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

Bakalářská práce

EEG/ERP portál - transformace do sémantického webu

Plzeň 2011 Filip Markvart

Prohlášení

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

V Plzni dne 9. května 2011

Filip Markvart

Abstract

This thesis describes a transformation of neuroinformatics metadata stored in the relational database to the semantic web standard. The main goal is to investigate existing tools and necessary modifications that ensure automatic transformation for converting input data. The modified tool is able to process additional semantic information that cannot be stored in the relational database.

The last part describes design and implementation of an application that uses annotations to add semantic information to transformed data.

Obsah

1 Úvod **1**

2 EEG/ERP portál **2**

3 Sémantický web **4**
3.1 Architektura sémantického webu 6
3.2 RDF 7
3.3 Ontologie 8
3.4 OWL 9

4 Relační databáze a sémantický web **11**
4.1 Jena 11
4.2 Jenabean 12
4.2.1 Použití nástroje Jenabean 13
4.3 Java anotace 14
4.4 Anotace nástroje Jenabean 17
4.5 Rozšíření nástroje Jenabean 19

5 Nástroj pro anotování POJO tříd **23**
5.1 Annotation File Utilities 23
5.2 Annotation tool 25
5.2.1 Analýza problému 25
5.2.2 Popis implementace 28

6 Testování **35**

7 Závěr **36**

1 Úvod

Vyšetření elektrické aktivity mozku a evokovaných potenciálů patří mezi standardní metody neurologického výzkumu. Tato měření jsou prováděna také v laboratoři Katedry informatiky a výpočetní techniky Západočeské univerzity v Plzni. Všechna zde naměřená data jsou uchovávána v databázi webové aplikace označené jako EEG/ERP portál. Aby bylo možné získané údaje prezentovat na serveru NIF (Neuroscience Information Framework) [4], je zapotřebí všechna data převádět do některého ze standardních formátů sémantického webu.

Cílem této práce je prozkoumat možnosti reprezentace dat v prostředcích sémantického webu a vybrat z nich nejvhodnější formát pro reprezentaci dat získaných z EEG/ERP portálu. Práce se v další kapitole zabývá samotnou transformací dat z portálu do vybraného formátu, kterou se snaží následně co nejvíce zautomatizovat tak, aby vyžadovala minimální uživatelské zásahy. V této části je také popsán nástroj použitý při transformaci, který je dále modifikován tak, aby umožňoval dodávat transformovaným datům další sémantické informace prostřednictvím Java anotací.

Následující kapitola se věnuje návrhu a implementaci nástroje pro explicitní přidávání sémantických informací anotacemi. Jeho testovaní je pak popsáno v předposlední kapitole této práce.

2 EEG/ERP portál

EEG/ERP portál je webová aplikace vytvářená pro Katedru informatiky a výpočetní techniky Západočeské univerzity v Plzni. Výzkumná skupina katedry se zabývá experimenty podle definovaných scénářů, při kterých měří evokované potenciály osobám, které jsou vystaveny specifikovaným vlivům. Při těchto experimentech se měří velké množství údajů, které je potřeba vhodně zaznamenávat tak, aby je bylo možné později jednoduše analyzovat a zpracovávat. Vzhledem k tomu, že samotná měření provádí větší počet osob, které se mohou sdružovat do skupin, je zapotřebí systému, který umožňuje přístup a práci s daty pro více uživatelů. Z tohoto důvodu byl vytvořen EEG/ERP portál, který umožňuje veškeré experimenty uchovávat.

Tato webová aplikace umožňuje všem registrovaným uživatelům vytvářet experimenty, vkládat naměřené hodnoty, vyhledávat vložená data vlastní či jiných uživatelů a údaje stahovat. Uživatelé, kteří provádějí experimenty, se mohou sdružovat do skupin a na portálu si vzájemně vyměňovat informace mezi sebou či s jinými skupinami. Aby ukládaná data z měření měla co nejlepší vypovídací hodnotu, je zapotřebí zaznamenávat nejen samotné naměřené přístrojové hodnoty, ale také přidružené údaje experimentu – tzv. metadata. Těchto údajů je velké množství a mezi základní patří metadata o experimentu, měřící a měřené osobě a také údaje o scénáři. Analýzou tabulek databáze bylo zjištěno, že mezi nejdůležitější uchovávaná metadata patří následující údaje [18]:

Metadata o experimentu

- měřící osoba
- měřená osoba
- použitý scénář
- počasí
- začátek měření
- konec měření
- teplota
- případné poznámky

Metadata o osobě

- skupina
- jméno
- příjmení
- pohlaví
- datum narození
- kontaktní údaje

Metadata o scénáři

- vlastník
- výzkumná skupina
- název
- popis
- soubor se samotným scénářem

Veškerá naměřená data i metadata jsou uchovávána v databázovém systému Oracle. Portál s těmito daty pracuje a získává je objektově relačním mapováním (framework Hibernate). Celý portál běží jako webová aplikace napsaná v jazyce Java a využívá mimo již zmíněných technologií také framework Spring MVC, Spring Security a Spring Core.

Aby bylo možné EEG/ERP portál zaregistrovat do NIF (Neuroscience Information Framework), je nutné veškerá naměřená data poskytovat v jazycích sémantického webu. Vzhledem ke struktuře metadat, jejichž transformaci je třeba provést, a požadavkům NIF se jeví jako nejpříznivější cesta převod do standardního formátu OWL (Ontology Web Language), který je prostředkem sémantického webu.

3 Sémantický web

V současné době je web tvořen obrovským množstvím informací v podobě webových stránek pocházejících od různých organizací, komunit či jednotlivců. Uživatelé k nim mohou snadno získávat přístup prostřednictvím URI (Uniform Resource Identifier) adresy, vyhledávačů nebo hypertextových odkazů [20]. Takováto podoba webu je sice poměrně jednoduchá, ale má své limity, protože v nepřeberném množství informací se lze snadno ztratit či jen sklouznout k irelevantním informacím. Následující příklad, který uvádí [20], názorně ukazuje hlavní problém. Chceme-li na webu vyhledat dokumenty, jejichž autorem je Eric Miller, vyhledávače nám vrátí spoustu odkazů na stránky, týkající se osob tohoto jména, či jen obsahující daný výraz nebo jeho část, ale nezajistí vždy nalezení skutečně požadovaných informací. Problémem je skutečnost, že vyhledávací roboti a podobné nástroje pouze hledají výskyt zadaných údajů bez ohledu na jejich význam, který je často klíčový.

Cílem sémantického webu je vyvíjet technologie, které by umožnily, aby stroje „rozuměly“ datům, se kterými pracují. Snahou sémantické webu ale není jen dostávat přesnější výsledky při vyhledávání informací, ale také získávat data z více různých zdrojů, porovnávat je a zpracovávat a díky tomu vytvářet informace nové. Aby bylo s daty možné takto pracovat, je nutné nepublikovat je jako samotné, ale přidružovat k nim přídavná data, která zachycují kontext, strukturu a obsah - metadata [5]. U výše uvedeného příkladu by to znamenalo, že pokud budeme publikovat dokumenty vytvořené osobou jmenující se Eric Miller, specifikujeme toto jméno do metadat jako autora dokumentu. Tím se zajistí provázání konkrétní osoby s konkrétním dílem. Užitím metadat je pak možné s obsahem webu pracovat jako s relační databází a v důsledku toho se dotazovat na její obsah pomocí jazyků jako je SPARQL (Protocol and RDF Query Language).

Celou myšlenku sémantického webu poprvé publikoval v květnu roku 2001 v časopise Scientific American zakladatel současného webu Tim Berners Lee [8]. Sémantický web vychází z několika základních principů, které je nutno uvést.

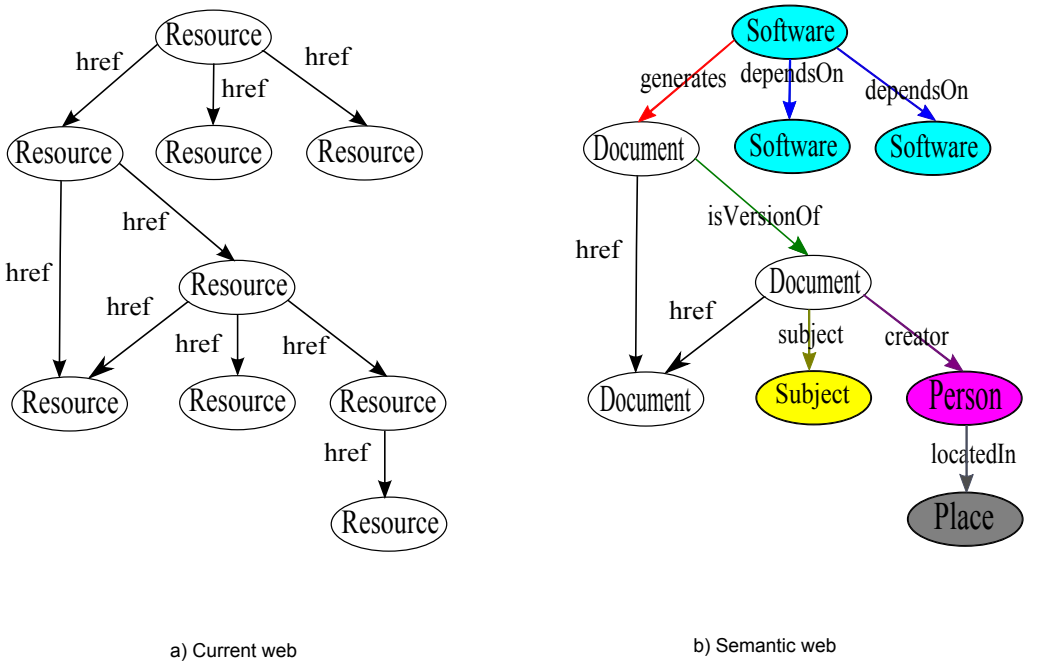
- **Vše je možné identifikovat prostřednictvím URI**

  Na veškeré objekty reálného světa jako jsou lidé, místa či jiné věci je možné se odkazovat v sémantickém webu identifikátory. Jakýkoliv

tvůrce webu tak může vytvořit jedinečný identifikátor URI a říci, že označuje nějaký konkrétní objekt skutečného světa.

- **Odkazy mezi jednotlivými zdroji a zdroje samotné lze typovat**

  Současný web sestává ze zdrojů a odkazů. Zdroje jsou webové dokumenty určené pouze pro koncové osoby, a neobsahují tak žádná metadata popisující jejich účel či vztah k ostatním dokumentům. Cílový čtenář si tyto informace dokáže obvykle odvodit z obsahu, což ale počítačové systémy neumí, nebo je to pro ně příliš složité. Řešením je nevytvářet mezi dokumenty obyčejné odkazy, ale jednotlivé vztahy mezi objekty také popisovat. Jak je patrné ze schématu na obrázku 3.1, vazby mezi objekty lze popsat hodnotami jako např. „ je autorem“, „závisí na“ nebo „ je verzí“. Díky existenci typování jak zdrojů, tak vazeb, je pak možné kupříkladu vyjádřit, že nějaký zdroj je pouze verzí jiného zdroje.



Obrázek 3.1: Schéma provázání dat v současném a sémantickém webu [20]

- **Tolerance neúplných informací**

  Stejně jako u současné podoby webu, není nutné, aby odkaz zdroje byl neustále platný. Nedosažitelnost cílového zdroje tak není funkční překážkou, protože nástroje sémantického webu zpracovávají pouze dostupné informace, na základě kterých vytváří závěry. Stejně tak je možné zpracovávat jen vybrané úseky informací, ze kterých budou získávány neméně hodnotné výsledky.

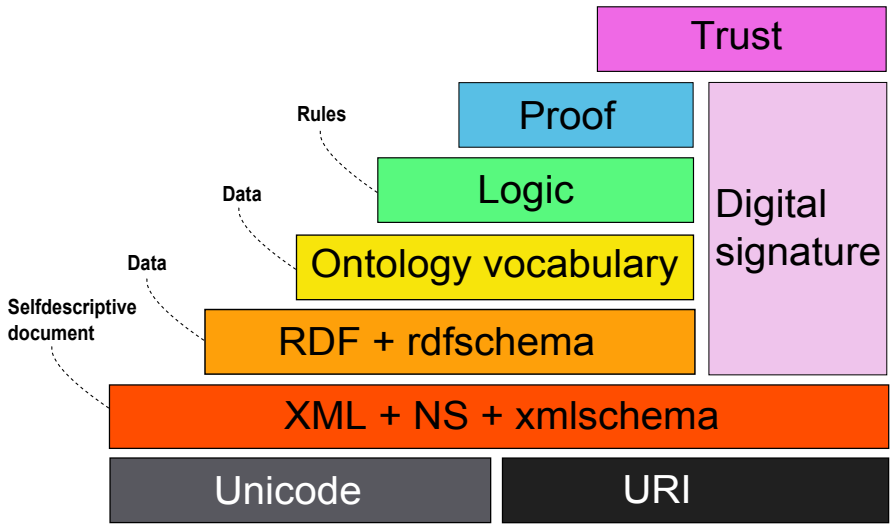
- **Podpora paralelního vývoje dat**

  Zcela běžně může docházet k tomu, že různé skupiny osob vytváří obdobná data na různých místech nebo v jiném čase. Stejně tak se může u nich lišit použitá terminologie, ač skutečný význam obsahu je stejný. Díky prostředkům sémantického webu je možné např. s pomocí typovaných odkazů zajistit významovou provázanost obsahově obdobných či na sebe navazujících dat i když je forma jejich zápisu odlišná. Je tak možné přidávat nové informace, beze změny původních dat, na které se je stále možné odkazovat [20].

