

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

## **Diplomová práce**

# **Datový model EEG/ERP portálu v prostředcích sémantického webu**

Plzeň 2013

Filip Markvart

# Prohlášení

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

V Plzni dne 30. dubna 2013

Filip Markvart

# **Abstract**

## **Data model of EEG/ERP portal using semantic web resources**

This thesis describes transformation of neuroinformatic portal data model to semantic web resources. The first part is aimed to describe semantic web technologies and tools that is used to analyse and compare advantages and disadvantages of relational and semantic web data models. The second part of thesis describes the proces of transformation original data model to a new proposed hybrid model. This part also introduces implemented tools that allows creating and visualizing a new data model. The last part of the work is aimed to testing the proposed data model, especially to the performance testing.

# Obsah

<b>1</b>	<b>Úvod</b>	<b>1</b>
<b>2</b>	<b>Sémantický web</b>	<b>2</b>
2.1	Architektura sémantického webu . . . . .	4
2.2	XML . . . . .	5
2.3	RDF . . . . .	6
2.4	RDFS . . . . .	8
2.4.1	Třídy . . . . .	8
2.4.2	Vlastnosti . . . . .	9
2.5	Ontologie . . . . .	11
2.6	OWL . . . . .	12
2.7	SPARQL . . . . .	13
<b>3</b>	<b>EEG/ERP portál</b>	<b>16</b>
3.1	Technologie portálu . . . . .	16
3.2	Persistentní data . . . . .	17
<b>4</b>	<b>Relační datový model</b>	<b>19</b>
4.1	Limity relační databáze . . . . .	21
4.1.1	Dědičnost . . . . .	21
4.1.2	Vztahy mezi relacemi . . . . .	21
4.2	Datový model EEG/ERP portálu . . . . .	22
<b>5</b>	<b>Datový model sémantického webu</b>	<b>25</b>
5.1	Dynamičnost modelu . . . . .	26
5.2	Dědičnost a hierarchie dat . . . . .	26
5.3	Problémy modelu . . . . .	27
5.4	Porovnání modelů . . . . .	27
<b>6</b>	<b>Nástroje pro vývoj sémantického webu</b>	<b>29</b>
6.1	Virtuoso . . . . .	29

6.2	Oracle 11g - Oracle Spatial . . . . .	30
6.3	Jena . . . . .	31
<b>7</b>	<b>Transformace datového modelu</b>	<b>33</b>
7.1	Rozdělení datového modelu . . . . .	33
7.2	Relační část modelu . . . . .	34
7.3	Sémantická část . . . . .	35
7.3.1	Transformace tabulek . . . . .	35
7.3.2	Transformace rozkladových tabulek . . . . .	36
7.3.3	Transformace atributů tabulek . . . . .	37
7.3.4	Transformace datových typů . . . . .	38
7.3.5	Transformace záznamů tabulek . . . . .	39
7.3.6	Dodání sémantiky modelu . . . . .	39
7.3.7	Řešení rozšiřujících tabulek . . . . .	40
7.3.8	Ukázka transformace . . . . .	40
7.3.9	Výhody navrženého modelu . . . . .	41
7.3.10	Nevýhody navrženého modelu . . . . .	41
<b>8</b>	<b>Implementace navrženého modelu</b>	<b>43</b>
8.1	Formální zápis modelu . . . . .	43
8.2	Konstrukce modelu . . . . .	45
8.2.1	Knihovna SemWebModelDbConnector . . . . .	45
8.2.2	Vytvoření modelu . . . . .	46
8.2.3	Vytvoření testovacích dat . . . . .	47
8.3	Využití modelu . . . . .	48
8.3.1	Operace čtení . . . . .	49
8.3.2	Filtrování . . . . .	50
8.3.3	Operace zápisu . . . . .	51
8.3.4	Operace mazání . . . . .	52
8.3.5	Přístup k relačním datům . . . . .	52
<b>9</b>	<b>Testování</b>	<b>54</b>
9.1	Funkční testování . . . . .	54
9.1.1	Manuální testování aplikace . . . . .	57
9.2	Výkonnéstní testování . . . . .	59
9.2.1	Testovací data . . . . .	59
9.2.2	Testovací operace . . . . .	60
9.2.3	Postup testování . . . . .	61
9.2.4	Testovací prostředí . . . . .	61
9.2.5	Naměřené hodnoty . . . . .	62
9.2.6	Diskuze výsledků . . . . .	66

**10 Závěr**

**68**

# 1 Úvod

EEG/ERP portál je webová aplikace sloužící výzkumným pracovníkům ke shromažďování a organizaci dat získaných při neuroinformatických experimentech v EEG laboratoři. Jejím cílem je ukládání naměřených dat v kontextu prováděného experimentu, který lze popsat rozsáhlou množinou různorodých údajů. Tato aplikace již prošla mnohaletým vývojem v jehož průběhu postupně docházelo ke změnám datového modelu kvůli přibývajícím požadavkům na uchovávaná data. Relační databáze jež slouží jako persistentní úložiště tak postupně byla rozšiřována o další tabulky, jejichž počet se k datu tvorby této práce pohybuje v řádu desítek. Většina realizací požadavků na ukládání dalších dat tak přímo znamená zásah nejen do databáze portálu ale také to do datové vrstvy, která ji využívá. V současné době tak databáze obsahuje velké množství tabulek uchovávající různá data, která jsou ale ve smyslu sémantiky často příbuzná a existuje mezi nimi vazba, která je prostřednictvím relačního datového modelu velmi obtížně popsatelná. Zároveň lze očekávat, že budou přibývat požadavky na uchování dalších dat, která navíc nemusejí mít jen homogenní strukturu (ve smyslu relační databáze), ale může se jednat i o množiny sémanticky příbuzných údajů – tzv. metadata, které budou vázány pouze k některým datům. Možnost ukládání strukturně heterogenních, ale sémantický příbuzných metadat je tak dalším otevřeným problémem.

Cílem této práce je prozkoumání struktury a nalezení sémantiky dat v současném datovém modelu relační databáze portálu a následná úprava tohoto modelu do podoby, která by dovolovala uchovat jak sémantiku dat, kterou není možné relačním modelem vyjádřit tak dodávat dynamicky datům přídavná metadata, aniž by muselo docházet k větším zásahům do datového modelu portálu. Pro realizaci úpravy datového modelu budou v této práci využity prostředky tzv. sémantického webu, který poskytuje množství standardů a technologií pro uchovávání, organizaci a správu dat. Tyto technologie a nástroje zde budou popsány a na základě jejich analýzy budou vybrány prostředky, které se využijí pro implementaci úpravy zmiňovaného datového modelu. Poslední část práce se venuje testování modifikovaného modelu a to především z výkonnostního hlediska. Díky této části by mělo být možné posoudit jak užitečnost samotné úpravy tak i použitelnost a efektivnost získaného modelu pro potřeby EEG/ERP portálu.

## 2 Sémantický web

Dnešní podoba webu, tak jak je všeobecně známa, je tvořena značným množstvím informací, které mají řadu autorů, v podobě různých organizací či jednotlivců, jež se liší jak svým obsahem tak i podobou publikace. Tyto informace jsou poměrně snadno přístupné díky jejich jednoznačné identifikaci prostřednictvím URI identifikátoru (za předpokladu, že jej známe). K usnadnění získávání dalších (často příbuzných) informací napomáhají tzv. hypertextové odkazy, jež usnadňují přístup k dalším zdrojům informací odstraněním požadavku na uživatelovu znalost identifikátoru cílového zdroje. Samotné hypertextové odkazy tak sice zajišťují provázání jednoho informačního zdroje s jiným díky znalosti jeho URI identifikátoru, ale nenesou už žádné další informace, které by například uživateli poskytly další údaje o cílovém zdroji. Takováto podoba umožňuje získávání informací jak koncovým uživatelům webu, tak v omezené podobě i vyhledávacím strojům, ale má své limity, neboť se v nepřeberném množství dat lze snadno ztratit, či se jen dostat k irelevantním informacím [32]. Základním úkolem sémantického webu, jehož první myšlenky prezentoval v roce 2001 zakladatel konsorcia W3C Tim Berners-Lee, je umožnit aby informace dostupné prostřednictvím webu byly srozumitelné nejen uživatelům, ale také počítačům, jež tato data zpracovávají [25]. Hlavním cílem je tedy vývoj standardů a technologií, které by umožňovaly přesnější a podrobnější vyhledávání, integraci dat a také automatizaci častých úkonů. Sémantický web je založen na několika principech, které budou níže uvedeny.

- **Jednoznačná identifikace entit prostřednictvím URI**

Veškerá data, reprezentující obvykle objekty reálného světa publikovaná prostřednictvím webu je možné jednoznačně odkazovat prostřednictvím identifikátoru URI. Díky této skutečnosti je tak možné realizovat i nepřímé odkazy na objekty, například osobu Petr Novák s emailem petr.novak@w3.org je možné identifikovat jako osobou, jejíž email má URI mailto:petr.novak@w3.org.

- **Zdroje i odkazy mezi nimi je možné typovat**

Současná podoba webu je tvořena zdroji a odkazy jež je vzájemně propojují. Zdroje, které jsou reprezentovány webovými dokumenty jsou publikovány za účelem poskytnutí informací lidskému uživateli, který dokáže ze samotného obsahu dokumentu získat i některá jeho metadata

(pokud jsou v určité formě součástí obsahu) a do jisté míry pak také vztah k ostatním dokumentům, na něž vedou případné odkazy. Stroje v podobě různých vyhledávačů či automatů pro shromažďování dat ale tuto schopnost nemají nebo je pro ně příliš náročná. Řešením sémantického webu je typování jak samotných zdrojů, tak i odkazů, které je provazují. Díky této skutečnosti je pak možné webovým dokumentům dodávat metadata jako např. autora, verzi či závislost na jiném dokumentu. Z hlediska typování odkazů je například možné jeden webový zdroj označit pouze jako odlišnou verzi jiného zdroje.

- **Tolerance neúplných informací**

U současné podoby webu může nastat situace, kdy některý zdroj není dostupný. V takovém případě uživatel ztrácí přístup k danému dokumentu, ale díky koncepci webu není nikterak ohrožena dostupnost ostatních zdrojů. V případě sémantického webu se situace nemění, nedostupnost některého zdroje není žádnou překážkou, neboť nástroje sémantického webu zpracovávají pouze ty informace, které jsou dostupné a z těch vytvářejí závěry. V důsledku je tak možné dojít při zpracovávání dat ke stejným výsledkům, jako v případě, když jsou zpracovávány jen některé vybrané informace, jejichž rozsah je explicitně definován.

- **Zpracování neověřených dat**

Při zpracovávání informací pocházejících z neověřených zdrojů je možné dohledávat prostřednictvím typovaných odkazů důvěryhodná data, jejichž obsah a odkazy poslouží jako ověřovací prostředek. Tento princip je možné uvést na následujícím příkladu. Aplikace zpracovávající data sémantického webu vyhledá informace, přičemž je kladen požadavek na vysokou pravděpodobnost správnosti výsledku. Pokud některá část nalezených informací pochází z neověřeného zdroje, je možné vyhledávat například jejich autora v odkazech zdrojů, které jsou důvěryhodné či ověřené. V případě úspěchu nalezení takového odkazu u více různých zdrojů je pak možné považovat zkoumaný zdroj s vysokou pravděpodobností rovněž za důvěryhodný a tím zajistit plnění požadavků na výsledek.

- **Podpora paralelního vývoje dat**

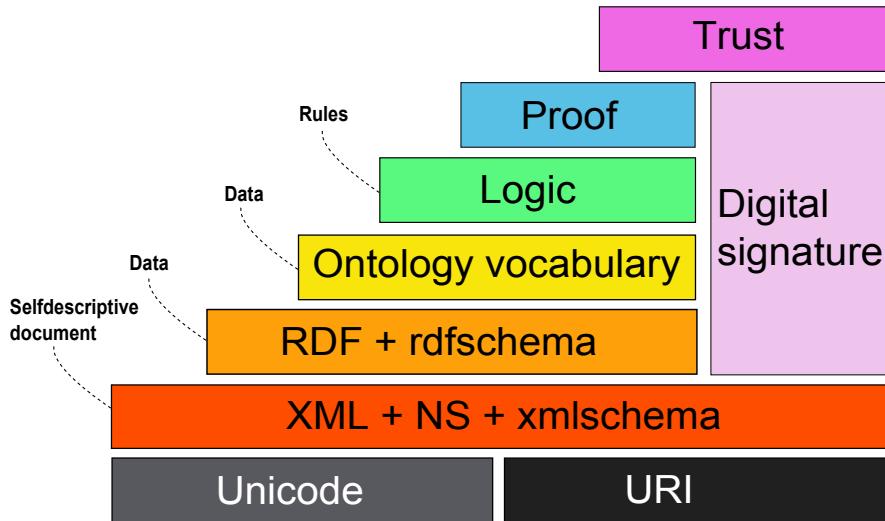
V průběhu času nezřídka nastávají situace, kdy autoři, či skupiny autorů publikují obdobná data na různých místech nebo v odlišném čase. Obsah těchto dokumentů se může navíc lišit svým jazykem či použitou terminologií, ač význam bude shodný. S využitím prostředků sémantického webu je ale možné prostřednictvím typovaných odkazů zajistit provázanost významově obdobných či na sebe navazujících dat

i přes překážku rozdílnosti jejich podoby zápisu. Navíc je také možné dodávat nové informace bez nutnosti úpravy původních dat, která tím pádem nezmění svoji strukturu [32].

## **2.1 Architektura sémantického webu**

Architektura sémantického webu sestává z více oddělených vrstev, mezi nimiž je zajištěna zpětná i dopředná kompatibilita [23]. Nejnižší vrstva je tvořena dvěma technologickými standardy – URI identifikátory sloužící pro jednoznačné pojmenování zdrojů dat a Unicode kódování mezinárodní znakovou sadou. Druhou vrstvu architektury, jež je patrná z obrázku 2.1, reprezentuje značkovací jazyk XML (Extensible Markup Language), který umožňuje tvorbu strukturovaného dokumentu za užití vlastních značek. Tato vrstva zároveň zajišťuje definici XML schématu včetně jmenných prostorů. RDF + rdfschema jež následuje je klíčovou vrstvou sémantického webu neboť dovoluje tvorbu vazeb a vztahů mezi jednotlivými zdroji, které jsou typované spolu s odkazy. Je tak možné definovat libovolné vztahy mezi objekty či jejich kategoriemi bez nutnosti specifikace významu samotných vazeb či objektů. Díky RDF schématu je vytvářena základní sémantika datového modelu, která už definuje význam některých elementů jako třídy či podtřídy. Vrstva ontologického slovníku, zastoupená jazykem OWL, nabízí pokročilou reprezentaci znalostí na úrovni deskripční logiky a umožňuje tak vytvářet složitější struktury sloužící k popisu různých vlastností objektů [25]. Poslední vrstvou, která je jako všechny předchozí zmíněně konsorciem W3C standardizovaná jsou digitální podpisy. Ty poskytují možnosti například pro detekci různých verzí dokumentů. Zbylé výše znázorněné vrstvy slouží pro definice a vyhodnocování odvozovacích pravidel a v současnosti jsou ve fázi vývoje [32].

Vrstvení jazyků sémantického webu je podstatné pro úroveň expresivity znalostního modelu, neboť s rostoucí vyjadřovací možností jazyka také roste složitost dotazovacích operací nad modelem. Je tedy nutné před započetím tvorby modelu najprve zjistit jeho požadovanou expresivitu a podle té zvolit pro zápis dat jazyk, který ji dovoluje obsáhnout. Využívá se tedy skutečnosti, že jazyk vyšší vrstvy zahrnuje vyjadřovací schopnosti vrstev nižších[25]. Další podkapitoly se budou zabývat podrobněji jednotlivými zmíněnými technologiemi.



Obrázek 2.1: Architektura sémantického webu [32]

## 2.2 XML

XML (eXtensible Markup Language) je značkovací jazyk sloužící pro popis hierarchických struktur textových dokumentů prostřednictvím tzv. tagů. Tag je konstrukce, která slouží k počátečnímu a koncovému ohraničení společně definovaného elementu. Tag lze chápat jako prostředek pro dodání metadat ke textové struktuře, jež ohraničuje. Příkladem může být následující zápis `<prijmeni>Novák</prijmeni>`, kde elementu *Novák* je dodána meta informace, že se jedná o příjmení. Samotné XML ale nedefinuje žádný sémantický význam tagů, slouží pouze pro specifikaci syntaxe na úrovni XML dokumentu. Pro definici (zejména hierarchické) struktury XML dokumentu slouží XML Schema, které umožňuje zápis pravidel, jež musí cílový dokument dodržovat pro zachování své validity. Ze strany sémantického webu ale nemají pravidla XML Schema žádný sémantický význam a slouží tak pouze pro definici struktury a syntaxe. Zmíněné schéma také definuje základní datové typy (čísla, řetězce, čas a pod.), které nabývají významu v sémantických jazycích jako je RDF [23].

## 2.3 RDF

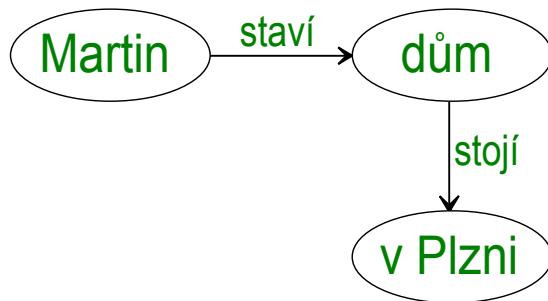
Technologický základ sémantického webu tvoří jazyk RDF (Resource Description Framework), jež slouží jako obecný rámec pro popis, výměnu a opětovné použití metadat [26]. Tento rámec poskytuje jednoduchý model sloužící pro popis zdrojů jež je nezávislý na jeho konkrétní implementaci [14]. Samotné informace o objektu jsou realizovány prostřednictvím tvrzení, jež se označují jako trojice (anglicky triple). Každou trojici tvoří spolu subjekt, predikát a objekt. Subjekt je libovolný objekt identifikovatelný prostřednictvím URI, který se snažíme prostřednictvím trojice popisovat, zaznamenat nějakou jeho vlastnost. Tato vlastnost se popisuje prostřednictvím predikátu, který vede ve směru od subjektu ke objektu, přiřazuje tedy subjektu nějaký objekt prostřednictvím této vlastnosti. Cílový objekt pak představuje hodnotu, které předchozí objekt nabývá pro daný predikát. Tento princip je možné znázornit na jednoduchém tvrzení, zapsaném větou „Martin staví dům.“ Subjektem trojice potom bude Martin, predikátem staví a objektem dům, tak jak je znázorněno na obrázku 2.2.



Obrázek 2.2: Příklad RDF trojice

Dle definovaného standardu [30] je možné, aby objektem byl jiný subjekt. Díky této skutečnosti je možné jednotlivé trojice spojovat do většího celku, který ve výsledku tvoří strukturu orientovaného grafu, kterou lze označit jako model [23]. Jednoduchým model tvořený dvěma trojicemi lze vytvořit využitím dalšího tvrzení „Dům stojí v Plzni.“ V předchozí trojici pak bude dům subjektem namísto objektu a dojde tak ke spojení dvou trojic (za předpokladu že v obou tvrzeních je myšlen stejný fyzický dům), tak jak je znázorněno na obrázku 2.3.

Z hlediska implementace mohou být uzly orientovaného grafu datového modelu tvořeny URI identifikátorem, anonymním listem nebo literálem [28]. URI identifikátor obsahuje pouze adresu zdroje v textové podobě ve znakové sadě Unicode a zastupuje tak konkrétní jedinečný objekt. Anonymní list (anglicky blank node) je prvek nahrazující URI identifikátor při absenci unikátní adresy zdroje. Tato entita přestavuje zdroj, který je sice v rámci



Obrázek 2.3: Příklad jednoduchého RDF modelu

grafu popisován prostřednictvím trojic, ale není potřeba (často z hlediska významu), aby byl dostupný i vně grafu. S URI má společné to, že musí nést adresu zdroje (její syntaxe není implicitně definována), ale liší se skutečností, že tato adresa musí být unikátní pouze v rámci obalujícího modelu, nikoliv vně grafu. Může tak nastat situace, že dva odlišné modely budou obsahovat (ze syntaktického hlediska) dva stejné anonymní uzly, což v případě URI možné není (při respektování specifikace). Tento list tak může představovat anonymní objekt sloužící ke vytvoření vazby mezi jinými objekty, které jsou z hlediska sémantiky významné [30]. Jako příklad je možno uvést následující tvrzení. „Martinův přítel zná předpověď počasí. Počasí bude deštivé.“ Zjednodušeným převodem předchozích vět na trojice v podobě subjekt – predikát – objekt získáme: *Martin* – *má* – *přítel*, *přítel* – *zná* – *počasí*, *počasí* – *bude* – *deštivé*. Subjekt resp. objekt přítel zde může být reprezentován anonymním listem, v případě že jedinou signifikantní informací (z vnějšího pohledu na datový model) je Martinova získaná znalost počasí, nikoliv už jeho přítel, jež mu ji zprostředkoval. Poslední možnou reprezentaci entity trojice je literál, jež nese Unicode textový řetězec, který slouží pro zápis koncové informace (užitečné pro lidského čtenáře). Literál může také navíc obsahovat položku, pro označení jazyka, ve kterém je textová informace zapsána [11].

Pro komponenty každé trojice grafu platí následující pravidla [28].

