tak zadá cestu k ontologii, zadá cestu kam uložit soubor z výstupními daty z formuláře, vybere třídu ontologie pro kterou chce formulář vygenerovat a knihovna formulář ve formátu HTML vygeneruje. Data jednotlivých položek formuláře by měla knihovna načítat jak z výstupního souboru tak z číselníků.



Obrázek 4.1: Diagram ukazující postup generování formuláře

Například pokud by uživatel chtěl vygenerovat formulář pro přidání nové osoby. Tento formulář obsahuje položku adresa. Ve výstupním souboru je již několik instancí adresy. Knihovna vygeneruje seznam těchto adres, které je pak možné ve formuláři vybrat. Pro lepší představu je možné tento postup vidět na obrázku 4.1.

Formuláře by měly být generovány s ohledem na snadné získání dat po jejich vyplnění a odeslání. Tato data by mělo být možné uložit. Více v sekci 4.1.2.

Dále by knihovna měla podporovat šablony pro výstup těchto formulářů. To znamená, že uživatel si bude moci vytvořit šablonu jak by měl výstupní formulář vypadat a podle toho se formulář vygeneruje.

Šablony by mělo být možné vytvářet jak pro celý formulář, tak pro všechny prvky formuláře. To znamená, že každý typ vstupního prvku formuláře by měl mít svou vlastní šablonu.

4.1.2 Editování dat

Knihovna by měla podporovat editaci již vytvořených dat. Měla by poskytnout uživateli možnost:

1. Získání uložených dat a jejich hodnot.
2. Tyto hodnoty by mělo jít dále upravovat, případně doplňovat o nové informace. Například doplnit údaje, které nebyly při prvním odeslání formuláře vyplněné.

4.1.3 Ukládání dat

Požadavky na ukládání dat jsou následující:

1. Ukládání dat z odeslaného formuláře.
2. Ukládání dat do různých výstupních formátů.

Knihovna by měla poskytnout možnost jak uložit data z odeslaného formuláře. To znamená, že uživatel po odeslání formuláře hodnoty zkontroluje, převede je do formátu, který bude knihovna podporovat a pak už jen zavolá metodu na zpracování těchto dat. Převod do podporovaného formátu by neměl být zbytečně složitý.

Data pak budou ukládána primárně ve formátu RDF/XML. Ale protože Apache Jena podporuje i další formáty bude možné data uložit ve formátech Turtle a N-Triples. V jakém formátu data uložit si bude moci zvolit uživatel.

4.2 Výběr frameworku

V následující sekci se budu zabývat funkčností výše zmíněných frameworků pro práci s ontologiemi a výběrem vhodného frameworku pro knihovnu.

4.2.1 The OWL API

Jedná se o framework v Javě pro vytváření, upravování a serializaci ontologií. Je to knihovna určená pro efektivní práci s ontologickými modely uloženými v paměti.

Dokáže načítat a zapisovat do formátů RDF/XML, OWL/XML, Turtle. Dále pak dokáže pouze načítat soubory formátu KRSS a OBO Flat File. Jelikož jsem měl k dispozici data k testování i ve formátu N-Triples, bylo by nutné nejdříve převést data do jednoho z podporovaných formátů. Ve své knihovně bych tak musel použít knihovnu na převod do podporovaného

formátu a nebo to nechat na koncovém uživateli. Toto byl jeden z důvodů proč jsem tuto knihovnu nepoužil.

Další důvod proč jsem knihovnu nepoužil je dle mého osobního názoru nepřehledná dokumentace. Dokumentace je nepřehledně rozmístěna na několika různých místech, například oficiální odkaz na úvod do OWL API („The Rough Guide to the OWL API“ [30]) je ve formě powerpointové prezentace, kde jsou slajdy z přednášky, která byla přednášena na Univerzitě v Manchesteru a chybí k nim tedy vysvětlení.

Poslední důvod je ten, že při hledání řešení nějakého problému je spousta problémů řešena pouze v knihovně Apache Jena.

4.2.2 Apache Jena

Podobně jako The OWL API je to framework napsaný v Javě. Poskytuje knihovny, které umožňují práci s RDF, RDFS, RDFa, OWL a SPARQL.

Slouží pro práci nejen s modely uloženými v paměti, ale i pro práci s modely uloženými v souborech. Umožňuje použití SPARQL což je dotazovací jazyk pro data uchovaná ve formátu RDF. Potom je možné se na data dotazovat vzdáleně pomocí HTTP protokolu, například pomocí Apache Fuseki [31]. Kromě těchto zmíněných možností podporuje připojení i k dalším typům úložišť dat, více v dokumentaci Apache Jena [32].

Dokáže pracovat s RDF daty ve formátech RDF/XML, Turtle, N-Triples, N-Quads a dalších [33]. Pro testovací data, která jsem měl k dispozici není nutné převádění do podporovaných formátů.

Dokumentace je oproti The OWL API přehledněji zpracovaná a je umístěna na jednom místě. Proto je hledání informací rychlejší a jednodušší. Kvůli většímu rozšíření mezi uživateli je hledání řešení problémů oproti The OWL API jednodušší.

Pro svou knihovnu jsem se rozhodl použít framework Apache Jena i kvůli tomu, že je tento framework používán na katedře a v projektech MRE a byl mi doporučen vedoucím práce. Kromě výše zmíněných důvodů byl jeden z důvodů ten, že jsem s Apache Jena již v minulosti pracoval a měl jsem tedy představu o tom jak framework funguje.

4.3 Překlad knihovny

Knihovna by měla být snadno přeložitelná, je psaná v jazyce Java a požadavkem od vedoucího práce bylo použití nástroje Apache Maven [34]. Maven je nástroj pro správu projektů a jejich snadné překládání. K překladu jazyku

Java však potřebuje ještě Java překladač. Aplikace by tedy měla být přeložitelná pomocí Java překladače ve verzi alespoň 1.8.

5 Implementace knihovny

V této kapitole se budu podrobněji věnovat implementaci knihovny.

Nejdříve vysvětlím jaké třídy a k čemu jsou v knihovně implementovány. Dále potom jak vypadají šablony a jak vytvářet vlastní šablony pro jednotlivé vstupní prvky formuláře. A nakonec ukážu jak knihovnu přeložit a jak jednoduše vygenerovat formulář pro danou třídu z ontologie.

5.1 Třídy

V této sekci jsou popsány třídy z knihovny, které jsou důležité pro koncového uživatele. U každé třídy je popis k čemu slouží a jak se používá.

5.1.1 GFormOntology

Zajišťuje veškerou funkčnost pro práci s ontologiemi. Mezi tuto funkčnost patří načítání ontologií do modelu [35], získávání informací z modelu, například objektové vlastnosti, vlastnosti datových typů, seznam tříd v ontologii a generování vstupních prvků formuláře pro danou třídu.

Slouží také jako prostředník pro práci s uloženými daty, případně k jejich ukládání.

Načítání ontologií

První ontologie je do modelu načtena automaticky při vytváření instance třídy `GFormOntology`. Její konstruktor viz výpis 5.1, přijímá jako parametr řetězec s cestou k ontologii a parametr typu `OntModelSpec` [36], který udává o jaký typ modelu se jedná. Cesta k ontologii může vést jak k lokálnímu souboru na PC tak ke vzdálenému souboru pomocí URL adresy.

```
1 public GFormOntology(String URL, OntModelSpec ontModelSpec)
```

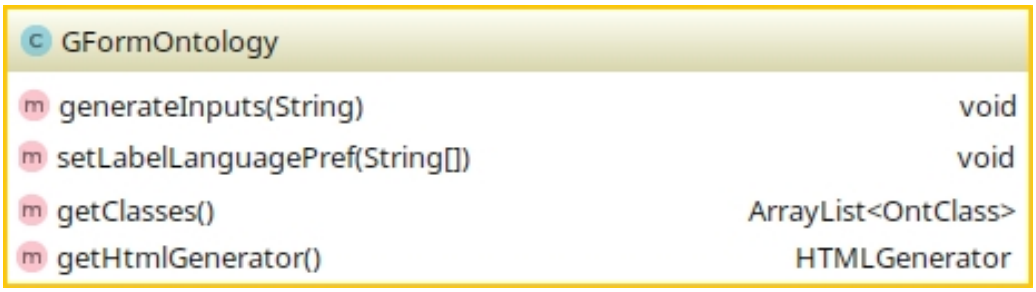
Výpis 5.1: Konstruktor třídy GFormOntology

Pokud chce uživatel do modelu načíst další ontologii je možné použít metodu `addToModel(String URL)`, která jako parametr přijímá řetězec s cestou k ontologii.

Generování seznamu vstupních prvků

Po vytvoření instance třídy `GFormOntology` a po přidání ontologie do modelu je možné vygenerovat první formulář.

V této sekci budou popsány metody, pomocí kterých lze vygenerování formuláře z ontologie dosáhnout. Jedná se o čtyři metody, které je možné vidět na diagramu na obrázku 5.1.



Obrázek 5.1: Základní metody dostupné uživateli pro vygenerování formuláře

Aby bylo možné vygenerovat HTML kód formuláře, je nutné knihovně říci pro jakou třídu z ontologie má kód vygenerovat. Pro získání seznamu tříd, které se v ontologickém modelu nacházejí je v knihovně implementována metoda `getClasses()`.