3.1 Architektura sémantického webu

Architektura sémantického webu je tvořena více vrstvami, které obsahují jednotlivé webové technologie a standardy. Z obrázku 3.2 je celá architektura patrná. Nejnižší vrstva určuje kódování mezinárodní znakovou sadou a jednoznačnou identifikaci objektů pomocí URI. Vrstva o úroveň výše zajišťuje definici dat pomocí XML (Extensible Markup Language) standardů. XML schéma zajišťuje strukturu XML dokumentů a zároveň XML rozšiřuje o datové typy. Vrstva RDF + rdfschema popisuje strukturu metadat a zajišťuje typování zdrojů a odkazů. Poskytuje tím jednoduchou sémantiku pro datový model a díky RDF schématu i hierarchii zdrojů.

Vrstva ontologického slovníku zajišťuje přidávání ontologií a umožňuje tak konstruovat složitě vázané struktury díky obsahově rozšířenému slovníku pro popisování vlastností. Digitální podpisy pak umožňují detekci různých verzí dokumentů a jsou spolu se všemi předchozími částmi standardizované konsorciem W3C. Zbylé vyšší vrstvy standardizované nejsou a jsou neustále vyvíjeny.



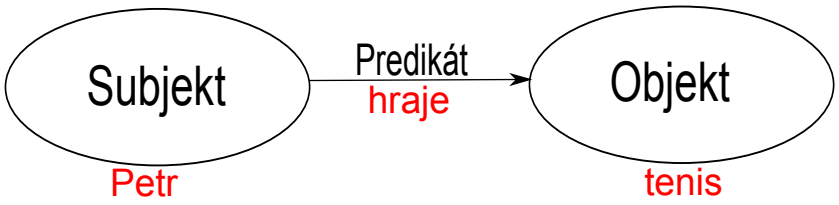
Obrázek 3.2: Architektura sémantickém webu [20]

3.2 RDF

Technologickým základem celého sémantického webu je podle konsorcia W3C standard RDF(Resource Description Framework). Rámec RDF poskytuje jednoduchý model pro popis zdrojů bez závislosti na jeho konkrétní implementaci [8]. Jedná se o jednoduchý jazyk, který umožňuje popisovat strukturu metadat tzv. trojicemi. Trojice je tvrzení, jež je tvořené subjektem, predikátem a objektem, které slouží k popisu zdroje. Pod touto trojicí je možné si představit tvrzení, že zdroj (subjekt) nabývá pro jistou vlastnost (predikát) nějakou hodnotu (objekt).

Subjektem může být libovolný objekt, který lze popsat identifikátorem URI, tedy např. webová stránka či předmět reálného světa. Predikát je vlastnost, kterou má popisovaný subjekt. Objekt je pak hodnotou, které nabývá příslušný predikát a uceluje tak popis vlastnosti subjektu. Dle standardu [19] je dokonce možné, aby objektem byl jiný subjekt (zdroj).

Vzniklou trojící tak může být např. tvrzení: „Petr hraje tenis.“, jak je patrné z obrázku 3.3.



Obrázek 3.3: Příklad RDF trojice

Datový model RDF lze reprezentovat grafy nebo trojicemi, ale pro vyjádření sémantiky webových zdrojů je nejvhodnější zápis syntaxí jazyka XML. Pomocí tohoto zápisu pak lze přiřazovat webovým zdrojům určité vlastnosti nebo mezi jednotlivými zdroji vyjadřovat vzájemné vztahy. Příklad uvedený na obrázku 3.3., zapsaný standardizovanou syntaxí XML [19] vypadá následovně.

```
<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/Petr">
    <dc:hraje>tenis</dc:hraje>
  </rdf:Description>
</rdf:RDF>
```

3.3 Ontologie

Ontologie umožňuje vytváření metadat, tedy dodávání významových informací, které jsou klíčové v sémantickém webu [14]. V informatice je ontologie definována jako „formální specifikace sdílené konceptualizace“ [15]. Jedná se tedy o vytváření abstraktního modelu v určité oblasti – definování pojmů včetně jejich vzájemných vztahů, které jsou vyjádřené v logickém jazyce zpracovatelném počítačem. Veškeré pojmy tak musejí být explicitně specifiko-

vány, aby informacím nerozuměli pouze lidé, ale i znalostně orientované systémy.
Ontologie se člení na 3 základní typy [14]:

Terminologické - slouží především k získávání informací, nejvíce se blíží ke slovníkům synonym

Informační - slouží jako nadstavba databází a zajišťují kontrolu integrity a určitou abstrakci databáze potřebnou k dotazování

Znalostní - slouží pro reprezentaci znalostí v oblasti umělé inteligence

Aby bylo možné ontologické informace formálně zapisovat, je zapotřebí jazyka, který by to umožňoval. Takových jazyků bylo vytvořeno poměrně velké množství a pro současnou podobu sémantického webu má největší význam OWL (Ontology Web Language).

3.4 OWL

OWL je webový ontologický jazyk, který na rozdíl od předchozího RDF dovoluje vytvářet větší slovník a podrobnější syntaxi. Jeho vznik byl zapříčiněn potřebou získávat informace samotnými stroji (oproti současnému způsobu, kdy informace jsou získávány člověkem) [14]. Oproti svému předchůdci má OWL širší vyjadřovací možnosti sémantiky, a tím zároveň i obsahu. Na rozdíl od jazyků RDF a RDFS (RDF Schema) umožňuje navíc definovat například:

- lokální omezení vlastností v rámci určité třídy, např. kardinalitu (sdílená nemovitost musí mít alespoň 2 vlastníky)
- disjunktnost či ekvivalenci tříd, např. třída Vlastník je disjunktní se třídou Nemovitost
- anonymní třídy (definované určitým logickým výrazem pro jednorázové použití)
- matematické charakteristiky vlastností [15]

Jazyk OWL byl standardizovaný konsorciem W3C ve 3 variantách, které se liší se svou expresivitou.

- **OWL Lite** je nejjednodušší verzí, která umožňuje definovat pouze hierarchii tříd a jednoduchá omezení.
- **OWL DL** je složitější variantou a zahrnuje veškeré konstrukce jazyka OWL, jejichž použití je ale omezené. Zkratka v názvu DL znamená, že tato verze využívá deskripční logiku.
- **OWL Full** je verzí, která zajišťuje maximální expresivitu. Umožňuje tak vytvářet velmi složité konstrukce bez omezení předchozích verzí.

Každá uvedená varianta je rozšířením verze svého předchůdce, který v ní může být plně vyjádřen [14].
Co se týče možností syntaxe jazyka OWL, existuje dle specifikace W3C několik standardů, jejichž použití je patrné z tabulky 3.1.

RDF/XML	Univerzální formát, který umožňuje migraci dat mezi různým softwarem
OWL/XML	Formát pro zpracování XML nástroji
Funcional Syntax	Formát vhodný pro zobrazení formální struktury ontologií
Manchester Syntax	Použití pro zápis a čtení DL verzí jazyka
Turtle	Užití pro tvorbu a získávání RDF trojic

Tabulka 3.1: Syntaxe jazyka OWL [21]

4 Relační databáze a sémantický web

EEG/ERP Portál je webová aplikace běžící nad relační databází. Hlavním funkčním požadavkem, který by aplikaci umožnil registraci v rámci projektu NIF, je transformace ukládaných dat do prostředků sémantického webu. Transformace dat relační databáze je v zásadě možná dvěma způsoby [18].

Prvním způsobem je provádět převod dat přímým přístupem k datům relační databáze. Z načítaných dat by se tak postupně vytvářely RDF struktury, které by tvořily celý RDF graf.

Druhým způsobem transformace je přístup k datům databáze prostřednictvím objektového datového modelu. Ten je možné snadno získat díky frameworku Hibernate [7], který zajišťuje automatický a komplexní převod dat do POJO objektů (Plain Old Java Object). S využitím existujících nástrojů je pak možné manuální transformací získat data ve formátu RDF či OWL.

Hlavním problémem prvního způsobu transformace je ale skutečnost, že vytvářené RDF struktury nemají dostatečnou sémantiku, protože ji není možné z databáze získat. Možným řešením je dodávat sémantiku manuálním mapováním tabulek s předem vytvořenou ontologií. Nevýhoda tohoto přístupu nastává v situaci, kdy je zapotřebí změnit strukturu databáze. V takovém případě je pak nutné i ručně pozměnit způsob mapování. Při transformaci druhým způsobem lze sémantiku dodávat přímo do objektového modelu. Datový model je možné snadno upravovat s využitím existujícího, dále popsaného nástroje, který dodává potřebnou sémantiku manuálně definovanými anotacemi.

Nástrojů, které provádí transformaci z relační databáze do jazyka RDF či sémanticky širšího OWL, několik existuje a jejich popis včetně zhodnocení je možné získat z [18]. Z existujících nástrojů se pro potřeby portálu jeví jako nejvhodnější framework Jena, respektive Jenabean.

4.1 Jena

Jena je framework, který slouží k vývoji aplikací sémantického webu. Tento nástroj je vyvinutý v jazyce Java jako open source. Framework v sobě obsahuje [2]:

- RDF API
- OWL API
- Dotazovací engine SPARQL
- Možnost čtení a zápisu RDF dat ve formátu RDF/XML, N3 nebo N-triples
- Ukládání RDF dat do databáze

Jena umožňuje snadnou práci s trojicemi a zajišťuje jejich ukládání do paměti ve dvou podobách [18]. První z možností je jednoduchý nízkoúrovňový graf, nad kterým jsou možné pouze základní operace pro vytváření nebo odebírání trojic a jejich vyhledávání. Druhou možnou variantou uložení je vysokoúrovňový model, který nabízí daleko větší množinu možných operací nad daty. Nástroj obsahuje také tzv. Ontology API (Application Programming Interface), které umožňuje pracovat s ontologiemi, např. ve formátech RDF či OWL. Tyto ontologie se rovněž ukládají v podobě modelu, který nesestavuje nový graf, ale vytváří novou vrstvu nad původním modelem [2].

Aby bylo možné tento nástroj využít k transformaci dat z relační databáze portálu do formátu sémantického webu, je zapotřebí tato data nejdříve získat a v požadované podobě je předávat ke zpracování. Díky frameworku Hibernate je možné data relační databáze převádět do objektového kódu ve formě POJO objektů. Vzhledem k tomu, že samotná Jena takovýto vstupní formát přímo nepodporuje, je zapotřebí využít nadstavby označené jako Jenabean.

4.2 Jenabean

Jenabean je nástroj využívající API frameworku Jena sloužící ke tvorbě dat ve formátech sémantického webu [1]. Hlavním nedostatkem tohoto open source nástroje je, že kromě krátkého tutoriálu neobsahuje žádnou dokumentaci. Jenabean umožňuje transformaci dat z objektově orientovaného kódu do ontologicky provázaných dat. Ta jsou vytvářena a uchovávána jako ontologický model Jeny. Díky této skutečnosti je možné vzniklý model uložit do souboru v podobě XML dokumentu. Nástroj rovněž podporuje opačnou transformaci dat – tedy z existujícího modelu Jeny do objektového kódu, a umožňuje i provádět SPARQL dotazy (prostřednictvím SPARQL enginu Jeny) nad

vzniklým modelem. Pro účely této práce však postačuje využívat první způsob transformace.

Tento nástroj získává vstupní data z POJO objektů. Pro potřebu transformace metadat z Portálu do prostředků sémantického webu se jeví jako nejvhodnější formát OWL, který dovoluje ukládat veškerou potřebnou sémantiku a ontologie. K získání výsledného souboru ve formátu OWL je ještě nutné využít nástroj OWL API [3], který dokáže ze výstupního souboru generovaného z Jeny prostřednictvím Jenabeanu vytvořit OWL/XML soubor.