- **subjekt** - může být tvořen URI identifikátorem nebo anonymním listem
- **predikát** - může být tvořen pouze URI identifikátorem
- **objekt** - může být tvořen URI, anonymním listem nebo literálem

Pro zápis trojic RDF grafu je sice možné využít grafické podoby, ale pro vyjádření sémantiky webových zdrojů je nejhodnější využít syntaxe jazyka XML. Níže uvedená ukázka kódu reprezentuje XML zápis trojice z obrázku 2.2.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description rdf:about="http://www.w3.org/Person/Martin">
    <dc:stavi>dům</dc:stavi>
  </rdf:Description>
</rdf:RDF>
```

Samotné RDF nedefinuje trojicím ani jejím částem sémantiku, ale dovoluje vyjádřit základní vztahy náležitosti prvků do kategorie prostřednictvím kontejnerů a kolekcí (např. *rdf:Bag*, *rdf>List*) [25]. Pro dodání základní sémantiky slouží schéma popsané v následující podkapitole.

## 2.4 RDFS

RDF Schema (Resource Description Framework Schema) funguje jako základní jazyk pro tvorbu ontologií s velmi jednoduchou sémantikou. Toto schéma rozšiřuje jazyk RDF o možnosti vyjádření vlastností objektů, konstrukce tříd objektů a popis jejich hierarchie [25]. Prostředky jazyka RDFS umožňují především vyjádření vztahů mezi zdroji, které lze rozdělit na dvě skupiny – třídy a vlastnosti.

### 2.4.1 Třídy

Skupiny zdrojů je možné rozčleňovat do skupin označovaných jako třídy (classes), jejichž členové se nazývají instance třídy. Na úrovni RDFS se rozlišují třídy od svých instancí a každá třída jich může mít neomezený počet. Dvě třídy mohou mít navíc shodnou množinu instancí tříd a zároveň tyto třídy mohou být navzájem různé. Bude-li se tedy například definovat třída *A* jako osoby pracující v kanceláři *1* a třída *B* jako osoby žijící ve městě *X*. Potom je možné aby různé třídy *A* a *B* měly stejné množiny instancí, za předpokladu,

že každá osoba pracující v kanceláři 1 bydlí ve městě X. Pro třídy platí také dědičnost – bude li třída B podtřídou A, pak všechny instance B jsou zároveň instancí třídy A. Koncept tříd RDFS definuje následující konstrukce [31]:

- **rdfs:Resource** představuje RDF zdroj, který je obalující třídou všech prvků – je tedy nejvyšší postavenou rodičovskou třídou, *rdfs:Resource* je zároveň instancí *rdfs:Class*
- **rdfs:Class** reprezentuje rodičovskou třídu všech RDF tříd zdrojů, čímž je *rdfs:Class* instancí *rdfs:Class* (sebe sama)
- **rdfs:Literal** třída je instancí *rdfs:Class*, která slouží pro reprezentaci RDF literálů a je podtřídou *rdfs:Resource*
- **rdfs:Datatype** je obalující třídou pro datové typy, které jsou její instancí, *rdfs:Datatype* je zároveň podtřídou i instancí *rdfs:Class* a každá její instance je podtřídou *rdfs:Literal*
- **rdf:XMLLiteral** je třída XML literálů, která je podtřídou *rdfs:Literal* a instancí *rdfs:Datatype*
- **rdf:Property** představuje rodičovskou třídu všech definovaných vlastností a je instancí *rdfs:Class*

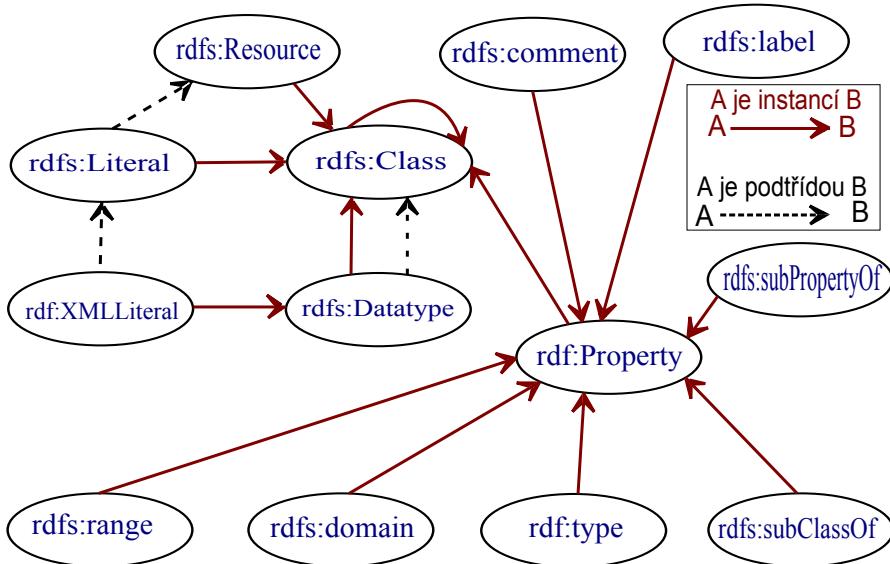
#### 2.4.2 Vlastnosti

Vlastnosti (properties) slouží k vyjádření vztahu mezi dvěma zdroji – na úrovni trojice mezi subjektem a objektem. Koncept RDFS definuje následující vlastnosti:

- **rdfs:range** je instancí *rdf:Property*, která slouží ke vyjádření, že objekt jehož predikát má definovaný *rdfs:Range* X bude zároveň instancí X, například z následujících trojic A – *rdfs:Range* B a X – A – C bude vyplývat, že C je instancí B, zároveň platí, že objekt s definovaným predikátem může být instancí více tříd (pokud má predikát definováno více *rdfs:Range*).
- **rdfs:domain** představuje instanci *rdf:Property*, která vyjadřuje, že zdroj (subjekt) jehož predikát má definovaný *rdfs:Domain* X bude také instancí X, tento zdroj může být jako v předchozím případě instancí více tříd při definování více *rdfs:Domain*

- **rdf:type** slouží k definování, že zdroj (subjekt) je instancí třídy definované objektem, pokud tedy platí  $A \text{ rdf:type } B$ , pak  $A$  je instancí  $B$
- **rdfs:subClassOf** je vlastnost sloužící k vyjádření náležitosti instancí jedné třídy jako instancí jiné třídy, pokud platí  $A \text{ rdfs:subClassOf } B$ , pak  $A$  je podtřídou  $B$  a všechny instance třídy  $A$  jsou zároveň instance třídy  $B$ , tato vlastnost je navíc tranzitivní, takže popsanou dědičnost je možné řetězit do libovolné délky
- **rdfs:subPropertyOf** slouží k vyjádření dědičnosti vlastností, pokud platí  $P_1 \text{ subproperty } P_2$ , pak pro trojici  $A \ P_1 \ B$  platí, že subjekt  $A$  má pro vlastnost  $P_1$  i  $P_2$  pro objekt  $B$
- **rdfs:label** umožňuje zdroji (subjektu) přidat textovou informaci jako lidsky čitelnou náhradu pro označení zdroje
- **rdfs:comment** dovoluje přidat zdroji popisek, který usnadňuje lidskému uživateli pochopit význam zdroje [31]

Vyjádření popisovaných vztahů mezi jednotlivými vlastnostmi a třídami je patrné z obrázku 2.4. Rámec RDFS dále ještě definuje vlastnosti a třídy pro kontejnery a kolekce, které je možné naleznout ve [31].



Obrázek 2.4: Vztahy mezi vlastnostmi a třídami RDFS

## 2.5 Ontologie

Znalostní modely, jež jsou popisované vyššími jazyky sémantického webu se označují jako ontologie. Ontologie ovšem není pouze abstraktním znalostním modelem, ale slouží i jako prostředek k získání interoperability, tedy schopnosti vzájemné spolupráce oddělených systémů na úrovni sdílení dat a poskytování služeb [25]. Definicí ontologie v informatice kterou uvádí Thomas Gruber, zakladatel ontologického inženýrství, je „formální specifikace sdílené konceptualizace“ [20]. V této definici je formálností myšleno vyjádření znalostí prostřednictvím jazyka s formálním a logickým základem. Pojem konceptualizace znamená definování konceptů na dané úrovni abstrakce, jež odpovídá požadavkům pro model domény. Tuto doménu pak vytvářejí osoby, pro které je daná ontologie závazně (vzájemnou dohodou) definovaná jako prostředek pro popis dat, jež musejí všichni členové dodržovat [25]. V důsledku se tak jedná o tvorbu abstraktního modelu v určené oblasti, kde se definují pojmy společně se vzájemný vztahy vyjádřené logickým jazykem. Cílem explicitní specifikace pojmu a vztahů je snaha o porozumění znalostně orientovaných systémů [19]. Ontologie lze z hlediska vývoje rozčlenit do tří kategorií:

- **Terminologické (lexikální)** Slouží především ke zachycení taxonomie definovaných pojmu a popisu jejich vzájemných vztahů. Jejich realizace se často podobá slovníkům synonym.
- **Informační** Zastávají roli služby stojící nad databází, která zajišťuje vyšší míru abstraktnosti při databázovém dotazování.
- **Znalostní** Dovolují reprezentaci znalostí především v oblasti umělé inteligence, kde jsou chápány jako logické teorie, jejichž elementární prvky se definují prostřednictvím formálního jazyka. Využívají se zejména ve znalostně orientovaných systémech [19].

Dle míry formalizace a cílového předmětu konceptualizace se rozlišují 4 kategorie ontologie.

- **Doménové** Zabývají se specifikací určité oblasti – domény.
- **Generické** Blíží se svou podobou doménovým, ale pokrývají širší množinu dat a zachycují tak obecnější koncepty, nejdou ovšem do příliš velké hloubky. Popisují například realitu prostřednictvím vzájemných

časoprostorových pozic objektů. Často se snaží zachytit všeobecné znalosti, které odpovídají běžně používaným lidským vědomostem.

- **Úlobové** Tyto ontologie jsou zaměřené zejména na odvozovací procesy namísto záznamu znalostí a slouží především jako modely pro řešení určitých problémů – např. pro diagnostiku či plánování.
- **Aplikační** Jsou většinou vázány na specifické aplikace, kde spojují doménovou a úlobovou ontologii [29].

Pro realizaci popsaných ontologií je zapotřebí jazyka, který by dovoloval formální zápis. Výše uvedené schéma RDFS je z hlediska sémantického webu základní podobou ontologie, kterou je dále možné rozšiřovat například využitím jazyků jako je OWL.

## 2.6 OWL

OWL (Web Ontology Language) je jazyk sémantického webu navržený pro získávání a zpracování informací prostřednictvím strojů (aplikací), namísto současného stavu, kdy jsou informace určeny pouze pro lidské uživatele. Tento jazyk využívá technologií XML, RDF a RDFS jakožto primárních prostředků pro tvorbu ontologických slovníků se základní sémantikou, kterou sám dále rozšiřuje o možnosti vyjádření složitějších výrazů a jejich vzájemných vztahů [3]. Mezi tato rozšíření patří například kardinalita, univerzální a existenční kvantifikace, matematické charakteristiky vlastností jako např. tranzitivnost, disjunktnost či inverze a anonymní třídy (sloužící často k jednorázovému využití) [21]. Vzhledem k tomu, že OWL nabízí poměrně pokročilou úroveň expresivity jazyka, je nutné brát v potaz výpočetní složitost algoritmů odvozovacích a usuzovacích nástrojů pracujících s vytvořenou ontologií. Z tohoto důvodu jsou specifikovány 3 varianty jazyka OWL, které se vzájemně liší mírou expresivity [25].

- **OWL Lite** Umožňuje definovat hierarchický systém tříd s jednoduchými omezeními vazeb [12]. Dále poskytuje prostředky pro zaznamenání symetrické, transitivní a inverzní vlastnosti a také jednoduchá omezení na velikosti množin vybraných objektů modelu – kardinalitu, která je ale v této verzi omezena pouze na přípustné hodnoty 0 a 1 [25].

- **OWL-DL** Tato varianta poskytuje maximální možnou expresivitu jazyka, u které je stále splněna podmínka rozhodovací úplnosti (jakékoliv rozhodovací pravidlo je možné použít na kteroukoliv část ontologie) a výpočetní splnitelnosti (veškeré výsledky odvozovacích operací budou získány v konečném čase) [25]. OWL-DL nabízí veškeré konstrukce jazyka OWL, které ale podléhají určitým omezením při jejich použití, např. není možné využít omezení kardinality pro vlastnosti definované jako tranzitivní. Z hlediska množiny dostupných jazykových konstrukcí dovoluje tato varianta oproti OWL Lite definovat navíc sjednocení, disjunkce a doplnky tříd nebo libovolné omezení kardinality [12].
- **OWL-Full** Je variantou, která poskytuje maximální možnou expresivitu, ke které využívá stejně jako předchozí verze všech konstruktů jazyka OWL. Tato varianta neklade žádná omezení pro výhodnocovací pravidla, díky čemuž ale není možné zaručit výpočetní splnitelnost. OWL-Full dovoluje ještě navíc uživatelskou změnu sémantického významu nativně definovaných konstrukcí jazyků RDF a OWL. Tato verze ovšem není prakticky příliš využitelná, nebot' v současnosti nejsou známé žádné efektivní algoritmy, které by dovolovaly nad modelem dat provádět usuzovací operace využívající všech vlastností OWL-Full [25].

Každá verze OWL je rozšířením svého předchůdce díky čemuž je ontologie vyjádřená v jednodušší variantě platná i pro verzi vyšší, např. tedy každá ontologie zapsaná ve OWL Lite bude plně validní ve OWL-DL (což ale obráceně neplatí) [12]. Mezi typické odvozovací úlohy v kontextu OWL patří například odvozování taxonomické struktury, ověřování příslušnosti instance ke třídě, klasifikace individuí vzhledem k definované ontologii či testování splnitelnost logické teorie [21].

## 2.7 SPARQL

Základním prostředkem pro dotazování nad daty sémantického webu je jazyk SPARQL (SPARQL Protocol and RDF Query Language). Tento jazyk standardizovaný konsorciem W3C umožňuje vytvářet dotazy nad RDF grafy prostřednictvím vzorů RDF trojic spolu s logickými operacemi konjunkce a disjunkce [25]. Konstrukce těchto dotazů je sice složitější než v případě SQL dotazů nad relačními databázemi, ale umožňuje ze zdrojové ontologie získávat komplexnější informace [23]. SPARQL dotaz se skládá ze 3 základních částí.

První část – označovaná jako prolog slouží k definování jmenných prostorů a prefixů, které budou v dalších částech dotazu použity. Druhou částí je hlavička dotazu, kde se určuje jaký typ dotazu se bude v další části provádět. Třetí a hlavní část SPARQL slouží k definování cílové ontologie (RDF grafu) pro dotazování (v případě, že cílový graf je implicitně definován, je tato část vynechána) a proměnných včetně samotných vyhledávacích podmínek v podobě grafových vzorů (graph pattern)[25]. Tento jazyk částečně vychází ze SQL, což je patrné i na jeho syntaxi. Např. klauzule FROM slouží pro výběr cílového grafu pro dotazování, či klíčové slovo WHERE slouží pro uvození zápisu souboru RDF trojic – tzv. grafového vzorce, jež je jádrem dotazu. Vyhodnocování dotazu poté probíhá takovým způsobem, že dochází k porovnávání grafového vzorce se s RDF daty [22]. Pro názornost je níže uveden zápis jednoduchého SPARQL dotazu, který slouží k vyhledání jmen a příjmení všech osob v implicitně zadáném grafu.

