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

# Diplomová práce

# Formuláře generované z ontologií

Plzeň, 2016 David Košek

# Prohlášení

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

V Plzni dne 11. května 2016

David Košek

# Poděkování

Touto cestou bych rád poděkoval Ing. Petru Včelákovi za odborné vedení diplomové práce, konzultace, cenné podněty a připomínky, které vedly ke zdárnému vypracování.

# Abstrakt

Diplomová práce se zabývá generováním formulářů z ontologie. Součástí práce je seznámení s oblastí RDF a OWL. Analýza se věnuje nalezení vhodných technologií pro generování formulářů se zaměřením na open source produkty. Dále seznámení s požadavky kladené na formuláře dle různých úhlů pohledu. Z analýzy přecházím do praktické části, kde jako první navrhnu vhodný způsob popisu v RDF. Poté implementuji nástroj pro generování formulářů v HTML a XML. Po vytvoření výsledné aplikace ji otestuji na ontologiích projektu MRE a zhodnotím dosažené výsledky. V závěru shrnu práci a navrhnu možné rozšíření.

# Abstract

The master thesis deals with generating forms of ontology. The thesis includes introduction to RDF and OWL fields. The analysis describes finding suitable technologies for generating forms with focus on open source products, and also introduction to requirements for forms from different points of view. From the analysis I continue to practical part where I propose suitable method of description in RDF. Afterwards, I implement tool for generating forms in HTML and XML. After creating the final application, I will test it on ontologies of MRE project and I will evaluate gained results. In the conclusion I will make summary of the thesis and I will propose possible extension.

# Obsah

# 1 Úvod

Diplomová práce se zabývá vývojem nástroje pro generování HTML[1] formulářů z ontologie umožňující obecný popis formuláře. Nástroj bude otestován nad existující databází projektu MRE[2].

Cílem práce je navrhnout a implementovat nástroj pro automatické vygenerování HTML formuláře včetně zpracování zadaných hodnot. Jednou částí nástroje bude zpracování obecného popisu formuláře do podoby pro vygenerování formuláře. Druhou částí bude samotné vygenerování HTML kódu do JSP[3] stránky s možností uložení zadaných hodnot pomocí aplikačního serveru. Nástroj bude navržen podle požadavků jednoho z vedoucích MRE systému. Součástí práce je rovněž analýza a výběr vhodných prostředků pro tvorbu nástroje.

Motivací k této práci je především usnadnění a automatizace pro vytváření a obsluhu formulářů pro projekt MRE, který je vyvíjen pod záštitou ZČU[4] na Katedře informatiky a výpočetní techniky. Zmíněný projekt v současnosti nevyužívá žádný obdobný nástroj. Nástroj bude přínosem především ve spolupráci s projektem MRE.

V následujících kapitolách se nejprve seznámíme s požadavky na nástroj včetně funkcionality. Následně uvedeme systémy pro reprezentaci ontologií. Poté provedeme analýzu vhodných prostředků pro vytvoření nástroje. Na základě dosud získaných informací zdůvodním výběr použitých prostředků. Hlavní těžiště této práce je v návrhu a implementaci samotného nástroje pro generování formuláře, který bude splňovat veškeré požadavky formulované v seznamu požadavků. Nakonec zhodnotíme dosažené výsledky a shrneme práci.

Přílohou diplomové práce je uživatelská dokumentace k nástroji. Kompletní zdrojové kódy i elektronická podoba tohoto dokumentu je rovněž uvedena ve formě přílohy na CD-ROM.

[1] HTML – HyperText Markup Language
[2] MRE – Medical Research & Education
[3] JSP – JavaServer Pages
[4] ZČU – University of West Bohemia in Pilsen

# 2 Ontologie a RDF

Kapitola popisuje teoretický úvod a jejím záměrem je seznámením čtenáře s danou problematikou.

## 2.1 Sémantický web

Současná síť WWW[1] je jen haldou webových stránek, která neustále roste. Nalézt relevantní informace je stále složitější. Ideou sémantického webu je doplnit síť webových stránek sítí výroků. [1]

Pro sémantický web jsou ontologie základní technologií. Mezi další technologie sémantického webu patří RDF[2], RDF Schéma a dotazovací jazyky nad daty (např. SPARQL).[2]

Sémantický web nemění způsob prezentace a funkcionalitu webových stránek, ale představuje jiný model práce s informacemi. Nemá žádný vliv na komunikaci mezi klientem a serverem.[3]

„*Sémantický Web je označení pro skupinu technologií, které umožňují informace na internetu formulovat tak, aby byly srozumitelné nejen pro lidi, ale i pro stroje, tedy hlavně software na nich běžící. U zrodu nestál nikdo jiný než Tim Berners-Lee a konsorcium W3C, které je dodnes hlavním tahounem vývoje v této oblasti. Sémantický Web není konkurencí současného webu, nýbrž jeho doplňkem, zlepšujícím využití potenciálu, který sítě typu internetu mají.*“ [4]

## 2.2 Ontologie

Ontologie slouží k popisu lidského zájmu/světa. Takováto oblast pak obsahuje jednotlivé třídy, které jsou propojeny relacemi. Objekty a jejich propojení ontologie popisuje pomocí 4 prvků: jedince, třídy, atributu a vazby. Někdy se uvádí také pátý prvek - událost.[5]

[1] WWW - World Wide Web
[2] RDF - Resource Description Framework

Pro pojem ontologie existuje velké množství definic. Uvedeme si dvě nejrozšířenější. První je od T. Grubera a druhá od W. Borsta, který definici upravil.[6]

1. *„Ontologie je explicitní specifikace konceptualizace"* (T. Gruber, 1993)
2. *„Ontologie je formální, explicitní specifikace sdílené konceptualizace"* (W. Borst, 1997)

V obou zmíněných definicích se vyskytuje slovo *konceptualizace*. Konceptualizací je myšlen systém pojmů, kterými lze popsat (modelovat) určitou část reálného světa. Dále z definic vychází, že ontologie musí být specifikována *explicitně*. Je tedy nutné jednoznačně definovat typ konceptu a podmínky jeho použití. V druhé definici je kladen důraz na *formální* specifikaci. Je tedy nutné užít konkrétní jazyk s přesně definovanou syntaxí, pro možné strojové zpracování. Další podmínkou vycházející z definice je, aby ontologie byla *sdílená*. To znamená, že nejde o individuální znalosti, ale o znalosti určité skupiny lidí.[6]

Cílem ontologie je definovat společné, jednotné chápání určité třídy pojmů. Je zřejmé, že „ jednotnosti" se snáze dosáhne v určité uzavřené oblasti - doméně. Proto dnes existuje poměrně velké množství tzv. doménových ontologií.[7]

### 2.2.1 Účel ontologií

Základní způsoby využití ontologií jsou obvykle [8]:

- podpora porozumění mezi lidmi (např. experty a znalostními inženýry),
- podpora komunikace mezi počítačovými systémy,
- usnadnění návrhu aplikací orientovaných na znalosti.

Ve všech zmíněných bodech lze ontologii uplatnit v mnoha aspektech různých oblastí a úloh (např. znalostní management, elektronické obchodování, zpracování přirozeného jazyka, atd.).[6]

### 2.2.2 Typy ontologií

Ontologie lze rozdělit do skupin podle různých hledisek. Zde si stručně uvedeme hlavní dimenze členění.

#### Členění podle paradigmat

Ontologie nejsou v informatice něčím novým, informační zdroje s podobnou strukturou se už dříve vyskytovaly v různých kontextech. Proto se tato problematika rozpadá na tři hlavní oblasti zmíněné v následujících bodech.[8]

#### Terminologické ontologie

Terminologické, často nazývané také jako lexikální ontologie lze přirovnat k tezaurům, využívaných v knihovnictví a dalších odvětvích orientovaných na textové zdroje. Hlavním rysem jsou termíny, které nejsou dále formálně definovány. Používané relace v těchto ontologiích mají především taxonomický charakter. Nejznámější terminologická ontologie je *WordNet*.[6]

#### Informační ontologie

Informační ontologie jsou rozvinutím databázových konceptuálních schémat. Hrají roli nadstavby nad primárními (např. relačně-databázovými) zdroji, pro které zabezpečují např. konceptuální abstrakci potřebnou pro pojmové dotazování a vyšší úroveň kontroly integrity než běžné nástroje.[6]

#### Znalostní ontologie

Ontologie navazují na výzkum v rámci umělé inteligence v oblasti reprezentace znalostí. Zde jsou ontologie chápány jako logické teorie. Vazba na další reálné objekty je relativně volná. Třídy a relace jsou definovány pomocí formálního jazyka.[6]

### Členění podle předmětu formalizace

Jedná se o nejběžnější způsob členění (řada variant navržená více autory), proto si zde uvedeme pouze hlavní typy.[8]

### Doménové ontologie

Doménové ontologie jsou nejpoužívanějším typem. Vždy se zaměřují na určitou specifickou oblast, kterou méně či více ohraničují (mohou existovat doménové ontologie popisující celou fyzikální vědu, nebo ontologie popisující pouze Newtonovy zákony).[6]

### Generické ontologie

Generické ontologie se snaží o zachycení obecných zákonitostí, které platí napříč věcnými oblastmi. Takovým příkladem může být zachycení zákonitosti času. Můžeme se setkat s výslovně vyčleňující ontologií tzv. *vyšší úrovně*, která se snaží o zachycení nejobecnějších pojmů a vztahů.[6]

### Úlohové ontologie

Úlohové ontologie se na rozdíl od ostatních nesnaží zachytit znalosti ze světa, takové jaké jsou, ale zaměřují se na jejich odvozování. Typickým příkladem může být diagnostika, nebo plánování. Někdy jsou tyto ontologie označovány jako generické modely znalostních úloh a metod jejich řešení.[6]

### Aplikační ontologie

Aplikační ontologie jsou nejvíce specifickým typem. Zpravidla zahrnují doménovou i úlohovou část (tím automaticky i generickou část). Jde o kombinaci modelů pro konkrétní aplikaci.[6]

#### Členění podle míry formalizace

Formalizace je do jisté míry definiční vlastností ontologie, ale smysluplné využití mají i zcela neformální či semi-formální ontologie. Zpravidla jde o glosáře (slovníky), kde jsou vysvětleny jednotlivé pojmy přirozeným jazykem. Ontologie vyjádřené ve formálním jazyce, lze dále rozšiřovat podle vlastností daného jazyka (např. úplnost a rozhodnutelnost - vychází z vlastností logického kalkulu). Většina formálních ontologií svým způsobem obsahuje také ontologii neformální.[8]

### 2.2.3 Struktura ontologie

Ontologií existuje celá řada typů, ale základní struktura znalostních ontologií bývá vždy obdobná. Základní součásti struktury ontologií jsou [6]:

- třídy,
- instance a individua,
- relace,
- omezení slotů,
- primitivní hodnoty a datové struktury,
- axiomy a odvozovací pravidla.

#### Třídy

Třídy jsou základním stavebním kamenem znalostních ontologií, které tvoří uspořádanou stromovou hierarchii.

Tyto třídy jsou podobné třídám, které známe z oblasti objektově orientovaných programovacích jazyků, například dědičností, ale také se liší např. nezahrnují procedurální metody, možnosti vícenásobné dědičnosti.[6]

V některých zdrojích jsou třídy zaměňovány za termín *koncept* (tedy pojem).

### Instance a individua

Individua nejsou pouze objekty, ale mohou být chápány také jako instance. Ty jsou na rozdíl přímo spojeny k nějaké třídě.

Individuum reprezentuje konkrétní reálný objekt světa. Jedná se o objekt z množiny, které lze označit pomocí třídy. Individuum může být do ontologie vloženo i bez příslušnosti k nějaké třídě, avšak tato možnost není některými ontologickými jazyky podporována.[9]

### Relace

Relace v ontologiích definují vztahy mezi jednotlivými instancemi a individui. Můžeme se setkat s tzv. sloty, které se většinou vztahují pouze na binární relace. To jsou relace pouze mezi dvěma objekty. Takové relace nejsou závislé na konkrétní třídě.

Jako zvláštní typ relace bývají chápány *funkce*. V souladu s běžným matematickým chápáním jde o relace, u nichž je hodnota $n$-tého argumentu jednoznačně určena předchozími $n$-1. Funkční slot se také označuje jako *atribut*.

Stejně jako u tříd je možnost tvorby hierarchického uspořádání.[10]

Sloty můžou nabývat různých omezeních vlastností. Vlastnosti popisují typy hodnot, množství hodnot (kardinalita) tj. kolik hodnot současně může instance nabývat a jiné rysy hodnot, které můžeme slotu přiřadit. Často je nutné omezit hodnoty slotu, k tomu slouží zmíněná kardinalita, která nám určuje, kolik hodnot může slot mít (většinou nabývá hodnot - *single, multiple, minimum a maximum*).[10]

Například:

- pacient podstoupil typ vyšetření (buď vyšetření zraku, či sluchu, nebo jiné) - single, ale může být z více *lékařských středisek* (multiple),
- lékaři mají více pacientů. (multiple).