Vstupní prvky mají popisky. Jelikož mohou být popisky vlastností v ontologickém modelu v různých světových jazycích, může si uživatel nastavit preferenci jednotlivých jazyků. Pro nastavení preference slouží metoda `setLabelLanguagePref()`, její vstupní parametr je seřazené pole řetězců se zkratkami světových jazyků. Ukázka seřazeného pole je na výpisu 5.2. Takto seřazené pole říká knihovně, aby při generování popisků jednotlivých prvků používala primárně český jazyk („cs“). Pokud však nenalezne žádný popisek v českém jazyce, může použít jazyk anglický („en“). Jestliže nenalezne ani tento popisek, může použít popisek u kterého není definován atribut značící v jakém jazyce je (prázdný řetězec).

```
1  setLabelLanguagePref(new String[]{"cs", "en", ""});
```

Výpis 5.2: Ukázka pole preference jazyků

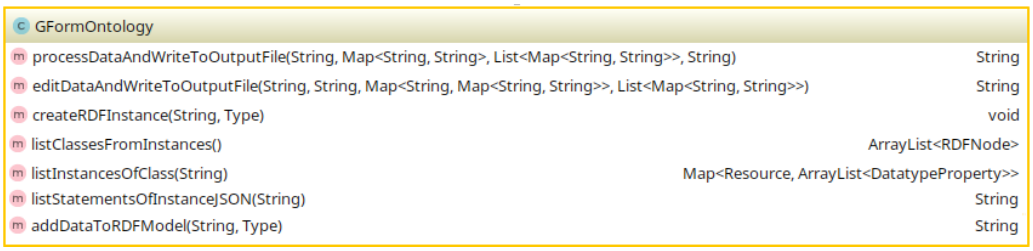
Když má uživatel k dispozici seznam tříd a nastavil si preferenci jazyků, může se pustit do generování vstupních prvků pro vybranou třídu. K tomu je připravena metoda `generateInputs()`, která má jako vstupní

parametr řetězec s URI adresou třídy ontologie. Tato metoda vygeneruje vstupní prvky formuláře a předá je instanci třídy `HTMLGenerator`, která dále generuje HTML kód formuláře.

Aby uživatel získal HTML kód formuláře zbývá už jen zavolat metodu `getHtmlGenerator().generate()`, která vrátí vygenerovaný kód ve formě řetězce. Třída `HTMLGenerator` má několik nastavovacích metod, pomocí kterých lze nastavit formát výstupu vygenerovaného kódu. Podrobněji jsou tyto metody popsány v sekci 5.1.4.

Ukládání a načítání dat

V této sekci budou podrobněji popsány metody dostupné uživateli pro ukládání, případně editaci existujících dat. Metody a jejich vstupní parametry je možné vidět na diagramu na obrázku 5.2.



Obrázek 5.2: Základní metody pro editaci dat

Aby bylo používání knihovny snadné pro koncového uživatele, byla navržena tak, aby uživatel pracoval pouze s jednou třídou. Pro větší přehlednost zdrojového kódu je však aplikace rozdělena na více tříd a tato třída slouží jako prostředník k ukládání a načítání dat z formulářů. Poskytuje přístup k metodám ze třídy `GFormData`, které tuto funkčnost zajišťují.

Instance třídy `GFormData` není vytvářena automaticky a je možné ji vytvořit pomocí metody `createRDFInstance()`. Ta má jako vstupní parametry řetězec s cestou k souboru kam se budou ukládat výstupní data z formulářů a druhý parametr je typu `GFormData.Type` což je `enum`, který určuje formát v jakém se budou data ukládat. Možné formáty jsou RDF/XML, N-Triple a Turtle.

Po vytvoření instance třídy `GFormData` je možné přidat existující data do výstupního modelu. Pro načtení již existujících dat je implementována metoda `addDataToRDFModel()`. Tato metoda má stejné parametry jako metoda `createRDFInstance()` popsaná výše.

Po vytvoření instance `GFormData`, případně po přidání dat do výstupního modelu je možné začít ukládat data z odeslaných formulářů. K tomu je připravena metoda `processDataAndWriteToOutputFile()`, která však pouze volá stejnojmennou metodu ze třídy `GFormData`. Detailnější popis této metody je v sekci 5.1.2.

K načítání dat pak slouží tři metody. První z nich slouží k načtení tříd z výstupního souboru, které zde mají alespoň jednu instanci. Jedná se o metodu `listClassesFromInstances()`. Druhá metoda slouží k načtení seznamu instancí dané třídy, jedná se o metodu `listInstancesOfClass()`. Poslední je potom metoda `listStatementsOfInstanceJSON()`, která načítá data z jednotlivých instancí a výsledek vrací ve formě JSON objektu. Tyto tři metody jsou ze třídy `GFormData` a detailněji jsou popsány v sekci 5.1.2.

Pro editaci dat se používá metoda `editDataAndWriteToOutputFile()`, která je popsána v sekci 5.1.2.

5.1.2 GFormData

Třída pro práci s výstupními soubory. Poskytuje metody pro načítání výstupních dat do modelu [37], ukládání dat do souboru ve třech formátech (RDF/XML, N-Triple, Turtle), zpracování dat z formuláře a metody, které se používají pro editování dat.

Jedná se o metodu vracející seznam tříd, které mají ve výstupním souboru alespoň jednu instanci. Potom o metodu vracející seznam instancí dané třídy. A jako poslední metoda, která vrací vlastnosti dané instance.

Načítání dat

První výstupní soubor je do modelu načten při vytváření instance třídy `GFormData` viz výpis 5.3. Pokud soubor neexistuje, tak je vytvořen nový prázdný soubor.

```
1 public GFormData(File dataFile, Type type, GFormOntology
    ontology)
```

Výpis 5.3: Konstruktor třídy RDF

První parametr `dataFile` je výstupní soubor, kam se budou data ukládat a ze kterého se budou načítat zpátky do modelu. Druhý parametr je `type` a udává v jakém formátu jsou data v souboru uložena. Poslední parametr je instance třídy `GFormOntology`, která je potřeba například pro získání prefixů jednotlivých ontologií.

Dále je možné ručně načíst data z dalšího souboru do modelu. K tomu slouží metoda `addDataToModel()`, do parametru metody se zadá cesta k souboru ve formě řetězce a formát v jakém je soubor uložen. Data jsou po zavolání metody načtena do modelu pomocí metody `read()`, kterou poskytuje Apache Jena.

Zpracování dat z formuláře

Pro zpracování dat z formuláře a jejich uložení do výstupního souboru, slouží metoda viz výpis 5.4.

```
1 String processDataAndWriteToOutputFile(String classURI,
2         Map<String, String> responseMap,
3         List<Map<String, String>> filesList, String rdfAboutURI)
             { ... }
```

Výpis 5.4: Metoda pro zpracování dat z formuláře

Tato metoda obsahuje čtyři parametry. První z nich je `classURI`, který určuje o jakou instanci třídy z ontologie se jedná.

Druhý parametr `responseMap` je mapa obsahující dvojici řetězců. První řetězec je URI dané vlastnosti z ontologie a druhý řetězec je hodnota této vlastnosti. Například instance třídy Adresa obsahuje vlastnost datového typu určující poštovní směrovací číslo. Do mapy se tyto hodnoty zadají jako na výpisu 5.5 na řádku 2, ukázka jak se data uloží do výstupního souboru je na výpisu 5.6.

```
1 Map<String, String> map = new HashMap<>();
2 map.put("http://filek.cz/e.owl#psc", "30100");
3 ont.processDataAndWriteToOutputFile("http://filek.cz/e.owl#
     Adresa", map, null, "http://filek.cz/adr/5");
```

Výpis 5.5: Ukázka mapy pro poštovní směrovací číslo

Třetí parametr `filesList` je seznam obsahující mapu dvou řetězců. Do této mapy se vkládají informace o souborech nahraných z formulářů. Pokud formulář neobsahuje žádný soubor, předá se metodě nulový parametr. Mapa obsahuje celkem čtyři klíče a jejich hodnoty. Klíč `classURI` značí třídu z ontologie pro kterou byl soubor nahrán. Dále je zde klíč `fileName` a jeho hodnota je název souboru. Další klíč je `fileSize`, který obsahuje velikost souboru v bajtech. Poslední klíč je `fileModified` a určuje datum a čas kdy

Obrázek 5.3: Ukázka seznamu map pro dva soubory

byl soubor naposledy modifikován. Pro lepší představu jak takový seznam map vypadá je na obrázku 5.3 ukázán seznam map pro dva soubory.

Čtvrtý a poslední parametr metody je `rdfAboutURI` což je jednoznačný identifikátor instance dané třídy, na výpisu 5.6 je na řádku 3.

Metoda má návratovou hodnotu typu String. Tento řetězec nabývá hodnoty „OK“ pokud při zpracování nedošlo k žádné chybě. Pokud k chybě dojde vrací text chybové hlášky.

```
1 <rdf:RDF
2         xmlns:e="http://filek.cz/e.owl#">
3         <e:Adresa rdf:about="http://filek.cz/adr/5">
4                 <e:psc>30100</e:psc>
5         </e:Adresa>
6 </rdf:RDF>
```

Výpis 5.6: Ukázka výstupních dat

Editace dat

Aby byl uživatel schopen upravit existující data, potřebuje získat z uložených dat seznam tříd, které mají v uložených datech alespoň jednu instanci. K tomu slouží metoda `listClassesFromInstances()`.

Tato metoda vrací uživateli seznam prvků typu `RDFNode` [38]. Tento seznam tříd pak lze dále použít k získání instancí daných tříd.

Metoda pro získání instancí třídy `listInstancesOfClass()`, má jako vstupní parametr URI adresu třídy. Tu je možné zjistit právě z předcho-

zího seznamu tříd. Tato metoda vrací mapu obsahující dvojici `Resource` [39] a seznam `DatatypeProperty` [40]. První z dvojice obsahuje informace o instanci, například URI adresu. A v seznamu se pak nacházejí všechny vlastnosti, které byly u této instance vyplněny.

Tyto dvě metody by stačily pro vygenerování formuláře pro editaci dat. Avšak pro jednodušší zpracování dat pomocí JavaScriptu je zde ještě jedna metoda. Jedná se o metodu `listStatementsOfInstanceJSON()`. Tato metoda má jako vstupní parametr URI adresu instance. Pro tuto instanci pak vygeneruje JSON objekt, který obsahuje seznam čtveřic. Pro každou vlastnost dané instance je jedna čtveřice, která obsahuje: Subjekt, Predikát, Objekt a Label. Subjekt je URI adresa instance. Predikát je URI adresa vlastnosti z ontologie pro kterou byla tato hodnota uložena. Objekt je hodnota výrazu. A Label je hodnota `rdfs:label` nebo `dc:title` dané vlastnosti.