```
PREFIX zcu: <http://www.zcu.cz/ns/students>
SELECT ?name ?surname
WHERE {
    ?person x zcu:Person.
    ?person zcu:Name ?name.
    ?person zcu:Surname ?surname
}
```

SPARQL podporuje 4 druhy dotazů, které se vzájemně liší v typu vráceného výsledku [22].

- **SELECT** Tento typ dotazu nejvíce odpovídá běžnému SQL dotazování, neboť zadáným proměnným nastaví příslušné hodnoty a ty pak vrací v podobě tabulky [33].
- **ASK** Dotaz typu ASK vrací pouze TRUE/FALSE odpověď a slouží ke otestování, zda zadaný grafový vzorec má pro daný graf řešení. ASK tak slouží ke testování vytvořených logických hypotéz, ale neumožňuje už vrácení konkrétních řešení [24].
- **DESCRIBE** Dotazy typu DESCRIBE navrací podgraf dotazovaného grafu, který obsahuje všechny dostupné informace o zdroji, jež vyhověl zadánému grafovému vzoru [22]. Při vyhodnocování dotazů dochází ke porovnávání zdrojů získaných ze vstupního grafového vzoru se zdroji celého grafu. Pro nalezené odpovídající zdroje jsou pak získány všechny

vázané relevantní informace, ze kterých je ve výsledku složen navracený graf [33].

- **CONSTRUCT** Tento speciální druh dotazu navrací jako výsledek nový RDF graf dle definované grafové šablony. Výsledný RDF graf je konstruován podle částečných řešení dotazovací sekvence, jež vyplňují proměnné ze zadané grafové šablony na základě které jsou tvořené výsledné trojice pro cílový navracený graf [24]. Tato dotazovací operace se do jisté míry podobá XSLT transformaci XML dokumentu.

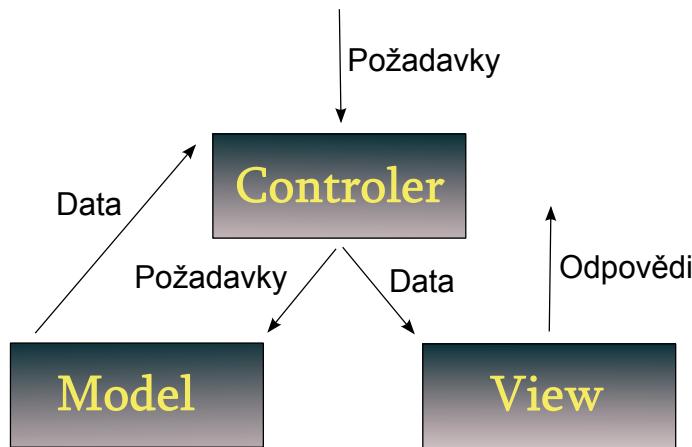
## 3 EEG/ERP portál

EEG/ERP portál je webová aplikace, sloužící vědeckým pracovníkům k ukládání, stahování a vyhledávání EEG/ERP experimentů včetně jejich přidružených metadat. Tato aplikace je vyvíjena pro Katedru informatiky a výpočetní techniky Západočeské univerzity v Plzni. Portál funguje jako komplexní aplikace, která neslouží jen ke zpracování dat experimentů, ale poskytuje také funkce pro správu uživatelů, jež mohou mít odlišné role a také se sdružovat do výzkumných skupin jež mohou vzájemně spolupracovat. Portál je vyvíjen jako open-source aplikace pod GNU licencí. Jeho základní funkcionalitu je možné shrnout do několika následujících bodů [18].

- Ukládání, aktualizace a stahování naměřených dat experimentů včetně metadat
- Registrace nových uživatelů, jejich sdružování do skupin včetně určení rolí
- Sdílení dat a metadat experimentů mezi výzkumnými skupinami
- Publikační a diskusní prostředky pro uživatele v podobě článků a jejich komentářů
- Fulltextové vyhledávání nad celou databází
- Přihlašování uživatelů skrze interní účty či prostřednictvím sociálních sítí
- Rezervace výzkumné laboratoře

### 3.1 Technologie portálu

Z technologického hlediska je EEG/ERP portál webová aplikace se standardní třívrstvou architekturou MVC (Model – View - Controler) - viz obrázek 3.1, běžící na aplikačním serveru Tomcat. Portál je vyvíjen v programovacím jazyce Java EE, přičemž při implementaci je využito několika frameworků, zejména pak Spring.



Obrázek 3.1: Architektura MVC

Prezentační vrstva je realizována prostřednictvím technologie JSP (Java Servlet Pages), zabezpečení a přihlašování uživatelů pak využívá frameworku Spring Security. Dále je pluginem Spring Social zajištěno prostřednictvím definovaného API propojení webové aplikace se sociálními sítěmi Facebook, Twitter a LinkedIn. Spring Social také slouží ke integraci možnosti přihlašování uživatelů prostřednictvím účtů zmíněných sociálních sítí.

Aplikační vrstva využívá zejména technologií Spring MVC a Spring Core a zajišťuje zpracování veškerých uživatelských požadavků pro ukládání a čtení dat, která získává z datové vrstvy. Persistentní data jsou uložena v relační databázi Oracle 11g, přičemž k jejich zpracování je použito objektově-relační mapování frameworku Hibernate. Ten využívá XML souborů pro mapování tabulek na POJO objekty, jež tvoří persistentní vrstvu. Framework Hibernate zároveň umožňuje odstínit datovou vrstvu od implementační závislosti na zvolené databázi, jejíž případná obměna by následně nevyžadovala příliš velké zásahy do zdrojových kódů portálu [7].

## 3.2 Persistentní data

Databáze Oracle jež slouží ke ukládání persistentních dat uchovává veškeré naměřené hodnoty zaznamenaných experimentů. Tato data mají z pohledu databáze podobu binárních souborů s velikostí řádu desítek až stovek MB.

Tato naměřená data by ovšem ztrácela hodnotu, kdyby byla uchovávána jako samostatná. Proto databáze obsahuje další tabulky, jež nesou metadata prováděných experimentů. Kontext experimentu je popsán především hodnotami o prostředí laboratoře a zúčastněných osobami. Prostředí laboratoře je charakterizováno například teplotou, použitým hardwarem a softwarem. Nedílnou součástí popisu experimentu je také scénář zaznamenávající samotný postup měření, který má podobu binárního souboru. Dalšími důležitými daty jsou také informace o použitých elektrodách měřících EEG, u kterých je důležitý jejich typ a poloha. Co se týče osob účastnících se experimentu, zaznamenává se měřící a především měřená osoba. U osob jsou uchovávány osobní a kontaktní údaje a také data nutná pro přihlašování do portálu. Osoby se mohou sdružovat do různých výzkumných skupin, jež nesou seznam svých členů a vlastníka. Podstatným údajem jsou také identifikátory skupin vázané k hodnotám číselníků charakterizující vlastnosti experimentu, které tak napomáhají výzkumníkům zaznamenávat vlastní vybrané předdefinované hodnoty. Další tabulky ještě zaznamenávají historii stahování, zveřejňované články a jejich textové komentáře. Rezervace laboratoře obsahuje údaje o počátečním a koncovém času události a také rezervující osobě.

## 4 Relační datový model

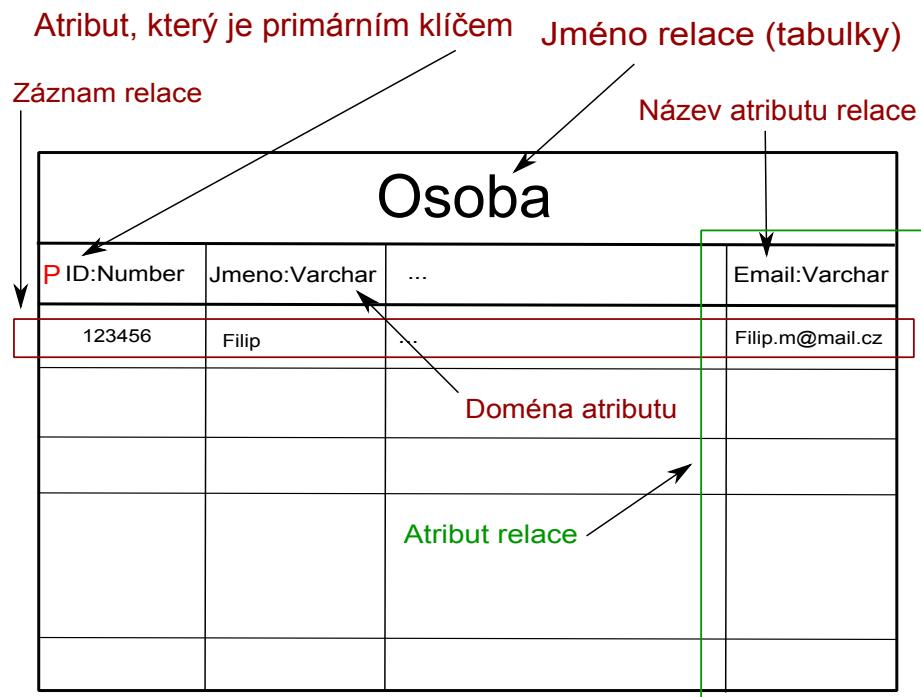
Jak již bylo zmíněno v předchozí kapitole, webová aplikace EEG/ERP portál uchovává data v relační databázi – lze na ně tedy pohlížet jako na relační datový model. Vzhledem k tomu, že další část práce se bude zabývat úpravou současného datového modelu portálu, bude vhodné uvést zde základní popis a vlastnosti obecného relačního modelu.

Základem relačního modelu dat, který byl popsán již v roce 1970 je takzvaná relace neboli tabulka [6]. Z matematického hlediska můžeme  $n$ -ární relaci chápat jako libovolnou podmnožinu kartézského součinu  $n$  množin [27]. Množinou je zde myšlena doména jednoho konkrétního sloupce tabulky, tedy všechny přípustné hodnoty, kterých může libovolný řádek v daném sloupci tabulky nabývat. V důsledku tedy máme tabulku tvořenou  $n$  sloupci ( $n$ -ární relace) a  $x$  řádky, kde každý řádek reprezentuje jeden prvek z množiny tvořené kartézským součinem  $n$  množin. V případě, že by každá z  $n$  množin byla konečná (řádky daných sloupců mohou nabývat konečného počtu hodnot), bude konečný i kartézský součin těchto množin a tabulka tak může mít jen konečný a tudíž omezený počet neopakujících se řádek (duplicity v tabulce neuvažujeme, nebot' by neměly žádný smysl – jednalo by se o redundantní data) [27].

Uvedená reprezentace matematického pojmu relace jako databázové tabulky je základem takzvaného schéma relace. Toto schéma obsahuje název relace, jména všech atributů (sloupců tabulky) a jejich integrálních omezení – tedy domény [5]. Domény jednotlivých atributů jsou v systémech řízení báze dat reprezentovány datovými typy, tedy např. číselný typ, textový řetězec či datum. Tabulku lze tedy chápat jako určitou formu znázornění relace [8]. Jedna tabulka slouží zpravidla k uchování údajů o jednom druhu objektů [10].

V případě portálu tak například tabulka *Person* uchovává pouze data o osobách – jeden řádek neboli záznam tedy reprezentuje údaje o jednom člověku, které jsou strukturované do sloupců, např. jméno, příjmení, email apod. Názornější představu konkrétní relace je možné získat z obrázku 4.1. Aby bylo možné s daty tabulky pracovat, zavádí se aparát, který zajišťuje jednoznačnou identifikaci každého záznamu - primární klíč. Primární klíč je atribut či skupina atributů, která je pro každý záznam tabulky jedinečná (stejně jako název tabulky v databázi) [10]. V případě zmíněné tabulky *Person* by tak mohl posloužit například atribut email, nebot' žádné dvě osoby nemo-

hou logicky mít stejnou emailovou adresu (což by mělo být zajištěno ze strany



Obrázek 4.1: Znázornění relace

domény dané emailové adresy). V případě že tabulka neobsahuje žádný atribut, který by potřebnou jedinečnost záznamů zajišťoval, je nutné jej přidat. To je realizováno např. atributem ID, obsahujícím číslo z číselníku, jehož hodnota je inkrementována po každém přidání dalšího záznamu do tabulky *Person*. Tento identifikátor položky sice jednoznačně určuje daný záznam v tabulce, nikoliv už však v rámci celé databáze (pokud uvažujeme více tabulek) či vně databáze. V jedné databázi je tedy možné mít například tabulky *Person* a *Experiment*, jejichž primárními klíči bude atribut s názvem ID typu celého čísla. V takovém případě je pak i možné aby obě tabulky obsahovaly 2 různé záznamy jež budou jednoznačně identifikovatelné stejnou číselnou hodnotou (ale v kontextu dané tabulky). Z toho důvodu je pro jednoznačnou identifikaci záznamu nutné znát jak hodnotu příslušného atributu obalující relace, tak i její název.

## 4.1 Limity relační databáze

### 4.1.1 Dědičnost

Relační databáze může samozřejmě obsahovat větší množství tabulek (za předpokladu že mají vzájemně různé názvy). Každá tabulka tak sdružuje záznamy, jejichž obsah si je v určitém smyslu blízký. Z hlediska objektově orientovaného přístupu, který je realizovaný např. nástrojem Hibernate, lze tabulkou chápat také jako třídu a její záznamy jako instance dané třídy. Každá takováto instance má pevně daný počet atributů, který je definován obalující třídou. Uvažujme například tabulku *Person*, která ponese údaje jak o osobách které jsou cílovým objektem experimentů tak údaje o výzkumných pracovnících. U obou skupin nás budou zajímat společné atributy jako jméno a příjmení, ale některé atributy budou různé. U testovaných osob to bude např. věk, zatímco u výzkumníků např. titul. Uvažujeme-li objektový přístup k takovéto tabulce, můžeme využít dědičnosti. Vytvoříme tedy třídu *Person*, která ponese všechny společné atributy a od ní bude zděděna třída pro výzkumníky a třída pro testované osoby. Tyto třídy budou obsahovat atributy, které jsou specifické pro danou skupinu. Z pohledu relačního modelu je tento problém řešitelný obtížněji. Je možné například vytvořit 2 samostatné tabulky pro dané skupiny či ponechat jednu tabulku, která ponese všechny atributy a každá skupina bude mít vyplňena jen jejich část. Podobný problém by vznikl například při potřebě dodat další atribut jen omezenému počtu osob jedné ze skupin. Z uvedeného příkladu je patrné, že dynamické rozšiřování záznamů obsažených v jednotlivých tabulkách je poměrně obtížné.

### 4.1.2 Vztahy mezi relacemi

Relační model dovoluje také zaznamenávat vztahy mezi jednotlivými tabulkami a provozovat tak záznamy z různých tabulek. Tyto vazby se realizují prostřednictvím atributů označovaných jako cizí klíč [10]. Hodnotou cizího klíče je pak hodnota primárního klíče tabulky, která je cílem vytvářené vazby. Z hlediska kardinality vztahu je možné rozlišit 3 základní případy *1:1* , *1:N* a *M:N* [4]. Vztah *1:1* vyjadřuje případ, ve kterém záznam jedné tabulky odpovídá záznamu jinému tabulky. Tento případ není příliš častý a většinou se řeší umístěním obou záznamů do jediné tabulky. Vztah *1:N* reprezentuje situaci, kdy se jednomu záznamu tabulky přiřazuje více záznamů jiné tabulky. Příkladem takové vazby může být vztah záznamů tabulek *Person*

a *Reservation*. Jedna osoba tedy může mít  $N$  rezervací laboratoře, ale každá rezervace má jako vlastníka právě jednou osobou. V modelu je tato informace realizovaná přidáním atributu *person\_id* do tabulky *Reservation*. Tento atribut se tak stává cizím klíčem a ponese hodnotu primárního klíče tabulky *Person* pro příslušný záznam cílové osoby. Samotná vazba tedy obsahuje informaci o zdrojové záznamu, cílovém záznamu a také název vazby který je reprezentován názvem atributu pro cizí klíč tabulky.

Posledním druhem vazby je vztah  $M:N$ , jež slouží k reprezentaci stavu, kdy  $M$  záznamům jedné tabulky odpovídá  $N$  záznamů jiné tabulky. Tento stav je v relačním modelu řešen rozložením na dvě vazby  $1:N$  a  $1:M$ , které jsou pak realizovány vytvořením další tabulky, jejíž záznamy budou obsahovat cizí klíče obou tabulek. Pro realizaci takové vazby je tedy nutné vložit do databáze další tabulku a v případě potřeby rozlišení druhu jednotlivých vazeb je zapotřebí bud' dalšího atributu jež by vztah popisoval, či pro každý druh vazby mít vlastní tabulku (což se týká i případu vazby  $1:N$ ). Pokud by ovšem bylo zapotřebí popsat také vztahy mezi jednotlivými vazbami nastává problém, který je řešitelný velmi obtížně [10].

## 4.2 Datový model EEG/ERP portálu

Současnou podobu relační databáze portálu tvoří přibližně 80 tabulek. Naměřená data experimentů (hodnoty EEG signálů) jsou uložena v podobě binárních souborů v tabulce *DATA\_FILE*, spolu se popisnými údaji jako je název či datový typ mající podobu textového řetězce nebo čísla. Tyto popisné údaje lze označit jako metadata binárního souboru naměřených dat. Metadata scénáře experimentu uložená v tabulce *SCENARIO* obsahují popisné informace dalších binárních souborů, jež jsou uloženy v tabulce *SCENARIO\_TYPE\_NONXML* obsahující popis průběhu měření. Obě tabulky *SCENARIO* a *DATA\_FILE* obsahují vazbu na tabulku *EXPERIMENT*, která obsahuje bud' přímo metadata experimentu (nikoliv však souboru naměřených dat) nebo zajišťuje provázání s jinou tabulkou jež je nese. Z aktuálního modelu je patrné, že jeden experiment je popsán jedním scénářem, ale může obsahovat více (binárních) souborů naměřených dat.

Tabulka *EXPERIMENT* s klíčovými metadaty obsahuje také dvě vazby na tabulku *PERSON*, jež slouží ke zaznamenání měřené osoby a experimentátora (vlastníka experimentu). Z hlediska datového modelu není mezi těmito osobami rozdíl, nebot' jsou všechny uloženy v jediné tabulce a tudíž jsou všechny osoby

popsány stejnou množinou atributů. Některé testované osoby například vůbec nemusejí potřebovat přístup do portálu, ale tabulka ve které jsou zaznamenány obsahuje atributy uživatelského jména a hesla (i když mohou nabývat hodnoty NULL). Stejná situace nastává i pro další referencované tabulky *ARTICLES*, *ARTICLES\_COMMENTS* či *RESERVATION*. Datový model tak nikterak nezabraňuje, aby měřené osoby měly své články, přidávaly k nim komentáře nebo rezervovaly laboratoř (vychází se z předpokladu, že portál slouží zejména pro výzkumné pracovníky, nikoli pro testované osoby).

Obdobná situace nastává v případě tabulky *RESEARCH\_GROUP*, sloužící ke sdružování osob do výzkumných skupin. Její vazby na tabulku *PERSON* zajistují, že každá skupina má svůj název, popis a také vlastníka. Každá skupina může mít samozřejmě více členů, což zajistuje rozkladová tabulka *RESEARCH\_GROUP\_MEMBERSHIP*, jež obsahuje i roli člena ve skupině (ve smyslu uživatelský práv). Samotná tabulka ovšem poměrně značně zasahuje do celého datového modelu, díky množství rozkladových tabulek sloužících ke kategorizaci hodnot specifických metadat experimentů ke vybraným skupinám. Díky tomu je možné, aby vybraná skupina mohla definovat pro své potřeby vybrané hodnoty pro určitá metadata – např. pro tabulku *HARDWARE* definovat množinu přístrojů, které ke svým výzkumům využívá. Díky rozkladovým tabulkám může navíc více skupin sdílet stejné hodnoty položek hardwaru. Tyto tabulky s dodatečnými metadaty experimentu jako například zmíněný *HARDWARE*, *SOFTWARE* či *WEATHER* sice popisují kontext měření, ale z hlediska datového modelu se jedná pouze o nekategorizovaná data vázaná (často nepřímo) na tabulku *EXPERIMENT*. Není tedy možné sdružit některé parametry měření do skupin – např. tabulky *PHARMACEUTICAL* a *DISEASE* označit jako medicínské parametry s specifikovat tak jednoduchou hierarchickou kategorii metadat experimentů.

Další problém, který řeší současný relační datový model jen částečně je dynamická rozšiřitelnost struktury dat (ve sémantickém smyslu). V průběhu vývoje portálu se postupně zvyšoval počet tabulek nesoucích specifická metadata experimentu a je i nadále rozšiřován. Pro případ vzniku potřeby popisu daného experimentu parametrem, jenž není definován jako atribut tabulky *EXPERIMENT* ani jinou samostatnou tabulkou, obsahuje databáze tabulky *EXPERIMENT\_OPT\_PARAM\_DEF* a *EXPERIMENT\_OPT\_PARAM\_VAL*. Ty umožňují zaznamenat další hodnoty ve tvaru parametr-hodnota a navíc je sdílet pro více skupin. Nevýhodou tohoto řešení je ale absence možnosti přidat danému parametru více klíčů (chceme-li se vyhnout změně tabulky přidáváním dalších atributů), či tyto údaje ještě více řetězit. Obdobná situace však nastává i v případě samostatných tabulek pro konkrétní metadata

experimentu.

Pro bližší představu o současném datovém modelu portálu je možné nahlédnout do přílohy D obsahující ERA model EEG/ERP portálu.

## 5 Datový model sémantického webu

Předchozí kapitola se zabývala daty, která jsou sdružovaná a reprezentována prostřednictvím relačního modelu. Stejná data ovšem mohou být také uchována prostřednictvím databáze založené na modelu sémantického webu, jemuž bude věnována tato kapitola. V kapitole sémantického webu byla nastíněna koncepce sémantického webu, včetně reprezentace dat. Reprezentací datového modelu sémantického webu je takzvaný RDF graf, který sestává z libovolného počtu RDF výrazů neboli trojic. Každá trojice je tvořena subjektem, predikátem a objektem, které jsou z pohledu sémantického webu dále nedělitelné [20]. Jedna trojice tak poskytuje jednu elementární informaci o jednom subjektu. Zda-li má objekt trojice podobu jedinečné konkrétní entity už je určeno tím, jestli je reprezentován URI identifikátorem, či se jedná pouze literál nebo anonymní list (blank node). Odlišná situace nastává u subjektu, který může být buď URI identifikátorem nebo anonymním listem. Reprezentace literálem zde již není možná, neboť by se popírala možnost provazování jednotlivých entit trojic, což je klíčová vlastnost sémantického webu [19].

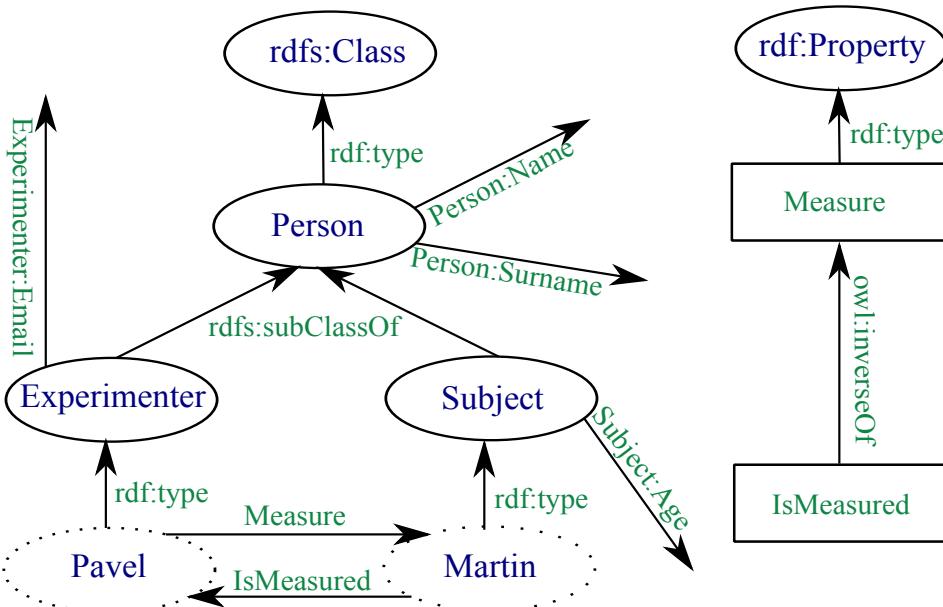
Podstatný rozdíl mezi datovým modelem sémantického webu oproti modelu relačnímu je absence jakýchkoliv tabulek. Struktura dat tak není předem specifikována a je tak možné vytvářet v podstatě objektový model jehož vnitřní uspořádání sice může být na první pohled méně přehledné až chaotické ve srovnání s relační variantou, ale tento model je z hlediska struktury dat samopopisný. Záznamy relační databáze sice nabývají významu díky kontextu tabulky resp. jejích atributů, ale sémantické informace o vzájemném vztahu více záznamů různých tabulek už chybí. V datovém modelu jenž je tvořen systémem provázaných trojic jsou sice elementární informace uložené v jednotlivých tripletech, ty ale získávají další informační hodnotu díky kontextu s nimi provázaných trojic. V důsledku tohoto faktu tak každá neizolovaná elementární informace (trojice) v modelu obsahuje sémantickou informaci o vztahu s ostatními daty. Datový model tak ve výsledku obsahuje jak cíleně uložená data (ve smyslu relační databáze) tak i jejich vzájemný kontext, který navíc vytváří z modelu organizovaný systém.

## 5.1 Dynamičnost modelu

Z výše uvedených informací je patrné, že datový model sémantického webu je oproti relačnímu snáze rozšiřitelný a je tak vhodnější pro uchování dat jejichž struktura je v sémantickém smyslu více heterogenní. Bude-li například potřeba vytvořit databází většího množství osob, které se dále člení na určité skupiny jež se vyájemně liší sledovanými parametry na úrovni skupin i jednotlivců je v principu možné řešit problém dvěma způsoby. Vytvořením relačního modelu je možné kategorizovat osoby prostřednictvím tabulek respektive jejich atributů, které ale bude nutné dále rozšiřovat s rostoucím počtem nových odlišných skupin a měnit tak do jisté míry pevnou strukturu dat. V případě řešení prostřednictvím sémantického webu je možné přidávat specifické informace pouze vybraným skupinám či jednotlivcům, aniž by se muselo zasahovat do struktury ostatních dat. Je tak možné organizovat jednotlivé objekty (osoby) do různých i prolínajících se skupin a ty popisovat hromadně nezávisle na ostatních, což je myšlenka dědičnosti.

## 5.2 Dědičnost a hierarchie dat

Jazyk RDFS zavádí do datového modelu sémantického webu základní organizační hierarchii. Uložené objekty je možné kategorizovat do jednotlivých tříd stejně jako vlastnosti a ty pak vzájemně hierarchicky organizovat. Jednotlivé objekty tak můžou dědit od svých předků (díky konstrukcím *rdfs:range* a *rdfs:domain*) určité vlastnosti a na své úrovni je sami dále rozšiřovat. Celý tento princip je možné popsat na jednoduchém příkladu znázorněném na obrázku 5.1. Třída (*rdfs:Class*) *Person* jež definuje dvě vlastnosti (*rdf:Property*) *name* a *surname* má dva potomky (*RDFS:SubclassOf*), jež tyto vlastnosti dědí. Potomek *Experimenter* dále rozšiřuje své vlastnosti o atribut *Email* a potomek *Subject* definuje atribut *Age*. Informace, že experimentátor *Pavel* (jenž je instancí třídy *Experimtnator*) provádí měření na objektu *Martin* je zachycena predikátem (*rdf:Property*) *Measure*. Inverzní činnost „je měřen“ popisuje vlastnost *IsMeasured*, přičemž definice inverzního vztahu dovoluje vyjádřit jazyk OWL.



Obrázek 5.1: Znázornění hierarchické organizace dat

### 5.3 Problémy modelu

Jak již bylo zmíněno, tato podoba modelu je výhodná spíše v případech velké různorodosti uchovávaných dat, pro homogenní a z hlediska rozšiřování statická data je stále výhodné řešení využívající tabulek. Sémantický web však slouží především k uchovávání metadat, původní data jež by mohla mít podobu větších binárních souborů by bylo obtížné ukládat tímto způsobem. Technologie a standardy sémantického webu definují datové typy prostřednictvím schématu jazyka XML, jenž sice připouští také binární typy např. base64Binary, ale pravděpodobně není vhodné je využít pro uložení většího binárního souboru do tohoto datového modelu z důvodu výkonnosti a sémantické využitelnosti v rámci modelu [34].

### 5.4 Porovnání modelů

Pro jednoduché porovnání datového modelu sémantického webu a relačního datového modelu poslouží níže uvedená tabulka 5.1, jež srovnává výhody (+)

a nevýhody (-) obou modelů.

	Relační model	Model sémantického webu
Dědičnost	+	-
Dynamická rozšířitelnost modelu	+	-
Typování meziobjektových vazeb	+	-
Unifikace identifikátorů objektů	+	-
Hierarchická organizace dat	+	-
Sémantika datového modelu	+	-
Ukládání rozsáhlých datových objektů	-	+
Uchování (sémanticky) homogenních dat	-	+
Počet dostupných nástrojů pro uchování persistentních dat	-	+

Tabulka 5.1: Srovnání výhod a nevýhod datových modelů

Z výše uvedené tabulky je patrné, že vhodnost použití vybraného modelu je odvislá od typu a struktury dat, jež by měl datový model uchovávat. Jednotlivé řádky tak stručně shrnují vhodnost modelu pro vybranou vlastnost uchovávaných dat. Poslední položkou je počet dostupných nástrojů umožňujících uchování persistentních dat, kterou je míňeno množství softwarových řešení databázových úložišť umožňujících uchování dat ve vybrané, modelu přirozené podobě. Pro relační databáze je takových nástrojů poměrně široká řada, ale pro úložiště dat sémantického webu je výběr o poznání menší, navíc mnoho z těchto dostupných nástrojů je spíše experimentálního charakteru. Jejich popisu a výběru se věnuje následující kapitola.

# 6 Nástroje pro vývoj sémantického webu

Jak již bylo zmíněno v kapitole Sémantický web, jazyk RDF je jedním ze základních stavebních prvků technologie sémantického webu. Současná podoba tohoto jazyka vychází ze specifikace W3C [28] z února 2004. Za dobu uběhlou od publikace zmíněné specifikace do současnosti (2013) vznikla poměrně široká sada nástrojů od jednoduchých utilit až po komplexní softwarová řešení jež umožňují zpracovávat data v jazyce RDF resp. OWL. Rozsáhlý a vcelku přehledný seznam těchto nástrojů (k datu vzniku této práce celkem 183) dostupný ze zdroje [2] poslouží jako primární seznam pro výběr vhodného nástroje. Pro účely této práce se zaměříme na hledání nástroje jež slouží primárně jako persistentní úložiště RDF dat a zároveň poskytuje veřejné Jena/Sesame API pro komunikaci a umožňuje dotazování jazykem SPARQL nad daty. Dalsí nutnou podmínkou je, aby nástroj poskytoval hybridní úložiště umožňující uchovávat jak relační data tak data v jazyce sémantického webu (zmíněné požadavky vyplývají z následující kapitoly). Tyto poměrně náročné podmínky ovšem splňují pouze dva nalezené nástroje, jejichž popis bude následovat.

## 6.1 Virtuoso

Virtuoso je hybridní databázový server umožňující práci nad více datovými modely. Nástroj poskytuje klasickou relační databázi, RDF databázi trojic, XML databázi a také dokumentově orientovanou databázi. Verze aplikace se zpoplatněnou licencí poskytuje oproti GPL licencované variantě navíc virtuální databázi a také replikaci dat. Virtuální databáze umožňuje zastřešit více databázových systémů jediným centrálním se kterým klient komunikuje. Ve výsledku tak umožňuje klientovi pracovat s více databázemi, které se mu transparentně jeví jako jediná. Díky tomu je například možné prostřednictvím jediného dotazu získat výsledek ze všech databází najednou. Replikace dat slouží k zajištění konzistence dat takovým způsobem, kdy při např. dvou databázích slouží hlavní databáze k zachycení aktuálních změn dat, které postupně zasílá vedlejší databázi jež obsahuje původní data. Díky tomu obsahuje vedlejší databáze stále konzistentní data neboť při nezdařeném zápisu aktualizace může zažádat hlavní databázi o jejich opětovné zaslání.

Databázový server poskytuje grafické webové rozhraní, které slouží ke komplexní administraci a také je možné jeho prostřednictvím zadávat libovolné SQL/SPARQL dotazy. Virtuoso implementuje nad relační databází engine poskytující funkcionality v podobě standardu SQL-92, podporu pro pohledy, standardní datové typy, procedury, kurzory a také transakce. SPARQL dotazy je možné zadávat stejnými prostředky jako SQL dotazy, či je dokonce vnořovat do SQL dotazů. Díky tomu je možné pracovat s relačním modelem dat a RDF modelem zároveň. Samotné dotazování a další komunikace klientské aplikace se serverem může být realizováno prostřednictvím driverů ODBC/JDBC nebo Java knihovnou poskytující Jena API. Neméně důležitou součástí podpory Java aplikací je i distribuce knihoven pro Hibernate (Hibernate driver a Hibernate dialect). Virtuoso také podporuje import souborů s RDF daty ve formátu N3, Turtle a také nejběžnější RDF/XML [15].

## 6.2 Oracle 11g - Oracle Spatial

Oracle Spatial je rozšířením databázového systému Oracle, jež je primárně určeno pro zpracování geoprostorových dat [17]. Toto rozšíření s sebou ovšem přináší také podporu pro sémantické technologie, která je klíčová pro tuto práci. Standardní instalace Oracle 11g release 2 (současná databáze EEG/ERP portálu), ani žádná z předchozích verzí tuto podporu standardně po instalaci neposkytuje. Její zapnutí je ale pouze záležitostí spuštění instalačního skriptu a následného vygenerování tabulkového prostoru pro systémové tabulky a všechna následně ukládaná data. V tomto tabulkové prostoru je již možné vytvořit sémantickou síť, která ponese všechny systémové RDF tabulky a pohledy, jež jsou nezbytné pro provádění SPARQL dotazů. Při práci s databází je zapotřebí mít k dispozici datový model, který ponese všechny RDF trojice, jež budou vytvářeny. Tento model bude vázán na tabulku, která ponese všechny vytvářené trojice. Do tohoto modelu je již možné zapsat první trojici [16]. Pro názornost jednoduchosti tohoto procesu bude uvede výše popsaný postup v podobě SQL příkazů [16].

1. `?/md/admin/catsem11i.sql`
2. `CREATE TABLESPACE rdf_tblspace DATAFILE  
'/oradata/orcl/rdf_tblspace.dat' SIZE 1024M AUTOEXTEND  
ON NEXT 256M MAXSIZE 16384M;`
3. `EXECUTE SEM_APIS.CREATE_SEM_NETWORK('rdf_tblspace');`
4. `CREATE TABLE family_rdf_data (id NUMBER,`