#### Primitivní hodnoty a datové struktury

V předchozích bodech jsme si popsali relaci jako vztah mezi objekty. Toto tvrzení nebylo úplně přesné, protože argumenty relací mohou být i primitivní datové hodnoty, které žádnému objektu neodpovídají. V tomto případě se jedná o takzvané*dato-typové sloty*. Obor hodnot takového slot bývá definován pomocí základních datových typů (string, integer, ...), intervalem, nebo výčtem.[6]

#### Axiomy a odvozovací pravidla

Ontologie obsahuje výrazy, které explicitně vymezují příslušnost k určitým třídám a relacím. Do ontologie je možné zařazovat také logické formule. Ty nám vyjadřují například ekvivalenci tříd či relací, disjunkci tříd, či rozklad třídy na podtřídy apod. Takovéto formule jsou označovány za *axiomy*, či *pravidla*.[6]

#### Ontologické jazyky

Ontologie musí být zapsána ve strojově čitelném jazyce a může být reprezentována více jazyky (formální, semi-formální, neformální). Zaměříme se na jazyky, které souvisí s webovými technologiemi. Aktuálně se hlavní vývoj ubírá oblastí formálních jazyků ontologií sémantického webu. Tam patří a budou nás zajímat jazyky zejména rodiny OWL[3] a RDF včetně slovníku RDFS[4]. RDF a OWL jsou považovány za základní technologie sémantického webu.[6] Staršími jazyky jako jsou SHOE[5] a Ontobroker se zabývat nebudeme.

### 2.2.4 OWL

Jazyk OWL je základním jazykem sémantického webu. Tento jazyk vznikl ze staršího jazyka DAML+OIL, který sloužil jako výchozí bod. OWL je jazyk pro popis tříd a relací, který rozšiřuje původní RDFS a dokáže vytvořit komplexní vztahy i mezi třídami. Třídy lze definovat uzavřeně a určit tak,

[3]OWL - Web Ontology Language
[4]Resource Description Framework Schema
[5]SHOE - Simple HTML Ontology Extension

které prvky patří které třídě. OWL nabízí možnosti kombinování, oddělování či skládání, jazykové konstrukce, anonymní třídy a další. OWL používá RDF/XML syntaxi. OWL využívá RDF a RDFS slovníků, které rozšiřuje o vlastní výrazy. [12]

Ontologický jazyk dělá ontologickým jazykem především formální sémantika. Hlavní vlastností je, že popisuje smysl vědomosti přesně. [3]

Ukázka OWL kódu (Zdroj. kód 2.1), která říká, že třída *Vyšetření* je podtřídou *Pacient*, a každá její instance musí být spojena relací *podstoupil_vyšetření* s alespoň 1 instancí třídy *preventivní_prohlídka*.

```
<owl:Ontology rdf:about="http://www.example.org/patient">
  <rdfs:comment>An examination OWL ontology</rdfs:comment>
  <owl:class rdf:ID="Vyšetření"/>
  <rdfs:subClassOfrdf:resource="Pacient" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="podstoupil_vysetreni
               "/>
            <owl:someValuesFrom rdf:resource="
               preventivní_prohlídka"/>
        </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:label>Patient examination</rdfs:label>
</owl:Ontology>
```

Zdrojový kód 2.1: Ukázka OWL.

Nejen pro vývoj jazyka OWL, ale i pro ostatní jazyky je nutné hledat kompromis mezi efektivitou odvozování a vyjadřovací silou. Proto organizace W3C navrhla tři varianty právě podle hledaného kompromisu pro tento jazyk.[3]

1. OWL Lite - slouží především na podporu uživatelům, kteří potřebují jednodušší klasifikační hierarchie a jednodušší omezení. Dedukce má nižší formální složitost. Nejjednodušší verze jazyka,
2. OWL DL - už složitější klasifikační hierarchie. Dedukce má většinu problémů rozhodnutelnou. Není plně kompatibilní s RDF,
3. OWL Full - nabízí celou výrazovou sílu a celé RDF. Dedukce je často nerozhodnutelná. Plně kompatibilní s RDF.

### 2.2.5 OWL2

OWL 2 je ontologický jazyk pro vytváření sémantického webu s formálně definovaným významem. OWL 2 ontologie poskytují třídy, vlastnosti, individua a datového hodnoty jsou uloženy jako dokumenty sémantického webu. OWL 2 má nové rysy oproti standardnímu OWL (bohatší datové typy a datové rozsahy, omezení kardinality a další). OWL 2 rozšiřuje původní OWL a je zpětně kompatibilní.[11]

## 2.3 RDF

RDF (Resource Description Framework) je popis zdrojů vypracovaný organizací konsorciem W3C[6], původně navržený pro reprezentaci metadat. Lze využít jako způsob modelování informací v různých typech syntaxe. Jedná se o obecný rámec informací popisující zdroj tak, aby byl jeho popis čitelný strojově, tak lidsky. RDF je standardizovaný formát a jeho hlavním účelem je vyjadřovat informace o zdrojích na webu. RDF lze popsat i pomocí orientovaného grafu. Zdrojem může být kterýkoliv zdroj, který lze jednoznačně identifikovat pomocí URI[7]. Tato URI nám určuje, o jaký jednoznačný zdroj se jedná. [13]

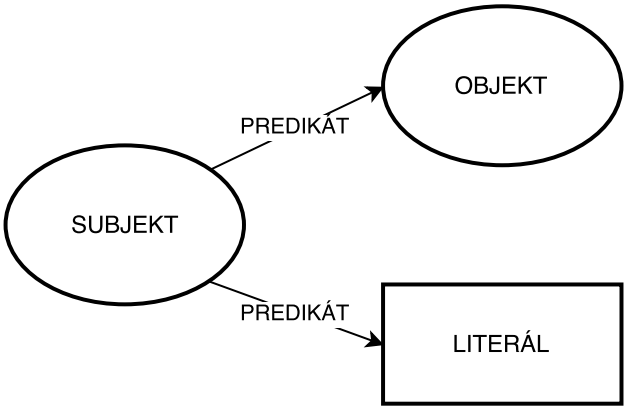
### 2.3.1 RDF model

Modelem RDF je reprezentována sémantická struktura dat. Lze jej přirovnat k prostředku jako základní rámec reprezentace informací na webu. K popisovanému zdroji se snažíme přiřadit výraz ve tvaru subjekt - predikát - objekt. Takovému výrazu se používá termín trojice.[13]

RDF popisuje určitý zdroj, ten má přiřazené určité vlastnosti a tyto vlastnosti mají přiřazené hodnoty. Subjekt nám určuje, o jaký zdroj se jedná, predikát určuje charakter a zároveň vzájemný vztah mezi objektem a subjektem.[13] Například výrok *Pacient má zlomenou ruku*, pokud tento výrok chceme pospat trojicí, tak *Pacient* lze přirovnat k subjektu, *má zlomenou* k predikátu a *ruku* k objektu.

[6] W3C - World Wide Web Consortium
[7] URI - Uniform Resource Identifiers

RDF model vyjádřený orientovaným grafem (2.1). [14]




Obrázek 2.1: Zobrazení dvou trojic v RDF.

### 2.3.2 Zdroje a URI

Základní ideou modelu RDF je vhodný způsob popisu zdrojů v termínech vlastností a jejich hodnot a identifikace věcí prostřednictvím webových identifikátorů, nazývaných jednotné identifikátory zdroje (URI). Model RDF pracuje s URI, které lze najít na všech místech trojce (tj. subjektu, objektu tak predikátu). Objekt může nabývat přímé textové hodnoty tj. literál, nebo může být objekt dalším zdrojem, či listem vlastností.[15]

Zdroj jako takový, je entita, kterou lze popsat RDF výrazem. Příkladem zdroje můžou být pacienti, vyšetření, nebo nemoci.[15]

### 2.3.3 RDF syntaxe

RDF model můžeme formulovat jako trojici popsanou v předchozím bodě. RDF lze zapisovat pomocí XML[8] (tzv. RDF/XML). Vedle toho existuje celá řada používaných syntaxí.[16]

[8]XML - Extensible Markup Language

Uvedeme si jednoduché srovnání syntaxí.[16]

- RDF/XML
  - povinně zpracovávaná syntaxe,
  - umožňuje tvorbu RDF nástroji XML.
- N-Triples
  - syntaxe založená na plných URI,
  - dobře se parsuje,
  - z důvodů plných URI, velká míra komprese,
  - možnost paralelního zpracování.
- Turle
  - dobře lidsky čitelné,
  - využití při dotazování.
- Notation3 (N3)
  - strojové zpracování,
  - nadstavba (pravidla, integritní omezení...).

