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

Bakalářská práce

Návrh vstupního formuláře do lékařského informačního systému

Plzeň, 2013 Mario Kamburov

Prohlášení

Prohlašuji, že jsem bakalářskou práci na téma

"Návrh vstupního formuláře do lékařského informačního systému"

pod vedením vedoucího bakalářské práce vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 9 května 2013 ……

(Mario Kamburov)

Poděkování

Chtěl bych poděkovat paní Doc. Dr. Ing. Janě Klečkové za vstřícný postoj, motivaci a cenný čas věnovaný vedení mojí bakalářské práci.

Abstract

KAMBUROV, Mario. Proposal entry form to the medical information system: Bachelor thesis. Pilsen, 2013. Bachelor thesis (Bc.) - University of West Bohemia in Pilsen. Faculty of Applied Sciences. Department of Computer Science and Computing, 2013.

Proposal entry form to the medical information system

The main objective of this bachelor thesis is to familiarize with the creation of semantic web applications and create a website to assist in health care. In my work I am using Java Server Page technology and Jena framework, on which I can easily generate forms that are based on ontology that has already been supplied by the Department of Computer Science and Computing, connected to a standard DASTA, which is defined by the Ministry of Health of the Czech Republic. The aim is also to store data from a website form into RDF / XML format. An important part of the work is to propose own algorithms for data processing. The practical part focuses primarily on creating forms that operate on a general principle. At the end I am also focusing on testing of the selected data set.

Keywords:
Ontology, DASTA, JSP, RDF, XML, Jena, Java, Semantic Web, Healthcare, HTML, JavaScript

Obsah

1. Úvod

V současné době, každý z nás už měl určité administrativní vyřizování s lékařem a je mu známo, že využití informačních systémů a technologií jak ve zdravotnictví, tak v sociálních službách v České republice není na stejné úrovni při porovnání s ostatními státy Evropské Unie či Spojenými státy americkými. Nejednotný vývoj těchto technologií by mohl způsobit zbytečné problémy při komunikaci mezi lékaři, lékárníky a pacienty, následovně by mohly vzniknout problémy i v administrativě. Ve svoji práci implementuji internetovou aplikaci zpracovávající lékařské údaje pacientů, pomocí formulářů, jejichž cílem je usnadnit zpracování pacientských údajů a vylepšit komunikaci mezi praktickým lékařem, nemocničním lékařem, lékárnami, zdravotními institucemi atd.

V této bakalářské práci je nastíněna problematika vytváření sémantických webových stránek pro zdravotnictví s cílem usnadnění byrokratické práce lékařů a zdravotníků. Webová aplikace pracuje s ontologiemi, které splňují požadavky Ministerstva zdravotnictví České republiky. Ve svojí práci jsem pracoval konkrétně se standardem DASTA. Ontologie využívající tento standard byla dodána katedrou informatiky a výpočetní techniky Západočeské univerzity. K vývoji této aplikace jsem si zvolil výkonnou moderní technologii JSP (Java Server Page), poskytující řadu možností a užitečných funkcí pro předávání dat a tím i snadnější práci při vývoji systému.

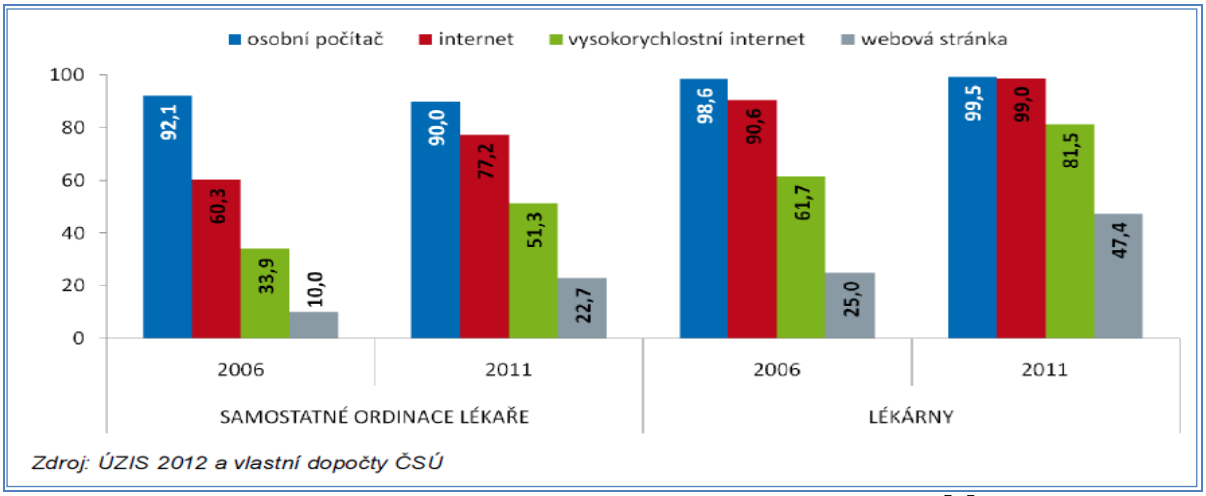
Pro zpracování dat jsem používal výkonnou technologii Frameworku Jena, umožňující snadnou práci se strukturovanými daty. Jejíž pomocí lze jednoduše vygenerovat výstup jednotlivých formulářů do obecného, opakovaně použitelného a snadno-přenosného formátu. Ve svojí práci se zabývám formátem pro ukládání dat RDF.

Na konci mojí práce jsem se věnoval testování funkčnosti systému u různých typů formulářů, týkající se popisu pacienta při příjmu do nemocnice. Soustředil jsem se na aplikaci realizovaných algoritmů i na zpracování dat u reálných údajů pacienta.

2. Analýza momentálního stavu IT ve zdravotnictví v ČR

2.1. Pohled na statistiku

V rámci svojí bakalářské práce jsem provedl malý výzkum týkající se využívání informačních technologii ve zdravotnictví v České republice. Zajímavé je, že jsem narazil na velice rozmanité statistické údaje. Analýza těchto údajů je prováděna pomocí ročního výkazu E(MZ) 4-01 o zaměstnavatelích, evidenčním počtu zaměstnanců a smluvních pracovníků. Průměrná návratnost vyplněných dotazníků byla kolem 90% za období předešlých dvou let. Z celého průzkumu je jasně vidět procentuální využívání ICT ve zdravotnictví v ČR. Šetření se týká v první řadě praktických lékařů a soukromých lékáren.



Obrázek 1: Porovnání využití IT ve zdravotnictví v roce 2006 a 2011. [1]

Z obrázku č. 1 je zřejmé, že skoro všechny zdravotní subjekty používají vlastní počítač a rok od roku s vlivem moderních technologií se značně zvyšuje i využívání internetových technologií. Při vyvíjení vstupních formulářů do lékařského informačního systému jsem se řídil dnešními moderními systémy podobného charakteru, mající za cíl usnadnit práci lékaře, zdravotního pracovníka nebo pacienta. To s čím jsem se setkal v internetu byl nedostatek jakékoliv online služby pro lékaře, lékárníka či pacienta.

S postupem času se rozvoj informačních technologií v ČR i celosvětově vyvíjí čím dál, tím rychleji a tímto se naskytuje stále více možností, jak tyto služby využívat. Pomocí těchto služeb lze zjednodušit práci praktického nebo nemocničního lékaře, při komunikaci s dalšími organizacemi, či usnadnit administrativní činnost v oboru zdravotnictví.

2.2. Aktuální stav ICT ve zdravotnictví

Tato podkapitola popisuje aktuální stav rozvoje a využití informačních a komunikačních technologií ve velkých tuzemských nemocnicích a zdravotnických zařízeních.

Jako páteří českého zdravotnictví jsou považovány všechny velké tuzemské nemocnice, v nichž jsou každoročně hospitalizovány stovky až tisíce pacientů ze všech sociálních vrstev. S ohledem na to, že péče o všechny tyto pacienty je primárně hrazena ze systému veřejného zdravotního pojištění, by se dalo říci, že se pomalu začínají promítat důsledky globální ekonomické krize do rozvoje ve sféře ICT. [2]

Vzhledem k tomu, že výdaje na pořízení a technickou podporu informačních technologií ve velkých tuzemských nemocnicích dosud rostly a v blízké budoucnosti zřejmě porostou i nadále, je zajímavé, že poptávka a zájem o jejich rozvoj stále roste. Z toho by se dalo odvodit to, že české nemocnice by potřebovaly kompaktní, moderní a hlavně dostupnější či levnější řešení v administrativě. Ovšem cílem je dosáhnout zvýšenou efektivitu fungování a to právě výběrem, vhodného informačního systému s jeho případnou integrací s dalšími využívanými aplikacemi či systémy, což usnadní spolupráci mezi zdravotními subjekty do budoucna. Problém však může nastat ve chvíli, kdy bude potřeba uplatnit sjednocování těchto různých systémů a aplikací, následně by to mohlo zvýšit celkové náklady. [3]

Ovšem klíčovou funkcí těchto systémů je spolupráce s obdobnými systémy s cílem vyvíjení schopností pro komunikaci a spolupráci mezi sebou. V mnoho moderních amerických e-Health projektech jsou již implementovaná podobná řešení, která zvyšují efektivitu fungování celkové práce lékařů, zdravotníků a systémů, zabývající se veřejnou správou. Klíčem k úspěchu v moderních státech bývají globalizační trendy, které ovlivňují společnost již dlouhá léta.

S vzhledem k nárůstu moderních technologií a vývoje webu v dnešní době, kdy již existuje verze 2.0 a stále narůstájicí interaktivita pacientů online se také hovoří i o koncepty Health 2.0 či Medicine 2.0 ve zdravotnictví, určené lékařům. Pojem Web 2.0 byl poprvé použit roku 2003 (O'Reilly Media). Neoznačuje novou generaci webu, ale pouze nový způsob jeho využívání, především z hlediska zrychlení řídících procesů uvnitř podniků či veřejných organizací. Přesně tyto projekty Health 2.0 apod. by měli fungovat univerzálně, na obdobném principu pod vlivem nových technologií a to tak, že by měly řešit právě tu chybějící do dnes komunikaci mezi lékařem, laboratořemi a jinými zdravotnickými centry. Dále by měly řešit právě tu chybějící informaci mezi jednotlivými body, které se týkají zdravotnictví. [3]

K řešení problému sjednocování a integrace informačních systémů navzájem se využívá řada moderních technologií pro předávání dat. V mojí práci se zabývám navrhováním formulářů do lékařského informačního systému, založených na obecném principu. Tyto formuláře budou dále používané katedrou pro výzkum a vývoj lékařských informačních systémů určených pro snadnější integraci a sjednocování s dalšími systémy a aplikacemi v oboru zdravotnictví. V pozadí může stát například předávání dat v určitém standardizovaném formátu. Teď si můžeme položit otázku – jak tyto údaje předávat dál tak, aby byly užitečné i jiným organizacím? S touto problematikou se zabývají konkrétní již existující v České republice standardy pro předávání dat mezi informačními systémy zdravotnických zařízení.

3. Používané standardy pro předávání dat v ČR

K dnešnímu dni v České republice existuje několik nejpoužívanějších standardů pro výměnu dat mezi informačními systémy ve zdravotnických zařízení – DASTA, SITS, HL7, DICOM atd. Při tvorbě svého projektu jsem se řídil datovým standardem DASTA Ministerstva zdravotnictví České republiky, který bude v bakalářské práci použit.

3.1. DASTA

DASTA (Datový standard MZ ČR – obecně DS) je Datový standard Ministerstva zdravotnictví České republiky a slouží k předávání dat mezi informačními systémy zdravotnických zařízení v rámci České republiky jakož i na Slovensku. Před 15 lety byla zavedená první verze českého národního standardu pro výměnu informací ve zdravotnictví DASTA. V posledních 9-10 letech DASTA přetrpěla několik změn a transformací z textových souborů přes DTD soubory až ke dnešnímu dni současná verze 4 plně využívá možností XML (je zcela postavena na XML schématech a využívá výhod jmenných prostorů) a je běžně používána jak u nemocničních, tak i u lékařských informačních systémů v českém zdravotnictví. [4]

Standard DASTA je implementován ve většině českých zdravotnických informačních systémech. Prozatím tento standard nenašel žádné uplatnění v zahraničí kromě Slovenska.

3.2. Celkový přehled verzí DASTA

Verze	Datum vydání	Formát
DS 01.20	1.1.2001	"TXT"
DS 02.01	1.5.2002	"DTD"
DS 03.01	2.6.2003	"DTD"
DS 04.01	1.1.2007	"XML"

Tabulka 1: Přehled verzí DASTA

Dnes jsou nejpoužívanější a stěžejní verze DS 3 a DS 4, přechod na DS 4 probíhá v současnosti. Nynější nejnovější verze se od 1.1 2013 stala DS 4.08.01

3.3. Možnosti standardu DASTA

Dnes DASTA umožňuje z osobních údajů pacientů předávání různé informace. Zde jsou uvedené příklady:

- identifikační údaje o pacientech
- základní informace (trvalé bydliště, výška, hmotnost, pohlaví)
- urgentní informace (alergie, krevní skupina)
- platební vztahy, pojišťovny, pracovní neschopnosti
- anamnéza
- využívané léky
- očkování
- momentální diagnóza
- různé druhy vyšetření (lékařské zprávy, laboratorní vyšetření)
- kontakt (adresy - trvalý pobyt, přechodná adresa, email, telefon)
- textové zprávy lékaře (osobní poznámky o pacientovi)
- EKG vyšetření
- Dekurz

DS se ve svých datových blocích dělí hlavně na interní a externí číselníky. Ke dnešnímu dni jich existuje více než 300. Mezi nejvýznamnější číselníky patří blok číselníků pro NZIS a blok číselníků pro laboratorní komplement, jehož nejvýznamnější součástí je Národní číselník laboratorních položek (NČLP). [5]

DASTA obsahuje řadu specifických bloků. Já se ve své bakalářské práci zabývám detailněji s blokem určený pro popis pacientských údajů. V závislosti na číselníku při generování formuláře využívám základní údaje pacienta, adresa, diagnóza, klinické události atd.

3.4. Výhody a nevýhody

Hlavní výhodou a také cílem DS číselníku je snadná identifikace konkrétních odborných lékařských termínů. Tím lze zabránit tomu, že může dojít k redundanci popisovaných dat.