Veškeré klíčové ontologie určuje Jenabean primárně ze struktury objektového kódu, tedy na základě příslušnosti jednotlivých atributů a metod ke třídám náležícím společným Java balíkům. Ve struktuře sémantického webu jsou ontologie zapisovány užitím jmenných prostorů namespace, které tvoří prefix jednotlivých názvů entit. Takto vygenerované ontologie nemusejí vždy odpovídat skutečnosti a může být zapotřebí je upravit. Jenabean umožňuje nastavit implicitně jmenné prostory, které lze využít k určení ontologických vazeb transformovaných dat jako celku, takže veškeré entity pak budou náležet k definovanému jmennému prostoru. V případě potřeby detailnějšího určení ontologie entit mezi sebou navzájem pomocí jmenných prostorů i samotných názvů je možné jednotlivé POJO objekty rozšiřovat Java anotacemi.

4.2.1 Použití nástroje Jenabean

API Jenabeanu poskytuje metody, díky kterým je možné provádět transformaci dat jednoduchým způsobem bez nutnosti podrobnějších znalostí práce s frameworkem Jena [6]. V prvním kroku je nutné vytvořit prázdný model Jeny, který bude následně obsahovat veškerá zadávaná data. Jena nabízí dvě možné varianty modelu. Prvním je základní model (*Model*), který dovoluje uchovávat pouze jednoduché trojice v podobě RDF dat. Druhou variantou je ontologický model (*OntModel*), který rozšiřuje základní model o možnost dodávat datům další ontologie, jež lze využít v rámci jazyka OWL [2]. Tento model bude dále použit, protože dovoluje uchovávat ontologická data, která budou přidávána prostřednictvím Java anotatací nad POJO třídami, jak je uvedeno v následující kapitole. Nad vytvořenou instancí modelu je pak možné provádět operace zápisu dat z POJO tříd prostřednictvím třídy *Bean2Rdf* a data zpětně z modelu načítat do objektové podoby užitím metod třídy *Rdf2Bean*. Nad instancí modelu je tedy možné pracovat prostřednictvím těchto tříd nebo lze využít zastřešující třídy *Jenabean* z Java balíku *thewebse-*

mantic.binding, která po bindingu modelu poskytuje veškeré základní operace nad modelem. Finální zápis vytvořeného modelu do souboru ve formátu XML je možný prostřednictvím metody *write* nad instancí modelu nebo voláním metody *outFile* se String parametrem názvu výstupního souboru ze třídy *Bean2Rdf*, která byla doimplementována. Následující příklad názorně ukazuje zápis dat POJO objektů uložených v ArrayListu *objectList* do ontologického modelu a jeho následný export do souboru.

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

Jenabean.instance().bind(m);

// zápis jednotlivých POJO objektů do modelu
for (int i = 0; i < objectList.size(); i++) {
      Jenabean.instance().writer().save(objectList.get(i));
}

// exportování modelu do souboru
Jenabean.instance().writer().outFile("outputFile.xml");
```

4.3 Java anotace

Programovací jazyk Java umožňuje od verze 1.5.0 zapisovat do zdrojových kódů anotace. Anotace jsou formou metadat, která poskytují informace o programu, jež nejsou jeho přímou součástí [9]. Jedná se o zápis, který umožňuje přiřadit jednotlivým elementům ve zdrojovém kódu přídavné informace, aniž by byly přímou součástí kódu a mohly tak přímo ovlivňovat jeho činnost. Anotace mají široké spektrum užití, v zásadě lze ale rozlišit tři základní případy [11].

- Dodání přídavných informací kompilátoru. Překladač může tyto informace využívat k detekci chyb nebo potlačení varování.
- Dodatečné informace pro zpracování dat při kompilaci. Některé softwarové nástroje využívají anotací jako dodatečných informací pro nastavení generování dalšího kódu, jako například při vytváření XML souborů.

- Informace zpracovávané za běhu. Některé anotace je možné získávat a využívat při běhu programu.

Zápis anotací se provádí při deklaraci tříd, metod, proměnných a dalších elementů programu. Anotace je možné používat jako samostatné – bez dalších uvedených parametrů, ale i s jedním nebo více. Příkladem první možnosti je *@override* anotace nad deklaracemi elementů překrývajícími elementy rodičovských tříd. Následující příklad ukazuje použití anotace v případě více elementů [11].

```
@Author(
   name = "Benjamin Franklin",
   date = "3/27/2003"
)
class MyClass() { }
```

Pokud by byl definován pouze jeden atribut s názvem „value“, není nutné uvádět jeho název a je možné využít jednoduššího zápisu níže:

```
@SuppressWarnings("unchecked")
void myMethod() { }
```

Aby bylo možné vytvářet vlastní anotace, je nutné nejdříve vytvořit anotační typ. Jedná se v podstatě o definici rozhraní, kde je nutné před klíčové slovo interface uvést znak *@*.
Následující příklad ukazuje možné řešení [11].

```
@interface ClassPreamble {
   String author();
   String date();
   int currentRevision() default 1;
   String lastModified() default "N/A";
   String lastModifiedBy() default "N/A";
}
```

Deklarace jednotlivých metod anotačního rozhraní musí být bez parametrů a jejich návratové hodnoty jsou omezeny na primitivní datové typy, *String,*

Class, enum a pole všech předchozích typů. Metodám lze také nastavit výchozí hodnoty, čehož lze využít, pokud se hodnota elementu v případě použití anotace mění jen zřídka. Pokud je hodnota elementu při anotaci explicitně zadána, musí se jednat o konstantu, která bude kompilátoru dostupná v čase překladu, v opačném příkladě je vyhozena výjimka. Při deklaraci anotačního rozhraní je také možné uvést takzvané meta anotace sloužící ke specifickému omezení použití anotace [9].

První meta anotací je *@Retention*, která slouží k definici dostupnosti anotace. Jejím elementem může být [10]:

- *RetentionPolicy.SOURCE* - dostupnost pouze ve zdrojovém kódu
- *RetentionPolicy.CLASS* - dostupnost pro kompilátor
- *RetentionPolicy.RUNTIME* - dostupnost virtuálnímu stroji za běhu programu

Druhou meta anotací je *@Target*, která svým elementem určuje omezení typů elementů, ke kterým může být anotace přidána. Těch existuje následujících 8 [10]:

- *ANNOTATION_TYPE* - anotace pro anotace
- *CONSTRUCTOR* - anotace konstruktoru
- *FIELD* - anotace pole
- *LOCAL_VARIABLE* - anotace lokální proměnné
- *METHOD* - anotace metody
- *PACKAGE* - anotace Java balíku
- *PARAMETER* - anotace parametru
- *TYPE* - anotace třídy nebo rozhraní

Následující příklad deklarace rozhraní definuje anotace dostupné za běhu virtuálnímu stroji a použitelné k anotaci metod [9].

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test { }
```

4.4 Anotace nástroje Jenabean

Jenabean akceptuje 7 druhů možných anotací, kterými je možné obohatit požadovanou ontologii transformovaných dat [1].

@Namespace anotace slouží ke změně implicitního jmenného prostoru pro celou třídu a zároveň se projeví u všech podřízených atributů a metod. Obsahuje jeden element datového typu String nesoucí jméno jmenného prostoru. Cílem je správně ontologicky zařadit vytvořenou třídu do systému.

@RdfType je anotace sloužící ke změně implicitního názvu třídy na hodnotu definovanou v elementu datového typu String uvedené anotace. Název je přenesen i na ontologie podřízených atributů a metod. V případě užití samotné anotace *@RdfType* je v ontologii zachován implicitní jmenný prostor. Anotaci je možné použít zároveň v kombinaci s předchozím *@Namespace* za účelem změny názvu třídy i jmenného prostoru zároveň.

Z následující příkladu je patrný praktický zápis prvních dvou uvedených anotací [1].

```
@Namespace("http://xmlns.com/foaf/0.1/")
@RdfType("Person")
public class Fellow {

        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
}
```

@RdfProperty anotace umožňuje společnou změnu jména i jmenného prostoru jednoho atributu třídy pomocí jediné anotace. Slouží pro určení jednoznačné ontologické vazby, např. na jiného rodiče. Příklad možného zápisu [1]:

```
public class ExampleBean {

@RdfProperty
("http://semanticbible.org/ns/2006/NTNames#parentOf")
public Collection<Human> children;

}
```

@Id je anotace, jež slouží k vygenerování unikátního klíče k atributům nebo metodám třídy, který je použit u názvu ontologie. Není vzájemně kombinovatelná s předchozí anotací. Jedná se o jedinou vyžadovanou anotaci, pokud není použita anotace *@RdfProperty*. Anotace nemá žádný element. Její použití je patrné z níže uvedeného příkladu [1].

```
public class ExampleBean {

@Id
private int myid;

}
```

@Symmetric anotace umožňuje nastavit elementu vlastnost *owl:SymmetricProperty*. Tato symetričnost v jazyce OWL znamená, že pokud dvojice prvků *(x,y)* je instancí symetrické třídy T, pak i dvojice prvků *(y,x)* je instancí třídy T [22]. Prakticky to znamená, že pokud by například osoba A byla přítelem osoby B, pak je i osoba B přítelem osoby A. Tuto anotaci je možné použít jak pro třídu, tak i pro její atributy. Anotace nemá žádný atribut a její použití je tak analogické jako v případě anotace *@ID*.

@Transitive je anotace, která cílovému elementu nastaví vlastnost *owl:TransitiveProperty*. To znamená, že pokud dvojice prvků *(x,y)* je instancí tranzitivní třídy T a zároveň dvojice prvků *(y,z)* je také instancí třídy T, pak i dvojice prvků *(x,z)* je také instancí třídy T [22]. Tuto vlastnost lze uvést na následujícím příkladu [16]: Pokud třída savec je podtřídou třídy zvíře a zároveň zvíře je podtřídou třídy organismus, pak i savec je podtřídou třídy organismus. Anotace je stejně jako v předchozím případě bez atributu a je ji možné využít pro třídu i její elementy.

@Inverse anotace nastavuje vybranému elementu vlastnost *owl:InverseOf*. V jazyce OWL je možné objektům i vlastnostem přiřadit inverzní prvek. Použití lze demonstrovat na příkladu: Osoba vlastní auto. Vlastnosti *vlastní* je možné určit inverzní vlastnost *má majitele*, která je vlastností objektu *auto*. *@InverseOf* je zároveň symetrická, což ve výsledku usnadňuje hledání vztahů mezi jednotlivými objekty a vztah mezi objekty auta a osoba je tak jasně definován [22].

Tato anotace ale není funkční, a to ani v poslední verzi Jenabean 1.0.6. Vzhledem k této skutečnosti a faktu, že poslední verze nástroje byla uvolněna v únoru 2010, je nutné její opravu provést.
Nástroj obsahuje anotační rozhraní *Inverse*, které má jediný atribut sloužící k definici URI identifikátoru inverze cílového elementu. Použití této anotace se ale nijak ve výstupních datech neprojevuje. Nově navržená a implementovaná úprava tedy zajišťuje načtení hodnoty atributu anotace a tato hodnota je pak zpracována a předána jako URI identifikátor hodnotě inverze cílového elementu, která je vytvořena pro příslušnou třídu či její atribut nesoucí tuto anotaci. Výsledné použití anotace je pak patné z níže uvedeného příkladu.

```
public class ExampleBean {

@Inverse
("http://semanticbible.org/ns/2006/NTNames#hasOwner")
public String hasCar;

}
```

Výstup OWL API:

```
<InverseObjectProperties>
  <ObjectProperty IRI="http://www.kiv.zcu.cz/hasCar"/>
        <ObjectProperty IRI="http://semanticbible.org/ns/
         2006/NTNames#hasOwner"/>
</InverseObjectProperties>
```

4.5 Rozšíření nástroje Jenabean

Jazyk OWL dle své specifikace [22] umožňuje dodávat datům více ontologií a sémantiky, než které nástroj Jenabean umí přidávat za použití Java anotací

nad POJO třídami. Těchto možností jazyka OWL využívá framework Jena, který je schopen data obohatit o většinu sémantiky specifikovanou v jazyce OWL užitím prostředků svého OWL API. To je zároveň nutný předpoklad potřebný pro úpravu nástroje Jenabean tak, aby mohl užitím anotací také dodávat sémantiku transformovaným datům, protože tento nástroj sám nevytváří žádný vlastní model, ale pouze pracuje s ontologickým modelem Jeny.

Na základě těchto skutečností byl zadavatelem vytvořen seznam požadovaných anotací, prostřednictvím kterých by mělo být možné dodávat datům požadovanou sémantiku v souladu se specifikací jazyka OWL. Názvy všech vytvořených anotací jsou zvolené tak, aby z nich bylo zřejmé, k jakým úpravám ve vzniklém OWL souboru bude jejich užitím docházet. Veškeré provedené úpravy nástroje jsou doprovázeny Javadoc komentáři popisujícími účel změn. To umožňuje jejich snadné vyhledání, neboť nástroj ve své původní verzi (až na několik výjimek) žádný Javadoc neobsahuje.

@DataRange je anotace, umožňující nastavit omezení datového typu pro vybraný atribut třídy. Jazyk OWL poskytuje vlastnost *rdfs:range*, která slouží k omezení rozsahu dat.