Ukázka kódu (Zdroj. kód 2.2) ilustruje použití RDF k popisu vyšetření pacienta.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns
   \#"
         xmlns:ds="http://mre.kiv.zcu.cz/ontology/dasta.owl#">
  <ds:MedicalExamination rdf:about="http://mre.kiv.zcu.cz/id/ex
     #jpGId2zrLb43jjNkWaTjHqbNB0oYS2UYYe4Pn9iT">
       <ds:patientID>12345</ds:patientID>
       <ds:datetimeEvent>2016-12-31T00:00:00+0100</ds:
          datetimeEvent>
       <ds:reportText>Zpráva o vyšetření...</ex:result>
       <ds:hasDiagnosis rdf:resource="http://mre.kiv.zcu.cz/
          ontology/dasta.owl#2zrLb43jjNkWaTjHqbNB0oY"/>
  </ds:MedicalExamination>
</rdf:RDF>
```

Zdrojový kód 2.2: Popis vyšetření pacienta v RDF.

### 2.3.4 RDF Schéma

RDF Schéma je součástí specifikací RDF, které rozšiřuje specifikaci o možnosti vytvářet si vlastní ontologie pro popis zdrojů. Je založený na XML a dovoluje blíže specifikovat vlastnosti k jednotlivým RDF objektům. K definování objektů, tříd a vlastností využívá tzv. slovníky.[13]

Slovník, aby byl užitečný musí být chápán ve stejném kontextu jak autorem, tak čtenářem dané informace. Slovníků postavených na RDFS existuje celá řada, jedním z nejznámějších je FOAF[9]. FOAF je určený k popisu lidí, aktivit a vztahů k ostatním lidem a objektům. Definované vlastnosti jsou například jméno, nebo emailová adresa. Pokud nám žádný z dostupných slovníků nevyhovuje, RDFS umožňuje definovat slovníky nové. RDF slovník je pouze množina URI referencí ve jmenném prostoru RDF.[13]

Jednotlivé zdroje mohou být rozděleny do více skupin, ty se v RDFS označují jako třídy. Třída nedefinuje objekt. Instancí třídy je pouze bezstavový zdroj reprezentovaný URI.[13]

### 2.3.5 Jmenné prostory

RDF model poskytuje slovníky čitelné jak pro člověka, tak pro stroj. Model jednoznačně identifikuje vlastnosti s použitím jmenných prostorů jejich slovníků. Pro dosažení stručnější zápisu se používá pro každý jmenný prostor jeho kvalifikované jméno tvořící prefix popisovaných prvků trojic. Například RDF slovník má prefix *rdf:* a má URI `http://www.w3.org/1999/02/22-rdf-syntax-ns#`.[15]

## 2.4 SPARQL

V souvislosti s RDF vzniklo několik dotazovacích jazyků pro manipulaci s RDF modely/grafy a k dotazování nad RDF grafy. SPARQL (rekurzivní zkratka SPARQL Protocol and RDF Query Language) je uznáván jako jeden z klíčových jazyků, který je standardizován pod konsorciem W3C. Ve srovnání s dotazovacím jazykem SQL[10] používaným v relačních databázích

[9] FOAF - Friend Of A Friend

[10] SQL - Structured Query Language

je určitá podobnost (např. u klíčových slov SELECT, WHERE, FROM). Hlavní rozdíl je v tom, že za základní informační prvek v RDF je považována trojice, a proto se také ve SPARQL dotaz skládá z množiny trojic.[17]

Pomocí SPARQL lze získávat informace z RDF grafů v podobě URI, zdrojů a znaků. Dále je možné rozdělit podgrafy RDF a konstruovat nové RDF grafy, které vznikly ze samostatného dotazu.[18]

# 3 Analýza dostupných řešení

V této části popíšeme frameworky, které mohou být užitečné při realizaci práce se zpracováním RDF dat, tak s generováním HTML kódu na základě popisu. Zaměříme se směrem na volně použitelné frameworky.

## 3.1 Existující řešení generování formulářů

Než začnu s analýzou dostupných frameworků pro svůj generátor, projdu si několik hotových řešení pro tvorbu (generování) webových formulářů.

#### Telosys Tools

Jednoduchý generátor kódu založený na podpoře Eclipsu. Od verze 2.0 je nástroj nezávislý a může být použit ke generování jakéhokoliv druhu kódu (Java, PHP, JavaScript, HTML, atd.). Telosys je zdarma a je open-source. Telosys využívá ke generování nastavitelné šablony a databázové úložiště. [19]

#### Orberon

Orberon se zaměřuje na konverzi XForms formulářů do HTML a JS. Je považován za jeden z největších řešení webových formulářů založené na XForms. Definice formulářů je možné vkládat přímo do JSP stránek nebo do XHTML. Pro běh vyžaduje servletový kontejner (např. Apache Tomcat). Orberon je dostupný pod licencí LGPL[1]. [20]

#### PHP Database Form

PHP Database Form dokáže generovat formulář přímo z databázové tabulky. Na základě dotazu výběru sloupců se vygeneruje formulář. Před zobrazením

[1]LGPL - GNU Lesser General Public Licens

formuláře se můžou nastavit pravidla pro kterýkoliv řádek formuláře. Framework mimo jiné i nabízí několik desítek grafických témat, samozřejmě se dá nastavit i vlastní vzhled.[21]

Rozsáhlejší webové frameworky jako Spring nebo Grails mají také podporu pro formuláře. U těchto frameworků je nutné formulář udělat manuálně, navíc je potřeba modelová třída. Formulář je rozdělen mezi více souborů z důvodu MVC, což způsobuje nepřehlednost.

## 3.2 Zpracování RDF

Pro zpracování RDF dat existuje již několik známých implementacích. Uvedeme si pouze ty nejznámější, ze kterých si při implementaci zvolíme nejvhodnější pro náš účel. Doporučeným řešením od vedoucího diplomové práce je framework Jena pro snadnou práci s daty a obsáhlé dokumentaci.

### 3.2.1 Apache Jena

Jena je open-source framework pro práci s RDF daty. Framework je implementován v programovacím jazyce Java. Poskytuje programová rozhraní a třídy pro tvorbu a manipulaci s RDF. V neposlední řadě nabízí třídy a rozhraní pro manipulaci s ontologiemi (Ontology API[2]). Jena poskytuje ucelené API pro manipulaci dat z RDF grafu a umožňuje i zápis do něj. Grafy jsou prezentovány jako abstraktní model. Tento model může být načtený ze souboru, databáze, URL[3] adresy nebo kombinací zmíněných. Pro dotazování nad těmito daty slouží dotazovací jazyk SPARQL 1.1.[22]

Poslední stabilní verze byla zveřejněna v roce 2015.

### 3.2.2 Sesame

Sesame je open-source framework pro zpracování RDF dat implementový pomocí jazyka Java. Nabízí snadno použitelné API. Zahrnuje odvozování

[2]API - Application Programming Interface
[3]URL - Uniform Resource Locator

a dotazování nad těmito daty. K dotazování nad daty využívá několik dotazovacích jazyků (např. SPARQL, SeRQL).

Sesame dokáže pracovat nad daty uloženými na různých místech, jako např. v relační databázi, v operační paměti a úložišť poskytovaných třetí stranou. Fmarework dále nabízí mnoho nástrojů pro vývojáře pro využití síly RDF a souvisejících norem. Sesame plně podporuje SPARQL 1.1 dotazy a aktualizace jazyků pro expresivní dotazování a nabízí transparentní přístup ke vzdáleným úložištím RDF pomocí stejného APi jako pro lokální přístup. Sesame podporuje všechny formáty hlavního proudu RDF souborů včetně RDF/XML, Turtle, N-Triples, N-Quads, JSON-LD, TriG and TriX.[23]

Poslední stabilní verze byla zveřejněna v roce 2016, a tudíž je považována za nejaktuálnější z uvedených.

### 3.2.3 Shrnutí možností zpracování RDF

Pro zpracování RDF dat jsem uvedl dva nejpoužívanější frameworky. Oba jsou dostupné přes Maven závislosti. Apache Jena oproti Sesamu vyniká svou propracovanou dokumentací a uvedenými tutoriály. Na doporučení vedoucího práce budu v implementaci využívat framework Apache Jena.

## 3.3 Generování HTML formulářů

Pro generování HTML kódu existuje mnoho řešení. Uvedeme si pouze ty, které pracují s programovacím jazykem Java. Při implementaci si vyberu framework nejvhodnější pro náš účel. Při analýze frameworků pro generování musíme myslet především na to, že budu mít nějaký předpis, podle kterého výsledný formulář vygenerujeme.

### 3.3.1 JFresnel

JFresnel je Java knihovna, která implementuje Fresnel specifikaci pro RDF API, jako jsou Jena a Sesame. Knihovna podporuje dotazovací jazyk SPARQL a jednoduché výběrové výrazy. Podporuje analyzovat Fresnel RDF data v rozsahu slovníků Fresnel specifikace.[24]

Poslední stabilní verze byla zveřejněna v roce 2010, ale i navzdory tomu je stále používaná.

### 3.3.2 FreeMarker

FreeMarker je framework založený na šablonách. Má své vlastní Java API, pomocí kterého lze generovat libovolný textový výstup (HTML kód webových stránek) na základě šablony a dat. Šablony se vytvářejí ve specializovaným jazyce FTL[4]. Data do šablony se připravují pomocí programovacího jazyka Java. Jednodušeji řečeno se spojí před připravená šablona s Java objekty a výstupem je text, který může mít podobu HTML.[25]

Nevýhodou pro nás je, že formuláře budou různého typu a pro nás by to znamenalo vytvořit velké množství šablon pro pokrytí, co nejvíce případů.

### 3.3.3 Backbone-forms

Přizpůsobitelný framework pro generování formulářů pomocí jednoduchých schémat. Tento framework vypadá velmi silně a to především díky jeho flexibilitě. Nabízí možnost vnořených formulářů, pokročilé a vlastní editory, vlastní HTML šablony. Nevýhodou pro nás tohoto frameworku je ten, že generuje HTML kód pomocí JavaScriptu a opět je směřován k používání šablon.

### 3.3.4 J2HTML

Jednoduchý, rychlý a plynulý Java HTML5 builder. Nepotřebuje žádné před připravené šablony. Podporuje všechny HTML elementy, včetně nastavení stylů, generování hlavičky, těla nebo celé HTML stránky. Záleží na nás, které části stránky se mají vygenerovat, výstupem builderu je text v podobě HTML. Nevýhodou je nepříliš obsáhlá dokumentace.

[4]FTL - FreeMarker Templetes Languages

### 3.3.5 Shrnutí možností generování HTML

Pro generování HTML kódu jsem uvedl řadu frameworků založených na různých bází. JFresnel nám slouží pouze pro vizualizaci dat. FreeMarker má vlastní Java API a je dostupný přes Maven, ale potřebuje před připravené šablony, které se vytváří ve speciálním jazyce FTL. Backbone-forms je závislí na JavaScriptu a používání před připravených šablon. Další jeho nevýhodou je nedostupnost přes Maven. Framework J2HTML je dostupný přes Maven a nepotřebuje před připravené šablony. Neobsahuje rozsáhlou dokumentaci, ale je velmi intuitivní. Na základě získaných informací ve své implementaci použiji J2HTML.

# 4 Analýza formulářových prvků

V kapitole se seznámíme s tím, jak vytvořit formulář, uvedu si základní prvky formuláře včetně možných atributů a také, jak formuláře použít. Zaměřím se především na prvky a atributy, které budu později používat.

## 4.1 Tvorba formulářů

Pro vytváření HTML5 formulářů se používá pouze několik tagů. Všechny formulářové elementy jsou sjednoceny v jednom párovém tagu `<form>`. Jednotlivá vstupní pole jsou tvořeny především pomocí tagů `<input>`, `<select>` a `<textarea>`. Někdy se můžeme setkat s rozdělením formuláře na více polí pomocí tagu `<fieldset>`.

Samotné HTML k vytvoření funkčního formuláře nestačí. To nám zajišťuje pouze vizuální stránku formuláře. K zprovoznění formuláře je potřeba nastavení URL skriptu, kde se data zpracují. K tomu nám slouží atribut `action`. Pokud `action` není uveden, data se odešlou téže stránce. Dalším důležitým atributem je nastavení metody pro odeslání dat. K tomu se používá atribut `method` a výchozí hodnota je GET. Ta říká, že se data předají jako součást URL adresy. Pokud data chceme odesílat nezávisle, takže nejsou vidět v URL, musíme nastavit metodu POST. Pokud chceme odesílat soubory musíme nastavit atribut `enctype`. Atribut `enctype` nastavuje způsob zakódování dat (např. pro zmíněný soubor se hodnota nastavuje na *multipart/form-data*).

## 4.2 Formulářové elementy a jejich atributy

Vytváří se pomocí tzv. tagů. Ty musejí být umístěny v základním formulářovém elementu `form`.Tagy mohou být párové nebo nepárové. Pro snazší popis jsem si elementy rozdělil do následujících skupin.

### 4.2.1 Vstupní pole

Prvek `input` patří mezi základní formulářové prvky. Jedná se o nepárový tag. Používá se k vytvoření textových polí, přepínacích polí, zaškrtávacích polí. K určení čemu vstupní pole slouží se používá atribut `type`. Tento atribut může nabývat hodnot popsané v tabulce 4.1:

| Type | Popis |
|---|---|
| `text` | Zobrazí se textové pole. |
| `password` | Vlastnostmi stejné jako `text`, pouze znaky se zobrazují jako hvězdičky. |
| `hidden` | Skryté pole (např. s před připravenou hodnotou). |
| `file` | Vstupní pole, jako nabídka pro přidání souboru. |
| `radio` | Přepínací tlačítko. |
| `checkbox` | Zaškrtávací tlačítko. |
| `button` | Vytvoří ze vstupního pole tlačítko. |
| `submit` | Tlačícko pro odeslání dat formuláře. |
| `reset` | Tlačítko pro resetování hodnot formuláře do původního stavu. |
| `image` | Tlačítko s obrázkem. |

Tabulka 4.1: Možnosti typu pro vstupní pole.

Ostatní atributy, které může `input` nabývat jsou závislé na typu vstupního pole. V tabulce 4.2 si uvedu atributy, které lze využít u většiny zmíněných typů.

| Atribut | Popis |
|---|---|
| `text` | Zobrazí se textové pole. |
| `autofocus` | Po načtení formuláře se oblast aktivuje sama. |
| `checked` | Předzvolená hodnota. |
| `disabled` | Deaktivuje oblast, nejde do ní psát ani měnit. |
| `max` | Nastavení horní hranice. |
| `maxlength` | Maximální počet znaků. |
| `min` | Nastavení spodní hranice. |
| `name` | Jméno odesílané s daty ke zpracování. |
| `pattern` | Regulérní výraz pro kontrolu vstupu. |
| `placeholder` | Nápověda k poli. |
| `readonly` | Obsah je určen pouze ke čtení. |
| `required` | Pole je povinné k vyplnění. |
| `step` | Nastavení kroku pro vstupní pole. |
| `value` | Hodnota vstupu. |

Tabulka 4.2: Atributy pro vstupní pole.

### 4.2.2 Select box

Jde o výběr z více možností. Tento tag je párový a pomocí něj lze připravit rolovací pole. Na výběr je jedna z několik možností, každá možnost v menu je tvořena pomocí tagu `option`. Ovšem pomocí atributu `multiple` lze nastavit výběr více možností najednou.

Příklad vytvoření selectu boxu pro výběr pohlaví (Zdroj. kód 4.1):