Ukázku JSON objektu, pro data z výpisu 5.6 je možné vidět na výpisu 5.7.

```
1 [{"Subject":"http://filek.cz/adr/5","Predicate":"http://filek.
    cz/e.owl#Adresa", "Object":"30100", "Label":"Adresa"}]
```

Výpis 5.7: Ukázka JSON objektu

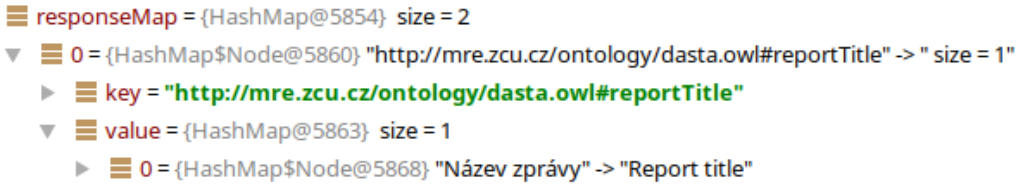
Po odeslání formuláře s editovanými hodnotami dat je nutné data zpracovat a změny uložit. K tomu se používá metoda ukázaná na výpisu 5.8.

```
1 String editDataAndWriteToOutputFile(String classURI, String
    rdfAboutURI, Map<String, Map<String, String>> responseMap,
2       List<Map<String,String>> filesList)
```

Výpis 5.8: Metoda pro editaci existujících dat

Parametry `classURI`, `rdfAboutURI` a `filesList` jsou stejné jako u metody pro zpracování dat z formuláře popsané výše.

U této metody nastala změna parametru `responseMap`. Protože některé instance tříd mohou v uložených datech obsahovat několik stejných vlastností s různou hodnotou, je třeba upravit správnou vlastnost. Mapa byla upravena a obsahuje řetězec a další mapu dvou řetězců. První řetězec určuje URI adresu vlastnosti, která se má upravit. V mapě řetězců se pak nachází klíč obsahující upravovanou hodnotu a hodnota klíče je nová upravená hodnota. Ukázka mapy je na obrázku 5.4. Na obrázku 5.4 je vidět, že se upravuje vlastnost `#reportTitle` s hodnotou „Název zprávy“ na novou hodnotu „Report title“.



Obrázek 5.4: Ukázka mapy pro úpravu dat

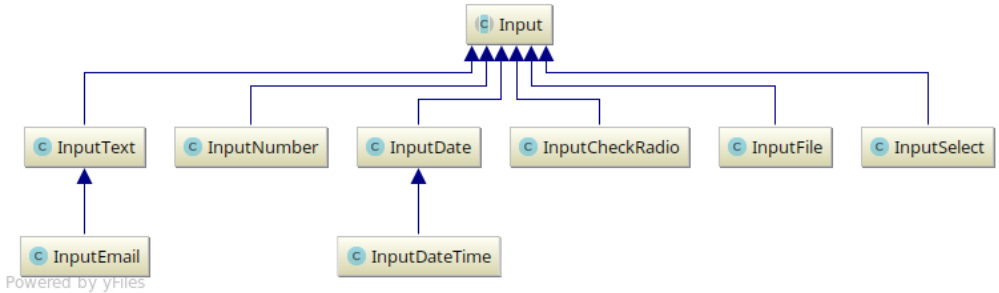
5.1.3 Input

Abstraktní třída, kterou dále rozšiřují jednotlivé typy vstupních prvků formuláře. Zajišťuje to, že každý vstupní prvek formuláře bude mít nastavenou vlastní šablonu a některé důležité HTML atributy, které mají všechny prvky společné. Jedná se o atributy jako HTML ID prvku, HTML jméno prvku, CSS třída prvku a podobně.

Dále obsahuje statickou metodu `getType()`, která jako parametr přijímá URI adresu `rdfs:range` vlastnosti z ontologie. Z této URI adresy určí o jaký typ vstupního prvku se jedná. Jaký prvek podle jaké URI se vygeneruje je možné vidět v sekci 3.1.1.

Tato třída však není pro uživatele důležitá, jelikož se používá pouze ve vnitřní implementaci třídy `GFormOntology`.

Prvky které tuto třídu rozšiřují je možné vidět na obrázku 5.5.



Obrázek 5.5: Diagram tříd ukazující prvky rozšiřující třídu `Input`

5.1.4 HTMLGenerator

Třída, která z poskytnutých šablon a ze seznamu prvků typu `Input` generuje výsledný HTML kód.

Uživatel má možnost pomocí metod z této třídy nastavit CSS třídu jednotlivých vstupních prvků a také šablonu těchto prvků. Také má možnost vygenerovat si HTML kód celého formuláře nebo pouze jeho vstupních prvků.

Instanci třídy je možné získat zavoláním metody `getHtmlGenerator()` ze třídy `GFormOntology`.

Nastavení prvků

Dříve než bude uživatel generovat formulář, měl by nastavit vlastnosti vstupních prvků.

Pro upravení CSS třídy prvku jsou zde nastavovací metody. Obecný tvar těchto metod je `set<type>Class()`, kde `<type>` je typ vstupního prvku. Například `setFileClass()`, je metoda která nastaví třídu prvku typu `file`. Vstupní parametr této metody je řetězec s názvem třídy.

Pokud uživatel nechce nastavovat CSS třídu každému prvku zvlášť, může použít metodu `setInputsClass()`, její vstupní parametr je také řetězec obsahující název třídy. Tato metoda pak nastaví všem vstupním prvkům stejnou třídu.

Dále si může uživatel upravit šablony jednotlivých vstupních prvků, tedy jak se bude generovat HTML kód daného prvku. Šablony je možné nastavit pomocí metod `set<type>TemplatePath()`, kde `<type>` je typ vstupního prvku. Například `setFileTemplatePath()`, jak název metody napovídá, nastaví cestu k šabloně pro prvek typu `file`. Parametr metody je řetězec s cestou k šabloně.

Pokud uživatel tyto parametry nenastaví, použijí se výchozí parametry nastavené v knihovně. Jak výchozí šablony vypadají, případně jak vytvářet vlastní je popsáno v sekci Šablony 5.2.

Generování formuláře

Po nastavení vlastností vstupních prvků je možné vygenerovat formulář. Pro generování celého formuláře je zde metoda `generate()`, která vrací řetězec s vygenerovaným HTML kódem formuláře. Dále pak metoda `generateOnlyInputs()`, která vygeneruje HTML kód pouze vstupních prvků. Tato metoda se dá použít například při generování obsahu stránky pomocí JavaScriptu. Uživatel získá formulář pomocí metody `getElementById()` a potom už jen upravuje hodnotu `innerHTML` tohoto formuláře.

Ukázka kódu jak se dá vygenerovat formulář je na výpisu 5.14.

5.1.5 TemplateUtil

Pomocná třída obsahující metody pro načítání šablon ze souboru. Jedná se o metodu `loadTemplate()`. Tato metoda má možné dva typy vstupního parametru. První z nich je `InputStream` a druhý je řetězec s cestou k šabloně. Pro oba vstupní parametry pak načte šablonu prvku ve formě řetězce.

Parametr typu `InputStream` je zde proto, že knihovna po zkompilování a vytvoření JAR souboru načítá výchozí šablonu právě z tohoto JAR souboru pomocí metody `getResourceAsStream()`, která vrací `InputStream`.

5.1.6 GlobalLogger

Pro logování je použita knihovna Apache Log4j 2 [41]. V této třídě je možné nastavit loggerům použitým v knihovně formát výstupu. Případně jestli se má logovat do konzole nebo do souboru.

Konfigurační soubor Log4j [42] je možné nahrát pomocí statické metody `loadLoggerConfig()`, vstupní parametr metody je cesta k souboru.

5.2 Šablony

Knihovna umožňuje uživateli vytvořit si vlastní šablonu pro každý vstupní prvek, který aplikace generuje. Šablony jsou HTML 5 kódy u kterých jsou hodnoty atributů nahrazeny klíčovým slovem. Každá šablona by měla obsahovat tato klíčová slova, která jsou později nahrazena hodnotami vygenerovanými třídou `HTMLGenerator`.

Dále jsou popsány jednotlivé výchozí šablony, které se použijí pokud si uživatel nevytvoří své vlastní. Výchozí šablony je možné nalézt ve zdrojových kódech knihovny ve složce `src/main/resources/templates/`.

5.2.1 Formulář

Výchozí šablona formuláře je ukázána na výpisu 5.9.

Tato šablona obsahuje několik klíčových slov. První je `{formClass}`, které udává CSS třídu formuláře. Potom je zde `{formId}` udávající CSS ID formuláře. Dále obsahuje `{formAction}` a to určuje jaká akce se provede po odeslání formuláře. Akce je URL odkaz na skript, který zpracuje odeslaný formulář. S tím souvisí další klíčové slovo a to `{formMethod}`, které určuje jakým způsobem se formulář odešle. Toto slovo může nabývat hodnot `POST` a nebo `GET`. Poslední klíčové slovo je `{formInputs}`, které je ve výsledném HTML kódu nahrazeno HTML kódem vstupních prvků.

```
<form class="{formClass}" id="{formId}" action="{formAction}"
        method="{formMethod}">
    {formInputs}
    <hr>
    <div>
        <input type="reset" value="Reset"/>
        <input type="submit" value="Odeslat"/>
    </div>
</form>
```

Výpis 5.9: Šablona formuláře

5.2.2 Textový vstupní prvek

Vstupní prvek je detailněji popsán v sekci 3.1.1. Šablona pro tento vstupní prvek je ukázána na výpisu 5.10.

```
<div>
    <label for="{inputId}">{inputValue}</label><br/>
    <input type="text" id="{inputId}" name="{inputName}"
        class="{inputClass}" {inputRequired}/>
</div>
```

Výpis 5.10: Šablona pro vstupní prvek typu text

Šablony pro vstupní prvky obsahují podobná klíčová slova jako šablona pro formulář, nachází se zde však pár odlišností.