```
    triple SDO_RDF_TRIPLE_S);  
5. execute SEM_APIS.create_sem_model('family',  
'family_rdf_data', 'triple');
```

První příkaz instaluje podporu sémantických modelů v databázi. Druhým příkazem je na disku vytvořen soubor, který poneše tabulkový prostor pro sémantickou síť a všechny datové modely. Jeho počáteční velikost je nastavena na 1 GB s možností dalšího zvětšení v případě vyčerpání dané kapacity. V tomto tabulkovém prostoru je ve třetím kroku vytvořena sémantická síť, ve které je možné vytvářet jednotlivé datové modely. Ve čtvrtém kroku je vytvořena první tabulka se dvěma sloupci, kde se vyskytuje nový datový typ *SDO\_RDF\_TRIPLE*. Jedná se o datový typ trojice ve smyslu RDF, který v sobě obsahuje subjekt, predikát a objekt (dle očekávaného standardu). Posledním příkazem byl vytvořen datový model nad předchozí tabulkou, který poslouží pro ukládání konkrétních trojic. Samotné vkládání trojic je pak poměrně jednoduché, a je možné jej nalézt ve [16].

Po instalaci podpory sémantických technologií je již tedy možné využívat databázi Oracle jako úložiště RDF dat. Ve výsledku je tedy k dispozici hybridní databáze pro ukládání dat jak v relační podobě tak ve RDF jazyce. Nad zmíněnými sémantickými daty je možné se dotazovat jazykem SPARQL a to skrze stejné prostředky jako v případě jazyka SQL. V důsledku toho je také možné vytvářet vnořené či jinak kombinované dotazy z obou jazyků, stejně jako tomu bylo u nástroje Virtuoso. K sémantickým datům databáze je možné přistupovat samozřejmě stejnou cestou jako k relačním – tedy prostřednictvím ODBC/JDBC driveru, ale Oracle také dodává driver poskytující standardní Jena API pro Java aplikace. Databázový systém podporuje odvozovací pravidla jazyka RDF, RDFS i OWL a od verze Oracle 10g je navíc možné, aby uživatel definoval vlastní odvozovací pravidla. Při vytváření dotazu je pak možné určit jakou množinu odvozovacích pravidel (RDF / RDFS / vlastní pravidla či jejich kombinace) chceme do dotazu zahrnout a ovlivnit tím tak jeho výsledek [16].

## 6.3 Jena

Jena je Java framework určený pro vývoj aplikací sémantického webu. Nástroj je vyvíjený od roku 2000 v podobě open source projektu, který je nyní dostupný pod Apache licencí. Framework Jena je distribuovaný jako knihovna jež poskytuje [1]:

- API pro zápis a čtení RDF dat ve formátu N3, RDF/XML či N-triples
- API pro práci s ontologiemi na úrovni jazyků RDFS a OWL
- Reasoner pro zpracování dat jazyků RDF a OWL
- Dotazovací engine SPARQL
- Ukládání RDF dat do databáze

Jena poskytuje komplexní API umožňující práci s daty sémantického webu. Veškerá zpracovávaná data jež mají podobu grafu jsou uchovávána prostřednictvím modelu. Framework poskytuje 2 druhy modelů, jež se liší svou expresivitou. Základní model umožňující zpracování dat na úrovni jazyka RDF je vhodný pro data s nízkou úrovní sémantiky. Druhý, ontologický model umožňuje maximální využítí nástroje, neboť poskytuje API pro práci s ontologiemi jazyka OWL. Samotný model je možné uchovávat buď pouze v paměti, přičemž je možné jeho data ukládát i opětovně načítat ze souboru, nebo v databázi, která poskytuje příslušný konektor umožňující využítí Jeny jako API k tomuto úložišti. S daty modelu je možné pracovat objektovým způsobem na úrovni trojic či jejich částí nebo prostřednictvím vykonávání SPARQL dotazů [1].

# 7 Transformace datového modelu

## 7.1 Rozdelení datového modelu

V předchozích kapitolách jež se zabývaly datovým modelem sémantického webu a relačním datovým modelem bylo možné získat základní představu o obou možnostech reprezentace dat. Rovněž byly nastíněny některé problémy současného relačního datového modelu portálu, jehož úprava je předmětem této práce. Z výše uvedené tabulky 5.2 je patrné že technologie sémantického webu využité pro reprezentaci datového modelu mají i nevýhody, a to zejména v oblasti ukládání rozsáhlých datových objektů, jež je primárním důvodem pro částečné zachování relačního modelu. V rámci této práce tedy bude navržen datový model EEG/ERP portálu, který bude z jedné části založený na relační databázi a z druhé části na objektové orientované databázi využívající technologie sémantického webu, bude se tedy jedna o hybridní datový model.

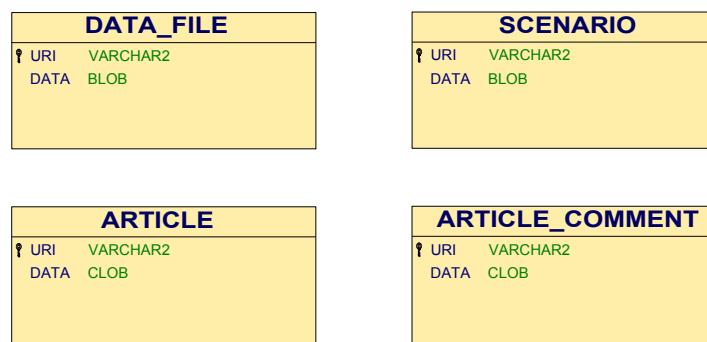
Pro rozdelení datového modelu je nutné určit pravidla, podle kterých bude rozdelení provedeno. K tomu bude využito koncepce sémantického webu, který je určený zejména pro ukládání a zpracování metadat. V současném datovém modelu je tedy nutné odlišit základní data od metadat. Následující rozdelení zároveň umožní vyřešit problém ukládání rozsáhlých datových souborů, neboť právě ty budou označeny jako základní data. Na EEG/ERP portál je možné pohlížet jako na aplikaci poskytující prostředky pro ukládání a zpracování experimentů a zároveň umožňující diskusní činnost uživatelů aplikace. Každý prováděný experiment je z pohledu databáze charakterizován dvěma základními objekty – binárním souborem zaznamenávajícím naměřené hodnoty a souborem scénáře, který určuje postup měření. Diskusní funkcionality portálu umožňuje uživatelům vytvářet textové příspěvky v podobě článků jež mají charakter rozsáhlejšího textového objektu a k těmto článkům mohou další uživatelé přidávat komentáře stejného datového typu. Popis této funkcionality portálu tak v důsledku umožňuje identifikaci základních dat jejichž úložištěm bude relační databáze. V ERA modelu portálu, jež je uveden v příloze D této práce, jsou zmíněné údaje uchovány v tabulkách v podobě rozsáhlých binárních/textových souborů. V relačním modelu tedy ponecháme uchovaná tato data:

- Datové soubory se záznamem experimentu

- Datové sobory se scénářem popisujícím postup experimentu
- Text článků publikovaných uživateli portálu
- Text komentářů publikovaných článků

## 7.2 Relační část modelu

Relační část datového modelu bude tvořena čtyřmi tabulkami z nichž každá poneše příslušná data dle výše uvedeného seznamu. Z důvodu jednoduchosti budou názvy tabulek převzaty z původního modelu dle původních tabulek nesoucí tato data. Aby bylo možné jednotlivé tabulkové záznamy jednoznačně identifikovat bude do tabulky přidán sloupec nesoucí tento údaj. Pro zachování koncepce sémantického webu ve výsledném modelu však nebude tímto identifikátorem pouze postupně inkrementované číslo jako v původním modelu, nýbrž se zavede identifikace prostřednictvím URI. Tímto způsobem dosáhneme jednotné identifikace objektů v celém výsledném datovém modelu, přičemž záznamy relační databáze jednoznačně označené prostřednictvím URI budou dále popisovány v datovém modelu sémantického webu, jež bude sloužit k uchování metadat. Ve výsledku tak zmíněná identifikace údajů prostřednictvím URI zajistí propojení obou částí cílového datového modelu. ERA model relační části nově vytvářeného modelu na obrázku 7.1 je pak poměrně jednoduchý ve srovnání s původní variantou díky absenci uchování metadat.



Obrázek 7.1: ERA model relační části transformovaného modelu

Následující vytvářená sémantická část modelu bude provádět transformaci původního relačního modelu, z jehož tabulek nesoucích zmíněné datové soubohy jsou ty atributy odstraněny (z důvodu jejich přesunu do této nové relační části modelu).

## 7.3 Sémantická část

Zbylá část dat původního modelu jež nebude obsažena ve relační části nového modelu je předmětem následující transformace. K jejímu provedení je potřeba nadefinovat obecná pravidla a postup, který by umožňoval transformaci obecnějšího modelu z důvodu jeho častých změn. I v průběhu tvorby této práce je původní model nadále modifikován a je tak zapotřebí mít k dispozici postup, který by umožnil vytvoření i úpravu nového hybridního modelu tak, aby pokrýval stejnou množinu dat jako původní model v později modifikované verzi. Postup vytvoření nového modelu bude sestávat ze dvou části a to z transformace původních dat do podoby sémantického webu a z dodání sémantiky tomuto modelu, tak aby mohl být do jisté míry samopopisný. Pro popis resp. zápis této části modelu budou využity jazyky RDF spolu se RDFS, které dovolují vyjádřit veškerou sémantiku nalezenou v aktuálním modelu.

### 7.3.1 Transformace tabulek

Při transformaci tabulek původního relačního modelu bude využito obdobného principu jako v případě objektově relačního mapování nástroje Hibernate [9]. Prvním krokem je identifikace původních tabulek, které neslouží pro rozložení vazby M:N mezi jinými tabulkami. Tyto tabulky budou v nově vznikajícím modelu představovat třídy a záznamy v nich obsažené budou reprezentovány instancemi těchto tříd. To je možné díky jazyku RDFS, který dovoluje vytvářet jednoduchý objektový model složený z tříd a jejich instance podobně jako např. v jazyce Java. Jednotlivé atributy tabulek pak budou realizovány jako predikáty (*rdf:Property*). Transformace struktury původní tabulky prostřednictvím jazyka RDF resp. RDFS se zajistí jazykovými konstrukcemi tak, jak je uvedeno v tabulce 7.1.

Každá takto transformovaná tabulka bude reprezentována jedním URI identifikátorem (stejně jako další objekty) a označením, že se jedná o třídu zajistí konstrukce *rdfs:Class*. Původní název tabulky bude součástí hodnoty

URI. Každý záznam tabulky bude rovněž popsán prostřednictvím URI a jeho náležitost ke třídě (je instancí třídy) popíše trojice, kde subjektem bude URI této instance, predikátem konstrukce *rdf:type* a objektem URI cílové třídy. Jednotlivé atributy tabulky se díky konstrukci *rdf:Property* stanou predikáty ve vytvářeném modelu a umožní tak vytvářet typovanou vazbu mezi instancemi a hodnotami jejich atributů (literálů) či mezi instancemi navzájem.

Tabulka	<i>rdfs:Class</i>
Záznam tabulky	<i>rdf:type</i>
Atribut tabulky	<i>rdf:Property</i>

Tabulka 7.1: Transformace struktur tabulky do sémantického webu

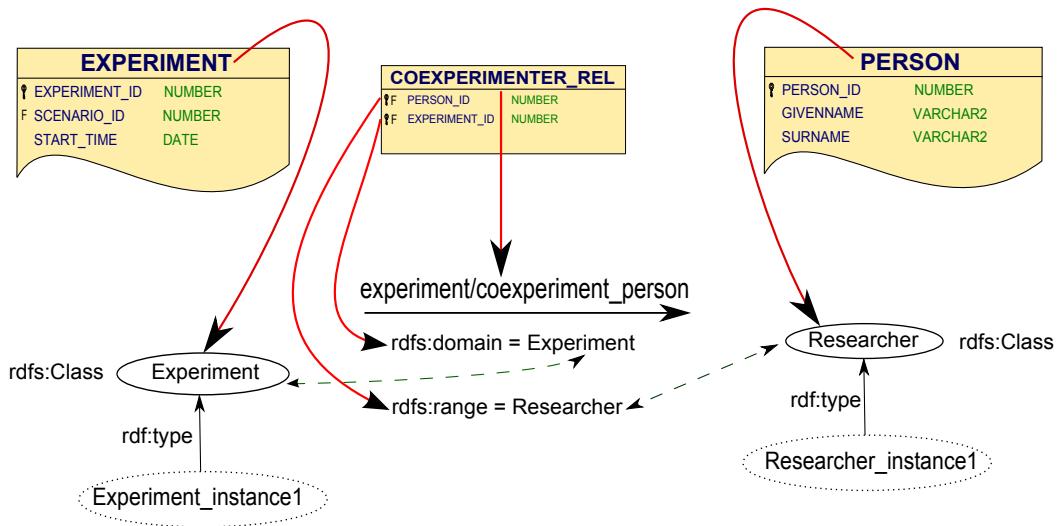
Transformované atributy (predikáty) je zapotřebí navíc svázat se třídami původních tabulek těchto atributů. Toho se docílí díky konstrukci *rdfs:domain*, která tak de facto určuje doménu těchto predikátů. Tento výraz ale především slouží k definování náležitosti instance ke své třídě, čehož se docílí vytvořením trojice jejímž predikátem bude transformovaný atribut a subjektem zmíněná instance.

### 7.3.2 Transformace rozkladových tabulek

Původní datový model je z velké části tvořen tabulkami jež rozkládají násobnou vazbu M:N jiných tabulek. Tyto tabulky nebudou transformovány do podoby tříd, nýbrž se pouze využijí jejich atributy, které budou přiřazeny k jedné z provázaných tabulek resp. tříd. Sémantický web připouští přiřazení více různých hodnot objektů subjektům pro stejný predikát. Je tedy možné například instanci třídy *Weather* přiřadit pro predikát *Research\_group* více cílových skupin. Ve výsledku však bude transformován pouze jeden atribut rozkladové tabulky na predikát, jež bude reprezentovat vazbu mezi první z původních provázaných tabulek prostřednictvím konstrukce *rdfs:range* a druhou tabulkou prostřednictvím domény predikátu (*rdfs:domain*). Výraz *rdfs:range* přiřazený k vybranému predikátu umožňuje vyjadřit náležitost instance ke rodičovské třídě (definované hodnotou *rdfs:range*) pokud je tato instance objektem trojice se zmíněným predikátem. V rámci této transformace bude využito *rdfs:range* zároveň k vyjádření cílového objektu pro predikát, stejně jako bylo využito domény pro určení zdrojového subjektu daného

predikátu. Rozhodnutí která z vázaných tabulek určí predikátu hodnotu *rdfs:domain* či *rdfs:range* je odvislá od sémantického významu této vazby jež bude propojovat výsledné instance tříd a určí tak směr vazby (v případě reprezentace modelu orientovaným grafem se bude určovat směr šipky daného predikátu – viz predikát *Measure* na obrázku 5.1).

Pro názornost tohoto postupu bude uvedena ukázka transformace původní tabulky *COEXPERIMENTER\_REL* nesoucí atributy *EXPERIMENT\_ID* a *PERSON\_ID*, tak jak ukazuje obrázek 7.2.



Obrázek 7.2: Znázornění transformace rozkladové tabulky

Tato tabulka umožňuje zaznamenat násobnou vazbu mezi záznamy tabulek *EXPERIMENT* a *PERSON* vyjadřující účast více výzkumníků podílejících se na provádění experimentu. Výsledkem aplikace výše uvedeného postupu bude vzniklý predikát jehož doména (*rdfs:domain*) bude mít hodnotu *Experiment* (třída) a *rdfs:range* hodnotu *Researcher*. Hodnotu URI tohoto predikátu lze zvolit např. *experiment/coexperiment\_person*.

### 7.3.3 Transformace atributů tabulek

Transformace atributů tabulek již byla popsána v předchozích odstavcích, ze kterých je zřejmý postup vzniku výsledného predikátu. K již uvedenému

postupu určení domény predikátu dle tabulky původního modelu je nutné ještě uvést postup určení rozsahu (*rdfs:range*) cílových objektů predikátem vázaných trojic. V případě atributů rozkladových tabulek je tento cílový rozsah určen právě jedním z těchto atributů. Rozsah (*rdfs:range*) transformovaného atributu u nerozkladové tabulky je určen doménou (ve smyslu relační databáze) tohoto atributu. Pokud je tato (relační) doména původního atributu určena cizím klíčem jiné tabulky s níž má atribut zajistit provázání, pak tato cílová tabulka (transformovaná na třídu) určuje i rozsah predikátu. V opačném případě kdy relační doména je určena pouze datovým typem omezujícím přípustné hodnoty atributu bude rozsah určen rovněž datovým typem, který ale bude definován prostřednictvím XSD datových typů. Určení toho typu je dán tabulkou XX uvedenou v následující podkapitole.

### 7.3.4 Transformace datových typů

Pro určení datových typů rozsahů (*rdfs:range*) predikátů a literálů transformovaných z hodnot atributů záznamů tabulek slouží následující tabulka 7.2. Ta popisuje transformaci datových typů relační databáze na základní datové typy sémantického webu definované XSD schématem.

Relační databáze	Sémantický web
<i>VARCHAR2</i>	<i>xsd:string</i>
<i>CHAR</i>	<i>xsd:string</i>
<i>NUMBER</i>	<i>xsd:integer</i>
<i>LONG</i>	<i>xsd:long</i>
<i>FLOAT</i>	<i>xsd:float</i>
<i>DATE</i>	<i>xsd:date</i>
<i>TIMESTAMP</i>	<i>xsd:dateTime</i>

Tabulka 7.2: Transformace datových typů relační databáze

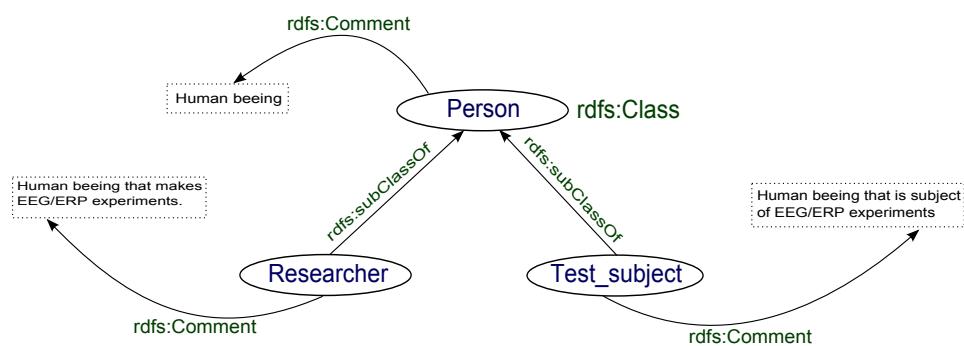
Názvy datových typů relační databáze vycházejí z názvů užívaných portálovou databází Oracle.

### 7.3.5 Transformace záznamů tabulek

Záznamy tabulek původního modelu se transformují na instance tříd (*rdf:type*) vzniklých z těchto tabulek. Pro každý záznam se tak vytvoří nové URI jež bude sloužit k jeho identifikaci a především jako subjekt trojic přiřazujících této instanci prostřednictvím vytvořených predikátů hodnoty objektů. Hodnoty těchto objektů závisí na hodnotě atributu původního záznamu. Bude-li původní hodnota atributu cizím klíčem jiné tabulky pak přiřazovaným objektem bude instance třídy vzniklá ze záznamu určeného tímto cizím klíčem. V opačném případě bude na základě původní hodnoty atributu vytvořen nový literál nesoucí tuto hodnotu, jehož datový typ bude odpovídat transformovanému datovému typu (dle tabulky 7.2) původní domény atributu (původní datový typ atributu).

### 7.3.6 Dodání sémantiky modelu

Nově vytvořenému modelu bude dodána sémantika prostřednictvím hierarchického uspořádání všech transformací získaných tříd a predikátů ke kterým budou navíc dodány popisné informace. Hierarchického uspořádání je možné dosáhnout využitím konstrukcí *rdfs:subClassOf* a *rdfs:subPropertyOf*. Z původního relačního modelu tvořeného množinou rovnocenných tabulek tak vzniká objektový model, který bude mít částečně stromovou strukturu. Jednoduchým způsobem je tedy dosaženo například oddělení zaznamenávaných osob jež provádějí experimenty od těch co jsou měřeným subjektem, tak jak znázorňuje obrázek 7.3.



Obrázek 7.3: Znázornění dodání sémantiky modelu

Ve výsledném modelu byla dále vytvořena rodičovská třída pro všechny třídy uchovávající údaje o měřících elektrodách či pro metadata popisující scénář měření. Kompletní navržená struktura tříd modelu v XML zápisu je zobrazena v příloze C. Význam tohoto zápisu je popsán v následující podkapitole Implementace transformace modelu.

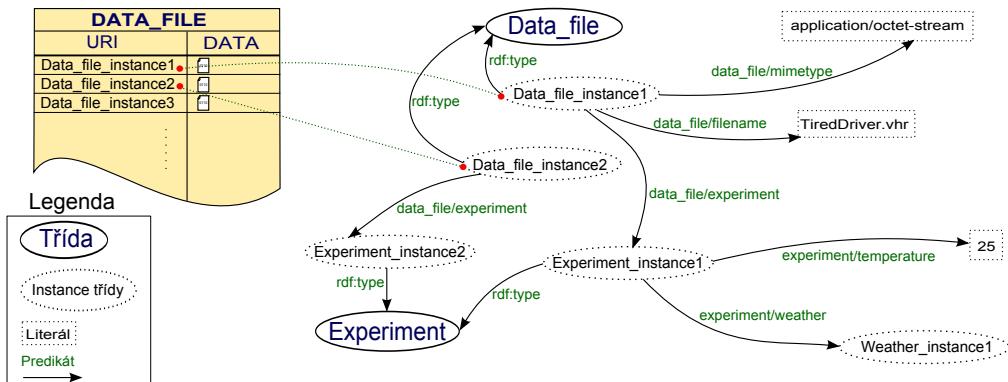
Každé vytvořené třídě a predikátu (property) je navíc dodávána prostřednictvím konstrukce *rdfs:comment* popisná informace (viz obrázek 7.3) umožňující uživateli získat představu o významu daných struktur modelu, které jsou jinak popsány pouze prostřednictvím URI (byť se sebelepě zvolenou hodnotou). V důsledku tak získáváme do určité míry samopopisný datový model.

### 7.3.7 Řešení rozšiřujících tabulek

Původní relační model obsahuje několik tabulek nesoucích ve svém názvu řetězec *PARAM\_DEF* a *PARAM\_VAL* jež slouží k uložení dalších rozšiřujících hodnot tabulek ke kterým jsou vázané. Do popisované transformace nebudou tyto tabulky zahrnuty nebot' by obsahem svých záznamu porušovaly návrh výsledného modelu. Funkce kterou tyto tabulky v původnímu modelu plní pouze pro vázané tabulky ale poskytuje výsledný transformovaný model všem uloženým datům, díky své koncepci. Toho je dosaženo prostřednictvím možnosti vytváření nových predikátů (property) modelu, které umožňují dodávat další nová metadata libovolným subjektům aniž by došlo k porušení původní struktury modelu. V případě požadavku vícenásobného využití nově definovaných metadat a jejich hodnot je rovněž možné rozšířit model o další třídy a její instance čímž lze dále jednoduše dynamicky datový model rozširovat. Názornou ukázkou ověřující toto tvrzení je testovací aplikace popisovaná v následující kapitole testování.

### 7.3.8 Ukázka transformace

Níže uvedený obrázek 7.4 graficky znázorňuje výsledek provedené transformace dle popisovaných pravidel. Vzhledem k velikosti datového modelu je zde zobrazena pouze jeho část, přičemž celý model zapsaný prostřednictvím XML souboru i jeho grafická vizualizace jsou obsažené na přiloženém CD.



Obrázek 7.4: Znázornění částí výsledného modelu

### 7.3.9 Výhody navrženého modelu

- Zápis významu objektů je přímou součástí modelu
- Jednoduché anotování dat bez změny struktury modelu
- Dynamicky rozšiřitelný model
- Možnost libovolných meziobjektových vazeb
- Typování meziobjektových vazeb

### 7.3.10 Nevýhody navrženého modelu

- Datový model rozdělen do dvou částí
- Ztráta restriktivních omezení z původního relačního modelu
- Možná ztráta přehlednosti modelu neuváženým rozšiřováním

Navržený model poskytuje poměrně širokou řadu výhod, díky které je možné ukládat data způsobem bližším lidskému myšlení. Zásadní nevýhodou modelu je ztráta restriktivních omezení, neboť není možné vynutit na úrovni modelu povinné údaje jako je např. u osob výzkumníků uživatelské jméno či heslo, bez kterých nemohou provést přihlášení do systému a dále jej využívat. Souvisejícím problémem je ztráta možnosti omezení počtu vazeb mezi instancemi tříd, není tak možné například omezit vazbu M:N na 1:1.

Všechny tyto záležitosti je nutné řešit v datové vrstvě aplikace jež s modelem pracuje. V rámci tohoto hodnocení navrženého modelu se vybízí také otázka jeho výkonnosti, která bude řešena v kapitole testování.

# 8 Implementace navrženého modelu

## 8.1 Formální zápis modelu

V předchozí kapitole byla navržena struktura nového datového modelu portálu, která je tvořena relační částí a částí sémantického webu. Aby bylo možné požadovaný model prakticky využít a otestovat, je nutné zajistit jeho konstrukci v databázovém systému. Pro účely této práce bude model vytvořen v databázových systémech Oracle a Virtuoso, které poskytují prostředky pro hybridní datové úložiště. Vzhledem k tomu, že datový model portálu je neustále modifikován z důvodu implementace nových funkcionalit je vhodné, aby bylo možné model vytvářet co nejvíce automaticky a umožnit tak co nejjednodušejí reprezentovat změny původního modelu do transformovaného.

Pro zápis navrženého modelu byl zvolen XML formát a z důvodu přehlednosti, snadné uživatelské modifikace, jednoduchého zápisu hierarchických struktur a následného aplikačního zpracování. Pro usnadnění práce uživatele je k dispozici také XSD šablona jež usnadňuje vytváření dokumentu a jeho případnou validaci. XML dokument tedy slouží k zápisu všech modelem požadovaných struktur, čímž jsou tabulky, třídy a predikáty (property).

Dokument musí začínat definicí tabulek, které jsou specifikovány svým jménem a typem dat jež budou ukládat (binární nebo textové soubory). Název tabulky je zvolen shodně s lokálním jménem URI identifikátoru třídy jejíž instance jsou reprezentovány ve zmíněné tabulce prostřednictvím URI. Touto volbou názvu se později v testovací aplikaci docílí propojení tabulek s třídami. Element definující tabulku má název *data\_table* a jeho atributy *name* a *type* slouží ke záznamu jména tabulky typu ukládaných dat. Pro názornost je níže uvedena definice všech 4 tabulek relační části transformovaného modelu.