Z druhé strany, z interkulturního pohledu, lze považovat tento standard za ne úplně korektní a to právě kvůli tomu, že chybí mezinárodní podpora. Je podporován pouze na území ČR, na rozdíl například od standardu SITS, který je téměř celosvětově podporován.

3.5. Příklad struktury bloků pro popis pacienta

Ve specifikaci DASTA jsou nadefinovány vždy nejnovější prvky každého bloku DS, jako například:

- jeho textovým popisem ve formě tabulek a poznámek
- jeho přepisem do tvaru DTD (v DS3) nebo XML schéma (v DS4)
- příkladem použití bloku [6]

V definici standardu je používáno průběžně několik základních informací o každém bloku:

- **[Příklad] -** Hypertextový odkaz na příklady řešení bloku - pouze ilustrace řešení.
- **Kód -** Identifikátor pro potřeby XML
- (malými písmeny a bez diakritiky).
- **T -** Typ pro XML (3 typy – a=atribut, e=element, d=data)
- **D -** Délka položky (pro potřebu databází příslušných IS)
- **V -** Výskyt (pro XML):
- **Plný název -** Volným textem ve formě plného názvu položky a případně i její stručné charakteristiky. [6]

*ip – pacient - varianta pro DS4

Základní blok nesoucí data vztažená pouze k jednomu pacientovi. Varianta pouze pro DS4.

{distribuováno od verze 4.01.01}

Kód	T	D	V	plný název	hodnota	podmínky, pokyny, poznámky	změny
id_pac	a	10	1	identifikace pacienta v IS odesilatele	číslo (znakově); v délce 9 nebo 10 znaků	pokyny: viz id_pac - pokyny poznámky: viz id_pac - poznámky	
Jmeno	E	24	?	jméno	lib. text	pokyny: 1. Pokud v bloku "dasta" hodnota atributu ur="N" (hlášení pracovní neschopnosti pro ČSSZ), je jméno povinné. 2. korektní nebo žádné poznámky: viz jmeno - poznámky	
Prameni	e	35	1	příjmení	lib. text	pokyny: pokud není známo, lze i např: "neznámý muž" poznámky: viz prijmeni - poznámky	
dat_dn	e		?	datum a čas narození	formát D nebo DT	pokyny: D nebo DT je zvoleno dle přesnosti zadání údaje	
Sex	e		?	pohlaví	M, F, X viz seznam hodnot		
ku	e		?	klinické události		poznámky: obsahuje soubor klinických událostí různých typů a fází	V DS4 nové.

Tabulka 2: Ukázka části struktury bloku pro popis pacienta podle DS4. [6]

V ontologickém datovém modelu je každý blok reprezentován určitou třídou. V praxi se tyto třídy nemusí shodovat na první pohled s intuitivním chápáním tříd objektu, jak již známe v Javě. Výše uvedený příklad struktury bloku pro popis pacienta obsahuje i další vlastnosti. Tyto vlastnosti mohou být jak datové – pouze textová hodnota, tak i objektové – odkaz na další formulář, patřící k původnímu. Například Klinická událost v bloku pro popis pacienta, která je definována standardem DS4 odkazuje na jinou třídu, popsanou pomocí jiného bloku v specifikaci dokumentu DASTA. V Ontologii, se kterou pracuji, je tato objektová vlastnost, nacházející se ve třídě pro popis pacienta, parametrizována třídou Klinická událost.

4. Analýza návrhu vstupního formuláře do lékařského systému

Shrnutím celkové situace, lze říci, že problematika navrhování jakékoliv IT řešení v oboru zdravotnictví je obsáhlý a složitý proces, spojený s řadou možností a řešení pro splnění požadavků lékařů. Existuje určitý rozdíl mezi administrativními metodikami využívanými jednotlivými státy. Na základě toho se zakládají různé světové a všeobecné standardy pro výměnu dat.

Za posledních několika let zájem o zachování a nasbírání dat vzrostl. To byl hlavní důvod vyvíjet mnoho dalších nových technologií pro práci a manipulaci se strukturovanými údaji, se kterými by výzkumníci pracovali dál. Právě na základě tohoto faktu se začínali objevovat nové technologické trendy jako ontologie nebo sémantické webové aplikace. [7]

4.1. Ontologie

Ontologii lze definovat jako model/nástroj, který reprezentuje znalosti jako soubor pojmů v dané oblasti. Ontologie také zachycuje vztahy mezi těmito pojmy. Je akceptována jako forma řízení znalostí. Zachycuje znalosti v rámci organizace jako model. Tento model, pak může zobrazit vztahy mezi subjekty v rámci celé organizace.

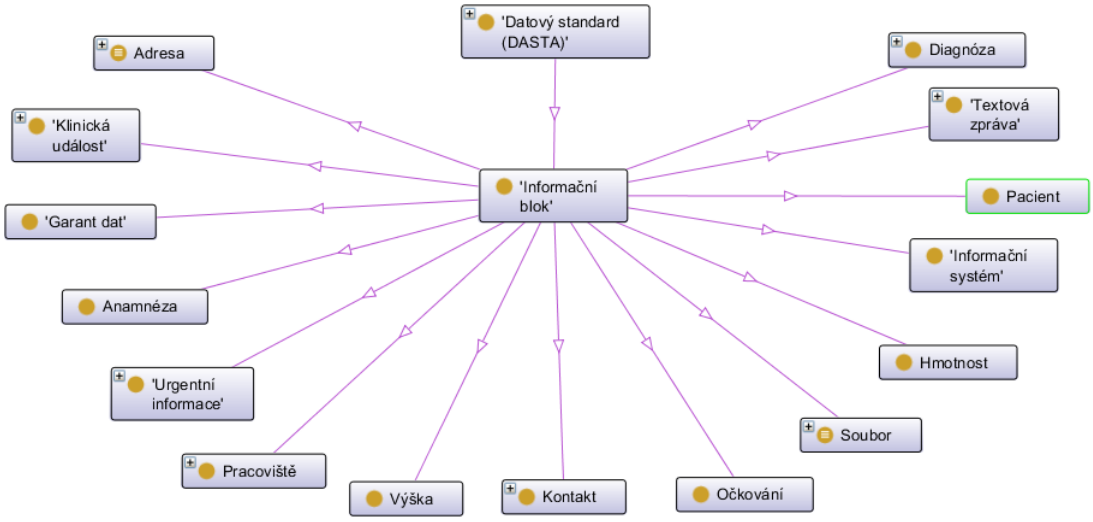
Dnes lidé mají přístup k více údajům, týkající se jednoho konkrétního subjektu. Hlavním problémem však je, že tyto údaje jsou k nalezení v mnoha různých formách. Všechny tyto informace zachycené ve všech těchto různých uložených formátech téměř znemožňují pochopit všechny vztahy mezi těmito daty. V tomto současném prostředí je velmi obtížné určit, jak se mohou zásady zachycené v textových dokumentech vztahovat k obchodním procesům zachycených v modelu a jak se tyto obchodní procesy vztahují k údajům zachycených v databázi atd. Údaje musí být zastoupeny v takovém formátu, ve kterém je možné objevit jeho vztahy (relace) s okolím. Ontologie představuje sběr dat takovým způsobem, který umožňuje zviditelnit jejich vztahy.

Hlavní dva standardy, které pokrývají konstrukci ontologie jsou RDF a OWL. V rámci svojí práce se budu zabývat standardem RDF.

4.1.1. Ontologie DASTA

Cílem vstupního formuláře, založeného na ontologii dodanou již výše citovanou katedrou, je sbírat základní údaje o pacientech. Následně budou tyto údaje předávány dál ve formátu RDF/XML, pro další zpracovávání. Hlavní datový model, ze kterého se budou generovat formuláře v Javě je již hotový a funkční. Konverze standardu DASTA v ontologii je již vytvořený datový model v prostředí Protégé, pomocí kterého se strukturují data do požadovaného formátu Ministerstva zdravotnictví ČR. Mým hlavním úkolem bylo používat tento nebo jiný (SITS), již hotový datový model pro vytváření formulářů.

Následující graf představuje základní část entit, kterých používá ontologie DASTA, se kterou jsem pracoval.



Obrázek 2: Ontologie DASTA

Obrázek č. 2 reprezentuje ontologii, obsahující různé entity, které mají svoje datové či objektové vlastnosti v pozadí. Tyto vlastnosti jsou definovány,

pomocí bloků v specifikaci standardu DASTA[1] a ty já generuji ve formě ustálených slovních spojení, pomocí formulářů na základě tohoto modelu ve svoji bakalářské práci.

4.2. Seznámení se standardem RDF

RDF je zkratka z Resource Description Framework, v češtině známo jako systém pro popis zdrojů. Je to jeden ze standardů World Wide Web Consortium (W3C)[2], jehož původním cílem bylo ukládat meta data, ale následně byl vyvinut jako model pro práci se znalostmi a jejich předávání.

Datový model RDF je syntaxe-neutrálního způsobu, jak reprezentovat RDF výrazy. Tato reprezentace datového modelu se používá k vyhodnocení ekvivalence smyslu. Dva RDF výrazy jsou ekvivalentní tehdy a pouze tehdy, pokud jejich reprezentace datového modelu jsou stejné.

Základní datový RDF model se skládá, na základě principu výroků (známo jako *statements*), ze tří typů prvků tzv. trojice:

- **Subject (Podmět)** - Každý subjekt je zdroj (anglický Resource). Zdroj může být celá webová stránka, jako je HTML dokument "*http://www.w3.org/Overview.html*" například. Zdroj může být součástí webové stránky, např. konkrétní XML element v dokumentu zdroje. Zdroj může být také objekt, který není přímo přístupný prostřednictvím internetu, například tištěné knihy. Zdroje jsou vždy pojmenovány identifikátorem URI a volitelným kotevním ID.

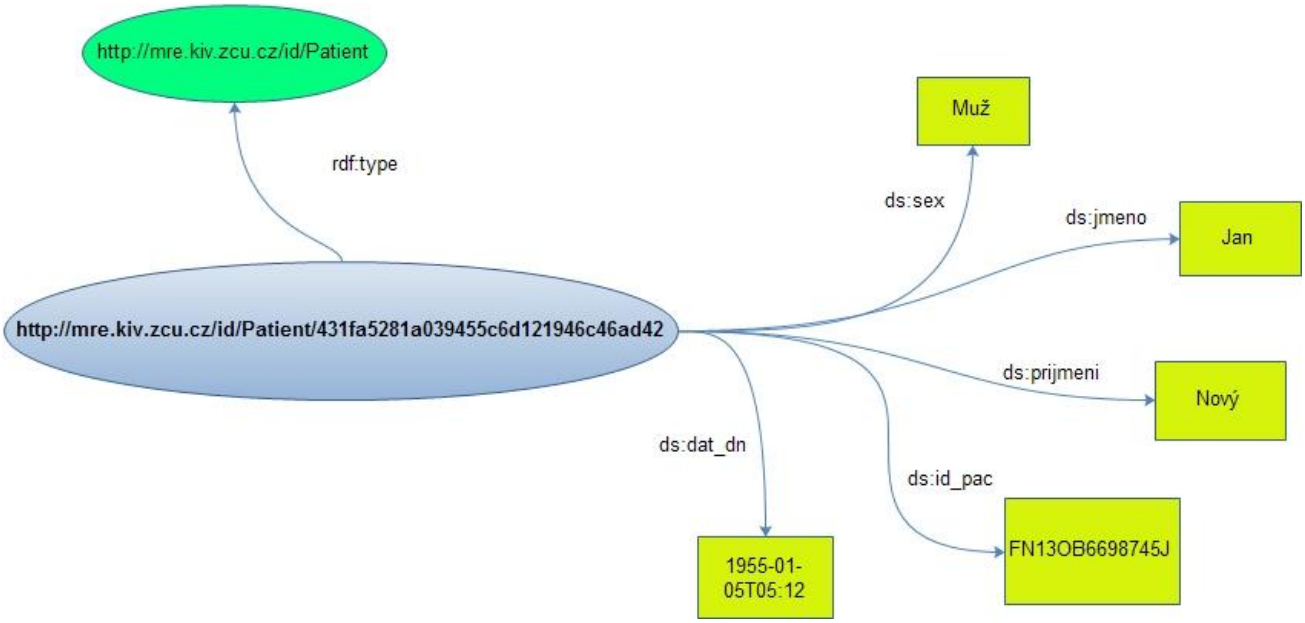
- **Predicate (Predikát)** - Predikát je zvláštní pohled, charakteristika, atribut, vlastnost nebo relace. Každá vlastnost má specifický význam, definuje své povolené hodnoty, typy zdrojů, které může popsat, nebo vztah s ostatními subjekty.

[1] viz.Kapitola 3.5 – příklad struktury bloků DS
[2] RDF: http://www.w3.org/RDF/

- **Object (Předmět)** - Objektem výroku (tj. jeho hodnota) může být dalším zdrojem, nebo to může být tzv. literál, tj. zdroj (specifikován pomocí URI), nebo jednoduchý řetězec nebo jiný primitivní datový typ definován jazykem XML.

Celkový výstup modelu RDF se může považovat za orientovaný graf. Jednotlivé subjekty a objekty lze chápat jako konečné uzly grafu. Vztahy mezi těmito uzly definují orientované hrany, které jsou reprezentovány predikátem, jinak řečeno – vlastnosti, které tyto subjekty a objekty spojují.

Na následujícím orientovaném grafu si ukážeme příklady RDF výroků.



Obrázek 3: Výroky vyjádřeny pomocí RDF trojice

Z obrázku č. 3 lze vyčíst, že Subjekt nebo li Zdroj (Resource), vyjádřen obloukem, je konkrétní, anonymní „**Pacient**“, predikátem jsou vlastnosti jako pohlaví, jméno, příjmení, identifikace pacienta, datum a čas narození či `type`[3], který konstatuje, že zdroj je instance třídy Pacient. Objektem neboli v tomto případě literálem (datový řetězec) jsou datové hodnoty těch vlastností, které

[3] Definice rdf:type podle W3C - http://www.w3.org/TR/rdf-schema/#ch_type

jsou vyjádřeny v grafu obdélníkem. Na tomto grafu lze přečíst minimálně 6 výroků. Jeden z nich může znít takto: „konkrétní anonymní pacient má identifikační číslo FN13OB6698745J“. Další výrok bude: „konkrétní anonymní pacient má datum a čas narození - 5:12 dne 05. 01. 1955.“, kde datum narození je ve standardizovaném formátu XSD `dateTime`[4].