OWL určuje datový typ referencí na existující XSD (XML Schema Datatype – schémový jazyk XML dokumentů) datové typy. V případě použití restrikce více různými datovými typy je výsledek interpretován jako průnik rozsahů těchto typů [22]. Definice vlastních datových typů, které by dovolovaly přesnější vymezení dat, jako například určení maximální či minimální hodnoty, jsou ale možné až v jazyce OWL 2 [6]. Nástroj OWL API, který provádí poslední krok transformace dat, sice tento jazyk podporuje, ale framework Jena nikoliv. Jena poskytuje prostřednictvím své třídy XSD datové restrikce na všechny primitivní datové typy XSD. Nástroj Jenabean je tak možné upravit, aby dovoloval přidání restrikce datového typu pro příslušný element, ale pouze referencí na existující XSD datový typ. XSD šablony dovolují vytvářet nové datové typy restrikcí či skládáním základních datových typů. Na tuto šablonu je pak sice možné při určení datového typu se odkázat, ale vzniká tím problém s přenositelností vzniklého OWL dokumentu, neboť je nutné vytvořenou šablonu buď zveřejnit nebo ji neustále přenášet spolu s dokumentem.

Vytvořená anotace tak má jeden atribut typu String, kterým se elementu určí restrikce buď na primitivní datový typ XSD uvedením jeho názvu, nebo uživatelský typ URI identifikátorem adresy vytvořené XSD šablony.

Při uvedení názvu XSD primitivního datového typu je nutné uvést jeho přesný název, s výjimkou typů *boolean, byte, double, float, int, long, short* a *String*, kde je nutné názvu předřadit prefix *x*, neboť se jedná o klíčová slova jazyka Java a implementace této anotace provádí dynamické čtení atributů tříd Jeny s příslušnými názvy. Například pro datový typ float je tak nutné uvést hodnotu *xfloat*. Použití anotace je patrné z níže uvedeného příkladu.

```
public class Person {

@DataRange("nonNegativeInteger")
public int age;

}
```

Výstup OWL API:

```
<DataPropertyRange>
  <DataProperty IRI="http://www.kiv.zcu.cz/age"/>
  <Datatype abbreviatedIRI="xsd:nonNegativeInteger"/>
</DataPropertyRange>
```

@VersionInfo anotace dovoluje vybranému elementu přidat vlastnost *owl:versionInfo*. Jazyk OWL umožňuje přidávat objektům anotaci, která nese textovou informaci o verzi příslušného objektu. Jediný atribut typu String této anotace slouží k přímému zapsání informace. Nejedná se tedy o žádný URI identifikátor, jako v případě předchozích anotací. Každý objekt může nést pouze jednu informaci o své verzi, a proto při vícenásobném použití anotace bude uchována pouze poslední hodnota. Anotaci je možné využít pro třídu i její atributy, jak je patrné z následujícího příkladu.

```
public class Measuration {

@VersionInfo("Version 1.1")
private Scenario scenario;

}
```

Výstup OWL API:

```
<AnnotationAssertion>
  <AnnotationProperty IRI="versionInfo"/>
  <IRI>http://www.kiv.zcu.cz/scenario</IRI>
  <Literal>Version 1.1</Literal>
</AnnotationAssertion>
```

5 Nástroj pro anotování POJO tříd

V předchozí kapitole byly popsané Java anotace nástroje Jenabean, jejichž užitím je možné obohacovat transformovaná data o další sémantiku, kterou není možné uchovávat v relační databázi z důvodu její nízké vyjadřovací schopnosti. Všechny tyto anotace musejí být zapisovány do POJO tříd, jež slouží jako persistentní objekty. Jejich zápis se provádí v místě deklarace samotných tříd, respektive jejich atributů, v závislosti na konkrétních typech anotace. Tato skutečnost klade na uživatele požadavek ručně editovat jednotlivé třídy, korektně zapsat příslušné anotace a následně provést jejich překlad. Samotnou transformaci je tedy možné provést až po těchto krocích.

Vzhledem k tomu, že cílem této práce je požadovanou transformaci co nejvíce zautomatizovat tak, aby kladla na uživatele minimální nároky co se znalostí použitých nástrojů týče, je nutné zbavit ji požadavku na ruční editaci zdrojových kódů POJO. Nejjednodušším řešením, které se intuitivně nabízí, by bylo načítání hodnot atributů anotací z externího souboru, jehož struktura by dovolovala snadné modifikace na jediném místě. Toto řešení ale není realizovatelné, neboť jazyk Java vyžaduje, aby hodnoty atributů anotací byly překladači známé již v době překladu, nikoliv až za běhu, kdy čtení externích souborů probíhá. Částečné řešení tohoto problému poskytuje nástroj Annotation File Utilities.

5.1 Annotation File Utilities

Annotation File Utilities je open source nástroj vyvinutý v jazyce Java, který umožňuje přidávat anotace Java třídám [17]. Poslední uvolněná distribuce obsahuje skripty pro operační systém Windows i Linux sloužící k ovládání tohoto nástroje prostřednictvím příkazového interpretu. První skript s názvem *insert-annotations* provádí vložení anotací do přeložené třídy, jejíž název je uveden jako první parametr a druhým parametrem je soubor obsahující specifikaci vkládaných anotací. skript *insert-annotations-to-source* slouží ke stejnému účelu, ale anotace přidává přímo do nepřeloženého zdrojového souboru, který je opět uveden jako první parametr. Třetí a poslední skript *extract-annotations* extrahuje veškeré anotace přeložené třídy, jejíž název je uveden jako jediný parametr, do souboru shodného jména s příponou *.jaif*. Soubor „jaif“ (Java anotation index file), který je vstupním parametrem

prvních dvou skriptů, slouží k definici vkládaných anotací a určení jejich přesného umístění v cílové třídě. Z dodávané dokumentace včetně mnoha ukázkových příkladů je patrné, že nástroj je velmi komplexní a umožňuje vkládat jakékoliv anotace do libovolného místa třídy v souladu se syntaxí jazyka Java. Tato komplexnost s sebou ale zároveň přináší složitější zápis „jaif“ souboru, který musí mít v záhlaví uvedené definice všech použitých anotací. Ty představují analogii k anotačním rozhraním Javy, takže je nutné uvést název anotace, její atributy včetně datových typů a také dostupnost (pro kompilátor či virtuální stroj). Na dalších řádcích souboru se provádí zápis cesty k elementům, které mají být anotovány, tedy název Java balíku, třídy a případně cílové metody či atributu. Posledním údajem je pak název samotné anotace včetně uvedení hodnot atributů (pokud nějaké má).

Jeden „jaif“ soubor tak může obsahovat definice vkládaných anotací pro více tříd, což by umožňovalo anotovat POJO třídy úpravou jediného souboru, ale stále je nutné opakovaně spouštět patřičný skript pro každou třídu a uživatel by také musel být obeznámen se syntaxí „jaif“ souboru, která není zcela triviální, jak je patrné z níže uvedeného příkladu.

```
package thewebsemantic:

annotation @RdfProperty:
@Retention(value=RUNTIME)
@java.lang.annotation.Target(value={TYPE_USE})
String value

package data.pojo:
class Person:
field surname:
@thewebsemantic.RdfProperty("http://www.kiv.zcu.cz/prijmeni")
```

Výše uvedený příklad ukazuje přidání anotace *RdfProperty* (včetně parametru) atributu *surname* třídy *Person*. Všechny tři skripty ve svém kódu volají knihovnu *annotation-file-utilities.jar*, která provádí samotnou operaci, a skripty jí předávají potřebné parametry. Díky této skutečnosti se otevírá možnost navržení programu, který by využíval zmíněnou knihovnu k anotování POJO tříd takovým způsobem, že by na základě požadavků uživatele přidával anotace akceptované nástrojem Jenabean vybraným třídám. Jenabean akceptuje pouze konečné množství anotací nad atributy a třídami, které by nástroj

znal a uživatel by z nich pouze vybral ty, které požaduje, zvolil cílový atribut a třídu a nástroj by vygeneroval potřebný „jaif“ soubor a prostřednictvím zmíněné knihovny anotace dodal. Realizaci toho nástroje popisuje následující podkapitola.

5.2 Annotation tool

Cílem navrhnutého a následně implementovaného nástroje je zajištění jednoduchého dodávání anotací již přeloženým POJO třídám s minimálními nároky na uživatelovy znalosti dané problematiky. Nástroj musí nabídnou seznam všech dostupných anotací, které Jenabean dokáže zpracovat, z nichž uživatel požadované vybere a rozhodne o jejich umístění. Seznam dostupných anotací musí být modifikovatelný, tak aby umožnil přidávat další nové anotace pro případ rozšíření Jenabeanu o schopnost zpracovávat další anotace. Vybranou konfiguraci anotací je také nutné ukládat, tak aby bylo možné ji později jednoduše upravit. Pro zajištění jednoduchého ovládání je potřeba vytvořit grafické uživatelské rozhraní, které bude celý nástroj řídit.

5.2.1 Analýza problému

Celý nástroj pracuje s přeloženými POJO třídami, které je nutné nejprve do aplikace načíst. Vzhledem k tomu, že upravovaných tříd může být větší počet, je zapotřebí umožnit jejich načítání v podobě jak jednotlivých souborů, tak i JAR (Java Archive [12]) souboru, ve kterém budou všechny třídy zabaleny. Protože JAR soubor, který primárně slouží jako snadno distribuovatelný archiv tříd pro vytvářené Java aplikace, je pouze adresář komprimovaný ve formátu ZIP, je možné využít standardní knihovny Javy k jeho extrakci. Všechny načítané POJO třídy je pak nutné zkopírovat respektive extrahovat z JAR archivu do pracovního adresáře aplikace, aby další operace nad těmito soubory mohly být jednotné. V dalším kroku je potřeba určit seznam názvů atributů jednotlivých tříd, neboť především ty budou anotovány. Názvy samotných tříd je možné převzít z názvů souborů, neboť dle specifikace Javy se musejí shodovat s názvy odpovídajících souborů. Seznam atributů je možné získat využitím Java Reflection API [13] (dále jen reflexe), které umožňuje načítat třídy za běhu aplikace a získávat pak o nich detailní informace. Poslední data, která jsou nezbytná pro funkčnost nástroje, tvoří seznam přípustných anotací. Ten je potřeba rozdělit na dvě části – anotace pro třídy

a pro jejich atributy. Seznamy budou načítány ze dvou textových souborů, respektující toto rozdělení. Ty ponesou pouze seznam názvů anotací včetně příznaků zda, mají parametr. Tím se docílí jednoduché modifikovatelnosti seznamu dostupných anotací.

Aby bylo možné se všemi načtenými daty efektivně pracovat, je zapotřebí vytvořit datovou vrstvu, která bude všechna data uchovávat. Ta bude mít hierarchickou strukturu, stejně jako elementy Java třídy, což zajistí dostatečnou přehlednost uchovávaných dat. Pro každou načtenou třídu pomocí reflexe získáme seznam atributů. Každému z nich pak přiřadíme seznam dostupných anotací (získaný z výše zmíněného souboru). Totéž se učiní i pro samotné třídy. Jednotlivé anotace ponesou své jméno, příznak zda má být k příslušnému atributu či třídě přiřazena a hodnotu anotačního parametru (pokud jej má). Všechna data pak budou zabalena do jednoho objektu, který ponese seznam tříd, které budou obsahovat seznam svých anotací a atributů, které rovněž ponesou svůj seznam anotací.

Poté, co uživatel vybere a nastaví požadované anotace, je zapotřebí provést jejich zápis do příslušných POJO souborů. Ty jsou uložené v pracovním adresáři aplikace a jsou jedním ze vstupů knihovny *annotation-file-utilities.jar*, jež bude samotný zápis provádět. Druhým nutným vstupem je „jaif“ soubor, který je potřeba na základě požadovaných anotací vygenerovat. V prvním kroku se musí získat z datové vrstvy seznam všech přidávaných anotací, a to v takovém formátu, aby z nich bylo možné jednoduchým sekvenčním čtením generovat „jaif“ soubor. Musí tedy obsahovat kromě svého názvu a případné hodnoty i cílovou třídu a atribut (pokud nejde o anotaci třídy), kam má být zapsána, což lze snadno určit díky hierarchii dat. Z tohoto seznamu se pak získá další seznam všech neopakujících se názvů anotací (včetně informace, zda mají parametr), který bude použit pro vygenerování hlavičky „jaif“ souboru, jež nese definice přidávaných anotací. Tuto hlavičku lze jednoduše sestavit, protože jednotlivé anotace se liší pouze svými názvy a přítomností parametru, který je vždy typu String, a zbylé části lze považovat za konstantní řetězce. Samotný zápis cílů konkrétních anotací a případných hodnot se pak provede sekvenčním načtením všech potřebných dat ze seznamu přidávaných anotací a jejich vložením mezi klíčová slova „jaif“ souboru, které lze opět chápat jako konstantní řetězce. Po sestavení obsahu „jaif“ souboru se provede následné volání hlavní metody anotační knihovny, a to opakovaně pro každou modifikovanou třídu, přičemž v každé iteraci bude její název uveden jako parametr. Jako poslední krok je zapotřebí zkopírovat všechny třídy (včetně neanotovaných) do uživatelem zadaného umístění, čímž je dosažen požadovaný výstup tohoto nástroje.