```
<data_table name="article" type="text"/>
<data_table name="article_comment" type="text"/>
<data_table name="data_file" type="binary"/>
<data_table name="scenario" type="binary"/>
```

Po definici tabulek modelu je dále nutné uvést všechny požadované třídy. Ty jsou určeny svými názvy (URI) a popiskem definujícím ontologický význam

třídy (viz obrázek 7.3). Element definující třídu má název *class* a atributy *name* a *description* slouží zápisu jména třídy a jejího popisku. Pro názornost je opět níže uvedena ukázka definice tříd *Experiment* a *Article*.

```
<class name="article" description="Published article."/>
<class name="experiment" description="EEG/ERP experiment."/>
```

Po definici všech tříd je možné zavést jejich nepovinný hierarchický systém. Ten se provádí výčtem potomků vybrané třídy a je možné jej libovolně řetězit. Klíčovým elementem označujícím rodičovskou třídu je *parent-class*, jehož tag ohraničuje potomky definované elementem *child* s jediným parametrem *name* pro název třídy. Definice potomků třídy *Person* je uvedena níže.

```
<parent_class name="person">
    <child name="person/researcher"/>
    <child name="person/test_subject"/>
</parent_class>
```

Dalším částí dokumentu je definice predikátů (property). Element pro tuto definici má název *property* a jeho atributy *name*, *range*, *domain* a *description* postupně slouží k určení jeho jména, rozsahu (*rdfs:range*), domény (*rdfs:domain*) a sémantického popisku. Ukázka použití je uvedena níže.

```
<property
    name="person/researcher/given_name"
    range="xsd:string"
    domain="person/researcher"
    description="Given name."/>

<property
    name="person/researcher/group_member"
    range="research_group"
    domain="person/researcher"
    description="Group membership."/>
```

Poslední části dokumentu je definice hierarchie predikátů, jejíž praktické využití je v této práci spíše okrajové, jak je uvedeno v kapitole testování.

Definice hierarchického systému je obdobná jako v případě tříd. Rodičovský predikát je uvozen elementem s názvem *parent\_property* s atributem *name* nesoucím jméno predikátu. Vnořené elementy děděných predikátu jsou opět definovány názvem *child* s atributem *name* pro jméno potomka. Ukázka zápisu dědičnosti predikátů je uvedena níže.