Tyto výroky vypadají následovně ve formátu RDF/XML:

```
<rdf:RDF
    xmlns:mre="http://mre.kiv.zcu.cz/id/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ds="http://mre.kiv.zcu.cz/ontology/2012/01/dasta.owl#" >
  <rdf:Description rdf:about="http://mre.kiv.zcu.cz/id/Patient/
431fa5281a039455c6d121946c46ad42">
    <rdf:type rdf:resource="http://mre.kiv.zcu.cz/id/Patient"/>
    <ds:prijmeni>Nový</ds:prijmeni>
    <ds:sex>Muž</ds:sex>
    <ds:jmeno>Jan</ds:jmeno>
    <ds:id_pac>FN13OB6698745J</ds:id_pac>
    <ds:dat_dn>1955-01-05T05:12</ds:dat_dn>
  </rdf:Description>
</rdf:RDF>
```

4.2.1. Identifikace zdrojů (URI)

Jak již jsme si ukázali příklady zdrojů v předchozí kapitole, teď si vysvětlíme, jak tyto zdroje vznikají.

URI neboli známo také jako Uniform Resource Indentifier lze přeložit jako jednotný identifikátor zdroje. Každý subjekt RDF výroků má svůj zdroj, který je buď pojmenovaný – URI, nebo nepojmenovaný tzv. Blank Node (Prázdný uzel). Nejenom Subjekty, ale i predikáty a objekty můžou mít jednotný identifikátory zdroje. Predikát lze také chápat jako zdroj, který reprezentuje vlastnost nebo li vazba mezi subjektem a objektem. Na druhou stranu objekt může být dalším zdrojem, nikoli jenom datová hodnota. [8]

[4] Konkrétní příklady formátu XSD – http://www.schemacentral.com/sc/xsd/t-xsd_dateTime.html

URI může být dále kvalifikováno jako lokátor, název nebo obojí. Termín "Uniform Resource Locator" (URL) odkazuje na podsadu identifikátorů URI, které kromě identifikace zdroje, poskytují možnosti pro vyhledání zdroje popisující jeho primární přístupový mechanismus (např. jeho "umístění" v sítí). [9]

Jednotlivá schémata nemusí být klasifikována pouze jako jeden z "názvů" nebo "lokátorů". Instance URI, patřící do jakékoliv daného schéma mohou mít charakteristiky názvů nebo lokátorů nebo obojího, často v závislosti na persistenci a jednotnosti v přidělování identifikátorů, tvůrcům RDF výroku. [9]

4.2.2. SPARQL

SPARQL (Protocol and RDF Query Language) je dotazovací jazyk, který je standardizován DAWG[5] a je považován za jeden z klíčových technologií sé [1]mantického webu. Dotazovacích jazyků pro RDF je více a stále se vyvíjí. 15. Ledna 2008 SPARQL 1.0 se stala oficiální doporučení W3C. Dotazovací jazyky pro RDF fungují obdobně jako dotazovací jazyky jiných relačních databází. Na základě konkrétního dotazu vybírají určité hodnoty. [8]

Příklad jeden z dotazů, který jsem používal během vývoje své aplikace vypadá následovně:

```
PREFIX owl:  <http://www.w3.org/2002/07/owl#>"
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX onto: <http://mre.kiv.zcu.cz/ontology/2012/01/dasta.owl#>
SELECT ?O
WHERE {
      onto:Patient owl:hasKey ?Property .
      ?Property rdf:first ?O
}
```

[5] DAWG (RDF Data Access Working group)– pracovní skupina W3C, http://www.w3.org/TR/2005/WD-rdf-dawg-uc-20050325/

Tento dotaz nám vrátí všechny vlastnosti třídy „Patient“, které jsou nadefinovány ontologií DASTA. Jak je vidět na první pohled SPARQL umožňuje skládání dotazů z trojitých vzorů (tzv. trojice).

4.2.3. Syntaxe

S cílem ukládání do souboru a snadnější výměna dat mezi lékařskými aplikacemi či systémy je potřeba výstupy lékařských formulářů serializovat do určitého formátu. K převádění do textové podoby slouží standardizovaná syntaxe. Standard RDF podporuje několik syntaxí, však tu, se kterou se zabývám já ve své bakalářské práci je RDF/XML. Za použití výkonné technologie Frameworku `Jena`[6] lze převádět výstupy internetových formulářů do požadované syntaxe.

4.3. Možná řešení

Během vývoje aplikace nastávaly různé situace, kdy bylo potřeba analyzovat jakým způsobem vyřešit určité problémy nebo jak implementovat konkrétní řešení.

4.3.1. Generování vlastností

Na základě několika postřehů při práce s Frameworkem Jena jsem váhal jak generovat konkrétní vlastnosti dané třídy. Jedna z možností byla použít rychlé a výkonné metody Frameworku Jena a prolistovat vlastnosti této třídy používané ontologie. Další možnost byla přes SPARQL dotaz. Nakonec jsem vybral kombinaci těchto metod, kde pomocí SPARQL dotaz získávám URI jednotlivých vlastností a následně je využívám pro zpracování a další manipulaci s nimi ve Frameworku Jena.

[6] Jena API - Viz. Kapitola 7.2.

4.3.2. Práce s ontologickými modely

Další podstatné rozhodnutí připadalo v úvahu při serializaci dat do požadované notace formátu RDF. Při vyplňování jednotlivých formulářů, uživatel (dále jen lékař) vytváří obsah a terminologickou náplň, týkající se konkrétního pacienta. V aplikaci každý formulář při svém uložení nebo aktualizaci vytváří zvláštní Model. Rozhodnutí spočívalo v tom, jaký model bych měl využívat k načítání ontologie DASTA do paměti a následně práci s ním a na druhou stranu jaký model bych měl využívat pro ukládání dat do souboru v požadovaném formátu. Výhodou tohoto Frameworku je to že poskytuje dodatečné funkce pro manipulaci s ontologií při vytváření ontologického modelu. Z hlediska obecnosti jsem se rozhodl použít ontologický model[7] pro práci a manipulaci s ontologií DASTA. Důvodem byla maximální kompatibilita s předchozí verzí Jena a přednastavení hodnoty jako načítání modelu do paměti pro rychlejší práci s ním (tzv. in-memory storage) nebo RDFS odvozování. Právě za pomoci takovéhoto odvozování (z angl. inference) lze získávat nadřazené nebo podřazené (rdfs:subClassOf) vlastnosti třídy. Jinými slovy funguje určitá dědičnost. Pomocí těchto ontologických modelů máme na výběr, zda chceme používat všechny možnosti ontologického rozhraní či nikoli. Pokud ano, tak je potřeba rozhodnout se, které z různých reasoners použít. Někdy `Reasoner`[8] přidává informaci do ontologického modelu, která není vhodná pro aplikaci nebo zbytečně zatěžuje model. Toto byl hlavní důvod, proč k ukládání lékařských dat do souboru, jsem se rozhodl, že budu používat nejjednodušší způsob vytváření modelu, pomocí `ModelFactory.createDefaultModel()`, který nemá žádné speciální odvozování nebo ontologické rozhraní. Více o vytváření modelu v kapitole 7.2 Jena API.

[7] Vytváření ontologických modelu - http://jena.apache.org/documentation/ontology/#creating-ontology-models

[8] Reasoner – část softwaru, schopná odvodit logické důsledky, na základě ontologického modelu.

5. Návrh formuláře do lékařského informačního systému

Samotný návrh vstupních formulářů je založen na sémantické technologii pro tvorbu webových aplikací. Za použití dodané již hotové ontologie jsem generoval formuláře ve formě ustálených slovních spojení a pomocí výběru menu. V určitých případech bylo potřeba doplnit funkční ontologie, odpovídající standardům Ministerstva zdravotnictví ČR o nutných anotacích, což mi umožnilo snadnější generování štítků v češtině a angličtině a dalších anotací, kterých bylo potřeba pro generování obecných formulářů.

5.1. Vymezení cíle práce

Hlavním úkolem práce bylo navrhnout webovou aplikaci, umožňující generování online formulářů, popisující pacientské údaje v různých formách, tak aby byly použitelné a upravovatelné. Každý z vygenerovaných formulářů představuje konkrétní třídu předem nadefinované ontologie, pomocí bloků, jako například (Anamnézy, Diagnózy, Lékařské zprávy apod.), které jsou nezbytnou součástí práce lékaře při vyšetření pacienta při příjmu do nemocnice.

5.2. Příprava formulářů

Příprava lékařských formulářů probíhala ve formě vytvoření základních webových stránek a seznámení s tvorbou sémantických webových aplikací. Za použití moderních technologií `JSP`[9], `HTML5` a další jsem připravoval základní vzhled webových stránek lékařských formulářů na úrovni prezentační vrstvy této aplikace. Po doplnění potřebných anotací v dodané ontologii jsem získal možnost generování štítku a popisku jak v angličtině, tak i v češtině, což mi velmi usnadnilo práci a tím jsem taky z hlediska obecnosti vytvořil znovupoužitelný kód.

[9] Java Server Pages - viz. Kapitola 7.1.

5.3. Popis pacienta

V dnešní době existuje mnoho informačních systémů i aplikací napomáhající lékařům v administrativní práci a digitalizaci všeho druhu papírování spojené s různým vyšetřováním pacientů. Při tvorbě formulářů jsem se vedl klíčovým standardem Ministerstva zdravotnictví ČR pro popis pacientských údajů - DASTA[10]. V praxi klasický příjem pacienta do nemocnice nebo jiného zdravotního zařízení může vyžadovat vyplnění mnoha formulářů, které se týkají jak momentálního zdravotního stavu pacienta, tak i jeho chronických, dlouhodobých onemocnění. Ve svoji aplikace navrhuji formuláře, usnadňující práci lékaře při vyplňování popisu pacienta ve formě vyskakovacích oken, ve kterých se nachází další formuláře, vztažené ke konkrétnímu pacientovi. Při návrhu webové aplikace jsem využíval základní formulářové tagy[11] a atributy[12] pro jednotlivá políčka, týkající se popisu pacienta. Nejpoužívanější z nich byla vstupní pole, definovány tagem input pro zadání vstupu z klávesnice. Další tagy se týkají objektových vlastností dané ontologie, kde v každém formuláři jsou reprezentovány tlačítkem pro přidání nové instance třídy. Například pokud lékař vyplňuje formulář Pacient a chce přidat více lékařských zpráv nebo více diagnóz k jedné lékařské zprávě, patřící ke konkrétnímu pacientovi atd.

5.4. Preferované editační prostředí

Pro práci s dodanými ontologiemi katedrou jsem si zvolil prostředí Protégé[13]. Pomocí něj lze velmi snadno a rychle editovat a přidávat potřebné anotace k jednotlivým vlastnostem nadefinovaných standardů DASTA atd.

Protégé je bezúplatná, open-source platforma, která poskytuje rostoucí sadu nástrojů pro postavení doménových modelů a aplikací založených na znalostech problematiky, pomocí ontologií. [10]

[10] DASTA - viz. Kapitola 3.1.
[11] Popis formulářových elementu (tagu) - http://www.w3schools.com/tags/tag_form.asp
[12] Popis atributů tagu input - http://www.w3schools.com/tags/att_input_type.asp
[13] Editační prostředí Protégé - http://protege.stanford.edu/

Platforma Protégé podporuje dva hlavní způsoby modelování ontologií přes Protégé-rámce a Protégé-OWL editorů. Protégé ontologie lze exportovat do různých formátů, včetně RDF (S) a XML Schéma.

Protégé je vyvíjeno v Javě, je rozšiřitelné a poskytuje hodně pluginů do stávajícího prostředí, které z něj dělají flexibilní základnu pro vývoj aplikací.

Protégé je podporováno silnou komunitou vývojářů, akademických, vládních a podnikových uživatelů, kteří používají Protégé pro znalostní řešení v rozmanitých oblastech jako například biomedicína, umělá inteligence a firemní modelování. [10]

5.5. Výstupy formulářů

5.5.1. Příklady užití RDF při práce s formuláři

Výstupem generovaných formulářů jsou soubory ve formátu RDF/XML. Algoritmy jsou navrhované tak, aby bylo možné aktualizovat jednotlivé výstupy pacientských dat kdykoliv. Editace umožňuje snadnou korekci starých údajů u konkrétního zdroje (např.: Přechodná adresa pacienta) za nové.

Pomocí Frameworku Jena, lze tyto výstupy generovat ve všech notací standardu RDF a to právě: N-TRIPLE, TURTLE, nebo RDF/XML.