```
<select name="pohlavi">
       <option value="muz">MUZ</option>
       <option value="zeny">ZENA</option>
</select>
```

Zdrojový kód 4.1: Select box pro výběr pohlaví.

Atributy `select` elementu jsou popsány v tabulce 4.3:

| Atribut | Popis |
|---|---|
| `text` | Zobrazí se textové pole. |
| `autofocus` | Po načtení formuláře se oblast aktivuje sama. |
| `disabled` | Deaktivuje oblast, nejde do ní psát ani měnit. |
| `multiple` | Možnost výběru více hodnot najednou. |
| `name` | Jméno odesílané s daty ke zpracování. |
| `required` | Pole je povinné k vyplnění. |
| `size` | Určuje počet zobrazených řádků pod sebou. |

Tabulka 4.3: Atributy pro select element.

S tímto elementem se pojí tag `option` a `optgroup`. Oba zmíněné jsou párové. První zmíněný element je položka seznamu, která se umisťuje mezi tagy `select`. Tento element může obsahovat dva atributy. Atribut `value`, který určuje hodnotu, která se odesílá jako hodnota daného pole. A atribut `selected`, který předem vybere položku. Druhý zmíněný element `optgroup` se používá jako název. Pomocí atributu `label` se zapíše nadpis jednotlivých položek seznamu.

Atributy pro tag `option`.

| Atribut | Popis |
|---|---|
| `text` | Zobrazí se textové pole. |
| `disabled` | Deaktivuje oblast, nelze vybrat. |
| `label` | Zobrazovaný text. |
| `selected` | Předzvolená hodnota. |
| `value` | Hodnota vstupu. |

Tabulka 4.4: Atributy pro option element.

### 4.2.3 Textové pole

Html tag `textarea` se používá pro zapisování delších textů. Jedná se o párový tag, který jako jeden z mála nemá atribut `value`, `textarea` obklopuje text umístěny mezi jeho tagy. Pomocí atributů lze definovat nejčastěji počet

znaků na řádek a počet viditelných řádků. Přehled všech atributů je popsán v tabulce 4.5:

| Atribut | Popis |
|---|---|
| `text` | Zobrazí se textové pole. |
| `autofocus` | Po načtení formuláře se oblast aktivuje sama. |
| `cols` | Počet znaků na řádek. |
| `disabled` | Deaktivuje oblast, nejde do ní psát ani měnit. |
| `maxlength` | Maximální počet znaků. |
| `name` | Jméno odesílané s daty ke zpracování. |
| `placeholder` | Nápověda k poli. |
| `readonly` | Obsah je určen pouze ke čtení. |
| `required` | Pole je povinné k vyplnění. |
| `rows` | Počet řádek.. |
| `wrap` | Nastavuje zalamování slov. |

Tabulka 4.5: Atributy pro textarea element.

### 4.2.4 Ostatní elementy

#### Label

Label neboli popisek pole, či štítek. Většinou se vyskytuje nad nebo vedle políčka, ke kterému se vztahuje. Jeho hlavní výhodou je aktivace pole, pokud se klikne na popisek. K tomu je zapotřebí nastavit atribut `for` a u vstupního pole se stejnou hodnotou atribut `id`. Vlastní text popisku se zadává jako obsah elementu. Popisek patří mezi párové tagy.

#### Button

Button se používá jako tlačítko. Hlavní jeho výhodou oproti zmíněnému tagu `input` typu button (v praxi se chová hodně podobně) je ta, že se do něj dá vložit libovolný HTML kód. Tag je párový a vložený kód může reprezentovat obrázky, nadpisy, atd.

Atributy pro tag button:

| Atribut | Popis |
|---|---|
| `text` | Zobrazí se textové pole. |
| `autofocus` | Po načtení formuláře se oblast aktivuje sama. |
| `disabled` | Deaktivuje oblast, na tlačítko nepůjde kliknout. |
| `name` | Jméno odesílané s daty ke zpracování. |
| `type` | Typ tlačítka (submit, reset, button). |
| `value` | Hodnota vstupu. |

Tabulka 4.6: Atributy pro button element.

Nejdůležitějším atributem je `type`. Výchozí hodnota atributu je submit (odesílací tlačítko formuláře).

### Datalist

Datalist je našeptávací datová struktura provázaná s nějakým tagem `input`. Vzdáleně se podobá chování tagu `select`. Rozdíl je v tom, že uživateli umožňuje vyhledávat nad danými možnostmi. Datalist má pouze jeden atribut a to je `id`. Ten je svázán s atributem `list` v input tagu.

# 5 Popis nástroje pro generování formuláře

Kapitola stručně popisuje základní požadavky na nástroj pro generování formuláře včetně jeho zpracování. Jejím záměrem je seznámení čtenáře s formulací požadavků.

Nástroj je rozdělen na dvě části. První z částí je zpracování konfiguračního souboru umožňující obecný popis do popisu pro vygenerování výsledného HTML formuláře. Druhá část se zabývá samostatným generováním kódu včetně zpracování a uložení dat.

Uživatel na začátku dostane možnost nahrání libovolného počtu konfiguračních souborů ve formátu *.n3. V případě, že uživatel žádné konfigurační soubory načíst nechce, použijí se výchozí. Následně se soubory zpracují a vrátí uživateli na výběr seznam formulářů dostupných z konfiguračních souborů. Krom toho bude mít uživatel možnost zadat jazykové možnosti, které se budou automaticky nabízet u všech textových vstupů. Po vybrání formuláře dojde na základě popisu k jeho vygenerování. Po vyplnění formuláře dojde k jeho obsloužení a uložení do příslušné podoby. Jméno formuláře, použitou metodu formuláře, URL adresu obsluhy formuláře atd. bude načítáno z konfiguračního souboru. V případě některé chybějící informace se použije výchozí hodnota.

### Formulářové prvky a jejich atributy

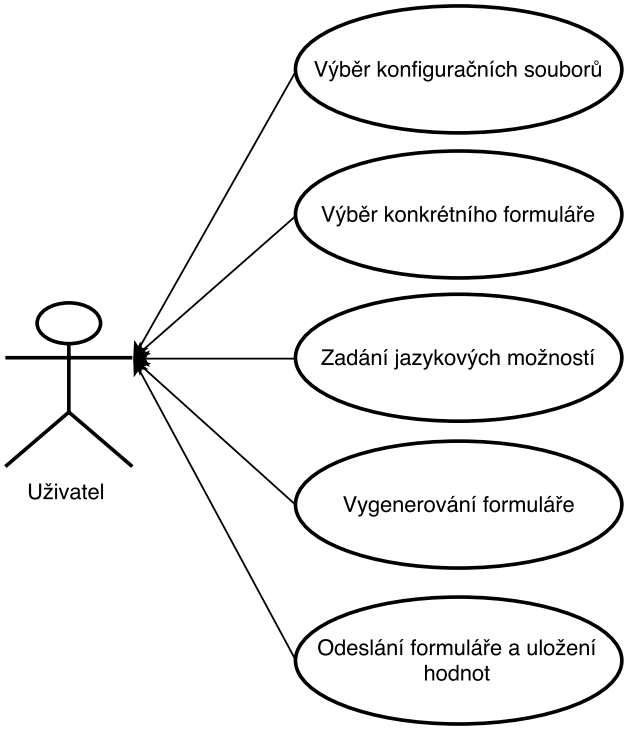
Nástroj bude umět generovat různá vstupní pole (input), výběr z více možností (select), rozsáhlé vstupní pole (textarea), štítek (label, zobrazovaný před nebo nad polem, ke kterému se vztahuje) a našeptávací datovou strukturu (datalist). Ke všem prvkům lze definovat atributy, jak obecné (*id, name,* ...), tak specifické dle jejich funkcionality.

### Obsluha formuláře

Pro obsluhu bude sloužit tlačítko formuláře, na které bude reagovat registrovaný servlet s připravenými metodami pro zpracování (*get, post*). Obě

z metod budou fungovat na stejném principu (vytvoření modelu, zpracování dat a uložení v požadovaném formátu).

Na závěr popisu nástroje jsem se rozhodl pro lepší představu zobrazit diagram použití (Obr. 5.1) pro uživatele.




Obrázek 5.1: Diagram použití pro uživatele.

# 6 Návrh popisu formulářů v RDF

V této části práci čtenáře seznámíme s návrhem popisu pro výsledný formulář pomocí RDF.

Popis formuláře navrhnu tak, aby respektoval HTML, snadno čitelný a rozšiřitelný. Při návrhu mám k dispozici existující popis dat v RDF a používaných ontologií z projektu MRE. Tento popis mi bude sloužit jako výchozí bod, i když s popisem formulářů přímo nesouvisí.

## 6.1 Jmenný prostor

Pokud chceme popisovat nějaké informace ve formátu RDF, setkáme se často s prefixy, které nám značí jednotlivé jmenné prostory. Můj navržený prefix pro popis formuláře je **form** a po domluvě s vedoucím diplomové práce je URI jmenného prostoru `http://mre.kiv.zcu.cz/ontology/form.owl#`. Pokud budu popisovat nějakou informaci pro formuláře, tvrzení začíná prefixem a následuje názvem vlastnosti. Například **form:nameForm** říká, že se jedná o název formuláře. Hodnota vlastnosti je uvedena jako literál, tedy například **form:nameForm 'name'**.

## 6.2 Definice formuláře

Tvrzení se skládá z námi známé trojce subjektu, predikátu a objektu. Jestli tvrzení popisuje formulář zatím nevíme, tudíž si musíme definovat, kdy tvrzení bude popisovat formulář. Prvotní a jediný návrh, co mě napadl bylo, když predikát bude **rdf:type** a objekt **form:Form**, na subjektu tento moment nezáleží. Pokud takové tvrzení najdeme v konfiguračním souboru, víme, že jsem narazil na popis formuláře.

Takové tvrzení by pro popis formuláře nestačilo. Pokud budeme vycházet z existujícího konfiguračního souboru z projekt MRE, tak lze vidět tvrzení (trojici) a k ní další vlastnosti už jako dvojice (vlastnost, hodnota).

Pro popis formuláře jsem navrhl několik základních vlastností, které jsou doporučené pro popis formuláře. Navržené vlastnosti později generátor bude

hledat, nejsou povinné, ve většině případech existují k těmto vlastnostem výchozí hodnoty nastavené v generátoru, ale jsou doporučené.

Navrhnuté základní vlastnosti pro popsání formuláře.

| Název | Typ hodnoty | Popis |
|---|---|---|
| form:nameForm | literál | Název formuláře |
| form:action | literál | Nastavuje formuláři adresu pro zpra-<br>-cování hodnot |
| form:method | literál | Nastavuje metodu odeslání dat<br>(POST/GET) |
| form:outputName | literál | Název výstupního souboru |
| form:saveAs | literál | Řetězec formátů, ve kterém se mají<br>hodnoty uložit |
| form:showProperties | zdroj (list) | List viditelných vlastností reprezen-<br>-tující prvky formuláře |
| form:hideProperties | zdroj (list) | List skrytých vlastností reprezen-<br>-tující prvky formuláře |

Tabulka 6.1: Základní navržené vlastnosti.

Jak bylo řečeno žádný z těchto navržených vlastností není povinný. Kromě případu **form:showProperties** a **form:hideProperties**, jsou připravené výchozí hodnoty. Pokud jsou dvě zmíněné vlastnosti vynechány, nebo jsou prázdné, formulář se stejně vygeneruje, ale nebude mít žádné formulářové prvky.

Ukázka (Zdroj. kód 6.1), jak takový popis formuláře může vypadat.

```
:TestBasicForm rdf:type form:Form ;
       form:nameForm "Test basic form";
       form:action "/genform/form/processsss";
       form:method "post";
       form:outputName "testBasicForm";
       form:group :mainGroup ;
       form:saveAs "rdf";
       form:showProperties (
         ds:name
         ds:surname
   ) .
```

Zdrojový kód 6.1: Popis formuláře podle navržené struktury.

## 6.3 Formulářové prvky

Nyní máme navrženou strukturu pro popis formuláře jako takového. Na základě takového popisu se lze vygenerovat základní formulář, ale bez prvků formuláře. Dále navrhnu, jak popisovat jednotlivé formulářové prvky. Víme, jaké vlastnosti jsou v listech **form:showProperties** a **form:hideProperties**, tak teď k nim stačí vyhledat odpovídající popis.

Opět si definujeme, jak takovou informaci o dané vlastnosti najít. Při návrhu jsem se inspiroval JFresnelem, když predikát pro vlastnost bude opět **rdf:type** a objekt **form:Format**, na subjektu opět nezáleží. To nám, ale daný prvek nespecifikuje. Proto musím návrh více upřesnit. Takže k návrhu přidám ještě dvojici a to vlastnost **form:propertyFormatDomain**, kde hodnota bude URI adresa prvku v listu vlastností. Tím určím jednoznačně hledaný prvek.

Pokud v konfiguračním soubor najdeme tvrzení podle tohoto návrhu, stále nemáme vyhráno. Máme už popsané informace o formuláři, o tom, jak nalézt formulářové prvky, ale potřebujeme ty prvky stále ještě popsat. Hodnota prvku může být literálem, nebo zdrojem. O zdroj půjde pouze v pár případech, bude-li formulářový element:

1. radio group,

2. select menu,
3. datalist.

ostatní vlastnosti budou mít přímou hodnotu.

Pouze některé vlastnosti prvku jsem navrhl jako povinné. V tabulce 6.2 je přehled těchto vlastností i s vysvětlením.

| Název | Typ hodnoty | Popis |
|---|---|---|
| form:elementType | literál | Určuje typ formulářového prvku |
| form:group | zdroj | Určuje skupinu |

Tabulka 6.2: Základní navržené vlastnosti.

Než navrhnu popis obecných atributů, specifikuji více možnosti, které můžou nabývat u vlastnosti **form:elementType**. Pro přehlednost je zobrazím opět pomocí tabulky (tab. 6.3), kde uvedu hodnotu vlastnosti a HTML prvek, kterému to odpovídá. Snažil jsem se aby navržené názvy odpovídali přímo konkrétní HTML tagům.

| Literál | HTML tag |
|---|---|
| text | input typ text |
| password | input typ password |
| date | input typ date |
| hidden | input typ hidden |
| range | input typ range |
| number | input typ number |
| month | input typ month |
| phone | input typ phone |
| email | input typ email |
| checkbox | input typ checkbox |
| textarea | textarea |
| radio | input typ radio |
| select | select |
| datalist | datalist |

Tabulka 6.3: Hodnoty pro vlastnost form:elementType.

Ukázka (Zdroj. kód 6.2), jak takový popis formulářového prvku může vypadat.