```
<parent_property name="article_property">
    <child name="article/author"/>
    <child name="article/research_group"/>
    <child name="article/title"/>
    <child name="article/time"/>
    <child name="article/subscription"/>
</parent_property>
```

Všechny uváděné názvy tříd a predikátů resp. jejich rozsahů a domén jsou (a do cílového dokumentu se tak zapisují) jen lokálními částmi názvu URI, jeho prefix se zadává pouze jednou při zápisu modelu z důvodu jednoduchosti a přehlednosti. Výsledný dokument s navrženým modelem je spolu s XSD šablonou součástí obsahu přiloženého CD.

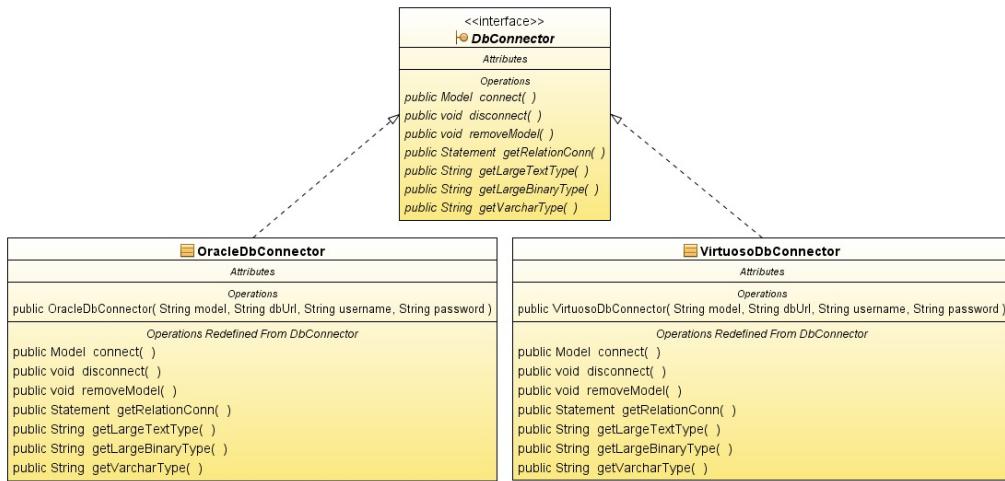
## 8.2 Konstrukce modelu

Pro navržený model zapsaný prostřednictvím XML dokumentu je nutné vytvořit aplikace, které jej zapíší do databáze a následně poskytnou API, které umožní s modelem pracovat. Tyto aplikace budou vyvíjeny stejně jako EEG/ERP portál v jazyce Java.

### 8.2.1 Knihovna SemWebModelDbConnector

Pro vytvoření modelu v databázovém systému je nejprve nutné zajistit komunikaci, jež dovolí provádět potřebné operace jak s relační částí, tak i se částí sémantického webu této databáze. K tomuto účelu je navržena knihovna, zastřešující tyto operace prostřednictvím metod interface, jež umožňuje transparentně komunikovat s vybranou databází bez závislosti, zda-li se jedná o databázový systém Oracle či Virtuoso. Tato knihovna poskytuje metody pro připojení a odpojení od systému a zajišťuje aplikaci jež ji využívá přístup

k vybranému modelu sémantického webu spolu s připojením k relačnímu modelu. Na obrázku 8.1 je zobrazen UML diagram této jednoduché knihovny jež je dále využita pro testovací aplikaci a nástroj pro konstrukci modelu. Veškeré implementační detaily jsou patrné z komentovaných zdrojových kódů jež jsou součástí přiloženého CD.



Obrázek 8.1: UML diagram knihovny SemWebModelDbConnector

### 8.2.2 Vytvoření modelu

Za účelem vytvoření cílového navrženého modelu byla implementována aplikace `SemWebModelCreator`. Ta zajišťuje načítání vstupního XML souboru s modelem a zpřístupňuje tato data aplikaci. Vstupní data jsou načítána ve třídě `DataLoader` prostřednictvím SAX parseru [7], který na základě získaných hodnot atributů pro dané elementy vytváří datové objekty zastupující požadované tabulky, třídy a predikáty modelu. K jejich obalení slouží třídy balíku `cz.zcu.kiv.eeg.semweb.model.creator.data`. Na základě načtených dat provádí třída `ModelCreator` zápis, který sestává z vytvoření obou částí modelů. Nástroj v prvním kroku navazuje prostřednictvím knihovny `SemWebModelDbConnector` spojení s databází, díky čemuž získává přístup k modelu sémantického webu. Pro zápis tříd a predikátů modelu je využito popisovaného nástroje Jena, jenž poskytuje API umožňující ontologický přístup k modelu na úrovni jazyka OWL. Dle načtených dat jsou tak postupně vytvářeny požadované třídy, kterým jsou nastaveny popisné sémantické anotace, a zároveň jsou

zařazeny do hierarchického systému prostým nastavením rodičovských tříd či potomků. Predikáty modelu jsou zapisovány obdobným způsobem, přičemž navíc je každému predikátu nastaven definovaný rozsah (*rdfs:range*) a doména (*rdfs:domain*).

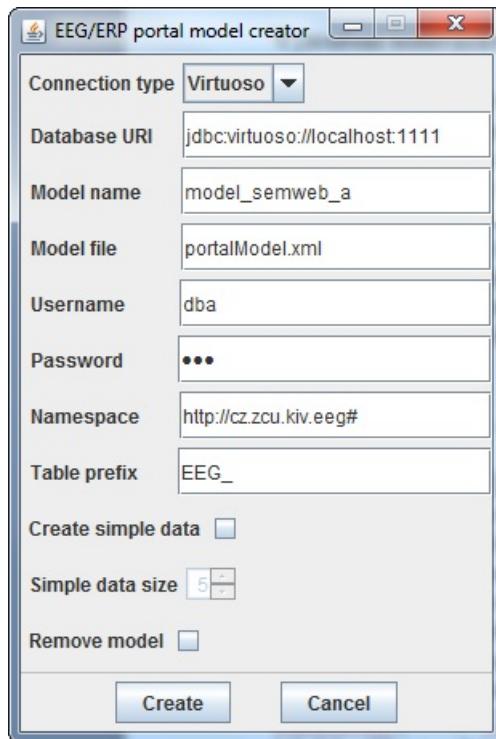
Po dokončení zápisu modelu sémantického webu je přistoupeno k vytváření tabulek relační databáze, což je umožněno díky spojení poskytnutým knihovnou SemWebModelDbConnector při jeho navazování v kroku před zápisem sémantického modelu. Vytváření tabulek je prováděno pouhým vykonáváním SQL příkazů *CREATE TABLE* pro všechny požadovaná tabulky lišící se pouze svým názvem a datovým typem atributu *DATA* (jež slouží k uchování rozsáhlých datových souborů).

### 8.2.3 Vytvoření testovacích dat

Aby bylo možné vytvořený model otestovat, je vhodné zajistit i generování testovacích dat. Generování těchto dat zajišťuje třída *SimpleDataCreator*, která vytváří pseudonáhodná data v podobě instancí modelem definovaných tříd. Každé instanci jsou navíc přiřazeny hodnoty pro všechny predikáty jež mají doménu shodnou s rodičovskou třídou zmíněné instance. Rozsah generovaných dat je možné specifikovat prostřednictvím celého čísla, které určuje počet generovaných instancí každé třídy.

Pro snadné použití tohoto nástroje má aplikace implementované grafické uživatelské prostředí využitím Java knihovny Swing. Po spuštění aplikace se zobrazí okno dle obrázku 8.2, jež slouží ke specifikaci databázové připojení a nastavení parametrů modelu.

Položka *Connection Type* slouží pro výběr cílové databáze modelu (Oracle nebo Virtuoso) jejíž URI je zadáváno v poli *Database URI*. Položka *Model name* určuje název vytvářeného modelu, jehož popisný XML soubor je zadáván v poli *Model file*. Nezbytnou položkou pro spojení s databází jsou uživatelské jméno a heslo v položkách *Username* a *Password*. Pole *Namespace* slouží pro definici URI prefixu všech vytvářených entit modelu, neboť XML soubor popisující model obsahuje pouze lokální části hodnot URI identifikátorů. Poslední pole *Table prefix* určuje prefix názvu všech vytvářených tabulek, čímž je později možné v rámci celé relační databáze jednoduše identifikovat tabulky tohoto modelu. Zaškrťávací políčko *Create simple data* slouží ke rozhodnutí, zda generovat testovací data, jejichž velikost určuje položka *Simple data size*. Poslední nastavitelná položka je políčko *Remove model*,



Obrázek 8.2: Okno nástroje pro vytváření modelu

jehož zaškrtnutím bude specifikovaný model smazán namísto jeho vytvoření. Jednoduchý uživatelský manuál pro použití tohoto nástroje je součástí přílohy A. Veškeré implementační detaily tohoto nástroje jsou zřejmě z komentovaných zdrojových kódů.

### 8.3 Využití modelu

V rámci zápisu navrženého modelu do hybridní databáze bylo využito API frameworku Jena, který umožňuje jednoduchou práci s konstrukcemi sémantického webu na úrovni jazyka OWL. Pro další praktické využití vzniklého modelu portálu je ovšem toto API příliš jednoduché, neboť neposkytuje metody pro zpracování komplexnějších operací. Z tohoto důvody vznikl požadavek na vytvoření knihovny, která by sloužila jako nadstavba tohoto frameworku a poskytovala API umožňující snáze provádět typické operace datové vrstvy EEG/ERP portálu.

Implementovaná knihovna SemWebModelApi jež byla vytvořena v rámci této práce poskytuje k cílovému modelu API podobným způsobem jako Jena a umožňuje jednoduchou práci s těmito daty, což je demonstrováno testovací aplikací popsanou v následující kapitole. Operace nad modelem dostupné prostřednictvím API navržené knihovny budou kategoricky rozdělené a popsány v následujících odstavcích.

### 8.3.1 Operace čtení

Knihovna SemWebModelApi poskytuje přístup k datovému modelu prostřednictvím metod třídy *PortalModel*. Operace čtení realizované implementovanými metodami umožňují práci se třídami, jejich instancemi a s predikáty. Pro získání tříd modelu slouží metody *listParentClasses* a *listSubClasses*, které umožňují získat URI identifikátory jednotlivých tříd a vytvořit na jejich základě hierarchický strom. Ten může později sloužit jako vstupní bod modelu, který bude poskytovat základní navigaci pro přístup k dalším objektům, kterými jsou především instance tříd, dle konvencí Jeny označované jako „*Individual*“. Pro přístup k těmto instancím slouží především metoda *listClassInstances*, která vrací potomky třídy předané prostřednictvím jejího URI. Tato metoda vrací list instance třídy *Item*, sloužící jako obalující prvek objektů modelu. Tato abstraktní třída má dva potomky *LiteralItem* obalující literály modelu a *UriItem*, jež umožňuje nést instanci třídy, případě predikát. Každá instance třídy *UriItem* nese URI obalovaného objektu a umožňuje svými metodami získávat další prvky vázané na tento objekt. To je klíčová vlastnost knihovny, která umožňuje s modelem pracovat v souladu s principy sémantického webu, neboť je možné pro vybranou instanci třídy získat její predikáty vázané objekty, které jsou opět předávané jako instance třídy *UriItem* (pokud není cílovým objektem literál). V důsledku je tak možné prostřednictvím jediného subjektu procházet postupně model a získávat požadované informace včetně koncových cílových literálů. Ty jsou reprezentovány instancemi třídy *LiteralItem*, která nese datový typ a příslušnou hodnotu tohoto literálu.

Třída *UriItem* disponuje metodami umožňujícími získání seznamu predikátů vázaných k instanci třídy, díky čemuž je možné pro každý objekt získat informace o jeho dostupných vlastnostech a pouze vybranou vlastnost dále zkoumat bez načítání nepotřebných dat což je možné díky nesené vazbě na model v každé instanci *UriItem*. Tento způsob umožňuje získávání vybraných informací o dané instanci třídy. Získání těchto instancí pro vybranou třídu

je, jak již bylo zmíněno, možné voláním metody *listClassInstances* třídy *PortalModel*. Aby nebylo nutné při hledání konkrétní instance provádět průchod celým listem, je možné nastavit prostřednictvím parametru metody filtr, který bude provádět restrikci položek navráceného listu.

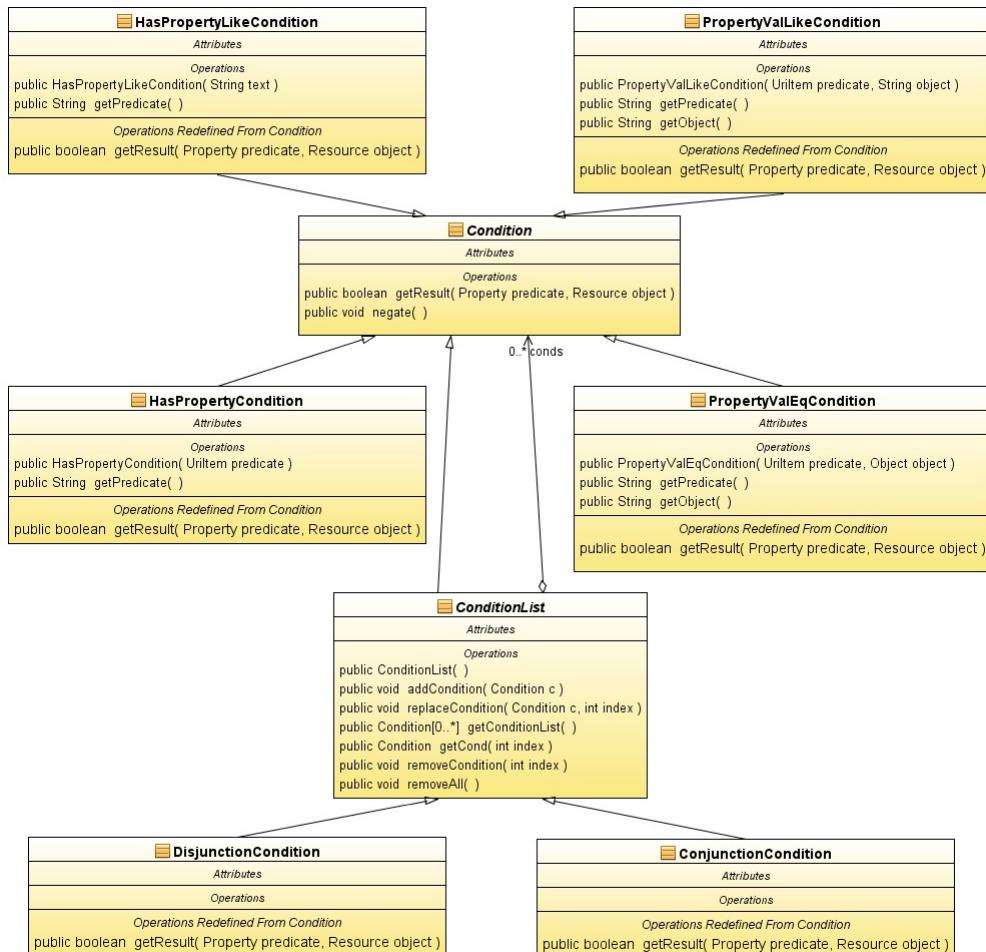
### 8.3.2 Filtrování

Pro nastavení výběru navracených instancí třídy slouží systém podmínek definující vlastnosti požadovaných objektů. Každý vytvářený filtr je tvořen množinou podmínek v logickém součtu, přičemž každá tato podmínka je tvořena další množinou podmínek v logickém součinu. Ve výsledků je tak možné nastavit libovolnou kombinaci nutných a postačujících podmínek filtrování. Tato navržená koncepce je patrná z UML diagramu tříd filtrování na obrázku 8.3.

Z tohoto obrázku je patrné, že díky rodičovské třídě *Condition* všech ostatních tříd je možné vytvářet i poměrně složitý systém podmínek, neboť každá podmínka může být tvořena množinou (listem) podmínek v logickém součtu či součinu. Výsledná koncová podmínka může být 4 typů, které jsou určená následujícími třídami.

- **HasPropertyCondition** - umožňuje nastavit URI predikátu, na který musí mít dotazovaný subjekt vázaný objekt (jakýkoliv), např. podmínka že dotazovaná osoba musí mít nastavenou hodnotu emailové adresy, přičemž tato hodnota může být libovolná
- **HasPropertyLikeCondition** - umožňuje nastavit textový řetězec, který musí být obsažen v URI predikátu svazující tento subjekt s jiným objektem, např. podmínka pro řetězec „name“, že dotazovaná osoba musí mít nastavenou hodnotu nějakého jména, lhostejno zda *givenname* či *surname*
- **PropertyValEqCondition** - umožňuje nastavit textový řetězec, jehož hodnoty musí nabývat objekt vázaný nastaveným URI predikátu, např. podmínka pro osobu, jejíž příjmení musí nabývat hodnoty *Novák*
- **PropertyValLikeCondition** - umožňuje nastavit textový řetězec, jehož hodnota musí být obsažena v textové reprezentaci objektu vázaného nastaveným URI predikátu. např. podmínka pro osobu, jejíž přímení

musí obsahovat řetězec *Novák*, odpovídající objektům příjmení *Novák* i *Nováková*



Obrázek 8.3: UML diagram tříd filtrování

### 8.3.3 Operace zápisu

Zápis nových, resp. aktualizace stávajících dat modelu je možné realizovat opět prostřednictvím metod třídy *PortalModel*. Touto cestou je lze vytvářet nové třídy (samostatně či jako potomky stávajících tříd), jejich instance a predikáty (opět s možností hierarchického usporádání). Aktualizace stávajících

hodnot objektů predikátem vázaných ke instancím tříd resp. přidávání nových hodnot je možné realizovat přímo prostřednictvím metod třídy *UriItem*, která danou instanci třídy nese. Obdobně lze aktualizovat metodou *updateValue* hodnoty existujících literálů ve třídě *LiteralItem*, které jsou vytvářeny jako nové hodnoty pro dané vlastnosti (predikáty) instancím tříd prostřednictvím metody *addPropertyValue* třídy *UriItem*.

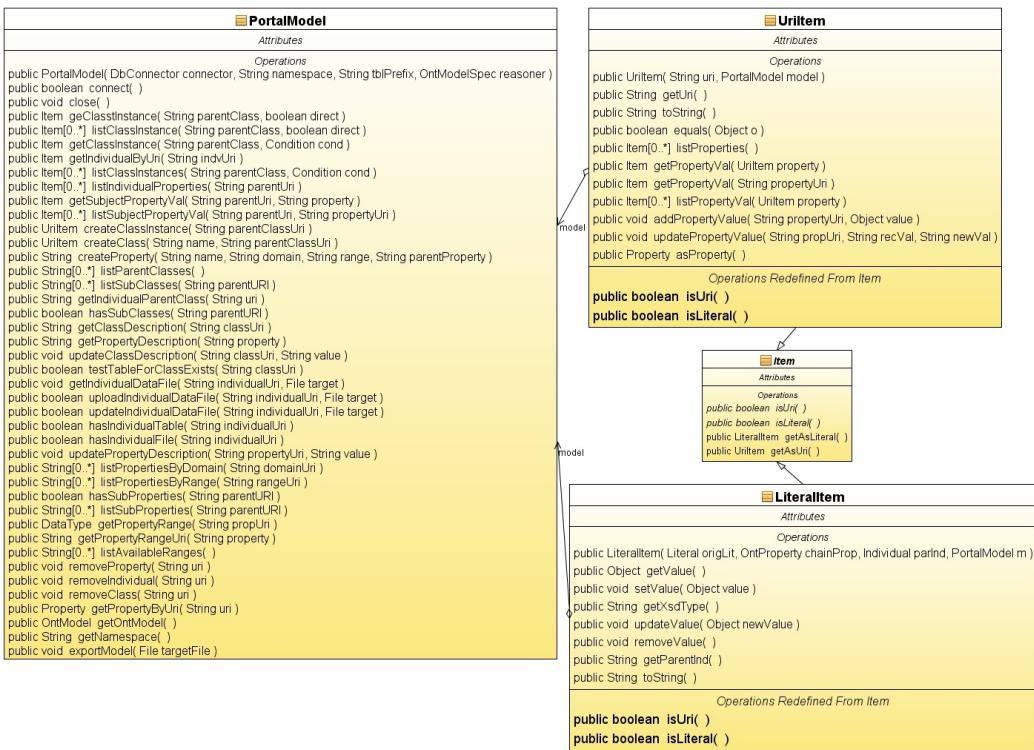
### 8.3.4 Operace mazání

Požadavky na mazání dat modelu lze opět realizovat prostřednictvím metod třídy *PortalModel*, jež umožňuje odstranění stávajících tříd, jejich instancí a predikátů. Odstraněním stávajícího predikátu dojde ke odstranění všech trojic modelu jež jej obsahují. V důsledku tak všechny instance třídy jež měly nějaký objekt vázaný tímto predikátem o tento údaj přijdou. Odstranění koncových objektů vázaných existujícím predikátem ke instanci třídy je možné realizovat metodou *removeValue* třídy *LiteralItem* (jedná-li se o literál) nebo přímým odstraněním instance třídy metodou *removeIndividual* třídy *PortalModel* (jedná-li se o instanci třídy). V tomto případě nedojde k odstranění definovaného predikátu z modelu. Poslední mazatelným prvkem modelu jsou třídy, jejichž odstranění zajišťuje metoda *removeClasss* třídy *PortalModel*. Jejím voláním nad stávající třídou dojde ke odstranění této třídy včetně všech jejích instancí potažmo i trojic vázaných na tyto instance. V důsledku tak odstraněním třídy dojde ke odstranění všech vazeb mezi instancemi nedotčených tříd na mazané instance tříd, neboť jejich odstraněním pozbývají tyto vazby smyslu.

Všechny výše popsané operace jsou patrné z UML diagramu tříd na obrázku 8.4. Detailnější představu o podporovaných operacích potažmo implementovaných metodách je možné získat z Javadoc dokumentace obsahující podrobný popis všech metod, jejichž názvy jsou významové.

### 8.3.5 Přístup k relačním datům

Pro sjednocený přístup k datům relační části modelu je využito shody názvů (bez definovaného prefixu) relačních tabulek s lokální částí názvu URI příslušných tříd modelu. Název navržené tabulky modelu bez prefixu *EEG* má hodnotu *DATA\_FILE* stejně jako lokální název třídy *data\_file* (nerozlišujeme-li velikost písmen) bez namespace prefixu, čímž je vytvořena pomyslná vazba



Obrázek 8.4: UML diagram tříd veřejného API knihovny SemWebModelApi

mezi záznamy tabulky a instance třídy `data_file`. Metodami `hasIndividualTable` a `hasIndividualFile` třídy `PortalModel` je pak při dodržení zmíněných konvencí možné zjistit, zda k instancím určité třídy existuje relační tabulka resp. soubory vázáné na konkrétní instance. Tyto soubory je pak možné nahrávat a aktualizovat voláním metod `uploadIndividualDataFile` a `updateIndividualDataFile` třídy `PortalModel`. Názorná ukázka tohoto přístupu je realizována testovací aplikací popsanou v následující kapitole.

# 9 Testování

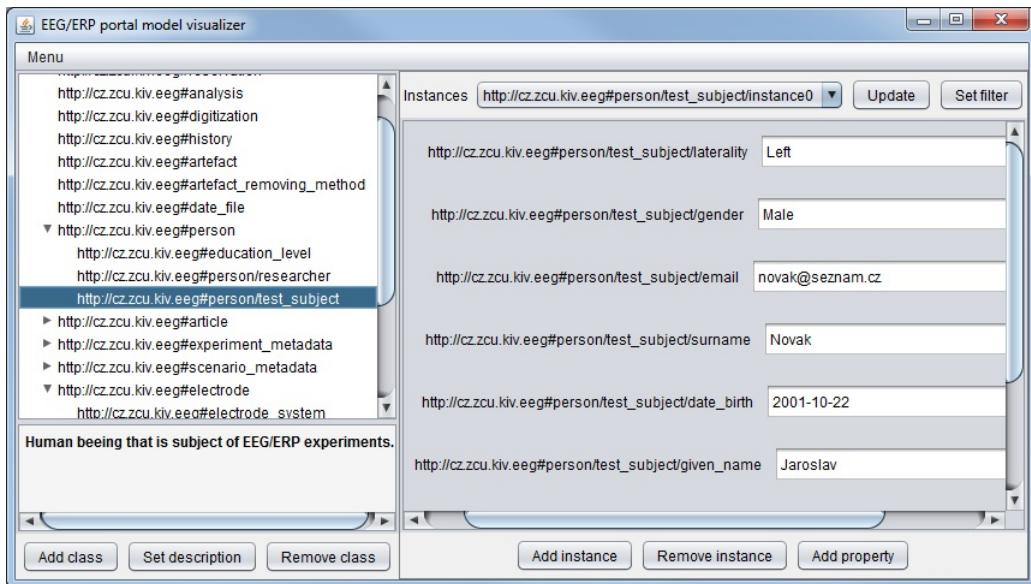
Testování navrženého transformovaného datového modelu a knihovny SemWebModelApi poskytující API pro operace modelu bude provedeno po stránce funkční i výkonnostní. Touto cestou by měla být ověřena použitelnost navrženého modelu jako alternativní náhrady současného datového modelu EEG/ERP portálu.

## 9.1 Funkční testování

Pro účely funkčního testování byla vytvořena aplikace s grafickým uživatelským rozhraním, která slouží ke vizualizaci datového modelu k němuž přistupuje metodami knihovny SemWebModelApi. Tato testovací aplikace umožňuje zobrazení a procházení objektů datového modelu a zároveň dovoluje uživateli provádět nad modelem všechny operace popsané v kapitole 8.3 poměrně jednoduchým způsobem. Cílem této aplikace ovšem není vytvoření uživatelsky přívětivého nástroje pro práci s modelem, ale pouze zjištění komfortnějšího způsobu testování modelu spolu s API vytvořené knihovny jež jsou výstupem této práce.

Testovací aplikace je implementována s využitím Java knihovny Swing, jež zajišťuje vytvoření celého grafického rozhraní. Po jejím spuštění se zobrazí obdobné okno jako v případě aplikace pro tvorbu modelu, jež slouží k zadání údajů pro spojení s databází. Aplikace nabízí na výběr připojení k databázi Oracle nebo Virtuoso, k čemuž je nutné vyplnit URI cestu k databázi, název (sémantického) modelu a uživatelské jméno a heslo pro přihlášení. Dalšími nutnými údaji jsou jmenný prostor modelu (*namespace*) a tabulkový prefix. Pro všechny zmíněné údaje je nutné zadat stejné hodnoty jako v případě aplikace pro tvorbu modelu (je-li požadavkem připojení k tomuto modelu). Poslední volitelnou položkou je *Reasoner type*, která slouží ke volbě použitého reasoneru. Aplikace nabízí tzv. mělký (*Shallow*) a hluboký (*Deep*) reasoner, jež se liší hloubkou procházení modelu při jednotlivých dotazech, což se logicky odráží na rychlosti testovací aplikace.

Po úspěšném navázání spojení s databází je zobrazeno okno testovací aplikace dle obrázku 9.1.



Obrázek 9.1: Hlavní okno testovací aplikace

Hlavní menu aplikace poskytuje možnost exportování načteného modelu do souboru. Tuto funkci poskytuje framework Jena, který zajišťuje zápis ontologického modelu ve syntaxi RDF/XML. Levý panel aplikace je zobrazuje hierarchický strom tříd modelu. Jeho vizualizaci zajišťuje komponenta *JTree* pro tvorbu stromů, jejíž první úroveň listů tvoří URI všech tříd bez definovaných rodičů získaný metodou *listParentClasses* třídy *PortalModel* knihovny SemWebModelApi. Pro každou třídu jsou postupně získaní její potomci, kteří jsou následně ve stromu hierarchicky zobrazeni. Výběrem konkrétního listu jež reprezentuje danou třídu je díky registrovanému posluchači stromu zobrazen ve spodním panelu editovatelný popis této třídy (viz atribut *description* elementů tříd XML souboru modelu). Prostřednictvím tlačítka spodních tlačítka levého panelu je možné vybranou třídu stromu odstranit, což je realizováno voláním dříve popisované metody *removeClass* třídy *PortalModel*. Přidání nové třídy je možné provést v příslušném dialogu otevíraném stiskem tlačítka *Add class*. Tomuto dialogu je předáno URI aktuálně vybrané třídy stromu, čímž je možné nově vytvářenou třídu nastavit jako potomka vybrané třídy.

Pravá část hlavního panelu aplikace ve své horní části obsahuje komponentu umožňující listování a výběr instancí třídy jež je aktuálně vybraná ve stromu tříd v levém panelu okna programu. Nad tímto listem instancí je

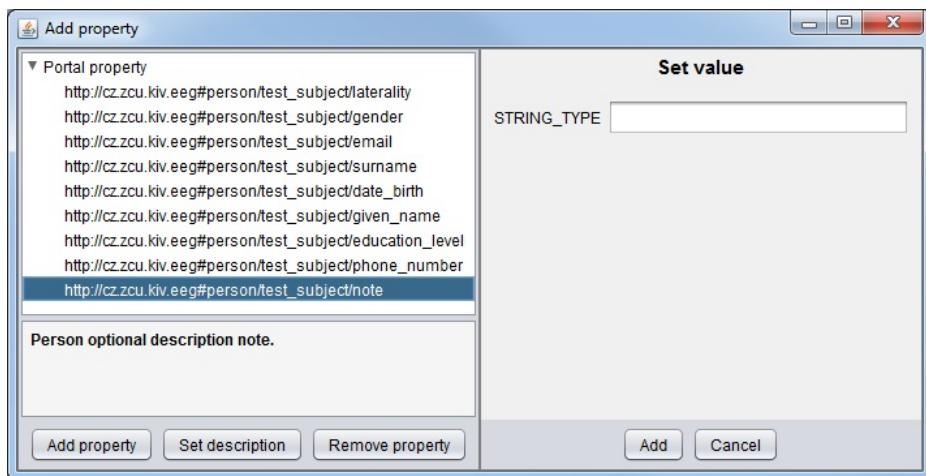
možné nastavit filtr tlačítkem *Set filter*, které otevírá dialog obsahující list podmínek v součinovém tvaru. Podmínky je možné přidávat, odstraňovat či editovat stávající podmínky. Přidáním nové nebo editací stávající podmínky se zobrazí dialog s listem podmínek v součtovém tvaru. Nad tímto listem je možné provádět stejné operace jako v případě podmínek v součinovém tvaru. Editací či přidáním podmínky listu podmínek v součtovém tvaru se zobrazí dialog, který slouží pro nastavení koncové podmínky. Na výběr jsou dispozici 4 druhů podmínek, tak jak je popsáno v kapitole 8.3.2.

Výběrem instance třídy z listovací komponenty horního pravého panelu testovací aplikace se v prostředním panelu zobrazí list všech URI predikátů a objektů svázaných s touto instancí. V případě že objekt je literálem, je zobrazena jeho textová reprezentace v editovatelném textovém poli jež umožňuje hodnoty aktualizovat či odstraňovat. Je-li objektem instance jiné třídy, pak je zobrazena komponenta umožňující výběr z instancí této třídy, kde vybranou položkou je ta, která je objektem. Tlačítkem *Go* nacházejícím se za touto komponentou je možné přejít ke vybrané instanci, jejíž predikáty s objekty se zobrazí namísto těch stávající instance.

V případě že vybraná třída má v relační části datového modelu odpovídající tabulku, pak je v listu predikátů a objektů vybrané instance zobrazeno na konci tlačítko, umožňující nahrání, aktualizaci či stažení příslušného souboru k této instanci. Instance tříd je možné stejně jako třídy přidávat a odstraňovat využitím příslušných tlačítek ve spodní části pravého panelu okna aplikace.

Pro přiřazení nových vlastností instanci třídy prostřednictvím predikátů a hodnot jimi vázaných objektů slouží tlačítko *Add property*, které otevírá nové dialogové okno dle obrázku 9.2.

Toto okno obsahuje v levé části strom predikátů, jejichž doménou je rodičovská třída aktuálně vybrané instance v hlavním okně. S predikáty lze provádět obdobné operace jako se třídami v hlavním okně. Výběrem predikátu ve stromu dojde v pravém panelu okna ke zobrazení komponenty sloužící ke nastavení objektu jež bude vybraným predikátem svázán ke instanci vybrané v hlavním okně aplikace. V případě že rozsahem (*rdfs:range*) vybraného predikátu je třída modelu, pak komponenta v pravém panelu okna umožňuje výběr z instancí této třídy. V opačném případě kdy rozsahem predikátu je primitivní datový typ (řetězec, číslo...), zobrazuje pravý panel komponentu umožňující zadat textovou podobu hodnoty, jež ponese nově vzniklý literál vytvořený jako objekt vázaný vybraným predikátem ke vybrané instanci třídy.



Obrázek 9.2: Okno pro nastavení predikátu modelu testovací aplikace

Tento mechanismus přidávání predikátů umožňuje vytvořit objekt a novým predikátem jej přiřadit k libovolné instanci třídy, čímž je možné této instanci jednoduše dodat požadovanou informaci, což v původním relačním modelu portálu realizovaly tabulky obsahující ve svém názvu řetězce *PARAM\_DEF* a *PARAM\_VAL*. Podrobný popis použití testovací aplikace je možné získat prostřednictvím uživatelského manuálu v příloze B.

### 9.1.1 Manuální testování aplikace

Vzhledem k tomu že výše popsaná testovací aplikace zpřístupňuje navržený model k čemuž využívá API vytvořené knihovny, je nutné ověřit správnost všech operací manuálním provedením níže popsaného testovacího scénáře. Tento scénář stručně popisuje seznam úkonů, které je nutné provést skrze grafické rozhraní testovací aplikace a následně ověřit správnost výsledku provedené operace oproti popsámu API knihovny SemWebModelApi.

- Vytvoření nové třídy bez rodičovské třídy
- Vytvoření nové třídy jako potomka existující třídy
- Vytvoření nové třídy se sémantickým popiskem
- Změna sémantického popisku stávající třídy

- Odstranění stávající třídy
- Vytvoření nové instance třídy
- Odstranění instance existující třídy
- Vytvoření filtru s podmínkou Has property
- Vytvoření filtru s podmínkou Has property like
- Vytvoření filtru s podmínkou Has property value
- Vytvoření filtru s podmínkou Has property value like
- Vytvoření filtru s více než dvěma podmínkami v součinovém tvaru
- Vytvoření filtru s více než třemi podmínkami ve součtovém tvaru
- Vytvoření filtru se dvěma podmínkami v součinovém tvaru, kde každá podmínka je tvořena dvěma podmínkami v součtovém tvaru
- Aktualizace hodnoty literálu vázaného predikátem na instanci třídy
- Odstranění literálu vázaného predikátem na instanci třídy
- Aktualizace hodnoty instance vázané predikátem na instanci jiné třídy
- Vytvoření nového predikátu s rozsahem datové třídy
- Vytvoření nového predikátu s rozsahem základního datového typu
- Odstranění existujícího predikátu
- Vytvoření nového predikátu se sémantickým popiskem
- Změna sémantického popisku stávajícího literálu
- Přidání nového objektu literálu vázaného predikátem k instanci třídy
- Přidání vazby stávajícím predikátem mezi dvěma instancemi odlišných tříd

Všechny kroky popsaného scénáře byly úspěšně provedeny, čímž byla ověřena správná funkčnost testovací aplikace.

## 9.2 Výkonnéstní testování

Aby bylo možné posoudit použitelnost navrženého modelu i z výkonnéstního hlediska, budou provedeny výkonnéstní testy modelu naplněného testovacími daty. Testování bude prováděno nad oběma databázovými systémy, čímž bude možné posoudit jejich vhodnost pro uchování dat navrženého modelu. Datový model bude postupně třikrát naplněn testovacími daty lišícími se svým objemem, čímž bude možné ověřit rychlost operací nad modelem v závislosti na jeho rozsahu. Testovací scénář bude pokrývat operace pro čtení, vytváření, aktualizaci a mazání dat jež budou reprezentovány typickými požadavky současné verze EEG/ERP portálu na relační databázi. Každý požadavek bude implementován jak do podoby dotazu sémantického webu tak i do varianty SQL dotazu nad relační databází. V důsledku tak bude možné získat přibližné porovnání rychlosti vykonávání typických portálových požadavků nad původním a nově navrženým datovým modelem.

### 9.2.1 Testovací data

Testovací data budou mít podobu pseudonáhodných hodnot, kterými budou v případě relační databáze zaplněny jednotlivé tabulky, v případě databáze sémantického modelu se bude jednat o instance definovaných tříd a literály vázané prostřednictvím predikátů na tyto instance. Objem dat obou variant modelu bude pro účely vzájemného porovnávání naměřených hodnot volen stejného rozsahu ve smyslu sémantiky. Tím je myšlen stav, kdy relační model bude obsahovat stejný počet experimentů, osob, scénářů, rezervací a dalších údajů jako sémantický datový model. Pro lepší představu tak bude uveden počet trojic sémantického modelu a k němu příslušný přibližný počet záznamů většiny tabulek relačního modelu. Například 100 záznamů/tabulka představuje 100 různých experimentů prováděných 100 různými osobami a pod. Testovací data budou zvolena v následujících třech variantách lišících se svým objemem.

- **Velikost A:** 60 000 trojic - 320 záznamu/tabulka
- **Velikost B:** 30 000 trojic - 160 záznamu/tabulka
- **Velikost C:** 10 000 trojic - 54 záznamu/tabulka

### 9.2.2 Testovací operace

Testovací operace jsou rozdělené do čtyř kategorií:

- operace vytváření **C**
- operace čtení **R**
- operace aktualizace **U**
- operace mazání **D**

Následující seznam obsahuje popis jednotlivých operací, jejichž implementace v podobě SQL dotazů resp. volání metod knihovny SemWebModelApi je patrná ze zdrojových kódů testovacích tříd.

- **C1 - Vytvoření nové osoby:** vytvoření nové osoby, které bude nastaveno uživatelské jméno, příjmení, datum narození, vzdělání a skupinové členství
- **C2 - Vytvoření nového experimentu:** přidání nového experimentu osobě a nastavení parametrů experimentu