Místo `{formId}` mají vstupní prvky klíčové slovo `{inputId}`, slovo `{formClass}` je nahrazeno slovem `{inputClass}`.

U všech vstupních prvků pak přibylo klíčové slovo `{inputName}` udávající jméno prvku. Tento atribut je u vstupních prvků důležitý, protože pokud se formulář odešle a je zde prvek bez jména, není možné získat z odeslaného požadavku jeho hodnotu.

Šablony vstupních prvků dále obsahují slovo `{inputValue}`, které určuje popisek co se má do vstupního prvku vyplnit. Neplatí to však u rozevíracího seznamu, podrobněji viz sekce 5.2.3.

Poslední slovo, které šablona pro textový vstupní prvek obsahuje je `{inputRequired}`, toto slovo určuje jestli uživatel prvek musí nebo nemusí před odesláním vyplnit.

5.2.3 Rozevírací seznam

Přepínač má dvě šablony. Jedna je pro vstupní prvek typu select a druhá pro prvek typu option.

Select

Tato šablona určuje jak bude vypadat rozevírací seznam. Šablonu je možné vidět na výpisu 5.11.

```
1 <div>
2     <label for="{inputId}">{inputValue}</label><br/>
3     <select id="{inputId}" name="{inputName}" class="{inputClass
          }"
4         {inputRequired}>
5         {inputSelectOptions}
6     </select>
7 </div>
```

Výpis 5.11: Šablona pro vstupní prvek typu select

Většina klíčových slov již byla popsána výše. Jediné slovo, které zde přibylo je `{inputSelectOptions}`. Toto slovo je nahrazeno položkami, které seznam obsahuje.

Option

Šablona pro jednotlivé položky rozevíracího seznamu. Šablonu je možné vidět na výpisu 5.12.

```
1 <option value="{inputValue}">{inputCaption}</option>
```

Výpis 5.12: Šablona pro vstupní prvek typu option

U tohoto prvku má klíčové slovo `{inputValue}` jiný význam než u ostatních prvků. Je to z toho důvodu, že rozevírací seznam se používá pro výpis instancí objektů. Při odeslání formuláře potřebujeme aby hodnota prvku byla URI adresa na danou instanci.

Aby se v seznamu nezobrazovala pouze URI adresa instance, je v šabloně nové klíčové slovo a to `{inputCaption}`. Toto slovo je nahrazeno informací o dané instanci podle hodnot anotací v pořadí `rdfs:label`, `dc:title` a `skos:prefLabel`, vybere se vždy první existující anotace. Usnadňuje to uživateli výběr správné položky. Pokud však tyto anotace neexistují, nahradí

se slovo výpisem dostupných vlastností dané instance. Jedná se o všechny hodnoty vlastností datového typu oddělené středníkem.

5.2.4 Číslo

Tento vstupní prvek má dvě specifická klíčová slova. První z nich je `{inputNumberMin}`, toto klíčové slovo nastavuje minimální číselnou hodnotu jaké může prvek nabývat. Druhé klíčové slovo je `{inputNumberMax}` a nastavuje maximální číselnou hodnotu jaké může prvek nabývat.

Šablonu pro vstupní prvek číslo je možné vidět na výpisu 5.13.

```
1 <div>
2     <label for="{inputId}">{inputCaption}</label><br/>
3     <input type="number" id="{inputId}" name="{inputName}"
4         class="{inputClass}" {inputValue} {inputNumberMin}
5            {inputNumberMax} {inputRequired}/>
6 </div>
```

Výpis 5.13: Šablona pro vstupní prvek typu number

5.2.5 Datum a čas

Vstupní prvky datum a datum s časem mají stejnou šablonu jako textový vstupní prvek. Jediný rozdíl je v atributu `type`. Jeho hodnota je u data `date`. U data s časem je jeho hodnota `datetime-local`.

5.3 Ukázka kódu generování formuláře

Pro snazší pochopení jak celý proces generování formuláře pomocí knihovny funguje uvedu jednoduchý příklad.

Na výpisu 5.14 je možné vidět ukázku kódu, který vygeneruje formulář z ontologie `dasta.owl`[1] pro třídu `Address` a HTML kód formuláře uloží do Stringu.

```
1 GFormOntology gfo = new GFormOntology("https://mre.zcu.cz/
      ontology/dasta.owl", OntModelSpec.OWL_DL_MEM);
2 gfo.createRDFInstance("output.xml",
3     GFormData.Type.RDF_XML_ABBREV);
4 gfo.setLabelLanguagePref(new String[]{"cs"});
5 gfo.generateInputs("http://mre.zcu.cz/ontology/dasta.owl#
      Address");
6
7 String outputHTML = gfo.getHtmlGenerator().generate();
```

Výpis 5.14: Ukázka kódu pro vygenerování formuláře z ontologie

Na 1. řádku se vytvoří instance třídy `GFormOntology`.

Na 2. řádku se vytvoří uvnitř třídy `GFormOntology` instance třídy `GFormData`, která přistupuje k uloženým datům. Parametry jsou cesta k výstupnímu souboru a formát uložených dat.

Na 4. řádku se nastaví preference jazyků.

Na 5. řádku pak metoda `generateInputs()` vygeneruje vstupní prvky formuláře pro třídu `Address` z ontologie.

Poslední řádek pak získá vygenerovaný HTML kód formuláře ve formě řetězce.

Na obrázku 5.6 je možné vidět jak vypadá formulář vygenerovaný pomocí kódu z výpisu 5.14. Jedná se čistě o HTML 5 kód bez jakýchkoliv CSS stylů. Vzhled formuláře je tak v rukou uživatele. Ukázka jak vypadá HTML kód tohoto formuláře je na výpisu 1 v příloze A této práce.

5.4 Překlad knihovny

Knihovna byla psaná v jazyce Java a pro překlad je nutný Java překladač alespoň ve verzi 1.8. Pro snadný překlad knihovny je možné použít nástroj Maven. Překlad pomocí nástroje Maven byl otestován na verzi 3.3.9.

[1] `https://mre.zcu.cz/ontology/dasta/`

Obrázek 5.6: Ukázka vygenerovaného fragmentu HTML5 formuláře

6 Ukázková webová aplikace

Součástí této bakalářské práce bylo vytvoření ukázkové webové aplikace, která bude používat vytvořenou knihovnu. Ukázková aplikace bude sloužit pro generování formulářů, zpracování dat z formulářů do formátu RDF a také jejich editaci. Zároveň tak demonstruje možnosti a použití vytvořené knihovny.

6.1 Návrh webové aplikace

Ukázková aplikace by měla fungovat na serveru s Apache Tomcat v minimální verzi 8.0.43. Dále by měla být snadno přeložitelná, jelikož u překladu knihovny je použit nástroj Maven, měl by být použit i u ukázkové aplikace.

Rozhodoval jsem se jestli použít nějaký framework pro webové aplikace psané v Javě, například Spring. Jelikož tato aplikace bude mít pouze dvě dynamicky se měnící webové stránky rozhodl jsem se žádný framework nepoužít.

Dále se budu věnovat návrhu jednotlivých částí ukázkové aplikace.

Nastavení aplikace

Aplikace by měla mít možnost nastavit kde bude uložen soubor do kterého se budou ukládat data z formuláře a také formát těchto výstupních dat. Dále by měla mít možnost nastavit cestu kam se budou ukládat soubory nahrané z formulářů, případně pokud se bude do aplikace nahrávat nová ontologie ze souboru, tak nastavit cestu kam se budou tyto ontologie ukládat.

Jednotlivé instance, které jsou uloženy ve výstupních datech jsou označeny atributem `rdf:about`, které obsahují URI s jednoznačným identifikátorem. Například na výpisu 2.1 je možné na 4. řádce vidět instanci třídy Adresa a její jednoznačnou URI – `http://filek.cz/adresa/5`.

Aplikace by tedy měla mít možnost jak nastavit formát těchto URI identifikátorů.

Generování formulářů

Generování formulářů může v případě většího množství dat trvat i několik desítek vteřin. Z tohoto důvodu by generování mělo probíhat pomocí požadavků posílaných serveru přes JavaScript. Při tomto řešení se aplikace nebude uživateli jevit jako zamrznutá, ale bude dále pracovat.

S tím souvisí zobrazování informací uživateli. Pokud nastane chyba při generování nebo odesílání formuláře, případně při dalších operacích, měla by aplikace zobrazit chybovou hlášku uživateli. To samé platí i pro úspěšné hlášky, například když se podaří formulář odeslat.

Vygenerovaný formulář by se měl zobrazovat na různých rozlišeních tak aby ho bylo možné stále bez problému vyplnit. Z tohoto důvodu jsem se rozhodl pro použití frameworku Bootstrap [43], který zajišťuje responzivitu webových aplikací.

Editace dat

Aplikace by měla poskytnout možnost jak upravit již existující data. Podobně jako u generování formulářů bude tato funkčnost zajištěna JavaScriptem.

Pro přehlednost by aplikace měla data rozdělit podle třídy instance. To znamená zobrazit rozevírací seznam, kde bude seznam všech tříd, které mají nějakou instanci v uložených datech. Po výběru třídy se zobrazí seznam všech instancí dané třídy. Po kliknutí na instanci se vygeneruje formulář umožňující editaci těchto dat. Existující data by se měla automaticky načíst do jednotlivých vstupních prvků.

6.2 Implementace aplikace

V sekci vzhled aplikace 6.2.1 se budu zabývat vzhledem aplikace a funkčností jednotlivých webových stránek aplikace.

V sekci 6.2.2 pak popíšu jednotlivé třídy a servlety webové aplikace.