```
<http://mre.kiv.zcu.cz/id/fresnel#123> rdf:type form:Format ;
 form:propertyFormatDomain <http://mre.kiv.zcu.cz/ontology/form
    .owl#surname> ;
 form:label "příjmení" ;
 form:elementType "text";
 form:required "true";
 form:placeholder "Your name";
 form:group :mainGroup .
```

Zdrojový kód 6.2: Popis formulářového prvku podle navržené struktury.

V tabulce 6.4 uvedu navržený popis obecných atributů s příkladem možné hodnoty. Kde se dá, který atribut použít, a co způsobuje je vysvětleno v kapitole 5 - Analýza formulářových prvků. Mým cílem je navrhnout popis atributů pro formulářové prvky.

| Název | Typ hodnoty | Příklad |
|---|---|---|
| form:autofocus | literál | true |
| form:checked | literál | true |
| form:disabled | literál | true |
| form:max | literál | 10 |
| form:maxLength | literál | 10 |
| form:min | literál | 0 |
| form:multiple | literál | true |
| form:pattern | literál | \d{5} |
| form:placeholder | literál | vlozte email |
| form:readonly | literál | true |
| form:required | literál | true |
| form:selected | literál | true |
| form:size | literál | 10 |
| form:step | literál | 10 |
| form:value | literál | test |
| form:values | zdroj | ( ds:sexMale ds:sexFemale ) |

Tabulka 6.4: Navržený popis pro atributy formulářového prvku.

Vlastnost *form:values* se očekává pouze u formulářových elementů radio, select a datalist. Ostatní vlastnosti (atributy) jsou navrženy podle názvů v HTML pro menší chaos mezi názvy.

Při implementaci generátoru, budu vycházet z návrhu popisu formuláře v RDF formátu. Výše uvedené atributy jsou navrhnuty tak, aby pokryly většinu případů pro vytvoření formuláře.

## 6.4 Formuláře z různých druhů pohledů

Pokud se podívám na formuláře z různých úhlů pohledu, zajímá mě hlavně použití nového formuláře, editace stávajících hodnot a zabezpečení odesílání dat.

Pro popis nového formuláře lze použít, kterýkoliv ze zmíněných atributů. Při tomto úhlu pohledu není uživatel ničím zatížen.

Pro generování formuláře s vyplněnými hodnotami je připravena vlastnost *form:value*, která vstupnímu poli předvyplní hodnotu. Důležité je potřeba myslet na údaje, které by neměly mít možnost editace. Pro tento účel je připravena vlastnost *form:readonly*, nebo *form:disabled* (použití se dá očekávat např. u rodného čísla).

Pokud uživatel bude chtít své data posílat např. přes šifrovací kanál, tak nejjednodušší cesta je přes nastavení URL adresy pro zpracování formuláře v konfiguračním souboru přes vlastnost *form:action*. Konfigurační soubor umožňuje různé úhly pohledu specifikovat přímo v konfiguračním souboru pomocí nastavení kontextu (vlastnost *form:group*). Můžeme mít více stejných názvů s drobnými úpravami rozlišené podle kontextu.

# 7 Implementace aplikace

V předchozí části jsem navrhl, jak bude vypadat popis formuláře v RDF formátu. Jak bude formulář velký a kolik bude mít prvků záleží pouze na předpisu. Na začátek stručně popíši vybrané prostředky pro implementaci a poté samostatnou implementaci aplikace včetně detailnějšího popisu generátoru. Jak pracovat s výslednou aplikací se uživatel dočte v příloze A (Uživatelská příručka). Veškerý zdrojový kód je napsaný v anglickém jazyce.

Pro aplikaci jsem zvolil programovací jazyk Java SE 1.7. Projekt je vytvořen jako Maven Project z důvodu snadného přístupu k potřebným knihovnám/frameworkům pomocí nastavení závislostí v souboru *pom.xml*, ke který popíšu níže. Při implementaci využívám stránky JSP a pro jejich obsluhu servlety. K nasazení a testování aplikace využívám aplikační server Tomcat 8.0.

Pro práci s RDF daty jsem si vybral doporučený framework Apache Jena. Především z důvodu kvalitní dokumentace a dobrého vysvětlení, jak pracovat s RDF daty. Pro generování HTML formuláře jsem si zvolil framework J2HTML, pro jeho snadné používání, které se mi hodí, jelikož si popis formuláře převádím do mé navržené struktury. Další výhodou je dostupnost přes Maven. Žádný z dalších frameworků k práci nevyužiji.

Než začnu popisovat implementaci jednotlivých částí nástroje popíši strukturu aplikace pomocí tabulky 7.1.

| Adresář | Popis |
|---|---|
| `kořenový adresář` | Obsahuje konfigurační soubor *pom.xml* a samozřejmě ostatní adresáře. |
| `src/main/java` | Obsahuje balíky s Java soubory. |
| `src/main/resources` | Obsahuje různé konfigurační soubory. |
| `src/main/webapp` | Obsahuje hlavně JSP stránky a soubor *web.xml* |

Tabulka 7.1: Tabulka popisující strukturu aplikace.

### pom.xml

Konfigurační soubor projektu, obsahuje obecné informace, nastavení pluginů a externích knihoven, na kterých projekt závisí. V mém případě jsou zde dvě zmíněné knihovny J2HTML a Jena. Mimo tyto jsem přidal závislosti pro práci se servlety. Maven automaticky vyhledá a nainstaluje potřebné knihovny.

### web.xml

Další konfigurační soubor, tento ovšem definuje chování na webu. Nachází se *src/main/webapp/WEB-INF/*. Dochází zde k namapování jednotlivých URL adres k jednotlivým servletům v Javě. Dále k namapování error stránek (např. kód chyby 404). Nastavuji zde i tzv. *welcome-file* (soubor, který se nemapuje, je první, který se zobrazuje na úvodní stránce, většinou *index.jsp*) a základní filter pro nastavení nastavení kódování.

V následující části stručně popíšu jednotlivé balíky a nastíním proces aplikaci pomocí diagramu.

### balík cz.zcu.fav.kiv.genform.filter

Obsahuje pouze základní encodovací filter do UTF-8.

### balík cz.zcu.fav.kiv.genform.servlet

Balík obsahující pouze servlety por obsluhu jednotlivých URL adres. Je zde servlet `SelectFormServlet.java`, který načítá jména formulářů z konfiguračních souborů, které vrací uživateli, aby si zvolil, který formulář vygenerovat. Následuje servlet `GenerateFormSelect.java`, který na základě vybraného formuláře, načte potřebné informace a vygeneruje HTML kód znázorňující formulář. Servlet `ProcessFormServlet.java` zpracovává vyplněný formulář a ukládá data do požadovaného formátu. Je zde ještě jeden servlet `ErrorNotFound.java`, který nás přesměrovává pouze na chybovou stránku při změně adresy na neexistující - kód 404 stránka nenalezena.

### balík cz.zcu.fav.kiv.genform.tags

Tento balík obsahuje třídy, které slouží pro generování jednotlivých vstupních polí, jako je input libovolného typu, select, textarea, datalist a speciální případ inputu, pro typ radio. Dále jsou zde pomocné třídy (přepravky) k formulářových prvkům.

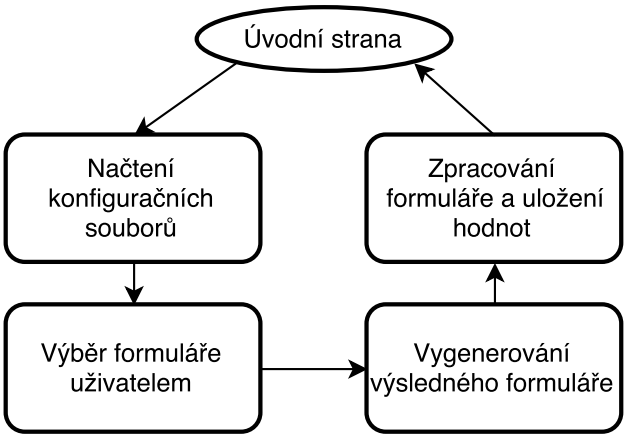
### balík cz.zcu.fav.kiv.genform.utils

Zde jsou třídy s metodami, které nám pomáhají při práci s aplikací. Je zde soubor `Metadata.java`, který obsahuje konstanty. Pak je zde soubor `Utils.java`, který implementuje rozhraní a obsahuje obecné metody. Jako poslední soubor je zde `MainViewForm.java`, který zajišťuje samotné vygenerování HTML formuláře.

### balík cz.zcu.fav.kiv.genform.vocabulary

V tomto balíku se nachází pouze mnou implementovaný slovník využívaný pro navržený jmenný prostor `http://mre.kiv.zcu.cz/ontology/form.owl#`.

Ukázka procesu aplikace (Obr. 7.1).




Obrázek 7.1: Zjednodušený procesní diagram aplikace.

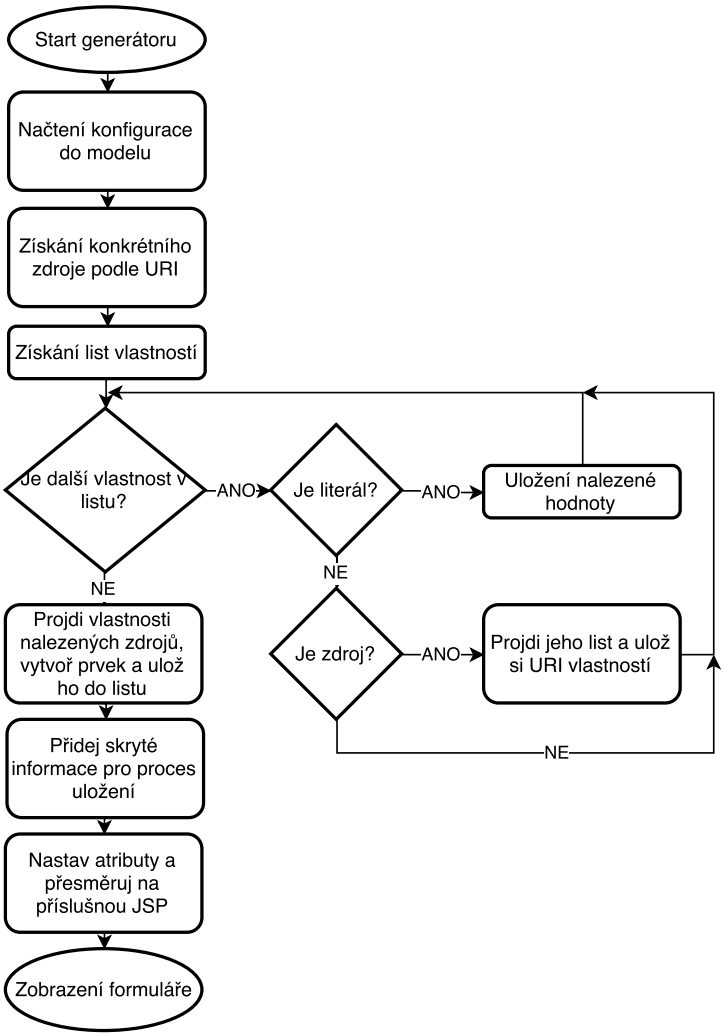
## 7.1 Zpracování vstupní konfigurace

Na začátku procesu je uživateli nabídnuto, zda chce načíst své vlastní konfigurační soubory. Soubory jsou omezeny pouze na příponu *.n3. Pokud uživatel nechce, může využít výchozích souborů umístěných v resources projektu. Názvy výchozích souborů jsou dané, pokud se rozhodnu v budoucnu tyto názvy změnit, je nutné je změnit i v souboru metadat (`Metadata.java`). Pokud uživatel vloží soubory s jinou koncovkou než je stanovená, tyto soubory jsou ignorovány. V případě, že nejsou načteny žádné použitelné soubory, načtou se výchozí (tj. *forms.n3* a *labels.n3*).

Po upřesnění názvů souborů, se kterými budeme pracovat vytvořím pomocí frameworku Jena a třídy `ModelFactory` výchozí model. Do modelu postupně přidávám obsah všech načítaných souborů, které načítám jako model typu N3 pomocí metody `loadModel`. Po vytvoření modelu složeného z konfigurací se nad modelem dotážu pomocí jazyka SPARQL a získám všechny vyhovující zdroje. To jsou záznamy, které jsou typu Form s požadovaným jmenným prostorem pro formulář (tedy *form:Form*). Po získání výsledků je iterativně procházím a ukládám si URI hodnoty a zkouším hledat v každém jméno formuláře (vlastnost *form:formName*). Pomocí aktuální URI hodnoty si z modelu vytáhnu hledaný zdroj nad kterým se pokusím vyhledat vlastnost reprezentující název formuláře. Nalezené hodnoty si ukládám do přepravky `TagAttribute.java`, která má dva atributy (jméno a hodnotu). V mém případě je URI hodnota a jméno nese název formuláře (pokud vlastnost nenajdeme, je URI zároveň názvem formuláře). Po zpracování všech vyhledaných výsledků jsou zobrazeny uživateli, aby si zvolil, který z možných formulářů chce vygenerovat. Do formuláře si přidávám ještě skrytý atribut nesoucí název konfiguračních souborů pro další práci.

## 7.2 Generátor formuláře

Než začnu popisovat implementaci generátoru, tak pro lepší představu nastíním jeho proces pomocí vývojového diagramu (Obr. 7.2). Vývojový diagram kvůli složitosti bude zjednodušený.




Obrázek 7.2: Zjednodušený vývojový diagram generátoru formuláře.

Při výběru ze seznamu dostupných formulářů uživatel specifikuje i jazykové možnosti, které se při generování přidávají ke všem vstupním polím typu text. Po vybrání generovaného formuláře přijmu vybranou URI hodnotu a jména konfiguračních souborů. Z počátku opět dojde k načtení modelu, jako v předchozím případě. Poté mě ale zajímá už konkrétní zdroj, který vyhledám pomocí URI hodnoty nad modelem. Tím získám už konkrétní informace o formuláři. Než budeme procházet jednotlivé vlastnosti formuláře, omezím se pouze na formulářový jmenný prostor (tj. pouze vlastnosti s prefixem *form:*). Při procházení vlastností získaných pomocí metody `listProperties` si vytáhnu objekt jako RDF uzel. Poté se rozhoduji, zda jsem načetl literál nebo další objekt informací.

V případě literálu (tj. přímá hodnota) hledám na této úrovni pouze základní informace o formuláři. Mezi tyto informace patří obslužná URL adresa formuláře (*form:action*), metoda formuláře (*form:method*), výstupní název ukládaného souboru s hodnotami (*form:outputName*) a typy (koncovky) ukládaného souboru (*form:saveAs*). Ostatní nalezené vlastnosti jsou ignorovány.

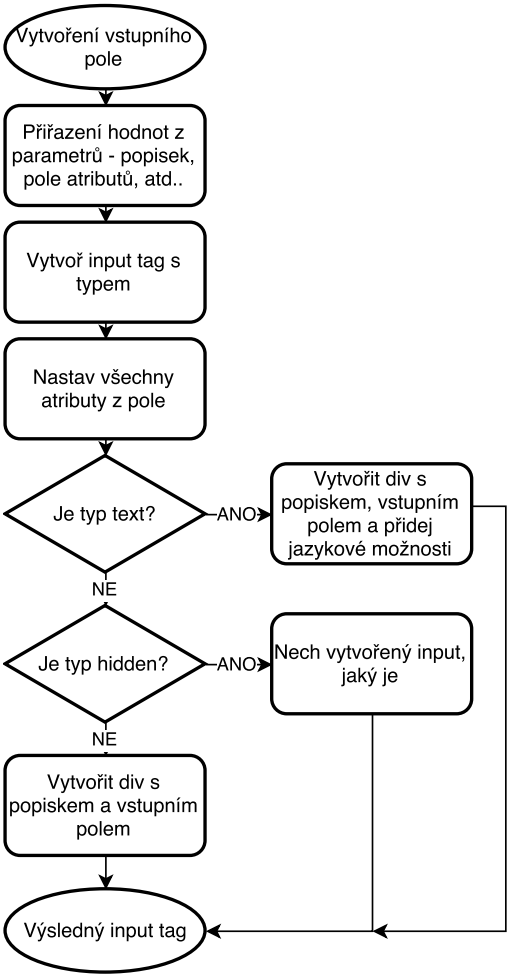
V případě nalezení dalšího zdroje (objektu) mě zajímají pouze dvě možnosti. Zda-li se jedná o vlastnosti, které mají být vidět (*form:showProperties*) nebo o vlastnosti skryté (*form:hideProperties*). V obou případech se chovám stejně a to tak, že získám opět jejich list vlastností a rekurzivně tento list projdu, jelikož mi vždycky vrací aktuální hodnotu a zbytek listu. Tyto hodnoty si uložím (jako URI) pro pozdější získání konkrétního popisu elementu.