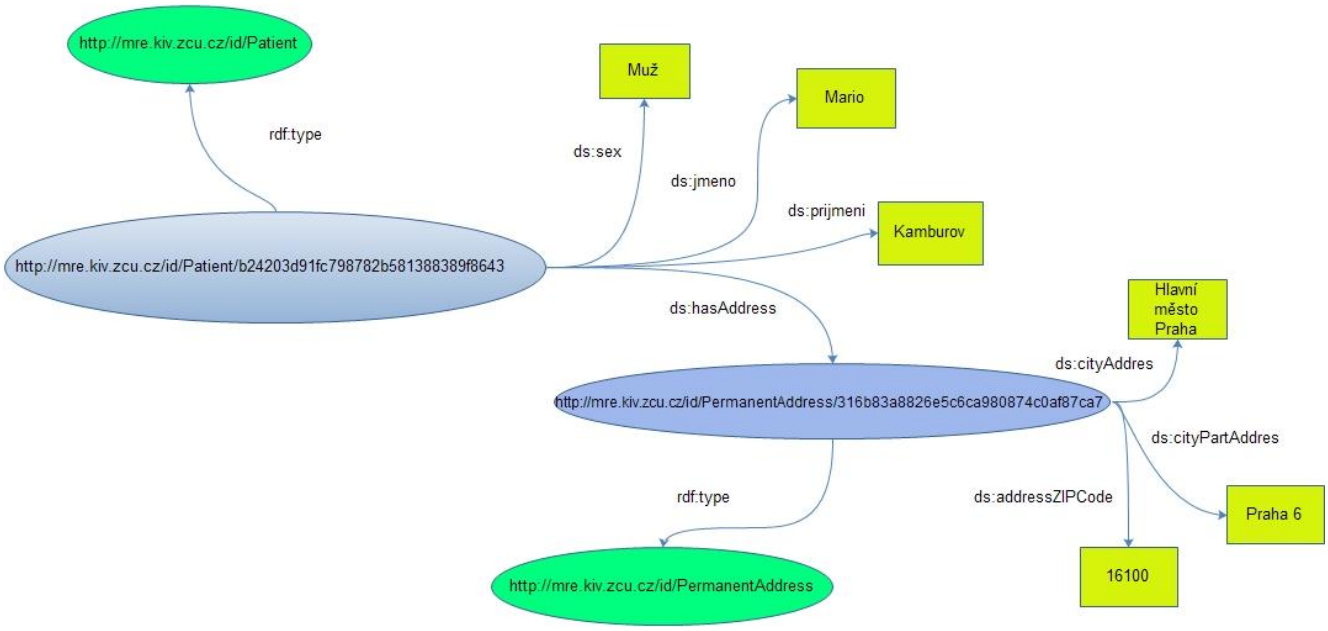
```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
    xmlns:mre="http:/$/mre.kiv.zcu.cz/id/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ds="http://mre.kiv.zcu.cz/ontology/2012/01/dasta.owl#" >

<rdf:Description rdf:about="http://mre.kiv.zcu.cz/id/PermanentAddress/
316b83a8826e5c6ca980874c0af87ca7">
    <rdf:type rdf:resource="http://mre.kiv.zcu.cz/id/PermanentAddress"/>
    <ds:cityPartAddress>Praha 6</ds:cityPartAddress>
    <ds:cityAddress>Hlavni město Praha</ds:cityAddress>
    <ds:addressZIPCode>16100</ds:addressZIPCode>
  </rdf:Description>

  <rdf:Description rdf:about="http://mre.kiv.zcu.cz/id/Patient/
b24203d91fc798782b581388389f8643">
    <rdf:type rdf:resource="http://mre.kiv.zcu.cz/id/Patient"/>
    <ds:prijmeni>Kamburov</ds:prijmeni>
    <ds:sex>Muž</ds:sex>
    <ds:jmeno>Mario</ds:jmeno>
    <ds:hasAddress rdf:resource="http://mre.kiv.zcu.cz/id/PermanentAddress/
316b83a8826e5c6ca980874c0af87ca7"/>
  </rdf:Description>
</rdf:RDF>
```

Příklad souboru dat 1: Soubor dat pacientských údajů

Na příkladu souboru dat č. 1, je znázorněn výstup pacientských údajů ve formátu RDF/XML. Z něj lze vyčíst, že pacient s unikátním identifikátorem, který je muž a má jméno Mario a příjmení Kamburov, má také odkaz na instance třídy trvalá adresa, která je unikátní a patří tomu pacientovi. Ta adresa má podle standardu DASTA položky Okres, Město, PSČ atd. Tento zápis vypadá následovně v grafickém znázornění:



Obrázek 4: Ukázka výstupu popsaný, pomocí RDF/XML

Z obrázku č. 4 lze vyvodit to, že formát RDF/XML se do jisté míry podobá formátu XML nebo-li je jasně vidět fungující stromová struktura grafu. RDF má velkou expresivitu, právě díky ní lze neomezeně popisovat poměrně složité datové struktury, což přináší jedině výhody ve svéře kde je potřeba popisovat velké množství dat, jako je zdravotnictví nebo v jiných organizacích či institucích, využívajicích informační systémy veřejné správy[14] nebo aplikace podobného charakteru.

Jak je vidět přidáváním libovolných datových hodnot pro konkrétní zdroj (v našém případě Trvalá adresa) přes frameworku Jena je velmi jednoduché.

[14] ISVS - http://www.isvs.cz/

Ovšem přidávání složitějších datových struktur je zajímavější. Na obrázku č. 4 si můžeme všimnout, že objektové vlastnosti v navrhnutých výstupech se řídí podle vzoru uzel-hrana-uzel. Při pohledu na soubor dat č. 1, můžete vidět, že při práci s knihovnou Jena jsem navrhnul, aby instance třídy Trvalá adresa konkrétního pacienta odkazovala na zdroj Trvalá adresa, pomocí vlastnosti `rdf:hasAddress` a tím odděluji zdroj Trvalá adresa, místo jejího dalšího vnoření. Ve výsledku máme přehlednou syntaxi. Tím výsledný RDF graf stále odpovídá svému významu. Tento příklad ukazuje jak implementovat prokládánou XML kvalitu RDF, pomocí tzv. datových ostrovů. Další možnost serializace modelu vyplněných dat do formátu RDF je pomocí návrhového vzoru Post-Con, který využívá kontejner obsahující vlastnosti zdroje.

5.5.2. Metadata pacientů

Denně lékaři vyplňují pacientské údaje, jak ručně tak i automaticky, ale aby existovalo určité rozlišení těchto dat a nenastávaly chyby při mnohonásobném zpracovávání údajů obdobných dat se používají opatření jako popisování dat. Je potřeba vědět v rámci konkrétního popisu pacienta (osobní údaje, diagnóza, anamnéza atd) zda li text je vyplněn ve správném poli nebo je použitelný pro danou konkrétní charakteristiku. Jakmile každá textová hodnota je zadaná lékařem do formuláře, pomocí využívání standardu RDF, okamžitě se přidávají i popisky, určující jejich vlastnosti. Tím lze odvodit také to, že zpracovávání pacientských údajů, se kterými lékaři pracují každodenně, nebude dávat smysl, aniž by existovala konkrétní metadata dat, popisující jejich vlastnosti. [11]

Výstupy webové aplikace jsou navržené tak, aby všechna data, týkající se jednoho konkrétního anonymního pacienta byla v jednom souboru. Tím způsobem lze shromažďovat nekonečný počet informačních dat a lékařských údajů o konkrétním pacientovi, tak aby byly snadno přenositelné a integrovatelné s jinými aplikacemi podobného charakteru. Ve svojí práci jsem navrhnul i aktualizaci již existujících pacientských údajů ve formě

očekávaných připadů využitím formulářů podobného typu. Například v již existujicím formuláři pro diagnózu určitého pacienta lze přidávat, měnit a mazat údaje aniž by ovlivňovaly předchozí údaje anonymního pacienta. Všechno toto je možné díky standardu RDF a použití moderných technologií a knihovny pro zpracování dat, které využívám ve svojí práci.

Ve svojí bakalářské práci generuji obdobné výstupy formulářů, výše uvedeným principem, na bázi vzoru uzel-hrana-uzel. Dále výše uvedený příklad, pouze znázorňuje strukturu ukládaných dat. V rámci svojí práce testuji aplikaci nad velkým množstvím dat, která výsledně vypadá mnohem komplexněji s tím, že princip zůstává stejný. Model je vždy možné upravovat a aktualizovat, a právě pomocí přehledné struktury a výkonné technologie Frameworku Jena je snadno upravovatelný. Při editaci konkrétního zdroje stačí pouze přistoupit odpovídajícím způsobem do konkrétního zdroje a přepsat nebo manupulovat s jeho hodnotami, dále i s údaji pacienta.

5.6. Dodatečné technologie

K řešení konkrétních menších problémů jsem ve svojí práci navrhnul použití několik nových technologií.

V situaci kdy zdravotní pracovník, případně lékař by chtěl přidat více lékařských zpráv nebo více než jednu trvalou či aktuální diagnózu bylo potřeba manipulovat s více daty stejného typu. Otázkou bylo však, jak tyto údaje předávat dál, manipulovat s nimi či identifikovat, která diagnóza je první a která je další v pořadí, ve kterém jsou přidávané. K vyřešení tohoto problému jsem použil menší knihovnu, která se podobá asociativnímu kontejneru s hash tabulkou `HashMap` v Javě. Navrhnuté ode mne řešení se zakládá na javovskou knihovnu Guava[15], která podporuje velké množství tříd pro manipulaci s kolekcemi dat. Hlavním důvodem jejího použití bylo vyřešení problému s relacemi 1 : N, kde k jednomu pacientovi jsou vztažené více lékařských zpráv, diagnóz různého druhu atd.

[15] Google Guava – knihovny pro manipulace s kolekcemi. viz.7.3

6. Implementace

V této kapitole nastiňuji navrhnuté ode mne řešení při vývoji webové aplikace. Správnost navržených algoritmů, lze ověřit testováním nad vybranou množinou dat. Následně plynoucí výstupy, navrhované v předchozí kapitole lze načíst zpět do aplikace, s cílem předvyplnění i následné editace formuláře s pacientskými daty.

6.1. Přidané anotace

Jak bylo již zmíněno v návrhu aplikace, přidal jsem potřebné anotace do ontologie s cílem snadnějšího generování vlastností konkrétních tříd. Každá z generovaných vlastností je specifická, proto je vhodné je rozlišovat obecným způsobem, pomocí anotací. Přidávané anotace jsou identifikovány v ontologii, pomocí zvolené ode mne URI, které hraje roli spíše jako název anotací než lokátor (http:/mre.kiv.zcu.cz/inputType). Ovšem právě na základě tohoto URI, které představuje Annotation Property[16] jsou určované jednotlivé typy polí v HTML, které budou zadávany z klávesnice. Pomocí něj lze snadno získat jednotlivé názvy přidaných anotací pro konkrétní typy vlastnosti (textové pole či datum apod.).

Název přidané anotace	Vzhled v HTML5	Popis
text	`<input type="text" ……>`	Klasické vstupní pole
radio	`<input type="radio" ……>`	Přepínač (aktivní vždy pouze jedna volba)
textarea	`<textarea name=……>`	Široké pole pro zadávání delších textů
date	`<input type="datetime-local"…>`	Reprezentuje datum a čas
hidden	`<input type="hidden" ……>`	Skryté pole bez možnost změny

selectbox	`<select>`
	`<option>`
	`</option>`
	`</select>`
		Výběr z několika možnosti (instance třídy)

addButton	`<button type="button" …>Přidat aktuální</button>`	Tlačítko, přidávající instance třídy

Tabulka 3: Přehled doplněných anotací

[16] Annotation Property - mohou být použity pro anotaci ontologii, axiomy nebo IRI. Struktura anotací je dále popsána zde.

V tabulce č. 3 jsou znázorněny náhodné ode mne zvolené názvy anotací, které hrají roli při generování popisek dat. Pomocí nich lze zajistit rychlejší a snadnější rozpoznání typu vlastností, při jejich generování ze standardizované ontologie. Podstatnější anotace jsou ty, které manipulují s objektovými vlastnostmi. V tabulce č. 3 se jedná o anotaci `selectbox` a `addButton`. Anotace `selectbox` odkazuje na další formulář, po vybrání určité hodnoty z rolovací nabídky, což vytváří instance vybrané třídy v původním formuláři. Další klíčovou, přidanou anotaci jsem pojmenoval `addButton`. Ta znovu odkazuje na další formulář a vytváří instance třídy v původním formuláři, ale u ní, však již není rolovací nabídka, protože je vztažená pouze k jedné třídě nadefinované ontologie (jedná se především o menší formuláře jako Hmotnost, Garant dat apod.).

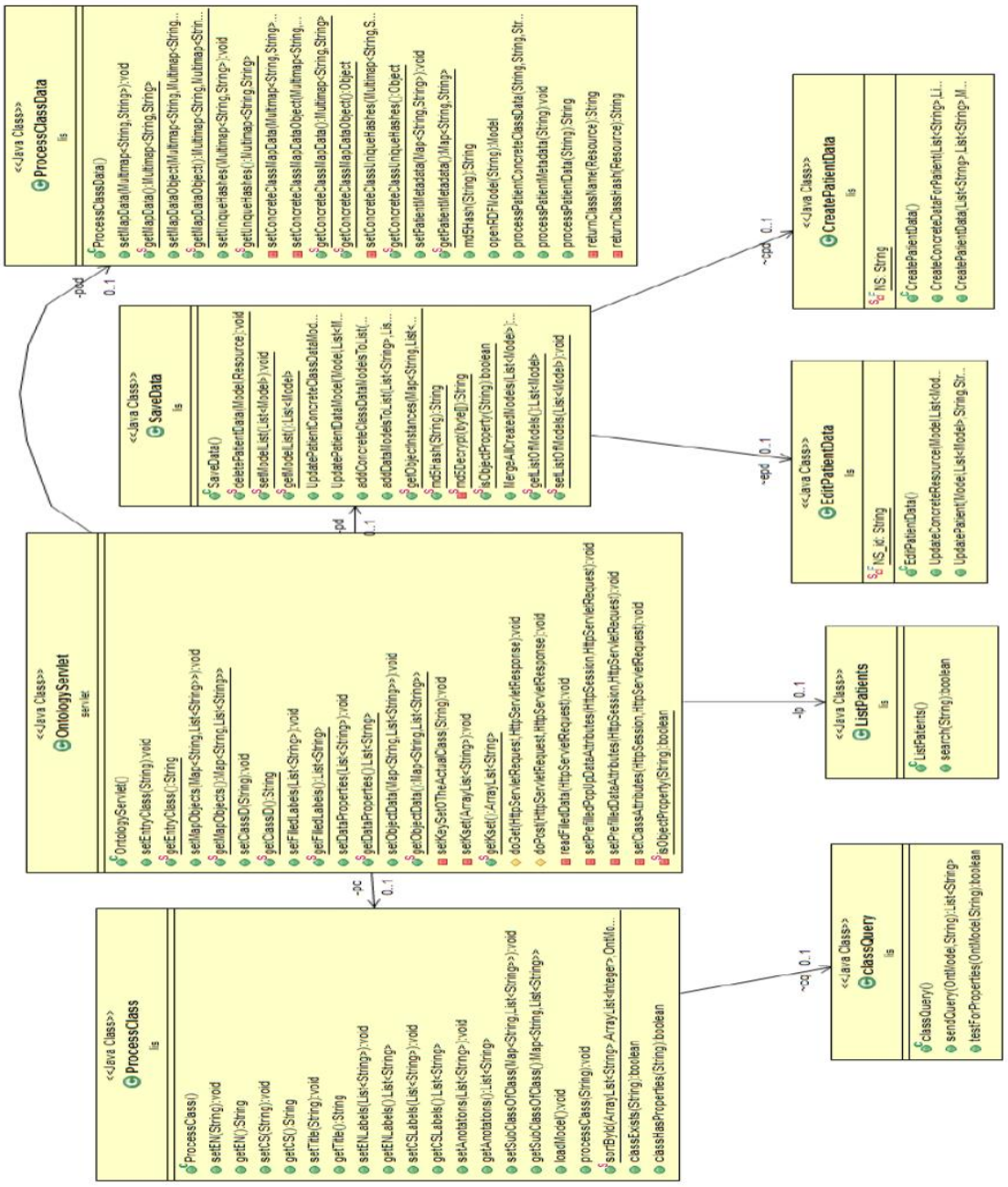
Další předpřípravou ontologie ke generování a práce s ní, se týkalo přidání anotací, určujících řazení jednotlivých popisek, tak aby posloupnost popisek byla asociativní a přehledná. Pomocí základních řadících algoritmů v Javě a přidáním identifikátorů v ontologii přes anotace, určujících konkrétní pořadí jednotlivých vlastností jsem generoval popisky přehledně a podle mého názoru užitečně, podle důležitosti a nejčastější použitelnosti. Ovšem jednoduchost editačního prostředí umožňuje snadná manipulace anotací, díky čemu lze přehodnotit důležitost jednotlivých prvků jako pořadí názvu lékařské zprávy nebo pořadí datumu aktuálního přidání apod., tak aby vyhovovaly lékařského subjektu či zdravotníkovi pracujícímu s formulářem.