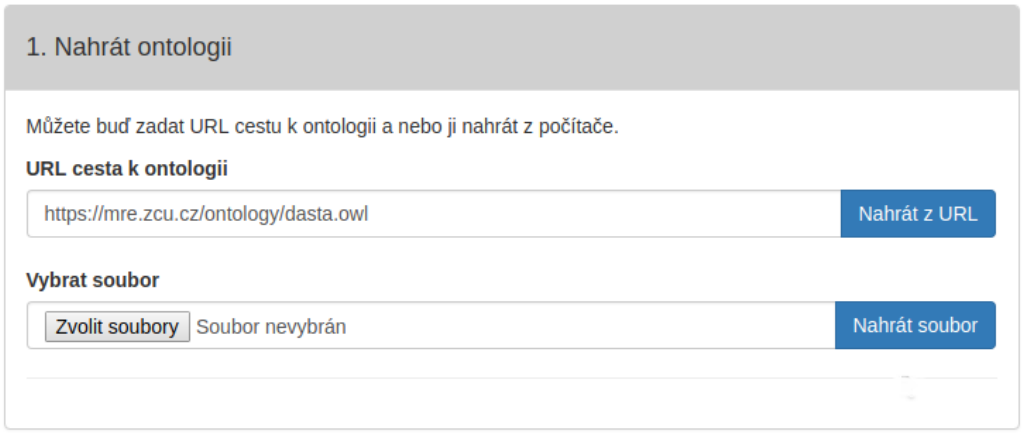
6.2.1 Vzhled aplikace

Jak jsem již v návrhu aplikace psal, rozhodl jsem se pro vzhled aplikace použít framework Bootstrap. S tímto frameworkem je vytvoření graficky přívětivé a responzivní aplikace velice jednoduché.

Stránka generování formulářů

Šablona pro tuto stránku se nachází ve složce `WEB-INF/view/generate.jsp` ve složce se zdrojovými kódy webové aplikace. Stránka pro generování formulářů, kromě generování formulářů, také slouží k nahrávání nových ontologií do modelu, případně přidávání existujících dat do výstupního modelu. Tato stránka je rozdělena na čtyři části.

První částí je formulář umožňující nahrávání ontologií do modelu. Tento formulář obsahuje dva vstupní prvky. Jeden vstupní prvek je pro nahrávání ontologií z URL adresy. Druhý vstupní prvek pak slouží k nahrávání ontologie z počítače. Tento formulář je možné vidět na obrázku 6.1.

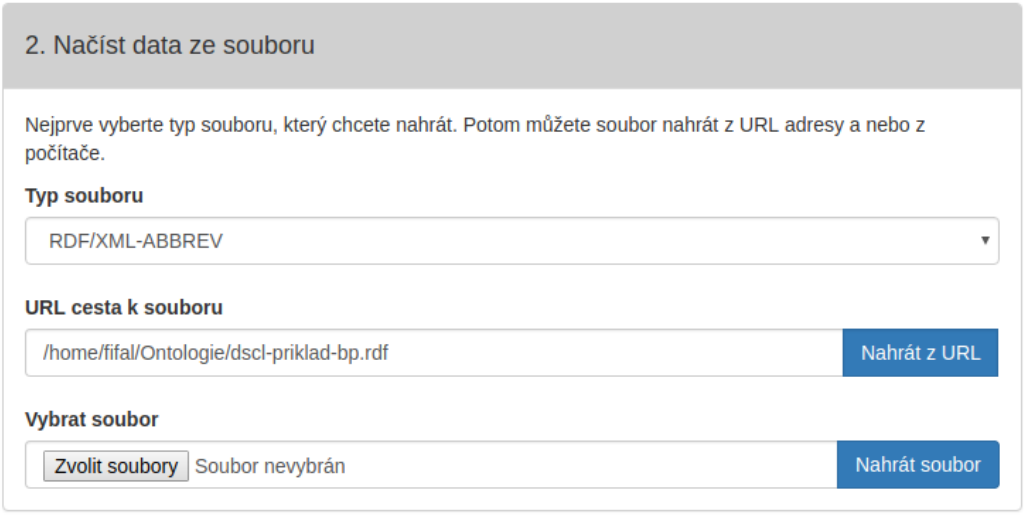


Obrázek 6.1: Formulář pro nahrávání ontologií

Druhou částí je formulář, který umožňuje nahrávání existujících dat do výstupního modelu. Tento formulář obsahuje tři vstupní prvky. Jedná se o rozevírací seznam, který obsahuje formáty vstupních souborů, které dokáže aplikace načíst. Další dva prvky jsou podobné jako u formuláře pro nahrávání ontologií. Jeden prvek slouží k nahrávání dat z URL adresy a druhý ze souboru. Ukázka tohoto formuláře je na obrázku 6.2.

Třetí částí je formulář sloužící k výběru třídy z ontologie pro kterou chce uživatel vygenerovat formulář. Formulář obsahuje dva vstupní prvky. Rozevírací seznam obsahující seznam všech tříd z ontologie pro které je možné vygenerovat formulář. Jelikož seznam může být velice rozsáhlý je možné v tomto rozevíracím seznamu vyhledávat podle zadaného řetězce. Po výběru třídy je možné vygenerovat formulář pomocí tlačítka, které se nachází pod rozevíracím seznamem. Formulář je možné vidět na obrázku 6.3 a upravený rozevírací seznam s možností vyhledávání na obrázku 6.4.

Pro vygenerování formuláře, pokud ještě nebyla nahrána žádná ontologie, musí uživatel nahrát soubor s ontologií nebo zadat URL cestu k ontologii. Po nahrání ontologie se vygeneruje seznam tříd pro které je možné vygenerovat formulář. Uživatel tedy vybere třídu pro kterou chce formulář vygenerovat a po kliknutí na tlačítko „Vygenerovat formulář“ se mu v poslední části stránky zobrazí vygenerovaný formulář, který je možné vyplnit a odeslat.



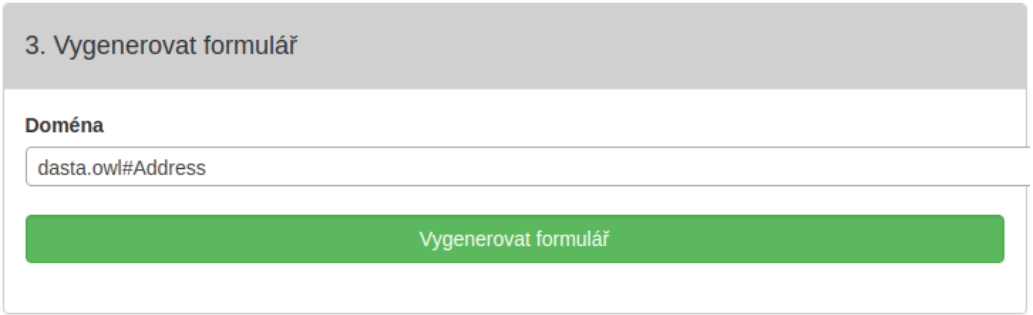
Obrázek 6.2: Formulář pro nahrávání datových souborů

Ukázka vygenerovaného formuláře je na obrázku 1 v příloze A této práce.

Stránka editace dat

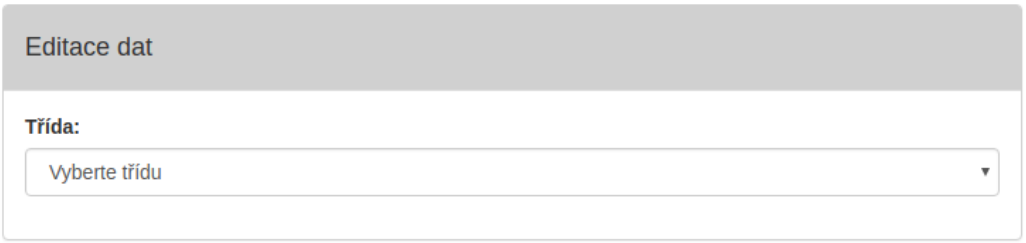
Šablona pro tuto stránku se nachází ve složce `WEB-INF/view/editdata.jsp`. Pokud se uživatel přepne na stránku kde je možné editovat data, zobrazí se mu na stránce pouze jeden vstupní prvek. Je to rozevírací seznam, který obsahuje všechny třídy z modelu, které mají v uložených datech alespoň jednu instanci. Tento prvek je možné vidět na obrázku 6.5.

Po výběru třídy se na stránce vygeneruje další rozevírací seznam, který tentokrát obsahuje seznam všech instancí dané třídy viz obrázek 6.6.



Obrázek 6.3: Formulář pro generování formulářů podle třídy ontologie

Obrázek 6.4: Upravený rozevírací seznam s možností vyhledávání



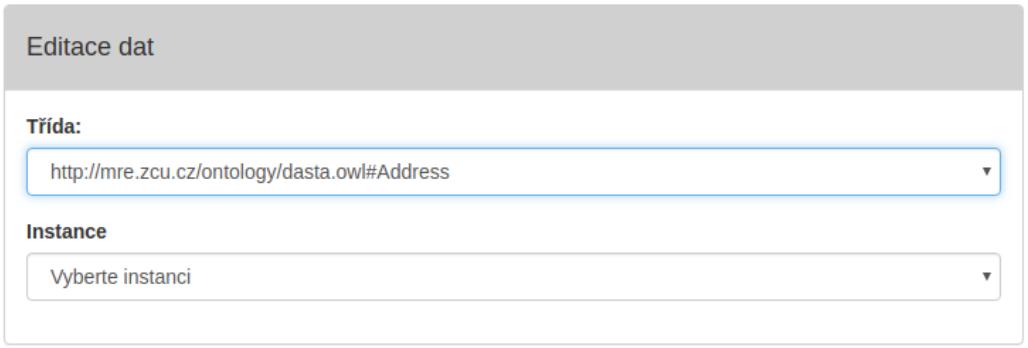
Obrázek 6.5: Vstupní prvek pro výběr třídy

A nakonec po výběru instance se zobrazí formulář, ve kterém je možné editovat všechny hodnoty této instance. Ukázku formuláře pro editaci dat je možné vidět na obrázku 2 v příloze A této práce.