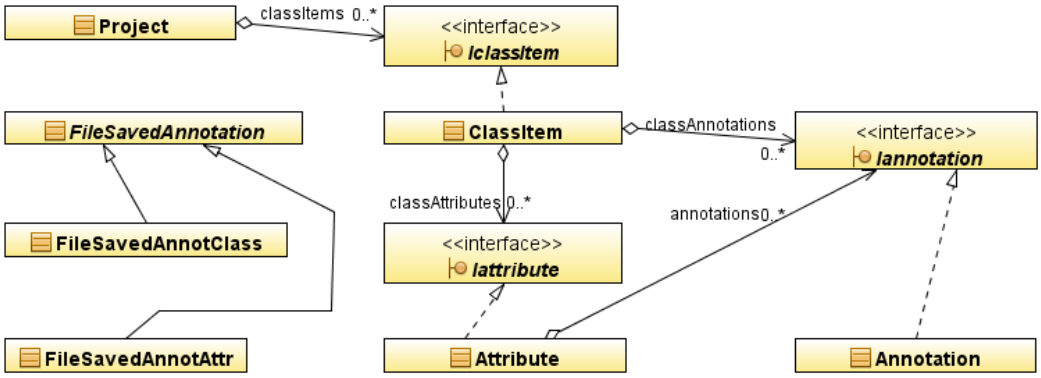
Vzhledem k tomu, že anotační knihovna umí provádět pouze zápis anotací do souborů, ale neumožňuje již jejich modifikace a odstraňování, je nutné před jejím prvním voláním původní POJO soubory zkopírovat do dalšího adresáře, který poslouží k uchování jejich původní podoby. Z něj budou vždy před samotným zápisem anotací tyto soubory zkopírovány do pracovního adresáře, kde budou anotovány. Tím se docílí, že výsledné POJO třídy budou obsahovat právě ty anotace, které uživatel vybral a žádné jiné (přidané již dříve).

Aby bylo možné po ukončení aplikace a jejím opětovném spuštění navázat na provedené úpravy anotací, případně je měnit, je nutné ukládat na disk všechny tyto údaje. Zároveň může vzniknout požadavek na úpravu jiné sady tříd, přičemž provedené změny na předchozích třídách je také potřeba zachovat. Nejjednodušším řešením celého problému je ukládat všechna data – tedy upravované POJO třídy a uživatelem nastavené anotace na disk zabalené do jednoho souboru jako projekt. Ten by pak bylo možné opětovně načíst a pokračovat v započatých změnách. Aby nedocházelo k ukládání redundantních dat, bude se vytvářet soubor, který ponese jen uživatelem vybrané a nastavené anotace včetně jejich cílového umístění, neboť ostatní data je možné opětovně získat reflexí. Soubor se vytvoří na základě seznamu přidávaných anotací, získaného stejným způsobem jako při generování „jaif“ souboru. Tento soubor se spolu se všemi POJO třídami z pracovního adresáře aplikace a soubory obsahujícími seznam přípustných anotací zabalí do ZIP archivu a uloží jako projekt na disk pod uživatelem definovaným jménem.

Posledním problémem, který je zapotřebí vyřešit, je otevírání uložených projektů. To bude probíhat analogicky jako načítání dat z JAR archivu při vytváření nového projektu s tím rozdílem, že je ještě potřeba načíst uložené změny (vybrané anotace). Po rozbalení archivu se všemi soubory do pracovního adresáře aplikace a načtení všech základních dat je nutné načíst soubor s uloženými anotacemi. Sekvenčním čtením se pak pro každou anotaci najde v datové vrstvě příslušná třída případně atribut a anotace se upraví na požadovanou hodnotu. Uložené soubory seznamů dostupných anotací pro třídy a atributy budou sloužit jako ochrana proti nekonzistenci dat v případě, že by se v uloženém projektu vyskytovaly anotace, které budou následně v seznamu těch základních, programu dostupných odstraněny. Při otevírání projektu se tak při získávání seznamu dostupných anotací budou načítat jak soubory určené pro samotnou aplikaci, tak i ty uložené v projektu. Ze všech získaných anotací se vytvoří jejich sjednocení, které bude tvořit nový seznam dostupných anotací pro aktuální projekt. Díky tomu bude možné ve starém projektu používat jak původní anotace, tak i nově přidané.

5.2.2 Popis implementace

Vytvořený nástroj je koncipován jako třívrstvá architektura, což je patrné ze struktury Java balíků. Veškerá zpracovávaná data jsou uchovávána a zpřístupňována prostřednictvím třídy *Project*. Ta nese list tříd *ClassItem*, jejichž jednotlivé instance představují jednu POJO třídu, která je anotována. Zde je uloženo jméno třídy, dále cesta k fyzickému souboru (potřebná k pozdější modifikaci) a listy tříd *Annotation* a *Attribute*. Třída *Annotation* reprezentuje jednu konkrétní anotaci, která je vázána na určitý element, tedy třídu nebo její atribut. Obsahuje název anotace, příznak, zda má být přidána a zadanou hodnotu atributu – pokud se jedná a parametrickou anotaci. V případě neparametrické anotace (což je rozlišeno dvoustavovou proměnnou) je tato hodnota prázdná. Třída *Attribute* pak reprezentuje jeden atribut POJO třídy, přičemž nese jeho název a list přítomných anotací reprezentovaných opět třídou *Annotation*. Datová vrstva obsahuje ještě abstraktní třídu *FileSavedAnnotation* a její dva potomky rozlišující, zda se jedná o anotaci atributu nebo třídy. Ty slouží k předávání jednotlivých anotací částem aplikace, které generují soubor „jaif“ a ukládají uživatelem vybrané konfigurace anotací jednotlivých tříd při ukládání projektu. Tato třída obsahuje, na rozdíl od třídy *Annotation*, všechny informace potřebné k zápisu anotace do „jaif“ souboru, tedy jméno třídy, atributu (pokud se jedná o anotaci atributu), anotace a její hodnotu, která je opět prázdná v případě neparametrické anotace. Celá struktura datové vrsty je patrná z UML diagramu na obrázku 5.1.



Obrázek 5.1: UML diagram datové vrstvy

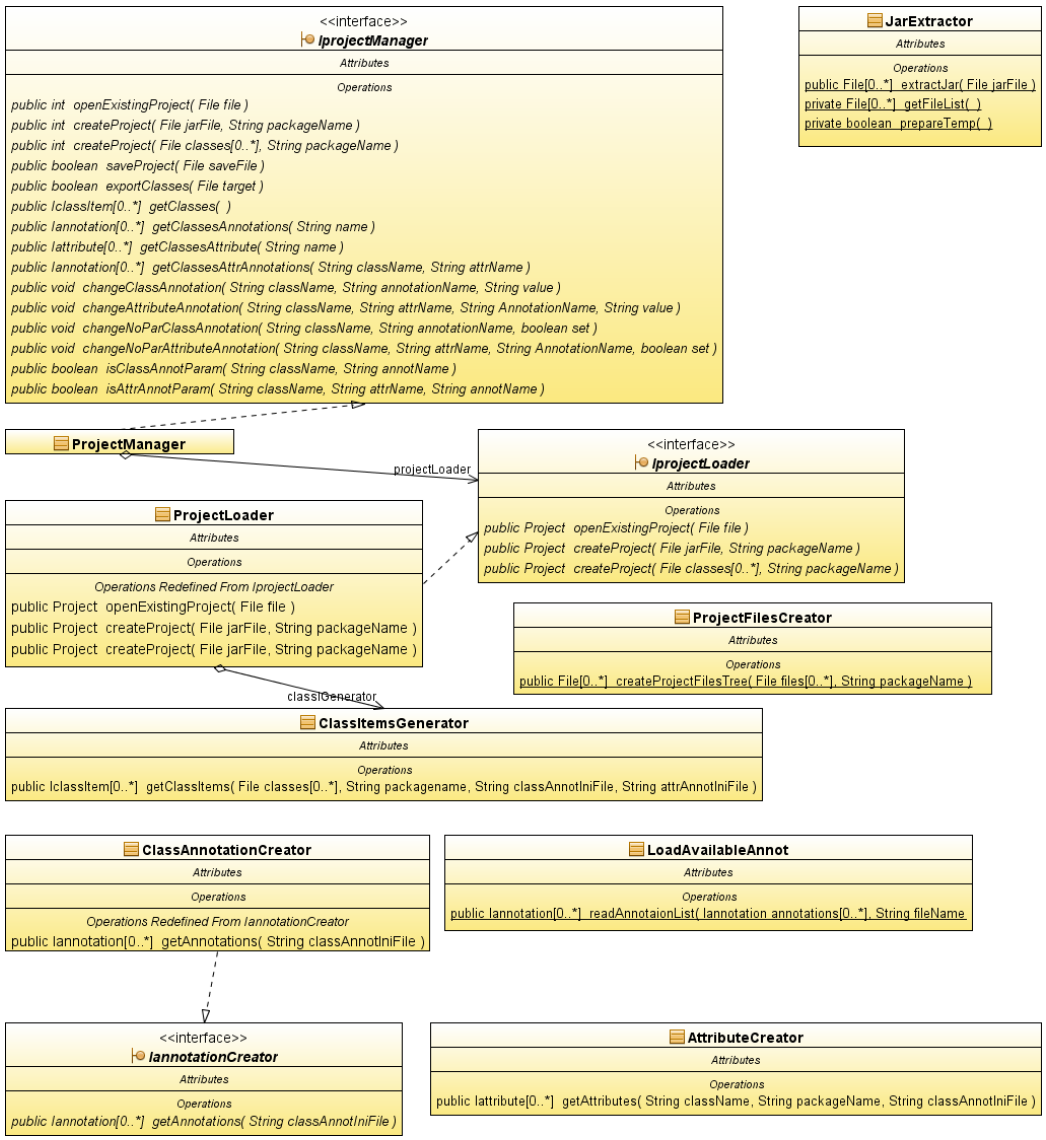
Aplikační vrstva sestává z většího množství tříd, přičemž její oddělení od prezentační vrstvy je realizováno prostřednictvím rozhraní *IprojectManager*. Tento interface definuje všechny metody jak k úpravám jednotlivých anotací projektu, tak k samotnému vytváření, otevírání a ukládání projektu, včetně exportování anotovaných tříd. Třída *ProjectManager*, která toto rozhraní implementuje, obsahuje instanci třídy *Project*, nad kterou všechny deklarované metody operují, neboť nese všechna data projektu. Třída požadované operace neprovádí přímo, ale deleguje je na podřízené objekty, kterým předává případné parametry, a sama pak jejich prostřednictvím vrací požadovaná data nebo chybový kód v případě, že zachytí výjimku z některé nižší vrstvy.

Třída *ProjectLoader* zajišťuje vyváření nových a otevírání již existujících uložených projektů. Vytvoření nového projektu poskytuje přetížená metoda *createProject*, která má v parametru buď cestu k JAR souboru nesoucímu všechny třídy, nebo list jednotlivých tříd. Druhým parametrem je název Java balíku, ve kterém byly všechny POJO třídy uložené. Jeho jméno je nezbytné pro pozdější načítání tříd reflexí. V případě, že jsou třídy uložené v JAR archivu, dojde k jejich extrahování do dočasného adresáře prostřednictvím třídy *JarExtractor*, jejíž jediná veřejná metoda vrací list s jednotlivými dekomprimovanými třídami. Dále je voláním metod třídy *ProjectFilesCreator* vytvořen pracovní adresář aplikace včetně podadresáře dle názvu Java balíku POJO tříd, do kterého jsou všechny třídy nakopírovány, aby mohly být reflexí získány jejich atributy.

Z takto umístěných souborů je již možné získat všechna potřebná data ze kterých třída *ClassItemsGenerator* sestavuje list instancí datových tříd *ClassItem*, jež tato data ponesou. Anotace samotných tříd jsou vytvářeny třídou *ClassAnnotationCreator*, která volá metodu třídy *LoadAvailableAnnot*. Ta načítá seznam dostupných anotací pro třídy ze souboru, jehož název získává v parametru, a na základě jeho obsahu sestaví list instancí třídy *Annotation*, který také vrací. List atributů sestavuje třída *AttributeCreator*, která z cílové třídy načte pomocí reflexe seznam jejích atributů, na jejichž základě je vytvořen list instancí třídy *Attribute*. Každá tato instance obsahuje název atributu, který má zastupovat, a seznam anotací, který je sestaven opět metodou třídy *LoadAvailableAnnot*, tentokrát ale volanou s parametrem souboru nesoucím seznam dostupných anotací pro atributy.