- **C3 - Přidání člena výzkumné skupině:** vytvoření nového skupinového členství vybrané osobě
  
- **R1 - Přihlášení uživatele:** nalezení hesla osoby dle zadанého uživatelského jména
- **R2 - Získání experimentů uživatele :** nalezení experimentů uživatele dle uživatelského jména a získání jeho parametrů - teplota měření, čas počátku měření a příjmení testované osoby
- **R3 - Získání skupiny uživatele dle rezervace :** nalezení osoby jež vytvořila rezervaci a určení jména skupiny jejíž je členem
  
- **U1 - Aktualizace emailu osoby:** aktualizace položky email u osoby nalezené dle uživatelského jména

- **U2 - Aktualizace teploty měření experimentu:** nalezení experimentu uživatele a aktualizace položky teplota měření tohoto experimentu
  - **U3 - Aktualizace počasí měření experimentu:** vytvoření nového počasí a jeho nastavení položce počasí (změna původní hodnoty) experimentu osoby
- 
- **D1 - Smazání osoby:** odstranění vybrané osoby včetně všech jejích parametrů
  - **D2 - Smazání experimentů osoby:** odstranění všech experimentů vybrané osoby
  - **D3 - Smazání emailu osoby :** odstranění položky email u vybrané osoby

### 9.2.3 Postup testování

Měření bude provedeno pro databázové systémy Virtuoso a Oracle. Pro každou množinu testovacích dat dané velikosti bude spuštěna sekvence testovacích operací, kdy každá elementární operace bude provedena sekvenčně desetkrát po sobě a bude změřen celkový čas, který tato opakování operace trvala. Tato hodnota bude zaznamenána do tabulky. Například čas uvedený u položky C1 znamená dobu trvání potřebnou ke vytvoření 10 nových osob. Měření nad danou množinou dat bude provedeno nad sémantickým modelem pro oba typy reasonerů – mělký (*Shallow*), hluboký (*Deep*) a nad relačním modelem.

### 9.2.4 Testovací prostředí

Testování bude provedeno na počítači s procesorem Intel Core I3 M380 (4 jádra, 2,53 GHz), operační pamětí 8 GB a operačním systémem Widows 7 Professional 64-bit.

Pro testování nad databázovým systémem Virtuoso je využita lokální instalace Virtuoso Open-Source Edition 2012-08-02 na testovacím počítači. Pro testování nad databázovým systémem Oracle je využit vzdálený školní server Katedry informatiky a výpočetní techniky, na kterém běží současná testovací databáze portálu Oracle 11g.

### 9.2.5 Naměřené hodnoty

Význam záhlaví jednotlivých sloupců tabulek 9.1 a 9.2 je následující:

- **Shallow** Doba trvání operace nad navrženým modelem při použití mělkého reasoneru
- **Deep** Doba trvání operace nad navrženým modelem při použití hlubokého reasoneru
- **RDB** Doba trvání operace nad původním relačním modelem

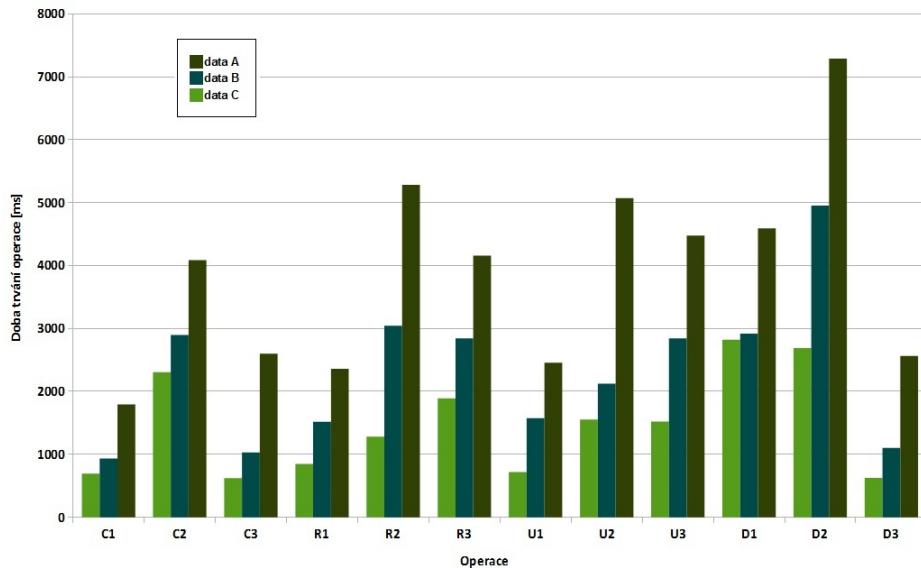
#### Databázový systém Virtuoso

Doba trvání operací [ms]									
	data velikost C			data velikost B			data velikost A		
	Shallow	Deep	RDB	Shallow	Deep	RDB	Shallow	Deep	RDB
<b>C1</b>	693	7 549	9	934	7 899	13	1 795	8 198	11
<b>C2</b>	2 304	24 185	27	2 897	24 538	29	4 085	26 974	41
<b>C3</b>	620	11 592	11	1 030	12 646	10	2 600	13 447	21
<b>R1</b>	846	3 100	8	1 516	3 900	10	2 385	4 810	9
<b>R2</b>	1 281	5 037	16	3 044	6 709	14	5 284	8 466	15
<b>R3</b>	1 891	4 543	8	2 842	6 684	9	4 158	8 980	12
<b>U1</b>	718	16 560	8	1 574	17 328	13	2458	19 156	16
<b>U2</b>	1 553	26 720	8	2 122	29 506	11	5 073	41 384	17
<b>U3</b>	1 522	35 756	16	2 843	36 860	16	4 477	45 142	18
<b>D1</b>	2 820	37 312	9	2 917	37 738	15	4 590	38 236	13
<b>D2</b>	2 688	48 400	8	4 953	48 464	6	7 289	54 062	7
<b>D3</b>	625	16 197	6	1 102	17 950	5	2 563	18 174	6

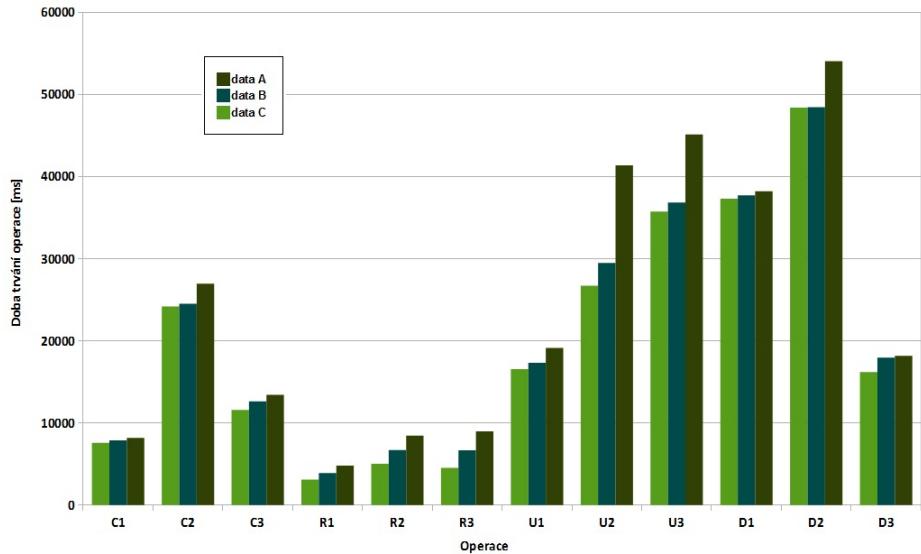
Tabulka 9.1: Naměřené doby trvání operací pro Virtuoso

Graf na obrázku 9.3 zobrazuje doby trvání jednotlivých operací nad navrženým modelem pro daná testovací data při použití mělkého reasoneru.

Graf na obrázku 9.4 zobrazuje doby trvání jednotlivých operací nad navrženým modelem pro daná testovací data při použití hlubokého reasoneru.



Obrázek 9.3: Doby trvání operací v závislosti na velikosti testovacích dat pro mělký reasoner nad databází Virtuoso



Obrázek 9.4: Doby trvání operací v závislosti na velikosti testovacích dat pro hluboký reasoner nad databází Virtuoso

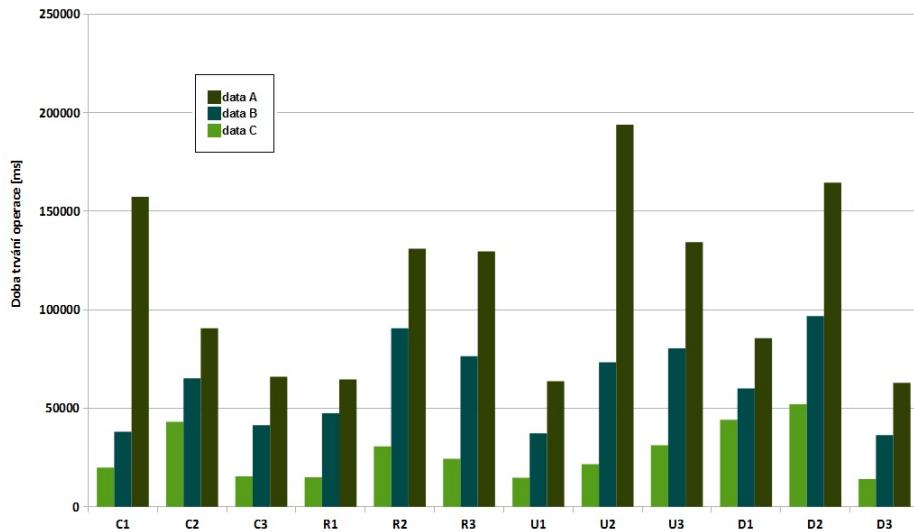
### Databázový systém Oracle

Doba trvání operací [ms]									
	data velikost C			data velikost B			data velikost A		
	Shallow	Deep	RDB	Shallow	Deep	RDB	Shallow	Deep	RDB
<b>R1</b>	14 981	94 164	145	47 480	116 530	130	64 650	139 273	129
<b>R2</b>	30 649	638 583	177	90 589	648 194	175	130 991	671 522	181
<b>R3</b>	24 345	32 804	109	76 412	76 860	121	129 702	136 253	117
<b>U1</b>	14 805	331 582	109	37 309	354 809	117	63 727	443 643	109
<b>U2</b>	21 645	571 614	130	73 350	645 015	124	193 903	695 671	124
<b>U3</b>	31 294	654 871	242	80 500	745 326	234	134 305	792 425	225
<b>C1</b>	19 963	97 377	112	38 187	100 027	99	157 325	114 963	96
<b>C2</b>	43 137	507 675	330	65 284	523 237	343	906 74	537 560	314
<b>C3</b>	15 520	262 620	116	41 400	302 746	125	66 029	309 472	108
<b>D1</b>	44 174	857 961	113	60 040	972 379	127	85 639	1 024 754	108
<b>D2</b>	52 071	1 230 946	126	96 824	1 253 632	123	164 563	1 445 638	121
<b>D3</b>	14 148	337 068	110	36 348	374 125	116	62 920	394 482	110

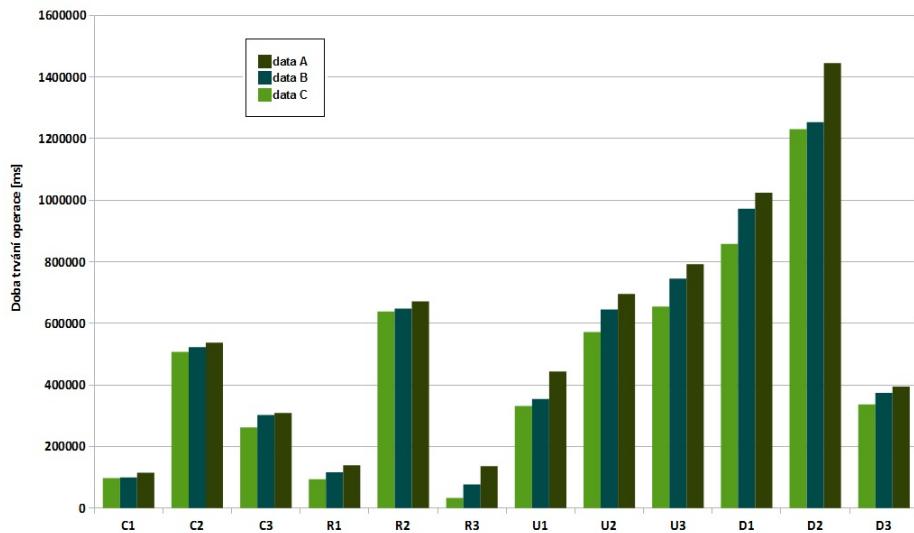
Tabulka 9.2: Naměřené doby trvání operací pro Oracle

Graf na obrázku 9.5 zobrazuje doby trvání jednotlivých operací nad navrženým modelem pro daná testovací data při použití mělkého reasoneru.

Graf na obrázku 9.6 zobrazuje doby trvání jednotlivých operací nad navrženým modelem pro daná testovací data při použití hlubokého reasoneru.



Obrázek 9.5: Doby trvání operací v závislosti na velikosti testovacích dat pro mělký reasoner nad databází Oracle



Obrázek 9.6: Doby trvání operací v závislosti na velikosti testovacích dat pro hluboký reasoner nad databází Oracle

### 9.2.6 Diskuze výsledků

Z tabulek naměřených hodnot časů trvání jednotlivých operací resp. dle grafů zobrazující závislosti těchto časů na velikosti datového modelu lze vyvodit následující závěry:

- Doba trvání operací nad relačním modelem je značně kratší než pro případ datového modelu sémantického webu
- Rychlosť operací při použití mělkého reasoneru je silně závislá na velikosti datového modelu
- Velikost datového modelu má poměrně malý vliv na rychlosť operací při použití hlubokého reasoneru
- Mělký reasoner je značně rychlejší oproti hlubokému reasoneru

Srovnáním dob trvání operací nad relačním a sémantickým modelem je patrné že relační model poskytuje o několik řádů rychlejší zpracování jak v případě databázového systému Virtuoso tak i Oracle. Pohledem na dobu trvání operací nad relačním modelem v závislosti na jeho velikosti lze usoudit, že vliv velikosti modelu (při zvoleném řádu rozsahu) je natolik malý, že překračuje možnost přesného měření dle zvoleného způsobu. V důsledku jsou tak hodnoty časů operací pro relační model v tabulkách 9.1 a 9.2 spíše orientační a slouží především pro porovnání řádů časů mezi modely. Z dalšího porovnání hodnot tabulek 9.1 a 9.2 je patrné že zpracování operací systémem Virtuoso je řádově rychlejší než pro Oracle. To je způsobeno zejména skutečností, že testovaný systém Oracle byl umístěn na vzdáleném školním serveru, címkž byla doba zpracování navýšena o síťovou komunikaci a navíc mohl být systém podstoupen dalšímu zatížení stran jiných aplikací např. testovací verze portálu.

Vliv použitého reasoneru je dle naměřených hodnot patrný už na testovacích datech nejmenšího objemu. Dalším poznatkem je skutečnost, že s růstem objemu dat se při použití hlubokého reasoneru prodlužuje doba zpracování operací jen minimálně. Ve výsledku se ovšem doba zpracování jedné operace při použití hlubokého reasoneru pohybuje v rozmezí hodnot 0,3 – 5,4 sekundy pro systém Virtuoso a 3,2 -144,5 sekundy pro systém Oracle, což tento reasoner činní prakticky nepoužitelným.

Při použití mělkého reasoneru narůstá doba potřebná ke zpracování operací úměrně s velikostí modelu a nejvyšší naměřené hodnoty časů jedné operace pro data velikosti A činí v průměru 0,7 sekundy pro systém Virtuoso a 19,3 sekundy Oracle. Mělký reasoner se tak při použití nad systémem Virtuoso jeví jako nejslibnější řešení přístupu k sémantickému modelu dat, nicméně praktické využití tohoto sémantického modelu pro aplikaci EEG/ERP portálu je z důvodu výkonnostní stránky poměrně diskutabilní.

# 10 Závěr

Cílem této práce bylo navržení nového datového modelu EEG/ERP portálů s využitím technologií sémantického webu. V průběhu práce byla provedena analýza současného datového modelu portálu, jejímž cílem bylo nalezení aktuálních nedostatků a problémů modelu. Popisované technologie a nástroje sémantického webu umožnily provést analýzu obecného datového modelu založeného na této technologii díky čemuž bylo následně realizováno porovnání obecného relačního modelu se sémantickým. Na základě provedené analýzy byl navržen nový datový model portálu, který využívá relační databázi k uchování objemnějších datových celků a zbylá data ukládá v objektové podobě v jazyce RDFS do nerelační databáze. Tato práce popisuje postup celé transformace původního modelu portálu (včetně dodání nové sémantiky) na úrovni obecných pravidel, která jsou doprovázena praktickými ukázkami realizace včetně koncové podoby výsledného modelu.

Pro usnadnění realizace transformace datového modelu byla vytvořena aplikace, která na základě popisu modelu prostřednictvím XML souboru umožňuje zapsat navržený datový model do databázových systémů Virtuoso a Oracle, kde mohou být dále prakticky využívány. V rámci práce byla dále vytvořena knihovna poskytující API, jež umožňuje jednoduchou cestou provádět nad sémantickým modelem operace běžně požadované aplikační vrstvou portálu.

Za účelem ověření nových možností a popisovaných výhod navrženého modelu byla dále vytvořena testovací aplikace, která prostřednictvím jednoduchého grafického rozhraní demonstruje možnosti uchování dodané sémantiky dat v modelu a také jednoduchost jeho dalšího dynamického rozširování.

Poslední část práce se zabývá výkonnostním testováním navrženého modelu, což je realizováno porovnáváním časů potřebných k vykonávání typických portálových operací nad datovým modelem. Ze získaných naměřených hodnot je patrná největší nevýhoda navrženého modelu což je značná výkonnostní ztráta oproti původnímu modelu. Pro praktické využití a tudíž i získání všech popisovaných výhod navrhovaného modelu portálu je tedy nutné zajistit dostatečný výkon pro databázový systém a také zvážit výkonnostní požadavky na cílovou aplikaci EEG/ERP portálu.

# Seznam zkratek a pojmu

**API** Application Programming Interface, rozhraní pro programování aplikací

**EEG** Electroencephalography, metoda vyšetření elektrické aktivity centrálního nervového systému

**ERP** Event Related Potential, vyšetření reakcí mozku na vnější podněty

**OWL** Web Ontology Language, ontologický jazyk sémantického webu

**RDF** Resource Description Framework, metodika modelování informací v sémantickém webu

**SPARQL** SPARQL Protocol and RDF Query Language, dotazovací jazyk nad RDF grafy

**URI** Uniform Resource Identifier, jednotný identifikátor zdroje informací

**W3C** World Wide Web Consortium, organizace vyvíjející webové standardy

**XML** Extensible Markup Language, značkovací jazyk vyvinutý W3C

**XSD** XML Schema Definition, XML schéma popisující strukturu XML dokumentu

**XSLT** Extensible Stylesheet Language Transformations, transformace sloužící ke převodu XML dat

**MVC** Model-View-Controller, návrhový vzor, architektura aplikace

**JSP** JavaServer Pages, technologie pro tvorbu dynamických webových stránek

# Literatura

- [1] *Jena – A Semantic Web Framework for Java*, 2010.  
<http://jena.apache.org/>, (přístup 15.4.2013).
- [2] *Semantic Web Tools*, <http://semanticweb.org/>, (přístup 19.2.2013).
- [3] James Hendler Dean Allenmang. *Semantic Web for the Working Ontologist*. Morgan Kaufmann, Burlington, 2007.  
ISBN 978-0-12-373556-0.
- [4] Gilfillan Ian. *Introduction to Relational Databases*,  
<http://www.databsejournal.com/sqletc/article.php/1469521/>  
Introduction-to-Relational-Databases.htm, (přístup 16.2.2013).
- [5] Kauler Jan. *Základní pojmy databáze*. České vysoké učení technické v Praze, [webzam.fbmi.cvut.cz/szabozol/ISZ/Kauler/506.doc](http://webzam.fbmi.cvut.cz/szabozol/ISZ/Kauler/506.doc), (přístup 17.2.2013).
- [6] Kauler Jan. *Relační datový model*. České vysoké učení technické v Praze, [webzam.fbmi.cvut.cz/szabozol/ISZ/Kauler/508.doc](http://webzam.fbmi.cvut.cz/szabozol/ISZ/Kauler/508.doc), (přístup 17.2.2013).
- [7] Kolena Jan. *EEG/ERP portal - reservation system for EEG/ERP laboratory*. ZČU, Plzeň, 2011.
- [8] Zendulká Jaroslav. *Relační model dat*. Vysoké učení technické v Brně, [http://www.fit.vutbr.cz/study/courses/DSI/public/pdf/nove/4\\_2.pdf](http://www.fit.vutbr.cz/study/courses/DSI/public/pdf/nove/4_2.pdf), (přístup 5.2.2013).
- [9] JBoss Community. *Hibernate*, 2010. <http://www.hibernate.org/>, (přístup 20.4.2013).
- [10] Kosek Jiří. *Jak pracují databáze na Webu*, <http://www.kosek.cz/clanky/iweb/12.html>, (přístup 15.2.2013).

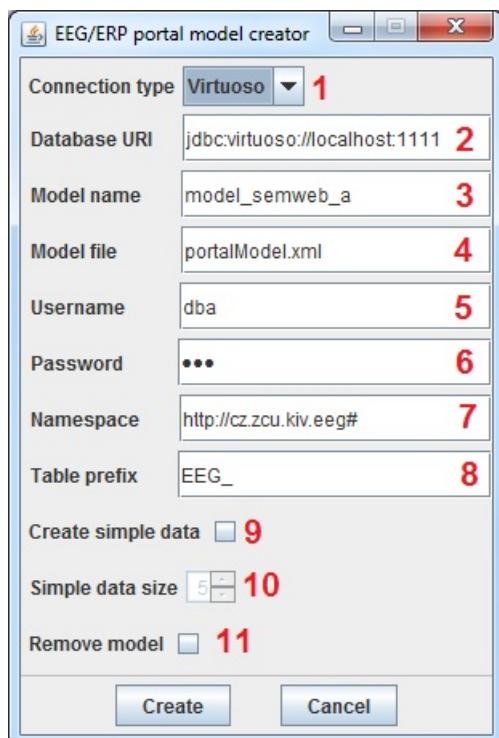
- [11] Ryan Blace Andrew Perez-Lopez John Hebeler, Matthew Fisher. *Semantic Web Programming*. Wiley, Indianapolis, 2009.  
ISBN 978-0-470-41801-7.
- [12] Marek Obitko. *Modularization of Ontologies*, 2007.  
<http://www.obitko.com/tutorials/ontologies-semantic-web/modularization-of-ontologies.html>, (přístup 10.2.2013).
- [13] Strbačka Martin. *Vytvoření publikačního standardu v oblasti evokovaných potenciálů*. ZČU, Plzeň, 2012.
- [14] Pitner Tomáš Matulík Petr. *Sémantický web a jeho technologie*.  
Masarykova univerzita, 2004. <http://www.ics.muni.cz/zpravodaj/articles/296.html>, (přístup 6.2.2013).
- [15] Openlink Software. *Virtuoso Universal Server*,  
<http://virtuoso.openlinksw.com/>, (přístup 19.2.2013).
- [16] Oracle. *Oracle Database Semantic Technologies Tutorial*,  
[http://download.oracle.com/otndocs/tech/semantic\\_web/pdf/oradb\\_semtech\\_tutorial.pdf](http://download.oracle.com/otndocs/tech/semantic_web/pdf/oradb_semtech_tutorial.pdf), (přístup 19.2.2013).
- [17] Oracle. *Oracle Spatial*, [http://www.orafaq.com/wiki/Spatial\\_FAQ](http://www.orafaq.com/wiki/Spatial_FAQ),  
(přístup 19.2.2013).
- [18] Brůha Petr. *EEG/ERP portál a prostředky sémantického webu*. ZČU, Plzeň, 2011.
- [19] Hanyáš Petr. *Sémantický web - tutoriál a demonstrační příklady*. Vysoké učení technické v Brně, 2007. [http://www.hanyas.net/download/soubor/bp\\_cesky.pdf](http://www.hanyas.net/download/soubor/bp_cesky.pdf), (přístup 6.1.2013).
- [20] Semantic web. *Ontology*, 2012.  
<http://semanticweb.org/wiki/Ontology>, (přístup 9.2.2013).
- [21] Svátek Vojtěch. *Sémantický web*, 2010.  
[http://nb.vse.cz/\\_svatek/rzzw/seweb-prehled.pdf](http://nb.vse.cz/_svatek/rzzw/seweb-prehled.pdf), (přístup 10.2.2013).
- [22] Svátek Vojtěch. *SPARQL*, 2010.  
[http://nb.vse.cz/\\_svatek/rzzw/SPARQL11.2.2013](http://nb.vse.cz/_svatek/rzzw/SPARQL11.2.2013).
- [23] Štencek Jiří. *Užití sémantických technologií ve značkovacích jazycích*. Vysoká škola ekonomická v Praze, 2009.  
<http://vse.stencek.com/semanticky-web/>, (přístup 4.2.2013).

- [24] Jamie Taylor Toby Segaran, Colin Evans. *Programing the Semantic Web*. O'Reilly Media, Sebastopol, 2009.  
ISBN 978-0-596-15381-6.
- [25] Vitvar Tomáš. *Sémantický web*. ČVUT, Praha, 2011,  
<http://www.cvut.cz/pracoviste/odbor-rozvoje/stranky/habilitace-a-inaugurace/habilitacni-prednasky/lecture.pdf>.
- [26] Matthew Moran Vipul Kashyap, Christoph Bussler. *The Semantic Web - Semantics for Data and Services on the Web*. Springer, Berlin, 2008.  
ISBN 978-3-540-76451-9.
- [27] Homola Vladimír. *Kartézský součin, relace*. Vysoká škola báňská,  
<http://homel.vsb.cz/hom50/SLBSTATS/MNO/GS01B.HTM>, (přístup 17.2.2013).
- [28] W3C. *RDFConcepts and Abstract Syntax*, 2004.  
<http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>, (přístup 7.2.2013).
- [29] W3C. *OWL Web Ontology Language Overview*, 2004.  
<http://www.w3.org/TR/owl-features/>, (přístup 9.2.2013).
- [30] W3C. *RDF/XML Syntax Specification*, 2004.  
<http://www.w3.org/TR/REC-rdf-syntax>, (přístup 6.2.2013).
- [31] W3C. *RDF Vocabulary Description Language 1.0*, 2004.  
<http://www.w3.org/TR/rdf-schema>, (přístup 8.2.2013).
- [32] W3C. *W3C Semantic Web Activity*, 2006.  
<http://www.w3.org/2001/12/semweb-fin/w3csw>, (přístup 8.1.2013).
- [33] W3C. *SPARQL Query Language for RDF*, 2008.  
<http://www.w3.org/TR/rdf-sparql-query>, (přístup 8.2.2013).
- [34] W3C. *XML Schema Part 2*, <http://www.w3.org/TR/xmlschema-2/>,  
(přístup 20.2.2013).

# Přílohy

## A Uživatelský manuál aplikace pro tvorbu modelu

Spuštění aplikace pro vytvoření modelu se provádí spuštěním dávkového souboru *runCreator.bat* (v prostředí Windows). Po jejím naběhnutí se zobrazí okno dle obrázku A.1. Význam jednotlivých vstupních položek jež jsou označeny červenými čísly je následující:



Obrázek A.1: Okno nástroje pro vytváření modelu

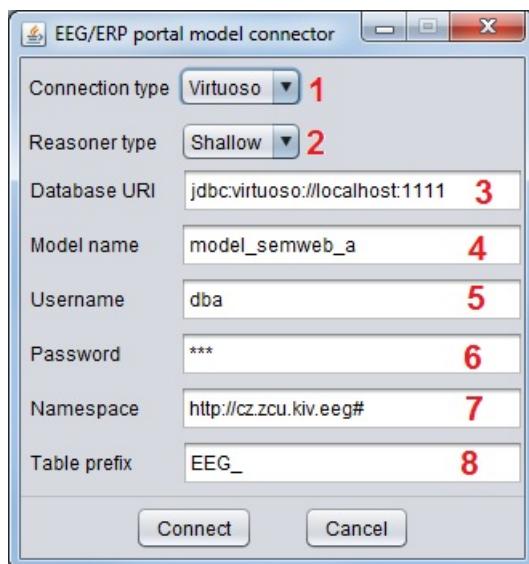
- 1 Výběr cílového databázového systému pro zapisovaný model - Oracle nebo Virtuoso
- 2 URI cesta ke zvolené databázi ve tvaru: *jdbc:<databáze>://<server>:<port>*
- 3 Název vytvářeného sémantického modelu
- 4 Cesta ke XML souboru s popisem modelu
- 5 Uživatelské jméno pro přístup k databázi

- 6** Uživatelské heslo pro přístup k databázi
- 7** Jmenný prostor objektů vytvářeného modelu – tento prefix bude použit pro každý objekt popsaný ve vstupním XML souboru
- 8** Prefix všech tabulek vytvářeného modelu – slouží ke přiřazení tabulek relační databáze ke sémantickému modelu
- 9** Vytvoření testovacích dat, která budou zapsána do vytvářeného modelu
- 10** Velikost testovacích dat – počet vytvářených instancí pro každou třídu
- 11** Odstranění existujícího modelu – namísto vytvoření nového modelu dojde ke odstranění existujícího, je nutné zadat cestu ke vstupnímu XML souboru, aby bylo možné odstranit i všechny relační tabulky přidružené k datovému modelu

Stisknutím tlačítka *Create* dojde k zahájení vytváření modelu o jehož průběhu informuje nové okno zobrazující aktuální stav zápisu. V případě jakékoliv chyby při zápisu modelu jsou veškerá chybová hlášení zapisována do logovacího souboru v adresáři *ModelCreator*.

## B Uživatelský manuál testovací aplikace

Testovací aplikace umožňuje vizualizaci datového modelu a provádění jeho úprav prostřednictvím grafického uživatelského rozhraní. Aplikace se spouští dávkovým souborem *runVisualizer.bat* (v prostředí Windows). Po spuštění aplikace je zobrazeno okno dle obrázku A.2, které slouží ke zadání dat pro připojení k modelu. Význam jednotlivých vstupních položek jež jsou označené červenými čísly je následující:

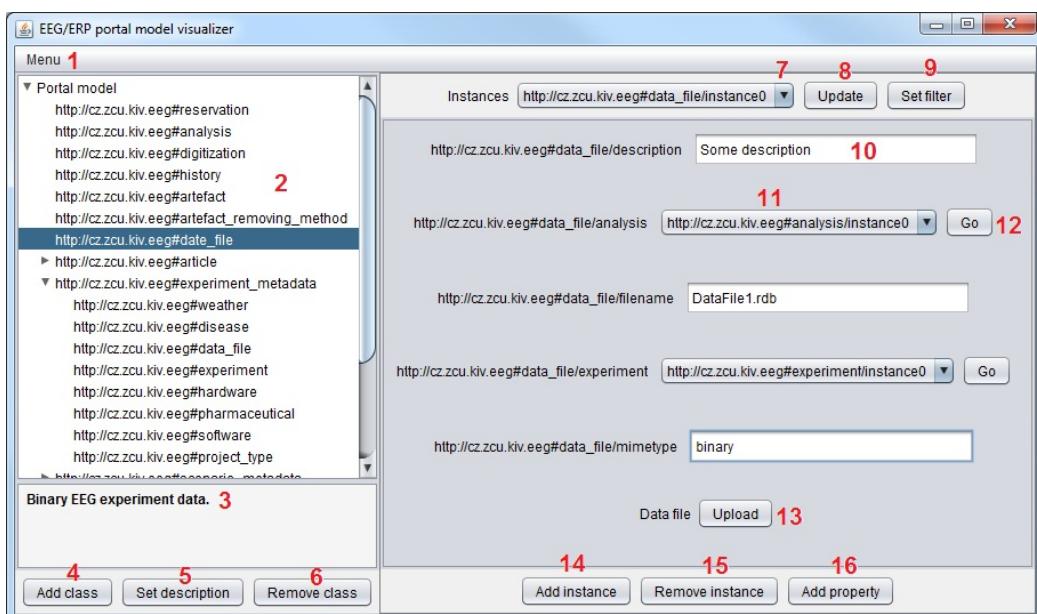


Obrázek A.2: Okno připojení testovací aplikace

- 1 Výběr cílového databázového systému pro zapisovaný model- Oracle nebo Virtuoso
- 2 Výběr použitého reasoneru pro vizualizaci modelu – mělký (*Shallow*)/hluboký (*Deep*)
- 3 URI cesta ke zvolené databázi ve tvaru: *jdbc:<databáze>://<server>:<port>*
- 4 Název vytvářeného sémantického modelu
- 5 Uživatelské jméno pro přístup k databázi
- 6 Uživatelské heslo pro přístup k databázi

- 7** Jmenný prostor objektů modelu – tento prefix bude předdefinován všem nově vytvářeným objektům prostřednictvím testovací aplikace
- 8** Prefix všech tabulek vytvářeného modelu – slouží ke přiřazení tabulek relační databáze ke sémantickému modelu

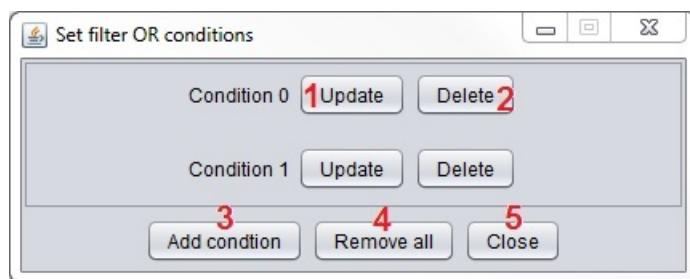
Po úspěšném připojení testovací aplikace ke zadánému modelu se zobrazí okno dle obrázku A.3. V případě jakékoliv chyby při načítání modelu je zapsána detailní informace o této chybě do logovacího souboru v adresáři *ModelVisualizer*.



Obrázek A.3: Hlavní okno testovací aplikace

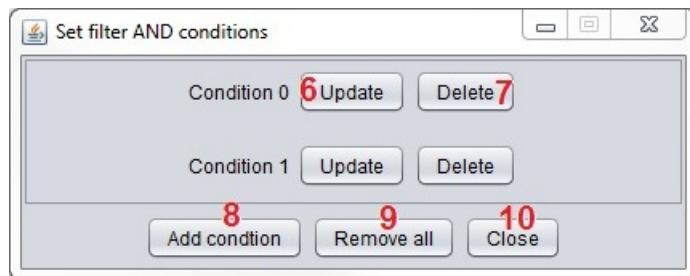
Hlavní okno testovací aplikace slouží ke zobrazení tříd modelu a jejich instancí, které je možné dále prohlížet či modifikovat. Hlavní menu aplikace (1) umožňuje prostřednictvím položky *Export model* uložit datový model ve RDF/XML formátu do souboru. Levý panel aplikace (2) zobrazuje hierarchický systém tříd modelu, kde je možné provádět jejich výběr dle URI identifikátorů tříd. Výběrem třídy dojde k zobrazení jejího sémantického popisu ve spodním panelu (3), který umožňuje tento popis editovat a tlačítkem *Set description* (5) uložit. Třídy modelu je možné odstraňovat tlačítkem *Remove class* (6) címž dojde i k odstranění všech instancí této třídy. Zároveň je možné vytvářet nové třídy tlačítkem *Add class* (7), kdy dojde ke zobrazení dialogu,

který slouží ke definici URI identifikátoru třídu, sémantického popisku a nastavení rodičovské třídy. Tato rodičovská třída je určena vybranou položkou v panelu tříd (2) a při vytváření třídy je možné zaškrťvacím políčkem určit zda nová třída bude potomkem této třídy či nikoliv. Výběrem třídy v panelu (2) dojde ke aktualizaci seznamu instancí tříd (7), přičemž první položka tohoto seznamu (7) je zobrazena v hlavním pravém panelu aplikace. Z tohoto seznamu (7) je možné provést výběr požadované instance třídy dle jejího URI či dále nastavit restriktivní výběr pro list těchto tříd nastavením filtru tlačítkem *Set filter* (9). Nastavení filtru se provádí v okně dle obrázku A.4, které umožňuje nastavení filtrovacích podmínek.



Obrázek A.4: Hlavní okno nastavení filtrovacích podmínek

Toto okno umožňuje přidávání nových (3), odstraňování (2) či aktualizaci stávajících (1) podmínek, které jsou vyhodnocovány jako logický součet výsledků dílčích podmínek, jež jsou nastaveny v okně dle obrázku A.5.



Obrázek A.5: Okno nastavení součinových podmínek

Toto okno umožňuje nastavení dílčích podmínek jako logický součin další množiny podmínek. Tyto součinové podmínky je možné opět přidávat (8), odstraňovat (7) či aktualizovat (6). Všechny dílčí součtové či součinové podmínky je možné hromadně uložit tlačítkem *Close* (5), (10) nebo je hromadně

odstranit tlačítkem *Remove all* (4), (9). Koncové podmínky jež tvoří množinu součinových podmínek je možné nastavovat v okně dle obrázku A.6.



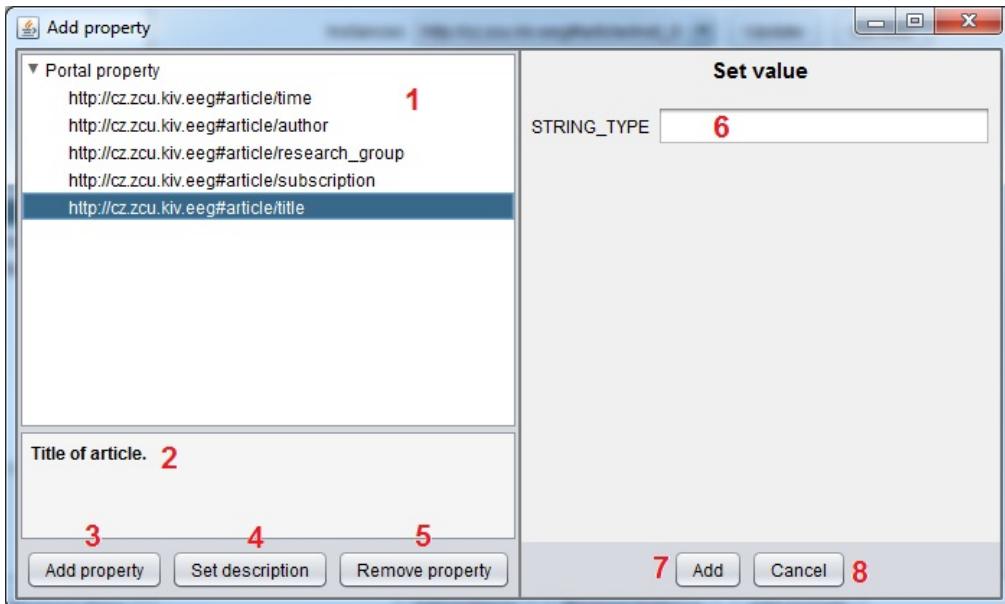
Obrázek A.6: Okno nastavení koncových podmínek filtrování

Toto okno umožňuje výběr (11) ze čtyř druhů podmínek popsaných v kapitole 8.3.2, kde jméno URI identifikátoru predikátu je zadáváno do horního textového pole (12) a hodnota cílového objektu vázaného predikátem je zadávána do spodního pole (13) (v případě, že jí daný typ podmínky vyžaduje). Po uzavření okna pro nastavení filtru je možné provést aktualizace seznamu instancí tříd dle nastaveného filtru tlačítkem *Update* (8) dle obrázku A.3.

Hlavní pravý panel aplikace zobrazuje pro vybranou instanci třídy seznam predikátů a k nim vázaných objektů. V případě, že cílový objekt je primitivního datového typu, je zobrazeno textové pole (10) s jeho textovou reprezentací, které umožňuje také aktualizaci hodnoty pouhou změnou obsahu tohoto pole a stiskem klávesy Enter. V případě potřeby úplného odstranění této hodnoty je možné textové pole vymazat a opět stisknutím klávesy Enter bude cílový objekt včetně svazujícího literálu odstraněn. V případě když predikátem vázaný objekt je instancí jiné třídy, je zobrazena položka pro výběr instance cílové třídy (11), kde je předvybrána aktuálně nastavená instance. Stiskem tlačítka *Go* (12) je možné tuto instanci zobrazit a dále s ní pracovat.

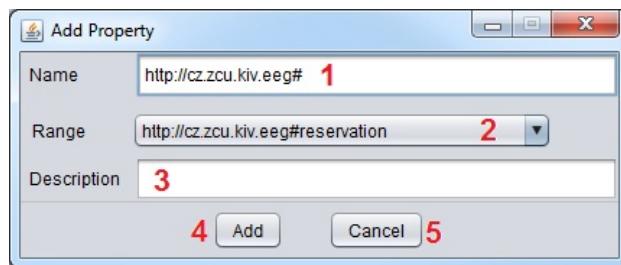
Při najetí myší na jméno predikátu je nad ním zobrazen jeho sémantický popisek. V případě, že třída aktuálně prohlížené instance je vázáná na relační tabulkou je jejím instancím v hlavním panelu zobrazeno tlačítko (12) pro nahrání datového souboru, které je po jeho úspěšném nahrání nahrazeno tlačítkem pro stažení a aktualizaci tohoto souboru. Instance tříd je možné odstraňovat tlačítkem *Remove instance* (16), vytvářet nové tlačítkem *Add instance* (14) či přidávat nové predikáty stávajícím instancím tlačítkem *Add property* (16), což je realizováno prostřednictvím nového okna tak, jak je

znázorněno dle obrázku A.7.



Obrázek A.7: Okno nastavení predikátů

Toto okno slouží ke spravování predikátů určených doménou třídy instance vybrané v hlavním okně aplikace. Levý panel okna (1) umožnuje vybrat existující predikát, kterému lze ve spodním panelu (2) upravit sémantický popisek a aktualizovat jej tlačítkem *Set description* (4). Dále je možné prostřednictvím tlačítka *Remove property* (5) odstranit stávající predikát, či vytvořit nový tlačítkem *Add property* (3), což je realizováno novým oknem dle obrázku A.8.



Obrázek A.8: Okno vytvoření nového predikátů

První pole (1) tohoto okna slouží ke nastavení URI nového predikátu, dále lze nastavit jeho rozsah (2), kterým může být primitivní datový typ či

jiná třída. Položka *Description* (3) umožňuje nastavení sémantického popisku přidávaného predikátu. Jeho přidáním tlačítkem *Add* (4) dojde k aktualizaci dostupných predikátů v předchozím okně (viz obrázek A.7). Výběrem predikátu v tomto okně dojde v pravém panelu okna k zobrazení textového pole (6) pro zadání hodnoty nové objektu pro vybraný literál (je li rozsah literálu primitivní datový typ) nebo se zobrazí položka pro výběr instance třídy (je li rozsah literálu nastaven na jinou třídu). Přidání nového objektu vázaného na vybraný literál instanci se proveden stiskem tlačítka *Add*(7), čímž je následně aktualizována vybraná instance v hlavním okně.

## C Navržený hierarchický systém tříd modelu