Nyní mám uložené obecné informace pro formulář včetně URI adres jednotlivých formulářových elementů, na které se dotazuji opět jazykem SPARQL. Připravím si dynamický list HTML tagů. To je třída `Tag` využívaná z frameworku J2THML. Začnu procházet list URI hodnot, které jsem získal z vlastností *showProperties* a *hideProperties*. Vždy danou URI dosadím do připraveného SELECTu a dotážu se nad načteným modelem. Tím začínám vytvářet nový formulářový prvek. Pro každý nový prvek si vytvářím list typu `TagAttribute`, což je moje pomocná třída, pro uložení jména atributu a jeho hodnoty. Obdobně si vytvářím list typu `TagOptionAttribute` pro uložení hodnot do případného typu select, datalist, atd. Po dotázání získám opět list vlastností daného prvku, kde se opět omezím na jmenný prostor formuláře. List vlastností začnu opět procházet jako v předchozím případě, vytáhnu si objekt jako RDF uzel. Jako první si zjistím typ formulářového elementu (*form:elementType*). Potom se rozhoduji, zda-li uzel je opět zdroj nebo ne. Pokud není, tak získám si textovou podobu atributu (např. pro vlastnost *form:required "true"* je textová podoba to *required*). Na základě toho se rozhoduji, zda je to popisek, který si uložím do připravené proměnné nebo ho vložím do listu atributů k aktuálně vytvářeného prvku. Pokud-li je to ale opět zdroj, postupuji stejně jako dříve, jenom jsem zanořený opět o úroveň níže. Zde očekávám pouze možnosti pro radio skupiny, nebo select list, či datalist. Proto si získám rovnou hodnotu, jméno a flag o tom, jestli je označená (*form:checked* a *form:selected*). Po získání listu atributů k danému elementu zavoláme příslušný konstruktor pro daný formulářový prvek. Při vytváření jednotlivých elementů si u více specifických prvků ukládám i jejich URI hodnotu, pod kterou jsou vyhledatelný ve formuláři. Tyto hodnoty si poté posílám s formulářem, abych byl na základě nic schopný rozhodnout, které elementy vrací objekt a které přímou hodnotu. Speciální případ je vstupní pole typu *checkbox* a *date*, ty sice objekt nevrací, ale potřebuji je při ukládání hodnot rozeznat (viz. podkapitola 7.3. Zpracování a uložení

dat).

### Generování formulářových elementů

Tak jako u předchozí části než začnu popisovat implementaci generování jednotlivých tagů, tak pro lepší představu nastíním proces pomocí vývojového diagramu (Obr. 7.3). Následující vývojový diagram znázorňuje vytvoření vstupního pole. Ostatní formulářové elementy jsem na implementoval velmi podobně s minimálními změny z důvodu specifičnosti prvků.




Obrázek 7.3: Vývojový diagram vstupního pole.

Před vytvořením elementu formuláře získám podle předchozího popisu list atributů, popisek a v případě selectu či datalistu i list možností. Na základě

těchto sesbíraných informací se rozhoduji, který formulářový prvek vytvořit, podle přečteného typu z konfigurace. Pokud typ elementu není zadán, prvek se nevytvoří. Možné prvky jsou buď *textarea*, *radio*, *select*, *datalist* a nebo výchozí je vstupní pole *input*, kde načtený typ signalizuje typ vstupního pole. Každý vytvořený prvek má přidaný atribut *name*, kde je hodnota URI daného prvku. Popíšu zde vytvoření vstupního pole, vytvoření ostatních formulářových prvků pracuje velmi podobně s minimálními úpravami, dle funkcionality prvku. Pro vytvoření vstupního pole nám slouží konstruktor `CreateInputElement`, který má tyto vstupní parametry: popisek, typ vstupního pole, jazykové možnosti pro typ text a list atributů, které se mají nastavit. Na začátku se vytvoří formulářový tag, v tomto případě vstup s daným typem (např. `input().withType(...)`). Teď když mám vytvořený tag nastavím mu všechny atributy z listu přes metodu `setAttribute`, kde jsou dva parametry, první z nich je název a druhý hodnota. Tímto způsobem lze prvku nastavit libovolný atribut. Poté pouze nastavuji vzhled elementu a přidávám k němu popisek. V případě, že se jedná o typ text, přidávám ještě možnost (jednoduchý select) volby jazyka. Obdobným způsobem se vytváří i ostatní elementy. Po vytvoření všech tagů je vložím společně dalšími získanými informacemi (action, method) do formuláře. Získám tak HTML kód, který si pošlu jako atribut na připravenou JSP stránku, kde HTML zobrazím. Toto vygenerované HTML je náš výsledný formulář vytvořený dle popisu v konfiguračním souboru.

### Hodnoty mimo formulář

Do formuláře mimo požadovaných hodnot vkládám další skryté pomocné hodnoty pro proces uložení. Jedná se o informace nesoucí jména prvků, které vrací objekt, to jsou select, radio, datalist. Dále to jsou informace o vstupních polích, které jsou potřeba modifikovat. Do této kategorie patří checkbox a list jmen datových položek. Nakonec si posílám i název, pod kterým se mají data uložit včetně formátu. Tyto hodnoty jsou důležité pro správné uložení do výsledného modelu, protože po odeslání formuláře získám z každého prvku jeho jméno a hodnotu.

## 7.3 Zpracování a uložení dat

Po odeslání formuláře zpracovávám data a ukládám je do výsledného modelu. Nejprve si rozdělím příchozí parametry na dvě skupiny. Parametry formuláře jako takového a moje skryté parametry nesoucí informace o prvkách a způsobu uložení. Poté vytvořím výchozí model opět pomocí `ModelFactory`. Modelu přidám používaný prefix formulářového prostoru. Poté vytvořím dva zdroje nad modelem. První nese typ ukázkového formuláře a druhý jednoznačně vygenerovaný identifikátor přidaný za URI formuláře pro jeho jednoznačné určení. Tento identifikátor je náhodný čtyřiceti znakový text složený z písmen a číslic pomocí třídy `RandomStringUtils`. Pak procházím moje skryté parametry a pokud-li nějaký najdu přidávám jeho hodnotu do modelu jako zdroj s URI hodnotou odkazující se dál (nejde o přímou hodnotu). Toto se týká pouze radio skipiny a selectu či datalistu. Nachází-li se ve formuláři datové elementy, najdu je a převádím do požadovaného tvaru tj. *yyyy-MM-dd'T'HH:mm:ssZ*. Pokud-li nastane problém s časem (výjimka/chyba při převodu času), nahrazuji ho aktuálním. Posledním specifickým případem je zaškrtávací pole. Formulář nám vrací hodnotu *on* v případě zaškrtnutí a to musíme nahradit za hodnotu boolean (pravda/nepravda). Následuje zpracování ostatních hodnot z formuláře, které jsou již ve formě jméno hodnota a nemusí se s nimi nijak operovat. Jediný případ, kdy do toho zasahuji je, pokud se jedná o jazykovou hodnotu, k tomu poté vyhledám příslušný vstup a jeho hodnotu uložím s vybraným jazykem. Nakonec model uložím pod jménem z konfigurace v příslušném formátu. Po uložení dojde k přesměrování na úvodní stranu.

# 8 Ověření a validace nástroje

V této části se budu soustředit na testování webové aplikace a to především na testy funkcionality generování formuláře, uložení hodnot a vložení možných hodnot do vstupných polí. V neposlední řadě provedu testování kompatibility mezi webovými prohlížeči. Pro testování nebudu využívat automatických testů.

Aplikace je testována na stroji Lenovo ThinkPad T520, Intel(R) Core(TM) i7-2670QM CPU @ 2.20GHZ, 16,00 GB RAM, Samsung SSD 850 EVO 250G, na systému Windows 7 Professional 64bitové verzi. Aplikace je spuštěna pomocí serveru Tomcat v 8.0.

Nejprve by bylo vhodné uvést, že na aplikaci nebyly vzneseny žádné grafické požadavky a celá aplikace je naprogramována podle popisu nástroje (viz Kapitola 5. Popis nástroje pro generování formuláře). Na základě toho se budu zabírat více směrem funkcionality. Za otestování však stojí vyzkoušet různé prohlížeče, protože v každém prohlížeči se některé formulářové prvky chovají jinak. Ovšem ale jako první otestuji obecnou funkcionalitu nástroje. Tím mám na mysli načítání různých vstupních konfigurací, výchozí konfiguraci, změny v URL adrese. Poté si nechám vygenerovat jednoduchý formulář s pár textovými vstupními poli, pouze pro otestování funkcionality nástroje. Pro samostatné otestování jednotlivých formulářových prvků jim věnuji vlastní podkapitolu (viz. 8.1 Testování generátoru).

Jak bylo řečeno, v každém webovém prohlížeči se formulářové prvky mohou chovat jinak. Do mého testování jsem zahrnul dle mého čtyři nejpoužívanější webové prohlížeče Google Chrome verze 50.0, Firefox v 45.0, Opera verze 36.0 a Internet Explorer verze 9.0. Aplikace není určena pro mobilní zařízení, a proto nebyla testována pro mobilní webové prohlížeče.

K testování použiji dva před připravené konfigurační soubory. Jedna konfigurace nám poslouží jako test základní funkcionality (načtení, vygenerování, uložení) a druhá nám popíše formulář se všemi požadovanými elementy včetně různých kombinací atributů (povinné pole, placeholder, pattern, atd.). Oba zmíněné konfigurační soubory jsou uloženy ve zdrojových souborech na CD (*src/main/resource/test*) včetně uložených souborů na základě jejich konfigurace.

Shrnutí výsledků obecného testování. Při načítání jiných souborů než konfigurací s koncovkou *.n3 se načtou výchozí konfigurační soubory. Pokud nevybere žádné konfigurační soubory a necháme si rovnou načíst výchozí, tak se

nástroj zachová dle očekávání a nabídne uživateli načtené jména formulářů (samozřejmě list těchto jmen může být prázdný, záleží na konfiguračním souboru). Po vybrání se zaktivuje tlačítko pro odeslání výběru a vygeneruje se nám formulář. Ten se po odeslání uloží a uživatel je přesměrován. Při pokusu změny URL adresy na neznámou, se nám zobrazí odchycená hláška 404. Toto testování proběhlo ve všech zmíněných prohlížečích a nebyla nalezena žádná chyba.

## 8.1 Testování generátoru

Pro testování generátoru v různých prohlížečích použiji jednu a tutéž rozšířenou testovací konfiguraci zmíněnou výše. Obecná funkcionalita víme, že funguje. Proto se zaměříme pouze na vstupní pole a jejich atributy. Generátor na základě konfigurace vygeneruje HTML formulář více či méně složitý a nastaví danému poli atributy, které jsou k němu popsány v konfiguraci bez toho, aniž by patřily k danému vstupní poli nebo ne. Tím se dá říct, že ke kterémukoliv vstupnímu poli, lze přiřadit jakýkoliv atribut, to nám nezpůsobí žádné potíže, protože ho všechny prohlížeče ignorují, pouze nám to znepřehledňuje výsledný HTML kód (který běžný uživatel stejně nevidí). Jak dopadly testy jednotlivých vstupních polí a atributů jsem pro přehlednost zobrazil v jednotlivých tabulkách.

V tabulce 8.1 je srovnání s ostatními prohlížeči, pokud se u daného elementu vyskytuje zelená fajfka, element se zobrazuje a reaguje dle očekávání, bez ohledu na nastavení jeho atributů. Pokud je v dané kolonce křížek, byl u tohoto elementu zjištěn nedostatek. Nalezené nedostatky nejsou chybou generátoru, ale jsou způsobeny různými webovými prohlížeči. Nalezené nedostatky by se daly ošetřit pomocí JavaScriptových funkcí, či komponent založených na JavaScriptu.

Tabulka výsledků testovaných elementů ve více prohlížečích.

| Element | Chrome | Firefox | Opera | IE |
|---|---|---|---|---|
| `input="text"` | ✓ | ✓ | ✓ | ✓ |
| `input="password"` | ✓ | ✓ | ✓ | ✓ |
| `input="date"` | ✓ | ✗ | ✓ | ✗ |
| `input="email"` | ✓ | ✓ | ✓ | ✗ |
| `input="month"` | ✓ | ✗ | ✓ | ✗ |
| `input="number"` | ✓ | ✓ | ✓ | ✗ |
| `input="range"` | ✓ | ✓ | ✓ | ✗ |
| `input="radio"` | ✓ | ✓ | ✓ | ✓ |
| `input="checkbox"` | ✓ | ✓ | ✓ | ✓ |
| `textarea` | ✓ | ✓ | ✓ | ✓ |
| `select` | ✓ | ✓ | ✓ | ✓ |
| `datalist` | ✗ | ✓ | ✗ | ✗ |

Tabulka 8.1: Tabulka výsledků testovaných elementů ve více prohlížečích.

U formulářového prvku `datalist` byl nalezen nedostatek ve hledání dostupných možností. Místo vyhledávání nad atributem `name` element vyhledává nad atributem `id`. U prohlížeče Firefox je problém se vstupním polem pracující s datem. Pole se chová jako textové pole a nenabízí žádný pomocný kalendář (datepicker). Prohlížeč Internet Explorer 9 nepodporuje všechny formulářové prvky z HTML5, a proto se většina testovaných vstupních polí chovala jako textové pole.

Tabulka 8.2 zobrazuje výsledky testovaných atributů napříč elementy. Například atribut `required` lze použít u většiny vstupních polí, ale takový atribut `rows` lze využít pouze u elementu `textarea`. Proto v tabulce vždy uvedu, u kterého elementu byl daný atribut otestován.

| Atribut | Chrome | Firefox | Opera | IE | Element |
|---|---|---|---|---|---|
| autofocus | ✓ | ✓ | ✓ | ✗ | input=”text” |
| disabled | ✓ | ✓ | ✓ | ✓ | input=”text” |
| maxLength | ✓ | ✓ | ✓ | ✓ | input=”password” |
| placeholder | ✓ | ✓ | ✓ | ✗ | input=”email” |
| readonly | ✓ | ✓ | ✓ | ✓ | input=”text” |
| required | ✓ | ✓ | ✓ | ✗ | input=”text” |
| value | ✓ | ✓ | ✓ | ✓ | input=”text” |
| checked | ✓ | ✓ | ✓ | ✓ | input=”checkbox” |
| selected | ✓ | ✓ | ✓ | ✓ | select - option |
| pattern | ✓ | ✓ | ✓ | ✗ | input=”phone” |
| step | ✓ | ✓ | ✓ | ✗ | input=”range” |
| min | ✓ | ✓ | ✓ | ✗ | input=”number” |
| max | ✓ | ✓ | ✓ | ✗ | input=”number” |
| cols | ✓ | ✓ | ✓ | ✓ | textarea |
| rows | ✓ | ✓ | ✓ | ✓ | textarea |

Tabulka 8.2: Tabulka výsledků testovaných atributů napříč elementy.