Z listu instancí třídy *ClassItem* vytvoří *ProjectLoader* novou instanci třídy *Project*, která je navrácena do třídy *ProjectManager*. Všechny anotace v projektu jsou po jeho vytvoření bez hodnot parametrů a příznaky nastaveny na

hodnotu nepřidávat anotaci. Popisované vytváření projektu je také znázorněno UML diagramem na obrázku 5.2.



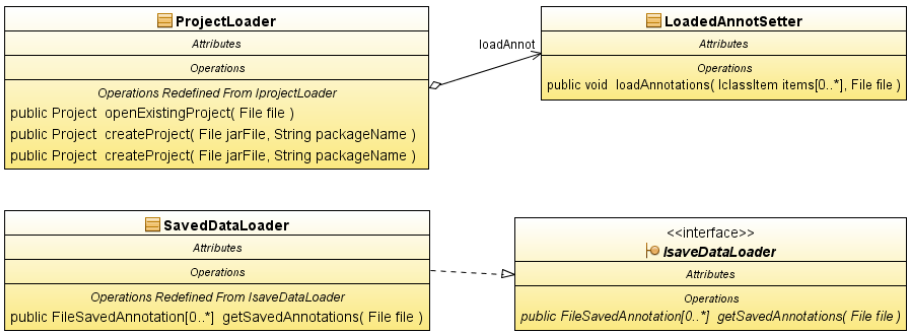
Obrázek 5.2: UML diagram tříd pro vytváření projektů

Otevírání uloženého projektu voláním metody *openExistingProject* probíhá podobně jako vytváření nového z JAR archivu. Uložený projekt je adresář komprimovaný do ZIP archivu, který nese přeložené POJO třídy a soubory *package.dat*, *annotations.dat*, *attrAnnots.ini* a *classAnnots.ini*. Poslední dva soubory obsahují seznamy přípustných anotací pro atributy a třídy, soubor

package.dat obsahuje pouze název balíku POJO tříd a *annotations.dat* nese seznam uložených anotací.

Sestavení listu datových tříd *ClassItem* probíhá stejně jako při vytváření nového projektu, jen s tím rozdílem, že jméno Java balíku není získáno jako vstup prostřednictvím uživatele, ale je načteno ze souboru *package.dat* a při sestavování listu přípustných anotací jsou načítány příslušné soubory zároveň z kořenového adresáře aplikace, i z dočasné složky nesoucí extrahovaná data projektu. Třída *LoadAvailableAnnot* vytváří z načtených souborů dva listy přípustných anotací, které sloučí do jednoho, odstraní v něm opakující se prvky a vrací jej.

Načítání uložených anotací provádí třída *LoadedAnnotSetter*. Její jediná veřejná metoda nastavuje všechny příslušné anotace v listu tříd *ClassItem*, který získává parametrem společně se souborem uložených anotací. Ten je čtený třídou *SavedDataLoader*, která na jeho základě vytvoří list instancí potomků třídy *FileSavedAnnotation*, jež nesou jednotlivé uložené anotace. Tento list je pak sekvenčně čten a pro každou položku je nalezena v předchozím listu příslušná cílová anotace třídy či atributu, u které se nastaví získaná hodnota (v případě parametrické anotace), a příznak, že má být přidána. UML diagram na obrázku 5.3. znázorňuje třídy využité při otevírání projektu.



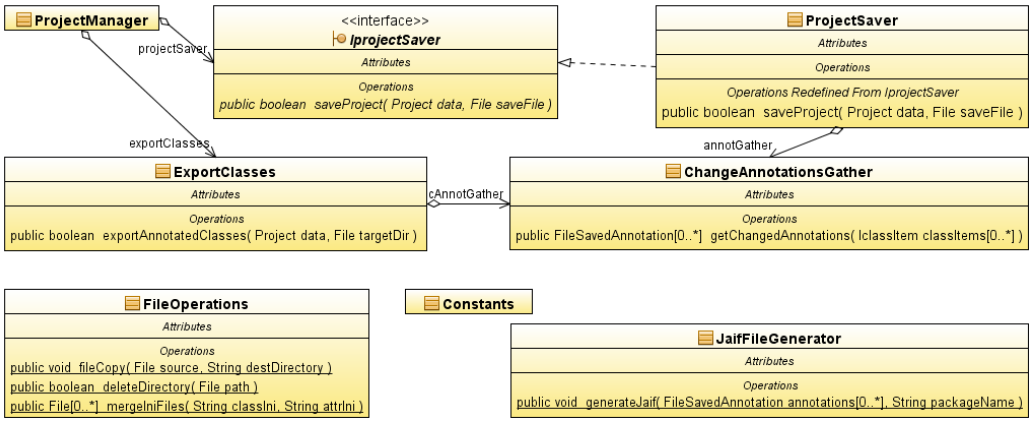
Obrázek 5.3: UML diagram tříd pro otevírání projektů

Ukládání projektů zprostředkovává třída *ProjectSaver*. Ta využívá třídu *ChangeAnnotationsGather*, která z instance třídy *Project* získá všechny anotace označené jako přidávané k vytvoření listu instancí potomků třídy *FileSavedAnnotation*. Ten nese seznam anotací, který je potřeba uložit do souboru *annotations.dat*, což realizuje metoda *saveAnnotations* třídy *AnnotationDataSaver*. Dále je vytvořen soubor *package.dat*, do kterého je zapsán název Java

balíku POJO tříd, jenž je uchováván ve třídě *Project*. Předchozí dva soubory jsou spolu s POJO třídami a soubory *attrAnnots.ini* a *classAnnots.ini* předány metodě *createJar* třídy *JarCreator*, která provede jejich zazipování do souboru na disk, jehož název volí uživatel.

Samotné modifikace POJO tříd dle vybraných anotací zajišťuje třída *ExportClasses*. Ta prostřednictvím třídy *ChangeAnnotationsGather* získá list instancí potomků třídy *FileSavedAnnotation*, který nese všechny požadované anotace. Z toho listu generují metody třídy *JaifFileGenerator* klíčový „jaif“ soubor, přičemž nezbytné konstantní řetězce získají ze třídy *Constants*. Z listu požadovaných anotací je vytvořen seznam tříd, které je potřeba o tyto anotace doplnit. Dále je v pracovním adresáři aplikace vytvořen podadresář, do kterého se zkopírují všechny POJO třídy v původní podobě bez anotací. To realizuje třída *FileOperations*, která zároveň před každým provedením samotného anotování těmito soubory přepíše soubory v původním umístění. Díky tomu jsou vždy anotovány původní soubory, které tím pádem budou mít právě ty anotace, které uživatel vybral a žádné jiné (dodané předchozím anotováním). Samotné anotování pak probíhá cyklickým voláním hlavní metody knihovny *annotation-file-utilities.jar*, které jsou předány v parametru „jaif“ soubor a cílová POJO třída.

Po dokončení poslední iterace se všechny třídy projektu (včetně neanotovaných) zkopírují do umístění zvoleného uživatelem, což zajišťuje opět třída *FileOperations* prostřednictvím svých statickým metod. Obrázek 5.4. obsahuje UML diagram tříd využívaných k ukládání projektu a exportování anotovaných tříd.



Obrázek 5.4: UML diagram tříd pro ukládání projektu a export dat

Veškerá nastavení a čtení hodnot anotací poskytuje třída *ProjectManager* voláním příslušných metod nad instancemi objektů datové vrsty.

Prezentační vrstva celé aplikace je realizována prostřednictvím knihoven AWT a Swing. Třída *MainWindow* slouží k zobrazení hlavního okna aplikace, které obsahuje základní menu. Na hlavním panelu okna se pak zobrazuje seznam atributů jednotlivých tříd včetně náhledu jejich anotací, který je ohraničen listem nesoucím název třídy. Třída *MainWindow* zároveň nese instanci třídy *Project*, jež předává dalším třídám posluchačů položek menu, které slouží k ukládání, otevírání a vytváření nových projektů, jejichž data zároveň zobrazují.

Třída *NewProjectListener* zajišťuje vytvoření nového projektu při vzniku požadavku prostřednictvím příslušné položky menu. V prvním kroku se zobrazí dotazovací okno, zda budou zpracovávané třídy vybrány jednotlivě nebo zabalené do JAR archivu. Tento archiv, respektive jednotlivé soubory jsou pak vybrány v následujícím okně. V posledním okně se zadá název Java balíku importovaných tříd. Všechna získaná data jsou předána volané metodě *createProject* instance třídy *Project*. Pokud dojde při vytváření nového projektu k nějaké chybě, je zobrazena odpovídající chybová hláška. V opačném případě se z vytvořeného projektu získají všechna data, která je potřeba zobrazit. To zajišťuje třída *DisplayProjectData*, která vytváří listy se seznamy atributů tříd, jejíž názvy jsou obsaženy v záhlaví každého listu. Zároveň je vytvořen jeden list nesoucí názvy všech tříd. Každý vytvořený list ještě obsahuje kromě seznamu atributů třídy, respektive názvů všech tříd, také náhled nastavených anotací u vybraného atributu, resp. třídy. Celý seznam je zobrazen jako tabulka *JTable* knihovny Swing, jejíž položky jsou obsaženy v datovém modelu třídy *JTableDataModel*. Při výběru více položek tabulky jsou v náhledu zobrazeny jen ty anotace, které jsou pro vybrané položky shodné, což testují metody třídy *CommonAnnots*. Tlačítko zobrazené pod náhledem slouží ke změně anotací všech vybraných položek, což zajišťuje třída *DisplayAnnotations*, volaná prostřednictvím třídy *ButtonChGroupListener* posluchače tlačítka.

Třída *DisplayAnnotations* zobrazuje nové okno, které obsahuje seznam názvů všech dostupných anotací pro vybrané atributy, resp. třídy. Pro každou anotaci je zobrazeno zaškrtávací políčko rozhodující, zda má být přidána, a v případě parametrické anotace také textové pole pro zadání hodnoty parametru. Dvě tlačítka v zápatí okna slouží pro uložení nebo zrušení změn. Při stisku tlačítka zrušení změn je okno pouze zavřeno. Při stisku tlačítka uložení změn se cyklicky otestují stavy všech zaškrtávacích políček a dle jejich

hodnot se v datech projektu nastaví hodnoty příznaků anotací pro přidání respektive i hodnoty parametrů získané z odpovídajících textových polí v případě parametrických anotací.

Třída *OpenProjectListener* umožňuje otevření již existujícího uloženého projektu při uživatelském výběru položky menu hlavního okna. Při obsluze tohoto požadavku se zobrazí okno pro výběr souboru uloženého projektu. Cesta k tomuto souboru je předána jako parametr volané metodě *openExisting-Project* instance třídy *Project*. Pokud není soubor poškozený, což by způsobilo vyvolání okna s chybovou hláškou, dojde k zobrazení veškerých dat stejným způsobem jako při vytváření nového projektu.

Ukládání otevřeného či nově vytvořeného projektu zajišťuje třída *SaveAs-ProjectListener*, která zobrazuje okno pro výběr umístění a názvu ukládaného projektu. Získanou hodnotu pak předává volané metodě *saveProject* instance třídy *Project*, která samotné uložení provede. Přidání anotací a export anotovaných tříd je zajištěn prostřednictvím třídy *ExportClassesListener* sloužící jako posluchač příslušného tlačítka menu, která zobrazí okno pro výběr umístění anotovaných tříd. Tuto hodnotu pak předá jako parametr metodě *exportClasses* instance třídy *Project*, jež provede samotné anotování tříd a jejich uložení do vybraného místa.

Podrobnější detaily o aplikaci jsou pak patrné ze zdrojových kódů programu, respektive z připojených podrobných Javadoc komentářů a z kompletního UML diagramu v příloze B. Postup při ovládání aplikace popisuje uživatelská dokumentace dostupná v příloze A.

6 Testování

V rámci této práce byla aplikace Annotation Tool navrhnuta a od základu implementována, proto je nutné provést její otestování. Pro datovou a aplikační vrstvu programu byly z důvodu jednoduchosti vybrány testy JUnit, které umožňují prověřit funkčnost jednotlivých komponent. V Datové vrstvě byly vytvořeny testy pro každou třídu s výjimkou abstraktní třídy *FileSavedAnnotation*, neboť ta je již testována prostřednictvím testů svých potomků. Pro aplikační vrstvu byly rovněž vytvořeny testy pro všechny třídy, s výjimkou třídy *Constants*, která obsahuje pouze veřejné statické konstanty, které nemá smysl testovat. Každý test slouží k testování veřejných metod a konstruktorů právě jedné třídy, podle níž je zároveň pojmenován. Vzhledem k tomu, že aplikace byla v průběhu vývoje průběžně testována metodou ladících výpisů, vytvořené testy odhalily jen několik drobných chyb, které byly následně opraveny. Testy při spuštění nad poslední verzí aplikace tedy probíhají se 100% úspěšností, což bylo také cílem.