6.2. Návrh vlastních algoritmů zpracování údajů

Při navrhování vlastních algoritmů týkající se zpracování vyplněných údajů jsem zvažoval následující scénáře nebo případy využití. Za prvé jsem potřeboval jednu třídu pro práci a manipulaci s načtením ontologie do paměti. Následně jsem s její pomocí generoval jednotlivé vlastnosti, které fungují jako popisky (nebo labels) jednotlivých polí ve webovém formuláři. Dále jsem potřeboval další třídu pro ukládání dat do požadovaného formátu standardu RDF,

tak aby data byla použitelná a snadno-integrovatelná s jinými aplikacemi podobného charakteru. Pomocí kolekce `Enumeration` v Javě jsem procházel všechny zadané hodnoty zdravotního subjektu a ukládal je do požadovaného formátu. Další případ užití, který jsem řešil, byl naimplementovat do zvláštní třídy akce, které se týkají náhledu pacienta, jakož i jeho editace či vymazání. Třída zpracovávající aktualizaci údajů spolupracuje s třídou pro uložení dat do souboru.

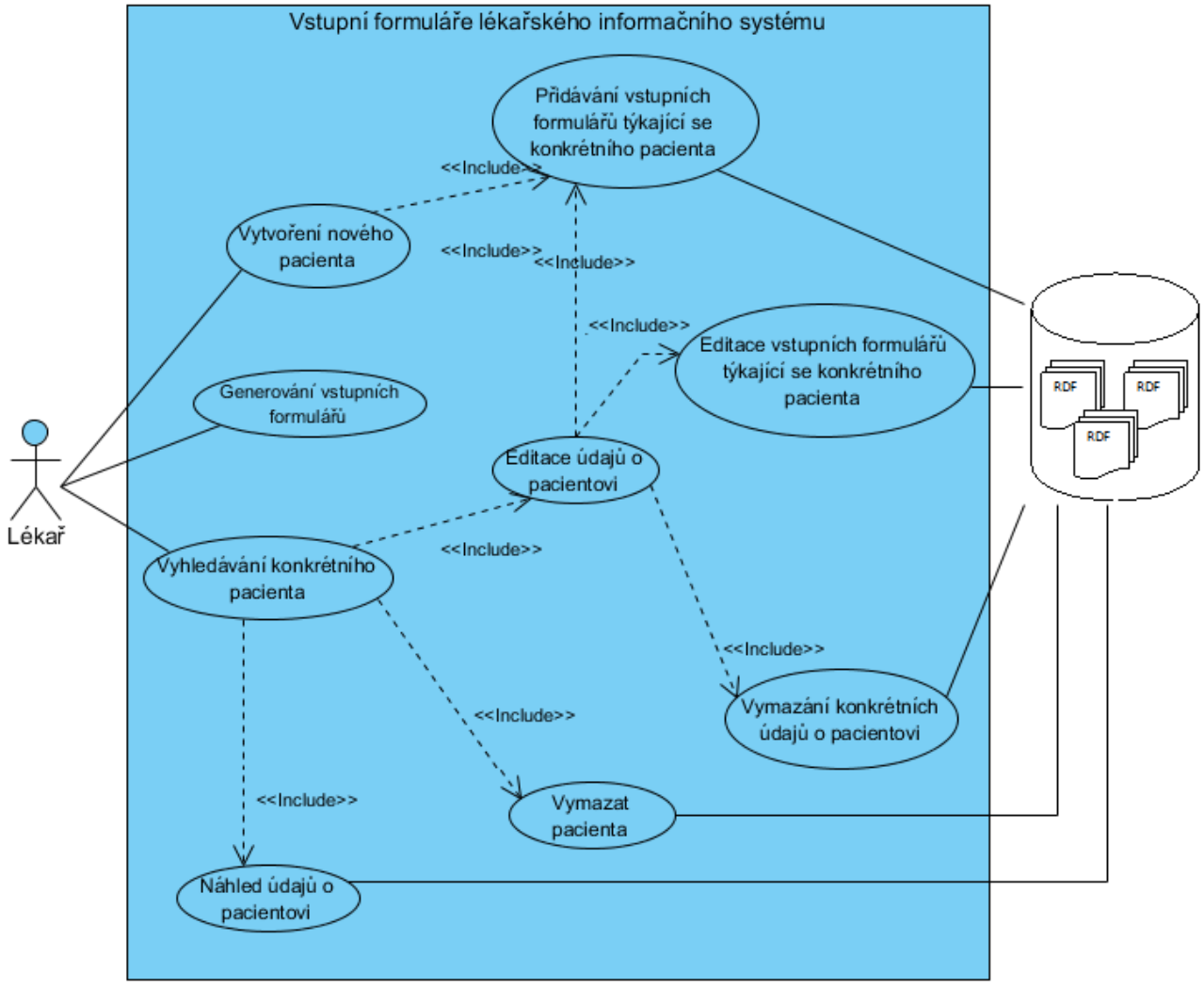
6.2.1. Diagram tříd



Obrázek 5: Diagram tříd navrhované aplikace

Na obrázku č. 5 je zobrazen navrhnutý ode mne koncept tříd webové aplikace lékařského formuláře. Za použití výkonných technologií JSP, Jena apod. lze vytvořit požadovanou interakci mezi servletem a klientem. Servlet zpracovává klientské požadavky tak, že na základě požadované akce lékaře (přidat pacient, vyhledat pacient, editovat údaje pacienta či smazat pacient) spolupracuje s těmito třídami. Každá z těchto tříd má svůj účel.

6.2.2. Případy užití (Use Case Diagram)



Obrázek 6: Diagram případu užití navrhnutého lékařského formuláře

Z UseCase diagramu, lze zpozorovat několik základních případů užití. Při návrhu aplikace jsem uvažoval o jakési hlavní postavě či aktéru, který by

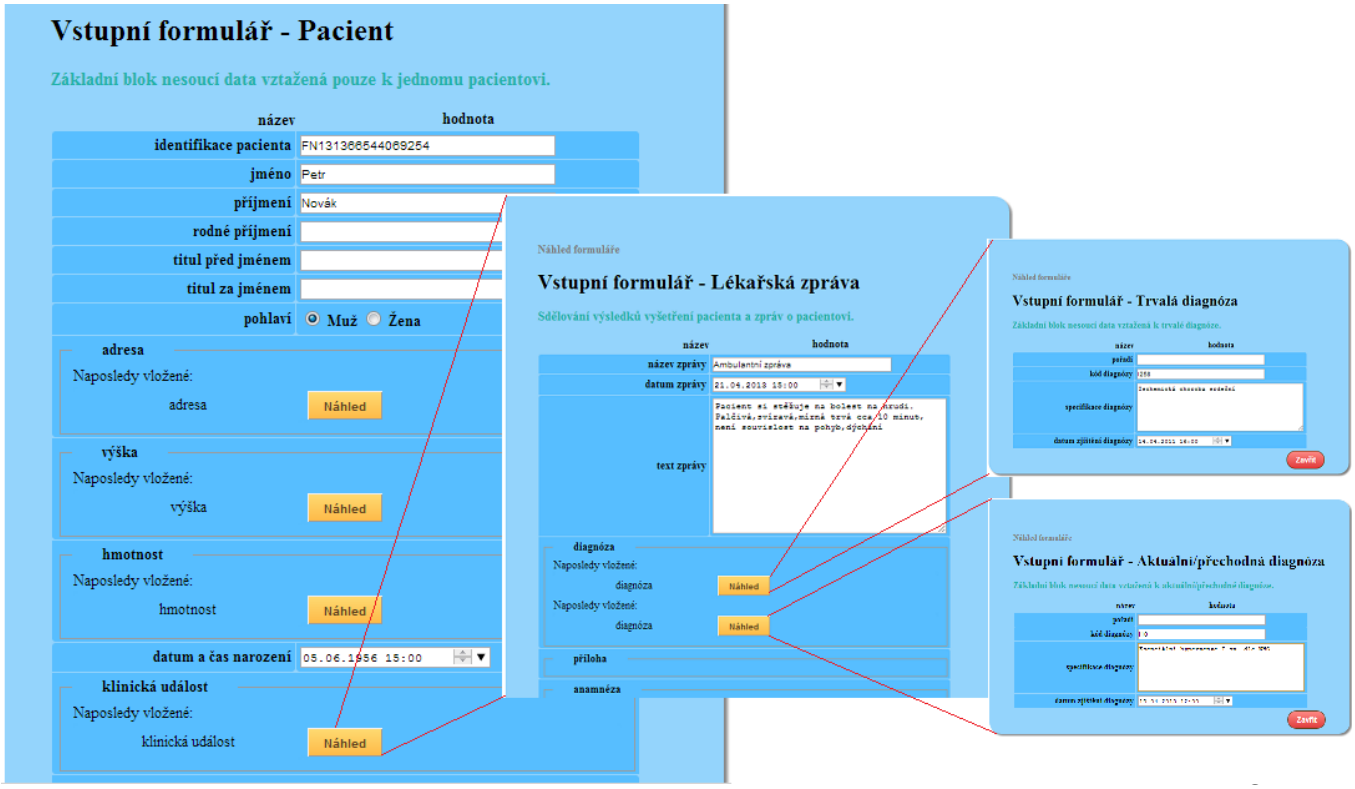
mohl být jak nemocniční, praktický osobní lékař, tak i správce systému nebo jiný zdravotní pracovník, který má přístup k lékařskému informačnímu systému. Implementoval jsem základní akce, známe také, ve frond-end-developingu, jako operace CRUD (Create, Update, Read, Delete). Zdravotní subjekt má možnost plně manipulovat s pacientskými daty – přidávat pacienty, vymazávat pacienty, editovat jejich údaje ve formě předvyplněných formulářů a podformulářů.

6.3. Ukázková aplikace

6.3.1. Struktura internetové aplikace

Vzhled internetové aplikace jsem navrhoval tak, aby aplikace splňovala zásady intuitivnosti a jednoduchosti při práci s formulářem:

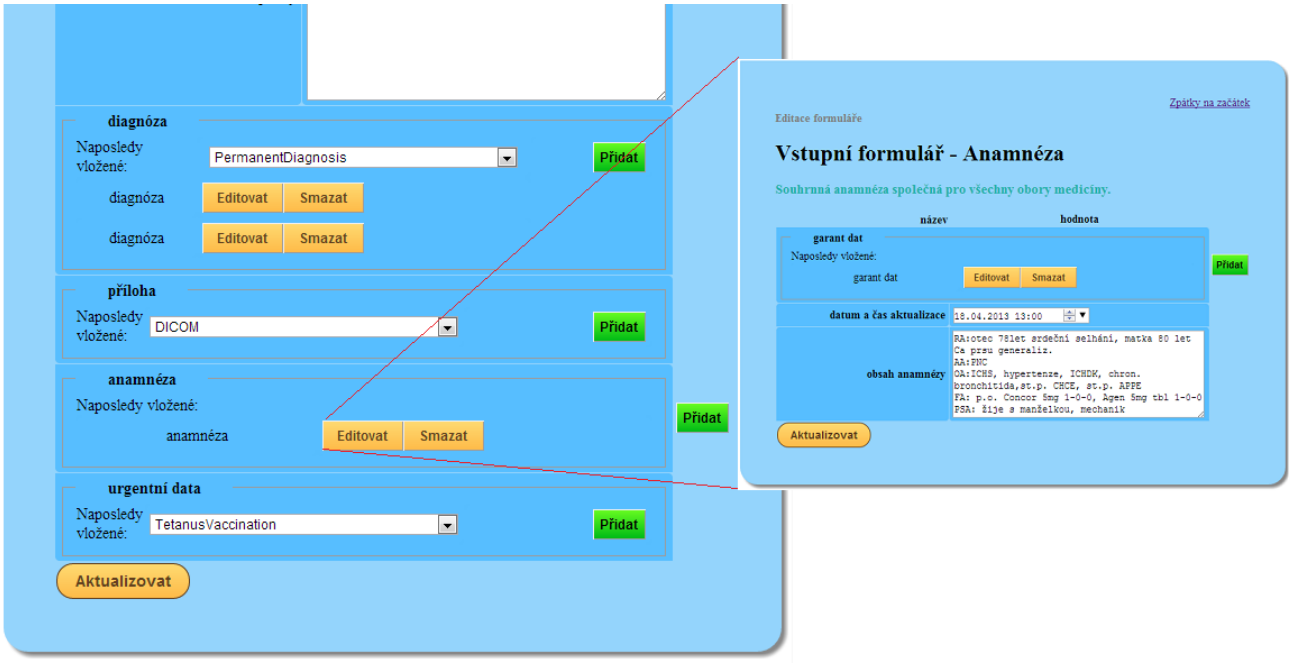
- ✓ Přehlednost – snadné ovládání, jednoduchá uspořádanost jednotlivých prvků, intuitivní orientace.
- ✓ Styl – vhodnost ve výběru kontrastů a barev pro modernější a formální vzhled.
- ✓ Průhlednost – predikovatelné reakce navrženého systému.