Podformuláře

Jelikož formuláře obsahují rozevírací seznamy obsahující seznam instancí, není možné tyto hodnoty vyplňovat ručně. Pokud by tedy uživatel generoval formulář pro osobu, která bude mít adresu bydliště, ale instance dané adresy se nenachází ve výstupních datech, neměl by uživatel žádnou možnost jak vyplnit správnou adresu. Proto je v aplikaci implementována možnost přidat novou instanci do tohoto rozevíracího seznamu.

Při rozevření daného rozevíracího seznamu se zde nachází položka „Přidat



Obrázek 6.6: Po výběru třídy se vygeneruje rozevírací seznam s instancemi

nový záznam“. Po kliknutí na tuto položku se vygeneruje podformulář, zobrazí se a je možné přidat nové hodnoty. Když uživatel tento formulář vyplní a odešle, v původním rozevíracím seznamu se vybere tato nová hodnota. Tato funkčnost byla implementována jak u generování formulářů, tak při editaci dat. Ukázka tohoto formuláře je na obrázku 3 v příloze A této práce.

6.2.2 Servlety a Třídy

V této sekci se budu věnovat implementaci jednotlivých servletů a tříd, které zajišťují fungování ukázkové webové aplikace.

Třída Global

V této třídě má uživatel možnost nastavení cesty složek pro ukládání souborů a ontologií na serveru. Dále nastavení cesty výstupního souboru a formátu dat ve kterém má být soubor uložen. Poslední nastavení je prefix `rdf:about` URI pod jakou se ukládají nové instance z odeslaných formulářů. Výchozí nastavení tohoto prefixu je `http://mre.zcu.cz/id/` kde za poslední lomítko je doplněno ID nově vytvořené instance

Servlet MainServlet

Tento servlet pouze zajišťuje zobrazení úvodní stránky webové aplikace.

Servlet QueryServlet

Toto je servlet, který slouží ke zpracování požadavků převážně od JavaScriptu. Jednotlivé požadavky se posílají na adresu `/query?action=<action>` kde za

`<action>` je dosazena akce která se má provést a také některé další povinné parametry, které jsou pro každý typ akce různé. Dále popíšu možné akce, jejichž povinné parametry je možné nalézt v okomentovaném zdrojovém kódu aplikace v metodě `hasValidParams()`.

- `readOntology` – akce sloužící pro načtení ontologie z URL adresy.
- `readOntologyFromFile` – načtení ontologie do modelu ze souboru.
- `readData` – akce pro načtení dat do výstupního modelu.
- `readDataFromFile` – načtení dat do výstupního modelu ze souboru.
- `generateForm` – tato akce slouží k vygenerování formuláře podle třídy ontologie.
- `listOntClasses` – tato akce vrací list všech tříd z ontologie pro které je možné vygenerovat formulář.
- `listClasses` – akce, která z výstupního modelu načte všechny třídy, které zde mají alespoň jednu instanci.
- `listInstances` – tato akce vygeneruje seznam instancí pro danou třídu ve výstupním modelu.
- `listProperties` – vrací seznam všech hodnot dané instance.
- `removeResourceFromInstance` – slouží k odstranění vlastnosti z instance.

Servlet FormServlet

Tento servlet zajišťuje zobrazování stránek pro generování formulářů, jejich editaci a také zpracovává odeslané vyplněné formuláře.

Stránka pro generování formulářů je dostupná na adrese `/generateForm`. Stránka pro editaci dat je na adrese `/edit`.

Vygenerované formuláře se posílají na adresu `/form` pomocí metody POST. Zpracování odeslaných formulářů probíhá následujícím způsobem: nejprve se získají všechny parametry z požadavku a zapíší se do hashmapy, dále se získají všechny soubory z požadavku, které jsou dále pomocí statické metody ze třídy `Util` zapsány na disk serveru, jedná se o metodu `processFile()`. Cesta k souboru se také přidá do hashmapy.

Pokud se jedná o formulář, který neobsahuje parametr `form-id` jedná se o nový formulář a je mu vygenerováno nové ID. Pokud toto ID obsahuje, znamená to, že uživatel pouze upravuje existující data.

Nakonec je zavolána metoda ze třídy `GFormOntology`, která hashmapu zpracuje. Jedná se o `processDataAndWriteToOutputFile()`, které je předána hashmapa a data jsou posléze zpracována a zapsána do výstupního souboru.

Třída FileUtil

Tato třída obsahuje metodu, která slouží ke zpracování souborů odeslaných na server pomocí formulářů. Zkontroluje zdali se soubor se stejným jménem již v adresáři se soubory nenachází. Pokud se zde soubor nachází, přidá na začátek názvu tohoto souboru číslo podle toho kolik souborů se stejným jménem se ve složce nachází. Tímto tak nedojde k přepsání již existujících souborů.

Potom už jen zapíše soubor na disk a vrátí instanci třídy `File`.

6.3 Nasazení aplikace

Aplikace byla programována pro použití na serveru s Apache Tomcat minimálně ve verzi 8. Nasazení aplikace bylo testováno na Apache Tomcat ve verzích 8.0.43 a 9.0.0.M17, kde aplikace fungovala bez problému.

Dále na verzi 8.5.13, kde se aplikaci nepodařilo nasadit. Při hledání problému jsem zjistil, že je to chyba této verze Tomcatu a měla by být vyřešena v další verzi.

Aplikace byla vyvíjena pro použití v prohlížeči Google Chrome a byla testována ve verzi Chrome 55.0.2883.87.

7 Diskuze

7.1 Zhodnocení výsledků

Podařilo se mi vytvořit knihovnu i ukázkovou webovou aplikaci používající tuto knihovnu. Přestože jsou obě části funkční, tak webová aplikace i knihovna by šla jistě vylepšit co se týče zdrojového kódu. U webové aplikace se jedná hlavně o kód psaný v JavaScriptu. Bylo to moje první seznámení s tímto jazykem a jeho možnostmi a proto není kód napsaný optimálně. U knihovny jsem se snažil zdrojový kód psát přehledně včetně komentářů. Ale čím více funkčností jsem do knihovny přidával, tím nepřehlednější se kód stával. Knihovna splňuje požadavky pro generování formulářů viz 4.1.1, pro editaci dat viz 4.1.2 i pro ukládání dat viz 4.1.3.

7.2 Možnosti rozšíření

Z důvodu nedostatku času se mi nepodařilo implementovat veškerou funkčnost popisovanou v návrhu webové aplikace. Aplikace byla vyvíjena a testována v prohlížeči Chrome ve verzi 55.0.2883.87. V tomto prohlížeči aplikace funguje bez problému. Na testování v ostatních prohlížečích mi nezbyl dostatek času. Proto v prohlížeči Mozilla Firefox, protože nepodporuje vstupní prvek typu datum a datum s časem, se místo těchto prvků zobrazí vstupní prvek typu text. JS kód ukázkové aplikace v prohlížeči Mozilla Firefox také nefunguje správně. Jedno z možných rozšíření webové aplikace by bylo zprovoznění těchto vstupních prvků a JS kódu i v další prohlížečích.

Knihovnu by šlo rozšířit. Bylo by vhodné, aby se dalo při generování prvků formuláře určit, jestli má být prvek ve formuláři zahrnut či nikoliv. Dále možnost určit u generovaného prvku atribut „pouze pro čtení“, například pro informace o nahraných souborech, které obsahují vlastnost „velikost souboru“. Tato hodnota by se zobrazovala, ale nemělo by být možné ji editovat.

7.3 Porovnání s existujícím řešením

Součástí diskuze mělo být porovnání vytvořeného řešení s již existujícím řešením. Jediné řešení, které jsem nalezl pro generování formulářů z ontologií

je aplikace ActiveRaUL viz sekce 3.4.3, která již nefunguje. Z tohoto důvodu nemohu knihovnu s tímto řešením porovnat.

Co se týče ostatních řešení ze sekce 3.4, vytvořená knihovna dokáže podobně jako ostatní řešení generovat HTML formuláře. Dokáže je však generovat z ontologií, což žádné jiné řešení nenabízí. Data dokáže jako jediná zpracovat do výstupního formátu RDF.

V porovnání s Google Forms (dále jen GF) viz sekce 3.4.4, dokáže knihovna generovat vstupní prvek typu číslo. V GF tento prvek není možné vygenerovat, místo čísla lze použít vstupní prvek text, do kterého je možné zadat i číslo, ale zadané hodnoty není možné před odesláním zvalidovat a proto je nutné data validovat až po odeslání. V čem GF vynikají oproti vytvořené knihovně je zobrazování vstupních prvků. Dostupné vstupní prvky v GF se po vytvoření formuláře zobrazují na všech prohlížečích stejně. Například vstupní prvek datum se zobrazí správně i v prohlížeči Mozilla Firefox, což vytvořená knihovna nepodporuje.

Podobný problém se zobrazováním vstupního prvku datum má i framework Angular Schema Form (dále jen ASF) viz sekce 3.4.5. ASF nativně nepodporuje vstupní prvek datum, ale protože podporuje doplňky třetích stran, je možné tuto funkčnost přidat pomocí doplňku Datepicker. Vytvořená knihovna doplňky nepodporuje a doplnit podporu vstupního prvku datum pro ostatní prohlížeče by znamenalo úpravu stávajícího kódu.

Knihovna xsd-forms (viz sekce 3.4.6) a framework ASF podporují úpravu vzhledu formulářů. V ASF je možné upravit vzhled vstupního prvku pomocí atributu `fieldHtmlClass` v JSON Schema objektu. Tento atribut přiřadí danému prvku CSS třídu. Atribut je pak nutné nastavit každému vstupnímu prvku zvlášť. U knihovny xsd-forms lze vzhled formuláře upravit pomocí JavaScriptu. Musí se však upravovat vzhled každého vstupního prvku až po vygenerování formuláře zvlášť. Vytvořená knihovna podporuje šablony a tak je vzhled prvku upraven pouze jednou a šetří tím čas.