V prezentační vrstvě aplikace JUnit testy vytvořeny nebyly, protože většina metod těchto tříd vrací nebo získává v parametrech objekty knihovny Swing, které by se tímto způsobem testovaly velmi obtížně. Z toho důvodu a ve snaze o zachycení co nejvíce chyb bylo ještě provedeno otestování běhu aplikace při zpracovávání reálných dat. Z relační databáze Portálu byly nástrojem Hibernate Tools vygenerovány a následně přeloženy POJO třídy, které slouží jako reprezentativní vstupní data.Ta byla následně importována do nového projektu, který byl opakovaně otevírán, pozměněn a opět uložen. V každé iteraci byly vyexportovány anotované třídy a následně otevřeny Class File Editorem ve vývojovém prostředí Eclipse, kde byla provedena ruční kontrola každé třídy, zda obsahuje právě ty anotace, které byly uživatelem v nástroji nastavené. Zároveň proběhla kontrola, zda se při opětovném otevření projektu zobrazí právě ty změny, které byly při jeho poslední úpravě uloženy. Tyto testy byly prováděny už při samotném vývoji aplikace a odhalily několik středně závažných chyb, které byly následně opraveny. Provedení těchto testů nad finální verzí programu už žádné chyby neodhalilo.

Díky výše uvedeným testům se podařilo úspěšně odladit stabilní, korektně pracující verzi aplikace, kterou je možné efektivně využívat k požadovanému účelu.

7 Závěr

Z první části práce, která se zabývala možnostmi reprezentace dat v prostředcích sémantického webu, plyne, že pro uchování dat portálové databáze je nejvhodnější zápis ve formátu OWL. Tento jazyk poskytuje dostatečné vyjadřovací možnosti a dovoluje zapisovat i sémantiku, kterou je požadováno datům dodávat.

Z další části, která se zabývá samotnou transformací, plyne, že nejvhodnějším způsobem převodu dat je vytvoření objektového datového modelu z portálové databáze v podobě POJO objektů prostřednictvím frameworku Hibernate, které slouží jako vstup nástroje Jenabean. Požadovaná data ve formátu OWL poté vygeneruje nástroj OWL API, jehož vstupem je právě výstup aplikace Jenabean.

Nástroj Jenabean zároveň dovoluje dodávat datům sémantiku prostřednictvím Java anotací nad elementy vstupních POJO objektů. V rámci této práce byly v nástroji opraveny nefunkční anotace a také implementovány nové, které dovolují dodávat další sémantiku, již jazyk OWL umožňuje zapsat. Opakovanou kontrolou výstupu transformace při použití anotací v POJO třídách se prokázalo, že úpravy nástroje byly provedeny úspěšně, a to i přes absenci jeho programátorské dokumentace.

Aby nebylo nutné ručně přidávat anotace jednotlivým POJO třídám a následně je překládat, byl vytvořen nástroj, který tento problém řeší. Navrhnutá a implementovaná aplikace Annotation Tool umožňuje přidávat přeloženým POJO třídám vybrané anotace jednoduše prostřednictvím grafického uživatelského rozhraní. Poslední část práce, která se zabývá testováním vytvořeného nástroje, prokázala jeho správnou funkčnost, a problém přidání sémantických informací se tak jeví jako úspěšně vyřešený.

Práci lze považovat za úspěšnou, neboť se podařilo navrhnout automatickou transformaci dat z EEG/ERP portálu do formátu OWL a to včetně možnosti jednoduchého přidávání sémantických informací výsledným datům. V práci by bylo možné dále pokračovat v implementaci dalších anotací do nátroje Jenabean či upravovat aplikaci Annotation Tool tak, aby byla schopna pracovat i se složitějšími typy anotací.

Seznam zkratek a pojmů

API Application Programming Interface, rozhraní pro programování aplikací

AWT Abstract Window Toolkit, knihovna Javy pro tvorbu grafického uživatelského rozhraní

EEG Electroencephalography, metoda vyšetření elektrické aktivity centrálního nervového systému

ERP Event Related Potential, vyšetření reakcí mozku na vnější podněty

JAR Java Archive, archivní soubor Javy založený na ZIP kompresi

NIF Neuroscience Information Framework, webové úložiště neuroinformatických dat a experimentů

OWL Web Ontology Language, ontologický jazyk sémantického webu

POJO Plain Old Java Object, třída jednoduchého objektu jazyka Java

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

ZIP souborový formát sloužící ke komprimování a archivaci dat.

Literatura

[1] *Jenabean*, 2010. http://code.google.com/p/jenabean, (přístup 5.1.2011).

[2] *Jena – A Semantic Web Framework for Java*, 2010. http://jena.sourceforge.net, (přístup 7.1.2011).

[3] *The OWL API*, 2011. http://owlapi.sourceforge.net, (přístup 26.3.2011).

[4] *Neuroscience Information Framework*, 2011. http://www.neuinfo.org, (přístup 21.4.2011).

[5] van Harmelen Grigoris, Antoniou Frank. *A Semantic Web Primer*. Massachusetts Institute of Technology, Cambridge, 2004. ISBN 0-262-01210-3.

[6] IBM. *IBM Jenabean*, 2011. http://www.ibm.com/developerworks/java/library/j-jenabean/index.html, (přístup 15.4.2011).

[7] JBoss Community. *Hibernate*, 2010. http://www.hibernate.org/, (přístup 6.1.2011).

[8] Pitner Tomáš Matulík Petr. *Sémantický web a jeho technologie*. Masarykova univerzita, 2004. http://www.ics.muni.cz/zpravodaj/articles/296.html, (přístup 5.1.2011).

[9] Oracle. *Annotations*, 2004. http://download.oracle.com/javase/1.5.0/docs/guide/language/annotations.html, (přístup 27.12.2011).

[10] Oracle. *Java Platform, Standard Edition 6 API Specification*, 2010. http://download.oracle.com/javase/6/docs/api, (přístup 27.12.2011).

[11] Oracle. *The Java Tutorials - Annotations*, 2010. http://download.oracle.com/javase/tutorial/java/javaOO/annotations.html, (přístup 27.12.2011).

[12] Oracle. *The Java Archive (JAR) File Format*, 2011. http://java.sun.com/developer/Books/javaprogramming/JAR, (přístup 4.4.2011).

[13] Oracle. *Using Java Reflection*, 2011. http://java.sun.com/developer/technicalArticles/ALT/Reflection, (přístup 27.3.2011).

[14] Hanyáš Petr. *Sémantický web - tutotiál a demonstrační příklady*. Vysoké učení technické v Brně, 2007. http://www.hanyas.net/download/soubor:bp_cesky.pdf, (přístup 6.1.2011).

[15] Svátek Vojtěch. *Sémantický web*, 2010. http://nb.vse.cz/ svatek/rzzw/seweb-prehled.pdf, (přístup 2.1.2011).

[16] Tauberer Joshua. *RDF about*, 2011. http://www.rdfabout.com/intro, (přístup 20.3.2011).

[17] University of Washington. *Annotation File Utilities*, 2011. http://types.cs.washington.edu/annotation-file-utilities, (přístup 2.4.2011).

[18] Papež Václav. *Neuroinformatická databáze a sémantický web*. ZČU, Plzeň, 2010.

[19] W3C. *RDF/XML Syntax Specification*, 2004. http://www.w3.org/TR/REC-rdf-syntax, (přístup 6.1.2011).

[20] W3C. *W3C Semantic Web Activity*, 2006. http://www.w3.org/2001/12/semweb-fin/w3csw, (přístup 8.1.2011).

[21] W3C. *OWL 2 Web Ontology Language*, 2009. http://www.w3.org/TR/owl2-overview/, (přístup 5.1.2011).

[22] W3C. *OWL Web Ontology Language*, 2011. http://www.w3.org/TR/owl-ref, (přístup 20.3.2011).

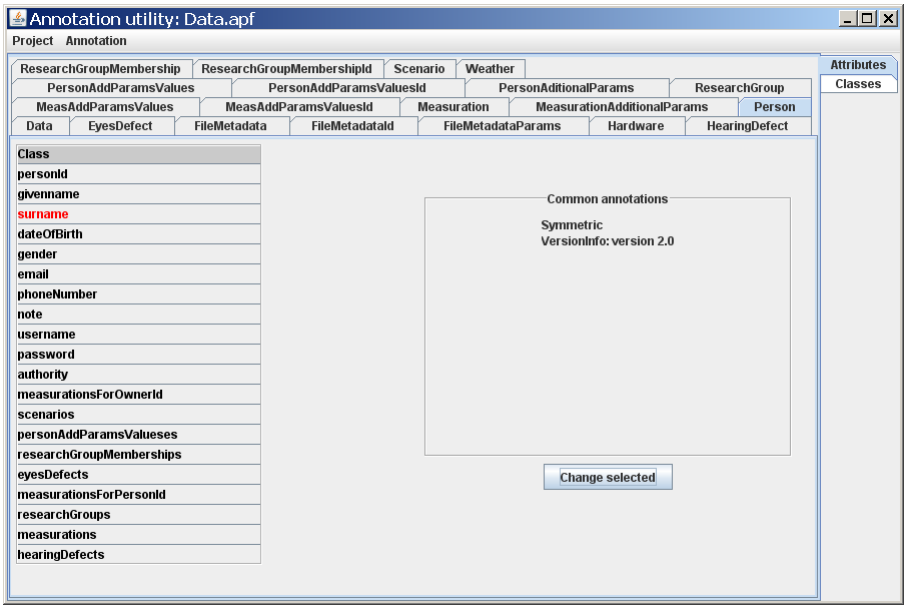
Přílohy

A Uživatelská dokumentace

Aplikace Annotation Tool je nástroj sloužící k dodávání Java anotací zkompilovaným POJO třídám. Pro sestavení a spuštění nástroje je zapotřebí Java JDK verze 6.0 či vyšší a nástroj Ant. Kořenový adresář se zdrojovými soubory obsahuje soubor *build.xml*, což umožňuje přeložit a sestavit program v příkazovém interpretu příkazem *ant*. Sestavením vznikne spustitelný soubor *AnnotationTool.jar* a dva konfigurační sobory s příponou *ini*. V souboru *classAnnots.ini* je zapsán seznam všech anotací dostupných pouze pro třídy. Každý řádek představuje jednu anotaci, přičemž musí začínat znakem *P* pro parametrickou anotaci nebo znakem *N* pro neparametrickou anotaci. Za ním následuje znak @, po kterém se uvede název anotace. Soubor *attrAnnots.ini*, jež má stejný formát zápisu, definuje seznam dostupných anotací jen pro atributy tříd. Modifikací těchto souborů se docílí změny seznamu anotací, které bude anotační nástroj uživateli nabízet.

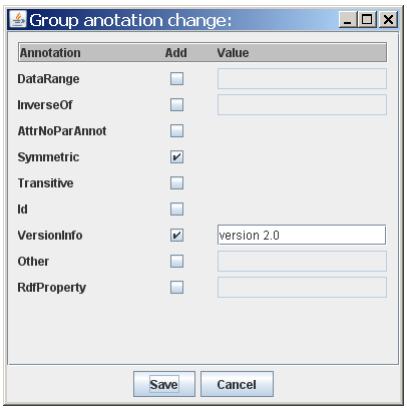
Program se spouští v grafickém prostředí operačního systému spuštěním souboru *AnnotationTool.jar* nebo z příkazového interpretu příkazem *java –jar AnnotationTool.jar*. Po spuštění aplikace se zobrazí okno, které obsahuje menu s nabídkami *Project* a *Annotation*. Položka *NewProject* nabídky *Project* slouží k vytvoření nového projektu. Uživatel je vyzván, zda chce do projektu POJO třídy importovat jednotlivě nebo zabalené do JAR archivu. V dalším okně se provede výběr požadovaných POJO tříd nebo JAR archivu v závislosti na předchozí volbě. V posledním zobrazeném okně se zadá název Java balíku, do kterého všechny importované třídy náleží. Pokud je struktura názvů balíků stromová, oddělují se názvy znakem „.“.

Položka *OpenProject* nabídky *Project* slouží k otevření uloženého projektu, který uživatel vybere prostřednictvím zobrazeného okna pro výběr souboru. Po vytvoření nového či otevření existujícího projektu se v hlavním okně aplikace (viz obrázek A.1) zobrazí seznam listů, který je přepínatelný v horní části okna, kde jsou uvedeny jejich názvy. Každý list, jehož název odpovídá jménu třídy, obsahuje seznam, jehož první položkou je jméno třídy a za ní následuje seznam všech jejích atributů. Výběrem položky seznamu se v pravé části okna zobrazí seznam jejích vybraných anotací, který je možné změnit tlačítkem *Change selected*. Pokud je v seznamu vybráno více položek, jsou v náhledu zobrazeny jen ty anotace, které mají vybrané položky společné. Tlačítkem *ChangeSelected* se pak mění anotace společně všem vybraným položkám najednou.