Obrázek 7: Rozhraní ukázkové aplikace.

Mojí snahou bylo vytvořit grafické uživatelské rozhraní lékařského systému, které podporuje jednotný vzhled všech formulářů a umožňuje uživatelům (lékařům) jednoduchý přístup k informacím jednotlivých pacientů.

Na obrázku č. 7 je vidět můj návrh rozhraní aplikace. Jak lze na první pohled zpozorovat rozhraní je realizováno pomocí vyskakovacích oken. Na ukázce je příklad náhledu již přidané lékařské zprávy, která obsahuje dvě diagnózy a další údaje pacienta. Tento návrh funguje obecně. To znamená, že po doplnění ontologie DASTA o chybějící vlastnosti a anotaci, lze generovat obdobné formuláře do nekonečna.



Obrázek 8: Rozhraní ukázkové aplikace pro editace pacientských údajů

Na obrázku č. 8 je rozhraní ukázkové aplikace určené k editaci jednotlivých formulářů a podformulářů jednoho konkrétního pacienta. Problém s výpisem pro možnost editace vícenásobných formulářů stejného typu jsem řešil, pomocí fieldset v Html, ve kterém se nachází všechny vyplněné formuláře stejného typu. K vyřešení tohoto problému jsem v použil kolekce `MultiMap`, zakládající se na knihovně Google Guava, kde jako klíče mohou být uvedeny hodnoty stejného typu, v tomto případě je diagnóza uvedena vícekrát. To co je rozlišuje jsou unikátní hodnoty klíče, předáváné dál v parametrech do dalších podformulářů ve formě vyskakovacích oken.

7. Použité technologie

V této kapitole představím všechny použité ode mne technologie v moji bakalářské práci. S polovinou z nich jsem se setkal poprvé a realizováním této aplikace jsem se v nich naučil dobře se orientovat.

7.1. Seznámení s technologii JSP

Java Server Pages (JSP) je technologie, která pomáhá vývojářům vytvářet dynamické generované stránky založené na HTML, XML nebo dalších typech zdrojů. Svůj rozvoj začíná v roce 1999 ve společnosti Sun Microsystems. Podobá se jazyku PHP, ale syntaxe je v Javě. K používání Java Server Pages je zapotřebí kompatibilní webový server se servlet kontejnerem, jako je například Apache TomCat nebo Jetty. Velkou výhodou JSP je její dobrá a rychlá manipulace se servlety. Tyto servlety umožňují například přečíst data odeslaná uživatelem (z HTML, XML formuláře apod.), zformátovat výstup do určitého formátu, vytvořit výsledky atd. Velkou výhodou je, v podstatě, interaktivita s uživatelem.

7.1.1. Práce s technologií JSP

Tato technologie umožňuje široké spektru možností pro vytváření kombinace statických a dynamických stránek. Takto vzniká možnost vytváření různých šablon pro jednotlivé stránky, založené na kombinaci několika způsobů a technologií.

Hlavním důvodem proč jsem se rozhodl pracovat s touto technologií oproti ostatním jako například PHP (Hypertext Preprocesor) byl to, že Java podporuje velké množství knihoven pro práci se zdroji a manipulaci s daty. Příkladem je použitá ode mne knihovna Jena, která manipuluje s ontologickými modely a následně i s výstupem ve formátu RDF. Další výhodou, kterou JSP má oproti ostatním technologiím je podpora většiny platforem.

Během vývoje nedošlo k plnému využití všech možností JSP a technologií, které JSP podporuje a používá. Vyskytuje se zde však možnost, aby mohly být

dále snadno přidané a integrované. Důvodem byl nedostatek času při seznámení a práci s platformou JSP. Jedny z klíčových technologií jsou:

- **JSTL** - JavaServer Pages Standard Tag Library (JSTL) je sbírka užitečných tagů JSP, která vystihuje základní funkce společné pro mnoho JSP aplikací. JSTL má podporu pro běžné, strukturální úkoly, jako je iterace a podmiňovací způsob, tagy pro manipulaci s XML dokumenty, SQL a další značky. [12]
- **JFS** - JavaServer Faces (JSF) je Java specifikace pro budování component-based uživatelské rozhraní pro webové aplikace. Byla formována jako standard prostřednictvím Java Community Process a je součástí platformy Java Enterprise Edition. JSF 2 používá Facelets jako výchozí templating systém. JSF verze 1.x používá JavaServer Pages (JSP) jako výchozí templating systém. [12]
- **Spring** - Spring Framework je open source aplikační framework (kontejner) pro platformu Java. Základním důvodem jeho vzniku a taky jeho největší prioritou jsou podnikové (enterprise) aplikace. Hlavní pozitiva je jeho vysoká abstrakce, což usnadňuje používání J2EE. Řídí se dle návrhového vzoru Inversion of Control, a proto název IoC kontejner. Ten napomáhá aplikaci tím, že přenáší odpovědnost za chování objektů na framework nikoli na samotné aplikace. Objekty vytvářené kontejnerům se nazývají `Beans`. [12]

7.1.2. Syntaxe

Syntaxe platformy JSP se podobá Javě. Při práce s technologií JSP je cílem psát co nejméně javovského kódu v prezentačních stránkách (JSP, HTML apod.) a ve zbytku prezentační části. Dokud při psaní zdrojových kódů je pravý opak, cílem je psaní co nejméně prezentačního kódu do tříd a co nejvíce javovského kódu. Toto lze dosáhnout pomocí snadné syntaxe JSP, která se podobá PHP a to ták, že během vytváření webových stránek lze kombinovat statický a dynamický kód různými způsoby. Základní a nejpoužívanější v mojí prácí jsou:

- Skriptlety – ve tvaru <% Java kód %>, které se vkládají do metody servletu _jspService (pomocí volání služby service).
- Výrazy – ve tvaru <%= Java výraz %>, které jsou vyhodnocený a vloženy do výstupu servletu.
- Direktivy – ve tvaru <%! Pole/Deklarace metody %>, které jsou vložené do těla třídy servletu.
- Komentáře – ve tvaru <%-- komentář --%>

V podstatě se jedná o smíchání Javy přímo s JSP/HTML kódem, což umožňuje tvorbu tzv. dynamických webových stránek.

7.1.3. Fungování servetů

Můžeme tvrdit, že servlety jsou obdoba CGI[17]. Z pohledu architektury MVC[18] pro vývoj dynamických webových aplikací lze porovnat servlety se střední vrstvou této architektury – `Controller`. Fungují spíše jako řídící prvek, který řeší, jak bude manipulovat s daty a zpracovávat vyplněné údaje.

Při vytváření servletu vzniká jedna jeho unikátní instance v paměti. Poté co bude servlet zavolán, buď pomocí formuláře, nebo rovnou při zadání do URL adrese, se vykoná metoda init. Toto volání probíhá pouze jednou, na začátku při prvním zavolání servletu. Následně lze spatřit nepatrné čekání na načtení stránky. Dále již funguje poměrně rychle a zpracovává požadavky klienta výkonně.

[17] CGI (Common Gateway Interface) – protokol pro komunikaci mezi serverem a programem, který odpovídá za dynamickou část webové prezentaci.
[18] MVC (Model View Controller) – návrhový vzor, který rozděluje datový model, prezentační část a řídídi logiku do 3 nezávislých vrstev.

Základní struktura servletu vypadá následovně:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class TestServlet extends HttpServlet {

  public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

    // Sem lze použít "request" k čtění příchozí HTTP hlavičky
    // nebo data z formuláře, poté co uživatel odeslal formulář přes submit

    PrintWriter out = response.getWriter();
    // pomocí out lze posílat výpisy rovnou prohlížečí
  }
public void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

    // Fungování na stejném principu jako doGet s několika rozdílu.
  }

}
```

Příklad zdrojového kódu 1: Ukázka základní struktury používaného servletu.

Obvykle programátoři mají obtíž při výběru mezi metodami `doGet()` a `doPost()`.

- doGet() se používá, když je potřeba zpracovat malé množství dat, převážně necitlivých.
- doPost() se používá při poměrně velkém množství citlivých údajů, které musí být odeslány. Příkladem může být odesílání dat po vyplnění formuláře nebo zasláním přihlašovacího jména či hesla. V případě manipulace s mými formuláři jde o logiku pro zpracování citlivých pacientských údajů. To je hlavní důvod proč použít metodu doPost() při manipulaci s nimi.

Jako hlavní výhody, které lze vytknout při práci se Servlety oproti CGI můžeme říci, že mají následující vlastnosti:

- Větší výkonnost – velké množství funkcí, které CGI neumí, jako například komunikace se serverem ve formě požadavků při předávání různých informací. Další hlavní výhodou servletu oprotí CGI, je že pro každý požadavek CGI vytváří nový proces, což zatěžuje

paměť a snižuje robustnost celého projektu. Dokud při zpracování vícenásobných požadavků si servlety umí poradit tak, že pro N požadavků vytváří N vláken stejného procesu. [12]

- Velká podpora – Servlety jsou psané v jazyce Java a tím mají tu výhodu, že jsou na většině open-source serverech podporovány. Dále servlety podporují velké množství knihoven, které se starají o profesionální zpracování dat. Týká se to především objektu, které lze použít při vývoje aplikace vesměs při komunikaci mezi serverem a klientem (nastavení sessions, cookies, formulářových dat atd.). [12]

7.1.4. Formuláře a servlety

Formuláře jsou jedním z nejjednodušších způsobů jak získávat data zadávaná uživatelem přes internet. Fungují jako zprostředkovatel mezi servletem a uživatelem, zadávající data do internetového formuláře. Pomocí stále vyvíjející se technologie HTML, lze rychle identifikovat jednotlivé atributy zadávaných hodnot. Každá ze zadávaných hodnot má svoje atributy, jako name, value, atd., kterými lze snadno procházet na straně serveru, právě pomocí servletu.

```
<form action="OntologyServlet" method="POST">
   <input type="text" name="parametrName" value="hodnota">
   <input type="submit" value="odeslat">
</form>
```

Příklad zdrojového kódu 2: Příklad zpracovávání dat, zadávaných lékařem.

Na příkladu zdrojového kódu č. 2 je vidět základní strukturu formuláře, zpracovávající zadávaná data (vesměs lékařské odborné výrazy, popisy, texty pacientských dat atd.). Hlavní zpracovávaná dvojice atributu jsou parametry name a value. Servlet následně zpracovává tyto dvojice jako řetězce ve tvaru:

```
parameterName = hodnota
```

Jak je vidět `parameterName` je konkrétní jméno atributu `name` tagu input, který určuje popisek zadávaných dat. Atribut `value` určuje konkrétní

hodnota daného popisku. Existuje možnost, že `parameterName` může mít více stejných hodnot. Vyplňovaná data (hodnoty) jsou zpracovávaná ve formátu UTF-8, podporující češtinu.

Jak je vidět při definici formuláře je dále definován atribut `action`. Ten je nastavený na URL adresu servletu, který potřebujeme zavolat a následně spustit. Tento servlet je poté spuštěn se zadanými parametry a hodnotami ve formuláři. Tyto dvojice zpracovávám na straně serveru ve vlastním algoritmu. Na jejímž základě pak plynou požadované výstupy ve formátu RDF/XML.

Jednotlivé názvy a jejich hodnoty získávám, při použití výkonné metody `getParameterNames()`, která vrací datový typ `Enumeration`.

```
Enumeration<String> parameters = (Enumeration<String>) request.getParameterNames();

    while(parameters.hasMoreElements()){

          String parameterName = (String) parameters.nextElement();
          String parameterValue = request.getParameter (parameterName);
          multipleObjectValues = request.getParameterValues(parameterName);

    }
```

Příklad zdrojového kódu 3: Ukázka procházení zadávaných dat lékařem.

K získávání těchto hodnot používám metodu `getParameter()`, definována rozhraním `HttpServletRequest`, která jako argument požaduje název parametru (z ukázky zdrojového kódu č. 2, je to `parameterName`). Tato metoda vrací datový typ `String`, obsahující konkrétní hodnotu (`parameterValue`), patřící danému popisku, dekódovanou UTF-8.

7.2. Jena API

Pro práce s daty ve formátu RDF/XML nebo jiné datové formáty ontologických modelů lze v Javě použít knihovnu Jena[19]. Je to velmi výkonný Framework, který napomáhá vývojářům při práci se soubory ve strukturované podobě. Používá se zejména pro tvorbu sémantických webových aplikací. To je též hlavní důvod, proč by se měl použít. Aplikační Framework Jena API jako nástroj založený na technologii Java, poskytující aplikační rozhraní pro manipulaci s informacemi v jazyce RDF. Tato technologie, která podporuje syntaxi RDF/XML, Turtle, N-Triples atd. byla vyvinuta Brianem McBridem.

Pomocí tříd a vnořených funkcí aplikačního Frameworku Jena lze vytvářet, číst, zapisovat a procházet RDF dokumenty.

Jena zahrnuje:

- API pro čtení, zpracovávání a psaní RDF dat v XML, N-trojic a Turtle formátů
- API pro manipulaci s OWL a RDFS znalostmi
- Používá mnoho metod pro uvažování, ze kterých se snaží odvodit určité odpovědi na bázi znalostního modelu
- Umožňuje ukládání velkého množství RDF trojic na disku, pro snadnější manipulace [13]

7.2.1. Čtení z modelu

Za pomoci Frameworku Jena lze snadno nadefinovat `OntologyModel`, na základě kterého můžeme pracovat dál s načtením modelu do paměti a manipulovat s jeho elementy. Všechny výše popsané metody a funkce umožňují snadnou práci s jednotlivými prvky daného modelu. Ve svojí práce jsem použil metody pro vytváření ontologického modelu. Pomocí tohoto modelu lze interpretovat snadněji jednotlivé prvky datového modelu DASTA, vytvářen v prostředí Protégé.

[19] Framework Jena API: http://jena.apache.org/

Příkladem definování ontologického modelu může vypadat následovně:

```
OntModel m = ModelFactory.createOntologyModel();
```

Tím je vytvořená instance modelu m, pomocí které lze volat metody knihovny Jena a manipulovat s jednotlivými prvky ontologického modelu. Například získávat názvy jednotlivých vlastností v češtině či angličtině, získávat titulní nadpisy jednotlivých bloků (v Protégé také známo jako třídy), určujících o jaký formulář se jedná (Vstupní formuláře – Pacient, Diagnóza apod.), dále získávat anotace k jednotlivým vlastnostem, popisujících jejich typ, identifikátor či další potřebné popisky, nutné k správnému generování formulářů. [13]

Nejpoužívanější metody pro získání jednotlivých elementů modelů, potřebné pro generování formulářů byly:

Používané metody pro čtení	Popis
`model.getOntProperty(NS + property)`	Získání datových či objektových vlastností dané třídy
`model.getOntClass(NS + className)`	Získání konkrétní třídy
`property.getLabel("en");` `property.getLabel("cs");`	Získání štítku nebo nadpisu konkrétné vlastnosti v češtině a angličtině
`model.getAnnotationProperty( arg0 );`	Získání anotací modelu. Jako argument je potřeba URI adresa vyhledávaného typu anotace
`property.getPropertyValue( arg0 )`	Získání hodnoty RDF vlastnosti daného zdoje.
`property.getComment("id")`	Získání komentáře dané vlastnosti. Může být v angličtině, češtině či speciálně zvolený
`property.getURI()`	Získání URI dané vlastnosti

Tabulka 4: Nejpoužívanější metody knihovny Jena

Z tabulky č. 4 lze vyvodit to, že pro generování formulářů, stačí využít několika základních metod z celé škály nabízených funkcí knihovny Jena. Jako velmi podstatnou pro generování jednotlivých typů políček v HTML bychom mohli říci, že je metoda `getAnnotationProperty( arg0 )`. Jako argument této metody je potřeba zadat URI konkrétného typu anotace, kterou vyhledáváme. Při doplňování ontologie jsem zvolil následující název:

```
http://mre.kiv.zcu.cz/inputType
```

Takto vyrobené ode mne URI funguje jako predikát neboli zdroj, který určuje, jaký typ vstupního pole bude mít daná vlastnost ve formuláři. Například vlastnosti - jméno a příjmení pacienta jsou typu text, z čehož vyplývá, že `input` tagy v HTML budou typu `text`.

7.2.2. Zápis do modelu

Další významnou funkcí knihovny Jena je zapisování. Zapisování zpracovaných dat do požadovaného tvaru lze provádět jednoduše pomocí vytváření nových modelů. V předchozí části jsem popisoval, jak lze číst hotový datový, ontologický, standardizovaný model. Teď nastíním, jak lze vyplněná data z formulářů jednoduše ukládat do formátu RDF/XML.

Jak jsem se již zmiňoval v předchozích kapitolách návrhu a analýzy moje řešení pro ukládání dat spočívá, ve vytváření zvláštních modelů pro každý formulář. Tyto modely se následně slučují v závislosti na požadované akci lékaře. Tyto akce jsou spojené s přidáním nového formuláře nebo jeho editace, či vymazání. Vytváření vypadá následovně:

```
Model model = ModelFactory.createDefaultModel();
```

Takto vytvořený datový model je prázdný. Pro potřebu jeho naplnění pacientskými daty jsem používal další významné a asociativní metody knihovny Jena. [13]

Pro naplnění modelu s daty a metadaty lze použít několik základních funkcí. Na začátku je vždy vhodné mít jeden hlavní zdroj, ke kterému jsou následně přidávané jednotlivé hodnoty a popisky. Tyto popisky mohou být jak čistě datové (tzv. literály), tak i objektové, což znamená, že odkazují na další zdroj. Místo datového řetězce mají odkaz na instance třídy. Ve svojí práci vytvářím pro každý formulář zvláštní zdroj a to tak, že v předchozím formuláři, který rodičovský tomuto podformuláři zanechávám instance jeho třídy. Tato instance se skládá z názvu dané třídy a unikátního hash. Takto ručím za to, že nedojde k redundanci a v případě vícenásobných formulářů stejného typu budou instance vždy unikátní.

Za využití základních metod knihovny Jena lze přidávat zdroje, vlastnosti a hodnoty v serializované podobě, tak aby popisovaly smysluplně zadávaná data z klávesnice. Pro vytvoření nového pacienta je potřeba nadefinovat jako hlavní zdroj jeho unikátní instance:

```
Resource resource = model.createResource( URI_Patient );
resource.addProperty(RDF.type, model.createResource(NS+"Patient"));
```

Takto vytvořený pacient s předem zpracovaným URI, které je založeno na základě unikátních hodnot tohoto konkrétního anonymního pacienta jako je jeho identifikační číslo či datum narození apod. Následně pomocí výkonných metod frameworku Jena lze vytářet zdroje: `model.createResource()`. Takto je vytvořen hlavní zdroj, který bude odpovídat tomu konkrétnímu pacientovi. Dále k tomu zdroji lze přidávat popisky a hodnoty, které ve svém algoritmu připravuji předem. Přidávání vlastností je analogické, pomocí metody `resource.addProperty(arg0, arg1)`. Tato metoda požaduje dva argumenty, které jsou konkrétním typem vlastnosti (arg0) a její hodnota (arg1). Všechny další vytvářené formuláře jsou založené na stejném principu, jenž se generují na základě vyskakovacích oken. Pro N vygenerovaných formulářů se vytvoří N modelů. Tyto modely se v závislosti na algoritmu zpracování dat

sloučí do jednoho konečného. Takto lze přidávat nekonečný počet lékařských zpráv, anamnéz a dalších formulářů k jednomu jedinému pacientovi.

V předchozím textu jsem řešil přidávání vlastností. Nyní nastíním, jak jsem vytvářel instance jiné třídy. Na následující ukázce použiji stejné metody jako v předchozím odstavci - `createResource()` a `addProperty`, ale v mnohem komplexnějším tvaru.

```
String URI_Patient = "http://nemocniceXYZ.org/Pacients/FN13P160313IO558";
String NS = "http://mre.kiv.zcu.cz/ontologie/2012/01/dasta.owl#";

  // Vytvoření prázného modelu
  Model model = ModelFactory.createDefaultModel();
  model.setNsPrefix("ds", NS);
  // Vytvoření hlavního zdroje - pacient
  Resource resource = model.createResource( URI_Patient );
    resource.addProperty(RDF.type, model.createResource(NS+"Patient"));
  // Přidávání vlastností
    Property p1 = model.createProperty(NS, "prijmeni");
          resource.addProperty(p1, "Kamburov");
    Property p2 = model.createProperty(NS, "jmeno");
          resource.addProperty(p2, "Mario");
    Property p3 = model.createProperty(NS, "datum_narozeni");
          resource.addProperty(p3, "6.1.1991");
    Property p4 = model.createProperty(NS, "hasAddress");
          resource.addProperty(p4, model.createResource(rURI_ID1);
    Property p5 = model.createProperty(NS, "hasAddress");
          resource.addProperty(p5, model.createResource(rURI_ID2);
  // Nový zdroj - trvalá adresa
  Resource patientAddress1 = model.createResource(rURI_ID1);
    Property p6 = model.createProperty(NS, "Mesto_obec");
          patientAddress1.addProperty(p6, "Praha");
    Property p7 = model.createProperty(NS, "PSČ");
          patientAddress1.addProperty(p7, "16100");
   // Nový zdroj - dočasná adresa
  Resource patientAddress2 = model.createResource(rURI_ID2);
    Property p8 = model.createProperty(NS, "Mesto_obec");
          patientAddress2.addProperty(p8, "Plzen");
    Property p9 = model.createProperty(NS, "PSČ");
          patientAddress2.addProperty(p9, "32300");
```

Příklad zdrojového kódu 4: Generování výstupu formuláře, pomocí knihovny Jena

Na příkladu zdrojových kódů č. 4 je na první pohled vidět snadná manipulace s jednotlivými elementy modelu. Ve svojí aplikaci ji ovšem generuji dynamicky, ale pro ukázku jsem uvedl příklad statického vytváření jednoho z mnoha možných modelů pro ukládání dat do serializované podoby.

To co stojí za pozornost je vytváření nových zdrojů. Příkladem jsou vlastnosti p4 a p5, které vytváří instance třídy `Adresa`. Tyto instance odkazuji na další zdroje. Tyto instance jsou tvořené unikátními hash funkcemi v pozadí. Následně vytvářím dva další zdroje, které mají stejné unikátní URI adresy jako vytvořené instance daných vlastnosti `hasAddress` předtím. To zaručuje funkčnost ze dvou pohledů. Z prvního hlediska se jedná o dvě přidané adresy objektů, odkazující na jiný zdroj, a z druhého hlediska se jedná o samostatné subjekty čili zdroje. Takto jednoduše lze zaručit fungování principu ukládání velkého množství dat ve standardu RDF, pomocí datových ostrovů. To lze taktéž spatřit ve výstupu tohoto kódu.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ds="http://mre.kiv.zcu.cz/ontologie/2012/01/dasta.owl#" >

<rdf:Description rdf:about="http://nemocniceXYZ.org/Pacients/FN13P160313IO558">
    <ds:hasAddress rdf:resource=
"http://mre.kiv.zcu.cz/id/PermanentAddress/86f30b8fcbd9e026331ee68cc634b993"/>
    <ds:hasAddress rdf:resource=
"http://mre.kiv.zcu.cz/id/TemporaryAddress/907e52bc4f567d34227a47185e74c4e9"/>
    <ds:datum_narozeni>6.1.1991</ds:datum_narozeni>
    <ds:jmeno>Mario</ds:jmeno>
    <ds:prijmeni>Kamburov</ds:prijmeni>
    <rdf:type rdf:resource=
"http://mre.kiv.zcu.cz/ontologie/2012/01/dasta.owl#Patient"/>
  </rdf:Description>

  <rdf:Description rdf:about=
"http://mre.kiv.zcu.cz/id/PermanentAddress/86f30b8fcbd9e026331ee68cc634b993">
    <ds:PSČ>32300</ds:PSČ>
    <ds:Mesto_obec>Plzen</ds:Mesto_obec>
  </rdf:Description>

  <rdf:Description rdf:about=
"http://mre.kiv.zcu.cz/id/TemporaryAddress/907e52bc4f567d34227a47185e74c4e9">
    <ds:PSČ>16100</ds:PSČ>
    <ds:Mesto_obec>Praha</ds:Mesto_obec>
  </rdf:Description>
</rdf:RDF>
```

Příklad souboru dat 2: Ukázka výstupu příkladu zdrojového kódu č. 4.

Na ukázce souboru dat č. 2 je jasně vidět přehledná serializace dat konkrétního pacienta. Pacient má dvě adresy, na které je možné odkazovat pomocí instancí. Dále jsou do dvou datových ostrovů rozdělené další údaje týkající se těch zdrojů (obě přidané adresy).

Takto lze zapisovat do výstupného modelu datové či objektové vlastnosti. Ve výstupu dostáváme přehledný, komplexní a standardizovaný popis pacientů, na základě vstupních formulářů.

7.3. Knihovna Google Guava

Knihovna Guava je open-source projekt, vyvíjen společnosti Google. Tato malá knihovna poskytuje řadu možností pro práce s kolekcemi dat. To byl hlavní důvod, proč jsem si ji vybral do svojí bakalářské práce. Další užitečné funkce, které tato javovská knihovna podporuje, jsou: Kolekce dat, kešování, pokročilé zpracovávání řetězců atd.

Poprvé jsem narazil na tuto knihovnu, když jsem potřeboval zpracovávat vícenásobné hodnoty se stejným klíčem. Standardní přístupný kontejner v Javě, při práce s kolekcemi, je například často používaný kontejner `HashMap`, který implementuje rozhraní `Map`. Tyto kolekce, nabízené javou, mohou být v různých variantách, ale bohužel ani jedna nepodporuje možnost přidávání vícenásobných hodnot se stejným klíčem. Právě to byl hlavní důvod využít knihovnu guava, která podporuje práci s kolekcemi a navíc nabízí kontejner `MultiMap`. Tento konkrétní kontejner knihovny guava nám umožňuje využít všechny funkce dosud známých kolekcí v Javě a dále navíc přidává užitečné doplňující funkce jako právě potřebných vícenásobných hodnot, patřících pod stejný klíče. [14] Definování vypadá následovně:

```
Multimap<String, String> instances = HashMultimap.create();
```

Pro práce s touto mapou, procházení jejích hodnot či seznam klíčů nebo kombinaci všech možností dohromady lze použít výkonné a poměrné rychlé iterování přes ni. To vypadá následovně:

```
for (Map.Entry<String, String> entry : instances.entries()) {
      System.out.println(" Key: " + entry.getKey() +
                         " Value: "+ entry.getValue()
                         );
            }
```

Jak je vidět, iterování přes jednotlivé hodnoty a klíče je snadné a elegantní. Takto jsem pracoval s velkým množství vyplněných dat pacienta.

7.4. Další technologie

V této kapitole si řekneme něco o doplňujících technologií, které jsem použil ve svojí práci. Velkou část z nich jsem použil na úrovní prezentační vrstvy.