Výsledky atributů u Internet Exploreru 9 se daly předpokládat. Pokud Internet Explorer 9 nepodporuje část formulářových prvků, nepodporuje ani novější HTML5 atributy. Prohlížeč je ignoruje.

## 8.2 Zhodnocení výsledků

Při testování obecné funkcionality nástroje nebyly nalezeny chyby. Problémy s HTML elementy se začaly objevovat až při změně webových prohlížečů. Tyto problémy jsou, ale obecným problémem a nejsou chybou generátoru. Nalezené problém jsou pouze menší nepříjemnosti. Aplikace je připravena na předání koncovému uživateli.

Doporučenými prohlížeči jsou Google Chrome, Opera a Firefox. U prohlížeče Internet Explorer bylo zjištěno nejvíce nedostatků. U Google Chromu a Opery je problém s elementem datalist a u Firefoxu je to problém se vstupním polem, které pracuje s datem.

# 9 Diskuze

## 9.1 Zhodnocení dosažených výsledků

Realizovaný generátor byl otestován na navrhovaném způsobu popisu v RDF. Generátor zpracuje vstupní konfiguraci a vygeneruje požadovaný formulář. Generátor vrací formulář s využitím HTML5 atributů a vstupních polí, tím dochází k menším nepříjemnostem v různých webových prohlížečích (viz tab.:8.1 a tab.:8.2). Tyto problémy by se daly odstranit použitím JavaScriptových funkcí. Pokud v konfiguraci neuvedeme žádné formulářové prvky, ale tvrzení nám říká, že jde o formulář, tak i přes to dojde k jeho vygenerování. Z toho plyne nutnost mít v konfiguraci seznam prvků.

Při zpracování konfiguračních souborů nebo ukládání hodnot není uživatel informován o běhu generátoru. Pokud nastane výjimka vypíše se pouze informační zpráva do serverové konzole. Stálo by za to při odchycení výjimky o tom uživatele informovat přímo ve webovém prohlížeči. Při ukládání hodnot do požadovaného souboru může nastat problém s přístupem do složky. Pokud uživatel nemá právo zápisu, ukládání skončí chybovým výpisem.

Validace dat by se také mohla zlepšit. Jediná validace, která na formuláři probíhá zajišťují HTML5 atributy (např. required, pattern, atd.). Po odeslání formuláře dochází pouze k uložení hodnot. Pokud bychom chtěli zavést validaci dat podle popisu v konfiguračním souboru, museli bychom opět načíst informace o jednotlivých vstupních polích.

V zadaní je i požadavek na generování formuláře v XML. Původně formou šablon mělo být možné si zvolit jinou šablonu a vygenerovat místo HTML i cokoliv jiného. Z nedostatku času a rozsahu jsme se s vedoucím domluvili na omezení zadání na oblast HTML5 formulářů, které je prioritou.

## 9.2 Porovnání řešení jiných autorů

Uvedené frameworky v kapitole 3.1 se zabývají generováním kódu na základě popisu. Žádný z uvedených se přímo nezabývá RDF daty, ale pouze samotným generováním. Telosys dokonce generovat kód do více druhů kódu,

zatím co vytvořený generátor pouze do HTML. Jeho nevýhodou je ovšem závislost na programovacím prostředí Eclipse. Pokud bychom dokázali převést RDF do XForms, tak samotné vygenerování pomocí Orberonu by už nebyl problém. Oproti vytvořenému generátoru je hlavní nevýhodou především naprostá závislost na JavaScriptu. Vlastní vygenerovaný formulář nevyužívá žádnou JavaScriptou funkci, ale lze použít. Mimo jiné Orberon ke svému běhu potřebuje na straně klienta cca 400KiB JavaScriptový soubor. PHP Database Form dokáže generovat formulář pouze podle dat z relačních databází. Omezení vlastního řešení spočívá na základě konfigurace ve formátu n3. Ale jeho nevýhodou pro můj účel je skriptovací jazyk PHP, ve kterém implementován. Tento framework jsem zde uvedl především proto, kdyby se někdy generátor předělával do jazyku PHP.

# 10 Závěr

Diplomová práce se zabývala navržením popisu formuláře pomocí RDF dat a implementováním nástroje, který bude schopný na základě navrženého popisu vygenerovat HTML formulář. V základním popisu nástroje je zmíněna hlavní funkcionalita. Po domluvě s vedoucím práce byly některé části blíže upřesněny. Vývojem nástroje chci dosáhnout větší automatizace na projektu MRE vedeným na KIV. Uživatel díky jednoduché konfiguraci v RDF formátu dokáže vygenerovat HTML formulář.

Povedlo se mi navrhnout jednoduchý a flexibilní popis pro generování formuláře z RDF dat. Flexibilita spočívá např. v odlišném chování v závislosti na kontextu. Generátor HTML je stabilní a jeho největší výhoda je, že lze přidat kterýkoliv atribut k elementu bez jakýchkoliv změn, tato výhoda je myšlena do budoucna vývoje HTML tagů a jejich atributů.

Na začátku práce jsem se zaměřil na teoretický úvod do dané problematiky a na technologie potřebné pro tvorbu nástroje vycházející z analýzy odborných publikovaných článků a literatury.

Na základě požadavků jsem navrhl popis formuláře pomocí RDF dat. Popis zahrnuje základních informace o formuláři, jednotlivé formulářové elementy včetně jednotlivých atributů. Ukázka konfigurace je v Příloze B a z ní vygenerovaný formulář je v Příloze C a v příloze D je ukázka uložených hodnot z formuláře.

Nad navrženým popisem byl implementován nástroj pro přečtení takové konfigurace a z ní možné vygenerování HTML formuláře. Jako první jsem začal implementaci zpracování vstupních souborů (konfigurací) a načtení dostupných formulářů. Pokračoval jsem implementací zpracování popisu vybraného formuláře do struktury pro vygenerování HTML kódu. Jako poslední jsem implementoval uložení hodnot do příslušného formátu.

Po vytvoření nástroje jsem provedl její testování. Díky tomu jsem odhalil drobné chyby, které byly následně opraveny. Hlavním zdrojem chyb bylo opomenutí programátora.

Aplikace byla vytvořena s použitím programovacího jazyka Java s pomocí stránek JSP pro zobrazení. Celá aplikace byla vyvíjena na aplikačním serveru Tomcat 8.0, kde by měla i do budoucna fungovat pod projektem MRE

vyvíjeným na Západočeské univerzitě v Plzni.

Nástroj by se mohl do budoucna vylepšit o další nové HTML prvky. Popis v RDF by se dal zjednodušit a zkusit navrhnout šablony, které by už popisovaly daný případ (např. šablona pro přihlašovací formulář).

# Literatura


[1] Petr Matulík, Tomáš Pitner. Sémantický web a jeho technologie [online]. [cit. 2016-05-04]. Dostupné z: http://webserver.ics.muni.cz/zpravodaj/articles/296.html

[2] Ing. Jaroslav Dytrych. Sémantický web. [online]. [cit. 2016-05-10]. Dostupné z: http://www.fit.vutbr.cz/study/courses/VPD/public/0910VPD-Dytrych.pdf

[3] Antoniou Grigoris, Van Harmelen Frank. A semantic web primer. Cambridge (Mass) : MIT Press, 2004. 238 s. ISBN-10: 0262012103.

[4] Jiří Procházka. Úvod do Sémantického Webu [online]. [cit. 2014-05-01]. Dostupné z: https://www.zdrojak.cz/clanky/uvod-do-semantickeho-webu/

[5] Lacy Lee E. Owl: representing information using the web ontology language. Victoria : Trafford Publishing, 2005. 302 s. ISBN 978-1412034487.

[6] Vojtěch Svátek. Ontologie a WWW [online]. [cit. 2014-04-09]. Dostupné z: http://nb.vse.cz/~svatek/onto-www.pdf

[7] Smrž Pavel, Pitner Tomáš. Sémantický web a jeho technologie (3) [online]. [cit. 2014-04-09]. Dostupné z: http://webserver.ics.muni.cz/zpravodaj/articles/307.html

[8] Uschold M., Gruninger M. Ontologies: principles, methods and applications. The Knowledge Engineering Review, Vol.11:2, 1996, 93-136.

[9] Roberta Cuel,Robert Young. Formal Ontologies Meet Industry: 7th . 2015. 137 s. ISBN 978-3-319-21544-0.

[10] Kořínková Presová, Silvie. Ontologie - metodika tvorby [online]. [cit. 2014-04-10]. Dostupné z: http://docplayer.cz/13481891-Ontologie-metodika-tvorby.html

[11] Pascal Hitzler,Markus Krotzsch,Sebastian Rudolph. Foundations of Semantic Web Technologies. 2010. 426 s. ISBN 978-1-4200-9050-5.

[12] Masahiro Hori, Jérôme Euzenat, Peter F. Patel-Schneider. [online]. [cit. 2016-05-06]. Dostupné z: https://www.w3.org/TR/owl-xmlsyntax/

[13] Ora Lassila, Ralph R. Swick. Resource Description Framework [online]. [cit. 2016-05-03]. Dostupné z: https://www.w3.org/TR/PR-rdf-syntax/

[14] Daconta, M. C., Obrst, L. J., Smith, K. T. What Is the Resource Description Framework? [online]. [cit. 2016-05-03]. Dostupné z: http://www.devx.com/semantic/Article/34816

[15] Alena Lukasová. RDF datový model [online]. [cit. 2016-05-05]. Dostupné z: prf.osu.cz/kip/dokumenty/rdf-datovymodel.pps

[16] Doc. Ing. Vojtěch Svátek, Dr. Základy jazyka RDF [online]. [cit. 2016-05-04]. Dostupné z: http://nb.vse.cz/~svatek/rzzw/RDF_old.pdf

[17] Eric Prud'hommeaux. SPARQL Query Language for RDF. [online]. [cit. 2016-05-10]. Dostupné z: https://www.w3.org/TR/rdf-sparql-query/

[18] Devedžič Vladan. Semantic web and education. New York : Springer, 2006. 353 s. ISBN 978-0387-35417-0.

[19] Telosys Tools code generator. [online]. [cit. 2016-05-10]. Dostupné z https://sites.google.com/site/telosystools/

[20] Orbeon, Inc. Orbeon Forms 2016.1. [online]. [cit. 2016-05-11]. Dostupné z http://www.orbeon.com/

[21] Kayson Group Limited. PHP Database Form. [online]. [cit. 2016-05-11]. Dostupné z http://phpdatabaseform.com/

[22] The Apache Software Foundation. Apache Jena. [online]. [cit. 2016-05-10]. Dostupné z: https://jena.apache.org/

[23] Dutch software company Aduna. Sesame. [online]. [cit. 2014-05-02]. Dostupné z: http://rdf4j.org/

[24] Inria. JFresnel. [online]. [cit. 2016-05-10]. Dostupné z http://jfresnel.gforge.inria.fr/

[25] The Apache Software Foundation. FreeMarker. [online]. [cit. 2016-05-10]. Dostupné z: http://freemarker.org/

# Seznam použitých zkratek

API *(Application Programming Interface)*
: Rozhraní pro programování aplikací.

FTL *(FreeMarker Templetes Languages)*
: Jazyk pro popis šablon.

HTML *(HyperText Markup Language)*
: Hypertextový značkovací jazyk.

JSP *(JavaServlet Page)*
: Technologie pro vývoj hlavně dynamických HTML stránek.

LGPL *(GNU Lesser General Public Licens)*
: Všeobecná veřejná licence GNU.

MRE *(Medical Research & Education)*
: Projekt Lékařský výzkum a vzdělání.

OWL *(Web Ontology Language)*
: Jazyk pro popis metadat.

RDF *(Resource Description Framework)*
: Jazyk pro popis metadat.

SHOE *(Simple HTML Ontology Extension)*
: Starší jazyk pro popis metadat.

SQL *(Structured Query Language)*
: Dotazovací jazyk pro relační databáze.

URI *(Uniform Resource Identifiers)*
: Definuje přesnou specifikaci zdroje.

URL *(Uniform Resource Locator)*
: Definuje adresu určující umístění dokumentu na internetu.

W3C *(World Wide Web Consortium)*
: Světová organice.

WWW *(World Wide Web)*
: Celosvětová síť souborů.

XML *(Extensible Markup Language)*
: Obecný značkovací jazyk.

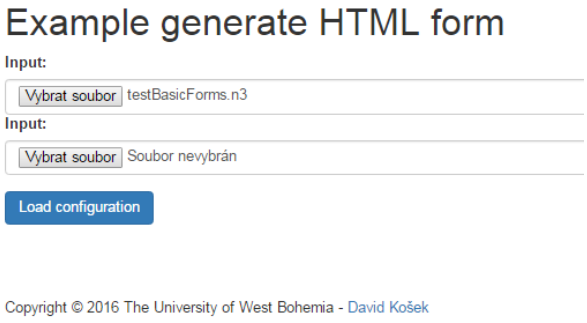
ZČU *(University of West Bohemia)*
: Zkratka pro Západočeskou univerzitu v Plzni.

# Příloha A – Uživatelská příručka

Ovládání aplikace je intuitivní, ale i přesto zde popíšu kroky, jak s nástrojem pracovat.

### Krok 1

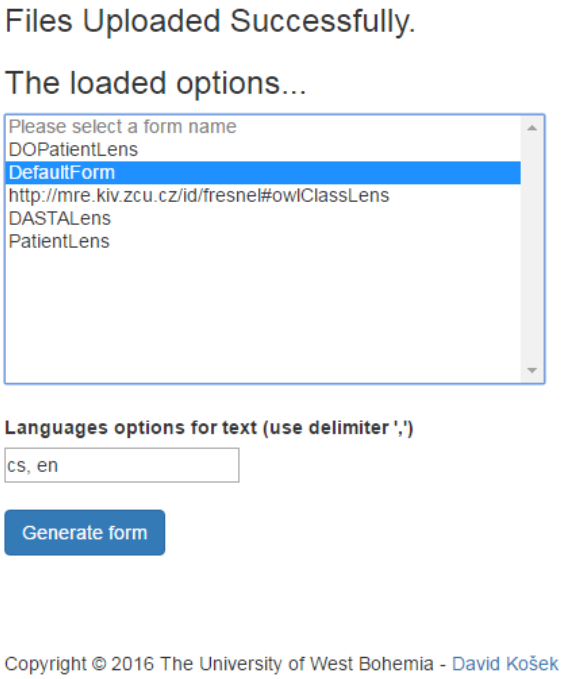
První krok je nahrání konfigurace ve formátu n3. Pokud uživatel konfiguraci nemá, může využít výchozí konfiguraci rovnou kliknutím na tlačítko.




Obrázek 10.1: Počáteční formulář pro výběr konfiguračních souborů.

### Krok 2

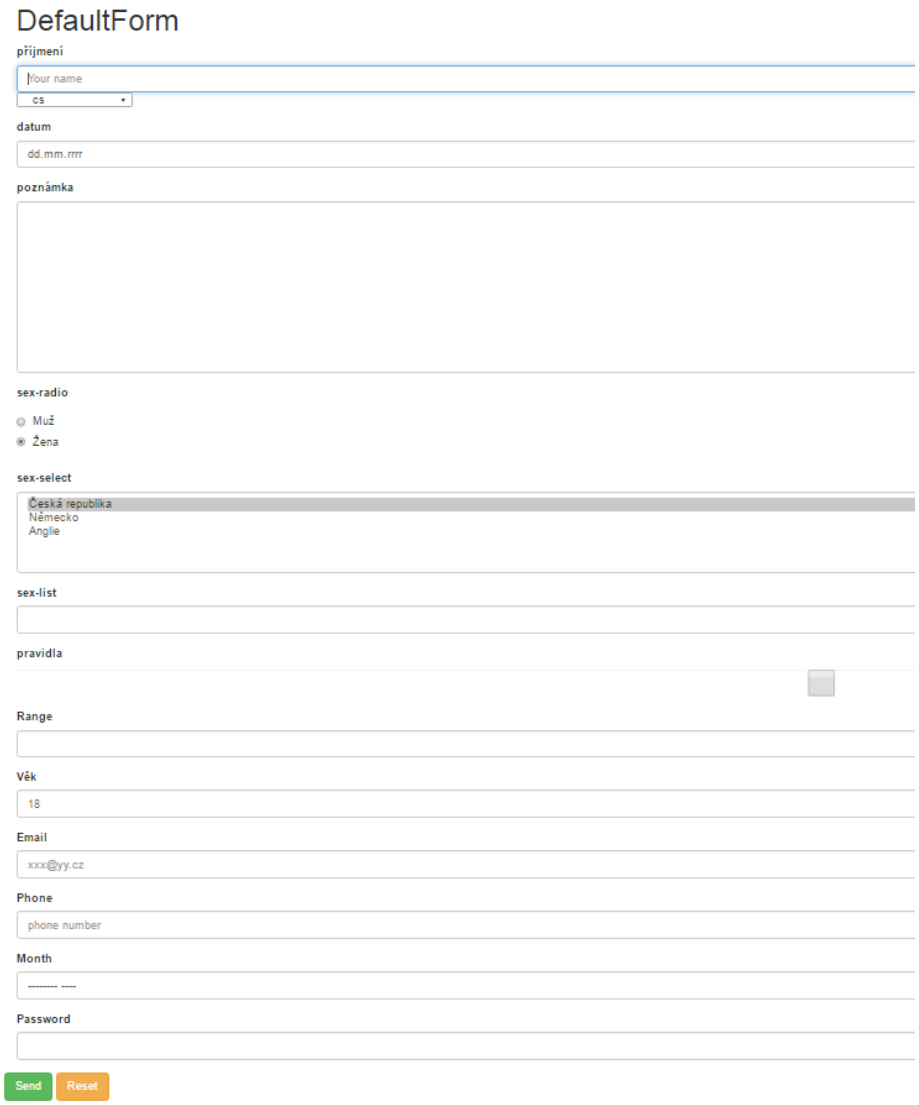
Po načtení konfigurace jsou uživateli nabídnuty možné formuláře k vygenerování. Tlačítko se aktivuje až si uživatel vybere některý z nabízených formulářů. Dále zde může uživatel zadat jazykové možnosti pro textové vstupní pole. Ty se vkládají za sebe a oddělují se čárkou.




Obrázek 10.2: Výběr formuláře.

### Krok 3

Uživateli je zobrazen požadovaný formulář na základě konfigurace. Pokud jsou na elementech formuláře nastavené některé validní funkce (např. pattern, required, atd.) jsou na potvrzení formuláře zkontrolovány a po případě je uživatel upozorněn, nebo jsou data odeslány k uložení.




Obrázek 10.3: Vygenerovaný formulář.

### Krok 4

Data jsou uloženy pod názvem a v požadovaném formátu. Uživatel je přesměrován zpět na hlavní stránku (Krok 1).

Pokud jsou některá pole povinná a nejsou vyplněné. Formulář se neodešle a

zahlásí chybu.


Obrázek 10.4: Chybová hláška.

# Příloha B – Ukázka konfiguračního souboru