8 Závěr

Tématem práce byly formuláře pro získávání RDF dat. Cílem bylo seznámit se s oblastí RDF a OWL, najít existující řešení pro popis a sběr dat prostřednictvím automaticky generovaných formulářů a analyzovat požadavky kladené na formuláře. Dále navrhnout a implementovat knihovnu v jazyce Java, která usnadní generování formulářů z ontologií a pro demonstraci knihovny vytvořit ukázkovou webovou aplikaci.

V první části práce jsem popsal a vysvětlil některé pojmy související s prací s ontologiemi a ukládáním RDF dat. Analyzoval jsem požadavky kladené na jednotlivé vstupní prvky formuláře rozdělené podle typu. Dále jsem analyzoval požadavky pro podmíněné formuláře a podformuláře. V další části jsem se věnoval návrhu a implementaci knihovny. Vybral jsem vhodný framework pro práci s ontologiemi a pro práci s RDF daty. U implementace knihovny jsem popsal jednotlivé třídy a jejich metody, které jsou nezbytné pro práci s knihovnou. Popsal jsem jakým způsobem se knihovna používá jako celek a jak si uživatel může vytvořit vlastní šablony pro generované formuláře a jejich vstupní prvky. Další částí práce bylo vytvoření ukázkové webové aplikace. V této části jsem se věnoval jak jejímu návrhu, tak implementaci. Poslední částí práce byla diskuze a porovnání aplikace s řešeními jiných autorů. V diskuzi jsem se věnoval výsledkům práce s ohledem na splnění specifikovaných požadavků a také možným rozšířením této práce.

Podařilo se mi vytvořit knihovnu podle specifikovaných požadavků. Nicméně během implementace a porovnávání knihovny s existujícími řešeními byly nalezeny možnosti pro další rozšíření. Tyto možnosti byly podrobněji popsány v sekci 7.2. Dle návrhu jsem vytvořil ukázkovou webovou aplikaci, která demonstruje funkčnost a možnosti použití vytvořené knihovny. Webová aplikace umožňuje nahrávání ontologií na server, z těchto ontologií je možné generovat formuláře. Formuláře lze vyplnit a odeslat. Data z odeslaných formulářů se ukládají do výstupního souboru a v aplikaci je pak možné tato data upravovat. Podobně jako u knihovny byly v aplikaci nalezeny možnosti k jejímu dalšímu rozšíření, které jsou popsány v sekci 7.2.

Literatura

[1] *RDF* [online]. W3C, 2014. [cit. 2016/12/26]. Dostupné z: https://www.w3.org/RDF.

[2] Petr Matulík, T. P. *Sémantický web a jeho technologie* [online]. Zpravodaj ÚVT MU, 2004. [cit. 2017/04/15]. Dostupné z: http://webserver.ics.muni.cz/bulletin/articles/296.html.

[3] *XML Schema* [online]. W3C, 2004. [cit. 2016/12/26]. Dostupné z: http://www.w3.org/2001/XMLSchema.

[4] Powers, S. *Practical RDF*. O'Reilly, 2003. ISBN 0-596-00263-7.

[5] *RDF Schema 1.1* [online]. W3C, 2014. [cit. 2016/12/28]. Dostupné z: https://www.w3.org/TR/rdf-schema/.

[6] *OWL* [online]. W3C, 2012. [cit. 2016/12/26]. Dostupné z: https://www.w3.org/OWL/.

[7] Černá, A. *Sémantický popis UML modelů pomocí OWL ontologií* [online]. Masarykova univerzita, Fakulta informatiky, 2013. [cit. 2017/01/25]. Dostupné z: https://is.muni.cz/th/172898/fi_m/thesis.pdf.

[8] Lacy, L. W. *OWL: Representing information using the web ontology language*. Trafford, 2005. ISBN 1-4120-3448-5.

[9] *OWL Class descriptions* [online]. W3C, 2004. [cit. 2017/04/20]. Dostupné z: https://www.w3.org/TR/owl-ref/#ClassDescription.

[10] *OWL Simple properties* [online]. W3C, 2004. [cit. 2017/04/20]. Dostupné z: https://www.w3.org/TR/2004/REC-owl-guide-20040210/#SimpleProperties.

[11] *OWL Properties* [online]. W3C, 2004. [cit. 2017/04/20]. Dostupné z: https://www.w3.org/TR/owl-ref/#Property.

[12] *RDF 1.1 N-Triples* [online]. W3C, 2014. [cit. 2016/12/27]. Dostupné z: https://www.w3.org/TR/n-triples/.

[13] *RDF 1.1 N-Quads* [online]. W3C, 2014. [cit. 2016/12/27]. Dostupné z: https://www.w3.org/TR/n-quads/.

[14] *HTML5* [online]. W3C, 2014. [cit. 2016/12/28]. Dostupné z: https://www.w3.org/TR/html5/.

[15] *HTML5 form additions* [online]. W3C, 2014. [cit. 2017/04/20]. Dostupné z: https://www.w3.org/wiki/HTML5_form_additions#New_form_controls.

[16] *HTML5 input types* [online]. W3C, 2017. [cit. 2017/04/20]. Dostupné z: https://www.w3schools.com/html/html_form_input_types.asp.

[17] *W3Counter Browser and Platform Market Share* [online]. W3C, 2016. [cit. 2016/12/28]. Dostupné z: https://www.w3counter.com/globalstats.php.

[18] *Open Data Kit* [online]. ODK, 2015. [cit. 2016/12/26]. Dostupné z: https://opendatakit.org/.

[19] *XForms 1.1* [online]. W3C, 2009. [cit. 2016/12/26]. Dostupné z: https://www.w3.org/TR/xforms/.

[20] *ActiveRaUL* [online]. ActiveRaUL, 2013. [cit. 2016/12/27]. Dostupné z: http://www.activeraul.org/.

[21] Butt, A. et al. *ActiveRaUL: A Web form-based User Interface to create and maintain RDF data* [online]. ISWC 2013, 2013. [cit. 2016/12/27]. Dostupné z: http://iswc2013.semanticweb.org/sites/default/files/iswc_demo_30.pdf.

[22] Google. *Google Forms* [online]. 2017. [cit. 2017/12/27]. Dostupné z: https://www.google.com/forms/about/.

[23] Chun, W. *Auto-generating Google Forms* [online]. Google, 2016. [cit. 2016/12/27]. Dostupné z: https://developers.googleblog.com/2016/06/auto-generating-google-forms.html.

[24] TEXTALK. *Angular Schema Form* [online]. 2017. [cit. 2017/04/26]. Dostupné z: http://schemaform.io/.

[25] Moten, D. *xsd-forms* [online]. 2017. [cit. 2017/04/26]. Dostupné z: https://github.com/davidmoten/xsd-forms.

[26] *The OWL API* [online]. owlcs, 2017. [cit. 2017/03/30]. Dostupné z: http://owlapi.sourceforge.net/.

[27] *Apache Jena* [online]. The Apache Software Foundation, 2017. [cit. 2017/03/24]. Dostupné z: https://jena.apache.org/.

[28] *Interface OntClass* [online]. The Apache Software Foundation, 2017. [cit. 2017/04/13]. Dostupné z: https://jena.apache.org/documentation/javadoc/jena/org/apache/jena/ontology/OntClass.html.

[29] *Interface Statement* [online]. The Apache Software Foundation, 2017. [cit. 2017/04/13]. Dostupné z: https://jena.apache.org/documentation/javadoc/jena/org/apache/jena/rdf/model/Statement.html.

[30] Ignazio Palmisano, t. O. A. t. *The Rough Guide to the OWL API: a tutorial* [online]. University of Manchester, 2011. [cit. 2017/02/17]. Dostupné z: http://owlapi.sourceforge.net/owled2011_tutorial.pdf.

[31] *Apache Fuseki* [online]. The Apache Software Foundation, 2017. [cit. 2017/04/13]. Dostupné z: https://jena.apache.org/documentation/serving_data/.

[32] *Jena documentation overview* [online]. The Apache Software Foundation, 2017. [cit. 2017/03/30]. Dostupné z: https://jena.apache.org/documentation/.

[33] *Reading and Writing RDF in Apache Jena* [online]. The Apache Software Foundation, 2017. [cit. 2017/04/13]. Dostupné z: https://jena.apache.org/documentation/io/#formats.

[34] *Apache Maven* [online]. The Apache Software Foundation, 2017. [cit. 2017/04/13]. Dostupné z: https://maven.apache.org/index.html.

[35] *Interface OntModel* [online]. The Apache Software Foundation, 2017. [cit. 2017/04/18]. Dostupné z: https://jena.apache.org/documentation/javadoc/jena/org/apache/jena/ontology/OntModel.html.

[36] *Class OntModelSpec* [online]. The Apache Software Foundation, 2017. [cit. 2017/04/18]. Dostupné z: https://jena.apache.org/documentation/javadoc/jena/org/apache/jena/ontology/OntModelSpec.html.

[37] *Interface Model* [online]. The Apache Software Foundation, 2017. [cit. 2017/04/18]. Dostupné z: https://jena.apache.org/documentation/javadoc/jena/org/apache/jena/rdf/model/Model.html.

[38] *Interface RDFNode* [online]. The Apache Software Foundation, 2017. [cit. 2017/04/18]. Dostupné z: https://jena.apache.org/documentation/javadoc/jena/org/apache/jena/rdf/model/RDFNode.html.

[39] *Interface Resource* [online]. The Apache Software Foundation, 2017. [cit. 2017/04/18]. Dostupné z:

https://jena.apache.org/documentation/javadoc/jena/org/apache/jena/rdf/model/Resource.html.