Obrázek A.1: Hlavní okno aplikace

Při požadavku změny anotací se zobrazí nové okno (viz obrázek A.2), které obsahuje seznam všech dostupných anotací pro dané položky. Vedle každého názvu anotace je zobrazeno zaškrtávací políčko pro označení, zda má být anotace přidána, a pokud se jedná o parametrickou anotaci, je vedle něj uvedeno textové pole, do kterého se tato hodnota zadává. Uložení změn se provede tlačítkem *Save* a jejich zrušení tlačítkem *Cancel* ve spodní části tohoto okna.



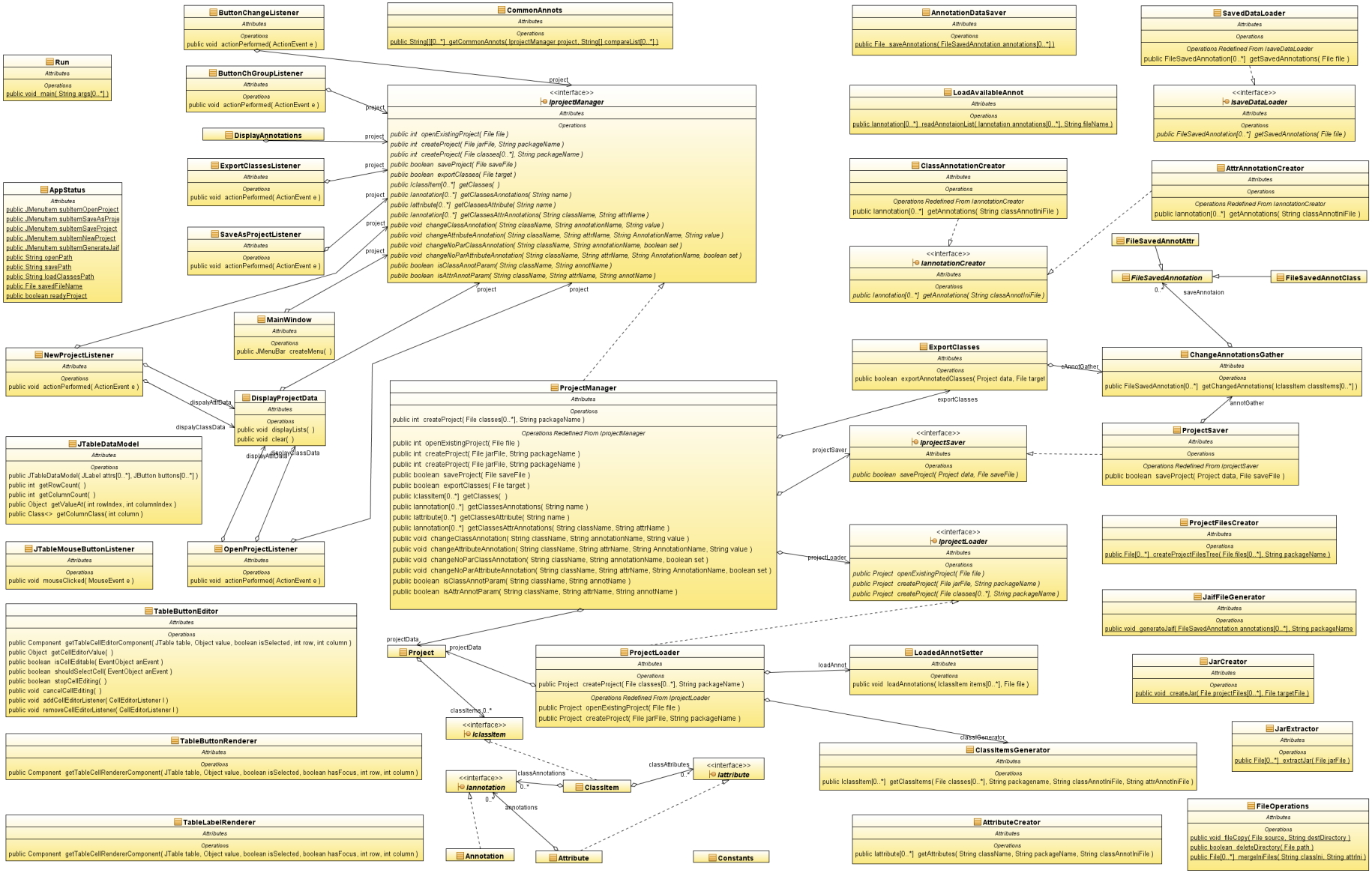
Obrázek A.2: Okno pro změnu anotací

V pravém horním rohu hlavního okna jsou umístěny ještě dvě záložky. První s názvem *Attributes* zobrazuje výše popsané listy, druhá záložka *Classes* přepíná na list, nesoucí seznam všech tříd, který slouží ke hromadné změně anotací tříd samotných. Uložení vytvořeného projektu na disk je možné skrze položku *Save as* nabíky *Project*, kdy je uživatel vyzván k výběru umístění a zadání názvu projektu. Položka *Save* slouží pouze k uložení provedených změn v již uloženém projektu.

K exportování anotovaných tříd slouží položka *Annotate Class files* nabídky *Annotation*. Při tomto požadavku je uživatel vyzván k výběru adresáře, do kterého budou soubory umístěny.

B UML diagram aplikace Annotation Tool

- viz následující list formátu A3


ButtonChangeListener
Attributes
Operations
public void actionPerformed(ActionEvent e)
CommonAnnots
Attributes
Operations
public String[][0..*] getCommonAnnots(IprojectManager project, String[] compareList[0..*])
AnnotationDataSaver
Attributes
Operations
public File saveAnnotations(FileSavedAnnotation annotations[0..*])
SavedDataLoader
Attributes
Operations
Operations Redefined From IsaveDataLoader
public FileSavedAnnotation[0..*] getSavedAnnotations(File file)
Run
Attributes
Operations
public void main(String args[0..*])
ButtonChGroupListener
Attributes
Operations
public void actionPerformed(ActionEvent e)
project
<<interface>>
IprojectManager
Attributes
Operations
public int openExistingProject(File file)
public int createProject(File jarFile, String packageName)
public int createProject(File classes[0..*], String packageName)
public boolean saveProject(File saveFile)
public boolean exportClasses(File target)
public IclassItem[0..*] getClasses( )
public Iannotation[0..*] getClassesAnnotations(String name)
public Iattribute[0..*] getClassesAttribute(String name)
public Iannotation[0..*] getClassesAttrAnnotations(String className, String attrName)
public void changeClassAnnotation(String className, String annotationName, String value)
public void changeAttributeAnnotation(String className, String attrName, String AnnotationName, String value)
public void changeNoParClassAnnotation(String className, String annotationName, boolean set)
public void changeNoParAttributeAnnotation(String className, String attrName, String AnnotationName, boolean set)
public boolean isClassAnnotParam(String className, String annotName)
public boolean isAttrAnnotParam(String className, String attrName, String annotName)
LoadAvailableAnnot
Attributes
Operations
public Iannotation[0..*] readAnnotaionList(Iannotation annotations[0..*], String fileName)
<<interface>>
IsaveDataLoader
Attributes
Operations
public FileSavedAnnotation[0..*] getSavedAnnotations(File file)
DisplayAnnotations
ExportClassesListener
Attributes
Operations
public void actionPerformed(ActionEvent e)
ClassAnnotationCreator
Attributes
Operations
Operations Redefined From IannotationCreator
public Iannotation[0..*] getAnnotations(String classAnnotIniFile)
AttrAnnotationCreator
Attributes
Operations
Operations Redefined From IannotationCreator
public Iannotation[0..*] getAnnotations(String classAnnotIniFile)
AppStatus
Attributes
public JMenuItem subItemOpenProject
public JMenuItem subItemSaveAsProje
public JMenuItem subItemSaveProject
public JMenuItem subItemNewProject
public JMenuItem subItemGenerateJaif
public String openPath
public String savePath
public String loadClassesPath
public File savedFileName
public boolean readyProject
SaveAsProjectListener
Attributes
Operations
public void actionPerformed(ActionEvent e)
<<interface>>
IannotationCreator
Attributes
Operations
public Iannotation[0..*] getAnnotations(String classAnnotIniFile)
FileSavedAnnotAttr
FileSavedAnnotation
FileSavedAnnotClass
0..*
saveAnnotaion
MainWindow
Attributes
Operations
public JMenuBar createMenu( )
NewProjectListener
Attributes
Operations
public void actionPerformed(ActionEvent e)
displayAttrData
displayClassData
DisplayProjectData
Attributes
Operations
public void displayLists( )
public void clear( )
ExportClasses
Attributes
Operations
public boolean exportAnnotatedClasses(Project data, File target
eAnnotGather
ChangeAnnotationsGather
Attributes
Operations
public FileSavedAnnotation[0..*] getChangedAnnotations(IclassItem classItems[0..*])
exportClasses
annotGather
ProjectManager
Attributes
Operations
public int createProject(File classes[0..*], String packageName)
Operations Redefined From IprojectManager
public int openExistingProject(File file)
public int createProject(File jarFile, String packageName)
public int createProject(File jarFile, String packageName)
public boolean saveProject(File saveFile)
public boolean exportClasses(File target)
public IclassItem[0..*] getClasses( )
public Iannotation[0..*] getClassesAnnotations(String name)
public Iattribute[0..*] getClassesAttribute(String name)
public Iannotation[0..*] getClassesAttrAnnotations(String className, String attrName)
public void changeClassAnnotation(String className, String annotationName, String value)
public void changeAttributeAnnotation(String className, String attrName, String AnnotationName, String value)
public void changeNoParClassAnnotation(String className, String annotationName, boolean set)
public void changeNoParAttributeAnnotation(String className, String attrName, String AnnotationName, boolean set)
public boolean isClassAnnotParam(String className, String annotName)
public boolean isAttrAnnotParam(String className, String attrName, String annotName)
projectSaver
<<interface>>
IprojectSaver
Attributes
Operations
public boolean saveProject(Project data, File saveFile)
ProjectSaver
Attributes
Operations
Operations Redefined From IprojectSaver
public boolean saveProject(Project data, File saveFile)
JTableDataModel
Attributes
Operations
public JTableDataModel(JLabel attrs[0..*], JButton buttons[0..*])
public int getRowCount( )
public int getColumnCount( )
public Object getValueAt(int rowIndex, int columnIndex)
public Class<> getColumnClass(int column)
projectLoader
<<interface>>
IprojectLoader
Attributes
Operations
public Project openExistingProject(File file)
public Project createProject(File jarFile, String packageName)
public Project createProject(File classes[0..*], String packageName)
ProjectFilesCreator
Attributes
Operations
public File[0..*] createProjectFilesTree(File files[0..*], String packageName)
JTableMouseButtonListener
Attributes
Operations
public void mouseClicked(MouseEvent e)
OpenProjectListener
Attributes
Operations
public void actionPerformed(ActionEvent e)
JaifFileGenerator
Attributes
Operations
public void generateJaif(FileSavedAnnotation annotations[0..*], String packageName
TableButtonEditor
Attributes
Operations
public Component getTableCellEditorComponent(JTable table, Object value, boolean isSelected, int row, int column)
public Object getCellEditorValue( )
public boolean isCellEditable(EventObject anEvent)
public boolean shouldSelectCell(EventObject anEvent)
public boolean stopCellEditing( )
public void cancelCellEditing( )
public void addCellEditorListener(CellEditorListener l)
public void removeCellEditorListener(CellEditorListener l)
projectData
Project
ProjectLoader
Attributes
Operations
public Project createProject(File classes[0..*], String packageName)
Operations Redefined From IprojectLoader
public Project openExistingProject(File file)
public Project createProject(File jarFile, String packageName)
loadAnnot
LoadedAnnotSetter
Attributes
Operations
public void loadAnnotations(IclassItem items[0..*], File file)
JarCreator
Attributes
Operations
public void createJar(File projectFiles[0..*], File targetFile)
classItems 0..*
<<interface>>
IclassItem
classGenerator
ClassItemsGenerator
Attributes
Operations
public IclassItem[0..*] getClassItems(File classes[0..*], String packagename, String classAnnotIniFile, String attrAnnotIniFile)
JarExtractor
Attributes
Operations
public File[0..*] extractJar(File jarFile)
TableButtonRenderer
Attributes
Operations
public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column)
<<interface>>
Iannotation
classAnnotations
0..*
ClassItem
classAttributes
0..*
<<interface>>
Iattribute
0..*
annotations
Annotation
Attribute
Constants
AttributeCreator
Attributes
Operations
public Iattribute[0..*] getAttributes(String className, String packageName, String classAnnotIniFile)
FileOperations
Attributes
Operations
public void fileCopy(File source, String destDirectory)
public boolean deleteDirectory(File path)
public File[0..*] mergeIniFiles(String classIni, String attrIni)
TableLabelRenderer
Attributes
Operations
public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column)