7.4.1. Javascript

JavaScript je multiplatformní objektově-orientovaný jazyk. Jehož zakladatelem je Brendan Eich. Vznikl přibližně před 20 lety. Tehdy Eich pracoval ve společnosti Netscape, která následně společně s ním vyvíjela tento jazyk. JavaScript lze chápat jako interpretovaný programovací jazyk pro internetové stránky nebo také jako doplněk do HTML stránek. Právě pomocí něj lze zaručit určitou dynamičnost stránek. Skript je interpretován na straně klienta. Využívání JavaScriptu je v dnešní době velmi častá praxe, právě proto jsem si zvolil použít jeho možností ve svoji práce. Syntaxe se nepodobá nějak obzvlášť syntaxi Javy, je to spíše kombinace C/C++ a Java. Rozdílem od technologií popisovaných v předchozích kapitolách je to, že se spouští na straně klienta až poté co naběhne celá stránka. Dokud JSP, který má obdobnou syntaxi jako PHP se spouští na straně serveru, ještě předtím, než naběhla stránka s prezentačním obsahem. [15]

Ve svojí práci jsem používal JavaScript převážně ve funkcích, které umožňují přesměrovat uživatele (lékaře) do dalších podformulářů ve formě vyskakovacích oken (známo též jako pop-up). Je to spojené s popisováním pacienta, na základě velkého množství generovaných formulářů. Taková funkce by mohla vypadat následovně:

```
function CreatePopUp(OntoClass, predicate){
      var uniq = '' + (new Date()).getTime();
      var url = "OntologyServlet?createPopUp="+OntoClass+"&id="+uniq;

window.open(url,predicate);
      var element = document.createElement('input');
          element.type = 'hidden';
          element.value = OntoClass+"#"+uniq;
          element.name = predicate;
          element.size = 30;
     document.getElementById("container").appendChild(element);
     }
```

Příklad zdrojového kódu 5: Ukázka funkce v JavaScriptu

Na ukázce zdrojového kódu č. 5 lze spatřit přesměrování do nového okna, pomocí funkce window.open v JavaScriptu. Dále při vytvoření nového podformuláře s předanými parametry ve formě předem připraveného URL, ukládám do skryté proměnné s identifikátorem container, hodnoty vytvořeného formuláře. Ty pak budou následně po odeslání formuláře na straně serveru zpracované jako instance na další zdroj (podformulář) ve výstupném souboru pacientských dat.

7.4.2. jQuery

jQuery je výkonná, moderní a široce podporovaná knihovna JavaScriptu. Vznikla na začátku roku 2006. Její vydavatelem je John Resigem. Je to freeware a open-source knihovna, která poskytuje řadu možností pro práci s jednotlivými DOM[20] elementy HTML. Její výhodou je interakce s JavaScriptem a HTML/CSS elementy stránky. Je snadno integrovatelná do jakéhokoliv projektu, s tím že je potřeba jí importovat ze souboru nebo přes přímý odkaz z internetu. Hlavní filozofie jQuery spočívá v tom, že odděluje prezentační část od struktury, charakteristiky a atributy HTML. Technologie je dále známa jako nevtíravý JavaScript. [16] Příkladem takového použití nevtíravého JavaScriptu si můžeme ukázat na následující ukázce:

```
$(function() {
      $('button').click(function(){
      var $this = $(this),

      var classAtr = $(this).attr('class');
      var createClass = $("#"+this.id).val();

      if(classAtr == "CreateAddButton"){
            CreatePopUp(createClass, this.id);
      }
});
});
```

Příklad zdrojového kódu 6: Ukázka použité funkci v jQuery

[20] DOM (Document Object Model) – objektově orientovaná reprezentace HTML dokumentu

Z příkladu zdrojového kódu č. 6 lze vyčíst, že za použití nevtíravého JavaScriptu je umožněna interakce mezi jednotlivými elementy HTML a JavaScriptem. Po kliknutí na tlačítka jQuery získává hodnoty jeho atributu či dalších potřebných elementu HTML, se kterými dále manipuluje (mění manipulátor událost či volá jiné JavaScriptové funkce).

7.4.3. HTML 5

HTML5 je v současnosti nejnovější verze jazyka HTML. Je stále ve fázi návrhu, což je prozatím jeho mínus, ale je v podstatné části je u moderních prohlížečů, podporována. Je plánováno schválení a oficiální nasazení většiny prohlížeče kolem roku 2014. [17]

Hlavní výhody a novinky této verze, se kterou jsem pracoval, ve svojí bakalářské práci jsou:

- Zkrácené zapisování značek
- Nové elementy a atributy značek
- Nové funkce založené na HTML DOM a JavaScriptu
- Redukce potřeby nadbytečných importů knihoven
- Kvalitnější zpracování chyb

7.4.4. CSS 3

CSS 3 (Cascading Style Sheets 3) je poslední nejnovější verze CSS. Tato technologie byla vyvíjená organizací W3C. Je stejně jako HTML5 stále ve fázi návrhu, ale oficiální zahájení se plánuje koncem roku 2014. CSS definuje, jak bude vypadat vzhled webových stránek. Na základě pár použitých funkcí CSS jsem ve svojí práci vytvořil přehledný a transparentní návrh formulářů, kde je jasně definována úroveň nadpisů, popisků (labels) či uspořádání vstupních polí. [18]

8. Testování

Testování funkčnosti formulářů má za úkol zjistit, odhalit a opravit problémová místa navrhovaných formulářů do lékařského systému. Testování lze rozdělit do dvou základních skupin. První část odhaluje chyby a řeší funkčnost generování jednotlivých formulářů, v závislosti na doplněné ontologii. Další část se zaměřuje na testování naprogramovaného rozhraní reálním lékařem a zadávání odborných dat do jednotlivých formulářů, popisujících pacienta při příjmu do nemocnice.

8.1. Testování funkčnosti generování

Pro správné generování formulářů bylo potřeba předpřipravit a doplnit stávající ontologii, dodaná katedrou, o chybějící vlastnosti či anotace. Z důvodu průběžného doplňování jednotlivých bloků a jejich patřících vlastností, by se mohlo říci, že správnost generování formulářů funguje na 85%. Zbývajících 15% beru jako rezervu z důvodu dalšího doplnění ontologie o potřebné třídy a vlastnosti. Funkcionalita generovaných, konkrétních vlastností, jednotlivých formulářů je přibližné 70 %. Důvodem je omezené množství přidaných prvků jednotlivých bloků standardu DASTA. Celkově mohu tvrdit, že po doplnění chybějících vlastnosti budou generované formuláře fungovat bez zásadních problémů. Tím lze ověřit funkčnost navrhovaných algoritmů, zakládající se na obecném principu.

8.2. Testování použitelnosti formulářů nad reálnými daty.

Při testování použitelnosti formulářů bylo mým úkolem zjistit, do jaké míry jsou navrhované stránky přehledné a transparentní pro lékaře, pracující s nimi. Bylo zjištěno, že vzhled je intuitivní na ovládání a uspořádání vstupních polí je vyhovující. Ovšem jako jediná nevýhoda navrhovaných vstupních formulářů, vytknuta reálním lékařem byla ta, že chybí širší spektru možností k přidání jednotlivých typů formulářů. Příkladem bylo formulář Anamnéza, který je pouze jeden a při příjmu pacienta do nemocnice lékař vyplňuje minimálně tři druhy anamnéz, týkající se momentálního stavu pacienta. Toto jsou

ovšem detaily, které se týkají celkového používání formulářů a lze je snadno přidat či dodělat.

8.3. Testování kompatibility

Další testování funkčnosti formulářů se týkalo kompatibility jednotlivých prohlížečů. K účelu testování nad různými prohlížeči lze použít řada metod a hotových platforem na internetu. Já jsem to testoval pro jistotu ručně na posledních verzí jednotlivých prohlížečů. Vyhodnocení výsledku však dospělo k tomu, že z důvodu použití nejnovějších technologií HTML 5 a CSS 3, jsou navrhované formuláře plně funkční pouze na prohlížečích Google Chrome a Opera. Výjimky ovšem dělají pouze prohlížeče Mozilla Firefox 20.0.1 a Internet Explorer 10. Jejích nevyhovující chování vůči navrhovaným formulářům je pouze 20% chybějících funkcí, což je při konečném vyhodnocení, poměrně vyhovující výsledek. Co se týče vzhledu a layoutu stránek, formuláře jsou na všech prohlížečích ve stejném zjevu.

Vzhledem k rychlému vývoji technologií lze tvrdit, že podpora HTML5 a kompatibilita prohlížečů vůči navrhovaných formulářů bude za nedlouho zcela funkční. V prohlížeči Internet Explorer 11 je plánovaná plná podpora jazyku HTML5 a v přicházejících verzí Mozilla Firefox je stále zlepšována jeho funkčnost.

9. Závěr

Na začátku svojí práce jsem se zabýval širším pohledem na problémy spojené s informačními systémy ve zdravotnictví, na základě čehož jsem dospěl k tomu, že je zapotřebí zavést moderní, výkonná a levná řešení, napomáhající v oboru zdravotnictví, která jsou snadno integrovatelná s ostatními aplikacemi podobného charakteru.

Během návrhu jsem si zvolil několik základních technologií a knihoven, pomocí kterých jsem manipuloval s ontologií či zapisoval kolekce dat do požadovaného formátu RDF/XML. V konečném výsledku lze tvrdit, že bez těchto výkonných knihoven by práce byla mnohem obtížnější, s jejich pomocí jsem dosáhnul požadovaný výsledek výstupu formulářů či jejich obecné generování. V dalších části návrhu a analýze jsem nastiňoval konkrétní možná řešení určitých problémů či teoretické seznámení s problematikou.

V kapitole implementace vysvětluji konkrétní způsoby, navržené ode mne, řešení, které jsem použil ve svojí práci. Tato část je zaměřená zcela prakticky na jednotlivé funkce, případy užití či podstaty navrhovaného algoritmu.

Interakce s uživatelem je zaručena, pomocí využití dodatečných technologií JavaScriptu a jQuery, které jsem popisoval v kapitole Použité technologie. Na praktických ukázkách vysvětluji její nejpodstatnější funkci, týkající se generování vícenásobných formulářů.

Na konci svojí práce jsem se věnoval testování aplikace nad reálnými daty, z čehož jsem dospěl k tomu, že momentální základní formuláře, které navrhovaný algoritmus generuje, fungují správně, ale z důvodu neustálého vyvoje a práce nad ontologií katedrou, je zbývající část formulářů ve fázi návrhu. Ovšem po jejím doplnění, budu moci tvrdit, že budou fungovat správně, tak jako základní část z nich, popisujících pacienta.

10. Citovaná literatura

[1] Český Statistický Úřad. (2012, Aug.) http://www.czso.cz/. [Online]. http://www.czso.cz/csu/redakce.nsf/i/informacni_technologie_ve_zdravotnictvi

[2] Lubomír Karpecki. (2010, July) http://businessworld.cz/. [Online]. http://businessworld.cz/it-strategie/ict-ve-velkych-nemocnicich-6615

[3] Martin Noska. (2010, Dec.) http://computerworld.cz/. [Online]. http://computerworld.cz/ostatni/jak-mohou-ict-zefektivnit-fungovani-zdravotnictvi-8203

[4] prof. MUDr. Antonín Jabor Ing. Miroslav Zámečník RNDr. Luděk Straka. [Online]. http://ciselniky.dasta.stapro.cz/

[5] Boris Eninger. (2011, May) Přenos dat z registru SITS ve formátu HL7. Západočeská univerzita v Plzní, Diplomová práce.

[6] Datový standard Ministerstva Zdravotnictví ČR. http://ciselniky.dasta.mzcr.cz. [Online]. http://ciselniky.dasta.mzcr.cz/CD/ds/

[7] Lukáš Adámek. (2008, May) Analýza sémantických technologií pro implementaci systémů v oblasti hodnocení ekologického stavu povrchových vod. Brno, MASARYKOVA UNIVERZITA, Fakulta Informatiky, Diplomová práce.

[8] Daniel Kouba. (2008, May) Transformace mezi Topic Maps a RDF/OWL. Praha, Vysoká Škola Ekonomická, Bakalářská práce.

[9] Tim Berners-Lee. (2005, Jan.) Uniform Resource Identifier (URI): Generic Syntax. [Online]. http://tools.ietf.org/html/rfc3986

[10] The Protégé Ontology Editor. (2013, Mar.) http://protege.stanford.edu/. [Online]. http://protege.stanford.edu/

[11] FI MU Josef Vochozka. (2011, Nov.) http://www.ics.muni.cz/. [Online]. http://www.ics.muni.cz/bulletin/articles/214.html

[12] David Adam. (2012, May) Portál pro DDM. Západočeská univerzita v Plzní, Diplomová práce.

[13] Apache Jena. (2013) the Apache Jena project. [Online]. http://jena.apache.org/

[14] Tom Jefferys. (2011, Sep.) http://tomjefferys.blogspot.cz/. [Online]. http://tomjefferys.blogspot.cz/2011/09/multimaps-google-guava.html

[15] Stephen Chapman. (2010, Apr.) http://JavaScript.about.com/. [Online]. http://JavaScript.about.com/od/reference/p/JavaScript.htm

[16] John Resigem. (2013, Apr.) The jQuery Project. [Online]. http://learn.jquery.com/

[17] HTML5 W3C. (2013, Apr.) http://www.w3schools.com/. [Online]. http://www.w3schools.com/html/html5_intro.asp

[18] CSS W3C. (2013, Apr.) http://www.w3schools.com/. [Online]. http://www.w3schools.com/css/css_intro.asp

11. Použité zkratky

Zkratka	Popis
DASTA, DS	Datový standard MZ ČR
JSP	Java Server Pages
RDF	Resource Description Framework
ICT, IT	Information and Communication Technologie
HL7	Health Level 7 International
DICOM	Digital Imaging and Communications in Medicine
SITS	Safe Implementation of Treatments in Stroke
W3C	World Wide Web Consortium
URI	Uniform Resource Identifier
HTML	HyperText Markup Language
XML	Extensible Markup Language
ID	Identifier
URL	Uniform Resource Locator
XSD	XML Schema Definiton
DAWG	RDF Data Access Working group
SPARQL	SPARQL Protocol and RDF Query Language
RDFS	RDF Schema
OWL	Web Ontology Language
JSF	JavaServer Faces
JSTL	JSP Standard Tag Library
MVC	Model View Controller
PHP	HyperText Preprocessor
UTF-8	UCS Transformation Format – 8bit.

12. Seznam obrázků, tabulek a příkladů

12.1. Seznam obrázků

12.2. Seznam tabulek

12.3. Seznam příkladů

13. Seznam příloh

Příloha A: Obsah DVD

A. Obsah DVD

`app/`	potřebné aplikace ke spuštění
`J2EE/`	prostředí Eclipse Juno pro J2EE
`Java/`	poslední verze JDK
`lib/`	použité knihovny
`Tomcat/`	obsahuje tomcat 7.0
`dist/`	nasaditelný balíček ve formátu WAR
`doc/`	elektronická verze textu BP
`ontology/`	navržené ontologie
`outputs/`	výstupy formulářů (reální data)
`prj/`	kompletní projekt pro Eclipse J2EE