[40] *Interface DatatypeProperty* [online]. The Apache Software Foundation, 2017. [cit. 2017/04/18]. Dostupné z: https://jena.apache.org/documentation/javadoc/jena/org/apache/jena/ontology/DatatypeProperty.html.

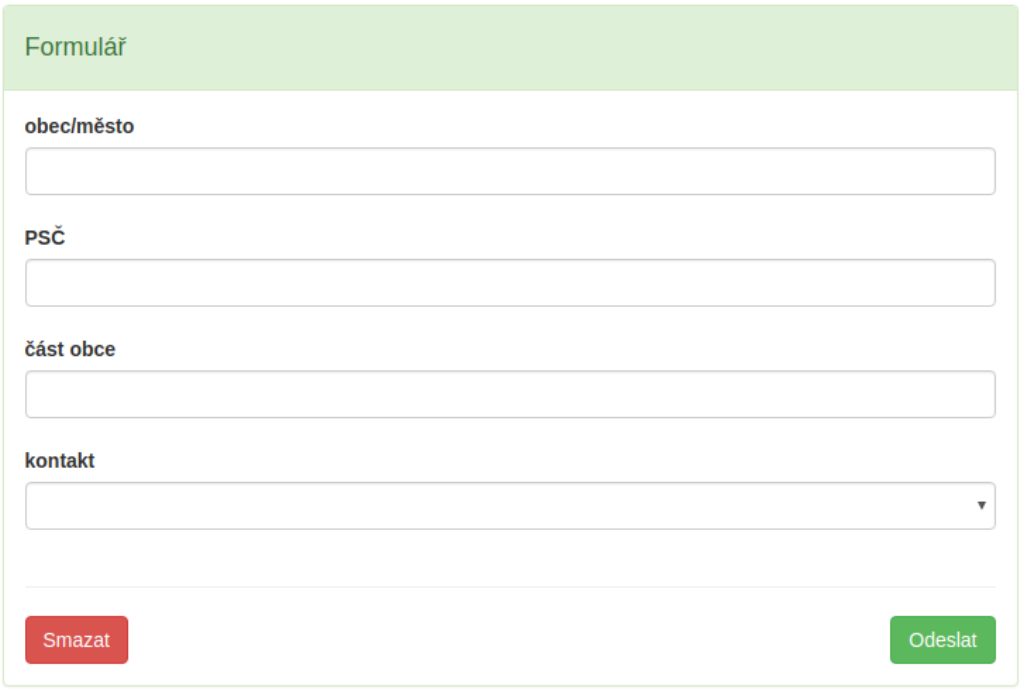
[41] *Apache Log4j 2* [online]. The Apache Software Foundation, 2017. [cit. 2017/04/18]. Dostupné z: https://logging.apache.org/log4j/2.x/.

[42] *log4j - Configuration* [online]. Tutorialspoint, 2017. [cit. 2017/04/18]. Dostupné z: https://www.tutorialspoint.com/log4j/log4j_configuration.htm.

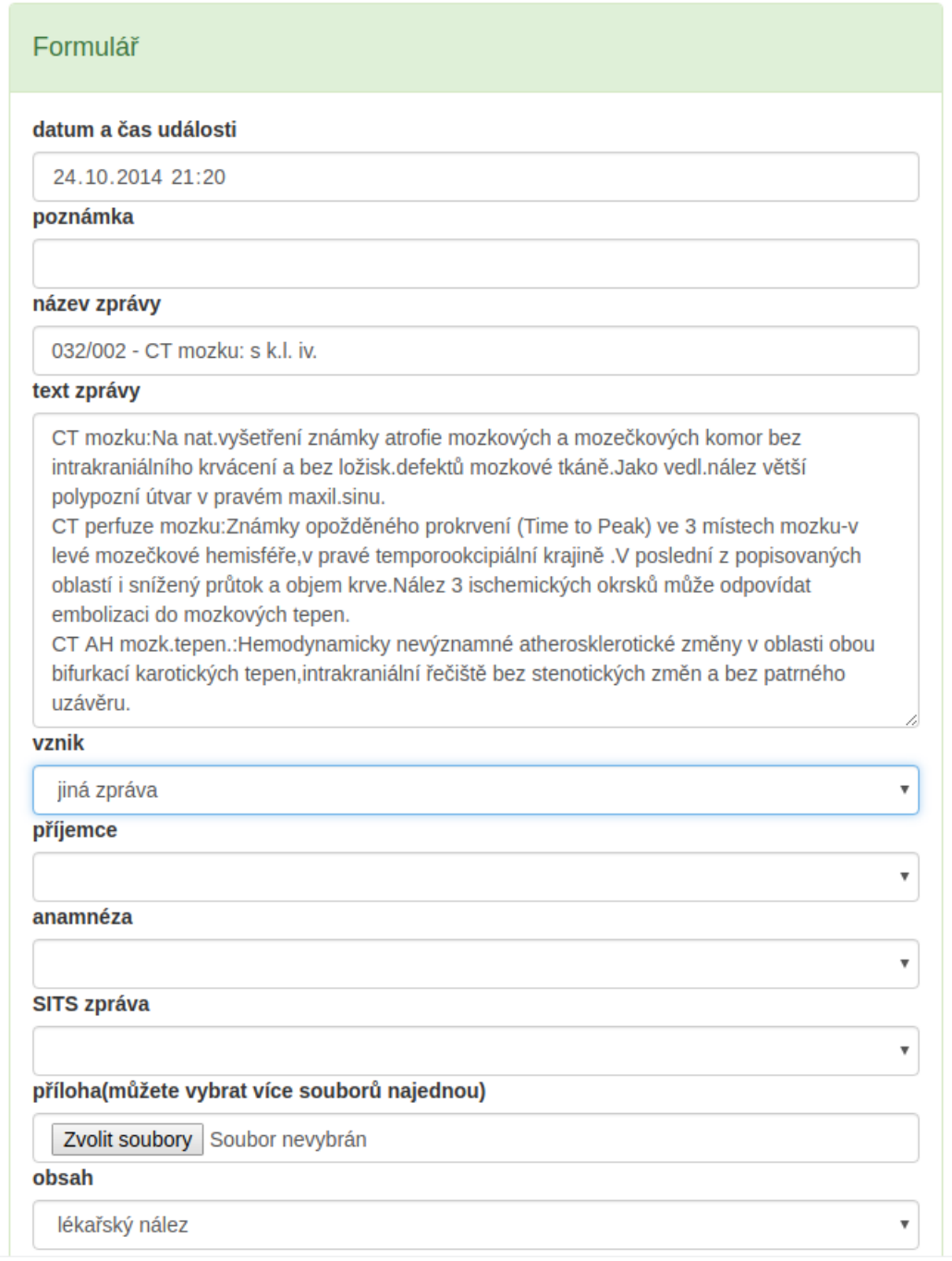
[43] *Bootstrap* [online]. Bootstrap, 2017. [cit. 2017/04/18]. Dostupné z: http://getbootstrap.com/.

Přílohy

A Ukázky vygenerovaných formulářů



Obrázek 1: Vygenerovaný formulář pro třídu Address



Obrázek 2: Složitější formulář pro editaci dat pro třídu MedicalExamination

Obrázek 3: Podformulář pro přidání nové hodnoty

B Výpisy zdrojových kódů

```
1  <form class="form" id="form" action="form.php" method="GET">
2  <div>
3          <label for="http://mre.zcu.cz/ontology/dasta.owl#
               addressCity">obec/mesto</label><br/>
4          <input type="text" id="http://mre.zcu.cz/ontology/dasta.
               owl#addressCity" name="http://mre.zcu.cz/ontology/
               dasta.owl#addressCity" class="form-control" />
5  </div>
6  <div>
7          <label for="http://mre.zcu.cz/ontology/dasta.owl#
               addressZIP">PSC</label><br/>
8          <input type="text" id="http://mre.zcu.cz/ontology/dasta.
               owl#addressZIP" name="http://mre.zcu.cz/ontology/
               dasta.owl#addressZIP" class="form-control" />
9  </div>
10 <div>
11         <label for="http://mre.zcu.cz/ontology/dasta.owl#
               addressCityPart">cast obce</label><br/>
12         <input type="text" id="http://mre.zcu.cz/ontology/dasta.
               owl#addressCityPart" name="http://mre.zcu.cz/ontology
               /dasta.owl#addressCityPart" class="form-control" />
13 </div>
14 <div>
15         <label for="http://mre.zcu.cz/ontology/dasta.owl#contact
               ">kontakt</label><br/>
16         <select id="http://mre.zcu.cz/ontology/dasta.owl#contact
               " name="http://mre.zcu.cz/ontology/dasta.owl#contact"
                class="form-control" >
17         <option value=""></option>
18         </select>
19 </div>
20 <div>
21         <input type="reset" value="Reset"/>
22         <input type="submit" value="Odeslat"/>
23 </div>
24 </form>
```

Výpis 1: Vygenerovaný HTML kód pro třídu Address

C Obsah přiloženého DVD

Součástí práce je DVD obsahující veškeré zdrojové kódy knihovny i ukázkové webové aplikace. Obsahuje text práce psaný v LaTeXu a jeho tisknutelnou podobu ve formátu PDF.

Struktura DVD

Kořenová složka DVD obsahuje tři složky:

- `gform_bptext` – složka obsahuje zdrojový text práce včetně všech použitých obrázků, které jsou v podsložce `img`.
- `gform_lib` – tato složka obsahuje všechny zdrojové soubory knihovny.
- `gform_demo` – v této složce se nacházejí všechny zdrojové soubory ukázkové webové aplikace.

V kořenové složce se dále nachází Bashový skript `compile.sh`, který zkompiluje knihovnu nástrojem Apache Maven, JAR soubor s knihovnou překopíruje do složky
`gform_demo/lib` a přeloží ukázkovou aplikaci nástrojem Apache Maven. Ve složce `gform_demo/target` vznikne WAR soubor s ukázkovou aplikací, který je možný nasadit na server.