```
@prefix : <http://mre.kiv.zcu.cz/id/fresnel#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix form: <http://mre.kiv.zcu.cz/ontology/form.owl#> .
@prefix ds: <http://mre.kiv.zcu.cz/ontology/dasta.owl#> .

:TestBasicForm rdf:type form:Lens ;
       form:nameForm "Test basic form";
       form:action "/genform/form/processsss";
       form:method "post";
       form:outputName "testBasicForm";
       form:group :mainGroup ;
       form:saveAs "rdf";
   form:showProperties (
       ds:name
     ds:surname
   ) .

<http://mre.kiv.zcu.cz/id/fresnel#1> rdf:type form:Format ;
form:propertyFormatDomain <http://mre.kiv.zcu.cz/ontology/dasta
   .owl#name> ;
form:label "jméno" ;
form:elementType "text";
form:placeholder "Your name";
form:group :mainGroup .

<http://mre.kiv.zcu.cz/id/fresnel#12> rdf:type form:Format ;
form:propertyFormatDomain <http://mre.kiv.zcu.cz/ontology/dasta
   .owl#surname> ;
form:label "příjmení" ;
form:elementType "text";
form:placeholder "Your surname";
form:group :mainGroup .
```

# Příloha C – Ukázka vygenerovaného formuláře včetně uložení hodnot


Obrázek 10.5: Ukázka vygenerovaného formuláře na základě konfiguračního souboru z Přílohy B.

Ukázka uložených hodnot z formuláře ve formátu RDF.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:form="http://mre.kiv.zcu.cz/ontology/form.owl#">
  <form:exampleForm rdf:about="http://mre.kiv.zcu.cz/id/form#
      B8OiLpEuMZxhDaJfT5vJYdmMmA6BwcypbJyTbo99">
    <form:surname xml:lang="cs">Košek</form:surname>
    <form:name xml:lang="cs">David</form:name>
  </form:exampleForm>
</rdf:RDF>
```

# Příloha D – Obsah CD-ROM

CD-ROM je umístěný v na přední straně desek diplomové práce. Obsah disku je uveden níže:

| | |
|---|---|
| **Dokumentace** | Adresář s tímto dokumentem a jeho zdrojovým textem včetně posteru. |
| **Zdrojové kódy** | Adresář se všemi zdrojovými kódy. |