```
<model>

<!-- DATA -->

<data_table name="article" type="text"/>
<data_table name="article_comment" type="text"/>
<data_table name="data_file" type="binary"/>
<data_table name="scenario" type="binary"/>

<!-- METADATA -->

<class name="article"
      description="Text arcicle published by person."/>
<class name="article_comment"
      description="Comment of article published by person."/>
<class name="reservation"
      description="EEG research laboratory room reservation."/>

<class name="person"
      description="Human being."/>
<class name="person/researcher"
      description="Human being making EEG/ERP experiments."/>
<class name="person/test_subject"
      description="Human beeing that is subject of EEG/ERP
      experiments."/>

<class name="education_level"
      description="Education level of human beeing."/>
<class name="research_group"
      description="Union of human beings that makes EEG/ERP
      experiments."/>
<class name="history"
      description="EEG/ERP experiment download history."/>
<class name="experiment"
      description="EEG/ERP experiment."/>
<class name="software"
      description="Software used for EEG/ERP experiments."/>
```

```

<class name="hardware"
      description="Hardware used for EEG/ERP experiments."/>
<class name="subject_group"
      description="Union of human beings that are EEG/ERP
      experiments subjects."/>

<class name="project_type"
      description="Experiment project type."/>
<class name="artefact_removing_method"
      description="Removing method of artifact."/>
<class name="weather"
      description="Experiment weather condition."/>
<class name="disease"
      description="Experiment test subject disease condition."/>
<class name="artefact"
      description="Experiment artefact"/>
<class name="pharmaceutical"
      description="Experiment pharmaceutical condition."/>
<class name="digitization"
      description="Experiment digitization."/>
<class name="analysis"
      description="Experiment analysis."/>
<class name="data_file"
      description="Experiment EEG/ERP measurement data file."/>
<class name="scenario"
      description="Experiment scenario description."/>

<class name="electrode_configuration"
      description="Configuration of measuring electrodes."/>
<class name="electrode_system"
      description="System of experiment electrodes."/>
<class name="electrode_location"
      description="Location of experiment electrodes."/>
<class name="electrode_type"
      description="Type of experiment used electrodes."/>
<class name="electrode_fix"
      description="Fixing of electrodes."/>

<class name="stimulus"
      description="Experiment stimulus."/>

```

```

<class name="stimulus_type"
      description="Experiment stimulus type."/>
<class name="scenario_stimulus"
      description="Stimulus of scenario."/>

<class name="group"
      description="Union of human beings."/>
<class name="experiment_metadata"
      description="Basic experiment metadata."/>
<class name="electrode"
      description="Experiment electrode device."/>
<class name="scenario_metadata"
      description="Experiment scenario."/>

<!-- Class hierarchy -->

<parent_class name="person">
    <child name="person/researcher"/>
    <child name="person/test_subject"/>
    <child name="education_level"/>
</parent_class>

<parent_class name="group">
    <child name="research_group"/>
    <child name="subject_group"/>
</parent_class>

<parent_class name="experiment_metadata">
    <child name="software"/>
    <child name="hardware"/>
    <child name="project_type"/>
    <child name="weather"/>
    <child name="disease"/>
    <child name="pharmaceutical"/>
    <child name="data_file"/>
    <child name="experiment"/>
</parent_class>

<parent_class name="electrode">
    <child name="electrode_configuration"/>

```

```
<child name="electrode_system"/>
<child name="electrode_location"/>
<child name="electrode_type"/>
<child name="electrode_fix"/>
</parent_class>

<parent_class name="scenario_metadata">
    <child name="stimulus"/>
    <child name="stimulus_type"/>
    <child name="scenario_stimulus"/>
    <child name="scenario"/>
</parent_class>

<parent_class name="article">
    <child name="article_comment"/>
</parent_class>
```

## **D ERA model databáze EEG/ERP portálu**

ERA model EEG/ERP portálu [13] viz následující list

Příloha č. 8: ERA model upravené databáze EEG/ERP portálu

Legenda:

- Nové tabulky
- Bezé změn
- Ta tabulky vznikly
- Ta tabulky vznikly

